Sistema Multiagente de Veículos Autônomos em uma Cidade Inteligente aplicado ao cenário da *Multi-Agent Programming Contest*

Giovanni Jakubiak de Albuquerque[1], Tiago Funk[1], Vilson de Deus Corrêa Júnior[1], Tiago Luiz Schmitz[1]

[1]Departamento de Engenharia de Software – UDESC – CEAVI

giovannijakubiak@hotmail.com, tiagoff.tf@gmail.com,
vilsonjrcorrea@gmail.com, tiago.schmitz@udesc.br

Resumo. *Neste artigo vamos apresentar uma arquitetura de agentes para veículos autônomos em cidades inteligentes, aplicados ao cenário da competição multi-agent programming contest. O artigo aponta as ferramentas e estratégias utilizadas pela equipe na competição.*

Palavras-chave: Cidades Inteligentes; Sistema Multiagente, Veículos Autônomos;

Abstract. *In this article we present an architecture of agents for autonomous vehicles in intelligent cities, applied to the scenario of the competition multi-agent programming contest. The article points out the tools and strategies used by the team in the competition.*

Keywords: Intelligent City; Multi-Agent System; Autonomous Vehicles;

1. Introdução

Devido ao avanço da tecnologia e a sua facilidade de acesso, ao caminhar em qualquer cidade percebemos a predominância de sistemas de informação. Os quais estão no bolso dos cidadãos em seus smartphones, nos semáforos em sistemas que otimizam o trânsito e até nas vagas de carros. As informações sobre a cidade são gerenciadas e consumidas ao ponto de algumas localidades já serem chamadas de cidades inteligentes. O portal da Fapesp conceitua cidade inteligente como uma construção evolutiva em um processo que envolve a constante busca em resolver problemas por meio de soluções disruptivas. As cidades inteligentes se integram com sensores, dispositivos eletrônicos e redes de comunicação, os quais são ligados com sistemas computadorizados, para análise de dados a partir de algoritmos inteligentes que tomam decisões**.**

Neste contexto podemos citar os veículos autônomos que percebem o ambiente por meio de sensores, planejam suas ações com base na percepção e atuam das mais diversas formas dependendo da sua natureza. Tendo inúmeras aplicações benéficas à sociedade, como por exemplo, caminhões de lixo autônomos ou Drones que fiscalizam fronteiras. Em uma cidade inteligente os veículos autônomos trocarão informações com os outros sistemas existentes visando sempre otimizar o uso de energia e de outros recursos.

O comportamento de um veículo autônomo é o comportamento padrão de um agente, que quando integrados a outros sistemas existentes na cidade obtém-se um sistema multiagente (seção 2). Sistemas multiagentes são caracterizados como um grupo de agentes que atuam em conjunto no sentido de resolver problemas que estão além das suas habilidades individuais. (GIRARDI, 2004)

Podemos aplicar os conceitos de sistemas multiagentes de veículos autônomos em uma cidade inteligente no cenário proposto pela *Multi-Agent Programming Contest*

(MAPC) (seção 3) da *Claustral University of Tecnology*, que existe desde 2005 e propõe um cenário no qual agentes devem compreender, raciocinar e tomar decisões de forma autônoma. Conforme Ahlbrecht (2018), a história de fundo da competição conta que no futuro, numa sociedade pós-apocalíptica, os recursos (itens e água) ficaram escassos, desta forma a equipe deve se organizar para obter esses recursos. Em suma consiste em duas equipes de vários agentes, cada um se movendo pelas ruas de uma cidade realista, com o objetivo de construir poços de água, coletar, montar e entregar itens.

A complexidade do cenário simula a realidade em diversos aspectos e envolve diversas variáveis que devemos levar em consideração para construir nossas estratégias (seção 6). Dentre as estratégias desenvolvidas pela equipe cabe destacar a de recarga dos agentes, produção de itens e o atendimento da demanda de trabalhos, que foram implementados de forma modular (seção 5), assim cada tarefa deve ser programada seguindo um padrão para ser facilmente inserida no sistema e utilizada pelos agentes.

