Aplicando Conceitos de Sistemas Multiagentes na Elaboração de um Jogo de Estratégia Simulado

Maicon Rafael Zatelli [1]
José Rodrigo Ferreira Neri [1]
Daniela Maria Uez [1]
Rafael Frizzo Callegaro [1]

Resumo: O uso do paradigma de agentes está cada vez maior no campo de desenvolvimento de jogos. As características dos agentes como autonomia, proatividade e interação social, podem auxiliar no desenvolvimento de jogos mais realistas e detalhados. Este trabalho tem como objetivo integrar diversas ferramentas e conceitos da área de inteligência artificial, especialmente sistemas multiagentes, para o desenvolvimento de um jogo de estratégia simulado. Além de agentes, foram utilizados os conceitos de organização, ambiente, reputação, sistemas especialistas e ontologias.

Abstract: The use of the agent paradigm is steadily increasing in the games development field. Some agent features, like autonomy, proactivity, and sociability help us to develop more realistic and detailed games. This work shows how to integrate some artificial intelligence tools and concepts, specially multi-agent systems, to develop a strategic game. Beyond the agents, this game uses organization, environment, reputation, expert systems, and ontologies.

1 Introdução

O paradigma de agentes tem sido cada vez mais utilizado pelos desenvolvedores de jogos. As características dos agentes, como autonomia, proatividade e interação social, podem auxiliar no desenvolvimento de jogos com maior riqueza em detalhes e inteligência mais credível [1, 2, 3]. Além disso, jogos são bons cenários para permitir o uso conjunto de diversos conceitos de inteligência artificial (IA). Esta é a principal motivação que levou ao desenvolvimento deste trabalho: integrar diversas ferramentas e conceitos da área de IA, especialmente sistemas multiagentes (SMA), para solução de um problema.

Este trabalho tem o objetivo de detalhar como foi o processo de desenvolvimento de um jogo utilizando conceitos de SMA e ilustrar a participação de cada ferramenta e técnica empregada. O desenvolvimento do jogo ocorreu no contexto de uma disciplina de mestrado e utilizou-se duas arquiteturas diferentes de agentes (reativa e BDI), os agentes foram desen-

[1]Programa de Pós-Graduação em Engenharia de Automação e Sistemas, UFSC, Caixa Postal 476, CEP 88040-900
{maicon, jrneri, dmuez, callegaro@das.ufsc.br}

volvidos em diferentes linguagens de programação (JADE e Jason) e também foram utilizados conceitos de organização, ambiente, reputação, ontologias e sistemas especialistas.

Como resultado conseguiu-se aproximar o comportamento dos personagens do jogo com a realidade de um jogador humano, uma vez que as ferramentas e técnicas utilizadas procuram justamente imitar ou facilitar a representação do comportamento humano. Por exemplo, o uso de sistemas especialistas permite simular o raciocínio de um especialista humano que atua em uma determinada área, os agentes possuem autonomia e conseguem se adaptar ao ambiente assim como os humanos, a reputação pode ser utilizada pelos agentes para encontrar os melhores parceiros para realizar negócios, e assim por diante.

Este trabalho está estruturado em seis seções. Na primeira foi dada uma breve introdução a respeito do objetivo deste artigo. Na segunda são introduzidos alguns trabalhos relacionados à aplicação de SMA em jogos. Na terceira é apresentada uma visão geral sobre o jogo. Na quarta é descrito como o jogo foi desenvolvido, detalhes da implementação e onde cada ferramenta foi utilizada. Na quinta são mostrados os principais desafios enfrentados durante o desenvolvimento. Por fim, na sexta seção, são enfatizadas as principais contribuições e na última seção são descritas as considerações finais.

2 Trabalhos Relacionados

Atualmente existem diversos trabalhos que utilizam SMAs no desenvolvimento de algum tipo de jogo. De fato, a ideia de aplicar SMAs em jogos não é recente. Por exemplo, em [4] já é proposto o uso de um SMA para controlar um time de robôs em um campeonato de futebol de robôs. Alguns outros trabalhos existentes apresentam o uso de SMAs para jogos de tabuleiro [5, 6, 7, 2]. Nesses trabalhos muitas vezes são utilizados outros conceitos além de simplesmente agentes. Por exemplo, em [7] é utilizado também o conceito de negociação. Por outro lado, em [6] são empregadas ideias de redes de contrato e uma concepção simplificada de organização baseada em metas.

Outras categorias de jogos também são exploradas com a aplicação de SMA tais como jogos baseados em tempo real, realidade aumentada, cartas e jogos sociais. Em [3], o uso de SMA é combinado com realidade aumentada para a concepção de um jogo 3D com a finalidade de melhorar a função cognitiva de pacientes com desordens neuropsiquiátricas. Nesse trabalho, os agentes são usados para monitorar e gerenciar o desempenho do jogador a medida que ele navega no ambiente. Como conceitos adicionais também são utilizadas noções organizacionais como metas e tarefas. Em um segundo trabalho é ilustrado o uso de SMA em jogos de estratégia de tempo real [8]. Devido a se tratar de um jogo de guerra, técnicas de coordenação de agentes são empregadas. Também com foco em jogos de tempo real, um *framework* é concebido em [9] para estudar a integração de SMA e aplicações gráficas. O objetivo é controlar elementos em ambientes 3D, como jogos e sistemas de realidade

virtual. Seguindo a mesma ideia, em [10] também é proposto um *framework* para jogos usando conceitos de SMA. O objetivo deste último trabalho está em prover um *framework* para planejamento tático e estratégico em jogos baseados em equipes, onde duas ou mais equipes se enfrentam ou simplesmente há uma única equipe que deve realizar tarefas para atingir o objetivo do jogo.

Em [11] são empregadas técnicas de aprendizado por reforço em conjunto com SMA em jogos de carta. Por sua vez, em [12], um SMA é desenvolvido para trabalhar juntamente com os *bots* disponibilizados pelo jogo social FarmVille. O objetivo do SMA é permitir desenvolver uma estratégia para auxiliar o jogador no gerenciamento das diversas tarefas que devem ser realizadas, uma vez que os *bots* do FarmVille não permitem especificar estratégias de jogo.

Alguns livros também trazem a aplicação de outras técnicas de IA no desenvolvimento de jogos, tais como lógica *fuzzy*, redes neurais e bayesianas, algoritmos genéticos, algoritmos de busca, sistemas markovianos, aprendizado de máquina, *planning*, porém em relação aos agentes ainda há espaço para exploração, principalmente considerando conceitos de organização, ambiente, negociação, reputação etc [13, 14]. Por fim, outras propostas que utilizam SMA em jogos podem ser encontradas em [15, 16, 17, 18, 19, 20].

3 Definição do Problema

Titan é um jogo de estratégia onde as equipes disputam o domínio de Titan, um satélite que orbita o planeta Saturno. A história se passa no futuro, quando os recursos na Terra escassearam e os humanos buscam em outros lugares do universo os recursos que faltam. O principal objetivo do jogo é garantir o domínio sobre Titan através da destruição das equipes inimigas. Titan possui diversos poços de recursos que as equipes devem encontrar e explorar. É pela obtenção destes recursos que se torna possível ampliar os exércitos. Cada equipe é composta por um ou mais times aliados, que trocam informações sobre o ambiente e sobre os adversários. Por sua vez, um time é composto por uma base, um conselho, capitães, soldados, robôs, naves e exploradores. Além disso, cada time possui um *blackboard* próprio, que nada mais é que um repositório de informações globais usado com o objetivo de permitir que os agentes compartilhem informações com os demais agentes do time, e um sistema especialista (SE) próprio, para auxiliar nos planos de ataque.

