UM SISTEMA DE APOIO AO JOGADOR PARA JOGOS DE ESTRATÉGIA EM TEMPO REAL

RENATO LUIZ DE FREITAS CUNHA

UM SISTEMA DE APOIO AO JOGADOR PARA JOGOS DE ESTRATÉGIA EM TEMPO REAL

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientador: Luiz Chaimowicz

Belo Horizonte

Maio de 2010

Cunha, Renato Luiz de Freitas
C972s Um sistema de apoio ao jogador para jogos de estratégia em tempo real / Renato Luiz de Freitas Cunha. — Belo Horizonte, 2010
xxvii, 85 f. : il. ; 29cm

Dissertação (mestrado) — Universidade Federal de Minas Gerais
Orientador: Luiz Chaimowicz

1. Inteligência artificial - Teses. 2. Estratégia em tempo real - Teses. 3. Interface humano-computador - Teses. I. Título.

CDU 519.6*82(043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Um sistema de apoio ao jogador para jogos de estratégia em tempo real

RENATO LUIZ DE FREITAS CUNHA

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. LUIZ CHAIMOWICZ - Orientador
Departamento de Ciência da Computação - UFMG

PROF. GEBER LISBOA RAMALHO
Centro de Informática - UFPE

PROFA. RAQUEL OLIVEIRA PRATES
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 21 de maio de 2010.

Para minha família: mamãe, papai e lili.

Agradecimentos

É indescritível a felicidade que senti quando terminei este trabalho. Não que a experiência tenha sido ruim. Pelo contrário, ela foi agradável, ainda que permeada por caminhos tortuosos. Caminhos esses, aliás, que só fui capaz de trilhar graças ao apoio dos muitos amigos, colegas e profissionais que me auxiliaram durante o processo. Foi um período bom e divertido que, felizmente, chegou a um fim.

A minha família, Luiz Carlos, Maria Natividade e Lívia Cristina, muito obrigado pela paciência, compreensão da ausência, incentivo e torcida durante todo o processo. A minha cara Alice, meu muito obrigado pelo estímulo para ingressar no mestrado, por, também, compreender minha ausência e a importância de terminar com sucesso este projeto e, é claro, por todo o carinho durante esses anos.

Ao professor e orientador Luiz Chaimowicz, pela disposição em aprender sobre jogos de estratégia em tempo real durante a orientação e pela presença, suporte, sugestões constantes durante o desenvolvimento deste trabalho e por ser um profissional exemplar, obrigado.

Aos amigos que fiz em BH, pelos altos papos sobre o futuro, por sua preocupação em fazer um bom trabalho, pela motivação em momentos importantes, obrigado. Em particular, agradeço a Vilar Neto, Luiz Cantoni e Rafael Barra por todas as conversas, ajudas com dúvidas existenciais e, claro, pela amizade. A Mariella Berger e Pe. José Pedro Luchi pelo estímulo em ingressar no mestrado e aos demais amigos de Vitória, pela força à distância.

Aos amigos e colegas do VeRLab, Erickson Nascimento, Renato Garcia, Armando Neto, Daniel Balbino, Douglas Macharet, Marcelo Borghetti, Celso Brennand, Rafael Colares, Thiago Goulart, Wolmar Pimenta, Antônio Wilson, Sandro Jerônimo, Luiz Henrique Rios e Cláudio dos Santos pelo apoio e ajuda, obrigado.

Ao Programa de Pós-Graduação da Universidade Federal de Minas Gerais, pelas excelentes estrutura e organização e à CAPES pelo apoio financeiro nesses dois anos.

Sobretudo, agradeço ao bom Deus, que me permitiu nascer e viver em um tempo bom e trabalhar com o que gosto.

Por fim, agradeço a todos que contribuíram, de forma direta ou indireta, para o desenvolvimento deste trabalho.

"Skepticism is healthy only to a limited extent. Being skeptical about proofs and programs (particularly your own) will probably keep your grades healthy and your job fairly secure. But applying much skepticism will probably also keep you shut away working all the time, instead of letting you get out for exercise or relaxation. Too much skepticism is an open invitation to the state of rigor mortis*, where you become so worried about being correct and rigorous that you never get anything finished."*

(A skeptic)

Resumo

Apesar da crescente popularidade dos jogos de Estratégia em Tempo Real (RTS) como plataforma para pesquisa em Inteligência Artificial (IA), os trabalhos tendem a focar na criação de um agente inteligente (ou, neste caso, um *bot*) capaz de ganhar um jogo RTS ou a implementar alguma das competências de um jogador de RTS em algum *bot*. Outras abordagens usam aprendizado de máquina para aprender regras sobre o domínio do jogo e tentar prever as próximas ações do inimigo. Essas abordagens tendem, no entanto, a ignorar tópicos de pesquisa como o desenvolvimento de agentes de IA que auxiliem o usuário em seus jogos.

Este trabalho apresenta uma abordagem para implementação de um agente de IA capaz de auxiliar o jogador através de dicas táticas e estratégicas. Seu objetivo, portanto, é o de estabelecer métricas para avaliar o estado local de jogos RTS, elaborar hipóteses sobre qual é o desempenho atual do usuário, raciocinar sobre formas de melhorar o desempenho do jogador e apresentá-las sob a forma de dicas de estratégia para resolver o problema de melhorar o desempenho do jogador com a restrição de usar informação local, tornando o ambiente parcialmente observável.

Os resultados obtidos com testes com usuário e de desempenho sugerem que o arcabouço experimental desenvolvido neste trabalho cumpre seus objetivos sem degradar o desempenho de um jogo já existente.

Abstract

Even though Real Time Strategy (RTS) games are successfully being used as a platform for Artificial Intelligence (AI) research and experimentation, existing work tends to focus only on the development of intelligent agents (bots) for winning an RTS game. Other approaches use Machine Learning techniques to learn the rules of the game to try to predict the player's next move. These approaches tend to ignore interesting research topics like the development of AI agents that help the user in his games.

This work presents an approach to implement an AI agent capable of helping out the player by giving him some tactical and strategical tips. The problem we are trying to solve is to improve the performance of the human RTS player, and our main objective is to develop metrics to evaluate the current game state, elaborate hypothesis on how to improve the player's and communicate this information to him formatted as a set of strategy tips.

User and performance tests suggest that the experimental framework developed in this work satisfies its main objectives without degrading an existing game's computational performance.

Lista de figuras

Lista de tabelas

Lista de acrônimos

Acrônimo	*Descrição*
AIIDE	*Artificial Intelligence and Interactive Digital Entertainment Conference*
API	*Application Programming Interface* (Interface de Programação de Aplicativos)
CaT	*Case-based Tactician*
CBP	*Case-Based Planning* (Planejamento Baseado em Casos)
CBPR	*Case-Based Plan Recognition* (Reconhecimento de Planos Baseado em Casos)
CBR	*Case-Based Reasoning* (Raciocínio Baseado em Casos)
DARPA	*Defense Advanced Research Projects Agency* (Agência de Projetos de Pesquisa Avançada de Defesa)
EBN	*Extended Behavior Network* (Rede Comportamental Estendida)
FOW	*Fog of War* (Névoa de Guerra)
FPS	*Frames per Second* (Quadros por Segundo)
FSM	*Finite State Machine* (Máquina de Estados Finitos)
GRL	*Generic Robot Language* (Linguagem Robótica Genérica)
HTN	*Hierarchical Task Network* (Rede de Tarefas Hierárquica)
IA	Inteligência Artificial
LAN	*Local Area Network* (Redes Locais)
MAPF	*Multiagent Potential Fields* (Campos Potenciais Multi-Agentes)
NPC	*Non Player Character* (Personagem Não Jogável)
ORTS	*Open Real Time Strategy*
PDDL	*Planning Domain Definition Language* (Linguagem de Definição do Domínio de Planejamento)
PF	*Potential Fields* (Campos Potenciais)
PR	*Plan Recognition* (Reconhecimento de Planos)

(Continua na próxima página)

Acrônimo	*Descrição*
RL	*Reinforcement Learning* (Aprendizado por Reforço)
RTS	*Real Time Strategy* (Estratégia em Tempo Real)
SHOP	*Simple Hierarchical Ordered Planner* (Planejador Simples Hierárquico Ordenado)
TBS	*Turn-Based Strategy* (Estratégia Baseada em Turnos)
TIELT	*Testbed for Integrating and Evaluating Learning Techniques*
TMK	*Task-Method-Knowledge*
TTS	*Text to Speech* (texto para fala)

Lista de algoritmos

Sumário

Capítulo 1

Introdução

Neste capítulo serão apresentados o domínio de jogos de Estratégia em Tempo Real (*Real Time Strategy*, ou RTS), a motivação deste trabalho, a definição do problema a ser tratado e as contribuições deste trabalho.

1.1 Motivação

A indústria de jogos é parte crescente da indústria do entretenimento. Esse crescimento provocou um aumento na necessidade de profissionais especializados no desenvolvimento de jogos e despertou o interesse acadêmico nesta área. Com o advento de computadores pessoais mais rápidos, jogos que demandavam mais recursos dos computadores se tornaram mais populares [Buro & Furtak, 2004]. Exemplos incluem jogos RTS, jogos de esportes e jogos de simulação física.

Os jogos RTS, apresentados na seção 1.2 e descritos na seção 2.1, são jogos em que os participantes comandam unidades e estruturas sob seu controle para, geralmente, garantir a segurança de uma área ou destruir uma região controlada pelo inimigo, ou o próprio inimigo.

A pesquisa em jogos RTS vem aumentando ao longo do tempo, tornando esses jogos um tópico sério de pesquisa, de forma similar aos jogos clássicos, como damas, xadrez e go. A razão para isso é que a pesquisa em jogos RTS abrange diversos aspectos fundamentais da pesquisa em Inteligência Artificial (IA), como planejamento, tomada de decisões em tempo real, aprendizado e modelagem de oponentes, raciocínio espacial e temporal, gerência de recursos, colaboração entre diversos agentes e planejamento de caminhos [Buro & Furtak, 2004].

Além dos motivos supracitados, diferentemente da pesquisa em jogos de estratégia clássicos, em que grandes avanços foram obtidos nos últimos anos (como a prova do jogo de damas, descrita em Schaeffer et al. [2007], por exemplo), há ainda grandes desafios na pesquisa

Figura 1.1. Captura de tela de Wargus, um jogo RTS.

em jogos RTS, devido não só ao tamanho dos mundos virtuais mas também ao número de estados possíveis presentes nesses jogos com o agravante de que o conhecimento que se tem sobre o mundo é imperfeito, o que faz dos jogos RTS um tópico interessante para pesquisa.

Apesar disso, a maioria dos trabalhos da comunidade de IA em jogos RTS tem se concentrado na codificação de domínios complexos de aprendizado de máquina e planejamento, sem estudar interações entre pessoas e máquinas. Como o desenvolvimento de uma linguagem para pessoas demonstrarem comportamento desejável para uma máquina, ou para máquinas auxiliarem pessoas em jogos[Metoyer et al., 2010].

1.2 Jogos RTS

Jogos RTS são um gênero de jogos de computador que normalmente são ambientados em uma guerra, onde as ações dos jogadores não são ordenadas por turnos, mas contínuas.

A figura 1.1 exibe uma captura de tela do jogo RTS Wargus. Nela encontram-se todos os elementos comuns à interface de um jogo RTS, como um mini-mapa do cenário no canto superior esquerdo da tela, uma unidade selecionada logo abaixo do mini-mapa, uma lista de comandos que podem ser enviados à unidade, dados sobre recursos na parte superior da tela e, no centro, uma visão do ambiente do jogo.

Do ponto de vista da IA, os elementos comuns a jogos RTS são limites de tempo severos

sobre as ações do jogador e uma forte demanda de IA de tempo real, que deve ser capaz de resolver tarefas de decisão rápida e satisfatoriamente. Os ambientes de jogos RTS possuem, também, ambientes parcialmente observáveis[1], com adversários que modificam o estado do jogo de maneira assíncrona e com modelos de tomada de decisão desconhecidos, tornando inviável a obtenção de informação completa sobre o estado do jogo. Sobre essa última característica, as implementações da indústria consistem em acessar diretamente as estruturas de dados do motor do jogo, tornando a implementação de oponentes de IA viável. Jogos de estratégia em tempo real são, portanto, aplicações de teste ideais para a pesquisa em IA de tempo real [Buro & Furtak, 2004; Buro, 2004; Ponsen et al., 2005].

1.3 Definição do problema e objetivos

Este trabalho trata do problema de melhorar o desempenho do usuário de jogos RTS. Seu objetivo é o de estabelecer métricas para avaliar o estado local de jogos RTS, elaborar hipóteses sobre qual é o desempenho atual do usuário e sobre formas de melhorá-lo e apresentá-las ao usuário no formato de dicas táticas e estratégicas. O processo de observação do estado do mundo é restrito ao conhecimento disponibilizado ao jogador. Em outras palavras, a abordagem usada neste trabalho é restrita a métodos que não utilizem informação global, mas apenas informação disponível ao usuário. Portanto, mais especificamente, o neste trabalho será desenvolvido e avaliado um sistema de dicas táticas e estratégicas que observará o usuário e sugerirá formas de melhorar seu jogo.

Um benefício secundário do desenvolvimento de tal sistema é a geração de conhecimento envolvendo não só o interfaceamento de módulos de IA a motores de jogos RTS como a geração de uma interface específica com um motor de jogos.

Como citado, o domínio dos jogos RTS, possui um espaço de estados grande, estimado como pertencente à ordem de $1{,}5 \cdot 10^3$ jogadas possíveis para cada estado [Aha et al., 2005]. Esse valor é substancialmente maior que o de jogos de tabuleiro clássicos, como o xadrez, que possui uma média de 30 jogadas possíveis. Com tantas possibilidades a serem exploradas, soluções baseadas puramente em busca exaustiva tornam esse problema intratável. Boas abstrações do espaço de estados são cruciais para resolver esse problema em tempo real e sem recorrer a subterfúgios, como o uso de conhecimento global para tomada de decisão.

Aplicações interessantes deste trabalho incluem trabalhos gerados pela própria indústria de jogos, posto que, ao possibilitar aos jogadores a melhoria de suas habilidades nas partidas, estima-se que as partidas se tornem mais competitivas. Essa aplicação poderia se dar através da criação de sistemas de tutoreamento implementados em jogos. Ao permitir aos usuários

[1] Diferente de jogos clássicos, em que todas as ações de todos os oponentes são conhecidas, em jogos RTS apenas uma parte deles está disponível aos jogadores.

aprender durante o jogo, é razoável esperar que jogadores casuais se interessem por jogos com esse tipo de sistema, corroborando a hipótese de que uma abordagem como essa é interessante.

Como será descrito na seção 2.3, há poucos trabalhos na literatura a abordar esse problema. Sua investigação, no entanto, parece promissora.

1.4 Organização deste documento

Este documento está organizado da seguinte forma: neste capítulo foi apresentado o tema a ser tratado na dissertação, com seus objetivos e motivações. O capítulo 2 descreve os desafios de jogos RTS detalhando suas características, apresentando as técnicas de IA comumente encontradas nesses jogos e apresentando e discutindo os trabalhos relacionados ao tema. No capítulo 3 as competências necessárias para que um agente seja um bom jogador de RTS são apresentadas em conjunto com as características avaliadas para o sistema de dicas descrito neste trabalho. O capítulo 4 apresenta o arcabouço experimental desenvolvido para este trabalho. No capítulo 5 são apresentados os experimentos e seus resultados e, no capítulo 6 são apresentadas as conclusões e trabalhos futuros.

Capítulo 2

Desafios de jogos RTS

Neste capítulo são apresentados alguns desafios de jogos RTS, iniciando com um detalhamento das características desses jogos, depois com uma breve discussão sobre a respeito dos motores de jogos de código aberto para desenvolvimento de jogos RTS e, então, com as técnicas de IA usadas nesses jogos. Por fim, com os conceitos bem definidos, serão apresentados os trabalhos relacionados a este.

2.1 Detalhamento das características dos jogos RTS

Jogos RTS são jogos de estratégia militar em tempo real[1] com elementos de construção de bases e gerência de recursos. O objetivo de uma partida de um jogo desse gênero é, normalmente, o de destruir os exércitos inimigos, situados num mapa (um campo de batalha virtual). Outros objetivos comuns envolvem defender a segurança de uma área ou de um conjunto de construções ou desenvolver um exército com características definidas no começo de uma missão. Há, ainda, partidas com tempo pré-determinado. Nessa última modalidade, o vencedor costuma ser aquele que destruir mais unidades inimigas, sem a necessidade de destruir totalmente o inimigo.

É importante notar, entretanto, que entre os desenvolvedores de jogos desse gênero não há um consenso sobre que características definem um jogo RTS. Sendo até possível que um jogo seja anunciado como RTS sem, de fato, possuir todas as características mais comuns para que seja definido como tal.

[1] Note que a definição de "tempo real" neste contexto não é algo tão forte quanto nos sistemas de tempo real, onde existem limites severos de tempo que, se descumpridos, podem resultar na falha de todo um sistema. No contexto deste documento (e dos jogos RTS), "tempo real" quer dizer "tempo contínuo", ou que não há divisão de turnos entre as jogadas. Ou seja, os diversos eventos do jogo podem ocorrer simultaneamente.

2.1.1 Componentes comumente encontrados em um jogo RTS

No início de um jogo RTS cada um dos jogadores é posicionado em algum lugar de um mapa bidimensional. Esse mapa representa o ambiente do jogo e contém, ou conterá, unidades, construções e recursos. Cada jogador costuma iniciar com algumas unidades básicas, ou com uma construção capaz de treiná-las. Comumente, a partir desse ponto, o jogador deve coletar recursos para poder edificar novas construções, treinar novas unidades, aumentar e manter sua base e desenvolver sua tecnologia. Em diversos jogos também é necessário edificar construções usadas como fonte de alimento para o exército.

Em todos os jogos RTS há a necessidade de construção de um exército, cujo tamanho pode variar de pequenos esquadrões formados por apenas duas unidades até grandes, com centenas delas. O objetivo final desse exército é o de combater os exércitos inimigos, em tempo real. Exceto no início de uma partida, quando não há muitas opções para explorar, todas essas ações costumam ocorrer em paralelo e, portanto, um bom jogador deve ser capaz de gerenciar, simultaneamente, as atividades de extração, produção e combate para obter sucesso no jogo.

A extração de recursos costuma ser parte fundamental desses jogos, pois eles tendem a ser a base de toda a cadeia produtiva dos exércitos. Os recursos do mapa costumam estar disponíveis a todos os oponentes, desde que eles sejam capazes de extraí-los. Em alguns casos, para ter acesso aos recursos também é necessário combater para conquistar o direito de exploração do mesmo. Geralmente, os recursos não são renováveis, o que contribui para a necessidade de planejamento estratégico em um jogo.

Apesar de não haver consenso sobre os elementos de um jogo RTS, há alguns elementos e conceitos comuns, que são descritos abaixo.

Unidades Uma unidade é qualquer objeto presente no jogo com o qual jogadores podem interagir. Neste grupo se destacam as unidades possuídas pelo jogador e as possuídas por seus inimigos. Ela pode ser, por exemplo, um soldado, uma aeronave, um coletor de recursos ou uma unidade neutra, como uma mina de ouro. Unidades normalmente são combatentes, capazes de atacar e destruir outras unidades ou construções. Elas podem se especializar em ataque terrestre, de curto ou longo alcance ou ataque aéreo. Outras unidades podem ser capazes de extrair recursos e, comumente, incapazes de atacar de maneira efetiva. A figura 2.1c exibe detalhes de uma unidade e a figura 2.1f exibe unidades trabalhadoras.

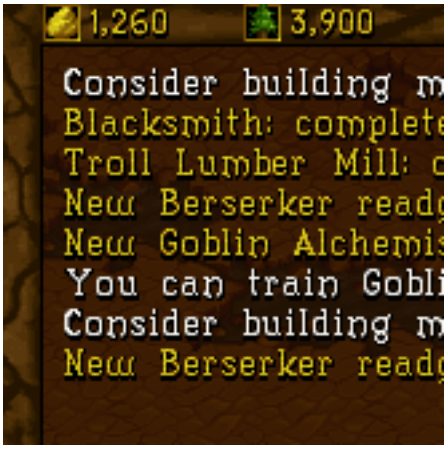
Construções Construções são normalmente usadas para produção e atualização de unidades, armazenamento de recursos, desenvolvimento de novas habilidades, ou defesa. A figura 2.1d exibe uma construção no ambiente do jogo.

(a) Mini-mapa.

(b) Mina de ouro e névoa de guerra.

(c) Dados da unidade "Berserker".

(d) Construção.



(e) Listas de recursos e mensagens.

(f) Unidades trabalhadoras em extração de madeira.

Figura 2.1. Alguns dos elementos comumente encontrados em um jogo RTS. Os exemplos foram retirados do jogo Wargus. Na figura são exibidos elementos típicos da interface na coluna esquerda e elementos do jogo na coluna direita.

Recursos Recursos são necessários para produzir unidades, construções ou atualizações e precisam estar armazenados antes de serem usados. Logo, recursos precisam ser extraídos constantemente pelos jogadores para que haja sucesso no jogo. A figura 2.1e exibe, em sua parte superior, os recursos acumulados por um jogador de Wargus. Na figura 2.1f são exibidas unidades trabalhadoras extraindo madeira e na figura 2.1b é exibida uma mina de ouro, fonte do recurso "ouro".

Névoa de Guerra A Névoa de Guerra (*Fog of War*, ou FOW) representa a falta de conhecimento que ocorre durante uma guerra. Em jogos RTS, a névoa de guerra é implementada como uma região do mapa em que, apesar do terreno ser conhecido, não é possível saber o que ocorre naquela região. Na figura 2.1b há um exemplo da névoa de guerra: A região mais clara é totalmente conhecida pelo jogador. Na região à esquerda e abaixo da mina de ouro, onde o terreno está sombreado, há névoa de guerra, pois o jogador conhece aquela região, mas é incapaz de saber o que acontece por lá. Já a região totalmente preta é desconhecida pelo jogador.

Microgerência Microgerência é um termo usado para designar a atividade exercida pelo jogador quando sua atenção é direcionada à gerência e manutenção de suas próprias unidades e recursos. Um exemplo de atividade de microgerência é a necessidade de um jogador garantir que nenhum de seus trabalhadores está ocioso, para garantir entrada de recursos constante e manutenção da base.

2.1.2 Motores de jogos RTS

Há vários motores RTS publicamente disponíveis, como Stratagus, Boswars, Glest, Spring Engine, RTSCup e *Open Real Time Strategy* (ORTS). Os motores estudados neste trabalho foram o ORTS e o Stratagus e ambos serão apresentados nas próximas seções. O principal motivo para escolha desses dois motores é o fato de já terem sido usados com sucesso por outros pesquisadores, conforme discutido na seção 2.3.

2.1.2.1 ORTS

Open Real Time Strategy (ORTS) é um motor multiplataforma e de código aberto criado com o intuito de prover um ambiente de jogos RTS invulnerável a ataques de revelação de mapas [Buro, 2002]. Por ter sido desenvolvido em um ambiente acadêmico, vários trabalhos já o usaram como ferramenta de pesquisa. Algumas características do ORTS são:

Arquitetura cliente-servidor Na arquitetura do ORTS, a simulação do estado do jogo ocorre somente no servidor. Os clientes são responsáveis, apenas, por enviar comandos com

as ações dos jogadores e de visualizar o estado do jogo enviado pelo servidor. Essa arquitetura permite que o ORTS seja invulnerável a ataques de revelação de mapas, pois tentativas de acesso indevidas podem ser detectadas pelo servidor e devidamente tratadas.

Possibilidade de implementação remota de IA Por possuir um protocolo aberto, módulos de IA podem ser executados em computadores diferentes do responsável por executar a simulação. Além disso, os clientes podem ser desenvolvidos em diferentes linguagens de programação, desde que implementem a Interface de Programação de Aplicativos (*Application Programming Interface*, ou API) necessária.

Facilidade de criação de *bots* Os desenvolvedores do ORTS encorajam o desenvolvimento de diferentes *bots* para o motor. Por isso, o repositório de código já possui um módulo, *sampleai*, com código de exemplo que pode ser usado como base para criação de um cliente para o jogo.

Controle absoluto e em paralelo de unidades Jogos RTS tradicionais permitem uma baixa taxa de envio de comandos para execução pelo motor, tornando a execução dos comandos essencialmente sequencial. Essa restrição é imposta pela interface com o motor, comumente acessível apenas por comandos de teclado e de mouse. No ORTS, é possível executar um comando por unidade por quadro de simulação, permitindo envio paralelo de comandos. Outra característica desse motor é que as unidades não possuem comportamento pré-definido, o que dá liberdade para experimentação.

Gerente de torneios Para comparação de desempenho de módulos de IA, o ORTS possui um gerente de torneios. Esse gerente é capaz de executar diversos combates entre os módulos em comparação e provê estatísticas do combate, como taxa de vitórias e derrotas e tempo de duração de disputas.

Uma descrição mais aprofundada do motor e de seu desenvolvimento pode ser encontrada em Buro [2002].

2.1.2.2 Stratagus

Stratagus é um motor multiplataforma de código aberto usado para o desenvolvimento de jogos RTS. Apesar de não ter sido projetado para ser usado em pesquisa, esse motor foi usado com sucesso por diversos pesquisadores e estudantes (em projetos de disciplinas). A introdução ao motor Stratagus apresentada nesta subseção é fortemente baseada na encontrada em [Ponsen et al., 2005].

O Stratagus é capaz de executar tanto jogos de um único jogador (contra um oponente controlado por computador) quanto jogos de múltiplos jogadores, através da Internet ou das Redes Locais (*Local Area Network*, ou LAN). O Stratagus foi desenvolvido oficialmente de 1998 a 2007, inicialmente na linguagem C e, posteriormente, em C++.

