Gestão de Elevadores

Relatório Final

Mestrado Integrado em
Engenharia Informática e Computação

Agentes e Inteligência Artificial Distribuída

Grupo T01_4:
António Ramadas - up201303568@fe.up.pt
Duarte Pinto - up201304777@fe.up.pt
Gustavo Silva - up201304143@fe.up.pt

Faculdade de Engenharia da Universidade do Porto

17 de Dezembro de 2016

Conteúdo

1 Objetivo

1.1 Descrição do cenário

Implementou-se um sistema para a gestão eficiente de elevadores num edifício, em que cada um é representado por um agente num sistema multi-agente.

Os agentes (elevadores) comunicam entre si informação relevante. Quando é necessário tomar uma decisão, o elevador pode atender ou não uma chamada dependendo do seu estado (andar onde se encontra, direção que está a seguir e peso transportado) e de informação sobre o estado de outros elevadores. Desta forma, os agentes podem combinar ações conjuntas para satisfazer mais rapidamente os pedidos dos utentes.

O programa permite a configuração dos seguintes parâmetros por parte do utilizador:

- número de pisos do edifício;
- número de elevadores;
- carga máxima de cada elevador.

Cada elevador demora um certo tempo a ir de um piso a outro, assim como em paragens para entrada e/ou saída de utentes.

São considerados métodos diferentes de chamada do elevador:

- dois botões de chamada (subir/descer);
- teclado para indicação do piso destino.

1.2 Objetivos do trabalho

O objetivo deste trabalho consiste em comparar o desempenho deste sistema multi-agente usando diferentes estratégias de cooperação com um sistema tradicional onde cada elevador possui uma estratégia fixa e individual: atende o pedido o elevador que se encontra mais próximo do piso onde a chamada foi efetuada.

2 Especificação

2.1 Identificação e caracterização dos agentes

2.1.1 Arquitetura

Serão considerados dois tipos de agentes: elevador e edifício. Existe apenas um agente do tipo edifício, que é o responsável por simular as chamadas dos utilizadores do sistema e alocar esses pedidos a elevadores.

Cada elevador tem conhecimento de:

- Lista de paragens que vai fazer;
- Posição;
- Estado (subir, descer, entrada/saída de pessoas ou em espera);
- Capacidade atual.

2.1.2 Comportamento e estratégias

Uma das estratégias comportamentais que implementamos é um algoritmo tradicional bastante básico segundo o qual um pedido é satisfeito pelo elevador que se encontra mais próximo do sítio onde foi chamado. Nesta situação, a chamada é feita carregando num dos botões de um dispositivo com um botão de subida e um de descida. De resto, o algoritmo é semelhante ao utilizado para determinar o movimento da cabeça de um disco rígido ao servir pedidos de leitura e escrita no mesmo (*LOOK disk scheduling algorithm*) [1]. A ideia é que o elevador continue a deslocar-se no mesmo sentido até que esteja vazio, parando apenas para a entrada e saída de utilizadores que pretendam seguir na mesma direção.

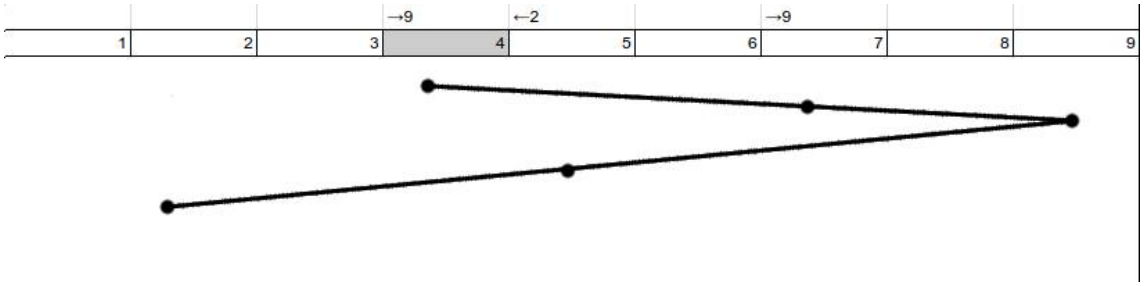


Figura 1: Algoritmo baseado no *LOOK disk scheduling algorithm* [1] - O elevador começa no piso 4 e vai para o 9. Assim que sai, o elevador é chamado no piso 5 e 9. Como o cliente no piso 5 pretende ir no sentido contrário aquele que o elevador está a fazer no momento, o elevador não pára no piso 5. Como o cliente no piso 7 pretende ir no mesmo sentido que o elevador e até para o mesmo destino, o elevador pára no piso 7. Quando chega ao piso 9 e deixa os dois clientes, o elevador desce para apanhar o cliente no piso 5 uma vez que o seu pedido ainda não foi atendido, e leva o cliente do piso 5 ao seu destino.

Ainda outra estratégia comportamental que implementamos é a de um algortimo também bastante simples que consiste em atribuir o pedido ao elevador que se encontra mais perto independentemente do seu sentido atual (subir ou descer). Como podemos facilmente concluir um pedido pode demorar bastante tempo a ser atendido, pois num grande edifício um pedido do piso mais baixo até ao topo pode demorar bastante tempo a ser concluído caso hajam vários pedidos no meio. Contudo, em certos casos esta solução pode revelar-se mais económica e rápida que o algoritmo acima. Veja-se o exemplo: um elevador está no piso 1 e está a seguir até ao topo (3º andar), mas surge um pedido no piso 0 para subir. No algoritmo acima, o elevador percorreria os seguintes pisos: 1->2->3->2->1->0->1->2 (total de 8). Neste algoritmo, o elevador percorreria: 1->0->1->2->3 (total de 5). Assim, neste caso haveria uma redução da travessia em 37.5%.