Inicialmente, um time tem uma base, que é responsável pelo gerenciamento dos seus recursos e também pela criação de um conselho. Além disso, outra função da base é sempre manter a existência de um capitão para seu time. Em caso do capitão morrer em combate, a base deve criar um novo capitão. Já o conselho é um agente que tem como responsabilidade organizar a defesa e o ataque, obedecendo às estratégias definidas pelo capitão e negociadas com os aliados. No decorrer do jogo, guerreiros são criados de acordo com as necessidades

de ataque ou defesa de cada time, se a base possuir quantidade suficiente de recursos (maná) para criá-los. Os guerreiros sabem quem são seus aliados, movem-se pelo ambiente, atiram, reconhecem os inimigos, os poços e os obstáculos existentes. Existem cinco tipos de guerreiros: robô, nave, explorador, soldado e capitão. Cada guerreiro tem uma função diferente no cenário do jogo e também características distintas, conforme listado abaixo:

- Soldado: é o guerreiro responsável pela defesa da base e pelo cumprimento das missões de ataque;
- Robô: é um guerreiro que possui as mesmas responsabilidades do soldado. A vantagem de usar os robôs é o fato de que estes são mais fortes do que os soldados e, portanto, mais difíceis de serem destruídos;
- Nave: é um tipo de guerreiro que tem a capacidade de sobrevoar o ambiente. As naves têm um campo de visão maior do que os guerreiros terrestres e não são impedidas pelos obstáculos, podendo voar sobre os inimigos, bases e poços. Seu principal objetivo é explorar o ambiente, localizando as bases inimigas e poços;
- Explorador: é o guerreiro cuja função principal é extrair recursos dos poços encontrados no ambiente. Todo o recurso extraído dos poços é armazenado na base para ser usado quando for necessário criar um novo agente;
- Capitão: é o guerreiro que tem como responsabilidade planejar e negociar ataques juntamente com outros capitães aliados. O capitão tem acesso a uma base de conhecimento que permite a ele planejar os ataques e também pode, quando necessário, criar outros guerreiros.

O comportamento de cada guerreiro é definido por um conjunto de propriedades. Tais propriedades são apresentadas abaixo, enquanto que a parametrização dessas características para cada tipo de agente é mostrada na Tabela 1.

- `Defesa`: indica a capacidade de defesa que o agente tem. A defesa funciona como se fosse um escudo que amortece o ataque dos agentes adversários;
- `Ataque`: indica o quanto de vida que o ataque do agente tira do adversário;
- `Vida`: indica a quantidade de vida que o agente possui. Depois de um ataque, a vida restante do agente é determinada pela fórmula $vida = vida - (ataque - defesa)$;
- `Visão`: determina o raio da visão do agente. Quanto maior a visão, mais longe o agente enxerga;

Tabela 1. Propriedades de cada agente do time

	Robô	Nave	Soldado	Explorador	Capitão
Defesa	5	2	4	10	10
Ataque	15	12	13	-	15
Vida	200	100	150	400	400
Visão	30	60	30	30	40
Distância ataque	15	20	20	-	30
Velocidade	5	8	4	6	6
Custo	600	900	400	1500	3000

- `Distância de ataque`: determina a distância a partir da qual o agente pode atirar em outros agentes ou em uma base;
- `Velocidade`: determina a velocidade que o agente se desloca no ambiente;
- `Custo`: é a quantidade de recursos que serão utilizados para a criação deste agente.

Os guerreiros utilizam o *blackboard* para armazenar as informações sobre a localização dos poços, dos inimigos e dos aliados que são vistos no ambiente. Sempre que encontrar um poço ou um inimigo que ainda não tinha sido descoberto, os guerreiros disponibilizam essa informação no *blackboard* para o restante do time. Essas informações são usadas pelo capitão para, juntamente com o SE, decidir se um ataque deverá ou não ser realizado.

Diferentemente do que acontece com outros jogos desenvolvidos com agentes, Titan é um jogo simulado. Os times iniciam ataques, organizam defesas e buscam poços sem necessidade de intervenção de um jogador humano. A única interação humana no jogo acontece na configuração inicial do ambiente, onde o jogador define a quantidade de times e de equipes existentes e, se desejar, o SE utilizado por cada time. Depois de iniciado o jogo, as batalhas se desenrolam sem a necessidade de intervenção humana até que uma das equipes seja vencedora.

4 Desenvolvimento

Os agentes foram implementados utilizando diferentes arquiteturas. A base é um agente reativo implementado em JADE [21], enquanto que os guerreiros são agentes BDI implementados em Jason [22]. Outros conceitos da área de agentes foram utilizados no desenvolvimento do jogo. O capitão, por exemplo, tem acesso a um SE para decidir quando os ataques devem ser realizados e um *blackboard* é utilizado para auxiliar cada time a montar

uma visão global do ambiente. Os agentes de times aliados cooperam entre si para realizar ataques contra as bases inimigas, que são negociados pelos capitães de cada time. Então o capitão repassa as decisões da negociação ao conselho, que é responsável por organizar as unidades de ataque para que realizem o ataque. Ao mesmo tempo, o conselho cria unidades de defesa a fim de garantir a defesa da própria base. Assim, ao mesmo tempo em que existe uma hierarquia entre os agentes, muitas decisões também podem ser tomadas pelos guerreiros independentemente de ordens recebidas de um agente superior.

4.1 O ambiente

De forma geral, o ambiente é o espaço onde todos os agentes vivem, ou seja, onde percebem e atuam [23, 24]. O ambiente do jogo foi desenvolvido utilizando o CArtAgO [25, 26]. Todo o ambiente é implementado dentro de um único artefato e nele estão os times, os agentes, os poços, os obstáculos, os *blackboards* e os SEs. É neste artefato que está descrito o comportamento do ambiente, permitindo que os agentes realizem uma série de operações e percepções.

4.1.1 Operações Dentre as operações que podem ser realizadas durante o jogo aqui estão listadas as mais importantes. Todas as operações podem ter sucesso ou podem falhar dependendo dos parâmetros informados pelo agente e de suas capacidades. Por exemplo, um agente que não tem a capacidade de extrair recursos de um poço não poderá executar as operações `installAt`, `desinstallAt` ou `extractResource`. Um agente que não está próximo a outro agente não pode executar uma operação `attackAgent`. O mesmo vale para a operação `attackBase`, e assim por diante.

- `joinMars`: permite a entrada do agente no ambiente;
- `goTo`: determina a direção para a qual o agente deverá se mover;
- `installAt`: permite que o agente explorador se instale sobre um poço, afim de explorar seus recursos;
- `desinstallAt`: permite que o agente explorador libere o poço que está ocupando;
- `extractResource`: permite ao agente explorador que está instalado sobre um poço extrair os recursos deste poço;
- `attackAgent`: permite que o agente ataque outro agente;
- `attackBase`: permite que o agente ataque uma base;
- `decreaseVelocity`: diminui a velocidade de locomoção de um agente;
- `increaseVelocity`: aumenta a velocidade de locomoção de um agente.

