5 A Aplicação LearnAgents

Este capítulo apresenta um sistema multi-agente para um complexo cenário de *procurement*, onde os agentes compram e vendem bens em leilões simultâneos para montar pacotes de viagens para seus clientes. O sistema é utilizado em ambientes competitivos onde os participantes competem por um número limitado de bens.

O *LearnAgents* (Sardinha et al., 2004c; Sardinha et al., 2005a) é um sistema multi-agente para o comércio automatizado em mercados eletrônicos com leilões simultâneos e assíncronos. O sistema é utilizado em ambientes competitivos onde os participantes competem por um número limitado de bens. A sua arquitetura define agentes que resolvem subproblemas do comércio eletrônico de bens, e apresenta uma técnica para combinar todas essas soluções. Os subproblemas típicos são predição de preços, planejamento de lances, alocação de bens, negociação, entre outros. O *LearnAgents* utilizou o ambiente do *Trading Agent Competition* (TAC) para testar o sistema desenvolvido, pois esse ambiente simula um ambiente competitivo e permite mensurar a performance do sistema contra um *benchmark* internacional.

O *Trading Agent Competition* é uma competição anual para incentivar a pesquisa e o desenvolvimento de sistemas para comércio automatizado em mercados eletrônicos. O TAC está em sua quinta edição e atrai trabalhos das principais instituições de pesquisa americanas, européias, e asiáticas, tais como as universidades de Harvard, Cornell, Brown, e Michigan. O TAC *Classic* (Wellman et al., 2001) é uma competição onde os participantes são donos de uma agência de viagem, e operam em um cenário complexo de *procurement*, ou compra de bens, em leilões simultâneos. Os participantes são agentes de software, e na competição não pode haver intervenção humana em hipótese alguma.

5.1.O TAC Classic

Os competidores no TAC *Classic* operam dentro de um mercado varejista. Todos os agentes compram e vendem bens em leilões simultâneos para montar pacotes de viagens para seus clientes. Esses bens podem ser passagens aéreas, hospedagens em hotéis, e ingressos para museus, parques, ou luta livre. No inicio de cada partida, cada agente recebe uma carteira de oito clientes com as suas respectivas preferências. A pontuação de cada agente é calculada pela diferença da utilidade dos pacotes montados e os custos dos bens adquiridos nos leilões, refletindo o lucro total obtido pelo agente.

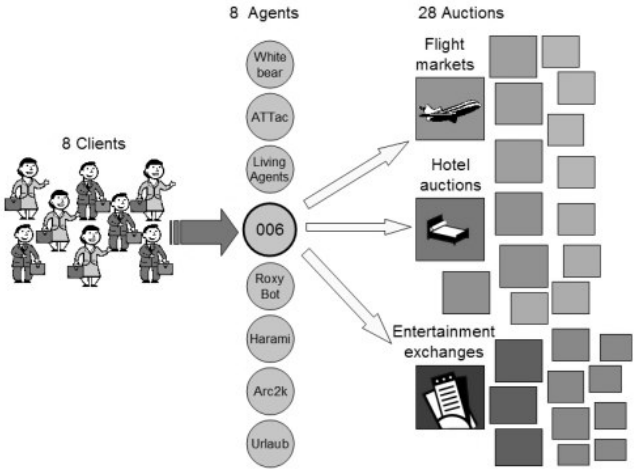

Figura 37: Ilustração do TAC Classic.

A figura 37 apresenta o ambiente do TAC *Classic*. Cada partida permite que oito agentes negociem bens para seus clientes. Os agentes devem atender as preferências de oito clientes, e os bens são comprados em vinte e oito leilões simultâneos. As interdependências são claras nesse jogo, pois um cliente só pode viajar se tiver uma passagem de ida e outra de volta. Além disso, cada cliente precisa de diárias num hotel durante a sua estadia.

5.2. Edições Anteriores do TAC

A primeira versão do ambiente do jogo utilizado no TAC foi projetada em 2000 por um grupo do Laboratório de Inteligência Artificial da Universidade de Michigan (Wellman et al., 2001). As versões dos anos seguintes passaram por

mudanças significativas para que o ambiente simulasse um mercado competitivo, até que em 2003, o ambiente do TAC foi re-projetado pelo Laboratório de Sistemas Inteligentes no Instituto Sueco de Ciência da Computação (SICS) e poucas mudanças foram feitas desde então.

A primeira edição do TAC foi sediada na cidade de Boston, Massachusetts, EUA, em Julho de 2000 dentro da *IV International Conference on Multiagent System (ICMAS-00)*. Devido às regras, a estratégia predominante nos agentes esperava comprar os bens no final da partida e utilizava lances altos com valores iguais ao valor marginal do bem. Por esta razão, uma preocupação central dos agentes que participaram nesta competição estava na resolução do problema de alocação dos bens adquiridos de acordo com as preferências dos clientes. Este problema de otimização é NP-Completo (Greenwald, 2003), e foi incorporado às funcionalidades do TAC no segundo ano. A tabela 8 resume os resultados obtidos da TAC'00. A pontuação é uma média dos resultados das várias partidas na rodada final. A estratégia utilizada pelo o ATTac (Stone et al., 2001; Strone et al., 2002), vencedor desta edição, é baseada em um modelo de programação inteira.

Posição	Agente	Pontuação Média
1	ATTac	3398.26
2	RoxyBot	3283.24
3	aster	3068.34
4	umbctac1	3050.99
5	ALTA	2198.01
6	m_rajatish	1872.71
7	RiskPro	1569.91
8	T1	1167.40

Tabela 8: Resultado do TAC de 2000.

A segunda versão do TAC, em outubro de 2001, teve a sua realização dentro da *Third ACM Conference on Electronic Commerce (EC'01)*, em Tampa, Flórida. As regras foram modificadas para que os agentes pudessem se beneficiar da compra de bens no início do jogo. A tabela 9 apresenta o resultado final da competição. O vencedor *LivingAgents* (Klaus, 2001) utiliza um sistema multi agente com várias heurísticas simples, e se aproveita da mudança nas regras para tomar a maioria das decisões bem no início de cada jogo. Esta estratégia permite uma economia considerável na compra das passagens aéreas. Além disso, o *LivingAgents* utiliza uma média dos preços de fechamento dos leilões dos hotéis

para calcular preços futuros, e assim poder calcular os lucros da melhor alocação por cliente.

Posição	Agente	Pontuação Média
1	livingagents	3670.0
2	ATTac	3621.6
3	Whitebear	3513.2
4	Urlaub01	3421.2
5	Retsina	3351.8
6	SouthamptonTAC	3253.5
7	CaiserSose	3074.1
8	TacsMan	2859.3

Tabela 9: Resultado do TAC de 2001.

As finais da terceira edição do TAC foram realizadas em julho de 2002 na cidade de Edmonton no Canadá. A competição foi co-localizada com o *Eighteenth National Conference on Artificial Intelligence (AAAI-02)*. As regras do TAC'01 para o TAC'02 permaneceram as mesmas, o que já demonstrava um certo grau de amadurecimento da competição. Além disso, a comunidade de pesquisa também demonstrava uma expansão, pois mais participantes foram inscritos neste ano. A tabela 10 apresenta os resultados do TAC'02. O agente *Whitebear* (Vetsikas et al., 2003) foi o vencedor desta edição, e apresentou uma técnica gulosa para o cálculo de alocação de bens junto com uma estratégia de lances "parciais", onde os lances são considerados independentes um dos outros. Esse agente calcula primeiro os bens a serem comprados usando um algoritmo guloso, e demonstra empiricamente que as alocações encontradas estão sempre próximas da alocação ótima. Depois, o agente decide o valor do lance para cada bem utilizando uma estratégia de lances adaptativa que não ultrapassa o valor marginal do bem.

Posição	Agente	Pontuação Média
1	Whitebear	3412.78
2	SouthamptonTAC	3385.46
3	Thalis	3246.27
4	UMBCTAC	3235.56
5	Walverine	3209.52
6	Livingagents	3180.89
7	kavayaH	3099.44
8	Cuhk	3068.77

Tabela 10: Resultado do TAC de 2002.

A quarta edição do TAC foi realizada dentro da *Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)* em agosto de 2003. A tabela 11 apresenta os resultados desta edição. Neste mesmo ano, uma nova modalidade da competição foi criada para atender o cenário de produção e venda de produtos. Essa modalidade se chama TAC *Supply Chain Management*. Os agentes competidores agora precisam programar a compra de matéria prima, a produção e a distribuição para poder atender a demanda dos clientes.

Posição	Agente	Pontuação Média
1	ATTac01	3199.77
2	PackaTAC	3162.59
3	Whitebear	3141.74
4	Thalis	3132.61
5	UMBCTAC	3108.24
6	NNN	3070.94
7	Walverine	3005.30
8	RoxyBot	2799.02
9	Zepp	1714.38

Tabela 11: Resultado do TAC de 2003.