Uma outra estratégia comportamental que propomos é a mais complexa seria a que faria uso da indicação do piso aquando da chamada do elevador. Este algoritmo visa agrupar no mesmo elevador os pedidos cujo destino sejam próximos uns dos outros. Em casos específicos este algoritmo revela-se bastante expedito e com um desempenho superior em qualquer um dos outros edifícios. De facto, esta estratégia já se encontra comprovada e a funcionar no Hotel New York Marriott Marquis [6] e um simples exemplo pode ser consultado no folheto promocional [4, p.3].

Por fim, a última estratégia é um algoritmo semelhante ao descrito anteriormente com algumas pequenas diferenças que fazem com que cada elevador fique responsável por uma zona. Se o piso de destino do pedido não estiver na lista de destinos do elevador, então ele não vai atender o pedido. Se a origem não for também da lista de destinos do elevador, o elevador aí já pode ir buscar mas com uma penalização.

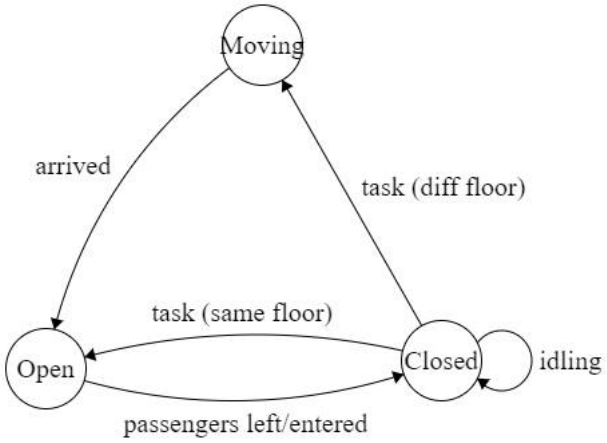
Por fim, apesar da diferença dos algoritmos, quando um elevador não puder atender um pedido que lhe foi atribuído e teve de passar pelo respetivo piso sem parar, a ação desencadeada será a de o elevador encaminhar o pedido para o *building* e depois este inicia um protocolo para decidir qual o novo elevador ao qual esse pedido irá ser alocado.

2.1.3 Máquina de estados

As ações dos elevadores são controladas por uma máquina de estados (Fig. 2) com três estados: em espera (CLOSED), em movimento (MOVING), com as portas abertas (OPEN).

No início da simulação, todos os elevadores encontram-se no estado CLOSED à espera de serem chamados. Assim que ocorre um pedido, o elevador que ficou responsável por realizar a respetiva tarefa passa para o estado MOVING ou para o OPEN, dependendo de se a chamada foi feita no piso em que o elevador se encontra ou noutro piso. No estado OPEN, as portas são abertas por uma certa quantidade de tempo enquanto os utentes saem e entram no elevador. Após esse tempo, as portas são fechadas e o elevador volta ao estado CLOSED, onde passa a executar o resto das tarefas ou se coloca em espera por novas chamadas.

Figura 2: Máquina de estados de um elevador



2.1.4 Processos de raciocínio

2.2 Protocolos de interação

O protocolo de cooperação rede contratual (Fig. 3) é utilizado sempre que houver um novo pedido. O objetivo da utilização deste protocolo é decidir qual o elevador que vai ficar responsável por atender esse mesmo pedido.

Assim que um pedido é recebido, é enviado um *call for proposals* do *building* para todos os elevadores. De seguida, os elevadores respondem com um *propose* no qual incluem o tempo que levarão até à satisfação do pedido, caso este lhes seja alocado. Este tempo tem em conta a lista de tarefas que cada elevador tem na fila de execução em cada momento. Assim que o *building* (*initiator*) receber as respostas de todos os elevadores, escolhe aquele que for capaz de satisfazer o pedido mais rapidamente, enviando-lhe uma mensagem *accept-proposal* e enviando a mensagem *reject-proposal* a todos os outros elevadores. O elevador que ficou responsável por executar a tarefa envia uma mensagem *inform-result* quando a terminar.

Em certas situações, como quando um elevador fica cheio e incapaz de satisfazer novos pedidos até que as pessoas saiam do mesmo, é necessário voltar a correr o protocolo de cooperação rede contratual para determinar novamente qual o elevador que se encontra em melhores condições para satisfazer os pedidos que foram ignorados por ter sido atingida a capacidade máxima. Isto também acontece quando uma nova pessoa que acabou de entrar no elevador marca um piso de destino que ainda não estava marcado, aumentando o tempo de realização do resto das tarefas que estavam agendadas para esse elevador. Nesse sentido, ele finaliza o *Contract Net Interaction Protocol* com uma mensagem do tipo *FAILURE*, pedindo-lhe para voltar a alocar os respetivos pedidos a um novo elevador. Note-se que o elevador agora alocado pode ser o mesmo, embora este vá demorar mais tempo a desempenhar a tarefa, pois terá que deixar sair pessoas antes de satisfazer o pedido.

3 Desenvolvimento

3.1 Plataforma/Ferramenta

Para este trabalho é sugerido o uso de uma das seguintes hipóteses: JADE, Respast+SAJaS ou Jadex. Após alguma pesquisa sobre as diversas ferramentas que permitissem simular o melhor possível um sistema multi-agente, a nossa escolha recaiu sobre a conjugação entre Repast e SAJaS.

Na verdade, o Repast permite a modulação baseada em agentes, isto é, visa criar, analisar e experimentar mundos artificiais populados por agentes. Logo, os elevadores serão agentes e o mundo um edifício. Adicionalmente, um importante fator para o uso desta ferramenta é a existência de uma unidade de tempo (*tick*) que permite atribuir eventos a determinados períodos. Assim, esta medida temporal permite medir o desempenho ao longo do tempo como o tempo de espera desde a chamada de um elevador até à chegada no respetivo tempo. Relativamente ao SAJaS, esta

	pOrig(x)	pDest(x)
x = 0	0.4	0.9
x>0	$\frac{1-pOrig(0)}{numberFloors-1}$	$\frac{1-pDest(0)}{numberFloors-1}$

Tabela 1: Como há mais pedidos do piso 0, pois como é do piso 0 que se entra no edifício para ir para todos os outros pisos, a probabilidade do piso de origem ser o 0 é relativamente maior. Salvo raras excepções, se o piso de origem não for 0, então há uma probabilidade muito grande probabilidade de o piso de destino ser o 0, portanto aumentamos bastante o pDest para x = 0

ferramenta é uma API para simulações baseadas em Jade que permite ocultar a inexistência de uma ponte entre simulação (Repast) e desenvolvimento sistemas multi-agente (Jade).