O objetivo deste artigo é explorar uma arquitetura de agentes para veículos autônomos através do cenário da MAPC. Para apresentar a arquitetura este artigo descreve como as estratégias empregadas pela equipe para coordenar e construir um comportamento social entre os agentes é capaz de completar os objetivos propostos.

2. Sistemas Multiagente

Conforme Bordini e Vieira (2003), a área de sistemas multi-agente ocupa-se da construção de sistemas computacionais a partir da criação de entidades de software autônomas, denominadas agentes, sendo implementadas utilizando o paradigma de programação orientado a agentes que, segundo Wooldridge (2001), a ideia chave é programar diretamente agentes em termos de noções mentais (como crença, desejo ou objetivo e intenção), que ainda conforme Bordini e Vieira (2003) tiveram sua inspiração em disciplinas além da ciência da computação, tais como: psicologia, ciência cognitiva, sociologia, entomologia, economia, teoria das organizações, teoria dramática, antropologia, entre outras.

Agentes deliberam sobre que objetivos vão atingir, baseados em suas próprias motivações, e com base nesses objetivos e observações do ambiente decidem que ações tomar, e que requisições feitas por outros agentes aceitar (BORDINI; VIEIRA, 2003).

3. Cenário da Multi-Agent Programming Contest

Organizada pela Clausthal University of Technology a Multi-Agent Programing Contest (MAPC) existe desde de 2005 e propõe um cenário no qual agentes devem compreender, raciocinar e tomar decisões de forma autônoma.

A história do cenário ocorre no ano de 2045dC, com uma grave crise de água que atingiu Marte, colonizada pelos humanos tempos atrás, e derrubou o sistema monetário. Desta forma o imperador mundial foi forçado a intervir, oferecendo uma recompensa notável para o grupo que construir os maiores poços de água para abastecer a humanidade. Além disso para melhorar a vida no novo planeta, utilizam-se dos chamados "*All Terrain Planetary Vehicles"*, que são veículos autônomos como carros, drones, motos, e caminhões, utilizados para coletar os recursos naturais existentes no planeta e fornecê-los para a população (AHLBRECHT, 2018).

Os poços são construídos utilizando o *massium* que é a moeda do jogo, obtida como recompensa pela realização de trabalhos que envolvem a entrega dos itens coletados e montados. A partida consiste em 3 três rounds, sendo executados cada um em um mapa diferente. Cada round tem uma duração de 1000 *steps*. *Steps* são a medida de tempo do jogo.

Cada agente tem seu papel (drone, moto, carro e caminhão) que é atribuído dinamicamente no início da partida. Cada papel tem suas peculiaridades: bateria (quanto tempo ele pode se mover sem recarregar), capacidade de carga (ou seja, quanto volume ele pode carregar), velocidade, visão (quão longe ele pode perceber) e habilidade (quão rápido ele pode completar certas tarefas), classificados por aumento de capacidade de carga e energia, e velocidade decrescente conforme o tamanho do agente (AHLBRECHT, 2018).

O mapa da cidade é retirado do OpenStreetMap e o roteamento é fornecido pelo servidor do concurso. Cada simulação apresenta um conjunto de itens diferentes com volumes e modo de obtenção específicos. No início da partida os agentes são posicionados aleatoriamente no mapa, assim como as várias instalações, entre elas lojas, estações de recarga, oficinas, nós de recursos, instalações de armazenamento e lixões (AHLBRECHT, 2018).

4. Ferramentas Utilizadas

O JaCaMo é um framework para Programação Multiagentes que combina três tecnologias: **Ja**son (linguagem BDI para programação agentes), **Ca**rtago (implementa o conceito de agentes e artefatos para programar o ambiente do sistema) e **Mo**ise (Modelo organizacional para sistemas multiagentes que define e representa organizações multiagentes). Cobrindo assim todos os níveis de abstrações que são necessários para o desenvolvimento de sistemas multiagentes sofisticados (HÜBNER et al, [2018], online).