5.3.A arquitetura do LearnAgents

Parte do sucesso do *LearnAgents* advém do uso da tecnologia de agentes, pois propiciam um novo paradigma para a concepção de sistemas inteligentes para atuarem em complexos ambientes distribuídos, competitivos e abertos. Uma técnica interessante para modelar um problema computacionalmente difícil com o paradigma de agentes é dividir esse mesmo problema em subproblemas mais simples. Assim, cada agente se torna responsável por um ou mais desses

subproblemas. A figura 38 apresenta a arquitetura do *LearnAgents*, e esta seção descreve a modelagem do sistema multi-agente usando uma linguagem de modelagem chamada ANote (Choren, 2002). Essa arquitetura utiliza um *design* semelhante ao sistema multi-agente SIMPLE (Milidiu et al, 2003a; Milidiu et al, 2003b; Milidiu et al, 2003c), porém com mais agentes, algoritmos de *machine learning* diferentes, e uma performance superior (Sardinha et al., 2004a).

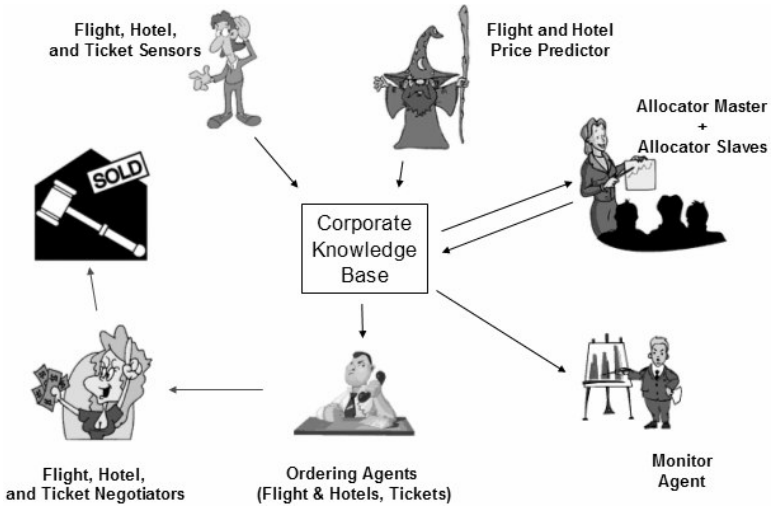

Figura 38: A arquitetura do LearnAgents.

Todos os nossos projetos de software utilizam uma técnica orientada a objetivos para a fase de requisitos. Esta fase utiliza o processo recursivo de decomposição de um objetivo do problema principal em vários objetivos de subproblemas mais simples. No caso do *LearnAgents*, esse processo recursivo definiu todos os objetivos dos subproblemas relacionados com o comércio de bens em um mercado competitivo.

O objetivo maior desse sistema era comprar pacotes de viagens para os clientes e obter o maior lucro possível. Como o mercado era competitivo, a negociação foi identificada como um dos objetivos evidentes para o processo de compra dos pacotes de viagens. Além disso, para obter o maior lucro possível é preciso possuir uma base de conhecimento do mercado para auxiliar o processo de negociação. A figura 39 mostra esse processo de decomposição dos objetivos.

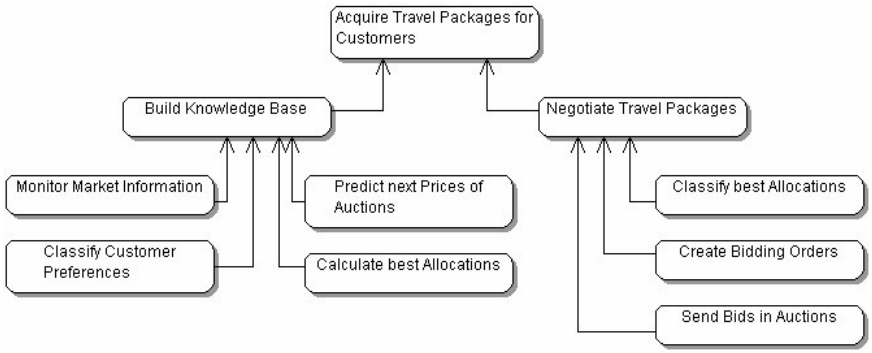

Figura 39: A decomposição dos objetivos relacionados com o comércio de bens.

A base de conhecimento deve possuir: (i) os preços correntes dos leilões - objetivo Monitorar Informações do Mercado; (ii) as preferências dos clientes - objetivo Classificar as Preferências dos Clientes; (iii) as predições dos preços dos leilões - objetivo Prever próximo Preços dos Leilões; e, (iii) alocações de bens para os clientes utilizando a informação armazenada nesta base - objetivo Calcular as melhores Alocações.

O processo de negociação requer: (i) um cenário representado pelas alocações que permita negociações de bens com alta lucratividade e com o mínimo de risco - objetivo Classificar as melhores Alocações; (ii) a escolha do valor do lance para cada bem desejado - objetivo Criar Lances para Ordens; e, (iii) envio de lances para os leilões de bens - objetivo Enviar Lances para Leilões.

A segunda fase do processo de modelagem é criar os tipos de agentes que sejam capazes de conduzir os objetivos da figura 39. Essa criação começa com processo de relacionamento de todos os objetivos de nível mais baixo e com tipos de agentes. A tabela 12 faz um mapeamento entre esses objetivos e tipos de agentes. A figura 40 apresenta os vários tipos de agentes e seus relacionamentos. Esta figura permite uma visão estática do projeto do nosso sistema multi-agente.

Goal	Agent
Monitor Market Information	Hotel Sensor, Flight Sensor, Ticket Sensor
Classify Customer Information	Hotel Sensor
Predict Next Prices of Auctions	Price Predictor
Calculate Best Allocations	Allocation Master, Allocation Slave
Classify Best Allocations	Ordering
Create Bidding Orders	Ordering
Send Bids in Auctions	Hotel Negotiator, Flight Negotiator, Ticket Negotiator

Tabela 12: A mapeamento entre subproblemas e tipos de agentes.

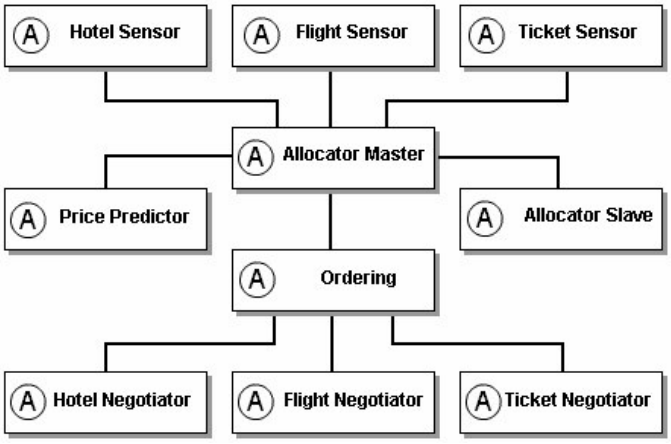

Figura 40: Arquitetura do LearnAgents.

Após a criação dos vários tipos de agentes, cada agente deve possuir uma especificação de cenários. Esses cenários possuem informações sobre o contexto de como cada agente deverá atingir seus objetivos. Segue abaixo, uma descrição resumida dos agentes criados e seus cenários:

Os Sensores dos Leilões (Hotel, Vôo, e Ticket) – Esses agentes são responsáveis por atualizar a base de conhecimento com os preços correntes dos leilões, e enviar mensagens para outros agentes sobre eventos importantes do ambiente externo. A figura 41 apresenta o digrama de cenário dos Sensores de leilões.

Monitor Market Information	
Main Agent	Sensor Agent
Preconditions	Event from environment
Main Action Plan	while true Collect current prices of auctions Update CKB Send message to Price Predictor end while
Interaction	Price Predictor
Variant Action Plan	

Figura 41: Diagrama de cenário dos Sensores.

Preditores – Através de uma série histórica dos preços de vôos e hotéis, os Preditores são capazes de prever para que valor o preço do leilão deve caminhar. A figura 42 apresenta o digrama de cenário dos Preditores.

Predict Prices	
Main Agent	Price Predictor Agent
Preconditions	Message from Sensors
Main Action Plan	while true Collect current prices in CKB Predict Auction Prices Update CKB Send message to Allocator Master end while
Interaction	Allocator Master
Variant Action Plan	

Figura 42: Diagrama de cenário dos Preditores.

Alocador Mestre e Escravos – O principal objetivo dos Alocadores é calcular vários cenários diferentes. Esses agentes utilizam os preços previstos e correntes para montar cenários diferentes. Todas as decisões de bens a serem adquiridos são baseadas nesses cenários. A figura 43 apresenta o digrama de cenário dos Alocadores.