4.1.2 Percepções O ambiente também emite sinais para informar os agentes quando alguns eventos ocorrem. Os principais sinais emitidos pelo ambiente são:

- `perceboAgente`: quando um agente `A` passa a ver um agente `B`;
- `perceboAgenteMira`: quando um agente `A` percebe um agente `B` em seu campo de ataque;
- `perceboPoco`: enviado quando um poço é visto por um agente;
- `sobrePoco`: quando um agente explorador estiver próximo ao poço o suficiente para se instalar e começar a extração dos recursos;
- `perceboBase`: quando um agente percebe uma base em seu campo de visão;
- perceboBaseMira: quando um agente percebe que uma base está em seu campo de ataque;
- `novoTurno`: o jogo acontece em turnos e em cada turno os agentes têm um limite de ações a executar. Este sinal indica o início de um novo turno, permitindo que os agentes voltem a executar ações no ambiente;
- `terminouJogo`: enviado quando um time vence o jogo;
- `baseDestruida`: todos os agentes recebem este sinal quando qualquer base é destruída;
- `agenteAtacado`: quando um agente é atacado ele recebe este sinal informando quanto de vida ainda lhe resta;
- `baseAtacada`: quando uma base é atacada todos os agentes daquele time recebem o sinal de base atacada.

4.2 A organização

Em uma organização há uma espécie de união entre os agentes que formam um grupo. A organização pode ser usada pra representar objetivos globais e os agentes passam a se relacionar para atingirem esses objetivos. Como metas locais e globais nem sempre são compatíveis, uma organização fornece uma visão geral para verificar se as ações dos agentes respeitam os objetivos globais ou não [27].

Os agentes que formam um time precisam trabalhar conjuntamente para que o objetivo de destruir o inimigo seja alcançado. Para garantir a organização desse sistema utilizou-se

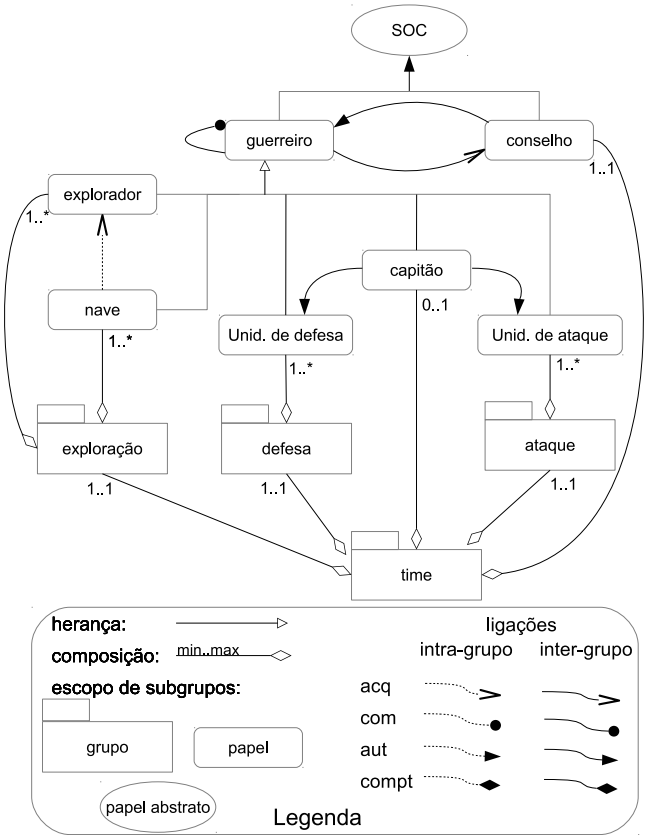

Figura 1. Especificação estrutural para o jogo `Titan`.

o modelo organizacional Moise [28, 29]. Conforme pode ser visto na Figura 1, na especificação estrutural foram especificados quatro grupos. O grupo `time` (grupo raiz) é composto pelos subgrupos `exploração`, `defesa` e `ataque`. Uma instância do grupo `time` deve possuir uma única instância do grupo `exploração`, uma do grupo `defesa` e uma do grupo `ataque`. Além disso, o grupo precisa ter um conselho mas não precisa necessariamente ter um capitão, já que, durante o jogo, o capitão pode ser morto e o grupo ficará sem capitão até que um novo seja criado. Os agentes do grupo `exploração` podem assumir o papel de nave ou o papel de explorador e a nave tem autoridade sobre o explorador. O grupo `defesa` é composto de zero ou infinitas unidades de defesa e o grupo `ataque` é composto de zero ou infinitas unidades de ataque.

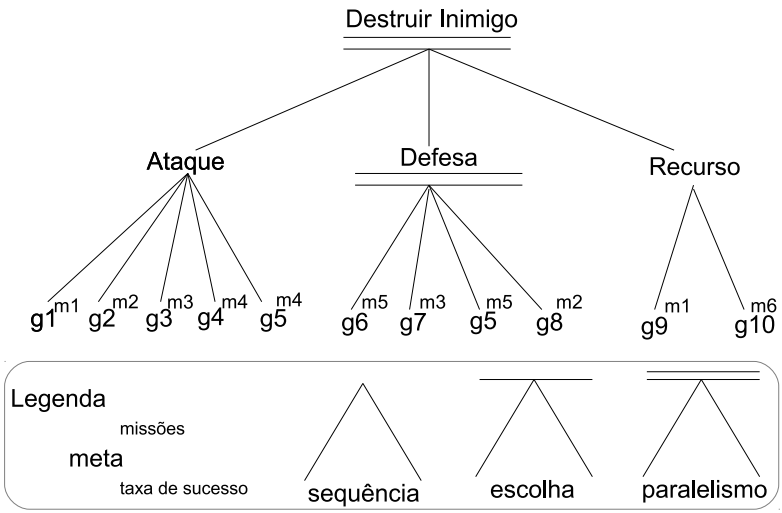

Figura 2. Especificação funcional para o jogo.

Os papéis de explorador, nave, unidade de defesa, unidade de ataque e capitão herdam as características do papel abstrato guerreiro e todos os guerreiros podem se comunicar entre si. Agentes do tipo robô e soldado podem assumir o papel de unidade de ataque ou unidade de defesa, de acordo com a necessidade do time. O papel assumido por um agente é definido pelo conselho no momento em que o agente é criado. Se uma unidade é criada e a defesa já está completa, a unidade assume o papel de ataque. Finalmente, o conselho tem autoridade sobre os guerreiros e estes têm conhecimento do conselho, enquanto que o capitão tem autoridade sobre as unidades de ataque e de defesa.

A Figura 2 apresenta um esquema social definindo a especificação funcional do jogo. Na raiz do esquema tem-se a meta global de destruir o inimigo. Para satisfazer a meta global, três submetas em paralelo precisam ser satisfeitas: a meta `ataque`, a meta `defesa` e a meta `recurso`. Essas três metas são decompostas em três planos, e esses planos são compostos por metas folhas. As metas dos planos `ataque` e `recurso` devem ser alcançadas em forma sequencial, ou seja, só é possível alcançar a meta `g2` depois que a meta `g1` for alcançada. Já as metas do plano `defesa` podem ser alcançadas de forma paralela, ou seja, pode-se alcançar a meta `g8` e `g6` ao mesmo tempo.

As missões são assumidas pelos agentes no momento em que estes assumem um papel em um grupo. Conforme especificado na Figura 2, o agente que se compromete com a missão `m1` é responsável pela satisfação de todas as metas desta missão, no caso, as metas `g1` e `g9`.

Tabela 2. Especificação normativa para o jogo

Papel	Relação deôntica	Missão
nave	obrigação	m1
capitão	obrigação	m2
conselho	obrigação	m3
unidade ataque	obrigação	m4
unidade defesa	obrigação	m5
explorador	obrigação	m6

Já o agente que assumir a missão `m2` deve alcançar as metas `g2` e `g8`.