No nosso projeto não fizemos o uso do Moise, desta forma utilizaremos a abordagem Agentes e artefatos (**JaCa)**, que utiliza as outras duas ferramentas: Jason e Cartago. Conforme Hübner et al., o modelo de programação JaCa é projetado e programado como um conjunto de agentes que trabalham e cooperam dentro de um ambiente comum.

O JaCa permite separar a programação dos agentes e do ambiente (artefatos). Ao programar os agentes o desenvolvedor se ocupa em implementar a lógica de controle das tarefas que devem ser executadas. Por outro lado, ao programar o ambiente, como uma abstração de primeira classe provendo as ações e funcionalidades exploradas pelos agentes para fazer suas tarefas.

Outra ferramenta é *ojAlgo*, que segundo Peterson (2008), é uma biblioteca de código aberto em Java que implementa métodos para resolução de problemas matemáticos, de álgebra linear e otimização voltado para o domínio financeiro, é um framework de álgebra linear completo com várias decomposições matriciais e a capacidade de utilizar diversos formatos numéricos como elementos de matrizes multidimensionais altamente eficientes. No nosso projeto o ojAlgo foi utilizado para minimizar a distância percorrida pelos drones na seleção quadrante a ser explorado, que é uma outra estratégia não abordada neste artigo.

O EISMASSim é baseado no *Environment Interface Standard* (EIS), um padrão proposto para interação agente-ambiente. Ele mapeia a comunicação entre agentes e o servidor *MASSim* (enviando e recebendo mensagens XML) para chamadas de método

Java. Em outras palavras, o *EISMASSim* é um ambiente de proxy no lado do cliente que lida com a comunicação com o servidor *MASSim* completamente sozinho (AHLBRECHT, 2018).

5. Sistema de Prioridade das tarefas do Agente

As principais estratégias empregadas pela equipe são desenvolvidas em dois níveis. No nível extra-agente os agentes devem interagir entre si e decidir quais tarefas cada um vai colocar em ação deixando sempre os outros a par do que ele está fazendo. Já no nível intra-agente definimos um sistema de prioridades na qual cada tarefa tem uma prioridade dinâmica, com isso conseguimos manter nosso sistema escalável de forma que qualquer nova funcionalidade seja facilmente adicionada.

A arquitetura modular propiciada pelo sistema de prioridade permite dividir as estratégias em blocos diferentes (arquivos asl[1]). Essa capacidade permite que sejam adicionados novos comportamentos de forma independente. Para adicionar uma nova funcionalidade, é necessário seguir um padrão que consiste em manter três crenças no agente: "todo", "steps", "expectedplan".

A crença "todo" diz respeito à o que é o comportamento e qual a sua prioridade no sistema, organizada da seguinte forma:

```
+todo(nomeDoComportamento,Prioridade)
```

Sendo "Prioridade" uma variável que recebe um inteiro com valor de 1 a 10. A crença "steps" é a que irá guardar lista de ações que o agente deve realizar para cumprir o seu objetivo, construída da seguinte forma:

```
+steps( nomeDoComportamento, PLAN)
```

Sendo "PLAN" uma lista de passos a serem seguidos para atingir um objetivo do comportamento. Um exemplo é o plano de armazenar um item:

```
[goto(storage2),store(item1,2)]
```

No qual "goto" é uma ação de ir para algum local, passando o nome da localidade da mesma e "store" é uma ação de guardar um item definindo seu tipo e a quantidade que irá ser guardado. Como essa lista é consumida ao longo da execução do programa, temos que ter salvo o que já foi realizado para poder recuperar em caso de o agente precisar realizar uma ação de maior prioridade, desta forma criamos outra crença "expectedplan" onde nela é guardado o mesmo plano que foi guardado na crença "steps".

A crença "expectedplan" é utilizada para a função de *rollback* do sistema que acessa a crença e recupera a lista de ações do plano salvo nela. Neste ponto o sistema compara a lista do que já foi feito na crença "steps" com a lista inteira vinda da crença "expectedplan", retirando da lista as ações já executadas e mantendo na lista somente para onde o agente deve ir e o que faltou ele fazer.