Distribute Allocation Problem	
Main Agent	Allocator Master Agent
Preconditions	Message from Price Predictor
Main Action Plan	while true Collect current prices in CKB Divide allocation problem to slaves Insert allocation problems in CKB Send message to Allocator Slaves end while
Interaction	Allocator Slaves
Variant Action Plan	

Solve Allocation Problem	
Main Agent	Allocator Slave Agent
Preconditions	Message from Allocator Master
Main Action Plan	while true Collect allocation problem in CKB Solve allocation problem Update solution in CKB Send message to Allocator Master end while
Interaction	Allocator Master
Variant Action Plan	

Combine Allocation Problem	
Main Agent	Allocator Master Agent
Preconditions	Message from Allocator Slave
Main Action Plan	while true Collect slave solution in CKB Combine solutions Update CKB Send message to Ordering end while
Interaction	Ordering
Variant Action Plan	

Figura 43: Diagrama de cenário dos Alocadores.

Ordenador – O Ordenador assume um papel de chefe dos negociadores, e utiliza os cenários calculados pelos Alocadores para tomar as suas decisões. De minuto em minuto, os Negociadores recebem ordens de compras com diretivas básicas: quantos bens devem ser comprados, e o valor máximo do lance que pode ser dado para um determinado bem. A figura 44 apresenta o digrama de cenário do Ordenador.

Classify Best Allocations	
Main Agent	Ordering Agent
Preconditions	Message from Allocator Master

Main Action Plan	while true
	Collect allocations in CKB
	for each Negotiator Agent do
	Decide goods to buy
	Decide good amounts
	Calculate good high prices
	Send message to Negotiator Agent
	end do
	end while

Interaction	Hotel Negotiator, Flight Negotiator, Ticket Neg.
Variant Action Plan	

Figura 44: Diagrama de cenário do Ordenador.

Negociadores – Para cada leilão há um Negociador especializado na compra de um tipo bem. Os Negociadores só começam a operar nos leilões depois que recebem as ordens de compra do Ordenador. A cada minuto, as ordens podem ser alteradas se houver alguma mudança de estratégia. Os negociadores têm o objetivo de comprar os bens seguindo as diretivas determinadas pelo Ordenador, porém toda transação deve ser feita pelo menor preço possível. A figura 45 apresenta o digrama de cenário dos Negociadores.

Negotiate Goods	
Main Agent	Negotiator Agent
Preconditions	Message from Ordering
Main Action Plan	while true Collect good orders in CKB Negotiate goods in auctions end while
Interaction	
Variant Action Plan	

Figura 45: Diagrama de cenário dos Negociadores.

Monitor – Esse agente é responsável por armazenar informações da base de conhecimento em um banco de dados. Esses dados são utilizados por uma ferramenta da CA (Computer Associates) chamada Forest and Trees para monitorar a performance de cada agente individualmente e o sistema multi-agente como um todo. Uma tela da ferramenta de monitoração pode ser vista na figura 46.

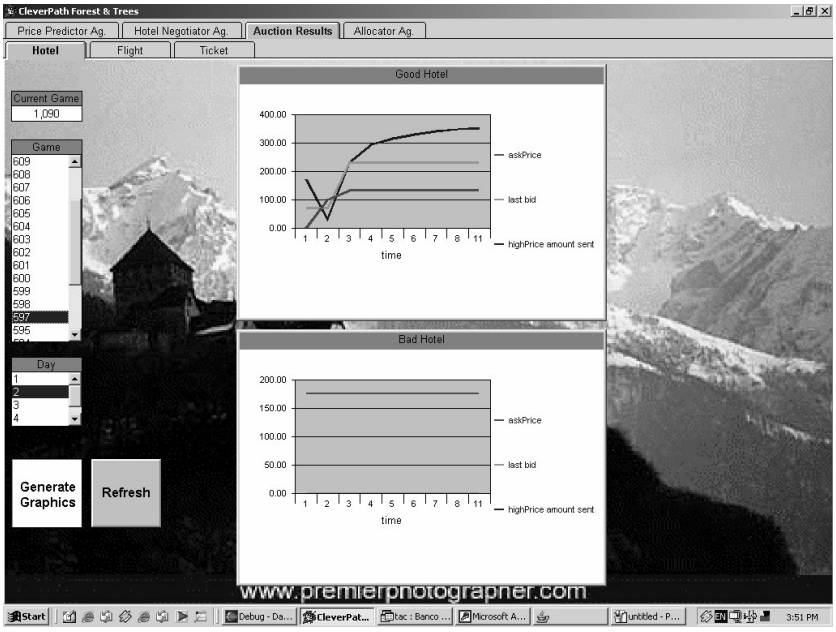

Figura 46: Interface do Monitor.

Um diagrama interessante que deriva dos diagramas de cenário é o diagrama de interação. Esse diagrama deixa explicito a comunicação entre os agentes, e permite uma visão dinâmica do sistema.

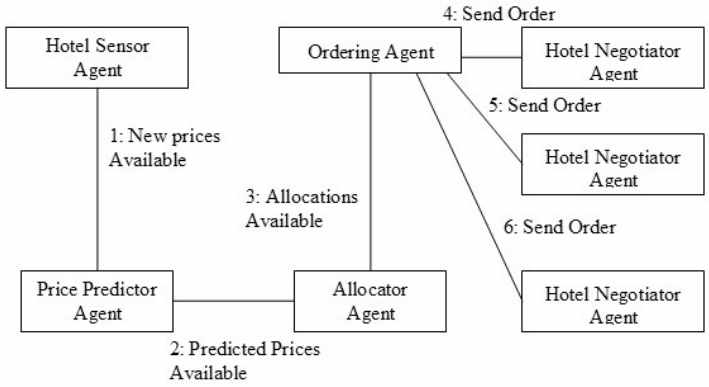

Figura 47: Diagrama do Interação do LearnAgents.

Na figura 47, o Sensor de Hotel recebe um evento do ambiente com novas cotações dos leilões de hotéis. O Sensor atualiza a base de conhecimento e envia uma mensagem para o Preditor informando que os preços foram atualizados. O Preditor calcula as novas previsões de preços dos leilões, e armazenas essas predições na base de conhecimento.

Os Alocadores calculam os diferentes cenários baseados nos preços correntes e previstos. Esses cenários são alocações de bens de turismo, e eles são calculados com um *solver* de programação inteira. O Alocador Mestre envia uma mensagem para o Ordenador após esse calculo dos cenários.

O Ordenador recebe a mensagem e decide a quantidade necessária de bens para atender a demanda dos clientes. Além disso, o Ordenador calcula o valor máximo de um lance para cada bem a ser comprado. Os Negociadores recebem a mensagem do Ordenador com os pedidos de compra e o valor máximo do lance. Os Negociadores possuem o objetivo de comprar esses bens pelo menor preço possível, sem exceder um lance maior do que o máximo definido pelo Ordenador.

5.4.O TAC *Classic* 2004 e Resultados

O TAC *Classic* 2004 foi a primeira participação do *LearnAgents* na competição. O TAC foi escolhido como ambiente para testar a arquitetura, pois apresenta características de um mercado competitivo e permite avaliar a performance do *LearnAgents* em relação a outros sistemas para comércio eletrônico.

Os participantes cadastram seus agentes de software no torneio pela Internet. Esses agentes de software passam então a disputar autonomamente as partidas via Internet, sem qualquer intervenção humana. A competição de 2004 reuniu os representantes das equipes em Nova York, dentro do *Third International Joint Conference on Autonomous Agents and Multi Agent Systems*.

A competição foi disputada em três etapas. Na primeira delas, o *Qualifying Rounds*, com os agentes conectados via Internet durante duas semanas, 24 horas por dia, o *LearnAgents* ficou na quinta colocação dentre os 14 participantes. Na rodada seguinte, o *Seeding Rounds*, o agente ficou boa parte do tempo disputando a quarta posição. Porém, devido a uma queda na madrugada do link de comunicação com a Internet, ele foi penalizado e caiu para a sétima posição. Mesmo assim, o *LearnAgents* ainda conseguiu recuperar uma posição e encerrar essa rodada em sexto lugar.

Na semifinal, o *LearnAgents* subiu então para a quarta posição. Para a final, foram selecionados apenas os 8 melhores competidores, que disputaram 35 partidas sucessivas. O *LearnAgents* consegui então confirmar sua robustez em

ambientes fortemente competitivos e assegurar a terceira posição. A frente do *LearnAgents* ficaram os agentes experientes *Whitebear* (Vetsikas et al., 2003) da Cornell University, seguido do *Walverine* (Cheng et al., 2003) da University of Michigan. Em sua quarta participação, o *Whitebear* consegui o bicampeonato, e o *Walverine* foi desenvolvido pelos criadores da competição.

Os resultados da tabela 13 mostram um grande equilíbrio entre os finalistas, e isso comprova a existência de muitas soluções robustas para esse jogo de varejo altamente competitivo. Em todas 35 partidas disputadas o *LearnAgents* não obteve qualquer resultado negativo, como mostra o gráfico da figura 48. Isso evidencia a boa performance da nossa estratégia em ambientes competitivos de varejo, pois é capaz de garantir lucros significativos e automatizar todos os processos de decisão.