A relação entre a especificação estrutural e especificação funcional é feita pela especificação normativa. Na especificação normativa são descritas as missões com as quais um papel tem permissão ou obrigação de se comprometer. Na Tabela 2 é possível observar que o papel de nave está obrigado a alcançar as metas da missão `m1`, o capitão tem obrigação de alcançar a missão `m2`, e assim por diante.

4.3 Comunicação entre agentes

Um dos aspectos mais importantes no desenvolvimento de um SMA é a interação entre os agentes. O principal mecanismo que permite esta interação é a comunicação através da troca de mensagens. Como sistemas multiagentes nem sempre são sistemas fechados, desenvolvidos por um único grupo de desenvolvedores, foi preciso criar protocolos padronizados objetivando que os agentes possam se comunicar com outros agentes mesmo que não tenham sido desenvolvidos especificamente para interagirem entre si. Algumas linguagens, como KQML, KIF e a FIPA-ACL, foram criadas com este fim. Neste trabalho utilizou-se duas dessas linguagens. A KQML é utilizada pela ferramenta Jason e o padrão FIPA-ACL é utilizado pela ferramenta JADE.

A *Knowledge Query and Manipulation Language* (KQML) é uma linguagem externa para a comunicação de agentes. Essa linguagem define um formato de envelope para as mensagens, no qual um agente pode explicitar a força das intenções ilocucionárias da mensagem. Cada mensagem tem uma performativa, que pode ser qualquer classe de mensagens, e um número de parâmetros, compostos por pares de atributos e valores [24].

Por sua vez, a *Foundation for Intelligent Physical Agents* (FIPA) desenvolveu a linguagem FIPA-ACL que é similar à linguagem KQML. A linguagem FIPA-ACL também define uma linguagem externa para as mensagens, porém diferentemente da KQML, a FIPA ACL possui somente 20 performativas para a interpretação das mensagens e não obriga uma

linguagem especifica para o conteúdo das mensagens [24].

Outra forma utilizada para comunicação entre os agentes é através do uso de um sistema de *blackboard*. Um sistema de *blackboard* é composto por uma coleção de entidades conhecidas como fontes de conhecimento, cada uma possuindo um conhecimento especializado, e uma estrutura de dados compartilhada que essas fontes de conhecimento utilizam para a comunicação. As fontes de conhecimento, neste caso os agentes, são capazes de ler e escrever no *blackboard*, assim contribuindo com uma solução parcial do problema [24].

No jogo, os agentes utilizam estas duas formas de comunicação: a troca de mensagens e o *blackboard*. A troca de mensagens é utilizada pelos agentes para comunicar o conselho quando nascem ou morrem, quando a base inimiga foi destruída e para incluir informações no *blackboard*. Além disso, os capitães usam mensagens para negociar os ataques com os aliados, que podem ser realizados com base nas informações do *blackboard* e com ajuda do SE. Dentre algumas operações que os agentes podem realizar no *blackboard* estão:

- `removeAgenteBB`: remove um agente do *blackboard*;
- `addAgenteBB`: adiciona um agente ao *blackboard*;
- `getQuantidade`: retorna uma mensagem contendo a quantidade de guerreiros conhecidos de determinado time;
- `setQuantidade`: atualiza a quantidade de guerreiros que um time possui;
- `getQuantidadeByTipo`: retorna uma mensagem contendo a quantidade de agentes de certo tipo que determinado time possui;
- `setQuantidadeByTipo`: atualiza a quantidade de guerreiros de determinado tipo que um time possui;
- `getAllCapitaesAliados`: retorna uma mensagem com dados a respeito de todos os capitães aliados conhecidos;
- `getCapitao`: retorna uma mensagem com informações a respeito do capitão de determinado time, quando este for conhecido;
- `addBase`: adiciona no *blackboard* uma nova base localizada;
- `addPoco`: adiciona um novo poço localizado ao *blackboard*, se o poço ainda não foi explorado;
- `removePoco`: remove um poço do *blackboard*. Esta operação é útil para ser usada quando o poço foi completamente explorado;

- `getAllBasesInimigas`: retorna uma mensagem com todas as bases inimigas já localizadas;

- `getLocalizacaoBase`: retorna uma mensagem contendo a localização da base de determinado time, quando esta for conhecida;

- `getPoco`: retorna as coordenadas de um poço ainda não explorado e que seja conhecido pelo time.

4.3.1 Negociação Negociação é uma técnica utilizada para que agentes possam chegar a um acordo sobre algum assunto de interesse mútuo. A negociação normalmente procede em uma série de etapas, com todos os agentes envolvidos fazendo uma proposta em cada etapa. As propostas dos agentes são feitas de acordo com a estratégia definida por eles, porém deve seguir um protocolo, ou seja, deve respeitar um conjunto de propostas possíveis em cada etapa. Se um acordo é alcançado então a negociação termina. Um exemplo de uma questão simples que envolve um cenário de negociação é onde dois agentes estariam negociando um preço de alguma coisa [24].

A fim de coordenar os diversos times aliados durante os ataques, o jogo utiliza uma espécie de negociação. A negociação é feita entre os capitães de times aliados. Um capitão só pode entrar em uma negociação se não estiver em outra negociação e se o time possuir unidades de ataque livres, ou seja, que não foram designadas para nenhuma missão. Antes de iniciar a negociação, o capitão precisa saber a quantidade de guerreiros conhecidos dos inimigos e quem são os comandantes aliados. De posse dessas informações o capitão inicia a negociação, que ocorre em quatro fases:

1. Início: o capitão informa aos aliados qual o inimigo que será atacado;

2. Inventário: levantamento da quantidade de guerreiros de todos os aliados que podem fazer parte do ataque;

3. Definição: consulta ao plano de ataque para saber se o ataque deve ser realizado ou não;

4. Ataque: envio de notificação às unidades para realizar o ataque.

A negociação é abortada em cada fase se algum aliado recusar a participar do ataque. Nesse caso, a negociação é reiniciada com os capitães que estão dispostos a colaborar. Por fim, sempre que a negociação for cancelada e reiniciada, todos os capitães participantes serão notificados.

4.3.2 Reputação dos agentes Bromley define reputação como sendo uma ferramenta social que objetiva reduzir a incerteza na interação entre indivíduos com atributos desconhecidos [30]. A interação entre os agentes é um aspecto inerente aos sistemas multiagentes, visto que os agentes precisam interagir para que os objetivos globais ou particulares possam ser atingidos. Assim, é comum que os sistemas multiagentes utilizem modelos de reputação para auxiliar os agentes na hora de determinar quem será seu parceiro de negócios. Nesse sentido, a reputação pode ser interpretada como um valor, determinado pela sociedade, que demonstra o grau de confiabilidade de um determinado agente.

Um sistema de reputação simples e centralizado foi implementado no jogo. O objetivo é permitir que times que se recusam a negociar com frequência sejam menos contatados. Um time tem um aumento na sua reputação se responder a uma negociação e puder ajudar o aliado. De forma análoga, um time perde pontos se não puder ajudar quando chamado. O time que inicia o processo de negociação também ganha pontos de reputação. Neste caso, o time recebe um ponto por cada um dos participantes da negociação. Quando a negociação for iniciada os capitães poderão escolher quais times irão chamar para fazer parte da negociação, de acordo com sua reputação. Times cuja reputação está baixa não são chamados, o que diminui as mensagens trocadas e torna a negociação mais rápida. Por fim, se o time inicia a negociação e no final desta ele decide não atacar, nenhum time irá ser pontuado.