Apesar do Jade ser uma ferramenta útil para desenvolvimento de sistemas multi-agente através de uma plataforma que permite a gestão de agentes e o uso de contentores, nós optamos por utilizar o SAJaS como API para desenvolvimento de agentes que depois são utilizados em simulações. Já o Jadex foi descartado por estar mais direcionado a agentes BDI (*Beliefs, Goals and Intentions*) que não se adequam ao propósito do tema.

Finalmente, o trabalho foi desenvolvido no Microsoft Windows 10 com recurso ao software de desenvolvimento Repast Simphony for Eclipse.

3.2 Módulos

Com o objetivo de melhor estruturar o programa, este foi dividido nos seguintes *packages*:

- lift_management - Módulo principal do sistema. Contém várias classes importantes, nomeadamente o Launcher e as configurações.
- lift_management.agents - Agentes Lift e Building.
- lift_management.agents.behaviours - Comportamentos (JADE Behaviours) dos agentes do sistema.
- lift_management.algorithms - Contém os diferentes algoritmos de alocação e cooperação implementados, assim como a classe LiftAlgorithm, da qual todos extendem.
- lift_management.calls - Constituído pelos dois tipos de chamadas e sistemas de chamadas (direcional ou indicador de destino).
- lift_management.gui - Contém as classes responsáveis pelos estilos dos agentes (elevadores e edifício), assim como a visualização de algumas estatísticas em tempo real.
- lift_management.models
- lift_management.onto - Responsável pela definição do vocabulário comunicacional e das relações entre os elementos do mesmo.

3.3 Estrutura

3.4 Detalhes relevantes da implementação

3.4.1 Geração automática de pedidos

3.4.1.1 Pisos

A geração dos pisos de origem e de destino foi feita com o objectivo de se aproximar o mais possível da realidade. Numa situação real, a maior parte do fluxo de um elevador passa sempre pelo *ground floor* (piso 0) e como tal a probabilidade das chamadas serem de ou para o *ground floor* é muito maior que para o resto dos pisos (Tabela 1).

3.4.1.2 Pesos

Para simular situações reais com a maior proximidade possível o peso dos humanos são gerados com base numa distribuição normal que reflecte a distribuição de peso da população na vida real (Fig 4) [5].

3.4.1.3 Grupos de pessoas

Por forma a melhor simular o que acontece na realidade, considerou-se que, por vezes, os utentes de um elevador vêm em pequenos grupos. Por este motivo, o gerador automático de pedidos não simula uma pessoa de cada vez, mas sim um grupo de pessoas com a mesma origem e igual intenção a nível de destino, com intervalos de tempo aleatórios entre eles [3]. A fórmula utilizada para simular o tamanho de um grupo, dependendo de um número aleatório $r\epsilon[\,0;1\,]$, é $\lceil(2x)^2\rceil$ (Fig. 5).

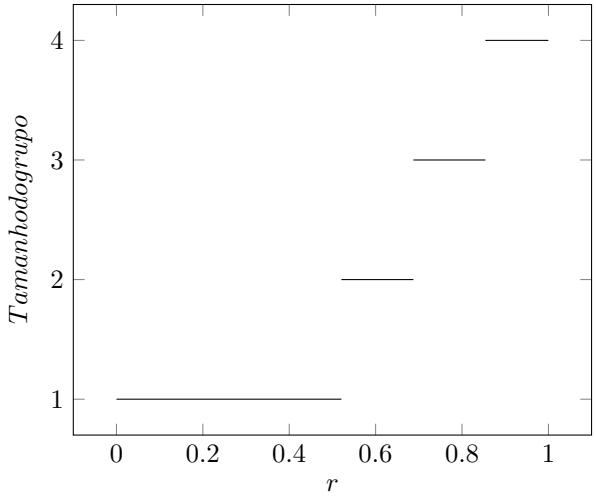


Figura 5: Número de pessoas constituintes do grupo gerado, dependendo de um valor aleatório $r\epsilon[\,0;1]$.

4 Experiências

4.1 Objetivo de cada experiência

Como referido anteriormente, foram utilizados 4 algoritmos diferentes com o mesmo objetivo. De modo a fundamentar melhor a conclusão foram feitas duas execuções de cada algoritmo com diferentes parâmetros. Para cada execução são indicados alguns dados obtidos no fim da execução. Por conseguinte, são indicados os:

- Númeroz de pedidos completos: número de pedidos que os elevadores satisfizeram (conluidos);
- Percentagem de ocupação média: tempo de atividade do elevador / tempo atual de execução
- Tempo médio de espera: tempo de espera de um pedido ser atendido (desde que é feito o pedido até entrar num elevador)

Na primeira execução visou-se uma situação em que o número de elevadores é capaz de satisfazer facilmente os pedidos. Assim, foram utilizados os seguintes parâmetros:

- Número de elevadores: 3
- Número de pisos: 6
- Intervalo de tempo: 50.000 ticks

Na segunda execução visou-se uma situação em que o número de elevadores é incapaz de oferecer facilmente uma boa experiência aos utentes. Assim, foram utilizados os seguintes parâmetros:

- Número de elevadores: 4
- Número de pisos: 25
- Intervalo de tempo: 50.000 ticks

Finalmente, em anexo são incluidos os gráficos obtidos como resultado da primeira execução onde é possível fazer uma análise mais detalhada sobre o progresso dos elevadores ao longo do tempo.