Para o sistema respeitar o comportamento e suas prioridades, a cada step os agentes realizam uma sequência de passos:

1) atualiza a crença "steps", removendo a primeira ação da lista (cabeça) que foi realizada no step anterior caso ela tenha sido completada com sucesso, para assim prosseguir nos passos para realização da tarefa.

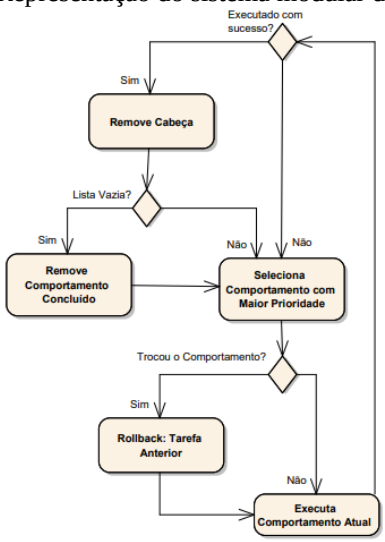
[1] Arquivos deste tipo contém o código em Jason do projeto, ou seja, a parte da programação do comportamento dos agentes.

2) verificar se não tem nenhuma outra tarefa com prioridade maior para ser realizada, caso não tenha o sistema passa para o próximo passo, caso tenha uma nova tarefa de maior prioridade, essa tarefa é selecionada para executar uma ação no step atual. O comportamento anterior que não foi completado, é recuperado para um estado válido através do *roolback.*

3) a realização propriamente dita da ação que está na cabeça da lista de ações, que dependendo do que ela conter vai ser realizada de forma diferente, pois pode ser uma ação nativa do cenário como o "goto", ou pode ser uma crença, que ao ser lida vai disparar outra ação, mantendo assim o fluxo de funcionamento das ações.

Esta é a principal estratégia do sistema e a partir dela que implementamos os diversos comportamentos para a realização de um objetivo comum, a figura 1 exemplifica esta estratégia:

Figura 1: Representação do sistema modular dos agentes.



Fonte: Autor, 2018.

6. Estratégias Utilizadas

As estratégias empregadas pela equipe foram pensadas de forma a atingir os objetivos principais da simulação. O conjunto das principais estratégias que serão abordadas neste artigo podem ser divididos em:

- Recarga dos agentes
- Produção de itens
- Atendimento de demandas

6.1. Recarga dos Agentes

Uma das funções primárias de um veículo autônomo (agente) é o gerenciamento da carga da bateria, pois o veículo não pode ficar sem carga. A importância de satisfazer tal condição se dá pelo fato do agente depender da bateria para realizar qualquer tarefa durante a simulação, ou seja, sem carga suficiente o agente é inaproveitável.

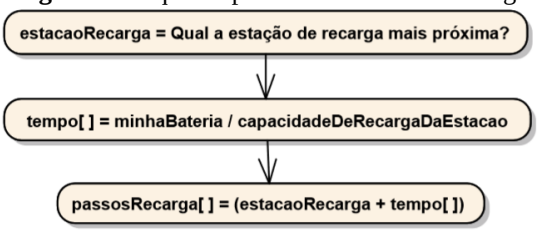
A cada atualização do sensor de carga da bateria o agente calcula baseado na sua localização geográfica atual se é possível chegar ao posto recarga mais próximo e retornar com bateria suficiente para continuar realizando suas tarefas. Com isso quando a carga da bateria menos o custo da viagem de ida e volta até o ponto de recarga, for

menor que uma constante de segurança o agente cria uma tarefa para recarregar a bateria, tal tarefa na lista de prioridades é de importância máxima.

Cada posto de recarga possui uma capacidade de recarga tendo como unidade de medida N unidades de bateria por *step*. Por exemplo: se o posto de recarga possui como capacidade de recarga 3 unidades por *step* e o agente possui como capacidade máxima de bateria 30 unidades, sendo assim para encher a capacidade total da bateria levará 10 *steps*. Caso o resultado contenha casas decimais é realizado arredondamento para cima.

A tarefa para recarregar a bateria consiste em montar uma lista com os passos a serem realizados, conforme exemplifica a figura 2:

Figura 2: Esquema para montar lista de recarga



Fonte: Autor, 2018.