A reputação no jogo foi integrada da seguinte maneira: quando um capitão deseja iniciar uma negociação ele pede para a base a lista de capitães aliados contabilizados no *blackboard*. A base faz a média da reputação de todos os times aliados, inclusive do próprio capitão que iniciou a negociação. Como em outros sistemas de reputação, somente as últimas 15 interações são consideradas pela base. A seguir, a base estipula uma linha de corte, para não considerar os capitães de times com baixa reputação. A linha de corte é estabelecida como sendo 20% da média das reputações dos times. Por fim, a base devolve todos os capitães pertencentes a times com reputação $>= 20$ da média.

Durante a negociação, todos os times que foram convocados a participar e desistiram recebem uma nota de penalidade em sua reputação (-1). Todos os times que chegaram ao final da negociação e aceitaram efetuar o ataque recebem uma avaliação positiva $(+1)$. Tanto a avaliação positiva quanto a avaliação negativa são dadas pelo capitão que iniciou a negociação. O capitão que iniciou a negociação também é avaliado pelos demais capitães participantes. A avaliação deste capitão se dá ao final da negociação. Se ao final da negociação for emitida uma ordem de ataque e todos aceitarem ir é sinal que a negociação ocorreu sem problemas, portanto todos os demais capitães participantes avaliam positivamente o capitão que teve a iniciativa de conduzir o ataque $(+1)$.

Com o uso da reputação foi possível reduzir a quantidade de trocas de mensagens necessárias e consequentemente o tempo demandado para uma negociação. A quantidade de trocas de mensagens é um critério interessante em SMA, pois um mesmo SMA poderia

estar distribuído em diversas máquinas resultando em um impacto ainda maior no tempo de execução.

4.4 Sistema Especialista

Um SE é um sistema que possui a capacidade de emular a habilidade de decisão de um especialista humano sobre certo domínio de conhecimento. Um exemplo clássico de um SE é o MYCIN [31, 32], que pretendia dar assistência no tratamento de infecções sanguíneas em humanos [33].

No jogo, o capitão tem acesso a um SE que o auxilia a definir se um ataque a determinada base inimiga deve ou não ser realizado em determinado momento do jogo. Para isso é levado em conta uma série de informações tais como a quantidade de guerreiros disponíveis, a quantidade de guerreiros que o inimigo possui recursos, entre outros dados relevantes sobre os times. Essas informações são obtidas tanto através de consultas ao *blackboard*, como diretamente informadas pelo capitão do respectivo time. Cada time possui um SE próprio,o que diferencia o comportamento dos capitães durante a negociação. Para que isso seja possível, a programação do SE é feita em arquivos separados e cada time tem o seu próprio arquivo com a descrição do seu SE, cujas regras são especificadas por um humano e os fatos são obtidos durante a execução do jogo. Para descrever o SE utilizou-se a ferramenta Jess [34], enquanto que para um capitão obter a resposta do SE ele utiliza uma ação do ambiente chamada `respostaSistemaEspecialistaAtaque`.

Um exemplo de um possível SE é ilustrado abaixo. Nele, o time que utiliza o SE responde positivamente para uma chamada de ataque toda vez que a quantidade total de unidades suas mais as unidades dos aliados supera a quantidade de unidades do inimigo em mais de duas unidades (linhas 6 e 7). As linhas 3 e 4 ilustram dois exemplos de fatos que são utilizados no SE (`totalUnidadesAliadas` e `unidadesInimigoPiorCaso`), enquanto que a linha 14 permite a obtenção da resposta do SE para ser utilizada pelo agente capitão. Como possíveis respostas estão `atacar` ou `esperar` (linhas 8 e 10, respectivamente). Note que este exemplo é bastante simplificado e é possível criar SEs mais complexos de acordo com a necessidade de cada time.

```
1.  (clear)(reset)
2.  (defrule testaAtaque
3.     (totalUnidadesAliadas ?aliados)
4.     (unidadesInimigoPiorCaso ?inimigo)
5.  =>
6.     (bind ?dif (- ?aliados ?inimigo))
7.     (if (> ?dif 2) then
8.         (bind ?*saida* "atacar")
```

```
9.      else
10.         (bind ?*saida* "esperar")
11.     )
12. )
13.
14. (deffunction retornaRespostaAtacar()
15.     (return ?*saida*)
16. )
```

4.5 Ontologia

Uma ontologia descreve os conceitos de um domínio de conhecimento e define relações entre eles. O principal objetivo da ontologia é capturar o conhecimento consensual sobre algum domínio de interesse. Studer define ontologia como "uma especificação explícita e formal de uma conceitualização compartilhada". Ela deve ser compreensível, acessível e manipulável pelos agentes que farão uso da mesma [35].

No jogo, a ontologia foi utilizada pelo agente base para que este soubesse qual papel organizacional cada um dos outros agentes pode assumir. Por exemplo, o papel de capitão só pode ser assumido pelo agente do tipo capitão, o papel de explorador só pode ser assumido pelo agente do tipo explorador ou nave. Esta relação entre requisitos necessários e papel não pode ser feita a partir do Moise e é importante para o jogo pois um agente com as características de capitão não possui os requisitos necessários para assumir o papel de explorador. Por esta razão viu-se a possibilidade do uso da ontologia de forma conjunta à organização criada no Moise, sendo os papéis existentes na organização também descritos por uma ontologia.

Para o desenvolvimento da ontologia foi utilizada a ferramenta Protégé [36], e a integração com o sistema foi feita através do uso do *reasoner* Hermit [37], que tem a finalidade de efetuar o raciocínio da ontologia. Por meio do mecanismo de inferência disponibilizado pelo Hermit foi possível inferir quais tipos de agentes podem assumir cada papel. Para que seja possível trabalhar com este recurso foi elaborada uma ontologia que detalha os equipamentos e títulos que um agente deve possuir para que possa assumir determinado papel. A Figura 3 exibe a classe `Equipamento`, que especifica a lista de equipamentos disponíveis no jogo. Cada agente pode possuir desde zero ou até vários destes equipamentos. Já a Figura 4 permite visualizar a classe `Titulo`, que contém a lista de títulos que uma unidade pode obter. O título é usado para permitir que um agente possa assumir os papéis de `Capitão` ou `Conselho`, que não implicam na necessidade de um agente possuir algum equipamento específico.

Após a elaboração destas classes foi necessário relacionar elas por meio de propriedades. Como exemplo, um papel de ataque só pode ser assumido por um agente que

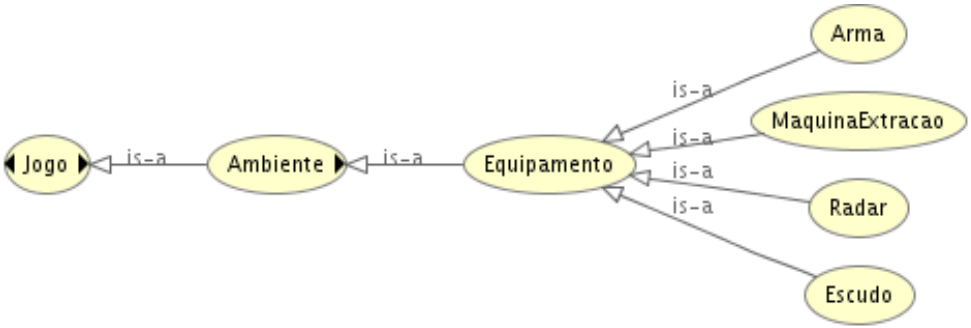

Figura 3. Classe `Equipamento`.