Jogos que usam o Stratagus são implementados na linguagem Lua [Ierusalimschy, 2006]. Alguns jogos desenvolvidos para o Stratagus estão disponíveis em sua página web [Stratagus, 2010]. Dentre eles, se destacam o Wargus, um clone do jogo Warcraft II e Battle of Survival, jogo onde os esforços da antiga equipe de desenvolvimento do Stratagus estão concentrados. Esses jogos compartilham a mesma API fornecida pelo motor. Algumas características que tornam o Stratagus atrativo para a pesquisa em IA incluem:

Configurável Esse motor permite a personalização de diversos parâmetros, habilitando usuários a criar jogos com características variadas. Assim, jogos podem ser adaptados para tarefas específicas, como ganhar jogos completos, vencer batalhas locais, gerenciar recursos, etc. Por exemplo, ao mudar arquivos de texto puro, novas unidades podem ser adicionadas, características de unidades existentes podem ser reguladas, novas construções e mapas podem ser adicionados e as condições de vitória, redefinidas.

Jogos implementados O Stratagus já possui jogos operacionais. Como todos os jogos compartilham a mesma API, um sistema de decisão pode ser avaliado em vários domínios.

Modificável Como supracitado, o Stratagus possui um interpretador de Lua integrado a seu código. Com esse interpretador, várias das estruturas de dados e funções internas do motor são acessíveis através de Lua, o que permite a mudança e adaptação do código em uma linguagem mais simples que C++.

Modo rápido O motor inclui um modo rápido, em que a taxa de atualização de vídeo é diminuída, aumentando a velocidade de execução do jogo. Esse modo é útil, por exemplo, para a condução de experimentos

Estatísticas Durante e após os jogos, dados relacionados ao jogo (como tempo de duração da partida, número de unidades perdidas, etc) estão disponíveis. Essas estatísticas são úteis para medição de desempenho de algoritmos.

Gravação O motor permite a gravação e reprodução de jogos gravados. Os jogos gravados podem ser usados para avaliação posterior e como fonte de treinamento para sistemas de IA que desempenhem tarefas como reconhecimento de planos ou planejamento estratégico.

Editor de mapas O Stratagus possui um editor de mapas, que pode ser usado para variar o estado inicial de partidas[2].

Determinismo Os cálculos executados pelo motor são determinísticos e, portanto, no caso da necessidade de sintonia de parâmetros de algoritmos sob experimentação, pesquisadores teriam a certeza de que as mudanças nos eventos do jogo seriam causadas pelos novos parâmetros, não por um possível não-determinismo do motor.

A combinação de fatores positivos aliada ao código aberto do motor e à existência de jogos completos permitiu que o Stratagus fosse usado como plataforma de experimentação de diversos pesquisadores.

2.2 Técnicas de IA em jogos RTS

Esta seção apresenta alguns conceitos, estruturas de dados e algoritmos comumente usados em jogos RTS para implementação de seus módulos de IA.

2.2.1 Componentes comuns em IA de jogos RTS

A IA de jogos RTS gerencia o processo de tomada de decisão de oponentes controlados pelo computador. Alguns problemas são comumente encontrados em jogos RTS, como combate, gerência de recursos, construção, comércio, planejamento de caminhos para múltiplos agentes, colaboração, planejamento de estratégias e tomada de decisões em tempo real. Apesar da organização da IA de jogos RTS específicos variar de jogo para jogo, a figura 2.2 exibe uma organização básica, comumente presente em jogos deste gênero baseada na apresentada em Millington [2006]. Na figura, as camadas de nível mais baixo fornecem abstrações a serem usadas por camadas de níveis superiores. As próximas subseções apresentarão cada um desses componentes, da camada mais inferior para a camada mais superior.

2.2.1.1 Movimento

A capacidade de movimentação é um dos requisitos mais fundamentais da IA de jogos e mesmo os primeiros jogos já possuíam alguma capacidade de mover seus personagens na tela.

Para movimentação de um único personagem, algoritmos que derivam das equações da cinemática são suficientes em casos mais simples e podem usar *waypoints*[3] fornecidos pelo planejador de caminhos em casos mais complexos.

[2] Na versão usada neste trabalho, no entanto, o editor de mapas estava indisponível, por incompatibilidades introduzidas por uma mudança de API.

[3] Um *waypoint* é um ponto que marca um lugar em um mapa ou no mundo real. Eles possuem coordenadas e, opcionalmente, descrições associadas a eles.

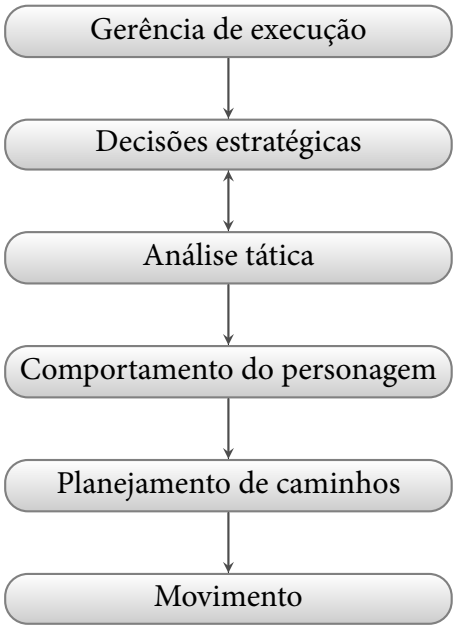


Figura 2.2. Uma forma de organização da Inteligência Artificial de jogos RTS. Nela, os mais baixos fornecem abstrações para serem usadas pelos níveis superiores.

Devido a possibilidade de uso de formações em jogos RTS, algoritmos de movimento de grupo também são necessários. A maioria deles deriva das técnicas introduzidas em Reynolds [1987] e aperfeiçoadas em Reynolds [1999]. Nessas técnicas, três comportamentos básicos influenciam os personagens, descritos abaixo.

Separação em que cada personagem mantém uma distância mínima de separação entre ele mesmo e seus vizinhos, exibido na figura 2.3a;

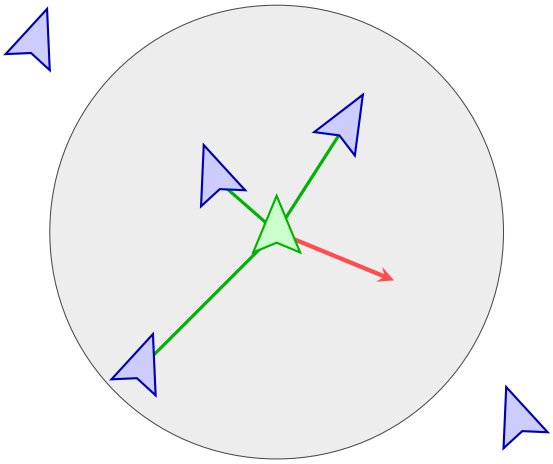
Alinhamento em que o grupo tende a se mover numa mesma direção, exibido na figura 2.3b.

Coesão em que o grupo deve permanecer unido, exibido na figura 2.3c;

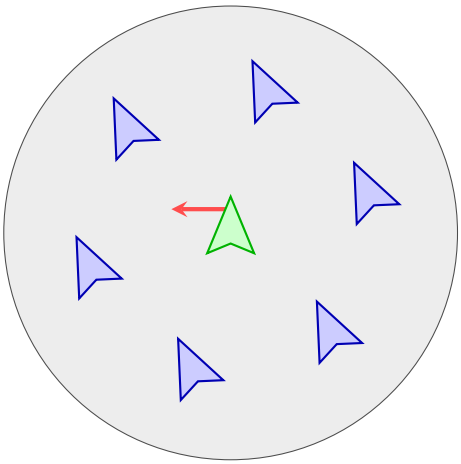
A figura 2.3 exibe exemplos desses três comportamentos. Com eles, o movimento local dos componentes do grupo é definido. O movimento global, por sua vez, é ditado pela mesma técnica usada no movimento individual através da criação, por exemplo, de um membro virtual do grupo responsável por obedecer aos algoritmos cinemáticos. Um sumário das técnicas utilizadas atualmente é apresentado em Tomlinson [2004].

2.2.1.2 Planejamento de caminhos

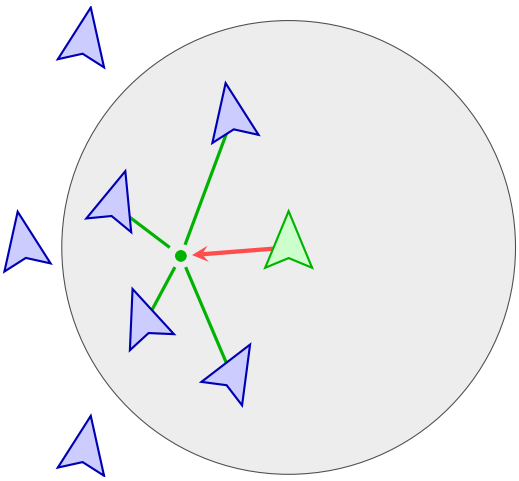
Em um jogo RTS, uma unidade pode ser ordenada a se mover para qualquer ponto no ambiente. Supondo ser possível encontrar uma rota para o ponto de destino, a unidade deve ser capaz de



(a) Separação, em que os agentes evitam se amontoar.



(b) Alinhamento, em que os agentes tendem a se mover numa mesma direção.



(c) Coesão, em que os agentes tendem a se mover para o ponto médio entre seus companheiros para manter o grupo unido.

Figura 2.3. Comportamentos de grupo básicos propostos por Reynolds [1987]. Nas figuras, o elemento central representa o agente para o qual as forças de interação serão computadas. O círculo, por sua vez, representa a vizinhança de cálculo das forças para o elemento central.

encontrar essa rota. Normalmente, espera-se que a rota encontrada seja a mais curta ou rápida possível. Ao problema de encontrar essa rota é dado o nome de planejamento de caminhos.

O planejamento de caminhos é um tópico ativo de pesquisa e diversas melhorias foram propostas nos últimos anos tanto em algoritmos de busca, como o A* [Sturtevant & Buro, 2005][Sturtevant & Buro, 2006], quanto em formas de representação do mundo[Forbus et al., 2001]. Essa pesquisa é motivada pelo fato de diversas buscas com o A* tradicional terem alto custo computacional. Uma melhoria relativamente recente, por exemplo, foi a introdução do HPA*[Botea et al., 2004], que permite buscas em grafos com diversos níveis de abstração, o que, além de aumentar o desempenho da busca, se assemelha mais a como pessoas planejam caminhos: primeiro planejando caminhos em alto nível e depois resolvendo detalhes à medida que o caminho é percorrido.

2.2.1.3 Comportamento do personagem

Excluído o sistema de animação, em jogos RTS o comportamento de um personagem está associado a seu processo de tomada de decisão.

O funcionamento das técnicas para tomada de decisão ocorre da seguinte forma: sua entrada é o conhecimento que um personagem possui e sua saída é uma requisição de ação Millington [2006]. Esse conhecimento pode ser dividido em conhecimento externo e conhecimento interno. O conhecimento interno é a informação que o personagem possui sobre seu próprio estado, como energia, objetivos e atividade que está realizando. Já o externo é a informação que o personagem possui sobre o ambiente à sua volta, como a posição de outros personagens, a disposição do terreno e assim por diante.

Existem diversas técnicas que podem ser usadas para tomada de decisão, como, mas não limitadas a, árvores de decisão, Máquinas de Estados Finitos (*Finite State Machines*, ou FSMs), lógica nebulosa, sistemas baseados em regras, ou arquiteturas comportamentais derivadas da robótica, como a arquitetura *subsumption*.

2.2.1.4 Decisões estratégicas

As decisões estratégicas tratam de elaborar um plano de ação com o intuito de alcançar um objetivo em particular, conforme descrito na seção 3.1.1.

Tradicionalmente, em jogos RTS, a estratégia costuma ser pré-definida e implementada por meio de *scripts*. Na figura 2.4 é exibido o laço final da estratégia *soldiers rush*, normalmente usada como *benchmark* para novas estratégias de Wargus. O *script* nela exibido é executado logo após a infra-estrutura da base, também implementada como um *script*, ter sido criada. Essa infra-estrutura é necessária para criação dos soldados que serão usados para atacar o oponente. O trecho exibido na figura é responsável por treinar cinco soldados, exibido na

```
1 local end_loop_funcs = {
2   function() return AiForce(1, {AiSoldier(), 5}) end,
3   function() return AiWaitForce(1) end,
4   function() return AiAttackWithForce(1) end,
5   function() return AiSleep(500) end,
6 }
```

Figura 2.4. Laço final da estratégia *soldiers rush* de Wargus. As estratégias de Wargus são implementadas como *scripts* e nesta figura é exibida a parte final do *script*, em que a IA treina cinco soldados continuamente para atacar seus inimigos.

linha 2, aguardar pelo treinamento exibido na linha 3, e, então, atacar o oponente, exibido na linha 3. Após o ataque ser iniciado, o *bot* aguarda por quinhentos quadros antes de reiniciar a sequência de ações do laço final.

A tendência, como apresentado na seção 2.3, é que as IAs de jogos RTS se tornem mais sofisticadas e com capacidade de adaptação às estratégias inimigas.

2.2.1.5 Análise tática

A análise tática do ambiente e do terreno provê informações para que o módulo estratégico possa planejar suas próximas ações. Técnicas como o uso de *waypoints* táticos, que envolvem a marcação do terreno com informações táticas, ou mapas de influência, que guardam informações sobre a influência militar de cada lado em cada ponto do terreno, podem ser usadas.

Existem diversos fatores que podem afetar a influência militar: a proximidade de uma unidade militar, a proximidade de uma base bem defendida, o tempo desde que uma localização foi ocupada por uma unidade, a geografia do terreno, o estado financeiro corrente de cada potência militar, o tempo atual e assim por diante. É possível usar todos esses fatores para criar uma IA tática, mas a maioria deles possui um efeito pequeno. A maioria dos jogos torna o mapeamento de influência mais simples usando uma hipótese simplificadora: a influência militar é um fator da proximidade de unidades e bases inimigas e seu poder militar.

2.2.1.6 Gerência de execução

Alguns componentes da IA de um jogo podem consumir muito tempo de processador, como o planejamento de caminhos, o processo de tomada de decisão ou a análise tática. Além disso, o perfil de execução da IA pode ser inconsistente, pois às vezes muito tempo é necessário ao planejar um caminho, mas pouco tempo é necessário para seguir uma rota calculada. Devido a esses problemas de perfil de execução do sistema de IA, normalmente implementa-se algum

sistema de gerência de execução. Esse tipo de tarefa não é, exatamente, função da IA, mas é necessária para que o jogo execute com suavidade para o jogador. Para reduzir a carga da IA sobre o sistema, suas tarefas costumam ser divididas através de diversos quadros do jogo, consumindo um tempo máximo pré-definido durante cada quadro e adiando para o próximo as tarefas ainda não concluídas.

2.2.2 Mapas de Influência

Esta seção tem por objetivo apresentar os mapas de influência, estrutura de dados usada na implementação inicial deste trabalho. O conteúdo desta seção é baseado em Tozour [2001a] e em Millington [2006].

2.2.2.1 Definição

Um mapa de influência é uma representação espacial do conhecimento de um agente inteligente a respeito do mundo e permite que um jogador controlado pelo computador seja capaz de desenvolver uma perspectiva tática do estado atual do mundo. Com um mapa de influência é possível conhecer onde as forças de um inimigo se localizam, onde há mais chances de encontrar um inimigo, onde se situa a "fronteira" entre exércitos, onde batalhas importantes aconteceram e onde se encontram as áreas mais vulneráveis a ataques inimigos.

Apesar de ser uma técnica bastante usada em jogos RTS, não existe um conjunto de algoritmos e estruturas de dados padrão que definam um mapa de influência, ou uma aplicação única desta técnica Tozour [2001a], pois a forma de implementação depende das necessidades táticas e estratégicas implementadas.

2.2.2.2 Estrutura

Os mapas de influência dividem o ambiente do mundo em diferentes áreas e podem ser aplicados sobre qualquer tipo de terreno, pois são implementados como uma estrutura superposta ao formato básico do terreno. O critério de divisão das áreas do terreno é que elas devem possuir características semelhantes para que possam ser agrupadas e analisadas. Para divisão do terreno, alguns métodos comuns são usados, como diagramas de Voronoi e divisão do mapa em malha. Uma suposição comumente feita é a de que a influência militar é, principalmente, uma função do poder relativo das unidades de combate e da distância. A figura 2.5 exibe um mapa de influência representado como uma malha gerado por uma construção e um trabalhador no jogo Wargus. Nela, é possível observar que há maior influência no ponto onde o trabalhador e o ponto superior esquerdo da construção se encontram, com a influência rapidamente diminuindo com a distância.

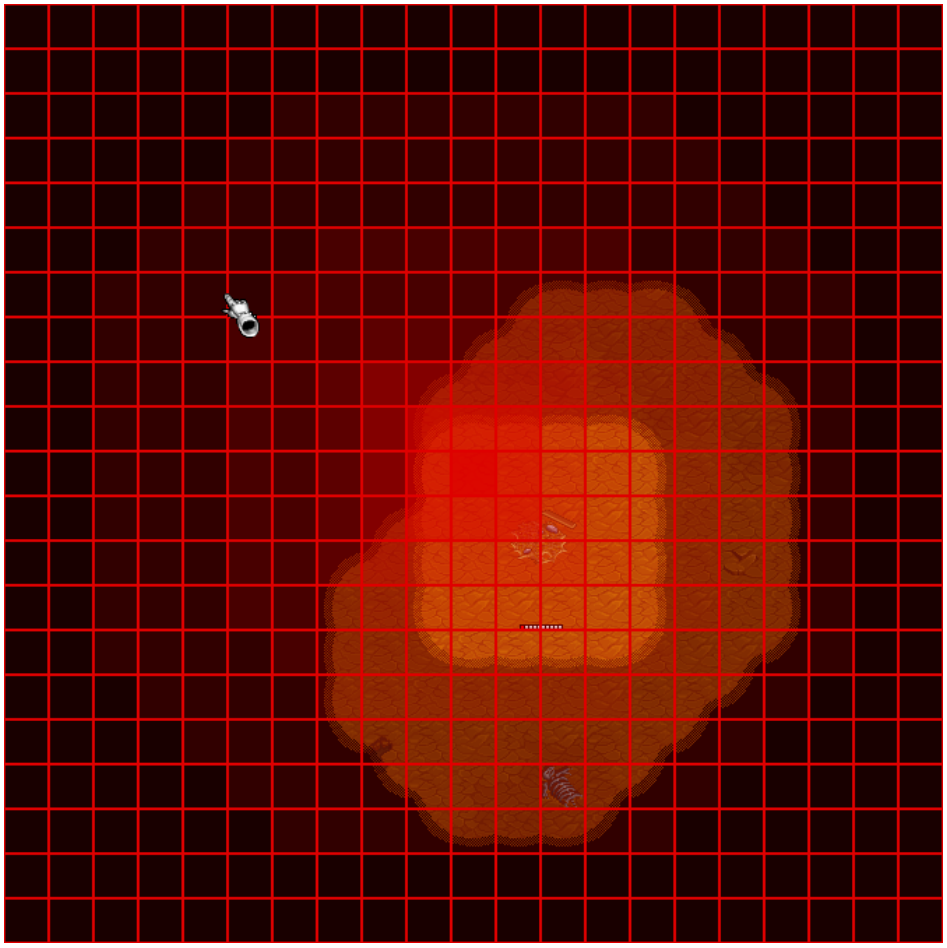


Figura 2.5. Renderização de depuração do jogo Wargus com um mapa de influência. Quanto maior a opacidade da célula, maior a influência. Nessa implementação, o mapa é representado como uma malha superposta ao ambiente e as linhas vermelhas exibem os limites das células. Esse mapa foi computado a partir da influência de um "Great Hall" em construção. O ponto de maior influência e, portanto o ponto com maior "poder" militar, é o do canto superior esquerdo da construção devido à representação interna da posição de uma unidade no motor Stratagus.

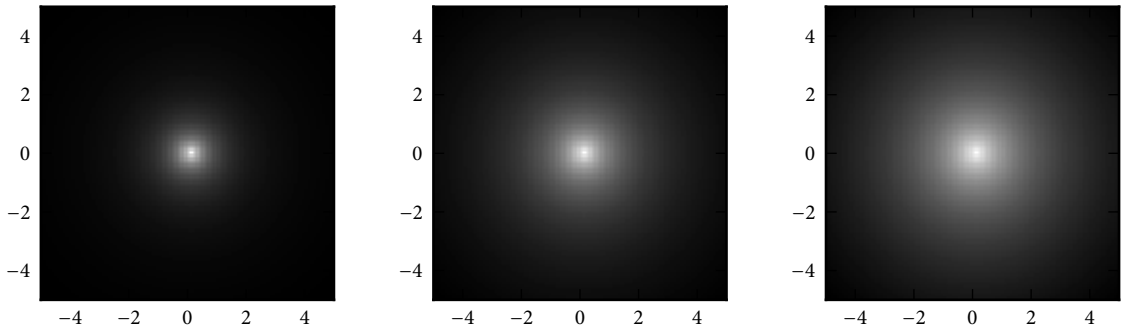
2.2.2.3 Influência simples

Como a intensidade da influência de uma unidade decai com a distância dessa unidade ao ponto onde se está calculando a influência, algumas funções são comumente usadas para modelar a queda da intensidade.

Mais formalmente, dado um ponto p no ambiente, o valor, em p, da influência de uma unidade com influência inicial I_0, a uma distância d é dada por

$$I_d = f(I_0, d). \tag{2.1}$$

Sendo a distância normalmente definida como a distância, euclidiana, do ponto p à



(a) Quadrática (b) Linear (c) Raiz quadrada

Figura 2.6. Comportamento de funções de dispersão de influência. Aqui, supõe-se que há um único agente gerador de influência no centro do mapa. Os valores exibidos nos eixos representam a distância (horizontal e vertical) ao centro. (a) exibe uma dispersão quadrática. Já (b) exibe uma dispersão linear, enquanto (c) exibe uma dispersão que depende da raiz quadrada da distância. As equações que regem as influências representadas em (a), (b) e (c) são apresentadas em (2.2), (2.3) e (2.4), respectivamente.

unidade u.

Há três métodos comumente usados como função de influência da equação (2.1). Na influência quadrática, exibida na equação (2.2), o valor da influência de uma unidade diminui com o quadrado da distância dessa unidade. Na linear, exibida na equação (2.3), o valor diminui como função direta da distância e na equação (2.4) é exibida uma que decai em função da raiz quadrada da distância. Supondo um mapa com apenas uma unidade (militar) na posição central, a figura 2.6 exibe o efeito das três funções de influência apresentadas.

$$I_d = \frac{I_0}{(1+d)^2} \tag{2.2}$$

$$I_d = \frac{I_0}{1+d} \tag{2.3}$$

$$I_d = \frac{I_0}{\sqrt{1+d}} \tag{2.4}$$

É importante notar que, para que a análise tática por mapas de influência funcione, é necessário que cada unidade no campo de batalha possua um valor de influência. No entanto, não é necessário que esse valor seja o mesmo que o poder ofensivo, ou defensivo, de uma unidade: uma unidade de reconhecimento, por exemplo, pode ter grande influência, por poder iniciar ataques, sem ter poder de fogo.

A influência de um exército é dada simplesmente pela soma da influência de cada unidade pertencente a esse exército. O lado que tiver a maior influência em uma região se torna o lado

de maior poder sobre a região. O nível de controle, por sua vez, é dado pela diferença entre as influências do lado mais forte e do segundo mais forte. Se a diferença for grande, é possível dizer que a região está segura.

2.2.2.4 Cálculo da influência

Para calcular o mapa de influência é necessário levar em consideração todas as unidades em todas as posições do ambiente. Exceto em casos simples, essa tarefa demanda muito tempo computacional, pois a complexidade de tempo do cálculo de um mapa de influência é $\mathcal{O}(nm)$, enquanto a complexidade de espaço é $\mathcal{O}(m)$, onde m é o número de posições do ambiente e n o número de unidades. Com alguns milhares de unidades e alguns milhões de posições possíveis, bilhões de cálculos seriam necessários.

Portanto, para reduzir o tempo de computação, algumas técnicas são usadas, como limitação do raio de efeito, filtros de convolução e *map flooding*. Nesta seção o método de limitação do raio de efeito será apresentado. Os outros métodos são descritos na literatura, como em Millington [2006].

Na limitação de raio de efeito, além de sua influência básica, cada unidade também possui um raio máximo de influência. A partir desse raio máximo, a unidade não exerce mais influência. É possível definir um raio máximo para cada unidade, ou usar uma função que remove o valor da influência se ela for menor que o limite dado pela função. Assumindo uma função de decaimento de influência linear, como a função da equação (2.3), o raio da influência é dado pela equação (2.5), com I_t sendo o limite da influência.

$$r = \frac{I_0}{I_t - 1}, \tag{2.5}$$

Esse método permite calcular a influência de todas as unidades do jogo, somando-as apenas dentro de seu raio de influência. Com essa otimização, a complexidade de tempo é reduzida para $\mathcal{O}(nr)$, com r sendo o número de locais dentro do raio médio das unidades[4].

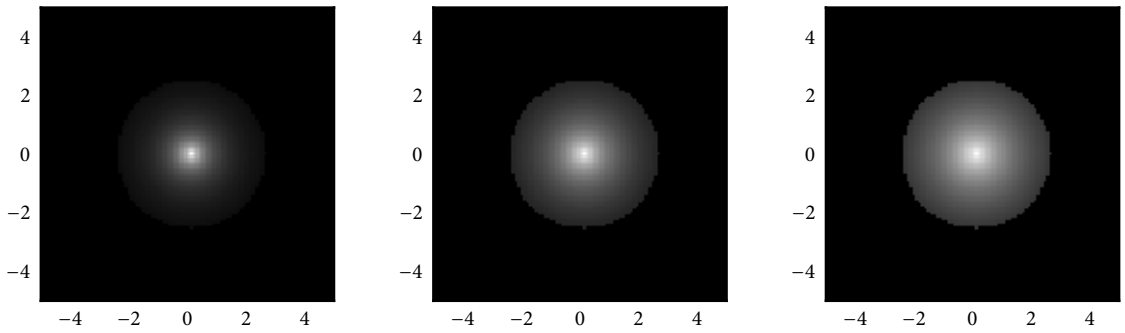
A figura 2.8 exibe o efeito do raio limitado aplicado às figuras 2.6a, 2.6b e 2.6c. Já a figura 2.7 exibe a influência de um pequeno exército com raio de influência limitado.