Conforme apresenta o primeiro quadro, o agente procura baseado na sua localização atual qual é o posto de recarga mais próximo, com a estação de recarga em mente, no segundo quadro ele analisa o estado da sua bateria atual com a capacidade de recarga da estação e assim monta uma lista com 1 ou N comandos de recarga (*recharge*).

Após o agente montar a lista de "passosRecarga" ele passa a executar de fato a tarefa de recarga percorrendo a mesma e conforme os passos vão sendo executados os mesmos são removidos. A partir do momento que a lista for vazia o agente considera que a tarefa está finalizada e remove a mesma de sua base de crenças e assim analisa novamente em suas tarefas quais tem a maior prioridade, ou retoma a anterior que estava fazendo.

6.2. Produção de Itens

A nossa estratégia é armazenar os itens coletados no armazém mais central, pois, em média, é esperado que os caminhos até as outras estruturas sejam equilibrados, evitando pontos excessivamente distantes. Os agentes que coletam se deslocam nos dois sentidos entre os pontos de coleta para o armazém central.

6.2.1. Coleta de itens

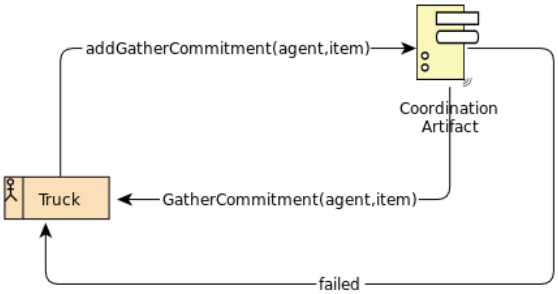
A estratégia de pegar itens depende do descobrimento da localização dos nós de recurso existentes no cenário. Para esta tarefa designamos os caminhões, pois os eles têm a maior capacidade de carga e mesmo com uma velocidade menor eles conseguem transportar mais itens em um período de 50 steps, garantindo o fornecimento dos itens. A tarefa de coleta de itens pode ser dividida, comprometimento com a tarefa, deslocamentos, coleta e depósito.

6.2.1.1. Comprometimento com a tarefa

No início do round o agente interage com o artefato de coordenação através da propriedade observável *addGatherCommitment(agent,item), caso* algum agente já tenha se comprometido em coletar o item é retornado um erro, do contrário é criada uma propriedade observável para que todos os agentes saibam quem é responsável por

coletar os itens, a figura addgatherCommitment representa como essa interação ocorre. O agente fica repetindo esse comportamento até que ele se comprometa em coletar o item ou não existam mais itens disponíveis para ser coletados. Este comportamento pode ser verificado na figura 3

Figura 3: Interação Agente e artefato



Fonte: Autor, 2018

6.2.1.2. Coleta e depósito

Dado que o agente já sabe o item que ele vai coletar é necessário calcular quantas unidades do item ele consegue carregar. Para isso pegamos o volume do item (VI), o volume que o caminhão (VC) consegue carregar e aplicamos a fórmula apresentada na figura 4:

Figura 4: Fórmula que verifica quantidade possível de carregar

$$QTD = \left\lfloor \frac{VC}{VI} \right\rfloor$$

Fonte: Autor, 2018

A variável QTD é a quantidade de itens que o caminhão consegue carregar calculada pelo chão da divisão do volume transportável pelo caminhão (VC) e o volume do item (VI). Esse resultado é usado para que o agente saiba quantas vezes terá de fazer a operação gather no ponto de coleta. As ações de coleta e armazenamento podem falhar. Para tratar essas falhas foi utilizado um plano que inspeciona o resultado da última ação e a menos que essa seja cumprida com sucesso ela será repetida até seu efetivo cumprimento.

6.2.2. Construção de itens:

No cenário existem dois tipos de itens, os que são apenas coletados (simples) e os que são construídos a partir de outros itens (complexos). Toda vez que um item é adicionado ao local de armazenamento o sistema confere qual item é possível de construir a partir dele, observa-se o item individualmente verificando quais os tipos de agentes necessários para sua construção e quais os itens necessários para montá-lo. Os itens complexos precisam de dois agentes na oficina para montá-lo, cada um executando uma operação diferente em conjunto.