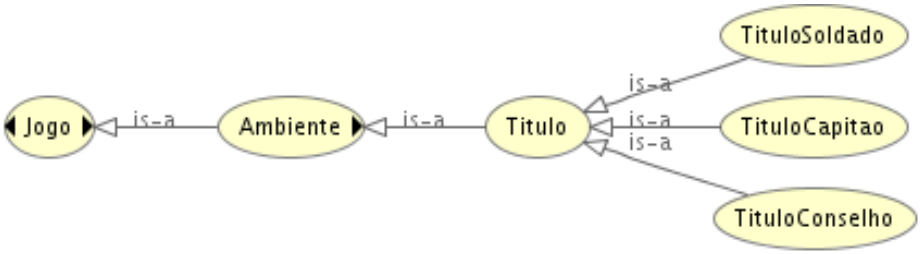

Figura 4. Classe `Título`.

possua uma arma de equipamento, um papel de exploração só pode ser assumido por um agente que possua um radar ou uma máquina de extração de recursos, um papel de capitão só pode ser assumido caso o agente possua um título de capitão, e assim por diante. O diagrama de todas as inferências obtidas pelo *reasoner* Hermit entre papéis e agentes é mostrado na Figura 5.

Na ontologia também foram implementadas a descrição dos artefatos contidos na classe `Artefato` e a relação de acesso a estes artefatos, ou seja, a definição de quais agentes possuem acesso um determinado artefato. A classe `Artefato` tem como objetivo fazer uma separação funcional do ambiente desenvolvido no CArtAgO, ilustrando que tipos de artefatos existem no jogo, visto que na implementação foi optado por utilizar um único artefato que controla todo o ambiente do jogo.

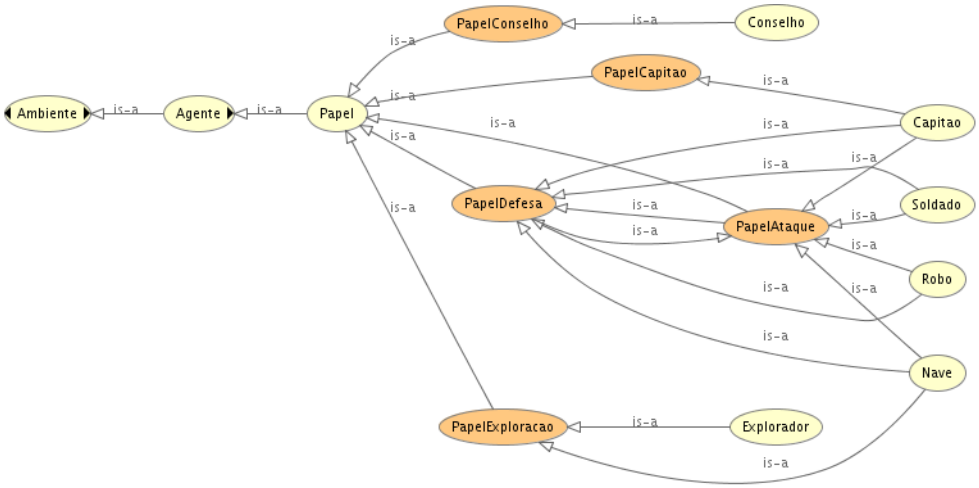

Figura 5. Ontologia inferida da classe `Agente`.

4.6 Interface Gráfica

Com a finalidade de visualizar o comportamento do jogo optou-se por desenvolver uma interface gráfica simples. Nesta interface, cada time é representado por uma cor. A Figura 6 mostra a descrição dos times que estão no jogo. Neste caso são quatro times: time 0, de cor azul, time 1, de cor verde, time 2, de cor vermelha e time 3, de cor cinza. Para cada time é apresentada a quantidade total de unidades, a distribuição dessas unidades por papéis, a lista de aliados, se houver, e a quantidade de recursos que o time possui (mana). O time 0 (azul), por exemplo, não possui aliados. Já o time 1 (verde) possui dois aliados: o time 2 (vermelho) e o time 3 (cinza).

Na Figura 7 pode-se visualizar o ataque coordenado dos times verde e cinza contra a base do time azul. Nesta representação gráfica, os soldados e robôs de ambos os times aliados atiram contra a base. Os tiros são representados por linhas amarelas. As letras `r` representam robôs, os `s` representam soldados, o `c` representa o capitão, o círculo maior representa a base e o asterisco circundado representa um poço. Poços ainda não explorados são representados em amarelo, poços já explorados são representados na cor preta.

Por fim, cada agente tem o seu próprio campo de visão (Figura 8), que é diferente para cada tipo de agente. Visualmente, o campo de visão é representado por um aro que circunda o agente. É através desse campo de visão que os agentes conseguem perceber os elementos do ambiente dentro de uma certa distância de onde ele está. Nesta figura pode-se ver também

```
Id: 0 Unidades: 1 Mana: 200 Aliados: null
PapelConselho(0)\PapelDefesa(1)\PapelExploracao(0)\PapelCapitao(1)\PapelAtaque(1)
Id: 1 Unidades: 19 Mana: 14300 Aliados: 2 / 3
PapelConselho(0)\PapelDefesa(18)\PapelExploracao(1)\PapelCapitao(1)\PapelAtaque(18)
Id: 2 Unidades: 11 Mana: 460 Aliados: 1 / 3
PapelConselho(0)\PapelDefesa(11)\PapelExploracao(0)\PapelCapitao(1)\PapelAtaque(11)
Id: 3 Unidades: 20 Mana: 17420 Aliados: 1 / 2
PapelConselho(0)\PapelDefesa(19)\PapelExploracao(2)\PapelCapitao(1)\PapelAtaque(19)
```

Figura 6. Descrição dos times.

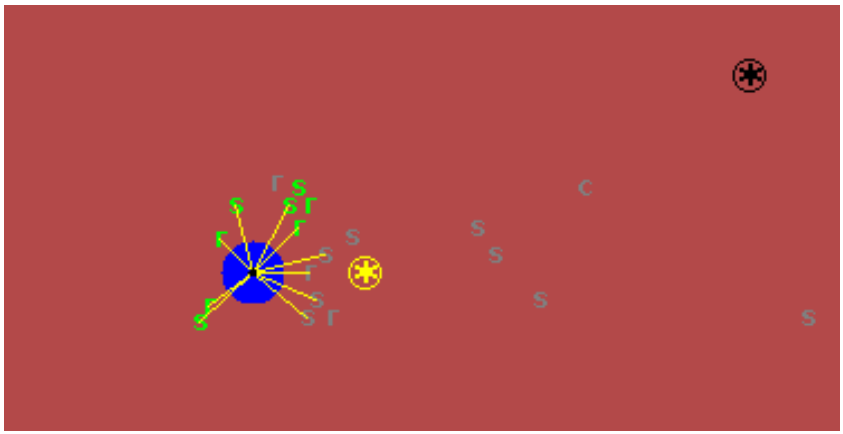

Figura 7. Visualização do jogo.

um explorador, representado por um pequeno círculo preenchido, extraindo recursos do poço, e o capitão do time cinza atacando um robô do time azul.