Posição	Agente	Pontuação Média
1	Whitebear04	4122.11
2	Walverine	3848.97
3	LearnAgents	3736.62
4	SICS02	3708.24
5	NNN	3665.97
6	UMTac-04	3281.43
7	Agent@CSE	3262.51
8	RoxyBot	2015.02

Tabela 13: Resultado do TAC de 2004.

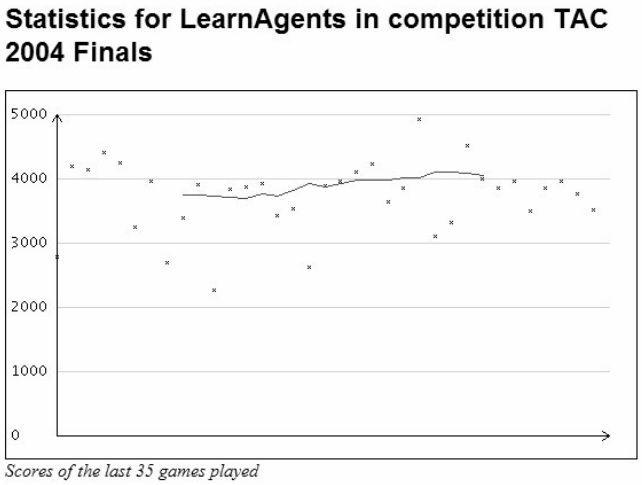

Figura 48: Resultado do TAC de 2004.

5.5.Os Algoritmos utilizados nos Agentes do LearnAgents

Esta seção apresenta os principais algoritmos utilizados no Preditor, Alocadores, Ordenador, e Negociador de Hotel. O restante dos agentes utiliza uma lógica simples que é muito parecida com o código apresentado nos cenários da seção 5.3.

5.5.1.Preditor

O agente Preditor utiliza uma técnica chamada Média Móvel (Bowerman et al. 1993) para calcular os preços dos leilões de hotéis. A técnica Exponencial Suavizada (Bowerman et al. 1993) também foi testada para calcular os preços dos leilões de hotéis, mas apresentou um erro de predição maior. Conseqüentemente, a fórmula da Média Móvel utilizada para prever os preços de cada bem é dada por:

$$s_i = \frac{1}{n} \sum_{j=i}^{i+n-1} a_j.$$

onde:

$\{a_i\}_{i=1}^{N}$ é a série histórica dos preços de cada bem (*N termos*)

$\{s_i\}_{i=1}^{N-n+1}$ é a seqüência de *n* preços previstos (janela de tamanho *n*)

O Preditor utiliza também uma técnica chamada Máxima Verossimilhança (Mitchell, 1997) para prever uma variável aleatória escondida *x* (de uma distribuição uniforme entre [-10,30]) escolhida para cada um dos oito leilões de vôo. O preço inicial de cada leilão de vôo é escolhido a partir de uma distribuição uniforme entre [250,400]. O processo que atualiza esse preço é um passeio aleatório, e é perturbado a cada 10 segundos por um valor escolhido de uma distribuição uniforme:

[-10, *x(t)*] if *x(t)* > 0,

[*x(t)*, 10] if *x(t)* < 0,

[-10, 10] if *x(t)* = 0,

onde *x(t)* = 10 + (*t* / 9:00)*(*x* - 10)

Os preços dos leilões de vôos são atualizados a cada 10 segundos, porém o Sensor de Vôo recebe essas cotações a cada 30 segundos, após três atualizações. A variável aleatória *x* é prevista utilizando a máxima verossimilhança:

$$\arg\max \prod P\,(y_t + y_{t-10} + y_{t-20} = s \mid x)$$

O valor de y_t é o incremento do preço no tempo *t*, e *s* é a diferença entre o preço atual da cotação e a última cotação recebida a 30 segundos atrás. A predição do preço futuro é calculada utilizando o valor esperado *E(quote)* utilizando a variável *x* e o processo descrito acima. A figura 49 apresenta a predição do leilão de vôo de chegada do dia 3 no jogo 6091 (Final do TAC Classic de 2004).

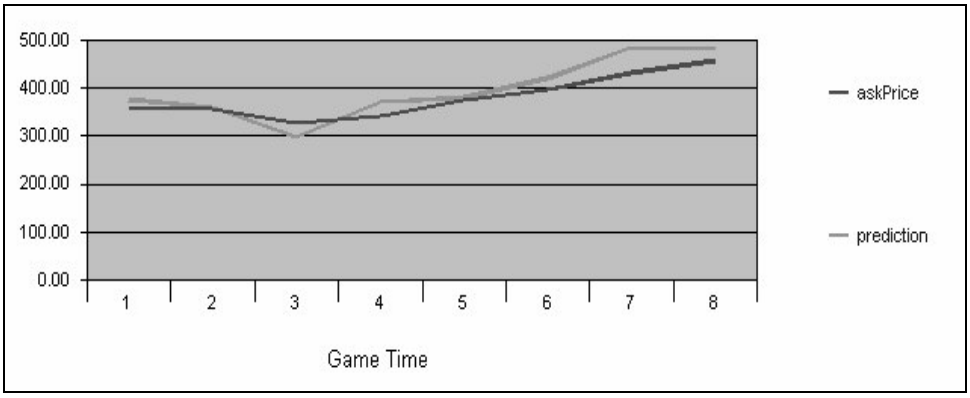

Figura 49: Resultado do TAC de 2004.

5.5.2. Alocador

O Alocador utiliza um modelo de programação inteira para criar diferentes cenários. Esse modelo recebe como entrada as preferências dos clientes, os preços correntes, os preços previstos, e os bens já adquiridos. O *solver* não é executado apenas para encontrar a melhor alocação de bens, mas o maior número de boas alocações geradas em 25 segundos.

Cada alocação define o número de bens por cliente que devem ser compradas em cada leilão com a sua respectiva utilidade. A função objetivo é descrito pela equação abaixo:

$$\sum_{client=1}^{8} Utility(client) - \sum_{client=1}^{8} Cost(client) - OtherCosts - HomogeneityFunc$$

A utilidade do cliente é medida em dólares, e precisa ser um pacote de viagem viável e um pacote de entretenimento viável. A utilidade é medida pela equação:

Utility(client) = 1000 - travel_penalty + hotel_bonus + fun_bonus

Um pacote de viagem é viável se cada cliente tiver uma passagem aérea para chegar e sair de TACTown, e diárias em hotéis entre o dia de chegada e o dia de saída. Além disso, o cliente não pode trocar de tipo de hotel na viagem, e só existem dois tipos de hotéis nesse jogo, o hotel BOM e o hotel RUIM. O agente pode ser penalizado pelo pacote de viagem se o pacote é viável, mas não atende exatamente a preferência do cliente. Esse cálculo da penalidade é dado pela fórmula:

travel_penalty = 100*(|*AA* - *PA*| + |*AD* - *PD*|)

onde,

AA é o dia preferido do cliente para chegar em TACTown,

PA é o dia preferido do cliente para deixar TACTown

AD é a data que o cliente chega em TACTown,

PD é a data que o cliente deixa TACTown.

O bônus de hotel é dado para cada agente que conseguir enviar um cliente para TACTown e hospedá-lo no hotel BOM. O cálculo é feito pela seguinte fórmula:

*hotel_bonus = GHI * HP*

onde,

GHI é um indicador de hotel BOM (assume o valor 1 se diárias são do hotel BOM, ou assume o valor 0 se diárias são do hotel RUIM),
HP é um premio recebido em utilidade.

O bônus para os ingressos de entretenimento é dado para o agente que consegue comprar um ou mais ingressos num pacote de viagem viável. Um pacote de entretenimento para cada cliente é considerado viável se houver apenas um ingresso por dia, e o dia desses ingressos coincide com as datas da estadia. O cálculo é feito pela seguinte fórmula:

*fun_bonus = AWI*AW + API*AP + MUI * MU*
onde,
AWI, API, e MUI são indicadores de ingressos (valor {0,1}) para cada tipo (Alligator wrestling, Amusement park, Museum), e AW, AP and MU são prêmios em utilidade recebido para cada tipo de ingresso.

O *Cost(client)* é a soma dos gastos por cliente com a compra das passagens de ida e volta, diárias em hotéis, e ingressos. O *OtherCosts* é a soma dos gastos de todos os bens não utilizados pelos clientes, perdas e penalidades. O *HomogeneityFunc* é uma função que inclui uma penalidade extra na função objetivo, e é dado pela seguinte equação:

$$\sum_{day=1}^{4} Penalty(hotelType, day).ExtraRoom(hotelType, day)$$

O *Penalty(hotelType,day)* é uma penalidade extra dada para cada diária excedente de um dado tipo (0 = Bad Hotel, 1 = Good Hotel) e dia (1 to 4) no caso de *hotel(hotelType,day) > maxHotel(hotelType,day).*

O *hotel(hotelType,day)* é o numero de diárias de hotéis alocadas de um dado tipo e um dado dia, e *maxHotel(hotelType,day)* é um parâmetro que define um número máximo de hotéis que não serão penalizados. Conseqüentemente:

ExtraRoom(hotelType,day)=0,
se *hotel(hotelType,day) < maxHotel(hotelType,day)*

ou

ExtraRoom(hotelType,day)=hotel(hotelType,day)-maxHotel(hotelType,day), se *hotel(hotelType,day)* > *maxHotel(hotelType,day)*

O *HomogeneityFunc* modela o risco de acumular muitas diárias para um dado dia e tipo de hotel. Essa função penaliza as alocações que possuem muitas diárias de hotéis de um mesmo tipo e em um mesmo dia.