4.1.1 Look Disk Algorithm

Resultados da primeira execução (Fig. 6 e 7):

- Número de pedidos completos: 162
- Percentagem de ocupação média: 66.2%
- Tempo médio de espera: 730 ticks

Resultados da segunda execução:

- Número de pedidos completos: 189
- Percentagem de ocupação média: 84.5%
- Tempo médio de espera: 3425 ticks

4.1.2 Destination Dispatch Algorithm

Resultados da primeira execução (Fig. 8 e 9):

- Número de pedidos completos: 190
- Percentagem de ocupação média: 68.85%
- Tempo médio de espera: 623 ticks

Resultados da segunda execução:

- Número de pedidos completos: 236
- Percentagem de ocupação média: 74%
- Tempo médio de espera: 2330 ticks

4.1.3 Closest Attends Algorithm

Resultados da primeira execução (Fig. 10 e 11):

- Número de pedidos completos: 156
- Percentagem de ocupação média: 50%
- Tempo médio de espera: 782 ticks

Resultados da segunda execução:

- Número de pedidos completos: 137
- Percentagem de ocupação média: 66.75%
- Tempo médio de espera: 4412 ticks

4.1.4 Zoning

Resultados da primeira execução (Fig. 12 e 13):

- Número de pedidos completos: 173
- Percentagem de ocupação média: 83%
- Tempo médio de espera: 835 ticks

Resultados da segunda execução:

- Número de pedidos completos: 76
- Percentagem de ocupação média: 48.6%
- Tempo médio de espera: 4240 ticks

4.2 Resultados

Perante os resultados obtidos torna-se evidente os claros benefícios do algoritmo *Destination Dispatch*. É possível um baixo tempo de espera, um bom uso dos elevadores e um elevado número de pedidos atendidos. Este algoritmo apesar de apresentar melhores resultados também apresenta uma elevada complexidade nos detalhes de implementação. Assim, é possível afirmar que este sistema é superior aos tradicionalmente utilizados.

Relativamente aos outros algoritmos, o *Zoning* é bastante positivo em pequenos edifícios, mas desfavorável em grandes edifícios. O *Closest Attends* é uma implementação *naive* bastante simples e como tal não acarreta resultados práticos positivos quando comparados com os outros algoritmos. Finalmente, o *Look Disk* é o sistema mais comum de todos que apresenta resultados satisfatórios.

5 Conclusões

Com este trabalho o grupo compreendeu melhor a estrutura de um sistema multi-agentes. O sistema de cooperação entre agentes e a definição de estratégias de comunicação revelou que a cooperação entre agentes permite atingir melhor resultados e é mais fácil modelar o raciocínio humano graças à criação de comportamentos.

6 Melhoramentos

Para melhorar a simulação, poder-se-ía criar um novo agente Person, representativo de um utilizador do sistema e das suas ações. Neste caso, seriam também simuladas situações em que as pessoas não chamassem o elevador corretamente, ou em que desistissem de esperar por ele.

7 Recursos

7.1 Referências

[1] http://courses.teresco.org/cs432_f02/lectures/17-files/17-files.html - "Lecture 17 - Disk Scheduling"

[2] Y. Gu, "Multi-objetive Optimization of Multi-Agent Elevador Group Control System Based on Real-time Particle Swarm Optimization Algorithm,"*Engineering*, Vol. 4 No. 7, 2012, pp. 368-378. doi: 10.4236/eng.201247048

[3] Susi, Tuomas, Janne Sorsa, and Marja-Liisa Siikonen. "Passenger Behaviour in Elevator Simulation."Elevatori 34.5 (2005): 28-37.

[4] https://goo.gl/CVlODq - Shindler Miconic 10 brochure

[5] https://www.oswego.edu/šrp/stats/wts_males.htm

[6] https://youtu.be/T6gzm_ifzg8 - "The Smartest Elevators with destination dispatch"

7.2 Software

- Jade 4.4
- Repast Simphony 2.3.1
- SAJaS v0.92b
- Microsoft Windows 10

7.3 Elementos do grupo

O trabalho foi realizado pelo grupo segundo as seguintes percentagens:

- António Ramadas - 33%
- Duarte Pinto - 33%
- Gustavo Silva - 33%

8 Apêndice

8.1 Manual do utilizador

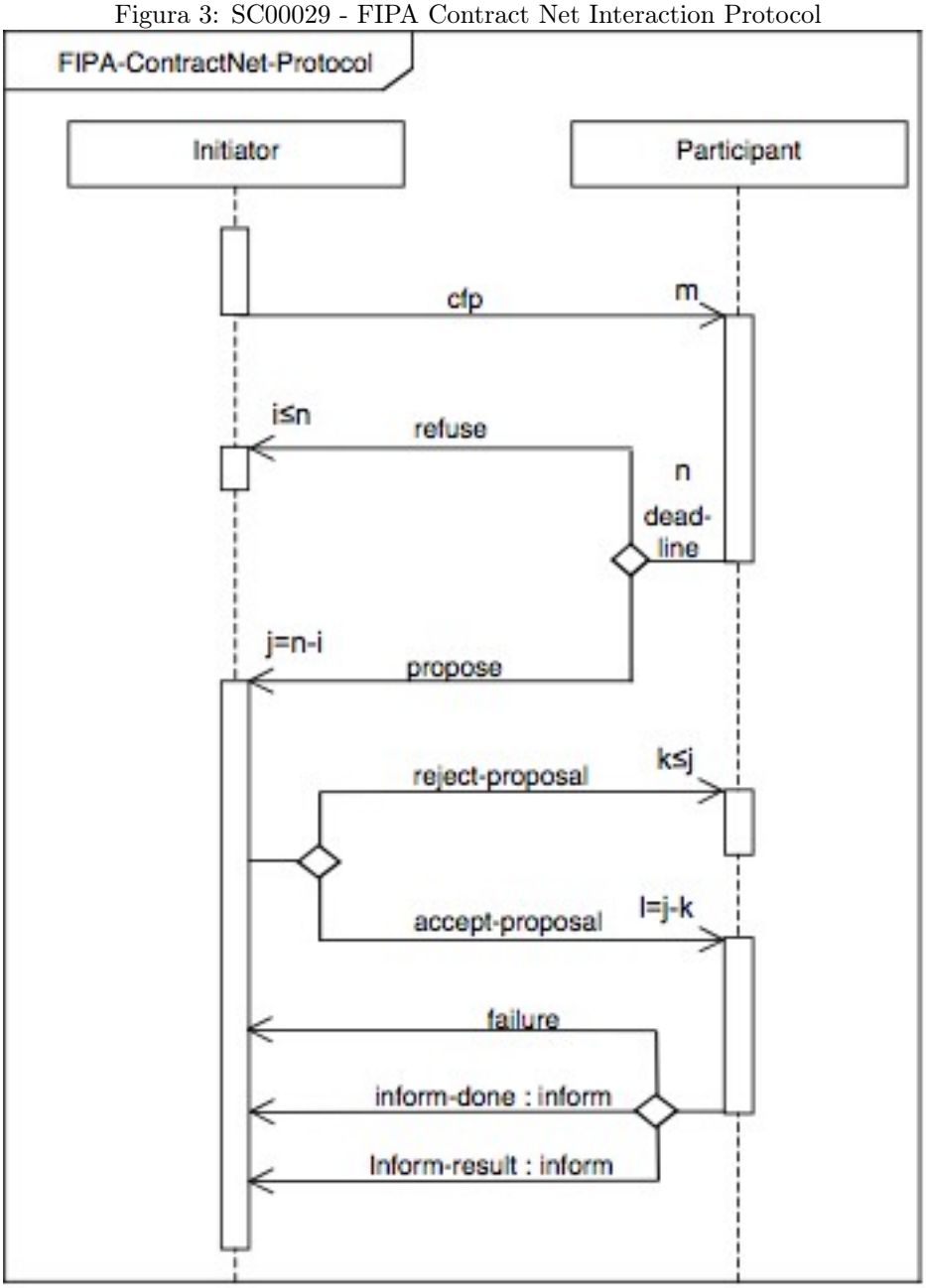
Para iniciar o programa, deve ser executada a função "main"da classe repast.simphony.runtime.RepastMain, passando como argumento o caminho para a pasta "Lift Management.rs". Devem também ser incluídas as bibliotecas que se encontram na pasta "lib".

De seguida, é mostrada a janela do Repast Simphony. Na aba "Parameters", é possível configurar os parâmetros de inicialização (Tab. 2).

Parâmetro	Tipo	Valor por defeito	Descrição
Algorithm	String	LookDisk	Pode ser um de: ClosestAttends, DestinationDispatcher, LookDisk, Zoning.
Call frequency	Integer	150	Maior valor implica uma maior frequência de geração de pedidos.
Default Random Seed	Integer	Aleatório	*Seed* para o gerador de números aleatórios.
Max weight per lift	String	500 450 500	Um valor por cada elevador. Cada valor corresponde ao peso máximo que o elevador pode carregar.
Number of floors	Integer	6	Número de pisos do edifício. Note-se que a numeração dos pisos começa no 0, portanto ter 6 pisos significa que o piso mais alto é o piso número 5.
Number of lifts	Integer	3	Número de elevadores do edifício.

Tabela 2: Parâmetros configuráveis do programa.

Figura 3: SC00029 - FIPA Contract Net Interaction Protocol



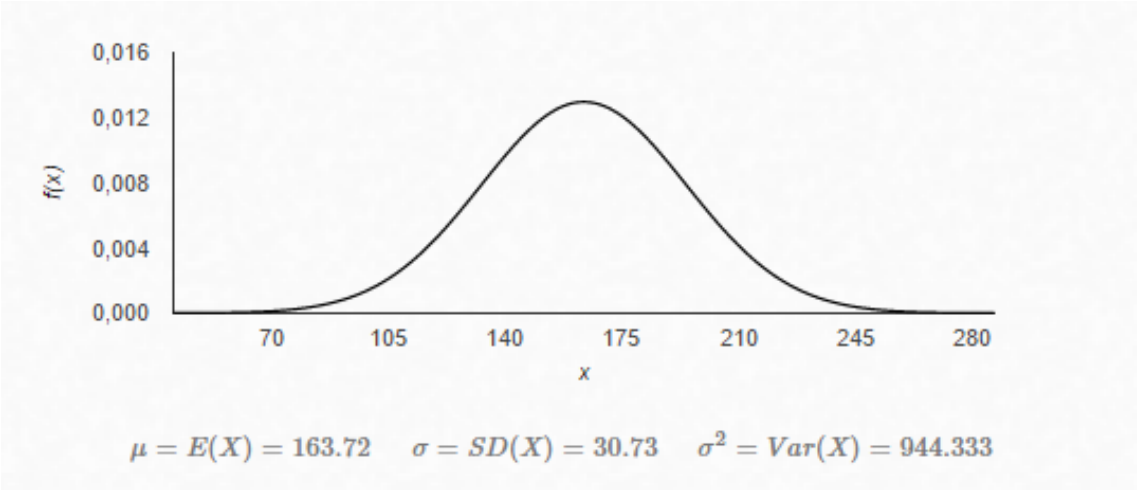
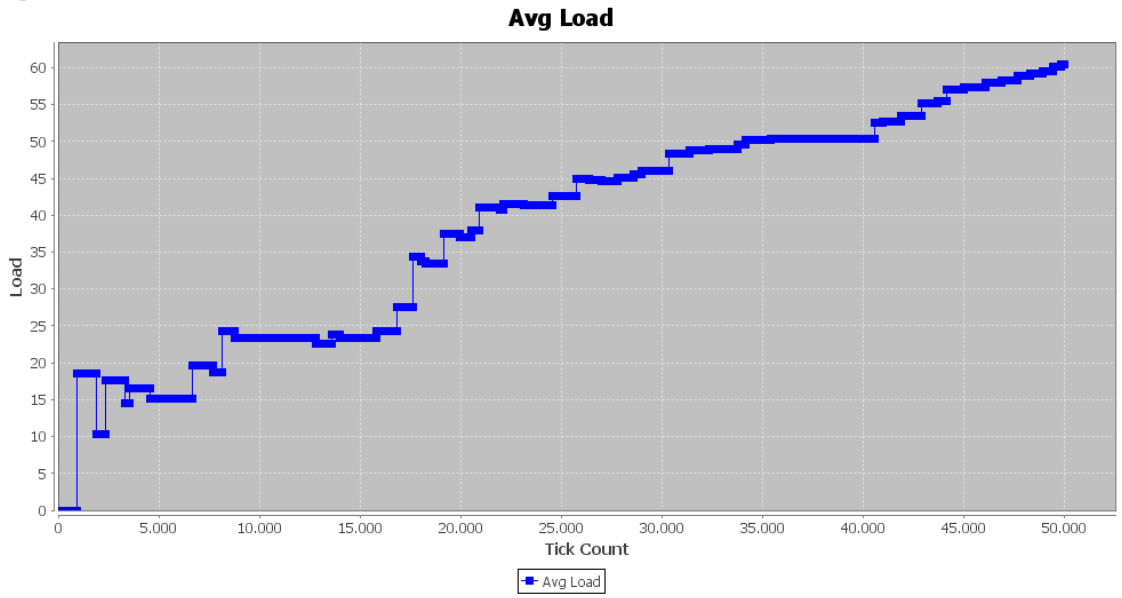