5 Desafios durante o desenvolvimento

Durante o desenvolvimento do jogo alguns problemas foram enfrentados. Por exemplo, quando diversos agentes precisavam acessar o ambiente desenvolvido na ferramenta CArtAgO e os acessos eram constantes, o ambiente tornava-se lento e acabava por prejudicar a execução do jogo. Para corrigir o problema foi necessária uma alteração na ferramenta. No caso da interface gráfica desenvolvida em Java, a lentidão foi corrigida com a criação de uma thread especifica para atualizar da interface. Também visando um melhor desempenho do

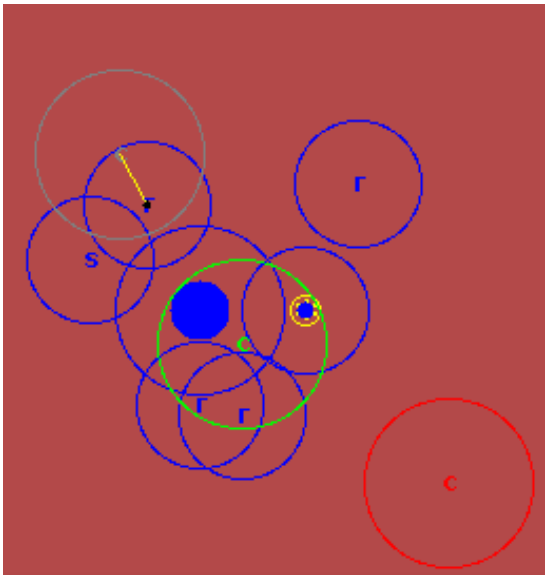

Figura 8. Visualização do jogo mostrando o campo visual de cada agente.

jogo optou-se por desenvolver um único artefato contendo todas as operações que os agentes podem executar no ambiente, ao invés de diversos artefatos.

A ferramenta JADE também apresentou alguns problemas. Em primeiro lugar, descobriu-se que o JADE não interage com o ambiente CArtAgO. Essa limitação teve de ser contornada com a criação de uma representação do agente JADE no ambiente do CArtAgO. Essa foi a maneira encontrada para que um agente desenvolvido em JADE pudesse interagir com o ambiente. Mesmo assim, o agente JADE tem acesso somente a algumas percepções e ações. Essa restrição precisou ser colocada, pois este agente não tem mecanismos de controle de concorrência. Nos demais agentes o acesso concorrente é controlado pelo CArtAgO. O agente JADE foi concebido como um agente reativo, já que a linguagem não oferece funcionalidades para criação de agentes BDI. A principal função deste agente é gerenciar o *blackboard*, ou seja, para atualizar ou consultar o *blackboard* é necessário enviar mensagens para o agente JADE. Este procedimento visa evitar que uma quantidade grande de acessos ao ambiente seja feita, evitando que este se tornasse um gargalo para o sistema. Outra limitação apresentada foi com relação a quando uma quantidade grande de mensagens era trocada. Neste caso, a execução se tornava lenta. Por isso acabou-se por reduzir o número de mensagens trocadas entre os agentes.

Por fim, o Moise apresentou problemas quando um agente era removido do sistema. Como o agente tem obrigação de realizar uma missão, ele não pode abandoná-la. Mesmo quando um agente é removido do sistema através da ação interna `kill_agent`, o Moise não libera o papel para outro agente. Para solucionar esse problema, foi feita uma alteração na especificação do esquema do Moise para permitir que os papéis tivessem um número indefinido de agentes, evitando alterar o código do próprio Moise devido a limitação de tempo para a construção do jogo. O Moise também apresentou limitações com metas do tipo *maintenance*, que são metas que precisam ser alcançadas continuamente, e metas que precisam ser alcançadas diversas vezes pelo mesmo agente. Esses tipos de metas não são disponibilizadas pelo Moise, porém uma possível ideia de como utilizá-las é apresentada em [38].

6 Contribuições do trabalho

Devido as limitações citadas acima, foi possível realizar algumas colaborações para as ferramentas utilizadas. No projeto CArtAgO foi corrigido e reportado o problema que havia em relação a lista de eventos pendentes dos agentes, na qual estes eventos nunca eram apagados, causando certa instabilidade no jogo. Tal correção já está disponibilizada nas versões atuais do CArtAgO.

Algumas contribuições também foram dadas ao projeto Moise, como a sugestão do desenvolvimento de metas do tipo *maintenance*. Esse tipo de meta deve ser executada pelo agente repetidas vezes até que este abandone seu papel. É uma meta que, na prática, nunca entra em um estado de satisfeita, mantendo-se sempre ativa. Outras colaborações para o projeto Moise foram reportar os problemas que ocorriam quando um agente abandonava o sistema, considerando principalmente o fato deste agente estar comprometido com uma meta obrigatória. Neste caso, não era possível um agente que está comprometido com certa meta abandoná-la, mesmo que o agente saia do sistema. Quanto aos problemas encontrados durante o uso da ferramenta JADE, descobriu-se que os desenvolvedores já têm conhecimento dessas falhas.

7 Considerações finais e trabalhos futuros

Com o desenvolvimento do jogo foi possível notar o impacto do uso de cada conceito e ferramenta utilizados. Como cada time pode utilizar um SE próprio e diferente dos demais, o sistema permitiu times com comportamentos estratégicos distintos, tornando o jogo mais interessante. O uso do ambiente CArtAgO mostrou-se de grande ajuda pois, através da simulação de um ambiente virtual por meio de artefatos, foi permitido separar de maneira coerente o desenvolvimento dos agentes dos demais objetos envolvidos no sistema, que não possuem autonomia. Já o Jason demonstrou ser uma linguagem adequada para o desenvolvimento de

agentes BDI, permitindo a criação destes tipos de agentes de maneira bastante simples e intuitiva. Por outro lado, o JADE não se mostrou tão adequado para a implementação destes tipos de agentes. Por exemplo, o JADE não trabalha com a questão de crenças, uma vez que, por natureza, ele não possui o foco na ideia de agentes BDI, mas sim na disponibilização de uma plataforma de comunicação para agentes. Tal mecanismo de crenças deve ser implementado pelo próprio programador. Quanto a dimensão da organização pôde ser aproveitada a ferramenta Moise, que apesar de alguns recursos ainda não estarem disponíveis, mostrou ser útil para controlar um grupo de agentes, levando-os a cumprir as metas de uma forma mais organizada e civilizada.

A ontologia permitiu o uso de um *reasoner* para mostrar ao usuário quais são os papéis disponíveis pela organização, porém também informar quantas unidades poderia assumir cada papel em tempo de execução. Dessa forma foi permitido ao usuário ter uma noção do que se passa durante o jogo em termos de números de unidades. A reputação também teve um papel importante durante a negociação, reduzindo significativamente a troca de mensagens entre capitães pertencentes a times que não estavam colaborando nos combates.

Finalmente, assim como os resultados conseguidos pelos trabalhos relacionados (seção 2), neste trabalho também foi possível observar um comportamento mais credível devido às técnicas de IA utilizadas. Por exemplo, o uso de um SE específico para cada time e sistemas de reputação, ambos trabalhando em conjunto com um SMA, permitiram configurar a negociação de forma mais próxima com a realidade humana tornando os ataques coordenados mais realistas.

Como trabalhos futuros espera-se melhorar o jogo desde a implementação de novos recursos, como também verificar questões de desempenho e escalabilidade. Dentre algumas das ideias estão tornar a execução do jogo distribuída, utilizar outras técnicas de inteligência artificial, incluir novos tipos de agentes e artefatos, e permitir que o usuário possa interagir com o jogo controlando algum dos times, ou seja, que o usuário pudesse jogar contra os agentes.