O Alocador também é responsável por calcular o valor máximo do lance que pode ser dado para um determinado bem. O Ordenador decide os bens a serem comprados, e pede para o Alocador calcular esse valor máximo do lance para cada bem escolhido. Esse valor máximo será utilizado pelos negociadores como um teto do lance na negociação. Seja *G*(g)* o resultado da função objetivo da alocação ótima assumindo que o bem *g* está dentro desta alocação. Para calcular o valor máximo do lance, o *solver* é executado com o preço de *g* igual a $p'_g = \infty$ e gera o resultado *G*(g)'*. O valor máximo do lance para o bem *g* é dado pela equação *G*(g) - G*(g)'*.

5.5.3.Ordenador

A alocação ótima e as alocações sub-ótimas são calculadas uma vez por minuto pelos agentes Alocadores. O Ordenador executa uma heurística baseada nessas alocações para definir os bens que devem ser comprados. Para cada cliente, uma alocação contém as seguintes informações: (i) um data de chegada *AA*, (ii) uma data de saída *AD*, (iii) um tipo de hotel *hotelType*, e (iv) um pacote de ingressos de entretenimento *e(day)*.

Seja *G** a alocação ótima e *G(n)* as alocações sub-ótimas, onde *n* é o índice das alocações sub-ótimas geradas e *n* assume um valor entre 1 ... *N*. O Ordenador envia ordens de compra para os negociadores baseados nas seguintes regras:

i. Enviar ordem de compra para passagem de ida no dia *AA* para o cliente baseado na alocação *G**, se e somente se as duas condições abaixo são verdadeiras:
    a. *a* > *p.N*, onde *a* é o número de vezes que a passagem de ida no dia *AA* aparece em *G(n)*, e *p* é uma taxa.

b. O preço previsto indica que o preço do leilão está subindo.

ii. Enviar ordem de compra para passagem de volta no dia *DD* para o cliente baseado na alocação *G**, se e somente se as duas condições abaixo são verdadeiras:
   a. $b > p.N$, onde b é o número de vezes que a passagem de volta no dia *DD* aparece em *G(n)*, e p é uma taxa.
   b. O preço previsto indica que o preço do leilão está subindo.

iii. Enviar ordem de compra para diária do tipo *hotelType* no dia *day* em *G** para cada cliente.

iv. Enviar ordem de compra para ingressos de entretenimento do tipo *e(day)* em *G** para cada cliente.

Essa heurística utiliza *G** e *G(n)* para definir os bens que devem ser comprados considerando a incerteza do mercado. Se uma passagem aérea em *G** não está presente em pelo menos p por cento das alocações em *G(n)*, então a decisão de compra é postergada para o próximo minuto quando os Alocadores re-calculam as alocações novamente.

5.5.4. Negociador de Hotel

O Negociador de Hotel utiliza a técnica Minimax (Russell et al., 1995) e aprendizado por reforço (Mitchell, 1997) para calcular lances de leilões de hotéis. O Minimax é uma técnica de busca para dois jogadores, utilizada pelo Negociador de Hotel para encontrar lances ótimos. O lance ótimo é aquele que permite a compra do bem desejado pelo menor preço possível. Há dois jogadores no Minimax: o MAX e o MIN. O Negociador de Hotel é modelado como o jogador MAX, e a resposta do mercado é modelada como o jogador MIN. Uma busca em profundidade é feita a partir de uma árvore onde a raíz é a posição corrente do jogo. As folhas dessa árvore são avaliadas pela ótica do jogador MAX, e os valores dos nós internos são atribuídos de baixo para cima com essas avaliações. As folhas de o nível minimizar são preenchidas com o menor valor de todos os seus nós filhos, e o nível de maximizar são preenchidos com o maior valor de todos os nós filhos. Essa técnica é utilizada como uma técnica de poda chamada Alpha-Beta (Russell et al., 1995).

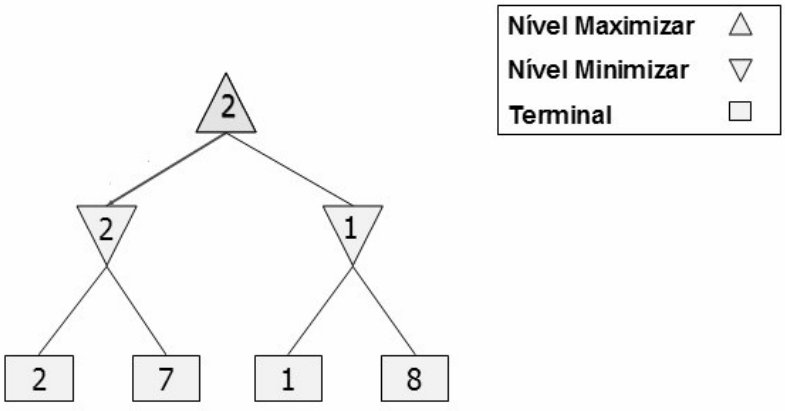

Figura 50: A árvore Minimax.

Para construir essa árvore de decisão no Negociador de Hotel, os estados de cada nó foram modelados com as seguintes informações:

i. *AskPrice*: o preço corrente do leilão;
ii. *LastAskPrice*: o preço do leilão do ultimo minuto;
iii. *deltaAskPrice = AskPrice – LastAskPrice*;
iv. *Gama*: uma constante;
v. *Bid*: o valor atual do ultimo lance;
vi. *LastBid*: o valor do lance enviado no ultimo minuto;
vii. *t* : o tempo do jogo.

A cada nó de um nível de maximizar, os nós filhos são atualizados com as seguintes alternativas:

(i) $Bid = LastBid + 0.5*Gama*deltaAskPrice$;

(ii) $Bid = LastBid + 2*Gama*deltaAskPrice$;

(iii) $Bid = LastBid + 5*Gama*deltaAskPrice$.

O valor de *LastBid* é atualizado com o último valor de *Bid*. Os valores de *AskPrice*, *LastAskPrice*, *Gama*, e do tempo *t* nesses filhos de MAX são mantidos com o mesmo valor. A figura 51 a ilustra esse processo de atualização.

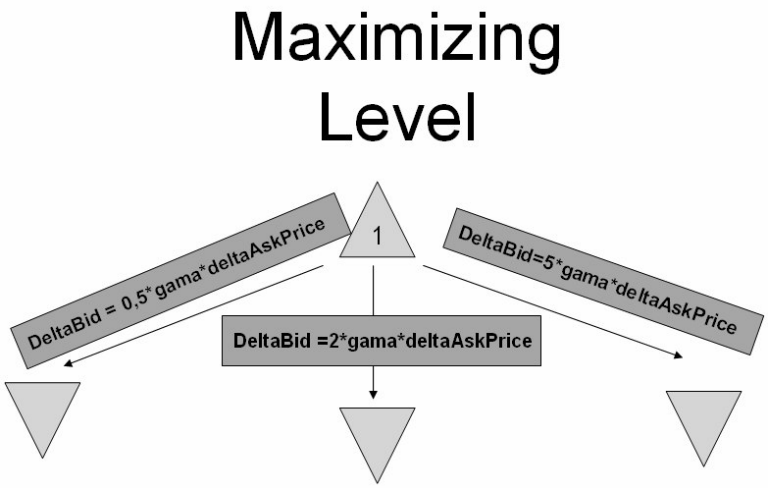

Figura 51: A árvore Minimax (nível Maximizar) no Negociador de Hotel.

A cada nó de um nível de minimizar, os nós filhos são atualizados com as seguintes informações:

(i) $AskPrice = AskPrice + 0.5*Gama*deltaAskPrice$;

(ii) $AskPrice = AskPrice + 2*Gama*deltaAskPrice$;

(iii) $AskPrice = AskPrice + 5*Gama*deltaAskPrice$.

O valor de *LastAskPrice* é atualizado com o último valor de *AskPrice*. Os valores de *Bid*, *LastBid*, e *Gama* nos filhos de MIN são mantidos com o mesmo valor. O tempo *t* é incrementado com um minuto nesta etapa. A figura 52 ilustra esse processo de atualização.