6.2.2.1. Agente responsável por montar o item:

Assim como os agentes que pegam os itens simples, os agentes responsáveis por montar os itens iniciam suas tarefas verificando se o seu item já tem algum responsável e se não tiver ele informa a todos que é o responsável pelo item. Em seguida ele

começa um processo de enumerar quais são os itens necessários para construir o respectivo item e quem precisa estar junto. Desta forma começamos a montar a lista de ações do agente, na qual a primeira ação vai ser ir ao local de armazenamento central para pegar os itens necessários. Já sabendo a lista de itens necessários para construir outro, chamamos outro plano que irá montar a ação de pegar os devidos itens necessários, esta ação será repetida quantas vezes for o tamanho da lista, iterando-a. Adicionada a ação de pegar os itens no local de armazenamento o próximo passo é levá-los para a oficina central que já é de conhecimento dos agentes. Chama-se a outra parte necessária, executa o comando de montar o item, que vai sendo repetido até que o outro agente chegue, caso ele não esteja no local esperando, e o item seja montado com sucesso. Para finalizar o item é levado para o local de armazenamento e lá é depositado para ser utilizado em outra ocasião.

6.2.2.2. Agente responsável por ajudar na confecção do item:

No momento que o agente responsável por levar o item pega os itens necessários para construção de outro, inicia-se o processo para chamar o agente responsável por ajudar na confecção, a ir à oficina e executar a ação. Quando é disparada por algum agente a chamada de ajuda utilizamos as informações de qual o tipo de agente necessário para tal ação, quem está chamando e o local que é para ir, desta forma o agente que está chamando verifica dentre os agentes do tipo que ele necessita quais estão livres para o ajudar. Após essa verificação, a solicitação é enviada e aguarda-se a resposta. Os agentes que vão ajudar recebem a solicitação e verificam qual o custo para realizar a tarefa com base em quanto ele terá que se deslocar. Em seguida informa ao agente que solicitou quanto é este custo. Quando todos os agentes responderem a solicitação, o solicitante verifica, dentre todos, qual o menor custo, desta forma ele dispensa os com maior custo e envia somente para o agente de menor custo a confirmação de que ele foi selecionado. O agente selecionado se dirige para a oficina e fica esperando até que o solicitante chegue e assim consigam montar o novo item.

6.3. Atendimento de Demandas

Esta pode ser considerada a estratégia mais importante do cenário, pois ela que define os vencedores. Ganha o jogo quem atende mais vezes as demandas de trabalhos (*jobs*) solicitados pelo servidor. Os *jobs* consistem em entregar os itens montados e coletados nos locais indicados. Um *job* novo é solicitado da seguinte forma pelo servidor da competição: `New job: 3x item9, 1x item6, 2x item7 reward(392) 1-77 storage0 loc(2.30026,48.8242) capacity(10271) Job.` Desta forma temos todas as informações necessárias para completa-lo. Primeiramente os itens necessários, neste caso: temos que entregar 3 *item9*, 1 *item6* e 2 *item7*, ganhando uma recompensa de 392 *massium.* O tempo para a realização do trabalho é a partir do *step* 1 até o *step* 77. O local de entrega é no *storage0* nas coordenadas passadas, e por fim a capacidade de itens que o local suporta.

A estratégia se dá da seguinte forma: primeiramente quando um trabalho é solicitado, todos os agentes recebem a solicitação como mostrada no parágrafo anterior. Os agentes que estão disponíveis verificam no *job* os seguintes quesitos para ver se podem realizar o trabalho: 1) Se a capacidade de volume que ele consegue carregar é o suficiente para completar o trabalho. 2) Se ele possui tempo para realizar o trabalho, ou seja, é estimado quando tempo o agente levaria para pegar os itens e levar até o local indicado. 3) Se não tem nenhum outro agente fazendo o trabalho. Após conferir esses quesitos, se forem positivos, inicia-se a montagem do plano para realização do *job.*