2.2.2.5 Aplicações interessantes

Mapas de influência permitem que a IA conheça as áreas seguras e inseguras de um mapa. Por essa característica, eles podem ser usados para planejar locais para ataque ou para guiar a movimentação de unidades. Um agente inteligente poderia, por exemplo, ao receber uma

[4] Apesar de, a primeira vista, não parecer uma grande melhoria, o raio médio r é muito menor que o número de posições em um nível do jogo, tornando esse método mais eficiente em tempo de execução.

Figura 2.7. Renderização de depuração de um mapa de influência de raio limitado gerado para um pequeno exército. A intensidade da influência das unidades é representada por quadrados vermelhos de opacidade variável. Quanto mais opacos, maior a influência naquela célula.



(a) Quadrática (b) Linear (c) Raiz quadrada

Figura 2.8. Comportamento de funções de dispersão de influência com raio limitado. Aqui, supõe-se que há um único agente gerador de influência no centro do mapa. Os valores exibidos nos eixos representam a distância (horizontal e vertical) ao centro. (a) exibe uma dispersão quadrática. Já (b) exibe uma dispersão linear, enquanto (c) exibe uma dispersão que depende da raiz quadrada da distância. Todas elas foram limitadas a um raio de 2.5 unidades de distância. As equações que regem as influências representadas em (a), (b) e (c) são apresentadas em (2.2), (2.3) e (2.4), respectivamente.

ordem de "atacar território inimigo", observar o mapa de influência atual, identificar a região menos segura e escolhê-la como ponto de partida para o ataque.

Uma característica dessa abordagem é que flancos naturalmente são representados como pontos de pouca segurança e, portanto, uma IA que prefere atacar pontos fracos tenderá a preferir ataques pelos flancos.

2.2.2.6 Dados das células

Além de conter informações de influência, as células de um mapa podem armazenar outras informações[5], como:

Patrimônio Uma estimativa do valor patrimonial de um jogador numa célula, como parte de uma vila ou de uma base miliar;

Visibilidade da área Um número indicando por quanto tempo a região esteve visível, ou invisível, para o jogador;

Contagem de corpos Estatísticas de quantas unidades foram destruídas numa célula e quando;

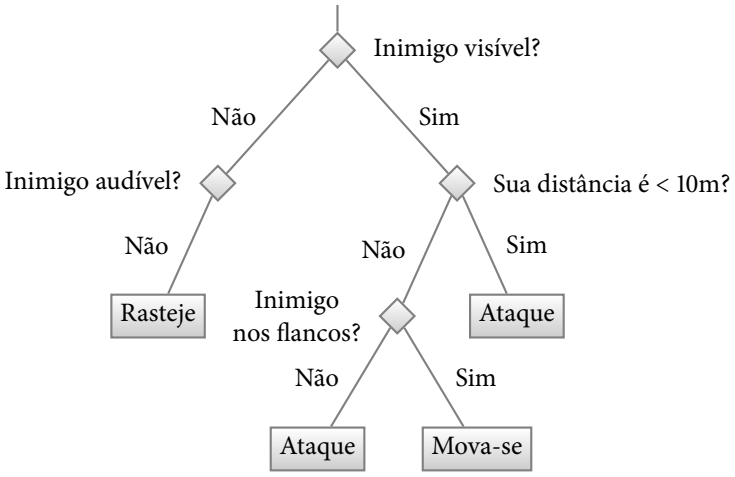
Recursos Quantidade de recursos ainda disponíveis para exploração;

Dificuldade de passagem Uma estimativa da dificuldade de atravessar uma região do mapa separada por tipo de movimento (voo, terrestre, etc). Esse valor pode ser usado para melhor propagar a influência para células vizinhas. Uma variante é o armazenamento de oito valores de dificuldade de passagem, um para cada direção que sai da célula.

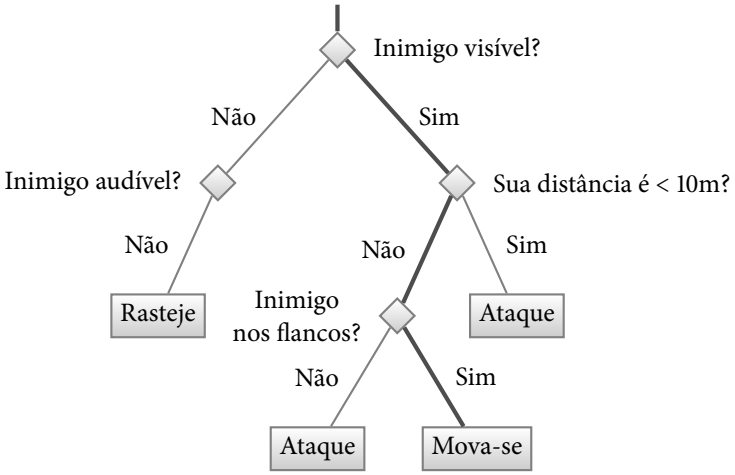
A partir do momento que mais informações são armazenadas em um mapa de influência, é possível combiná-los de forma a obter algum valor de utilidade desejável.

Como exemplo da vantagem de combinar mapas de influência, pode-se considerar o problema de exploração do mapa em um jogo RTS com FOW: uma boa IA fará reconhecimento do terreno com frequência. Uma boa heurística de exploração é pontuar as células do mapa que não foram atualizadas por mais tempo. Outro exemplo trata da coleta de recursos, em que construções usadas para coleta de recursos são tipicamente mais úteis em áreas fáceis de defender e que estejam próximas de grandes quantidades de recursos exploráveis. A combinação de mapas de influência com essas características permite escolher melhor onde posicionar os centros de coleta.

[5] Ou mais de um mapa pode ser implementado, cada um representando determinada característica



(a) Árvore de decisão inicial, sem decisões realizadas.



(b) Árvore de decisão inicial, com uma decisão de se mover realizada. O caminho da decisão na árvore é realçado.

Figura 2.9. Exemplo de árvores de decisão. A árvore de decisão codifica possibilidades de abordagem a um inimigo. Em (a) é exibida uma árvore sem decisões tomadas e em (b) o agente tomou uma decisão de afastar-se do inimigo.

2.2.3 Árvores de decisão

Uma árvore de decisão é uma ferramenta de suporte ao processo de tomada de decisão que usa uma árvore para modelar decisões e suas possíveis consequências. Em jogos elas são usadas para controlar de estágios básicos de animação a IA estratégica e tática [Millington, 2006, Capítulo 5].

A estrutura de dados e o algoritmo básicos são bem simples e de fácil implementação. Há, no entanto, variantes que tornam essas árvores mais complexas e poderosas.

Uma árvore de decisão é composta de pontos de decisão e cada uma possui uma decisão

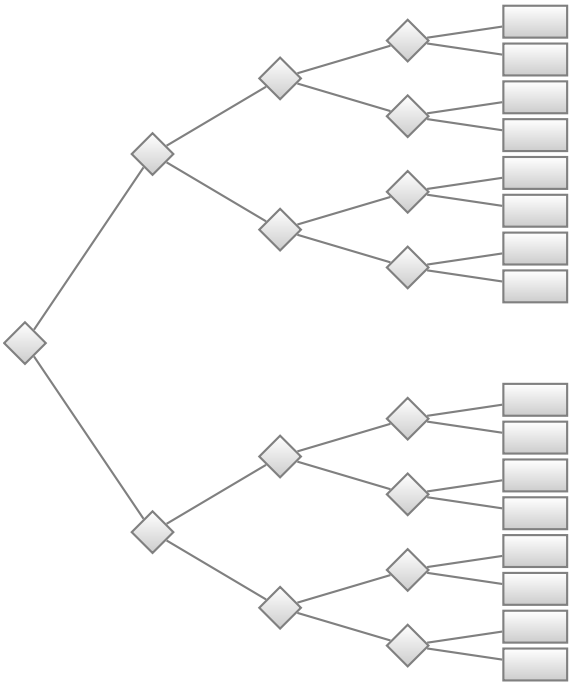


Figura 2.10. Árvore de decisão com 16 ações.

inicial: a raiz. A partir da raiz, cada nó da árvore é avaliado e, dependendo do resultado obtido, um nó filho do nó atual, até que um nó terminal seja alcançado e uma ação possa ser desempenhada. Cada decisão é feita baseada no conhecimento atual do mundo que o agente possui. Um exemplo de árvore de decisão é exibido na figura 2.9. É importante notar que uma mesma ação pode ser colocada em diversos nós da árvore. Na figura 2.9, por exemplo, ações de ataque podem ser encontradas em dois nós da árvore e o agente tenderá a atacar um inimigo, exceto quando não puder vê-lo ou quando estiver flanqueado.

2.2.3.1 Decisões

As decisões na árvore são normalmente simples e tipicamente fazem apenas verificação de valores, sem avaliar expressões booleanas, apesar de nada impedir que sejam feitas. Alguns tipos de dados e de decisões possíveis com esses tipos são exibidos na tabela 2.1. Além de tipos nativos, é comum que métodos de objetos em linguagens orientadas a objetos sejam usados como nós de decisão, o que permite processamento mais complexo nos nós das árvores.

2.2.3.2 Complexidade das decisões

Como decisões são construídas como uma árvore, o número de decisões que precisam ser consideradas é, normalmente, muito menor que o número de decisões na árvore. A figura

Tabela 2.1. Tipos de dados e de decisões comumente usados em árvores de decisão.

Tipo de dados	Decisão
Booleano	Valor verdadeiro ou falso
Enumeração (conjunto de valores)	Correspondência a um dos valores
Valor numérico	Valor pertencente a um intervalo
Vetor 3D	Módulo do vetor está dentro de um intervalo

2.10 exibe uma árvore com 15 decisões diferentes e 16 ações possíveis. Ao analisá-la é possível perceber que apenas quatro nós de decisão serão considerados em qualquer execução.

A vantagem das árvores de decisão é que elas são simples de se construir e podem ser construídas em estágios. Ou seja, uma árvore simples pode ser implementada inicialmente e, enquanto a IA é testada no jogo, decisões adicionais podem ser criadas para tratar casos especiais ou adicionar novos comportamentos.

2.2.3.3 Ramificação

Nos exemplos exibidos, todas as árvores são binárias, pois há decisões para apenas duas opções. Não há nada que impeça a criação de árvores com mais opções de ramificação, nem que impeça que decisões diferentes tenham quantidade de ramos diferentes.

Suponha, por exemplo, que há um guarda em uma estrutura militar. O guarda precisa decidir baseado no estado de alerta da base. Esse nível de alerta poderia pertencer ao conjunto de estados "verde", "amarelo", "vermelho", ou "preto". Ao usar as árvores de decisão apresentadas anteriormente, seria necessário construir uma árvore com três nós de decisão. Nela, um mesmo valor (de alerta) precisaria ser verificado por três vezes, que pode não ser um problema, caso ela seja configurada de forma que os nós para ações mais comuns apareçam primeiro na árvore. Ainda assim, é possível que a árvore precise realizar o mesmo trabalho diversas vezes até tomar a decisão final. Ao permitir uma árvore com número maior de ramos, é possível construir uma com apenas um nó de decisão, o que permite uma árvore com menor número de decisões.

Apesar das vantagens de árvores com vários ramos, é mais comum que apenas árvores binárias sejam usadas. Um dos motivos é que o código para os ramos é simplificado para um conjunto de testes binários. Apesar da simplicidade da árvore de múltiplos ramos, seu tempo de implementação não difere muito do tempo de árvores binárias. Outro motivo é que as árvores binárias de decisão são mais facilmente otimizadas e que alguns algoritmos de aprendizado de máquina precisam que elas sejam binárias Millington [2006].

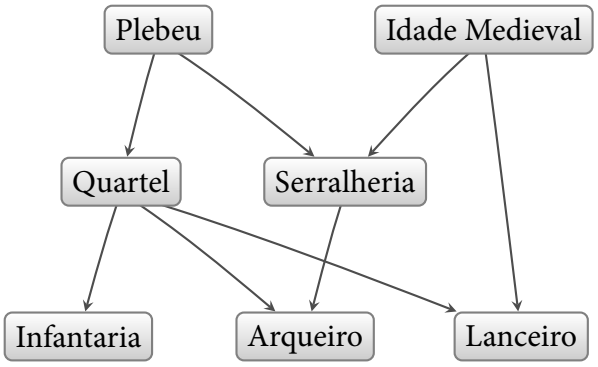


Figura 2.11. Exemplo de grafo de dependências tecnológicas (grafo tecnológico) simplificado. Neste grafo, as dependências de uma construção estão na origem da aresta direcionada. A construção de um *Lanceiro*, por exemplo, requer que a *Idade Medieval* tenha sido alcançada e que um *Quartel* tenha sido construído por um *Plebeu*.

2.2.3.4 Representação do conhecimento

Conforme comentado, as árvores de decisão podem trabalhar diretamente com tipos de dados primitivos. Um de seus benefícios é que normalmente não é necessário traduzir o formato de representação dos dados do código para um formato usável pela árvore de decisão.

Um efeito colateral dessa característica é que as árvores de decisão são comumente implementadas de modo a acessarem o estado do jogo diretamente, o que pode levar a erros de programação difíceis de encontrar. Se uma árvore de decisão for pouco usada, pode não ficar óbvio em que pontos há erros de programação. Durante o desenvolvimento, a estrutura do estado do jogo muda regularmente, o que pode quebrar decisões baseadas em estruturas particulares, o que pode levar a mais problemas. Por essas razões, alguns desenvolvedores preferem isolar todo o acesso ao estado do jogo por uma camada de abstração.

2.2.4 Grafos de dependência

Algumas técnicas, descritas em Tozour [2001b], foram usadas para avaliação do estado do jogador e de seus dos inimigos, quando observados. A técnica básica é a dos grafos de dependência, exemplificada na figura 2.11. O grafo de dependência é uma estrutura de dados que modela todas as dependências entre os diferentes ativos de um jogo.

Considerando o grafo da figura 2.11 como um grafo real de jogo e que um jogador quisesse treinar arqueiros, ele precisaria, primeiro de uma *Serralheria* e de um *Quartel*. Ambos construídos por *Plebeus*.

A forma principal de representação de dependências é a de dependência existencial, que indica quais condições deve ser satisfeitas antes que um determinado conjunto de ativos possa ser construído. As dependências de recursos também são representadas por dependências

existenciais, já que a falta de recursos disponíveis implica na impossibilidade de construir o objeto desejado.

Assim como nos mapas de influência, para poder planejar suas ações, um jogador deve manter diversos grafos de dependência, um para si mesmo e um para cada um de seus oponentes.

2.2.4.1 Planejamento econômico

Uma das aplicações mais comuns para grafos de dependência é usá-los na busca por um objetivo. Como exemplificado na figura 2.11, exibida na página 25, a partir do grafo de dependências, é possível saber a ordem de construção de unidades para alcançar um objetivo.

A escolha de que dependências preencher primeiro pode ser vista como uma solução de compromisso entre uma reação ao presente e planejamento para o futuro. Uma IA puramente reativa daria prioridade à construções que pudesse edificar imediatamente. Já uma focada em planejamento tentaria analisar todos os nós do grafo e encontrar um objetivo de longo prazo digno de ser perseguido.

Um grafo de dependência pode ser usado para analisar os pontos fortes e fracos das forças inimigas e, então, apontar suas dependências mais vulneráveis. Os fatores para definir uma dependência como "vulnerável" são seu custo de produção e grande número de filhos, pois dependências de alto custo podem ser bons alvos simplesmente por que são caras e dependências com muito filhos são bons alvos por que interrompem a cadeia produtiva inimiga.

2.2.4.2 Inferência

Outra utilidade de usar grafos de dependência é que é eles proveem uma base para inferência sobre recursos dos inimigos e suas estratégias mais comuns baseado em observações incompletas. Por exemplo, saber que um inimigo possui um "Barracks" permite saber que ele possui "Grunts" ou é capaz de treiná-los. Ou, ainda, se um "Dragon" é avistado, é possível ter 100% de certeza que o inimigo possui, ou possuía no momento de criação do "Dragon", um "Dragon Roost".

Essa forma de inferência baseada em dependências pode, ainda, ser usada para, ao avistar uma unidade avançada, supor que há uma probabilidade do inimigo possuir outros tipos de unidades avançadas num processo de inferência bayesiana, técnica apresentada em Russell & Norvig [2003].

2.3 Trabalhos relacionados

Esta seção apresenta os trabalhos relacionados a este e é organizada da seguinte forma: primeiro são apresentados trabalhos relacionados à Inteligência Artificial em jogos RTS em termos gerais e, posteriormente, trabalhos com tema mais próximo ao deste. Seu objetivo é fazer um levantamento das técnicas desenvolvidas nesta área, assim como os problemas associados e as soluções adotadas. Também são discutidos trabalhos que usam técnicas da robótica em jogos, um tópico de pesquisa inicial deste trabalho. Esta revisão proporciona o embasamento necessário para a compreensão de onde este trabalho se situa no universo de jogos RTS e as bases teóricas para seu desenvolvimento.

Comparada à história da pesquisa em IA, a pesquisa em jogos digitais é nova. Um dos primeiros trabalhos a motivar a pesquisa em jogos foi apresentado em Laird & Jones [1998], onde os autores, ao desenvolver pilotos artificiais para um projeto da Agência de Projetos de Pesquisa Avançada de Defesa (*Defense Advanced Research Projects Agency*, ou DARPA), concluíram que muitos dos problemas por eles encontrados nesse trabalho eram compartilhados pelos desenvolvedores de jogos que buscavam produzir oponentes realísticos. Em um trabalho subsequente, van Lent et al. [1999], os autores integraram a arquitetura cognitiva Soar a jogos de tiro em primeira pessoa para a implementação de oponentes inteligentes, ou *bots*, que usavam sua base de conhecimento para abstração de detalhes do jogo.

Posteriormente, em Laird & van Lent [2000], foi discutido que os jogos eletrônicos seriam um ambiente onde os objetivos fundamentais da IA, como compreender e desenvolver sistemas inteligentes que possuíssem capacidades humanas, poderiam ser buscados e desenvolvidos.

Há outros trabalhos que integram a Soar a jogos, como Laird [2002], que apresenta o progresso do grupo Soar/Games no período de 1998 a 2002 e cita os resultados obtidos com trabalhos baseados em jogos como Quake 2, Descent 3 e Unreal Tournament. Nesse último, é construída uma expansão para o jogo de modo a apresentar uma nova história e novas decisões de IA. Os detalhes da integração de Unreal Tournament à Soar se encontram em Magerko et al. [2004], que detalha os requisitos da modificação, o projeto do software e os principais componentes da modificação e da IA.

Mais recentemente, Wintermute et al. [2007] integrou a Soar a um motor chamado ORTS [Buro, 2002] (apresentado na seção 2.1.2.1) para a criação de agentes capazes de jogar jogos de RTS. Nesse trabalho, um *middleware*[6] capaz de agrupar unidades ou construções, coordenar o planejamento de trajetórias e de controlar o comportamento em baixo nível das unidades foi criado para permitir que o computador fosse capaz de raciocinar sobre jogos de RTS de forma semelhante à humana, usando abstrações para tratar elementos do jogo.

[6] Um *middleware* é um *software* que atua como um intermediário entre duas ou mais aplicações que precisam se comunicar.

Alguns trabalhos que usam o arcabouço do ORTS foram publicados nos últimos anos, como Chung et al. [2005], que usa simulações de Monte Carlo para planejamento em jogos de RTS. Nesse trabalho, uma máquina de busca para planejamento em domínios estocásticos, MCPlan, foi criada, com uma implementação de teste e caracterização de parâmetros de desempenho.

Sailer et al. [2007], de membros do grupo de pesquisa criador do ORTS, apresenta um arcabouço de planejamento que usa simulação estratégica em conjunto com uma estratégia de aproximação baseada em equilíbrio de Nash. Nesse trabalho, um problema de movimento de tropas é resolvido e comparado a uma abordagem baseada em *scripts*, técnica comumente encontrada em jogos RTS comerciais. A solução usada nesse artigo sugere um ganho de desempenho sobre a abordagem tradicional, mas carece de dados comparativos com implementações que utilizem o nível de detalhes encontrado em implementações comerciais.

Stoykov [2008], por exemplo, usou o ORTS para implementação de uma abordagem competitiva com objetivo de melhorar o projeto de IA para simulação militar. Nesse trabalho é apresentada uma proposta de metodologia a ser seguida para melhorar projetos de agentes de IA. Outra contribuição importante é que ele demonstra a utilidade do gerente de torneios do ORTS para amostragem e realização de várias simulações. Uma das deficiências da abordagem desse trabalho é que nele o interesse está em apenas construir um agente inteligente capaz de vencer outro agente já existente. Ao alcançar este objetivo, o autor se dá por satisfeito, quando poderia usar sua própria abordagem competitiva para melhorar ainda mais seu agente.

Alcázar et al. [2008] usa uma abordagem com a Linguagem de Definição do Domínio de Planejamento (*Planning Domain Definition Language*, ou PDDL) para planejamento no ORTS no modo de jogo RTS completo. Nesse trabalho, os autores apresentam as decisões tomadas para modelar um dos domínios da competição do ORTS. Uma contribuição importante, dada a dificuldade de especificar condições e efeitos usando métricas como custos e recompensas.

Ainda em abordagens baseadas em planejamento, Hoang et al. [2005] explora o uso de representações em Rede de Tarefas Hierárquica (*Hierarchical Task Network*, ou HTN) e *Task-Method-Knowledge* (TMK) para programação de IA estratégica. A abordagem em TMK é particularmente interessante, pois modelos TMK são usados pelo *Testbed for Integrating and Evaluating Learning Techniques* (TIELT)[7] que, por sua vez, torna a avaliação de sistemas de decisão mais simples. O trabalho conclui que planejadores HTN podem ser usados para gerar planos corretos para comportamento da IA.

Já Lee-Urban et al. [2007] apresenta o uso de HTN em jogos RTS com o Planejador Simples Hierárquico Ordenado (*Simple Hierarchical Ordered Planner*, ou SHOP) para desenvolver uma arquitetura de planejamento e aprendizado de IA chamada Learn2SHOP. O trabalho de-

[7] Uma ferramenta patrocinada pela DARPA com o objetivo de integrar algoritmos de aprendizado de máquina com jogos de computador.

monstra, através de experimentos, que a integração é promissora e que a abordagem proposta é capaz de desempenhar simulações no ambiente para obter informações sobre o estado do jogo.

Em Hagelbäck & Johansson [2008], o ORTS é usado para a implementação de Campos Potenciais Multi-Agentes (*Multiagent Potential Fields*, ou MAPF), uma forma de Campos Potenciais (*Potential Fields*, ou PF) para múltiplos agentes, para o controle de movimento de unidades. O objetivo principal desse trabalho é descobrir se os MAPF são uma abordagem viável para a implementação de *bots* configuráveis para jogos de RTS. No artigo é apresentada uma metodologia para a construção de uma solução baseada em MAPF com fases bem definidas. Além disso, são descritos com detalhes os PF implementados. Apesar de o *bot* implementado ter ganhado cerca de apenas 30% das batalhas no torneio do ORTS de 2007, os autores concluíram que, dado mais tempo para pesquisa em como implementar PF em jogos, não haveria por que acreditar que PF não poderiam ser usados com sucesso em jogos RTS. Hagelbäck & Johansson [2009] apresenta melhorias ao trabalho original e demonstra a evolução pela qual passou o agente desenvolvido que, de perdedor de torneios, passou a campeão de todas as categorias em que participou no torneiro de jogos RTS do *Artificial Intelligence and Interactive Digital Entertainment Conference* (AIIDE) de 2008.

Como em Hagelbäck & Johansson [2008], outros trabalhos usaram técnicas originárias da robótica em jogos eletrônicos, como Khoo et al. [2002], que implementou dois sistemas de controle de Personagens Não Jogáveis (*Non Player Characters*, ou NPCs) baseados em técnicas comportamentais. Nesse trabalho, o jogo de tiro em primeira pessoa Half-Life, em conjunto com o *kit* de desenvolvimento de *bots* FlexBot e a Linguagem Robótica Genérica (*Generic Robot Language*, ou GRL), é usado como plataforma de implementação de dois *bots*, Ledgewalker, um conjunto de comportamentos que executam em paralelo e arbitrados por uma pilha de prioridades, e Groo, um *bot* mais complexo criado para resolver os problemas encontrados no Ledgewalker. Groo possui comportamentos separados, tendo controladores alocados para controle de movimento outros controladores para os comportamentos restantes. A conclusão a que se chegou nesse trabalho foi que o código de ambos os *bots* era eficiente e estável, concluindo que técnicas comportamentais poderiam ser usadas em jogos.

Mamei & Zambonelli [2004] usa campos potenciais em Quake III, um jogo de tiro em primeira pessoa. Para isso, dois problemas de coordenação foram propostos: definição de um ponto de encontro para os *bots* no mapa e encurralamento do jogador, a presa. No primeiro caso, a política de coordenação expressa que cada *bot* percorra o campo potencial como uma força de atração até que todos convirjam para o ponto combinado. No segundo caso, os *bots* são atraídos pelo campo da presa para alcançá-la ao mesmo tempo em que repelem o campo de outros *bots* para se afastarem. O efeito final é que, além de se aproximarem da presa, os *bots* a circundam. Esse trabalho sugere que o uso de campos potenciais em jogos é promissor, fato

corroborado por Hagelbäck & Johansson [2008].

da Silva Corrêa Pinto & Alvares [2006], por sua vez, faz uma revisão da aplicação de duas técnicas da robótica comportamental, *subsumption* e Redes Comportamentais Estendidas (*Extended Behavior Networks*, ou EBNs). Esse trabalho conclui que arquiteturas robóticas comportamentais possuem vantagens como rápida execução, ausência de necessidade de planejamento, tarefa que costuma tomar muito tempo, e robustez, garantida pelo fato de que, ainda que uma camada comportamental falhe, outras camadas continuarão a operar. Esse trabalho é interessante por citar algumas desvantagens, como as da arquitetura *subsumption*, onde os módulos comportamentais não são facilmente alteráveis após sua implementação, a dependência que os módulos de camadas superiores possuem sobre os de camadas inferiores e sua maior limitação: a necessidade de especificar explicitamente cada conexão entre os módulos e sua arquitetura fixa. A abordagem usando EBN, por sua vez, não sofre do problema de dependências entre módulos que afeta a arquitetura *subsumption*.