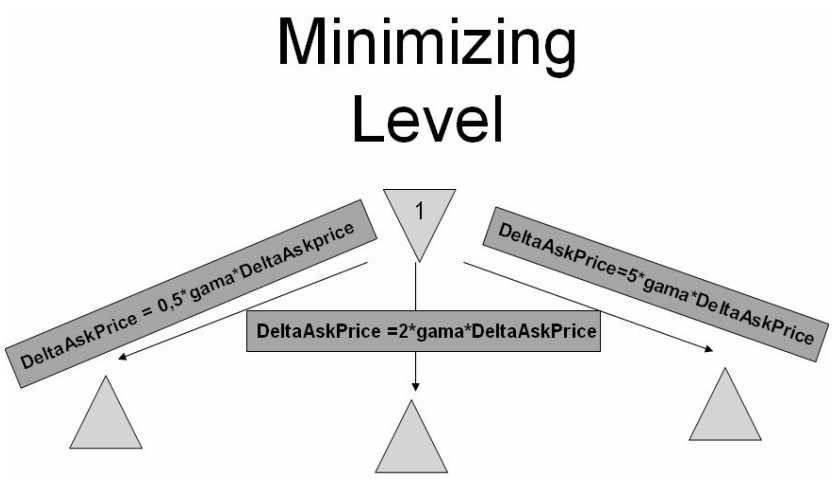

Figura 52: A árvore Minimax (nível Minimizar).

O Negociador de Hotel utiliza uma função avaliação na árvore Minimax para avaliar estados intermediários. Essa função utiliza uma rede neural chamada Perceptron (Mitchell, 1997) com um neurônio artificial. As entradas do Perceptron são os estados de um nó da árvore Minimax, e a saída é um valor entre 0 e 1. Os estados avaliados com valores próximo de 1 são considerados bons estados, e conseqüentemente representam que o lance enviado consegue adquirir o bem desejado. O objetivo do Negociador de Hotel é adaptar os pesos do Perceptron para que esses estados finais representem lances ótimos.

O maior problema é configurar os pesos do Perceptron para que a avaliação leve apenas a casos de sucesso e onde o lance é ótimo. A técnica de aprendizado utiliza reforço combinado com a regra do Perceptron para atualizar os pesos. Ao invés do aprendizado supervisionado (Russell et al. 1995) onde um conjunto de treinamento é apresentado ao Perceptron. O Negociador mantém um histórico de todos os nós por onde passou para implementar essa estratégia. A regra do Perceptron é utilizada em todos esses nós armazenados assim que o leilão fecha:

$w_i = w_i + \eta \, . \, (\, y(t) - d(t) \,) \, . \, x_i$ (Regra do Perceptron)

onde:

w_i é o i-ésimo peso do Perceptron;

x_i é a i-ésima entrada do estado da arvore Minimax;

$y(t)$ é o valor atual da saída da função avaliação;

$d(t)$ é o valor esperado da saída da função avaliação que é calculado com o aprendizado por reforço;

η é a taxa de aprendizado.

O primeiro nó utilizado pela regra do Perceptron é a folha, que representa o último lance enviado antes do leilão fechar. O Negociador atinge um estado bem sucedido, se todos os bens são comprados conforme a ordem de compra enviada pelo Ordenador e a diferença entre o valor do lance e o preço corrente do bem no leilão não é superior a uma constante k. O estado considerado bem sucedido recebe um $d(t) = 1$. Em todos os outros casos $d(t) = 0$.

Todos os estados por onde o Negociador passou também serão utilizados pela regra do Perceptron num sentido de baixo para cima na árvore. A regra do aprendizado por reforço usada para calcular *d(t)* é igual a:

$d(t-\Delta t) = d(t) + \beta . (reward(t) - d(t))$,

No caso do nó folha ser considerado bem sucedido.

$d(t-\Delta t) = d(t) + \beta . d(t)$,

No caso do nó folha NÃO ser considerado bem sucedido

onde:

d(t) é o valor esperado de saída no instante *t*;

ß é a taxa do aprendizado por reforço;

reward(t) = 1 – (Bid(t) – AskPrice(t)) / AskPrice(t) é a recompense obtida no instante *t*.

Total number of requested hotel goods from the Ordering Agent	**Total number of hotel goods acquired by Hotel Negotiators**	**Percentage (%)**
1597	1515	94,87

Tabela 14: Performance do Negociador de Hotel nas finais do TAC Classic de 2004.

O Ordenador envia ordens de compra para cada um dos Negociadores de leilão de hotel. Essas ordens de compra possuem o numero de bens desejados e seu respectivo lance máximo que pode ser dado. A tabela 14 apresenta a performance da técnica implementada no Negociador de Hotel para o TAC *Classic* 2004. Ordenador enviou 1597 ordens de compra nas 35 partidas da final, e o Negociador foi bem sucedido em 94,87% dessas ordens.

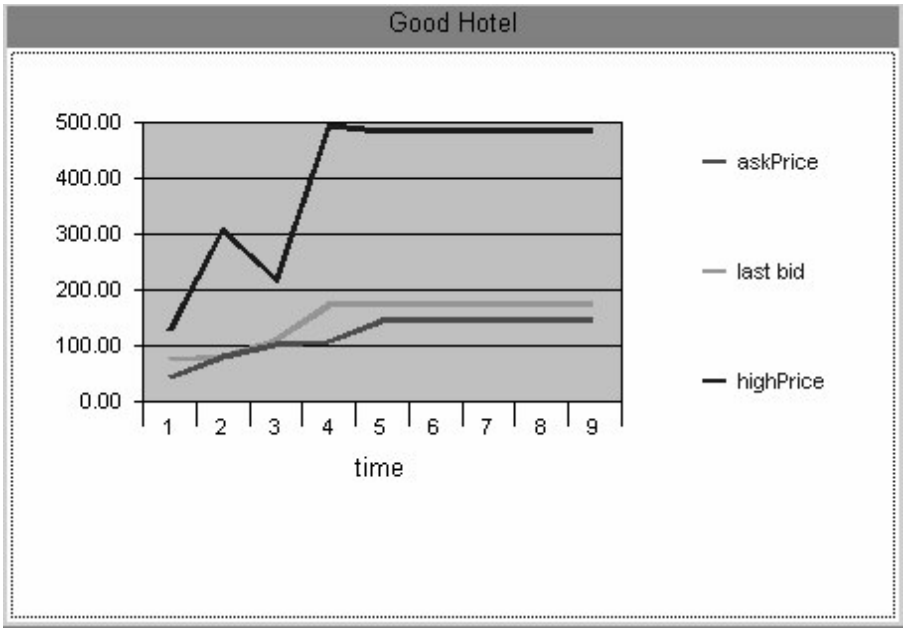

Figura 53: Gráfico dos lances enviados para leilão pelo Negociador de Hotel.

Figura 53 apresenta um gráfico com os lances enviados para o leilão do hotel BOM do dia 1 no jogo 6089 (finais do TAC *Classic* 2004). Os lances enviados estão sempre próximos do preço valor do bem. Isso garante que o bem esteja sendo adquirido. Porém, o lance enviado não pode exceder o valor do lance máximo definido pelo Ordenador.

5.6. Utilizando o MAS-School no Processo de Desenvolvimento do LearnAgents

Na primeira fase do MAS-School, o engenheiro de software deve definir dois elementos centrais que estão relacionados com aprendizado: (i) *Objetivo Sistêmico*, SG, e (ii) *Medida de Performance Sistêmica*, SP, que mede o ganho de performance do sistema. No *LearnAgents*, o *Objetivo Sistêmico* foi definido utilizando a técnica orientada a objetivos da figura 39. Conseqüentemente, (i) SG: comprar pacotes de viagens para os clientes e obter o maior lucro possível, e (ii) SP: A pontuação média representando o lucro médio obtido do sistema no ambiente TAC.

Na segunda fase, os tipos de agentes definidos na tabela 12 são selecionados para utilizar técnicas de *machine learning*. O objetivo é estabelecer um *Problema*

de Aprendizado do Agente bem definido. Conseqüentemente, três características são definidas: (i) *Objetivo do Aprendizado*, G; (ii) *Medida de Performance*, P, que mede a melhora da performance no agente individualmente; e, (iii) uma *Experiência de Treinamento*, E, que define o processo de aquisição do conhecimento no agente com o aprendizado. No *LearnAgents*, o Preditor de Preços foi um dos agentes selecionados para utilizar um técnica de *machine learning*. O *Problema de Aprendizado do Agente* para o Preditor pode ser definido por:

(iv) G: Prever os preços de leilões de hotéis;

(v) P: O erro entre o preço previsto e o preço real; e

(vi) E: Utilizar um histórico de preços dos leilões.

Na terceira fase, um bom *design* do agente é criado para permitir a inclusão ou reuso de várias técnicas de *machine learning*. Além disso, o *design* deve proporcionar uma boa manutenção de código. A figura 54 apresenta o digrama de Classes (UML) do Preditor de Preços utilizando o padrão de projeto orientado a objetos da seção 4.2.