Figura 4: Variação dos pesos dos humanos que são gerados pelo edifício

Figura 6: Look Disk Algoritm: Média da carga atual de cada um dos elevadores ao longo da experiência.



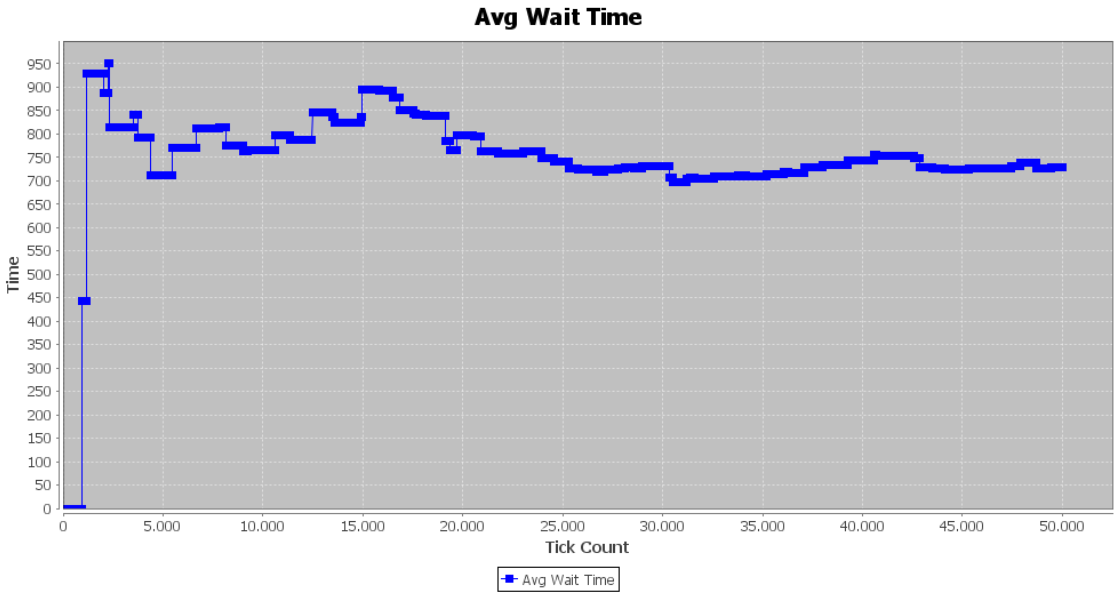


Figura 7: Look Disk Algoritm: Tempo médio de espera dos utentes ao longo da experiência.

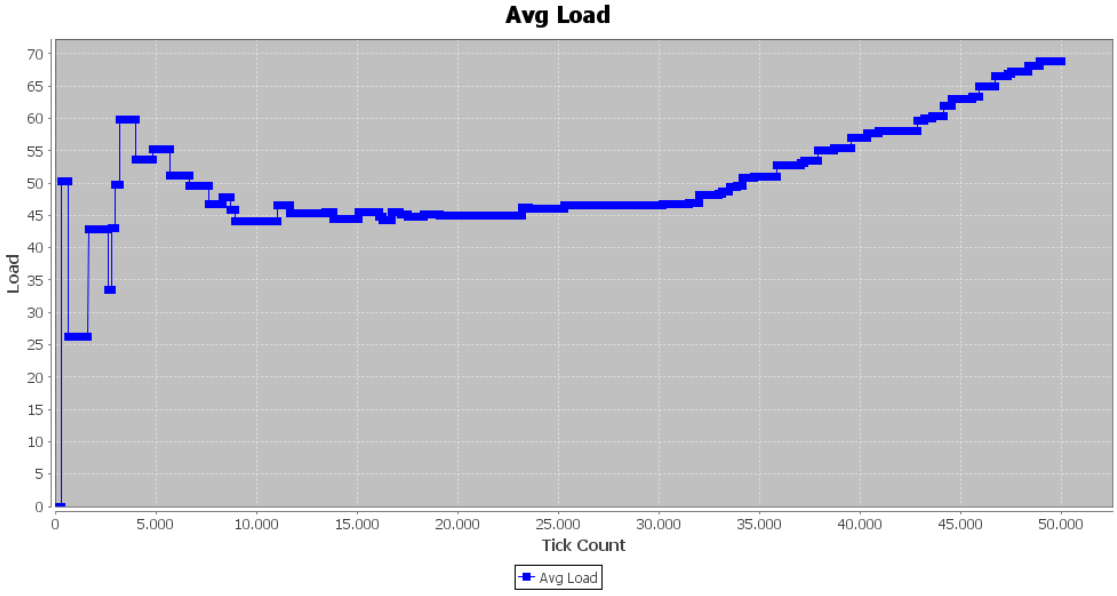


Figura 8: Destination Dispatch Algoritm: Média da carga atual de cada um dos elevadores ao longo da experiência.