Ainda em jogos de tiro em primeira pessoa, Orkin [2005] apresenta as lições aprendidas durante a implementação do planejamento em tempo real dos NPCs do jogo F.E.A.R. Sua contribuição é deveras interessante, pois apresenta todos os componentes de IA presentes em um jogo considerado pela crítica especializada como possuidor de uma IA de bom desempenho. Esse trabalho também apresenta exemplos de comportamentos emergentes, que surgiram sem a necessidade de programação explícita. Característica esta apresentada pela arquitetura dos módulos de IA do jogo.

Bergsma & Spronck [2008] apresenta uma arquitetura de IA que possui raciocínio espacial baseado em mapas de influência, técnica apresentada em Tozour [2001a]. Nessa abordagem, os valores de influência indicam o desejo de uma unidade em mover-se para células do mapa. Sua abordagem é interessante por combinar mapas de influência a uma rede neural. O objetivo dessa combinação foi o de determinar o desejo de movimentação de cada agente. Além disso, com essa abordagem foi possível conhecer, a cada instante, quais células do mapa deveriam ser atacadas. As estratégias foram geradas e melhoradas por um algoritmo evolutivo. Ainda é comentado que o uso de um critério de evolução no qual a geração atual precisa ser apenas melhor que a anterior é insuficiente para criação de uma estratégia que seja boa globalmente. Logo, uma das contribuições desse trabalho é a demonstração de que, a cada nova geração, a atual deve ser testada contra todas as vencedoras anteriores para obter uma melhor estratégia global. No entanto, nesse trabalho são abordados apenas jogos Estratégia Baseada em Turnos (*Turn-Based Strategy*, ou TBS) e seria interessante investigar o desempenho desse método em jogos RTS.

Já Ponsen et al. [2006] apresenta uma abordagem evolutiva para geração de táticas em jogos RTS. As táticas são selecionadas pela técnica de *dynamic scripting*, uma abordagem de Aprendizado por Reforço (*Reinforcement Learning*, ou RL) para criação de IA adaptativa capaz

de aprender, durante o jogo, quais táticas devem ser selecionadas para jogar de maneira efetiva. Uma contribuição apresentada nesse trabalho é uma simplificação do domínio do jogo Wargus baseada no tipo das construções edificadas por determinado jogador. Essa simplificação se mostrou efetiva e possibilitou que a computação envolvida no processamento da IA se tornasse viável para execução em tempo real.

Em Balla [2009] é abordado o problema de planejamento tático em jogos RTS. Sua contribuição principal é a descrição de uma formulação abstrata para o problema de planejamento de assalto tático[8]. Neste trabalho são consideradas duas ações básicas: "*Join*" (agrupar) e "*Attack*" (atacar), sendo objetivo da primeira reunir grupos de agentes e o da segunda de enviar grupos selecionados para o ataque. Experimentos demonstraram que diversas formas de organizar essas ações resultam em desempenho diferente do exército e também demonstraram que o agente desenvolvido era capaz de vencer todos seus oponentes táticos. Uma deficiência desse trabalho é que todos os experimentos foram desenvolvidos com um único tipo de unidade em mente e, portanto, para unidades diferentes ou grupos heterogêneos, os resultados podem variar.

Baumgarten et al. [2009] apresenta uma abordagem de desenvolvimento de um *bot* capaz de aprender através da análise de registros de jogos antigos e se adaptar para novos jogos pelo uso de Raciocínio Baseado em Casos (*Case-Based Reasoning*, ou CBR). Ainda que de maneira superficial, todo o processo de criação de um *bot* de jogos RTS é apresentado. Ainda é apresentado um conjunto extensivo de referências para as diversas técnicas usadas na implementação. Outra contribuição interessante desse trabalho é relacionada ao fato de seus resultados serem aplicados a um jogo existente, provando que a abordagem desenvolvida pode ser usada pela indústria.

Weber & Mateas [2009] apresenta uma abordagem para obtenção de *replays* de um jogo RTS para predição de estratégia usada pelo oponente. Com o objetivo de construir um modelo geral do "jogar" de um especialista do jogo Starcraft, o trabalho empregou diversas técnicas de aprendizado de máquina em mais de 5000 dados de jogos completos de jogadores experientes. Os resultados experimentais exibidos foram interessantes, dado que o método utilizado foi capaz de predizer, com até quatro minutos de antecedência, quais seriam as ações do oponente. Apesar de possuir objetivos semelhantes com o desta dissertação, a técnica usada no trabalho supracitado não pode ser aplicada neste caso, pois há poucos dados para serem extraídos, devido à ausência de conjunto oficial de *replays* para o jogo utilizado.

Metoyer et al. [2010] apresenta experimentos sobre o aprendizado de jogos RTS com o objetivo de esclarecer que tipo de informações devem estar presentes em um sistema de

[8] O objetivo das táticas em um RTS, normalmente, é o de destruir todas as tropas inimigas e é tipicamente alcançado através de uma sequência de assaltos bem orquestrados, enquanto uma postura defensiva adequada é mantida.

anotação de registros de jogos RTS para guiar o aprendizado de máquina no caso de haver poucos dados para extração.

Aha et al. [2005] apresenta o *Case-based Tactician* (CaT), agente de IA que usa CBR com o objetivo de vencer jogos RTS. Para isso, foi desenvolvido um algoritmo de recuperação de planos que, ao usar três fontes de conhecimento de domínio, retiram a necessidade de um oponente estático para a IA. CaT não tenta reconhecer planos adversários. Ao invés disso, ele adquire planos ligados à aplicação de um sub-plano em um dado estado, aprende a selecionar sub-planos e os executa em um ambiente mais complexo. Algumas discussões interessantes são apresentadas, como uma consideração sobre o tempo gasto em cada parte gerenciada em um jogo e uma estimativa do tamanho do espaço de estados de um jogo RTS.

Fagan & Cunningham [2003] diz que uma das formas de se usar a IA no projeto de jogos é pela criação de NPCs mais realistas. Sua ideia principal é a de observar o comportamento do jogador e identificar padrões recorrentes usando Reconhecimento de Planos (*Plan Recognition*, ou PR)[9]. A partir dos padrões observados, a abordagem descrita tenta prever as próximas ações do jogador. Para o simples domínio apresentado no trabalho, a abordagem Reconhecimento de Planos Baseado em Casos (*Case-Based Plan Recognition*, ou CBPR) é capaz de prever satisfatoriamente o comportamento do jogador mesmo sem uma biblioteca de planos personalizada, necessária em domínios mais complexos.

Madeira et al. [2004] descreve uma metodologia para iniciar o processo de aprendizado de um *bot*. A metodologia apresentada supõe que existe um *bot* que deve ser melhorado e, portanto, não é muito útil para a versão inicial do *bot*. Sua maior contribuição é a apresentação de um método para decomposição do problema para aplicação das técnicas de aprendizado.

Madeira et al. [2005], motivado pelo fato de que manter estratégias fixas durante diferentes partidas de jogos ser uma abordagem ruim, apresenta um algoritmo para geração de representações adequadas a jogos de guerra com o intuito de tratar das dificuldades de aplicação de RL a esses jogos. Para cumprir com seu objetivo, algumas soluções são apresentadas, como a decomposição do processo de tomada de decisão em um estrutura hierárquica; abstração do espaço de estados do jogo; obtenção de dados de treinamento pela adoção de um mecanismo de treinamento inicial[10] e generalização da experiência obtida. Apesar de não tratar de jogos RTS, a abordagem apresentada parece promissora também nesse domínio.

Madeira et al. [2006] sumariza os resultados obtidos em trabalhos anteriores com o objetivo de apresentar uma abordagem de RL para jogos de estratégia baseados em turnos. A

[9] PR é o processo em que um agente observa as ações de outro agente com o objetivo de inferir as ações, intenções e objetivos futuros do observado. As abordagens de PR ainda podem ser classificadas de acordo com o fato do processo ser deliberado ou *keyhole*: se o agente observado coopera para demonstrar suas intenções para o agente observador, o processo é deliberado. Caso contrário, o processo é *keyhole*.

[10] Descrito em trabalho anterior. [Madeira et al., 2004]

abordagem apresentada parece ter desempenho satisfatório, ainda que não fique claro, pelo texto, se houve variação de parâmetros importantes, como do mapa da batalha.

McCoy & Mateas [2008] apresenta um agente de IA composto de múltiplos componentes especializados para criação de um jogador completo de jogos RTS. Sua abordagem se dá através da análise de como jogadores especialistas conceituam a dinâmica de um jogo RTS. A partir dessa análise, o domínio do jogo é particionado em domínios de competência de acordo com os observados em humanos. Para isso, o trabalho conceitua as habilidades de jogadores especialistas para, então, descrever o arcabouço desenvolvido para criação do agente inteligente. Esse arcabouço possui diversos módulos especializados, como gerentes de recursos, gerentes de produção e assim por diante. Por fim, são apresentados resultados de que o agente desenvolvido é capaz de vencer, em mais de 50% das vezes, estratégias tidas como *benchmark*.

Ontañón et al. [2007], Sharma et al. [2007], Sugandh et al. [2008a], Sugandh et al. [2008b] e Mishra et al. [2008] apresentam trabalhos baseados em CBR desenvolvidos por um grupo de pesquisa em Computação Cognitiva.

Ontañón et al. [2007] usa CBR com o objetivo de fornecer a projetistas de jogos uma ferramenta para auxílio no treinamento de suas IAs. Assim, uma abordagem que usa extração de conhecimento comportamental a partir de demonstrações de especialistas é apresentada. A arquitetura proposta, portanto, permite que desenvolvedores de jogos sejam capazes de especificar o comportamento da IA através de demonstrações, simplificando o processo de desenvolvimento. Outras contribuições interessantes desse trabalho são uma arquitetura de planejamento e execução capaz de recuperar comportamentos mais adequados baseado no estado atual do jogo e a proposta de uma linguagem para expressar comportamentos e objetivos.

Sharma et al. [2007] combina as técnicas de CBR e RL para obter transferência de conhecimento[11] durante o jogo. Seus objetivos são o de desenvolver um sistema capaz de transferir o conhecimento em um RTS comercial combinando diversas técnicas de IA. Suas contribuições são a apresentação de uma arquitetura de múltiplas camadas que auxilia a IA tanto estrategicamente quanto taticamente e o compartilhamento das decisões de projeto para programação dos diversos níveis da arquitetura. Algumas deficiências desse trabalho são o fato da estratégia de mais alto nível ser imutável e de haver um pouca variação no conjunto de decisões táticas apresentado.

Sugandh et al. [2008a] estende os trabalhos anteriores ao usar ideias do planejamento tradicional para guiar a adaptação de planos em tempo real. Em seu arcabouço, quando um plano é obtido, um grafo de dependências é inferido e usado para adaptação do plano, o que permite ao sistema criar e adaptar planos de forma eficiente durante a execução de tarefas.

[11] Transferência de conhecimento é definido como o uso de conhecimento obtido em um conjunto de tarefas anteriores para melhorar o desempenho em uma tarefa semelhante, mas desconhecida.

Um problema da abordagem utilizada é que o sistema é apenas tão bom quanto os exemplos utilizados. Ou seja, caso sejam fornecidos exemplos ruins, o desempenho do sistema não será satisfatório. Sugandh et al. [2008b] apresenta mais detalhes sobre o trabalho anterior, contendo análises matemáticas e uma discussão mais aprofundada.

Mishra et al. [2008] apresenta o conceito de "situação" e um algoritmo de classificação de situação independente de domínio criado com o intuito de melhorar o desempenho da recuperação de planos em Planejamento Baseado em Casos (*Case-Based Planning*, ou CBP). Esse algoritmo foi, então, usado no arcabouço desenvolvido em trabalhos anteriores para experimentação, obtendo resultados promissores, sugerindo ser possível atender os requisitos de tempo real de um jogo RTS.

Após a apresentação dos trabalhos acima descritos, espera-se que o leitor seja capaz de compreender o estado atual da pesquisa em jogos RTS. Espera-se, também, que o leitor seja capaz de perceber que a pesquisa em jogos RTS é, de fato, promissora, dados os resultados obtidos nas diversas áreas estudadas.

Há alguns trabalhos da indústria de jogos que compartilham algumas semelhanças com o trabalho aqui desenvolvido. O primeiro é o sistema de tutoriais encontrado em jogos como Warcraft III, em que é criada uma campanha (ou outra subdivisão do jogo) com o único objetivo de apresentar a mecânica do jogo ao usuário novato. A abordagem de sistemas desse tipo difere da apresentada neste trabalho pelo fato de, terminada a campanha tutorial, o jogo supõe que o usuário é capaz de jogar sozinho e as dicas cessam de ser apresentadas. Além disso, as dicas nesses sistemas tutoriais têm o único objetivo de apresentar o funcionamento do jogo para que o usuário possa desempenhar ações básicas sem ensinar como tomar decisões melhores.

Outra abordagem da indústria é o sistema de conselheiros presente na franquia SimCity. Cada um dos conselheiros de SimCity é especialista em uma determinada área da administração de uma cidade. Exemplos incluem especialistas em segurança pública, saúde, educação e distribuição de energia. Os conselheiros, ao detectarem um problema na gestão do jogador, passam a chamar sua atenção através de ícones na interface do jogo. Os relatórios dos conselheiros indicam as fontes de problemas e sugerem, em alto nível, formas de como resolver esses problemas.

As diferenças entre a abordagem de SimCity e a deste trabalho é que, em SimCity não há necessidade de realizar microgerência de unidades. SimCity também não precisa ser executado em tempo real, já que é possível interromper a simulação, efetuar todas as mudanças necessárias no mundo e, então, prosseguir com a simulação. A IA dos conselheiros de SimCity também possui conhecimento total sobre o ambiente, o que contrasta com a restrição de usar apenas conhecimento local neste trabalho.

Capítulo 3

Sistema de apoio ao jogador

Neste capítulo serão apresentadas questões relacionadas à criação de um sistema de apoio ao jogador através da discussão dos aspectos do sistema apresentado neste trabalho. Para isso, primeiramente serão apresentadas as competências comumente presentes em um jogador experiente. Então, serão apresentados os objetivos do sistema de apoio ao jogador aqui apresentado, bem como uma discussão sobre as características das dicas usadas, do modelo do sistema e sua estrutura.

3.1 Competências presentes em um jogador experiente

Em geral, o domínio de um jogo envolve dominar as diversas habilidades envolvidas naquele jogo, seja através da experimentação, como humanos comumente aprendem, seja pela codificação de regras criadas por especialistas ou através de técnicas de aprendizado de máquina, no caso de jogadores de IA.

Em jogos RTS, como é necessário não só elaborar estratégias para vencer uma partida, mas também focar sua atenção em diversos aspectos do jogo. Seus jogadores, humanos ou não, devem, portanto, usar essas diversas competências, direta ou indiretamente, para alcançar a vitória. Por essas razões, o esforço para dominar jogos RTS é comparável ao do necessário para dominar o xadrez [McCoy & Mateas, 2008]. Como no xadrez, a capacidade de selecionar técnicas em diversos níveis de abstração em resposta a movimentos de oponentes é essencial.

O objetivo desta seção é apresentar as competências normalmente exigidas de um jogador, humano ou de IA para que ele tenha um bom desempenho em uma partida. Essas competências, quando implementadas em *bots*, tendem a ser mais bem definidas, já que podem ser implementadas em diferentes módulos que interagem entre si para definir a sequência de ações a ser executada. Já as pessoas tendem a jogar de forma mais integrada, sem separação clara do raciocínio nas diversas competências.

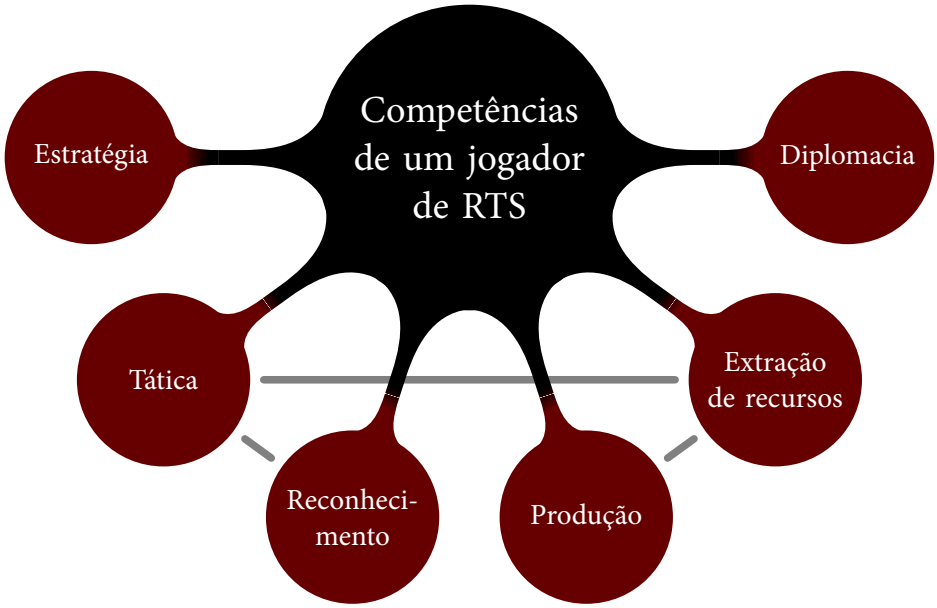


Figura 3.1. Mapa mental apresentando as competências esperadas em um jogador de jogos RTS. As linhas que conectam conceitos demonstram relacionamento entre eles. A estratégia se relaciona com todos os outros componentes e, portanto, suas ligações foram omitidas.

A divisão de competências é baseada na apresentada em McCoy & Mateas [2008] e é exibida na figura 3.1. A figura é organizada como um mapa mental, com as competências diretamente ligadas ao conceito central e as relações entre competências representadas por arestas no fundo da imagem.

3.1.1 Estratégia

A estratégia de um jogador pode ser definida como o conjunto de mais alto nível de ações a serem tomadas para alcançar seu objetivo. Ela tem o papel de definir o que será feito pelo jogador para que ele possa alcançar seu objetivo e coordena o estilo econômico, as construções presentes na base e o estilo geral de defesa e ataque de um jogador, além de outras características do exército.

É importante ressaltar que a estratégia costuma variar no decorrer do jogo. Isso acontece por que quando o jogo é iniciado, os jogadores não conhecem nada sobre a estratégia de seus oponentes. À medida que mais informações são descobertas sobre o jogo, as estratégias podem ser adaptadas para contrabalancear as estratégias dos oponentes.

3.1.2 Tática

A tática trata da disposição e da manobra de forças durante o combate ou na iminência dele. Esse conceito envolve a consideração de características do ambiente à volta da região de atuação das unidades envolvidas no combate, como a análise de características do terreno à volta do conflito.

Para ilustrar a interação entre tática e estratégia, suponha que um jogador decide atacar uma base inimiga em um instante de tempo. A decisão de atacar a base inimiga faz parte da estratégia do jogador, enquanto que a seleção de unidades e a efetivação do ataque fazem parte da tática a ser usada.

A microgerência de unidades também é responsabilidade da competência tática, pois ela é dependente do conjunto de unidades que compõem o o grupo militar. Por exemplo, não é responsabilidade das unidades individuais decidir qual unidade inimiga atacar, mas do módulo tático, que é capaz de ver o conjunto de aliados e inimigos.

3.1.3 Extração de recursos

Todas as atividades produtivas de um jogo RTS, como treinamento de unidades e edificação de construções, requerem o investimento de uma certa quantidade de recursos, já o tipo e a quantidade de recursos disponíveis variam de jogo para jogo.

A importância estratégica dos recursos em um jogo RTS é aumentada pelo fato da maioria dos recursos disponíveis não ser renovável, obrigando os oponentes a explorar o mapa em busca de novos recursos para que sua cadeia produtiva não seja interrompida e sua derrota se torne iminente.

A administração da extração de recursos é responsável por controlar quais trabalhadores extrairão recursos e quais serão realocados para tarefas de construção e reparos.

3.1.4 Produção

A gerência de produção em jogos RTS é responsável pelo treinamento de unidades, edificação de construções e desenvolvimento tecnológico. Através da gerência da cadeia produtiva, é possível cumprir requisitos estratégicos. Ao usar o grafo tecnológico como um grafo de dependência, conforme discutido na seção 2.2.4, é possível ordenar a construção e o treinamento de unidades para que um objetivo possa ser alcançado.

A construção e treinamento de unidades é, comumente, determinística. Ou seja, dado que os pré-requisitos de construção são cumpridos, a produção dura um tempo fixo, supondo que nenhuma das unidades envolvidas na produção será destruída antes do término da ativi-

dade. Assim, o conhecimento da cadeia produtiva de um jogo permite otimizar a ordem de construção de unidades para alcançar os objetivos de uma estratégia.

3.1.5 Reconhecimento

A exploração do ambiente possui um papel essencial em jogos RTS por dois motivos básicos:

1. Os recursos não são renováveis, portanto é necessário sempre buscar novas fontes de recursos para manter a cadeia produtiva do exército. Adicionalmente, a exploração de mais fontes de recursos simultaneamente aumenta o potencial do exército ser desenvolvido mais rapidamente.

2. É importante conhecer a posição da base inimiga para planejar um ataque efetivo e ter uma estimativa de onde virão os ataques inimigos. Existe também a possibilidade do inimigo construir novas bases para extração de recursos e o conhecimento da posição dessas expansões, normalmente menos protegidas que as bases principais, permite um ataque mais efetivo.

Pelos motivos supracitados, uma competência importante em jogadores de RTS é a capacidade de explorar o mapa com o objetivo de conhecê-lo melhor. Essa atividade de exploração normalmente é chamada reconhecimento. Também é através dela que é possível conhecer mais sobre o território inimigo para estimar a quantidade e o poder de suas forças, o que é crucial para a geração de um modelo do inimigo e seleção de uma estratégia apropriada.

3.1.6 Diplomacia

Apesar de ser pouco abordada em trabalhos acadêmicos, a diplomacia está presente em partidas *online* de jogos RTS, representada pela formação e destruição de alianças militares e comércio de recursos. Como é habitual que o objetivo dos jogos seja destruir todos os oponentes, as alianças costumam ser temporárias e, portanto, saber a hora de desfazer uma aliança constitui uma vantagem estratégica.

Apesar de dominada por humanos, os oponentes de IA costumam ignorar essa competência. Em jogos com muitos exércitos, no entanto, alianças militares são o que garantem a sobrevivência de jogadores até o estágio final do jogo.

3.2 Objetivos do sistema de dicas

Esta seção revisita brevemente o objetivo deste trabalho ao definir os objetivos do sistema de dicas. Se, para dominar um jogo, é necessário dominar as diversas competências nele

envolvidos, é natural pensar que o objetivo de um sistema de apoio ao jogador seja o de permitir que o jogador tenha condições de dominar tais competências. Um aspecto importante do sistema é que ele permita ao jogador conhecer e aprender o jogo de forma mais rápida que o uso de pura tentativa e erro. Caso contrário, tal sistema não seria útil.

Assim, é possível dizer que o objetivo sistema de apoio ao jogador proposto neste trabalho é o de permitir que os usuários aprendam a jogar melhor, ou possam tomar decisões melhores durante a execução do jogo sem a necessidade de aprender apenas por tentativa e erro. A ideia básica que suporta essa abordagem é a de que a exibição de dicas em pleno jogo permite ao jogador aprender melhor através dos conselhos exibidos pelo sistema. Na forma de uma analogia, o objetivo do sistema é o de desempenhar um papel semelhante ao de um amigo melhor conhecedor do jogo que dá dicas ao jogador de menor experiência para que o desempenho deste possa ser melhorado.

3.3 Características das dicas e do sistema

Há diversas questões envolvidas na definição e criação das dicas a serem usadas pelo sistema. Algumas delas dizem respeito à forma de obtenção do conhecimento usado para construção das dicas. Outras, à sua natureza, nível de detalhes e granularidade. Deve-se considerar, também, a forma e frequência de apresentação das dicas ao usuário e, por fim, questões quanto ao modelo e à codificação das dicas propriamente dita das dicas. Já sobre o sistema, é preciso abordar seu modelo geral.

3.3.1 Obtenção do conhecimento

Há, basicamente, duas formas de obter dados para criação das dicas a serem usadas pelo sistema: eles podem ser obtidos através do conhecimento de especialistas, como no caso do estudo de guias de estratégia projetados para o jogo (ou entrevista de um especialista no jogo) ou através da análise de partidas entre jogadores experientes. As abordagens supracitadas possuem vantagens e desvantagens:

Obtenção através de especialistas: A vantagem dessa forma de obtenção de conhecimento é a de que é possível, de forma direta, conhecer o que é importante, ou não, para dominar o jogo. Sua desvantagem é a necessidade de codificação manual das regras ditadas pelo especialista.

Análise de partidas: Possui a vantagem de ser possível extrair comportamentos gerais das partidas. No entanto, há a dificuldade de codificar as relações obtidas em instruções

inteligíveis para seres humanos. Além disso, há a necessidade de sintonia de parâmetros dos algoritmos de aprendizado, que é trabalhosa.

Existe, ainda, a possibilidade do sistema aprender através das ações de seus usuários à medida que novas partidas são jogadas. No entanto, neste trabalho optou-se por extrair as informações de especialistas através do estudo de guias de estratégia para geração das dicas.

3.3.2 Perfil do usuário e classes de dicas

Além de ser necessário definir as fontes de onde as dicas serão obtidas, também é necessário definir seu público-alvo: é razoável supor que as necessidades de um completo iniciante sejam diferentes das de um jogador de nível intermediário ou avançado. Logo, é pertinente que seja dado aos usuários o poder de selecionar seu nível de habilidade, ou que o sistema seja capaz de inferir o nível de habilidade do jogador. Neste trabalho não foram considerados diferentes níveis de habilidade e a codificação do sistema foi realizada baseada em um "usuário ideal" de nível iniciante-intermediário.