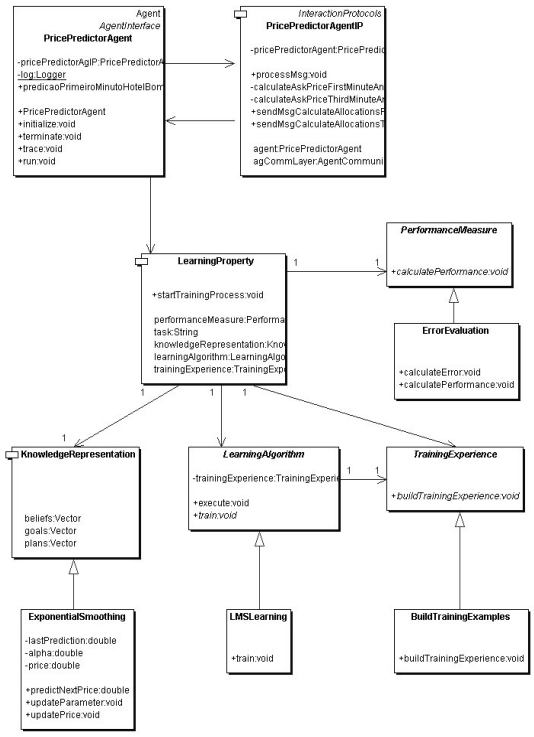

Figura 54: O diagrama de Classes (UML) do Preditor de Preços.

As decisões mais importantes na fase de implementação de uma técnica de *machine learning*, nos agentes selecionados são: (i) a representação do conhecimento; (ii) o algoritmo de aprendizado; (iii) o conjunto de treinamento utilizado pelo algoritmo de aprendizado. O conhecimento do Preditor de Preços foi modelado como uma função chamada *NextAskPrice*. Essa função recebe um preço do leilão *A* e gera uma preço futuro do leilão *N* (*NextAskPrice*: $A \rightarrow N$). A representação aproximada da função *NextAskPrice* utiliza uma função para calcular o preço futuro do leilão: $PredictedAskPrice(n) = \alpha * AskPrice(n-1) + (1 - \alpha) * PredictedAskPrice(n-1)$, onde α é um numero entre 0 e 1; e *n* é o *n*-ésimo jogo. Esta formula é codificada na classe *ExponentialSmoothing* da figura 54. O algoritmo de aprendizado Least Mean Squares (LMS) é utilizado para adaptar o α: $\alpha(n) = \alpha(n-1) + \beta * (AskPrice(n-1) - PredictedAskPrice(n-1))$, onde β é a taxa de aprendizado. Esse algoritmo é codificado na classe *LMSLearning* da figura 54. A classe *BuildTrainingExamples* da figura 54 codifica a consulta ao banco de dados com preços de leilões que o agente já participou. A classe *ErrorEvaluation* da figura 54 implementa a avaliação da performance.

Ao invés de desenvolver o sistema multi-agente com todos os agentes inteligentes em uma única fase, a fase de implementação utiliza uma proposta incremental para facilitar a integração e análise da performance dos agentes. A primeira versão do *LearnAgents* foi desenvolvida apenas com agentes de software reativos. O principal objetivo desta versão é testar a comunicação entre os agentes e a execução do sistema no ambiente do TAC. A pontuação média foi medida para poder avaliar a evolução do *benchmark*, mesmo sabendo que o sistema não apresentaria boa performance. A tabela 15 mostra o *benchmark* feito com as várias versões do *LearnAgents*.

Cada teste para avaliar a performance do sistema é feito em um simulador da competição real contra sete jogadores chamados *dummies* (TAC). O sistema joga cem partidas no *benchmark* para avaliar a performance média, e minimizar o efeito da variação das preferências dos clientes. O *download* do simulador da competição pode ser feito no endereço *http://www.sics.se/tac*.

LearnAgents Version	Agent Selected / Intelligent Module	Average Score	Games Played
0.0	---	-1855	100
1.0	Allocator / Solver – Integer Programming Model, Ordering Agent / Heuristic	1372	100
2.0	Hotel Negotiator / Minimax, N.N. and Reinforcement Learning	1752	100
3.0	Price Predictor (hotel) / Moving Average	2224	100
4.0	Allocator (Ticket) /Solver - Integer Programming Model, Ordering Agent / Heuristic	2856	100
5.0	Allocator / Solver – Integer Programming Model (second version)	3370	100
6.0	Price Predictor (flight) / Maximum Likelihood	3705	100

Tabela 15: Benchmark das Versões do LearnAgents.

O primeiro agente selecionado foi o Alocador, versão 1.0 da tabela 15. Esse agente calcula diferentes cenários baseados nos preços da base de conhecimento do sistema. Esses cenários são alocações de bens para montar pacotes de viagens, e são calculados usando um *solver* de programação inteira. Essa versão da formulação calculava apenas alocações com quantidades de hotéis e passagens aéreas. O *solver* não é executado apenas para encontrar a melhor alocação de bens, mas o maior número de boas alocações geradas em 25 segundos. Cada alocação define o número de bens por cliente que devem ser compradas em cada leilão com a sua respectiva utilidade. Após testar o Alocador com os casos de testes, o Ordenador também foi modificado com uma nova heurística antes de testar o sistema no simulador. O Ordenador executa uma heurística baseada nessas alocações para definir os bens que devem ser comprados.

O Negociador de Hotel foi o próximo agente selecionado para receber um módulo de inteligência. O agente utiliza a técnica Minimax, uma Rede Neural e Aprendizado por Reforço para calcular lances de leilões de hotéis. O principal objetivo desse agente é encontrar lances ótimos e permitir a compra do bem desejado pelo menor preço possível. O agente adapta os pesos da Rede Neural com Aprendizado por Reforço para encontrar estados que gerem lances ótimos. A

versão 2.0 da agência com o Negociador de Hotel inteligente aumentou a pontuação média para 1752.

O agente selecionado na versão 3.0 do sistema foi o Preditor. O agente utiliza uma série histórica dos preços de hotéis para prever o futuro valor dos preços dos leilões. O agente Preditor utiliza uma técnica chamada Exponencial Suavizada (Bowerman et al., 1993), e permite ao Alocador usar os preços previstos para calcular as alocações de bens. A versão 3.0 aumentou a pontuação média para 2224.

Na versão 4.0 do sistema, o Alocador foi novamente selecionado para alterar a formulação da programação inteira. Além de calcular a quantidade de hotéis e passagens aéreas, o *solver* calcula também a quantidade de ingressos de entretenimento. O Ordenador teve que ser alterado para tratar a compra dos ingressos, pois nas versões anteriores os ingressos não eram negociados. O *benchmark* dessa versão aumentou a pontuação para 2856, porém o sistema começou a apresentar atrasos nas respostas com o servidor do TAC. Isso apresentava clara evidência que os recursos computacionais da CPU estavam baixas. Um Alocador Escravo foi criado para calcular essas alocações de ingressos, e foi executado em outra CPU da rede. O Alocador Escravo usava a mesma formulação de programação inteira para calcular as alocações de ingressos.

O Alocador foi selecionado novamente na versão 5.0 do sistema para incluir na formulação alguns parâmetros, e uma função que penaliza as alocações. Essa função modela o risco de acumular muitas diárias para um dado dia e um dado tipo de hotel. Essa versão aumentou a pontuação média para 3370.

O último agente selecionado foi novamente o Preditor. Esse agente foi modificado para calcular também a predição dos preços de leilões de vôos. O Preditor utiliza uma técnica chamada Máxima Verossimilhança para prever uma variável aleatória escondida x do processo que atualiza os preços dos leilões de vôos. A predição do preço futuro é calculada com o valor esperado do processo. Nessa versão 6.0, o sistema aumentou a pontuação média para 3705.

A figura 54 apresenta um gráfico com a evolução do *benchmark* do *LearnAgents*. Este gráfico apresenta a evolução da inteligência do sistema multi-agente. Isso demonstra a eficácia do método MAS-School, pois possibilita um

ganho constante na performance da inteligência, com uma técnica simples de modelar e implementar agentes inteligentes com aprendizado.

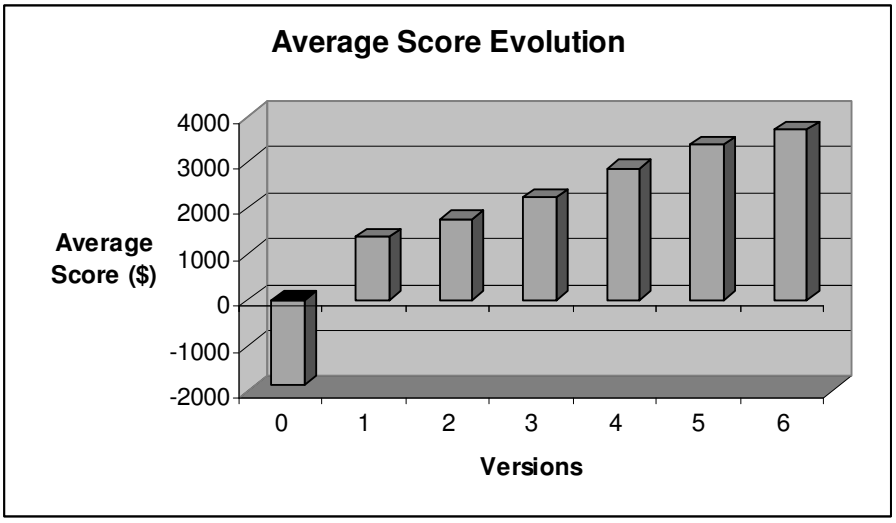

Figura 55: O Gráfico da Evolução do Benchmark do LearnAgents.