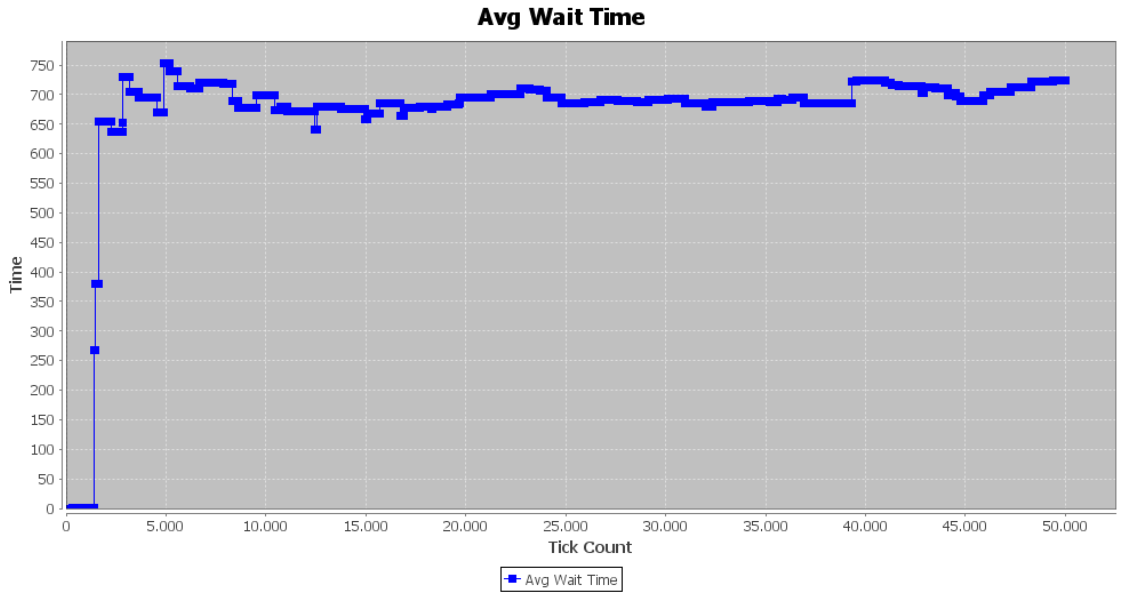


Figura 9: Destination Dispatch Algoritm: Tempo médio de espera dos utentes ao longo da experiência.

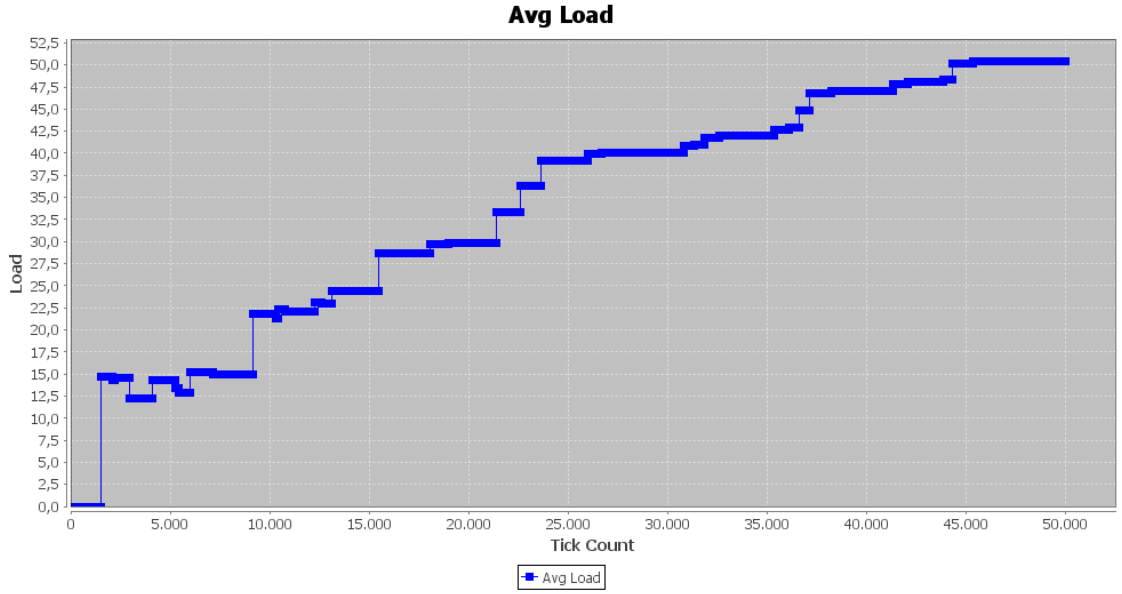


Figura 10: Closest Attends Algoritm: Média da carga atual de cada um dos elevadores ao longo da experiência.