O primeiro passo da lista para realizar o trabalho é ir até o local de armazenamento central (onde foram depositados os itens das estratégias citadas na seção 6.2.1 e 6.2.2). Depois é verificado quais itens são necessários e as suas quantidades, esta função é executada dando o seguinte comando "*retrieve(ITEM, QUANTIDADE)",* para isso o agente deve estar no local de armazenamento e assim ele recupera o item. Estando com todos os itens, o próximo passo é leva-los ao local indicado, e para finalizar executa o comando "*deliverJob(NOME_DO_JOB)"* que já faz o agente entregar automaticamente todos os itens necessários. A montagem deste plano é feita instantaneamente, desta forma o agente executa o plano e entrega o *job*, para ganhar recompensa para a equipe.

7. Considerações Finais

Ao analisar as principais estratégias empregadas pela nossa equipe percebemos o quanto elas podem ser adaptadas ao mundo real e se mescladas fazem ainda mais sentido. Com as estratégias que citamos neste artigo, podemos fazer uma analogia a um agente autônomo que entrega correspondências, por exemplo, o qual terá que ir a um local para buscar as correspondências e ir entregando até que seu combustível chegue ao limite para ele abastecer ou recarregar novamente. A arquitetura modular empregada nos leva perceber a facilidade que uma nova funcionalidade tem de ser adiciona ao sistema. A *Multi-Agent Programming Contest* ocorrerá em setembro, e servirá como teste da eficiência das nossas estratégias tendo em vista que poderemos compará-las com as das demais equipes, até lá estaremos concentrados na finalização e refinação das estratégias bem como a preparação para a competição.

Referências:

FAPESP. Chamada: Cidades Inteligentes tem resultado da etapa de enquadramento. Disponível em: <http://www.fapesp.br/10362.phtml**>.** Acesso em 14 ago. 2018.

GIRARDI, R. Engenharia de Software Baseada em Agentes. In: CONGRESSO BRASILEIRO DE CIÊNCIA DA COMPUTAÇÃO, 8th. Itajaí, 2004. p. 6-8.

DE FARIAS, J. E. P. Cidades inteligentes e comunicações. Revista de Tecnologia da Informação e Comunicação, v. 1, n. 1, p. 28-32, 2011.

WOOLDRIDGE, M. Intelligent agents: The key concepts. In: ECCAI. Advanced Course on Artificial Intelligence. Springer, Berlin, Heidelberg, 2001. p. 3-43.

BORDINI, R. H.; VIEIRA, R. Linguagens de Programação Orientadas a Agentes: uma introdução baseada em AgentSpeak (L). Revista de informática teórica e aplicada. Porto Alegre. Vol. 10, n. 1 (2003), p. 7-38, 2003.

HÜBNER, J. F. et al JaCaMo Project | Multi-Agent Programming Framework. Disponível em: <http://jacamo.sourceforge.net/>. Acesso em: 25 jul. 2018.

ALHBRECHT, T. Massim: Simulation platform for the Multi-agent Programming contest. Multi-Agent Programming Contest. Disponível em: <https://github.com/agentcontest/massim/blob/master/docs/eismassim.md >. Acesso 26

jul. 2018.

Calculate distance and bearing between two Latitude/Longitude points using haversine formula in JavaScript. Disponível em: <http://www.movable-type.co.uk/scripts/latlong.html>. Acesso em: 27 jul. 2018.

LONGARAY, A. A.; BEUREN, I. M. Cálculo de minimização dos custos de produção por meio da programação linear. VIII Congresso Brasileiro de Custos – São Leopoldo, RS, Brasil, 2001.

PETERSON, A. ojAlgo v24: Linear Algebra, Optimisation and Maths for Finance. Disponível em: <https://dzone.com/articles/ojalgo-v24-linear-algebra-opti>. Acesso em: 25 jul. 2018.

NAPOL, I. Além do passeio: Volvo desenvolve um caminhão de lixo autônomo. Disponível em: <https://www.tecmundo.com.br/volvo/116847-passeio-volvo-desenvolve-caminhao-lixo-autonomo.htm>. Acesso 1 ago. 2018.