Relacionado ao nível de experiência associado a uma determinada dica, há a necessidade de definir, também, as competências abordadas pelo sistema. Idealmente, todas as competências devem ser contempladas. No entanto, algumas possuem prioridade maior sobre outras: a competência diplomática pode, por exemplo, ser encarada como não essencial, dado que jogos com um único jogador contra um oponente de inteligência artificial tendem a não possuir aspectos diplomáticos. De forma semelhante, apesar do domínio das diversas competências ser necessário para o especialista, o bom tratamento da extração de recursos é essencial, já que outras atividades do jogo dependem da existência de recursos armazenados. Dessa forma, um aspecto desejável em sistemas de dicas, mas não abordado neste trabalho, é a possibilidade de criação de classes de dicas, de modo que os usuários sejam capazes de configurar se determinada classe deve ser exibida, ou não, ou sua prioridade diante de outras.

Outro aspecto importante das dicas é sua granularidade, que está, de certa forma, relacionada ao nível de experiência do jogador. Por exemplo, uma dica do tipo "Desenvolva 'Paladinos'", ainda que equivalente a "Construa uma 'Igreja' usando um 'Plebeu' e, nela, desenvolva 'Paladinos' para lutar contra 'Cavaleiros da Morte'" possui um significado diferente para jogadores de níveis diferentes. Um jogador inexperiente pode ser incapaz de compreender como seguir a primeira dica, mas capaz de seguir a segunda, enquanto um jogador experiente pode considerar a segunda redundante. Assim como supõe-se um usuário ideal, também é suposto que o usuário preferirá dicas mais precisas neste trabalho.

3.3.3 Apresentação das dicas

Outros aspectos relevantes das dicas são relacionados à sua forma e frequência de apresentação. Idealmente, a forma de apresentação das dicas deve ser integrada à própria interface do jogo, de modo que o jogador não seja distraído por diferenças na interface. De forma semelhante, é importante que seja definida uma frequência agradável para apresentação das dicas, pois se o sistema intervir muito no jogo ele poderá ser considerado intrusivo e se o sistema fizer muito poucas intervenções poderá ser considerado inútil. Neste trabalhou, optou-se por apresentar as dicas através da interface do jogo, através de um sistema de registro de eventos e por síntese de voz. Detalhes sobre a apresentação serão descritos na seção 4.4.1.

3.3.4 Modelo e codificação

Definidas a forma de obtenção das dicas, as competências contempladas e seu nível de detalhe, é possível modelar as dicas como serão usadas pelo sistema. Considerando que o sistema de dicas as apresentará ao jogador de acordo com o estado do jogo, é necessário usar algum algoritmo capaz de executar o processo de tomada de decisão para escolha das dicas. Pelos motivos em favor das árvores de decisão descritos na seção 2.2.3, essa ferramenta foi a escolhida para codificação das dicas.

Como as árvores de decisão geram uma ação assim que os nós de decisão sejam satisfeitos, em princípio a prioridade das dicas dever ser decidida *a priori*, de acordo com a construção da árvore. Logo, uma suposição implícita dessa abordagem é a de que certos componentes estratégicos estão presentes em boas estratégias, ou que existe uma estratégia ótima para jogos RTS, o que pode não ser verdade.

Capítulo 4

Arcabouço experimental

O objetivo deste capítulo é descrever o arcabouço experimental desenvolvido neste trabalho. Inicialmente será apresentada uma visão geral do processo de desenvolvimento e, então, cada etapa será descrita.

4.1 Visão geral

Conforme comentado, apesar de haver diversos trabalhos que focam no desenvolvimento de agentes inteligentes para jogos RTS, há poucos trabalhos desenvolvidos com o objetivo de auxiliar o jogador. A ideia neste trabalho, portanto, é a de explorar o conhecimento de domínio depositado em guias de estratégia[1] desenvolvidos especificamente para jogos RTS e utilizá-lo para desenvolver um sistema especialista, doravante chamado sistema de dicas, capaz de auxiliar o jogador a aprender mais rapidamente o domínio de jogos RTS.

Estima-se que uma parte importante do aprendizado está em praticar a atividade que se quer aprender. Portanto, o uso de um sistema semelhante ao aqui apresentado pode permitir não só o aprendizado de estratégias básicas, mas o repasse de conhecimento de um jogador experiente a outro através de demonstração e prática.

Em alto nível, a abordagem utilizada consistiu em

1. Prospectar os motores de jogos RTS mais usados e selecionar o motor a ser usado para a implementação deste trabalho;

2. Especificar as dicas a serem usadas pelo sistema;

3. Instrumentar o código do motor para funcionamento com o sistema de dicas;

[1] A criação de guias de estratégia é prática comum em comunidades dos mais diversos jogos, um exemplo, para o caso do xadrez é a Encyclopaedia of Chess Openings [Abramov et al., 2010].

4. Implementar um sistema de coleta de informações sobre o estado do jogo;
5. Construir e implementar a árvore de decisão;
6. Realizar experimentos para avaliar o sistema.

Neste capítulo, cada passo da abordagem será descrito. No entanto, a apresentação dos experimentos e seus resultados será postergada até o capítulo 5.

4.2 Prospecção e escolha do motor

Durante a prospecção de possíveis motores de jogos, ficou claro que aqueles que recebiam maior atenção acadêmica eram o ORTS e Stratagus[2]. Portanto, esses foram os escolhidos para avaliação.

No desenvolvimento deste trabalho havia um interesse particular em métodos de análise tática. Sabendo que mapas de influência, descritos na seção 2.2.2, são uma técnica comumente usada para implementar análise tática, foi decidido que a implementação de mapas de influência seria usada como método de avaliação da facilidade de programação dos motores.

Durante a implementação, foi constatado que, apesar do motor ORTS possuir módulos de análise tática, usá-los não era simples devido a ausência de documentação e exemplos simples de como as classes do motor poderiam ser usadas. Já no motor Stratagus, a implementação dos mapas de influência foi completada sem problemas e, portanto, esse foi o motor escolhido para o desenvolvimento deste trabalho. Por ser possível utilizar mapas de influência para realizar consultas sobre o mundo, sua implementação é descrita na seção 4.5.

4.2.1 Considerações sobre os motores

ORTS e Stratagus apresentam soluções diferentes para o mesmo problema. O objetivo desta subseção é o de comentar algumas dessas diferenças e as experiências obtidas neste trabalho quando da avaliação desses motores.

Um dado relevante e que comumente não é citado quando características de sistemas são comparadas é a legibilidade e facilidade de edição do código-fonte. Apesar de ambos os motores supracitados possuírem seu código facilmente acessível, o Stratagus foi desenvolvido por mais tempo que o ORTS, o que permitiu a estabilização das APIs e geração de documentação do código. O efeito dessa diferença é que o código do ORTS possui menos documentação, e o usuário costuma ser obrigado a ler o código para compreender o comportamento de funções e métodos.

[2] Comumente combinado ao jogo Wargus.

Apesar de, como o Stratagus, o ORTS disponibilizar suas estruturas de dados a uma linguagem de *script*, a linguagem usada no ORTS foi desenvolvida especificamente para esse motor. Tendo, como efeito, a falta de documentação e de uma grande comunidade de usuários à qual recorrer em caso de dúvida. A abordagem do Stratagus, nesse sentido foi mais acertada, pois utiliza Lua, um padrão de fato na indústria de jogos [Millington, 2006], que não compartilha os problemas da linguagem de *script* do ORTS.

Quanto à IA padrão, no Stratagus o comportamento básico das unidades já vem implementado, mas o acesso aos métodos de mais baixo nível do código é negado aos usuários, que ficam sem capacidade de controlar todos os aspectos da IA, a menos que decidam por mudar o código do motor. A abordagem do ORTS é igualmente insatisfatória por prover apenas os controles de mais baixo nível, o que significa que o usuário deve programar o comportamento mais básico de suas unidades antes de focar na solução de seu problema. O ideal seria que os motores fornecessem uma solução de meio-termo, que possuísse comportamentos predefinidos e que pudessem ser, opcionalmente, re-escritos.

Por já possuir jogos prontos e *scripts* que implementem a IA de *bots* semelhantes aos encontrados em RTS tradicionais, o Stratagus possui uma vantagem sobre o ORTS que, até a data de última avaliação, possuía apenas jogos que implementavam as modalidades de sua competição de IA, que são: planejamento de caminhos colaborativo, combate estratégico, combate tático e um jogo RTS completo, com um pequeno grafo tecnológico. Para este trabalho, a experimentação com o Stratagus se tornou mais viável e foi esse o motor escolhido.

4.3 Especificação das dicas

Para implementação do sistema de dicas foi estudado o guia de estratégia de Warcraft II [Blizzard, 2010]. O motivo de estudar especificamente esse guia é, como ficará claro nas próximas seções, que o jogo Wargus, implementado no motor Stratagus, é um clone do jogo Warcraft II e, portanto, regras e estratégias aplicáveis ao Warcraft II também são aplicáveis ao Wargus.

Alguns itens em particular foram usados como fonte de inspiração para o da base de conhecimento deste trabalho, como:

Trabalhadores são a chave para o desenvolvimento As unidades de construção são responsáveis por toda a cadeia produtiva e, portanto, são as unidades mais importantes do jogo e a chave para a vitória.

Quanto mais trabalhadores, melhor A menos que o ouro seja limitado, quanto mais trabalhadores um jogador tiver, melhor, pois eles gerarão uma taxa de entrada de recursos grande o suficiente para manter o exército.

Treine trabalhadores continuamente Especialmente no início do jogo, é importante que não haja espera no desenvolvimento da base, pois qualquer atraso pode ser fatal em estágios mais avançados do jogo.

Mantenha sua quantidade de ouro e madeira balanceada É importante possuir o suficiente de ambos os recursos. Possuir apenas um recurso impede o desenvolvimento da cadeia produtiva do jogo e deve ser evitado.

Reconhecimento do ambiente é a chave para a vitória Para saber como reagir (e agir) contra o inimigo, é necessário saber o que ele faz. O que implica em reconhecimento do ambiente, uma das atividades mais importantes do jogo.

Ordens de construção Assim como o xadrez possui catálogos com táticas de aberturas de jogos, há catálogos documentando ações em jogos RTS para construção e treinamento de um conjunto de unidades no menor tempo possível. No início do jogo é importante ser capaz de construir soldados o mais rápido possível e, portanto, aprender ordens de construção é essencial para garantia da vitória.

4.4 Instrumentação do código do motor

Conforme discutido, foi necessário instrumentar o código do motor para permitir a atualização do estado e processamento do sistema e dicas e a geração e atualização dos mapas de influência. Após estudo do código, sua instrumentação foi simples, bastando adicionar as chamadas de função relevantes ao laço principal do jogo[3].

A instrumentação para implementação dos métodos de observação de dados foi simples e sem grandes complicações. A figura 4.1 exibe a estrutura, em alto nível, do laço principal do programa. Nela, as modificações adicionadas ao laço principal estão envolvidas por uma caixa e devidamente identificada. As outras são operações já existentes e ainda presentes no motor. "Outras ações" representa todas as atividades de manutenção de estatísticas e estruturas de dados realizadas pelo motor no laço principal.

Quanto à atualização do estado do sistema de dicas, ela é uma operação que lê os dados disponíveis no motor, os filtra para que apenas dados locais estejam acessíveis ao sistema e atualiza suas estruturas de dados de acordo. Já a etapa de execução lê as estruturas de dados recém-atualizadas e executa as árvores de decisão.

Outras adições ao motor foram necessárias, como a implementação do sistema de notificação, responsável por disponibilizar as dicas ao usuário, e ligeiras alterações em pontos

[3] Localizado no arquivo `src/stratagus/gameloop.cpp` relativo à raiz do código do Stratagus, na função `GameMainLoop()`.

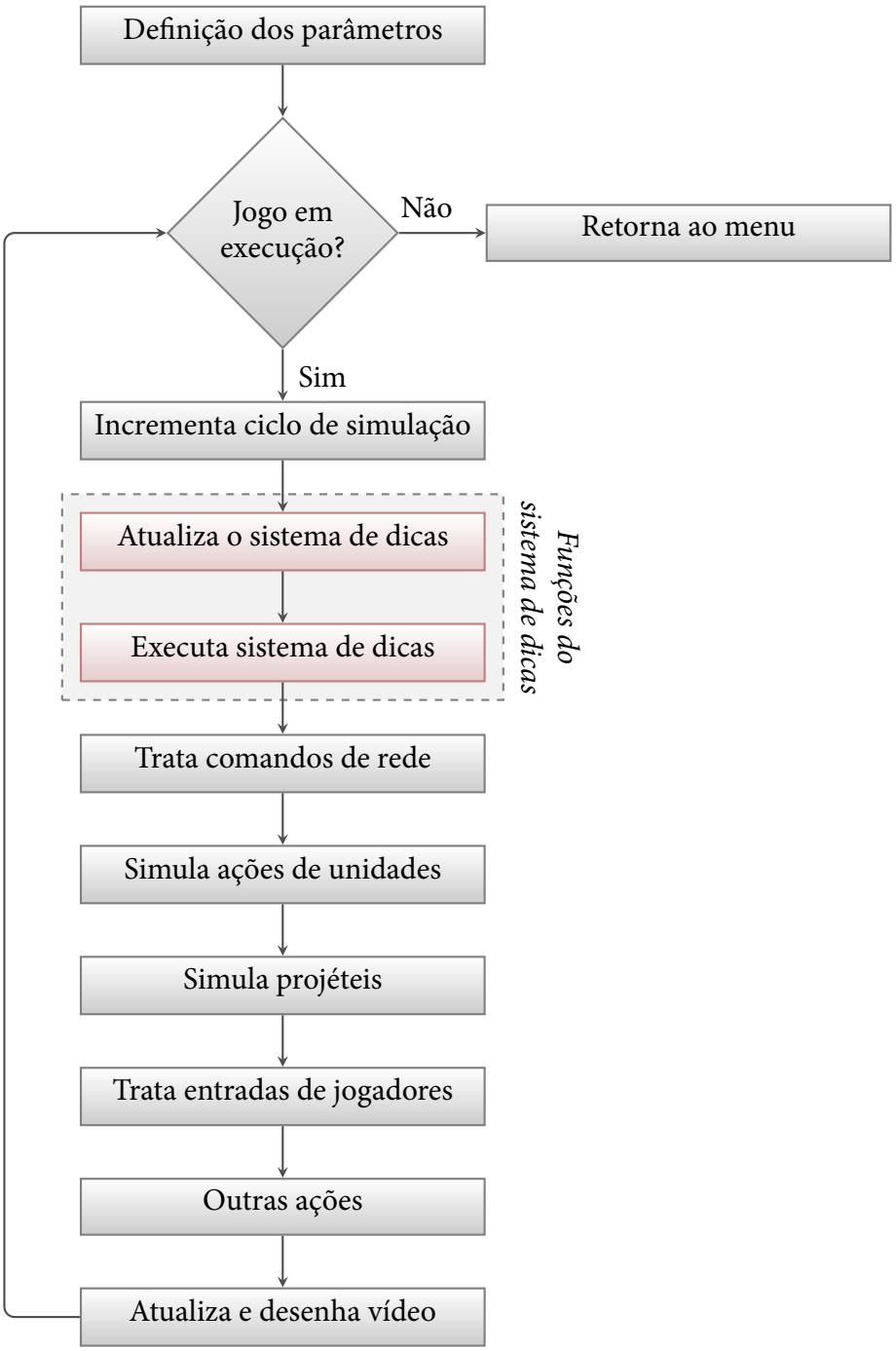


Figura 4.1. Abstração do laço principal do Stratagus. Nesta representação do laço principal do Stratagus, as operações destacadas pela caixa pontilhada, são as que foram implementadas para este trabalho e as demais são nativas do Stratagus.

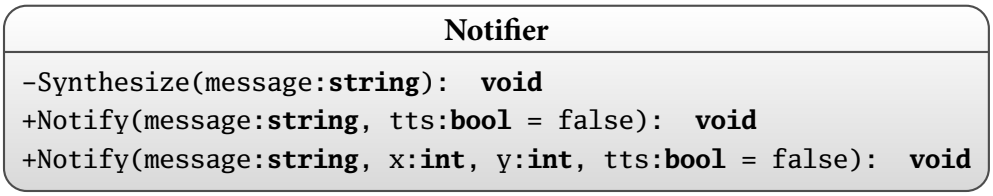


Figura 4.2. Diagrama exibindo uma versão simplificada da interface de programação da classe de notificação.

específicos do motor para criação de um subsistema semelhante a um de *callbacks*. A implementação desse último sistema foi necessária para que fosse possível informar ao sistema de dicas quando unidades eram treinadas e construções edificadas, informação necessária para a implementação de dicas sobre ordem de construção inicial.

4.4.1 Sistema de notificação

Como critério de projeto, foi definido que as mensagens aos usuários seriam disponibilizadas de três formas: através da própria interface de exibição de mensagens do jogo, exibida na figura 2.1e (página 7), através do subsistema de registro de eventos (*log*), implementado para armazenamento das mensagens geradas, e por síntese de voz.

Sendo o Stratagus um motor desenvolvido para executar em uma única *thread*, a síntese das mensagens, quando necessária, foi projetada para ser realizada em um processo separado, executado pelo sistema de texto para fala (*Text to Speech*, ou TTS) *Festival* [Black & Taylor, 1997].

Uma versão simplificada da interface de programação da classe de notificação é exibida na figura 4.2. Nela, o método `Synthesize` é responsável por iniciar um novo processo e solicitar ao *Festival* a síntese de texto para voz. As duas variantes do método `Notify` existem devido ao fato de haver mensagens que são relacionadas a alguma posição no mapa e mensagens que não são. Para as que possuem significado no mapa, a variante com as coordenadas do ponto no mapa é usada, para as que só notificam dados, a variante sem coordenadas é chamada. Em ambas, caso o argumento `tts` seja verdadeiro, o método `Synthesize` é chamado internamente.

4.5 Implementação do sistema de coleta de informações

O algoritmo básico da etapa de coleta de informações do sistema é descrito no algoritmo 4.1. Ele faz uso intenso da notação de teoria dos conjuntos para simplificação da apresentação. As operações de conjuntos descritas são diretamente implementadas em linguagens com suporte a *list comprehensions*. A chamada a `ResetLocalState` é descrita no algoritmo 4.2 e a `IsLatticeBuilding` é descrita em 4.3. Por fim, a notação com colchetes da linha 13 é baseada

na notação de Iverson, que atribui valor um à expressão entre colchetes caso ela seja verdadeira e valor zero em caso contrário, exibida na equação (4.1).

$$[P] = \begin{cases} 1 & \text{se } P \text{ verdadeiro,} \\ 0 & \text{caso contrário} \end{cases} \tag{4.1}$$

Entrada: Estruturas de dados de jogadores e unidades do Stratagus
Saída: Estado atual do sistema de dicas atualizado

1: ResetLocalState() ◁ limpa o estado de variáveis usadas para coleta de dados
2: Workers ← $\{w \mid w \in \text{PlayerUnits} \wedge \text{IsWorker}(w)\}$
3: Visibleenemies ← $\{e \mid e \in \text{Units} \wedge \text{IsVisible}(e) \wedge \text{IsEnemy}(e)\}$
4: Newenemies ← $(\text{Visibleenemies} \setminus (\text{Knownenemies} \cap \text{Visibleenemies}))$
5: Knownenemies ← $\text{Knownenemies} \cup \text{Newenemies}$
6: Goldmines ← $\{g \mid g \in \text{Units} \wedge \text{IsVisible}(g) \wedge \text{IsGoldMine}(e)\}$
7: Idleworkers ← $\{w \mid w \in \text{Workers} \wedge \text{IsIdle}(w) \wedge \text{IsRemoved}(w)\}$
8: Totalworkers ← $|\text{Workers}|$
9: Newgoldmines ← $\text{Goldmines} \setminus \text{Oldgoldmines}$
10: Numtowers ← $|\{t \mid t \in \text{PlayerUnits} \wedge \text{IsTower}(t)\}|$
11: Goldpotential ← $\text{sum}(\{\text{GoldLeft}(g) \mid g \in \text{Goldmines}\})$
12: Buildings ← $\{b \mid b \in \text{PlayerUnits} \wedge \text{IsLatticeBuilding}(b)\}$
13: Castlestage ← $[(\text{“Castle”} \in \text{PlayerUnits}) \vee (\text{“Fortress”} \in \text{PlayerUnits})]$

Algoritmo 4.1: Etapa de atualização do sistema de dicas. Neste algoritmo, supõe-se a existência de uma função `sum`, capaz de somar os valores dos elementos de um conjunto. Também são usadas a notação de Iverson, descrita na equação (4.1), e operações de seleção de elementos de conjuntos. `ResetLocalState()` é descrito no algoritmo 4.2.

4.5.1 Mapas de influência

Como supracitado, a primeira tarefa para experimentação no Stratagus foi a implementação dos mapas de influência. Sendo úteis para realizar consultas sobre o terreno e sua influência militar, eles poderiam ser utilizados pela etapa de processamento do sistema de dicas.

Uma possível aplicação dos mapas de influência seria a de alertar o usuário caso unidades sem poder de ataque entrassem em território inimigo, como exibido na figura 4.3. Na figura, um trabalhador está cercado por torres inimigas e poderia ser destruído, caso seu inimigo julgasse necessário. Ao detectar influência inimiga, o sistema poderia notificar o usuário, que poderia agir de acordo. No entanto, apesar de sua utilidade, eles acabaram não sendo usados no sistema de dicas. Uma das razões para essa decisão foi o uso indesejável de informação

Entrada: Estruturas de dados do sistema de dicas
Saída: Estruturas de dados do sistema de dicas atualizadas
1: Buildings ← {∅}
2: Idleworkers ← {∅}
3: Newgoldmines ← {∅}
4: Oldgoldmines ← {∅}
5: Shouldchop ← false
6: Shouldmine ← false
7: Shouldscout ← false
8: Toomanytowers ← false
9: Totalworkers ← 0
10: Goldpotential ← 0
11: Visibleenemies ← {∅}
12: Allies ← Listofallies()
13: Enemies ← Listofenemies()
14: Oldgoldmines ← Goldmines
15: Oldenemies ← Enemies

Algoritmo 4.2: `ResetLocalState()`, função de limpeza das estruturas de dados do sistema de dicas para nova atualização.

Entrada: Uma unidade, u, do jogo Wargus
Saída: Valor booleano que indica se a unidade pertence à abstração de estados
1: **return** $[u \in \text{StateLattice}]$

Algoritmo 4.3: `IsLatticeBuilding(`u`)`, algoritmo que verifica se a unidade passada pertence à abstração de estados descrita na seção 4.6.1. O conjunto StateLattice contém todos os elementos exibidos na figura 4.4.

global para geração dos mapas e pouco conhecimento sobre abordagens com tratamento de incerteza.

4.6 Construção e implementação da árvore de decisão

4.6.1 Abstração de estados

Como citado no capítulo 1, o espaço de estados de um jogo RTS é grande e soluções puramente baseadas em busca se tornam intratáveis. Assim, métodos para abstração do espaço de estados são necessários. Um dos possíveis métodos de abstração foi apresentado em Ponsen et al. [2006] e usado neste trabalho para inferência sobre o estado atual dos exércitos do jogo.

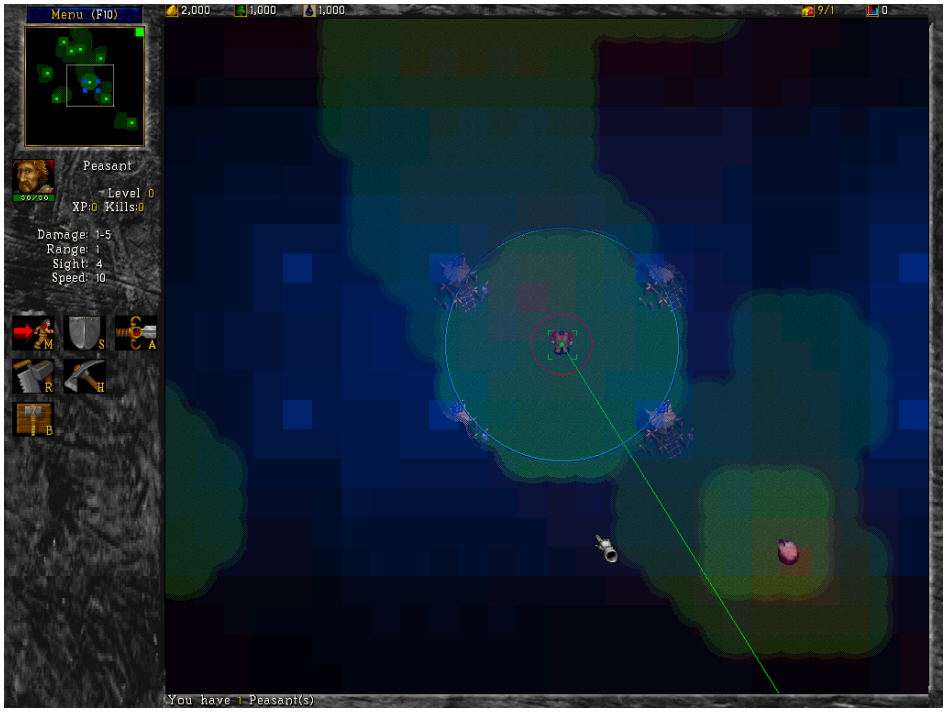


Figura 4.3. Trabalhador cercado por torres inimigas. Essa situação ocorreu por que o planejador de caminhos otimiza apenas a distância percorrida, sem se preocupar com as forças inimigas no caminho da unidade. É importante notar que, nessa implementação, o mapa de influência é uma informação global, o que poderia ser considerado trapaça.

A técnica consiste em supor que o nível de desenvolvimento do exército de um jogador é função das edificações por ele construídas. A cada combinação de tipos de edificações construídas é atribuído um estado. Um efeito colateral dessa atribuição é que sempre que uma edificação de um novo tipo é construída, há uma mudança de estado. A figura 4.4 exibe a abstração gerada para o Wargus. Como exemplo, suponha que um jogador da raça Orc esteja no estado de número 1 do grafo. Nesse caso, o jogador possui as unidades "Great Hall" e "Barracks" em qualquer quantidade. Ao construir, por exemplo, um "Troll Lumber Mill", o jogador sai do estado 1 e vai para o estado 2.