A tabela 16 mostra novamente o resultado do *LearnAgents* no TAC Classic de 2004. Um detalhe importante sobre o resultado do *benchmark* no simulador é a semelhança existente entre os resultados do melhor *benchmark* e o resultado da competição. O melhor *benchmark* obteve uma pontuação média de 3705 e a pontuação média da competição foi de 3736. Isso evidencia a boa performance e a alta adaptabilidade dos nossos agentes em ambientes com jogadores diferentes. Isso só é possível graças as técnicas de inteligência e *machine learning* incluídos nos agentes do *LearnAgents*.

Posição	Agente	Pontuação Média
1	Whitebear04	4122.11
2	Walverine	3848.97
3	LearnAgents	3736.62
4	SICS02	3708.24
5	NNN	3665.97
6	UMTac-04	3281.43
7	Agent@CSE	3262.51
8	RoxyBot	2015.02

Tabela 16: Resultado do TAC de 2004.

5.7.Utilizando o ASYNC na Implementação do LearnAgents

No ASYNC, cada agente no sistema deve possuir duas classes concretas. A primeira classe a ser implementada deve herdar da classe *Agent* e implementar a interface *AgentInterface*. Essa classe concreta deve conter métodos que representam as ações internas dos agentes. Os métodos *initialize()*, *run()*, *terminate()*, e *trace()* devem ser implementados nesta classe concreta.

Esse mesmo agente do sistema deve ter outra classe concreta. Essa classe deve herdar da classe *InteractionProtocols*, e deve conter métodos para implementar os protocolos de interação. O método *processMsg()* deve ser implementado para processar todas as mensagens recebidas pelo agente. A referencia para o objeto *AgentCommunicationLayer* é utilizada para que o agente possa utilizar a camada de comunicação. A figura 56 ilustra a instanciação do agente Ordenador na aplicação *LearnAgents*, e a instanciação do formato da mensagem utilizando FIPA ACL (*http://www.fipa.org/specs/fipa00061/*).

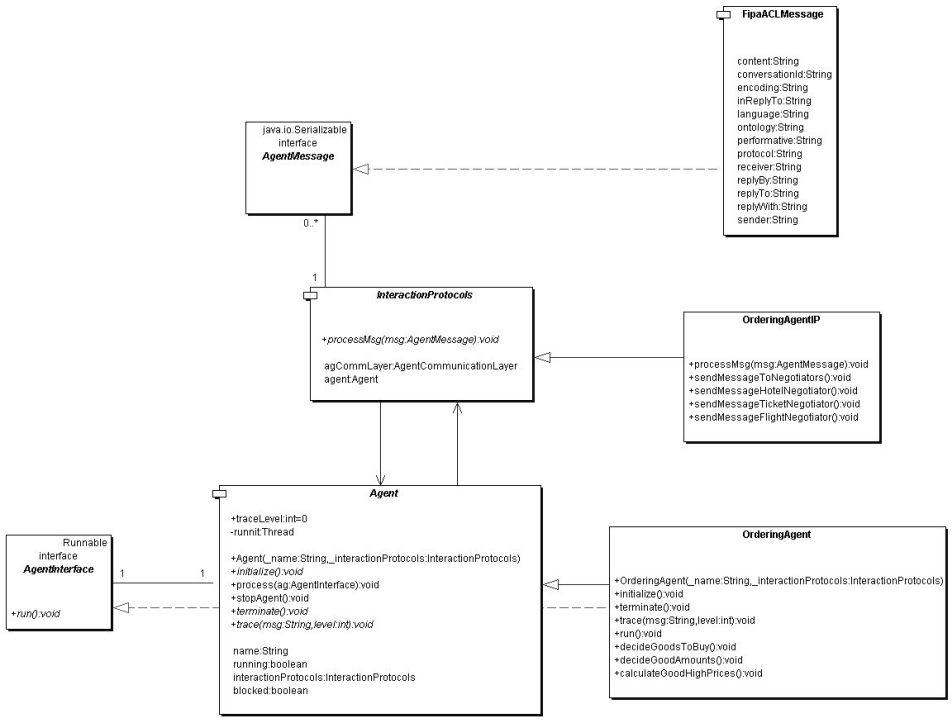

Figura 56: A instanciação do Agente Ordenador e Formato da Mensagem na aplicação. LearnAgents.

As ações internas do Ordenador foram modeladas no diagrama de cenários da figura 57. Essas ações do agente não dependem da interação com outros agentes no sistema. As ações “Coletar as Informações da Base de Conhecimento Corporativa”, “Decidir os bens a serem comprados”, “Decidir as quantidades dos bens”, e “Calcular os lances máximos” não dependem da interação com outros agentes. Conseqüentemente, os métodos *collectInfoCKB()*, *decideGoodsToBuy()*, *decideGoodAmounts()*, e *calculateGoodHighPrices()* da classe *OrderingAgent* modelam as ações descritas acima. O código Java a seguir apresenta o mapeamento das abstrações em ANote para o *framework* ASYNC.

Classify Best Allocations	
Main Agent	Ordering Agent
Preconditions	Message from Allocator Master

Main Action Plan	while true
	Collect allocations in CKB
	for each Negotiator Agent do
	Decide goods to buy
	Decide good amounts
	Calculate good high prices
	Send message to Negotiator Agent
	end do
	end while

Interaction	Hotel Negotiator, Flight Negotiator, Ticket Neg.
Variant Action Plan	

Figura 57: Diagrama de cenário do Ordenador.

```
69.   public class OrderingAgent extends Agent implements
  AgentInterface
70.   {
71.   public OrderingAgent(String name, InteractionProtocols iP){
72.            super(name, iP);
73.   }
74.   public void initialize() {
75.   ...
76.   }
77.   public void terminate() {
78.   ...
79.   }
80.   public void trace(String msg, int level) {
81.   ...
82.   }
83.   public void run() {
84.            int NumNegotiators=3;
85.            while(true) {
86.                collectInfoCKB();
87.                for(int i=0;i<NumNegotiators;i++){
88.                     decideGoodsToBuy();
89.                     decideGoodAmounts();
90.                     calculateGoodHighPrices();
```

```
91.                     ((OrderingAgentIP)getInteractionProtocols()).
92.                     sendMessageToNegotiators();
93.                 }
94.             }
95.     }
96.     public void collectInfoCKB() {
97.     ...
98.     }
99.     public void decideGoodsToBuy() {
100.    ...
101.    }
102.    public void decideGoodAmounts() {
103.    ...
104.    }
105.    public void calculateGoodHighPrices() {
106.    ...
107.    }
108.    }
```

Os protocolos de interação do Ordenador também são modelados pelo diagrama de cenários da figura 57. Os protocolos de interação definem como o agente deve se comunicar e interagir com outros agentes no sistema. A ação "Enviar Mensagem para os Negociadores" exige uma interação com outros agentes no sistema. Conseqüentemente, essa ação deve ser colocada no método *sendMessageToNegotiators()*. Esse método executa os métodos *sendMessageHotelNegotiator()*, *sendMessageTicketNegotiator()*, e *sendMessageFlightNegotiator()*. Esses métodos utilizam o método *sendMsg()* da camada de comunicação para enviar as mensagens para os agentes negociadores.

```
1. public class OrderingAgentIP extends InteractionProtocols
2. {
3.     public void processMsg(AgentMessage msg) {
4.           FipaACLMessage msgReceived = (FipaACLMessage)msg;
5.           ...
6.     }
7.     public void sendMessageToNegotiators(){
8.           sendMessageFlightNegotiator();
9.           sendMessageHotelNegotiator();
10.          sendMessageTicketNegotiator();
11.    }
12.    public void sendMessageHotelNegotiator() {
13.          FipaACLMessage msg = new FipaACLMessage();
14.          getAgCommLayer().sendMsg("HotelNegotiator",msg);
15.    }
16.    public void sendMessageTicketNegotiator() {
17.          FipaACLMessage msg = new FipaACLMessage();
18.          getAgCommLayer().sendMsg("TicketNegotiator",msg);
19.    }
20.    public void sendMessageFlightNegotiator() {
21.          FipaACLMessage msg = new FipaACLMessage();
22.          getAgCommLayer().sendMsg("FlightNegotiator",msg);
23.    }
24.    }
```

O *framework* ASYNC foi fundamental para o desenvolvimento do *LearnAgents*, pois ele possui um mapeamento claro entre abstrações de agentes e código OO, permite re-uso de código e conseqüentemente uma diminuição no tempo de desenvolvimento, e também diminui a complexidade de implementar agentes de software.