8 Agradecimentos

Agradecemos aos professores Jomi Fred Hübner (UFSC) e Ricardo José Rabelo (UFSC) por comentários e sugestões a respeito do trabalho.

Referências

[1] Felipe Carvalho. Comportamento em grupo de personagens do tipo black&white, 2004.

[2] Stefan J. Johansson. On using multi-agent systems in playing board games. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, AAMAS '06, pages 569–576, New York, NY, USA, 2006. ACM.

[3] Priscilla F. de Abreu, Vera Maria B. Werneck, Rosa Maria E. Moreira da Costa, and Luis Alfredo V. de Carvalho. Employing multi-agents in 3-d game for cognitive stimulation. In *Proceedings of the 2011 XIII Symposium on Virtual Reality*, SVR '11, pages 73–78, Washington, DC, USA, 2011. IEEE Computer Society.

[4] Sung-Wook Park, Jung-Han Kim, Eun-Hee Kim, and Jun-Ho Oh. Development of a multi-agent system for robot soccer game. In *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, volume 1, pages 626 –631 vol.1, apr 1997.

[5] Henric Fransson. Agentchess - an agent chess approach, 2003.

[6] Fredrik Haard. Multi-agent diplomacy, 2004.

[7] Fredrik Olsson. A multi-agent system for playing the board game risk, 2005.

[8] Johan Hagelbäck and Stefan J. Johansson. Using multi-agent potential fields in real-time strategy games. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems - Volume 2*, AAMAS '08, pages 631–638, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems.

[9] Antonio Barella, Carlos Carrascosa, and Vicente Botti. Jgomas: game-oriented multi-agent system based on jade. In *Proceedings of the 2006 ACM SIGCHI international conference on Advances in computer entertainment technology*, ACE '06, New York, NY, USA, 2006. ACM.

[10] Chek Tien Tan and Ho-lun Cheng. A combined tactical and strategic hierarchical learning framework in multi-agent games. In *Proceedings of the 2008 ACM SIGGRAPH symposium on Video games*, Sandbox '08, pages 115–122, New York, NY, USA, 2008. ACM.

[11] H. Fujita, Y. Matsuno, and S. Ishii. A reinforcement learning scheme for a multi-agent card game. In *Systems, Man and Cybernetics, 2003. IEEE International Conference on*, volume 5, pages 4071 – 4078 vol.5, oct. 2003.

[12] R. Neves, L.P. Reis, P. Abreu, and B.M. Faria. A multi-agent system to help farmville players on game management tasks. In *Information Systems and Technologies (CISTI), 2012 7th Iberian Conference on*, pages 1 –6, june 2012.

[13] David M. Bourg and Glenn Seeman. *AI for game developers*. O'Reilly, 2004.

[14] Ian Millington and John Funge. *Artificial intelligence for games*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edition, 2009.

[15] David Vickrey and Daphne Koller. Multi-agent algorithms for solving graphical games. In *Eighteenth national conference on Artificial intelligence*, pages 345–351, Menlo Park, CA, USA, 2002. American Association for Artificial Intelligence.

[16] Zhisheng Huang, Anton Eliëns, and Cees Visser. 3d agent-based virtual communities. In *Proceedings of the seventh international conference on 3D Web technology*, Web3D '02, pages 137–143, New York, NY, USA, 2002. ACM.

[17] Tingting Wang and Jiming Liu. Evaluating the minority game strategy in agent role assignments. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, AAMAS '05, pages 1347–1348, New York, NY, USA, 2005. ACM.

[18] P. J. 't Hoen, S. M. Bohte, and J. A. La Poutré. Learning from induced changes in opponent (re)actions in multi-agent games. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, AAMAS '06, pages 728–735, New York, NY, USA, 2006. ACM.

[19] Z. Kobti and S. Sharma. A multi-agent architecture for game playing. In *Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on*, pages 276 –281, april 2007.

[20] Eric Raboin, Ugur Kuter, and Dana Nau. Generating strategies for multi-agent pursuit-evasion games in partially observable euclidean space. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 3*, AAMAS '12, pages 1201–1202, Richland, SC, 2012. International Foundation for Autonomous Agents and Multiagent Systems.

[21] Italia Telecom. Jade - java agent development framework. Disponível em `http://jade.tilab.com/`, 2011.

[22] Rafael H. Bordini, Jomi F. Hübner, and Michael Wooldridge. *Programming Multi-Agent Systems in AgentSpeak using Jason (Wiley Series in Agent Technology)*. John Wiley & Sons, 2007.

[23] Yves Demazeau. From interactions to collective behaviour in agent-based systems. In *In: Proceedings of the 1st. European Conference on Cognitive Science. Saint-Malo*, pages 117–132, 1995.

[24] Michael Wooldridge. *An Introduction to MultiAgent Systems*. Wiley, 2nd edition, July 2009.

[25] A. Ricci, M. Viroli, and A. Omicini. CArtAgO: An infrastructure for engineering computational environments in MAS. In Danny Weyns, H. Van Dyke Parunak, and Fabien Michel, editors, *3rd International Workshop "Environments for Multi-Agent Systems" (E4MAS 2006)*, pages 102–119, AAMAS 2006, Hakodate, Japan, 2006.

[26] A. Ricci, M. Piunti, M. Viroli, and A. Omicini. Environment programming in cartago. In *Multi-Agent Programming II: Languages, Platforms and Applications, Multiagent Systems, Artificial Societies, and Simulated Organizations*. Springer, 2009.

[27] F. Gers. *Multi-agent system for distributed data fusion in peer-to-peer environment.* PhD thesis, University of Jyväskylä, 2002.

[28] M. Hannoun, O. Boissier, J. S. Sichman, and C. Sayettat. Moise: Un modèle organisationnel pour la conception de systèmes multi-agents. In *Acts des 7èmes Journèes Francophones Intelligence Artificielle Distribuee & Systèmes Multi-Agents*, pages 105–118. Hermès Science Publications, 1999.

[29] Jomi Fred Hubner. *Um Modelo de Reorganização de Sistemas Multiagentes*. PhD thesis, Escola Politécnica da Universidade de São Paulo, 2003.

[30] D. B. Bromley. *Reputation, Image and Impression Management*. John Wiley & Sons, April 1993.

[31] E.H. Shortliffe. *Computer-based medical consultations: MYCIN*, volume 388. Elsevier New York, 1976.

[32] B. G. Buchanan and E. H. Shortliffe. *Rule Based Expert Systems: The Mycin Experiments of the Stanford Heuristic Programming Project*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1984.

[33] P. Jackson. *Introduction to Expert Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3rd edition, 1998.

[34] Ernest Friedman-Hill. Jess, the rule engine for the java platform. Disponível em `http://www.jessrules.com/`, 2011.

[35] Rudi Studer, V. Richard Benjamins, and Dieter Fensel. Knowledge engineering: Principles and methods. *Data and Knowledge Engineering*, 25(1-2):161–197, March 1998.

[36] Stanford Center For Biomedical Informatics Research. Protégé. Disponível em `http://protege.stanford.edu/`, 2011.

[37] Information Systems Group. Hermit owl reasoner. Disponível em `http://hermit-reasoner.com/`, 2011.

[38] Jomi F. Hübner, Rafael H. Bordini, and Michael Wooldridge. Programming declarative goals using plan patterns. In *Proceedings of the 4th international conference on Declarative Agent Languages and Technologies*, DALT'06, pages 123–140, Berlin, Heidelberg, 2006. Springer-Verlag.