Neste trabalho, essa abstração foi usada para identificar o estado de desenvolvimento do exército do jogador. Conforme exibido na figura, caso o jogador esteja no estado 1 (ou em estados anteriores, não representados), supõe-se que seu exército encontra-se em estado inicial. Nos estados de 2 a 8 o exército está em estágio de formação e pronto para combate inicial. Nos estados de 9 a 13, novas unidades se tornam disponíveis e o exército se encontra em um estágio médio, já com a capacidade de criação de unidades de combate avançadas, como cavaleiros e ogros. Já no estágio avançado, representados pelos nós 14 a 20, o jogador tem acesso, gradativamente, a todas as unidades do jogo, sendo capaz de construir um exército com unidades avançadas e combinações complexas de unidades.

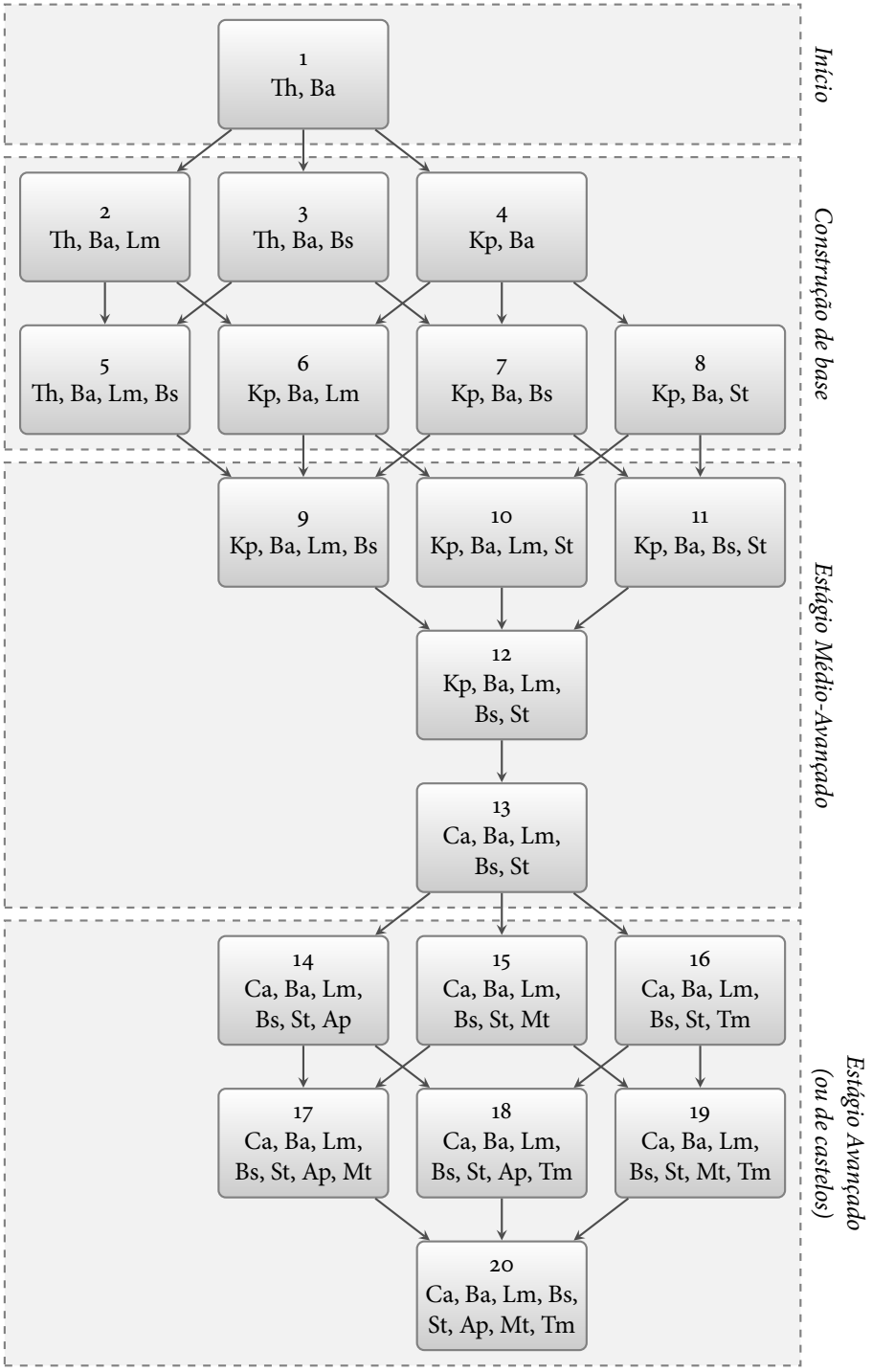


Figura 4.4. Grafo de abstração de estados para o Wargus. Nesta imagem, as transições são ativadas pela edificação de novas construções. Cada estado, por sua vez, é definido pelos tipos de edificações construídas. As convenções usadas para representar as construções são exibidas na tabela 4.1.

Tabela 4.1. Convenções de nomes usados nas figuras 4.4 e 4.5. Na primeira coluna é exibida a abreviação usada nessa figura. Nas segunda e terceira colunas são exibidos os nome de construção equivalentes a cada raça do jogo.

	Raça	
Convenção	Humana	Orc
Ap	Gryphon Aviary	Dragon Roost
Ar	Archer	Troll Axethrower
Ba	Barracks	Barracks
Bl	Ballista	Catapult
Bp	Battleship	Ogre Juggernaught
Bs	Blacksmith	Blacksmith
Ca	Castle	Fortress
Dr	Gryphon	Dragon
Fm	Gnomish Flying Machine	Goblin Zeppelin
Fo	Foundry	Foundry
Gi	Gnomish Inventor	Goblin Alchemist
Gs	Gnomish Submarine	Giant Turtle
Gt	Guard Tower	Guard Tower
Kn	Knight	Ogre
Kp	Keep	Stronghold
Lm	Elven Lumber Mill	Troll Lumber Mill
Mt	Mage Tower	Temple of the Damned
Om	Paladin	Ogre Mage
Ra	Ranger	Berserker
Re	Refinery	Refinery
St	Stables	Ogre Mound
Sy	Shipyard	Shipyard
Th	Town Hall	Great Hall
Tm	Church	Altar of Storms
Tr	Transport	Transport

4.6.2 Grafo tecnológico

Para inferência do estado atual do inimigo, os grafos tecnológicos de ambas as raças do jogo foram adicionados ao sistema de dicas. Um modelo dos grafos é exibido na figura 4.5. Como as raças do jogo são balanceadas, a estrutura do grafo é a mesma para ambas e, portanto, optou-se por abstrair a representação através do uso de um modelo. A tabela 4.1 exibe os nomes equivalentes das unidades para cada raça.

No processamento da árvore, quando o número de inimigos conhecidos aumenta, os novos inimigos são adicionados ao conjunto `Newenemies`, como descrito no algoritmo 4.1, e, então, é aplicado o algoritmo de ordenação topológica [Cormen et al., 2002, Seção 22.4] ao

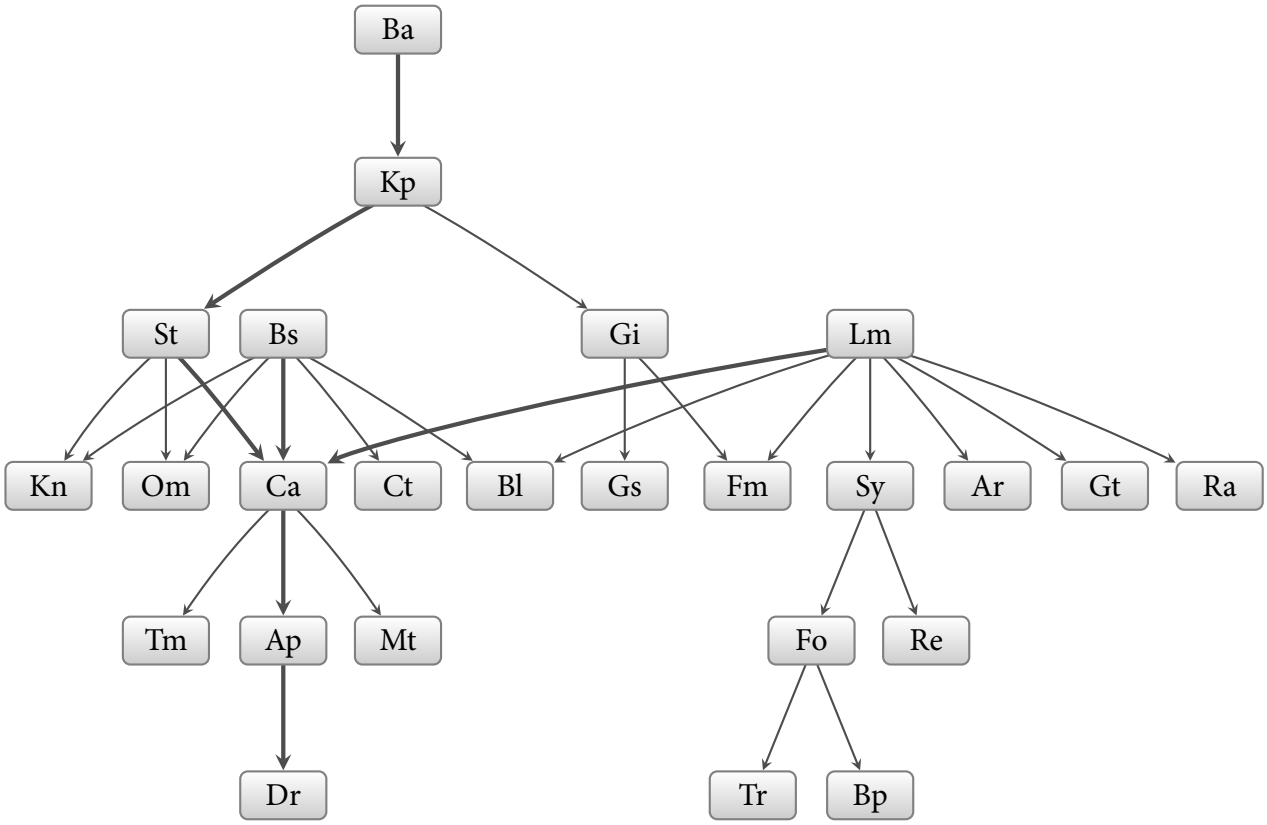


Figura 4.5. Grafo tecnológico completo do jogo Wargus. A convenção de nomes usados nos nós do grafo é exibida na tabela 4.1. As arestas mais grossas exibem os nós de preferência da estratégia *Air Attack* de Wargus, descrita na seção 5.1.1.

subgrafo formado pelo nó correspondente ao do novo inimigo detectado e seus pais no grafo tecnológico. Neste trabalho foi feita a suposição de que, se uma unidade de determinado tipo é encontrada, o exército ao qual ela pertence possui acesso a todos os seus pré-requisitos. Assim, sempre que uma nova unidade é encontrada, o sistema de dicas é capaz de avisar o usuário sobre quaisquer novas dependências, permitindo que o jogador tenha maior conhecimento sobre o estado do exército de seu oponente. Uma visão geral é exibida no algoritmo 4.4.

4.6.3 Árvores de decisão

Baseado no conhecimento obtido nos guias de estratégia, foram definidas três categorias a serem tratadas pelo sistema de dicas: gerência de recursos, estratégia básica e contra-estratégia. Das recomendações encontradas no guia de estratégia de Warcraft II, a maioria das dicas listadas pode ser classificada como pertencente a um dos conjuntos definidos por cada uma dessas três categorias e, portanto, essa forma de divisão pareceu adequada ao problema tratado. Do universo de dicas presentes no guia, foram selecionadas aquelas que pareciam adicionar mais valor ao jogo, estando presentes durante todo o jogo. As próximas seções descreverão

Entrada: Newenemies, conjunto de novas unidades inimigas
Saída: Notificação sobre as novas unidades e suas dependências

```
for unit in Newenemies do
    Notify("Nova unidade inimiga encontrada", unit)
    Deps ← TopologicalSort(unit, TG)
    for dep in Deps do
        if [dep ∉ Knownenemies] then
            Notify("Nova dependência encontrada", dep)
            Knownenemies ← Knownenemies ∪ {dep}
        end if
    end for
    Enemycastlestage ← [("Castle" ∈ Knownenemies) ∨ ("Fortress" ∈ Knownenemies)]
end for
```

Algoritmo 4.4: `TechnologyTree()`. Algoritmo que percorre o conjunto de novas unidades inimigas em busca de suas dependências de criação no grafo de dependências tecnológicas. No algoritmo, a variável `TG` representa o grafo tecnológico da raça da unidade inimiga encontrada e `TopologicalSort(unit, TG)` implementa a ordenação topológica no subgrafo de `TG` formado por `unit` e seus pais.

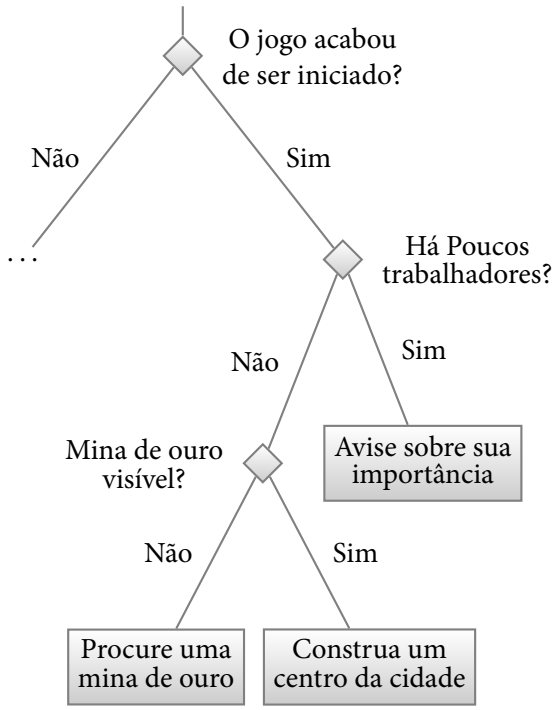


Figura 4.6. Subárvore da árvore de decisão modelada para este trabalho. A subárvore exibe as verificações realizadas no início de uma nova partida.

as dicas implementadas. Como exemplo, a figura 4.6 exibe um trecho da árvore de decisão projetada. Na subárvore da figura são exibidas as verificações realizadas quando do início de uma nova partida em que, se houver poucos trabalhadores, o sistema notifica o usuário de sua importância e, caso não haja mina de ouro disponível, o sistema sugerirá ao jogador que encontre uma antes de construir um centro da cidade.

4.6.3.1 Contra-estratégia

Combinações de unidades avançadas À medida que o estado do jogo avança, unidades mais poderosas se tornam disponíveis. No entanto, essas unidades possuem uma gama de ataques maior, exigindo maior controle do jogador. Há, ainda, unidades que se tornam mais efetivas quando usadas conjuntamente com outras. A ideia dessa dica é, supridos os requisitos para criação de unidades avançadas, sugerir as combinações de unidades e habilidades que são mais efetivas para o jogador.

Ataque a unidades avançadas Devido ao grande poder de ataque das unidades avançadas, não só é necessário possuir poder de fogo para combatê-las, como é importante usar táticas compatíveis. O objetivo dessa dica é, ao ser detectado um combate, verificar se unidades avançadas, como "Ogre Mages" ou "Paladins" estão envolvidas no combate para sugerir ações apropriadas.

4.6.3.2 Estratégia básica

Trabalhadores contínuos Essa dica interage com a próxima para lembrar ao jogador que trabalhadores são a unidade mais importante do jogo e que, para garantir a produção do exército, novos trabalhadores devem estar sempre treinados.

Quantidade de trabalhadores Essa dica observa a quantidade de trabalhadores treinados no exército e determina se esse número está compatível com o esperado para aquele estágio do jogo, se não estiver, o usuário é notificado para treinar mais trabalhadores. Outros fatores precisam ser considerados, como quantidade de ouro disponível em minas visíveis.

Construção de unidades extra Construções de treinamento, como "Barracks", "Dragon Roost" e "Gryphon Aviary" podem ser incapazes de suprir a necessidade do exército em momentos próximos a combates. Essa dica observa o estado do jogo e sugere a edificação de novas construções se necessário.

Reconhecimento de ambiente Essa dica verifica o ambiente como conhecido pelo jogador e faz sugestões de exploração no caso de no início do jogo não haver minas de ouro

próximas e sugere o uso de unidades próprias para reconhecimento, como "Goblin Zeppelins" e "Flying Machines".

Grafo tecnológico Sempre que uma nova unidade inimiga é percebida, essa dica percorre o grafo tecnológico da raça do inimigo[4], localizando as dependências daquela unidade e atualizando a crença do sistema de dicas sobre o estado inimigo de acordo. Caso alguma nova dependência seja encontrada, o jogador é notificado.

Encontrar mina de ouro Assim que o sistema de dicas percebe que o jogador não explorou o mapa em busca de minas de ouro, ou que o potencial das minas de ouro conhecidas pelo jogador é muito baixo, a ideia dessa dica é informar ao jogador da possível escassez de ouro e sugerir providências como a exploração do mapa.

Tratamento de torres Jogadores iniciantes (e até implementações de IA para Warcraft) tendem a construir muitas torres de guarda para defesa de suas bases. No entanto, as torres são construções caras e vulneráveis, sendo efetivas somente como forma de conter alguns ataques mais simples. O objetivo dessa dica é, caso seja detectada essa tentativa de defesa, lembrar ao jogador que é melhor construir unidades ofensivas que torres.

Ordem de construção inicial Essa dica implementa a ideia básica descrita na seção 2.2.4.1, Planejamento econômico: ela percorre os nós iniciais do grafo tecnológico para sugerir ao jogador uma forma de deixá-lo com uma base minimamente capaz de treinar soldados e se defender. Ela se baseia em sugestões de ordens de construção documentadas em guias de estratégia.

Sugestões para estágio de castelos Como a dica anterior, o objetivo dessa dica é, ao detectar que o jogador pode construir castelos, sugerir ao jogador ordens de construção interessantes para acessar as unidades mais poderosas do jogo.

4.6.3.3 Recursos

Trabalhadores ociosos Como implementado em alguns jogos RTS, essa dica detecta trabalhadores ociosos e informa ao jogador quais são suas posições.

Trabalhadores e sua importância Essa dica tem por objetivo lembrar ao jogador que trabalhadores não devem ser desperdiçados, pois são a unidade mais importante do jogo.

Valor ótimo de extratores É comum em jogos RTS que fontes de recursos suportem um número máximo de trabalhadores explorando-as e para qualquer valor superior, há

[4] Ainda que ela seja desconhecida, ao avistar a nova unidade, ela passa a ser conhecida.

perda de produtividade. O objetivo dessa dica é, ao perceber que fontes de recursos estão saturadas, notificar ao usuário.

Nível de recursos baixo demais Ao perceber que o jogador possui poucos recursos, o sistema de dicas observa onde os trabalhadores do jogador estão alocados e se há um número de trabalhadores suficiente para manter o exército. Se não houver, o sistema indica ao jogador como é possível melhorar sua economia.

Recursos desequilibrados O sistema de dicas observa constantemente a quantidade de recursos acumulados e a taxa de entrada de recursos dos jogadores e, caso julgue necessário, o sistema indica ao jogador que ele deveria equilibrar sua economia.

Muitos recursos acumulados A menos que o jogador tenha explorado toda a árvore tecnológica de seu exército, há sempre algo em que investir recursos. Sendo o jogo de estratégia militar e não de acúmulo de recursos, se o jogador acumular muitos recursos sistema de dicas tenta sugerir onde gastá-los.

É importante notar que o valor absoluto de "muitos" e "poucos" recursos varia ao longo da partida. Quanto mais avançado se torna o exército de um jogador, maior o valor absoluto de "muito" e "pouco".

Capítulo 5

Experimentos e resultados

Este capítulo apresenta os resultados experimentais deste trabalho. Os experimentos foram divididos em dois grupos: testes de desempenho e testes com usuários. As próximas seções os descrevem e apresentam seus resultados.

5.1 Testes com usuários

Testes com usuários foram realizados para avaliar o desempenho do sistema de dicas quanto à qualidade e utilidade das dicas geradas. O objetivo dos testes era o de avaliar se usuários de jogos RTS considerariam o sistema útil e se as dicas geradas eram de qualidade. Mais especificamente, o ponto crítico da avaliação com usuários era conhecer, qualitativamente, a opinião dos usuários quanto à utilidade das dicas. Para que o teste fosse mais objetivo, foram selecionados somente usuários que já haviam jogado esse tipo de jogo pelo menos uma vez, para evitar que dificuldades de compreensão da interface interferissem no desempenho dos jogos.

O teste consistiu em submeter os usuários a duas partidas do jogo Wargus. Em ambas, a tarefa a ser executada por eles era a de tentar ganhar o jogo. No entanto, ficou claro para todos os testadores que o resultado das partidas (vitória ou derrota) não era crucial para o teste e que esse objetivo foi colocado apenas facilitar a comparação das partidas. Em uma das partidas, o sistema de dicas foi desativado. Na outra, ele foi ativado. No total, seis usuários participaram dos testes.

5.1.1 Estrutura dos testes

Os testes do sistema foram realizados em laboratório adaptado para testes com usuário e todos os comandos enviados ao jogo pelo usuário foram gravados em um formato de *log*

```
1  ReplayLog( {
2    Map = "Mysterious Dragon Isle",
3    MapPath = "maps/default.smp",
4    MapId = 1,
5    Type = 1,
6    Race = 0,
7  -- ...
8  Log( { GameCycle = 29136, UnitNumber = 33, UnitIdent = "unit-peasant",
9    Action = "stop", SyncRandSeed = 200750291 } )
10 Log( { GameCycle = 29257, UnitNumber = 33, UnitIdent = "unit-peasant",
11   Action = "build", PosX = 0, PosY = 38,
12   Value = [[unit-human-watch-tower]], SyncRandSeed = 1960903967 } )
13 Log( { GameCycle = 29406, UnitNumber = 73, UnitIdent = "unit-archer",
14   Action = "attack", PosX = 3, PosY = 35,
15   DestUnitNumber = 90, SyncRandSeed = -200263507 } )
16 -- ...
17 })
```

Figura 5.1. Parte da descrição de um *replay* de Stratagus. As reticências indicam código que foi omitido. Nas linhas de 1 a 6 é exibida parte da identificação do *log*. As linhas de 8 a 15 exibem comandos enviados a unidades. "GameCycle" representa o passo da simulação em que a ação foi enviada, "UnitNumber" representa o número da unidade que recebeu o comando, "UnitIdent" representa seu identificador, "Action" descreve a ação desempenhada pela unidade, "SyncRandSeed" é o *hash* calculado para aquele quadro, "PosX" e "PosY" representam as posições no mapa onde a ação deve ser desempenhada e "DestUnitNumber" a unidade a ser afetada pela ação.

próprio para *replay* em caso de necessidade. A figura 5.1 exibe partes de um *replay* do Stratagus. A estrutura usada para o teste é descrita a seguir.

1. O participante foi recepcionado na sala de testes;
2. Como nem todos os usuários conheciam o universo de Warcraft, as regras do jogo e sua mecânica básica foram apresentadas;
3. A estrutura dos testes foi descrita para o participante;
4. O sistema de dicas foi, então apresentado e foi deixado claro aos usuários que o que estava sendo avaliado era o sistema de dicas e a qualidade das dicas, não o desempenho dos jogadores;
5. O participante, então, preencheu o questionário pré-teste, descrito na seção 5.1.2;
6. O participante jogou, nesta ordem, uma partida sem o sistema de dicas e, depois, uma partida com o sistema de dicas ativado;

Tabela 5.1. Mapeamento dos nomes genéricos da estratégia *Air Attack* para nomes de unidades. Na primeira coluna é exibido o nome usado na estratégia exibida na figura 5.2. Nas segunda e terceira colunas são exibidos os nomes de unidades equivalentes a cada raça do jogo.

	Nome da unidade de acordo com sua raça	
Nome genérico	Raça Humana	Raça Orc
AiCityCenter	Town Hall	Great Hall
AiWorker	Peasant	Peon
AiLumberMill	Elven Lumber Mill	Troll Lumber Mill
AiBarracks	Barracks	Barracks
AiBlacksmith	Blacksmith	Blacksmith
AiBetterCityCenter	Keep	Stronghold
AiStables	Stables	Ogre Mound
AiBestCityCenter	Castle	Fortress
AiAirport	Gryphon Aviary	Dragon Roost
AiFlyer	Gryphon Rider	Dragon

7. Numa conversa informal o participante descreveu suas impressões sobre o sistema de dicas e, para aqueles que perguntaram, foi descrito o funcionamento do sistema;

8. O participante preencheu o questionário pós-teste, descrito na seção 5.1.2.1 e o teste foi encerrado com um agradecimento pela participação do usuário.