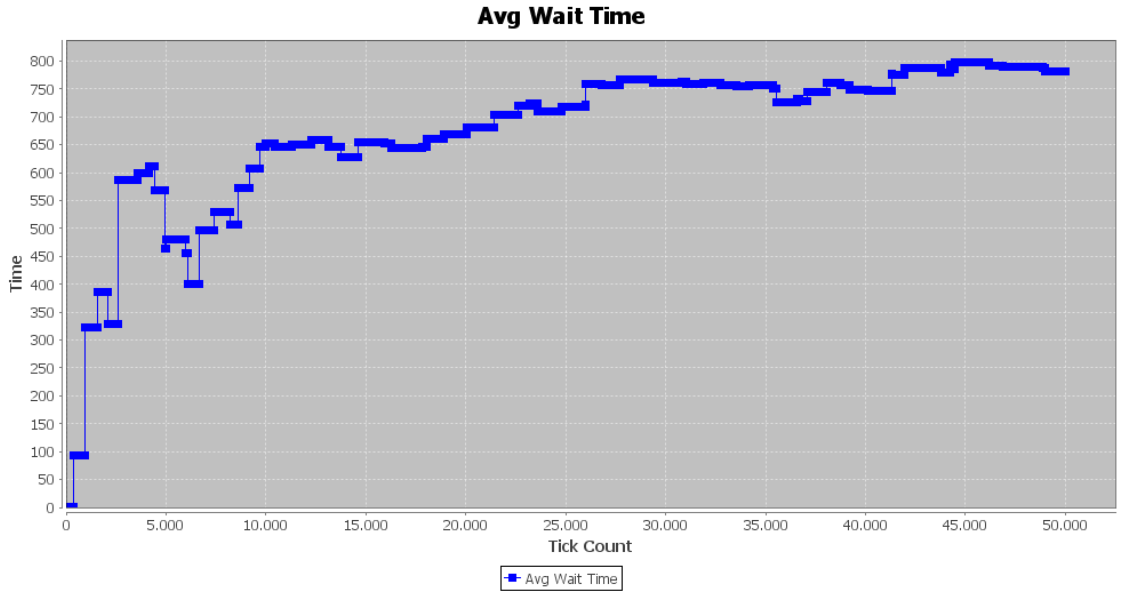


Figura 11: Closest Attends Algoritm: Tempo médio de espera dos utentes ao longo da experiência.

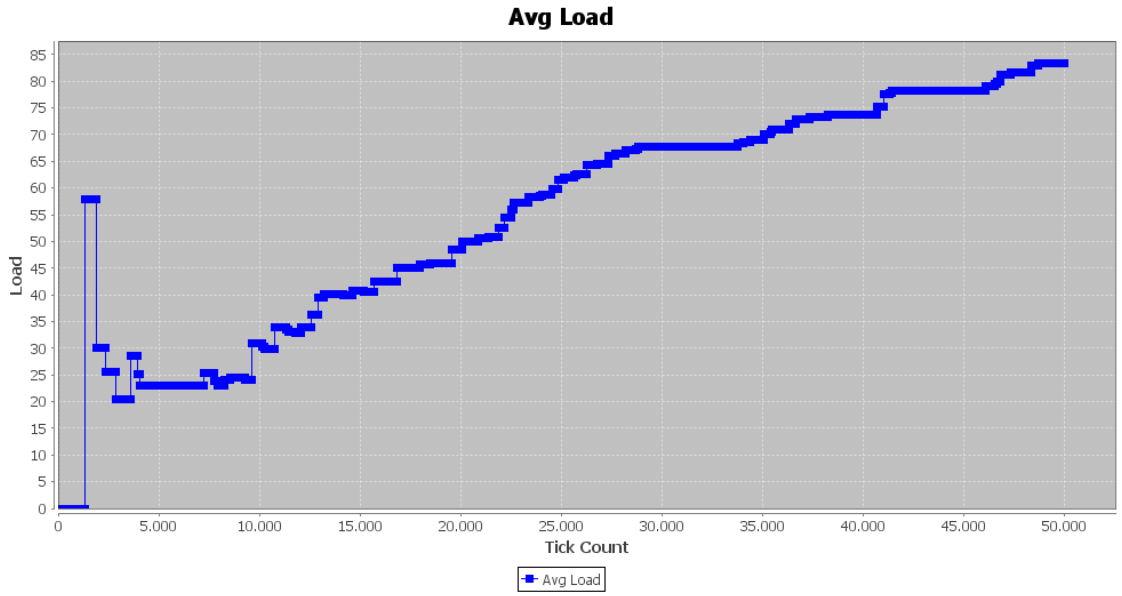


Figura 12: Zoning Algoritm: Média da carga atual de cada um dos elevadores ao longo da experiência.

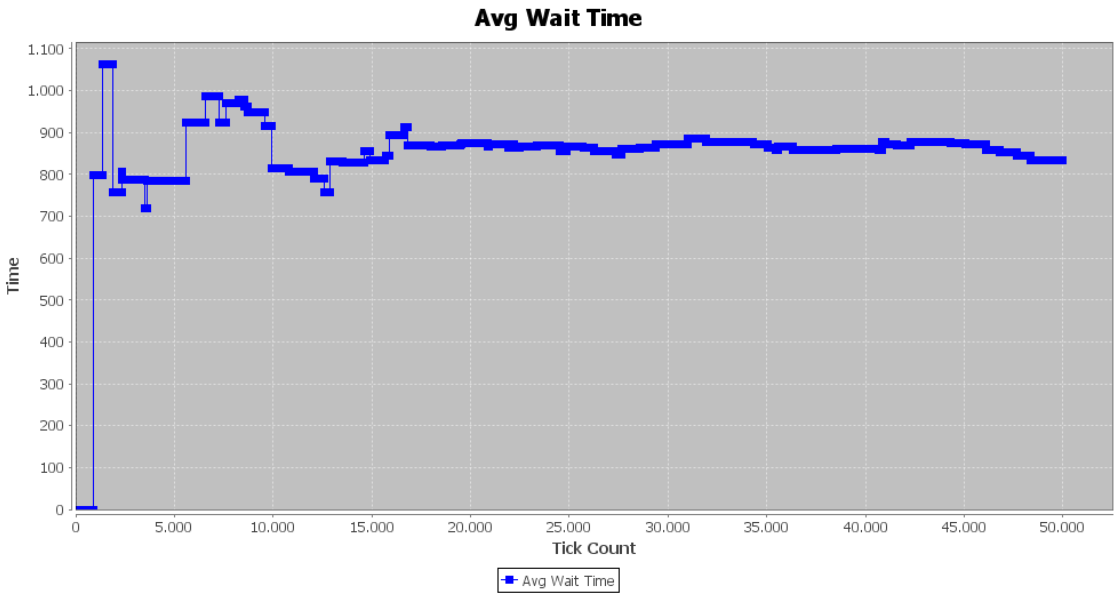


Figura 13: Zoning Algoritm: Tempo médio de espera dos utentes ao longo da experiência.