Em todos os testes os usuários jogaram contra uma das estratégias padrão de Wargus chamada *Air Attack*, estratégia que privilegia ataques aéreos. A figura 5.2 exibe trechos da implementação dessa estratégia. Por ser uma estratégia aplicável a ambas as raças do jogo, ela usa nomes genéricos de unidade. A tabela 5.1 relaciona os nomes usados na figura 5.2 aos nomes de unidades específicos de cada raça do jogo. O objetivo da estratégia da figura 5.2 é o de alcançar o nó "Dr" da figura 4.5 o mais rápido possível e, então, realizar ataques com unidades aéreas. Considere o seguinte exemplo, baseado na raça humana: Na figura 5.2, a execução é iniciada com a necessidade de possuir um "Town Hall". Após ter sido completada, nove "Peasants" são criados para extração de ouro e madeira, um "Elven Lumber Mill" é construído e, então, um "Barracks" e um "Blacksmith" são construídos. Com isso, o "Town Hall" é atualizado para um "Keep", "Stables" é construído para habilitar a atualização do "Keep" para um "Castle". Com isso, todas as dependências para criação de um "Gryphon Aviary" estão completas e a IA é, finalmente, capaz de construir unidades de ataque aéreo. "Gryphon", nesse caso. A partir desse ponto, os ataques se iniciam. Nos testes com os usuários, isso acontecia em torno do ciclo 24000 de simulação, cerca de 12 minutos após o início do jogo.

```
1   function() return AiNeed(AiCityCenter()) end,
2   function() return AiWait(AiCityCenter()) end,
3   function() return AiSet(AiWorker(), 9) end,
4   function() return AiNeed(AiLumberMill()) end,
5   function() return AiNeed(AiBarracks()) end,
6   function() return AiWait(AiBarracks()) end,
7   function() return AiNeed(AiBlacksmith()) end,
8   function() return AiUpgradeTo(AiBetterCityCenter()) end,
9   function() return AiWait(AiBetterCityCenter()) end,
10  function() return AiNeed(AiStables()) end,
11  function() return AiWait(AiBestCityCenter()) end,
12  function() return AiNeed(AiAirport()) end,
13  function() return AiForce(2, {AiFlyer(), 1}) end,
14  function() return AiAttackWithForce(2) end,
```

Figura 5.2. Trecho que representa a etapa de construção de base da estratégia *Air Attack*. Essa estratégia privilegia ataques aéreos e, portanto, otimiza seu estágio de construção para possibilitar a criação de unidades aéreas o mais rápido possível. Os nós pertencentes a essa estratégia são ligados pelas arestas realçadas da figura 4.5.

5.1.2 Questionários

Como descrito, dois questionários foram elaborados para avaliação dos usuários, tendo o questionário pré-teste o objetivo de conhecer o perfil dos jogadores e o questionário pós-teste o de conhecer as opiniões dos jogadores quanto ao sistema de dicas. A cada usuário foi atribuído um identificador único, para poder agrupar suas respostas aos questionários. A todos os usuários ficou claro que seus dados pessoais não seriam publicados, nem usados para outros fins que não a organização dos dados dos questionários.

As perguntas referentes ao questionário pré-teste são exibidas na figura 5.3. Já o questionário pós-teste é descrito abaixo. A cada usuário foi atribuído um identificador único no formulário pré-teste. Para o formulário pós-teste a entrada de dados pessoais não foi necessária, devido à existência do identificador.

5.1.2.1 Questionário pós-teste

As figuras de 5.4 a 5.6 exibem cada uma das seções do questionário pós-teste. O objetivo desse questionário foi conhecer a opinião dos usuários sobre a qualidade do sistema de dicas e foi dividido em quatro partes, uma tratando de questões gerais sobre o sistema de dicas (figura 5.4), uma com questões sobre a utilidade do sistema de dicas (figura 5.5), outra sobre o sistema de síntese de voz (figura 5.6) e, finalmente, uma que permitia ao usuário fazer suas próprias considerações e apresentar opiniões sobre aspectos não abordados nas seções supracitadas.

Dados pessoais

Nome

Sexo

Idade

Questionário sobre jogos

Você possui intimidade com jogos RTS?

- ○ Desconheço totalmente
- ○ Conheço um pouco
- ○ Conheço completamente

Qual foi a última vez que você jogou um jogo RTS?

- ○ No último mês
- ○ Nos últimos seis meses
- ○ No último ano
- ○ Nos últimos três anos
- ○ Mais de três anos
- ○ Nunca joguei

Como você classificaria sua habilidade como jogador de RTS?

- ○ Péssimo
- ○ Ruim
- ○ Razoável
- ○ Bom
- ○ Ótimo

Figura 5.3. Perguntas do questionário pré-teste. Seu objetivo foi o de identificar o perfil do jogador.

Questões gerais sobre o sistema de dicas

Você já viu algum sistema semelhante a esse?

- ○ Sim
- ○ Não

Se sim, qual (ou quais?)

Você achou a abordagem interessante? "Abordagem" sendo haver um sistema de dicas ativo em pleno jogo.

- ○ Não
- ○ Sou indiferente
- ○ Sim

Figura 5.4. Seção sobre questões gerais do sistema no questionário pós teste.

Qualidade das dicas

O sistema de dicas foi útil?

- ○ Inútil
- ○ Na maior parte das vezes inútil
- ○ Indiferente
- ○ Na maior parte das vezes útil
- ○ Útil

Em que partes o sistema foi mais útil? Selecione qualquer habilidade do sistema que lhe foi útil. Se ele tiver sido totalmente inútil, ignore esta pergunta.

- □ Gerência de recursos
- □ Dicas táticas básicas
- □ Reconhecimento
- □ Contra-estratégia

Em que partes poderia haver mais dicas disponíveis?

- □ Gerência de recursos
- □ Reconhecimento do mapa
- □ Táticas de batalha
- □ Táticas de disposição de base
- □ Táticas de defesa
- □ Contra-estratégia
- □ Combinações de unidades para ataque
- □ Dicas de como investir os recursos

Frequência das dicas Dada a frequência atual de dicas dadas pelo sistema, como você as mudaria?

- ○ Muito menos frequentes
- ○ Menos frequentes
- ○ Equilíbrio ideal
- ○ Mais frequentes
- ○ Muito mais frequentes

Figura 5.5. Seção sobre a qualidade das dicas no questionário pós teste. Apesar de não exibida, também havia uma parte onde os usuários poderiam descrever características do sistema não abordadas pelas perguntas acima.

Dicas e síntese de voz

Nem todas as dicas foram sintetizadas, você acha que deveriam ter sido?

- ○ Sim
- ○ Não

Algumas dicas, além de serem exibidas na tela, foram sintetizadas para voz. Quão útil foi esse recurso?

- ○ Inútil
- ○ Na maior parte das vezes inútil
- ○ Indiferente
- ○ Na maior parte das vezes útil
- ○ Útil

Outras considerações sobre a síntese de voz Escreva aqui informações que você considera relevantes sobre a síntese de voz e que não foram abordadas pelas perguntas acima.

Figura 5.6. Seção sobre a síntese de voz no questionário pós-teste do sistema de dicas.

5.1.3 Teste piloto

O primeiro de todos os testes foi considerado o piloto. Nele, alguns problemas foram detectados e corrigidos para os testes subsequentes: durante a elaboração dos testes, estimou-se que os tempos de construção e treinamento de unidades poderiam ser muito grandes para realização de testes, o que poderia cansar os usuários. Com isso, foram geradas duas versões do jogo, uma a executar as tarefas de construção e treinamento na velocidade normal e outra em que essas velocidades foram dobradas.

Após a realização do primeiro teste, o participante relatou que "os testes foram realizados em uma configuração de jogo muito rápida, o que tornou o acompanhamento das dicas um pouco difícil". Com esse comentário, foi decidido que os testes posteriores seriam baseados na versão não acelerada do jogo.

Como o usuário submetido ao primeiro teste foi incomodado pela velocidade mais rápida, os resultados desse teste foram descartados da avaliação geral do sistema.

5.1.4 Resultados

Com as respostas dos usuários ao questionário pré-teste, tem-se que todos os usuários que participaram do teste são do sexo masculino e idade média entre 23 e 29 anos. Dados sobre a a intimidade dos jogadores com jogos RTS, como eles avaliam sua habilidade e a última vez desde que haviam jogado um jogo RTS são exibidos nas tabelas 5.2 a 5.4.

A observação dessas tabelas revela que o perfil dos usuários condiz com o perfil buscado

Tabela 5.2. Auto-avaliação dos usuários quanto a sua intimidade com jogos RTS, pergunta feita no questionário pré-teste. "Conhecer completamente" os jogos RTS deve ser compreendido como que o usuário já jogou RTS um número de vezes suficiente para poder jogar um jogo diferente dos que já está acostumado sem muitos problemas.

Avaliação dada	Participantes que avaliaram dessa forma	Porcentagem
Desconheço totalmente	0	0%
Conheço um pouco	3	50,0%
Conheço completamente	3	50,0%

Tabela 5.3. Respostas dos usuários quanto a última vez que jogaram jogos RTS, pergunta feita no questionário pré-teste.

Avaliação dada	Participantes que avaliaram dessa forma	Porcentagem
No último mês	0	0%
Nos últimos seis meses	1	16,7%
No último ano	2	33,3%
Nos últimos três anos	1	16,7%
Mais de três anos	2	33,3%
Nunca joguei	0	0%

Tabela 5.4. Auto-avaliação dos usuários quanto a seu nível de habilidade em jogos RTS, pergunta feita no questionário pré-teste.

Avaliação dada	Participantes que avaliaram dessa forma	Porcentagem
Péssimo	0	0%
Ruim	1	16,7%
Razoável	4	66,7%
Bom	1	16,7%
Ótimo	0	0%

quando da preparação dos testes: usuários com alguma intimidade com jogos RTS. Essa escolha de perfil foi feita para evitar que a percepção dos usuários quanto ao sistema de dicas não fosse afetada pela falta de intimidade com jogos RTS.

As tabelas 5.5 a 5.12 sumarizam as respostas dadas pelos jogadores avaliados no questionário pós-teste (descrito nas figuras 5.4 a 5.6) sobre os diversos aspectos do sistema de dicas.

Sobre os resultados das tabelas 5.5 e 5.6, estima-se que as opiniões negativas dos usuários, inutilidade do sistema e alta frequência de dicas geradas, se devam ao fato da maioria das dicas ativadas serem relacionadas à gerência de recursos, o que pode tê-las tornado repetitivas e, de certa forma, inconvenientes. De maneira semelhante, apesar de haver dicas codificadas relacionadas ao reconhecimento do ambiente, essas não foram ativadas com frequência satisfa-

tória, de modo que, quando foram ativadas, os usuários não as perceberam, já que não eram sintetizadas e apenas tinham seu texto exibido na interface do jogo.

Apesar das opiniões negativas quanto à frequência das dicas, os usuários não só as consideraram úteis (tabela 5.5) como consideraram a abordagem do sistema de dicas válida (tabela 5.9). Aspectos do sistema, no entanto, precisam ser melhorados, dado que, segundo a percepção dos usuários, o sistema de dicas se concentrou em aspectos de gerência de recursos e dicas táticas básicas (tabela 5.7). Os tópicos mais interessantes, segundo eles, para geração de novas dicas são em táticas de batalha, de disposição de base e de defesa (tabela 5.8), o que indica interesse dos usuários em um dos aspectos mais importantes do jogo: o combate.

Já sobre o sistema de síntese de voz, as opiniões dos usuários refletem que a proposta de síntese de mensagens lhes foi agradável (tabela 5.10), ainda que suas opiniões tenham ficado divididas entre a síntese ou não de todas as mensagens (tabela 5.11). Apesar da igual preferência entre síntese ou não de todas as mensagens, é importante notar que os usuários não foram submetidos a execuções em que todas as mensagens eram sintetizadas e, a julgar pela frequência de ativação de determinados ramos da árvore de decisão durante a execução, os usuários provavelmente ficariam irritados com sua frequência.

A tabela 5.12 indica que a maioria dos usuários participantes não conhecem sistemas semelhantes ao proposto neste trabalho. Os que foram citados como semelhantes, Warcraft III e SimCity foram discutidos na seção 2.3.

Pelas observação das reações dos usuários durante os testes, percebeu-se que, para eles, não existe separação clara entre o que seriam dicas de estratégia (e, portanto, relacionadas a ações no jogo) e dicas sobre a usabilidade da interface do jogo e sua mecânica. Como dicas sobre a interface do jogo não fizeram parte do escopo deste trabalho, alguns usuários se viram frustrados com essa decisão de projeto, pelo fato de não possuírem intimidade com Warcraft II e a interface ser diferente das por eles conhecidas.

Algumas considerações, no entanto, são difíceis de separar. Por exemplo: quando uma construção do tipo "Lumber Mill" ("Lm" na tabela 4.1) é construída próxima a florestas e há trabalhadores extraindo madeira na mesma floresta, a taxa de extração será maior que se a construção estivesse mais distante. Apesar de parecer uma conclusão óbvia quando se considera o funcionamento do processo de extração, essa conclusão não é para jogadores menos experientes em pleno jogo. Portanto, projetistas diferentes poderiam chegar a conclusões diferentes sobre quais dicas deveriam ser codificadas. Uma melhor alternativa seria implementar também essas dicas básicas e permitir ao usuário ativá-las ou não de acordo com sua preferência.

Os usuários foram convidados a sugerir melhorias ou mudanças para o sistema de dicas. Após análise de seus comentários, percebeu-se que as mudanças por eles propostas estão mais relacionadas a critérios de maior abrangência de competências do jogo RTS que a mudanças

Tabela 5.5. Utilidade do sistema de dicas conforme avaliado pelos usuários. As informações foram fornecidas por 6 usuários do sistema.

Avaliação dada	Participantes que avaliaram dessa forma	Porcentagem
Útil	0	0%
Na maior parte das vezes útil	5	83,3%
Indiferente	0	0%
Na maior parte das vezes inútil	1	16,7%
Inútil	0	0%

arquiteturais no sistema de dicas, o que pode indicar que a organização do sistema foi agradável aos usuários.

Na implementação atual, o sistema de dicas possui apenas duas opções quanto às dicas: habilitação ou desabilitação completa. Comentários dos usuários indicam que seria interessante que o sistema possuísse maior nível de controle, permitindo habilitar ou desabilitar dicas individuais ou grupos de dicas. Em particular, uma dica que foi muito habilitada, relacionada à existência de trabalhadores ociosos, causou irritação em alguns jogadores, confirmando a pertinência dessa sugestão. A causa da repetição decorre da estrutura da árvore de decisão, que habilita ações sempre que as condições são satisfeitas e, portanto, talvez fosse interessante experimentar a efetividade do sistema com outras estruturas de dados.

Outras considerações dos usuários incluem verificações para não deixar o jogador ocioso (e, portanto, em desvantagem contra um jogador não ocioso) e um aumento de interatividade com o sistema, de modo a permitir que consultas pudessem ser feitas ao sistema de dicas. Um questionamento comum dos jogadores era o que eles deveriam fazer para ter acesso a determinadas unidades. Essa é uma sugestão pertinente e parece razoável supor que a utilidade do sistema de dicas aumentaria caso houvesse funcionalidade semelhante. Houve também comentários sobre a calibração do sistema e, em retrospecto, seria interessante se o usuário pudesse definir valores de trabalho para o sistema de dicas. Por exemplo, os limites para notificação de razão de ouro e madeira são fixos, uma adição interessante ao sistema seria permitir que os próprios usuários pudessem definir suas constantes e variá-las ao longo do jogo, caso necessário.

5.1.4.1 Desempenho dos usuários

Todos os testes dos usuários foram gravados usando a funcionalidade de *replay* provida pelo motor Stratagus. O motor possui, implementado por padrão, um sistema de pontuação de jogadores que permanece ativo e atualizado durante toda uma partida. Durante a análise dos testes foi decidido observar qual o comportamento da pontuação dos jogadores ao longo do

Tabela 5.6. Opinião dos usuários quanto a frequência das dicas. Dada a frequência das dicas apresentadas, os usuários foram convidados a responder se a frequência das dicas era ideal ou se eles a mudariam.

Avaliação dada	Participantes que avaliaram dessa forma	Porcentagem
Muito menos frequentes	0	0%
Menos frequentes	3	50,0%
Equilíbrio ideal	2	33,3%
Mais frequentes	1	16,7%
Muito mais frequentes	0	0%

Tabela 5.7. Dicas mais úteis para o usuário. Avaliação dos usuários quanto à competências do jogo em que as dicas geradas foram mais úteis.

Avaliação dada	Participantes que avaliaram dessa forma	Porcentagem
Gerência de recursos	6	100%
Dicas táticas básicas	3	50,0%
Reconhecimento	1	16,7%
Contra-estratégia	0	0%

Tabela 5.8. Competências onde deveria haver mais dicas disponíveis segundo os usuários. Dado o conjunto de dicas geradas pelo sistema, os usuários foram convidados a sugerir quais competências adicionais o sistema de dicas deveria possuir.

Avaliação dada	Participantes que avaliaram dessa forma	Porcentagem
Gerência de recursos	0	0%
Reconhecimento de ambiente	2	33,3%
Táticas de batalha	5	83,3%
Táticas de disposição da base	4	66,7%
Táticas de defesa	4	66,7%
Contra-estratégia	2	33,3%
Combinações de unidades para ataque	3	50,0%
Como investir recursos	2	33,3%

Tabela 5.9. Opinião dos usuários sobre a abordagem do sistema de dicas. Se os usuários achavam que a abordagem usada pelo sistema de dicas foi adequada (como mensagens na tela e síntese de voz) sua resposta foi "Sim". Caso contrário, "Não".

Avaliação dada	Participantes que avaliaram dessa forma	Porcentagem
Sim	6	100%
Não	0	0%

Tabela 5.10. Utilidade da síntese de voz por parte do sistema de notificação. As informações foram fornecidas por 6 usuários do sistema.

Avaliação dada	Participantes que avaliaram dessa forma	Porcentagem
Útil	2	33,3%
Na maior parte das vezes útil	2	33,3%
Indiferente	1	16,7%
Na maior parte das vezes inútil	1	16,7%
Inútil	0	0%

Tabela 5.11. Opinião dos usuários quanto a síntese de todas as dicas. Essa pergunta foi feita por que, por decisão de projeto, nem todas as dicas foram convertidas de texto para fala. O objetivo dessa pergunta era saber se eles concordariam com essa decisão após experimentar o sistema.

Avaliação dada	Participantes que avaliaram dessa forma	Porcentagem
Sim	3	50,0%
Não	3	50,0%

Tabela 5.12. Usuários que conheciam sistemas semelhantes ao apresentado neste trabalho. Se os usuários conheciam sistemas semelhantes, sua resposta foi "Sim". Caso contrário, "Não".

Avaliação dada	Participantes que avaliaram dessa forma	Porcentagem
Sim	2	33,3%
Não	4	66,7%

tempo. Como em todos os testes os usuários possuíam apenas um oponente, foi decidido que a função de desempenho a ser usada seria a pontuação do usuário subtraída da pontuação de seu oponente, de modo que, quando positiva, ela indicasse que o usuário estava ganhando o jogo e, quando negativa, que o usuário estava perdendo. A pontuação do jogador é definida pela quantidade de danos causados, representados pelas unidades destruídas pelo jogador. A tabela 5.13 exibe o valor, em pontos, concedido ao usuário após a destruição de unidades de cada tipo. A pontuação total do usuário é dada pela soma dos pontos individuais adquiridos.

Para cada conjunto de testes foram plotados os gráficos de desempenho dos usuários. Sua observação permitiu perceber que a pontuação dos jogadores aumentou com a adição do sistema de dicas. O gráfico 5.7a exibe o desempenho de um jogador que teve seu desempenho próximo ao desempenho comum dos jogadores que perderam a primeira partida. O que sugere que a adição do sistema de dicas pode ter contribuído para a melhoria do desempenho dos jogadores.

Essa métrica, quando usada sem outras evidências pode levar a conclusões erradas.

Tabela 5.13. Valores, em pontos, concedidos ao usuário por destruição de unidade. Alguns nomes de unidades foram encurtados, sem ambiguidade, por questões de espaço na folha.

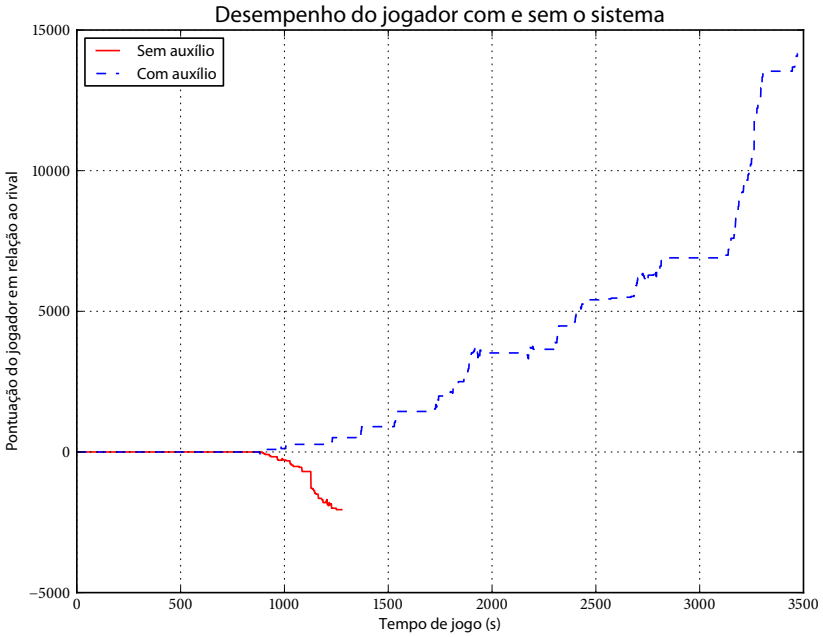
Unidade	Pontos	Unidade	Pontos
Wall	1	Tower	95
Critter	1	Farm	100
Peasant/Peon	30	Lumber mill	150
Flying Machine/Zeppelin	40	Runestone	150
Tanker	40	Barracks	160
Footman/Grunt	50	Oil Rig	160
Transport	50	Blacksmith	170
Archer/Axe Thrower	60	Shipyard	170
Ranger/Berserker	70	Foundry	200
Dwarves/Sappers	100	Guard Tower	200
Knight/Ogre	100	Refinery	200
Ballista/Catapult	100	Town Hall	200
Mage/Death Knight	100	Stables/Ogre Mound	210
Demon	100	Inventor/Alchemist	230
Paladin/Ogre Mage	110	Church/Altar	240
Legendary Hero	120	Mage Tower / Temple	240
Submarine/Turtle	120	Cannon Tower	250
Destroyer	150	Aviary/Roost	280
Gryphon/Dragon	150	Keep/Stronghold	600
Battleship/Juggernaut	300	Castle/Fortress	1500

Considere, por exemplo, o gráfico exibido na figura 5.7b. Nele, é possível ver que a pontuação do jogador sem o sistema foi em sua maior parte crescente. No entanto, isso aconteceu por que o usuário, apesar de ter ficado sem recursos, havia construído diversas torres de defesa ao redor de sua base. Assim, sempre que forças inimigas iam atacá-lo, elas eram destruídas pelas torres de defesa, mas não sem antes danificar as torres. Com o avanço das forças inimigas, os trabalhadores do usuário foram destruídos, constituindo uma derrota com pontuação alta.

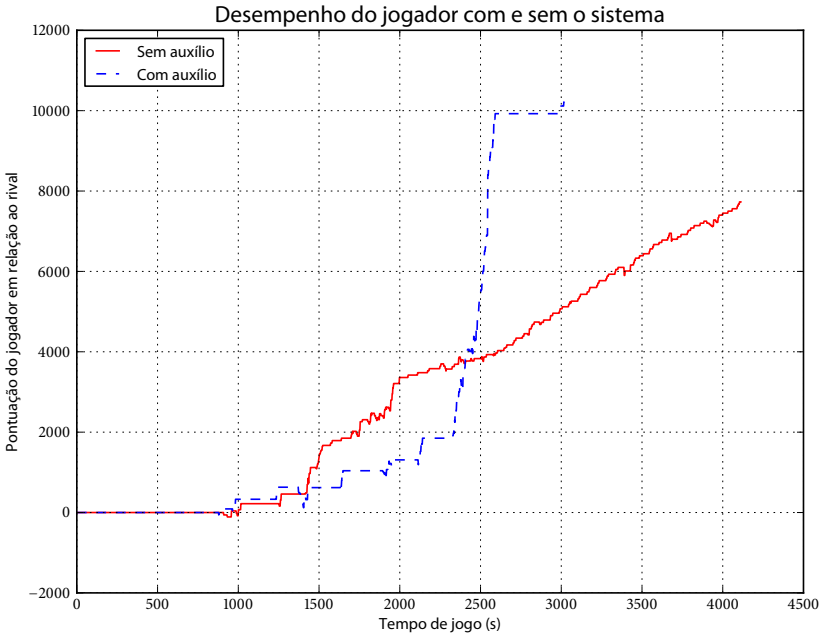
5.2 Testes de desempenho

Em jogos, é prática comum adotar um valor fixo de Quadros por Segundo (*Frames per Second*, ou FPS) para garantir a qualidade do sistema. A partir do FPS fixo, é possível determinar o tempo máximo que o laço principal do jogo pode gastar em execução.

No caso do Stratagus, um valor de 30 FPS é usado, o que quer dizer que cada iteração do laço principal deve executar em, no máximo, $\frac{1}{30}$ = 33,3ms para fazer com que a experiência do usuário seja agradável. O objetivo dos testes descritos nessa seção é o de observar o desempenho



(a) Desempenho típico de um jogador sem e com o sistema de dicas.



(b) Desempenho de um jogador que, apesar de derrotado sem o sistema de dicas, conseguiu boa pontuação.

Figura 5.7. Desempenhos de jogadores com e sem o sistema de dicas. Em (a) é exibido o desempenho típico de jogadores observado durante os testes e (b) exibe o desempenho "anômalo" de um jogador que, mesmo derrotado, conseguiu boa pontuação. O valor exibido no eixo vertical é a diferença entre a pontuação do usuário e a pontuação de seu oponente (de computador). A pontuação de um jogador é função de unidades destruídas. As regiões planas do gráfico representam intervalos em que não houve combate.

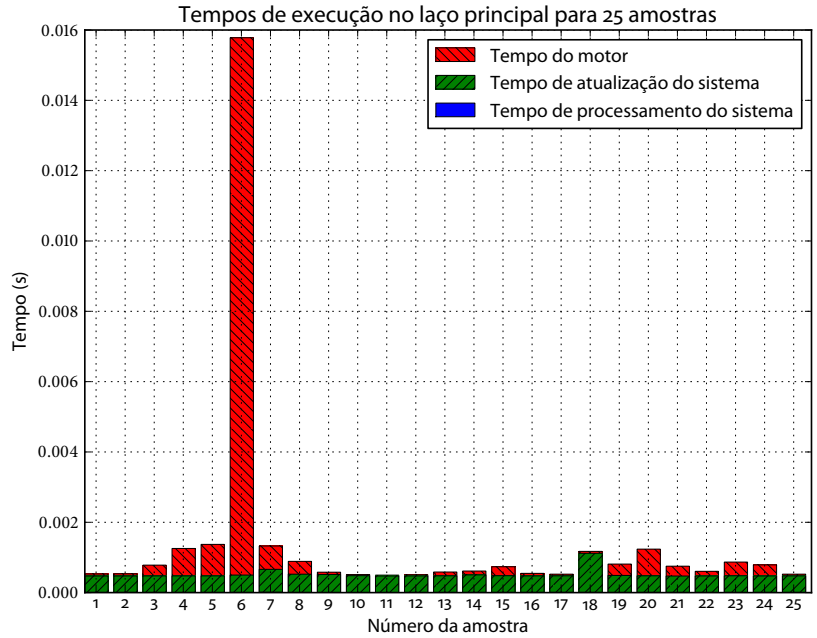
computacional do sistema implementado comparado ao resto do motor e verificar se a adição de dicas degradou a eficiência do código, excedendo o limite de 33,3ms.

Para execução dos testes, a limitação de FPS do motor do jogo foi desabilitada pois, se permanecesse habilitado, o tempo mínimo de execução de um quadro seria de 33,3ms. A taxa de atualização de vídeo, no entanto, permaneceu constante.

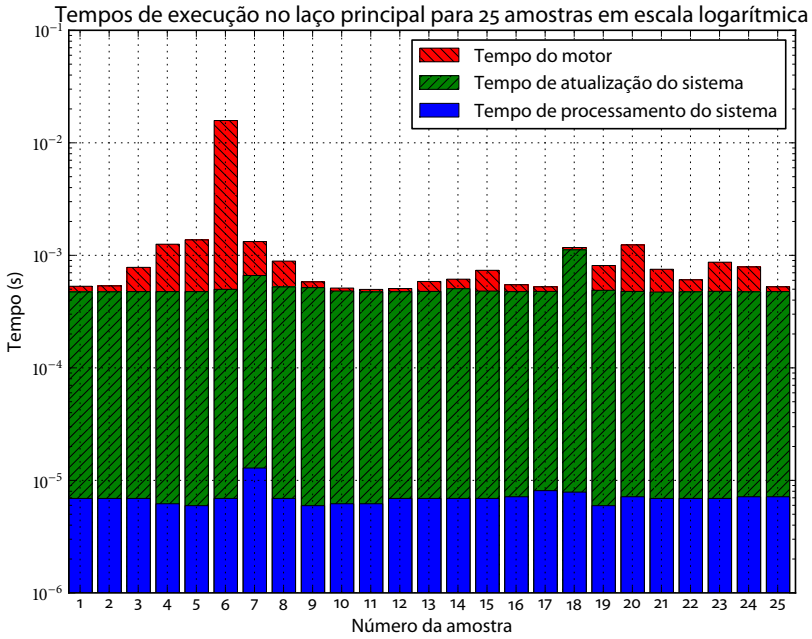
Os testes de desempenho foram realizados em um computador com *chipset* Intel® 945G, processador *dual core* Core 2 Duo T7400, com frequência de 2.16GHz e 4MB de cache L2 para cada núcleo e 4GB de memória DDR2-800 *dual channel* executando o sistema operacional Arch Linux 64bits com *kernel* versão 2.6.33 e biblioteca padrão de C glibc 2.11.1. Os binários foram compilados pelo compilador g++, versão 4.5.0 com argumentos de otimização "O2" (argumento padrão para compilação do Stratagus).

Para medição do desempenho do sistema, o código do motor foi instrumentado para computar o tempo de execução das etapas de atualização e processamento do sistema de dicas e o tempo total de execução de uma iteração do laço principal do motor. Esses tempos foram medidos uma vez por segundo e o processo de medição ocorreu através da execução dos *replays* dos jogos dos usuários que participaram do teste do sistema.

As figuras 5.8a e 5.8b exibem o tempo de execução para 25 amostras diferentes dos tempos de execução em escala linear e logarítmica. A escolha da posição da primeira amostra na massa de dados foi aleatória e as outras 24 amostras foram feitas a partir dessa primeira. Foi necessário representar os mesmos dados em dois gráficos diferentes para que fosse possível observar tanto as diferenças em ordens de grandeza quanto as diferenças de tempo em escala logarítmica. Estima-se que o maior tempo de execução por parte do motor durante a amostra de número seis tenha sido relacionada a tarefas de manutenção interna do motor, ou à atualização de vídeo. No entanto, a informação mais importante a ser extraída desses gráficos é que o tempo de execução do sistema de dicas é praticamente constante e não influencia negativamente o motor a ponto de atrapalhar suas outras atividades. As tabelas 5.14 a 5.16, que agrupam amostras em intervalos de tempo, exibem informações que corroboram essa afirmação. Na tabela 5.14 é exibido que o tempo de atualização do sistema de dicas consome, em mais de 99% do tempo leva menos de três milissegundos para ser processada. A tabela 5.15 exibe dados semelhantes, mas, como exibido nos gráficos possui tempo de execução ainda menor. A tabela 5.16, por sua vez, exibe dados que indicam que, mesmo com a adição do sistema de dicas, ainda mais de 94% dos quadros amostrados executaram em menos de três milissegundos, indicando que a adição do sistema de dicas não influenciou negativamente a execução do motor.



(a) Tempo de execução de um conjunto de quadros selecionado aleatoriamente.



(b) Tempo de execução do mesmo conjunto exibido em (a), mas exibido em escala logaritmica.

Figura 5.8. Tempos de execução de um conjunto de quadros do jogo. Nos gráficos são exibidos um conjunto de quadros típicos do jogo. Em (a) é exibido o tempo de execução absoluto dos quadros. Já em (b), uma escala logarítmica é usada. O alto valor registrado pela amostra de número seis é devido à atualização do vídeo pelo motor. Como os dados foram coletados em modo rápido do motor, apenas em algumas iterações do laço principal é que há atualização do vídeo, por isso a diferença nos tempos de outros quadros.

Tabela 5.14. Desempenho da etapa de atualização do sistema. Na tabela, os tempos de atualização das estruturas de dados para sete jogos foram divididos em 10 grupos de $3ms$.

Intervalo de tempo (ms)	Quantidade de amostras	Porcentagem
$0 \leq t < 3$	444134	99,0166%
$3 \leq t < 6$	2250	0,5016%
$6 \leq t < 9$	824	0,1837%
$9 \leq t < 12$	521	0,1162%
$12 \leq t < 15$	342	0,0762%
$15 \leq t < 18$	175	0,0390%
$18 \leq t < 21$	119	0,0265%
$21 \leq t < 24$	74	0,0165%
$24 \leq t < 27$	37	0,0082%
$27 \leq t < 30$	26	0,0058%
$t \geq 30{,}0$	43	0,0096%

Tabela 5.15. Desempenho da etapa de processamento do sistema. Na tabela, os tempos de processamento para sete jogos foram divididos em 10 grupos de $3ms$.

Intervalo de tempo (ms)	Quantidade de amostras	Porcentagem
$0 \leq t < 3$	447860	99,8473%
$3 \leq t < 6$	4	0,0009%
$6 \leq t < 9$	7	0,0016%
$9 \leq t < 12$	164	0,0366%
$12 \leq t < 15$	32	0,0071%
$15 \leq t < 18$	85	0,0190%
$18 \leq t < 21$	140	0,0312%
$21 \leq t < 24$	53	0,0118%
$24 \leq t < 27$	15	0,0033%
$27 \leq t < 30$	11	0,0025%
$t \geq 30{,}0$	174	0,0388%

Tabela 5.16. Desempenho do laço principal do jogo com as adições do sistema. Na tabela, os tempos de processamento do motor para sete jogos foram divididos em 10 grupos de $3ms$.

Intervalo de tempo (ms)	Quantidade de amostras	Porcentagem
$0 \leq t < 3$	423313	94,3747%
$3 \leq t < 6$	10674	2,3797%
$6 \leq t < 9$	3209	0,7154%
$9 \leq t < 12$	1552	0,3460%
$12 \leq t < 15$	2186	0,4874%
$15 \leq t < 18$	1438	0,3206%
$18 \leq t < 21$	1098	0,2448%
$21 \leq t < 24$	1641	0,3658%
$24 \leq t < 27$	1211	0,2700%
$27 \leq t < 30$	566	0,1262%
$t \geq 30{,}0$	1657	0,3694%

Capítulo 6

Conclusões e trabalhos futuros

6.1 Conclusões

Neste trabalho foi apresentado um sistema de apoio ao jogador para jogos RTS baseado em árvores de decisão. Também foram apresentados aspectos da IA de jogos RTS, algumas técnicas usadas para implementá-los e as competências necessárias para dominar esse tipo de jogo.

Através dos experimentos foi possível perceber que a abordagem gerou resultados positivos e possui potencial para implementação em jogos reais, além de ser possível estender o sistema para suprir melhor as necessidades do usuário.

Os testes com usuário desempenharam papel fundamental para determinação dos pontos positivos e negativos da abordagem e os testes beta permitiram que erros de programação pudessem ser detectados antes que os testes com usuário tivessem início, poupando tempo durante essa etapa. De um modo geral, durante os testes os usuários se mostraram motivados pelo sistema de dicas, sugerindo que uma abordagem semelhante seja aplicável à indústria de jogos.

Diferente de algumas abordagens usadas em jogos, uma vantagem da abordagem deste trabalho é que ele não possuía o objetivo de guiar as decisões do usuário, mas de auxiliá-lo a melhorar seu desempenho, fossem quais fossem suas decisões. A forma de exibição das mensagens, no entanto, poderia ser incrementada, já que a única forma usada para persisti-las foi através do uso de uma janela externa ao jogo, o que confundiu os usuários. Durante o processo de desenvolvimento foi cogitada a possibilidade de implementar uma região na interface do jogo para persistir as mensagens, mas a falta de intimidade com o código responsável pela interface gráfica do motor inviabilizou essa abordagem.

Em retrospecto, uma abordagem que talvez fosse mais interessante para a coleta de dados seria a exportação do estado interno do motor via rede, permitindo que o processamento ocorresse em outro processo, impedindo que erros de programação no sistema de dicas

viessem a comprometer a correta execução do jogo, como aconteceu durante o processo de desenvolvimento deste trabalho. Outra vantagem dessa abordagem seria a possibilidade de usar outras linguagens de programação com maior poder de expressão que C++, linguagem em que o motor foi escrito.

Como percebido pelos usuários, a abordagem de árvores de decisão pode não ter sido a melhor escolha para estrutura de dados, já que foi necessário condicionar o tempo de ativação de alguns ramos da árvore ao tempo em que uma ação foi executada por último, para evitar mensagens repetidas. Outra abordagem que poderia ter sido usada para evitar a repetição de mensagens seria, no caso das mesmas decisões acontecerem consecutivamente, usar um tempo variável para as dicas, para evitar incomodar o usuário com dicas repetidas. Os testes com usuários foram úteis e aumentaram a confiança na robustez e qualidade do sistema. No entanto, só foi possível executá-los com suavidade graças aos testes beta, que identificaram diversos erros de programação que só eram ativados em plataformas específicas.

Apesar da utilidade do motor Stratagus, talvez ele não tenha sido a melhor escolha para esse trabalho, pois, apesar de usado por diversos grupos de pesquisa, as adições feitas por cada grupo tornam os arquivos gerados incompatíveis entre si. Além disso, por Warcraft II ter sido um jogo sem suporte nativo a *replays* e por Wargus não ter tido popularidade suficiente para haver *replays* de bons jogos disponíveis publicamente, toda a observação do comportamento do sistema de dicas durante o desenvolvimento foi baseada na execução de jogos de IA contra IA: o código do motor foi modificado para, ao invés de receber comandos do jogador, receber comandos dw estratégias já existentes para o jogo. Como as estratégias de Wargus são fixas, alguns ramos da árvore criada eram executados mais comumente que outros. Essa limitação gerou problemas durante a execução dos testes com usuários: como a calibração do código foi baseada no resultado dos jogos de IA contra IA e dado o fato de que os usuários não jogaram otimizando um caminho no grafo tecnológico, alguns ramos da árvore foram executados demasiadamente, enquanto outros quase não foram ativados. Conversas com outros pesquisadores sugeriram que soluções baseadas em *data mining* teriam mais sucesso caso fosse utilizado um jogo mais popular, como Starcraft, que possui milhares de *replays* disponíveis na Internet para serem analisados.

6.2 Trabalhos futuros

Problemas na implementação atual incluem alta repetitividade de algumas dicas quando as condições são satisfeitas e ausência de dicas em domínios considerados úteis pelos usuários. Esses problemas seriam sanados através da regulagem de parâmetros e cadastramento de novas dicas. No entanto, uma deficiência fundamental do sistema de dicas atual é que é necessário

programar manualmente as regras para as dicas. Uma adição essencial para este trabalho seria a criação de um formato de regras, de modo que a implementação das regras do sistema pudesse ser feita de forma independente do motor, permitindo, também, a criação de editores de regras.

Outras sugestões para trabalhos futuros são listadas abaixo:

- Detecção automática de planos através de alguma técnica de aprendizado de máquina, como foi aplicado com sucesso em Ontañón et al. [2007], ou em Weber & Mateas [2009]. A abordagem usada em Weber & Mateas [2009], no entanto, pode não ser muito útil para o caso do Wargus, posto que Warcraft II é um jogo que não possuía inicialmente a capacidade de realizar *replays* e, portanto, a quantidade de dados de jogos disponíveis publicamente é limitada. Ainda outra alternativa para detecção de planos inimigos e locais seria usar a abordagem descrita em Goldman et al. [1999], em que é usada teoria probabilística para detecção de planos. Quanto à detecção de planos, ela pode ser aplicada tanto aos oponentes (com informação imperfeita) quanto às ações do jogador.

- Implementação de alguma técnica de aprendizado de máquina, como Aprendizado por Reforço, para treinamento do sistema de dicas, de modo que ele pudesse aprender novas táticas e fosse capaz de ensinar as técnicas aprendidas a jogadores inexperientes. Ainda outra adição possível seria a criação de um formato de intercâmbio de dicas de estratégia, de modo que as táticas aprendidas pelo agente de um jogador pudessem ser compartilhadas com outros jogadores.

- Outra adição interessante para este trabalho seria a avaliação da efetividade do sistema em situações de humanos contra humanos. Ainda em situações de humanos contra humanos, seria interessante permitir ao sistema que, em uma nova partida, ele fosse capaz de buscar *replays* do oponente atual e tentar inferir seu estilo de jogo e, além de fornecer dicas sobre como combater o estilo de jogo, possibilitar ao usuário obter estatísticas sobre seu oponente. Com a existência de servidores que permitem batalhas online, estatísticas sobre jogadores costumam estar disponíveis, o que torna essa abordagem viável.

Referências bibliográficas

Abramov, L.; Bagirov, V.; Botvinnik, M.; Cvetkovic, S.; Filip, M.; Geller, E.; Gipslis, A.; Gufeld, E.; Hort, V.; Kasparov, G.; Korchnoi, V.; Krnic, Z.; Larsen, B.; Matanović, A.; Minev, N.; Nunn, J.; Parma, B.; Polugaevsky, L.; Suetin, A.; Sveshnikov, E.; Taimanov, M.; Ugrinovic, D. & Uhlmann, W. (2010). Encyclopaedia of chess openings. http://www.sahovski.com/products/eco/index.php. Accessed in April 18th, 2010.

Aha, D. W.; Molineaux, M. & Ponsen, M. J. V. (2005). Learning to win: Case-Based plan selection in a Real-Time Strategy games. Em *ICCBR*, pp. 5–20.

Alcázar, V.; Borrajo, D. & Linares, C. (2008). Modelling a RTS planning domain with Cost Conversion and rewards. Em Botea, A. & López, C. L., editores, *ECAI*, Patras, Greece.

Balla, R.-K. (2009). UCT for tactical assault battles in real-time strategy games. Dissertação de mestrado, Oregon State University.

Baumgarten, R.; Colton, S. & Morris, M. (2009). Combining ai methods for learning bots in a Real-Time Strategy Game. *International Journal of Computer Games Technology*, 2009:10.

Bergsma, M. & Spronck, P. (2008). Adaptive spatial reasoning for turn-based strategy games. Em *Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Black, A. W. & Taylor, P. A. (1997). The Festival Speech Synthesis System: System documentation. Relatório técnico HCRC/TR-83, Human Communciation Research Centre, University of Edinburgh, Scotland, UK. Disponível em http://www.cstr.ed.ac.uk/projects/festival.html.

Blizzard (2010). Warcraft™ II strategy. http://classic.battle.net/war2/strategy.shtml. Acessado em 20 de abril de 2010.

Botea, A.; Müller, M. & Schaeffer, J. (2004). Near optimal hierarchical path-finding. *Journal of Game Development*, 1.

Buro, M. (2002). ORTS: A hack-free RTS game environment. Em *Proceedings of the International Computers and Games Conference*.

Buro, M. (2004). Call for AI research in RTS games. Em *Proceedings of the 4th Workshop on Challenges in Game AI*, San Jose.

Buro, M. & Furtak, T. M. (2004). RTS games and real-time AI research. Em *Proceedings of the Behavior Representation in Modeling and Simulation Conference (BRIMS)*, pp. 51–58.

Chung, M.; Buro, M. & Schaeffer, J. (2005). Monte Carlo planning in RTS games. Em *CIG*. IEEE.

Cormen, T. H.; Leiserson, C. E.; Rivest, R. L. & Stein, C. (2002). *Algoritmos Teoria & Prática*. Elsevier, 2ª edição.

da Silva Corrêa Pinto, H. & Alvares, L. O. (2006). Behavior-based robotic architectures for games. Em *Game Programming Gems 6*, capítulo 3.3, pp. 235–244. Charles River Media, Inc.

Fagan, M. & Cunningham, P. (2003). Case-based plan recognition in computer games. Em *Proceedings of the Fifth International Conference on Case-Based Reasoning*, pp. 161–170. Springer.

Forbus, K. D.; Mahoney, J. V. & Dill, K. (2001). How qualitative spatial reasoning can improve strategy game AIs. Em *15th International Workshop on Qualitative Reasoning (QR01)*. AAAI Press.

Goldman, R. P.; Geib, C. W. & Miller, C. A. (1999). A new model of plan recognition. Em Laskey, K. B. & Prade, H., editores, *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann.

Hagelbäck, J. & Johansson, S. J. (2008). Using multi-agent potential fields in real-time strategy games. Em Padgham, L.; Parkes, D. C.; Müller, J. & Parsons, S., editores, *AAMAS (2)*, pp. 631–638. IFAAMAS.

Hagelbäck, J. & Johansson, S. J. (2009). A multiagent potential field-based bot for real-time strategy games. *International Journal of Computer Games Technology*, 2009:10.

Hoang, H.; Lee-urban, S. & Muñoz-avila, H. (2005). Hierarchical plan representations for encoding strategic game AI. Em *Proceedings of the Artificial Intelligence and Interactive Digital Entertainment Conference*. AAAI Press.

Ierusalimschy, R. (2006). *Programming in Lua*. Lua.org. ISBN 85-903798-2-5.

Khoo, A.; Dunham, G.; Trienens, N. & Sood, S. (2002). Efficient, Realistic NPC control systems using Behavior-Based techniques. Em *Proceedings of the AAAI 2002 Spring Symposium Series: Artificial Intelligence and Interactive Entertainment*, Menlo Park, CA.

Laird, J. E. (2002). Research in human-level AI using computer games. *Communications of the ACM*, 45(1):32–35.

Laird, J. E. & Jones, R. M. (1998). Building advanced autonomous AI systems for large scale real time simulations. Em Freeman, M., editor, *Proceedings of the 1998 Computer Game Developers' Conference*, pp. 365–378.

Laird, J. E. & van Lent, M. (2000). Human-level AI's killer application: Interactive computer games. Em *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pp. 1171–1178. AAAI Press / The MIT Press.

Lee-Urban, S.; Parker, A.; Kuter, U.; Munoz-Avila, H. & Nau, D. (2007). Transfer learning of hierarchical task-network planning methods in a Real-Time Strategy games. Em *Proceedings of the Workshop on Artificial Intelligence Planning and Learning*.

Madeira, C.; Corruble, V. & Ramalho, G. (2005). Generating adequate representations for learning from interaction in complex multiagent simulations. Em *IEEE/WIC/ACM International Joint Conference on Intelligent Agent Technology*.

Madeira, C.; Corruble, V. & Ramalho, G. (2006). Designing a reinforcement learning-based adaptive AI for large-scale strategy games. Em *Proceedings of the Second Conference on Artificial Intelligence and Interactive Digital Entertainment*.

Madeira, C.; Corruble, V.; Ramalho, G. & Ratitch, B. (2004). Bootstrapping the learning process for the semi-automated design of a challenging game ai. Em *AAAI Workshop on Challenges in Game AI*.

Magerko, B.; Laird, J. E.; Assanie, M.; Kerfoot, A. & Stokes, D. (2004). AI characters and directors for interactive computer games. Em *Proceedings of the 2004 Innovative Applications of Artificial Intelligence Conference*, pp. 877–883. AAAI Press.

Mamei, M. & Zambonelli, F. (2004). Field-based motion coordination in Quake 3 arena. Em *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 1532–1533.

McCoy, J. & Mateas, M. (2008). An integrated agent for playing real-time strategy games. Em *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*.

Metoyer, R.; Stumpf, S.; Neumann, C.; Dodge, J.; Cao, J. & Schnabel, A. (2010). Explaining how to play real-time strategy games. *Knowledge-Based Systems*, 23(4):295–301. ISSN:0950-7051.

Millington, I. (2006). *Artificial Intelligence for Games*. Series in Interactive 3D Technology. Morgan Kaufman, 1 edição. ISBN 13: 978-0-12-497782-2.

Mishra, K.; Ontañón, S. & Ram, A. (2008). Situation assessment for plan retrieval in Real-Time Strategy games. Em Althoff, K.-D.; Bergmann, R.; Minor, M. & Hanft, A., editores, *ECCBR*, volume 5239 of *Lecture Notes in Computer Science*, pp. 355–369. Springer.

Ontañón, S.; Mishra, K.; Sugandh, N. & Ram, A. (2007). Case-Based planning and execution for Real-Time Strategy games. Em Weber, R. & Richter, M. M., editores, *ICCBR*, volume 4626 of *Lecture Notes in Computer Science*, pp. 164–178. Springer.

Orkin, J. (2005). Agent architecture considerations for Real-Time Planning in games. Em Young, R. M. & Laird, J. E., editores, *AIIDE*, pp. 105–110. AAAI Press.

Ponsen, M.; Munoz-Avila, H.; Spronck, P. & Aha, D. W. (2006). Automatically generating game tactics through evolutionary learning. *AI Magazine*, 27(3):75–84.

Ponsen, M. J.; Lee-Urban, S.; Muñoz-Avila, H.; Aha, D. W. & Molineaux, M. (2005). Stratagus: An open-source game engine for research in real-time strategy games. Relatório técnico, Navy Center for Naval Research Laboratory.

Reynolds, C. W. (1987). Flocks, Herds, and Schools: A distributed behavioral model. *Computer Graphics*, 21(4):25–34.

Reynolds, C. W. (1999). Steering behaviors for autonomous characters. Em *Game Developers Conference 1999*.

Russell, S. J. & Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2 edição.

Sailer, F.; Buro, M. & Lanctot, M. (2007). Adversarial planning through strategy simulation. Em *CIG*, pp. 80–87. IEEE.

Schaeffer, J.; Burch, N.; Björnsson, Y.; Kishimoto, A.; Müller, M.; Lake, R.; Lu, P. & Sutphen, S. (2007). Checkers is solved. *Science*, 317:1518–1522.

Sharma, M.; Holmes, M.; Santamaria, J.; Irani, A.; Isbell, C. & Ram, A. (2007). Transfer learning in Real-Time Strategy Games using hybrid CBR/RL. Em Veloso, M. M., editor, *IJCAI*.

Stoykov, S. (2008). Using a competitive approach to improve military simulation artificial intelligence design. Dissertação de mestrado, Naval Postgraduate School.

Stratagus (2010). Stratagus: A real time strategy engine. http://stratagus.sourceforge.net/games.shtml. Acessado em 18 de abril de 2010.

Sturtevant, N. & Buro, M. (2005). Partial pathfinding using map abstraction and refinement. Em *Proceedings of the twentieth national conference on Artificial Intelligence*, pp. 1392–1397, Pittsburg.

Sturtevant, N. R. & Buro, M. (2006). Improving collaborative pathfinding using map abstraction. Em *Proceedings of the Second Artificial Intelligence and Interactive Digital Entertainment Conference*, pp. 80–85.

Sugandh, N.; Ontañón, S. & Ram, A. (2008a). On-Line Case-Based plan adaptation for Real-Time Strategy games. Em Fox, D. & Gomes, C. P., editores, *AAAI*, pp. 702–707. AAAI Press.

Sugandh, N.; Ontañón, S. & Ram, A. (2008b). Real-Time plan adaptation for Case-Based planning in Real-Time Strategy games. Em Althoff, K.-D.; Bergmann, R.; Minor, M. & Hanft, A., editores, *ECCBR*, volume 5239 of *Lecture Notes in Computer Science*, pp. 533–547. Springer.

Tomlinson, S. L. (2004). The long and short of steering in computer games. *International Journal of Simulations*, 1:33–46.

Tozour, P. (2001a). *Game programming gems*, capítulo 3.6 - Influence Mapping, pp. 281–297. Charles River Media, Inc.

Tozour, P. (2001b). *Game programming gems*, capítulo 3.7 - Stragic Assessment Techniques, pp. 281–297. Charles River Media, Inc.

van Lent, M.; Laird, J.; Buckman, J.; Hartford, J.; Houchard, S.; Steinkraus, K. & Tedrake, R. (1999). Intelligent agents in computer games. Em *Proceedings of The Sixteenth National Conference on Artificial Intelligence*, pp. 929–930. AAAI Press.

Weber, B. G. & Mateas, M. (2009). A data mining approach to strategy prediction. Em Lanzi, P. L., editor, *Proceedings of the IEEE Symposium on Computational Intelligence & Games*. IEEE.

Wintermute, S.; Xu, J. & Laird, J. E. (2007). SORTS: A human-level approach to Real-Time Strategy AI. Em Schaeffer, J. & Mateas, M., editores, *AIIDE*, pp. 55–60. The AAAI Press.