



Universidade de Brasília - UnB  
Faculdade UnB Gama - FGA  
Engenharia de Software

# **SIMULADOR DE AMBIENTES ACÚSTICOS ORIENTADO A UMA ABORDAGEM MULTIAGENTES**

Autor: Gabriel Augusto Barbosa  
Orientador: Prof. Dr. Maurício Serrano

Brasília, DF  
2015





Gabriel Augusto Barbosa

# **SIMULADOR DE AMBIENTES ACÚSTICOS**

## **ORIENTADO A UMA ABORDAGEM MULTIAGENTES**

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Prof. Dr. Maurício Serrano

Coorientador: Prof<sup>a</sup>. Dr<sup>a</sup> Milene Serrano

Brasília, DF

2015

---

Gabriel Augusto Barbosa

SIMULADOR DE AMBIENTES ACÚSTICOS ORIENTADO A UMA ABORDAGEM MULTIAGENTES/ Gabriel Augusto Barbosa. – Brasília, DF, 2015-

129 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Maurício Serrano

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB  
Faculdade UnB Gama - FGA , 2015.

1. Simulador Acústico. 2. Multiagentes. I. Prof. Dr. Maurício Serrano. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. SIMULADOR DE AMBIENTES ACÚSTICOS ORIENTADO A UMA ABORDAGEM MULTIAGENTES

CDU 02:141:005.6

---

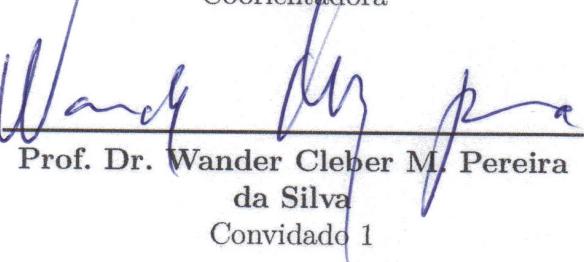
Gabriel Augusto Barbosa

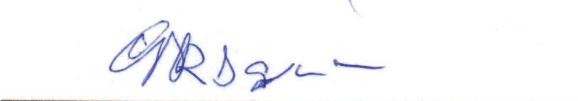
## SIMULADOR DE AMBIENTES ACÚSTICOS ORIENTADO A UMA ABORDAGEM MULTIAGENTES

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Trabalho aprovado. Brasília, DF, 08 de dezembro de 2015:

Maurício Serrano  
Prof. Dr. Maurício Serrano  
Orientador

Milene Serrano  
Profª. Drª Milene Serrano  
Coorientadora  
  
Wander Cleber M. Pereira  
da Silva  
Convidado 1

Carla Silva Rocha Aguiar  
Profª. Drª. Carla Silva Rocha Aguiar  
Convidado 2  


Brasília, DF

2015



*Este trabalho é dedicado às crianças adultas que,  
quando pequenas, sonharam em se tornar cientistas.*



# Agradecimentos

Agradeço primeiramente a Deus, que plantou em mim um sonho que hoje se materializa, e pelos dons a mim proporcionados para que eu atinja meus objetivos nessa caminhada.

Agradeço aos meus pais, Arnaldo Pereira Barbosa e Giselma Augusto de Oliveira, pela atenção e amor, pelo exemplo de garra e perseverança e principalmente, pelos esforços a mim dedicados. Vocês são minha maior inspiração.

Agradeço também pelo imensurável apoio da minha irmã e amiga Ana Luísa Augusto Barbosa, que por mais difíceis que fossem as circunstâncias, sempre esteve ao meu lado.

Agradeço ao grupo de professores de Engenharia de Software da Faculdade UnB Gama por dedicarem seu tempo e sua sabedoria para minha formação acadêmica. Em especial, sou grato a alguns professores que citarei devido a importantes contribuições pessoais e profissionais.

Agradeço ao Prof. Ricardo Matos Chaim pelas oportunidades e projetos desenvolvidos durante o curso.

Agradeço ao orientador Prof. Maurício Serrano pela orientação ao longo deste trabalho e por todas as experiências acadêmicas e profissionais compartilhadas; por todos os ensinamentos fornecidos que são fundamentais para minha formação profissional, seus ensinamentos abriram horizontes que eu não havia explorado ainda, estendendo ainda mais minha aprendizagem em diferentes aspectos da Engenharia de Software.

Agradeço à Profa. Milene Serrano, coorientadora deste trabalho, por suas orientações, atenção e generosidade em seus ensinamentos. Suas contribuições foram muito valorosas para o meu desenvolvimento como engenheiro de software.

Agradeço à Karolliny Braz de Araújo pelo apoio, amor e paciência ao longo da minha formação; por me ajudar a construir um grande futuro e por compartilhar meus sonhos.

Agradeço aos meus companheiros de graduação, em especial aos meus colegas Jeann Feitosa Figueiredo, Ewerson Jackson Oliveira Carvalho e Luan Guimarães Lacerda, pelo incentivo e apoio constantes, além dos conhecimentos compartilhados ao longo dessa formação. Vocês continuarão presentes em minha vida com certeza.

Agradeço à equipe da SEPLAN do Tribunal de Contas da União, em especial à Patrícia Jussara Mendes e à Renata Miranda Passos Camargo, pela oportunidade do

convívio que me proporcionou vários desafios e experiências fundamentais para meu amadurecimento e formação durante meu período de estágio.

Agradeço aos demais amigos e familiares pelo apoio, amizade e por participarem ativamente da minha vida pessoal.

Por fim, a todos que direta ou indiretamente fizeram parte da minha formação, o meu muito obrigado.

*“Que os vossos esforços desafiem as impossibilidades,  
lembrai-vos de que as grandes coisas do homem foram  
conquistadas do que parecia impossível.  
(Charles Chaplin, tradução nossa)*



# Resumo

Um projeto acústico deve ser cuidadosamente estudado, pois aspectos como dimensões e materiais aplicados influenciam diretamente no som do ambiente. O controle da acústica em um ambiente onde as dimensões já foram determinadas é realizado através da aplicação de materiais que influenciam no som. O emprego desses materiais deve ser cuidadosamente estudado a fim de aplicar materiais que contribuam com um bom projeto acústico e evitar o emprego de materiais supérfluos. O uso de simulações computacionais vem sendo cada vez mais frequente no dia a dia do projetista acústico, a fim de atingir projetos mais eficientes e que atendam às necessidades acústicas esperadas pelo cliente. Neste trabalho, será apresentado o projeto de um simulador que permite a simulação do comportamento do som dentro de um ambiente fechado. Esta simulação consiste em uma representação do som como uma partícula que percorre um determinado ambiente, identificando pontos de colisão e gerando as respectivas reflexões e absorções sonoras a fim de simular o evento de reverberação dentro do ambiente inserido. Este projeto visa a criação de uma máquina de raciocínio capaz de simular o comportamento do som dentro de um ambiente, permitindo então, o uso desta máquina na implementação de novos simuladores. Trata-se de uma ferramenta de software livre baseada no paradigma multiagentes, recorrendo a alguns recursos da orientação a objetos, onde seu principal foco é a comunidade de desenvolvimento de software livre.

**Palavras-chave:** simulador acústico. reflexão. absorção. reverberação. multiagentes.



# Abstract

An acoustic design should be carefully studied because aspects, such as dimensions and materials, applied have a direct influency at the sound of the environment. The control of the sound in an environment, where the dimensions have been determined, is accomplished through the application of materials which influence the sound. The use of these materials should be carefully studied in order to apply materials that contribute to a good acoustic design and avoiding the use of unnecessary materials. The use of computer simulations has been increasingly common in everyday acoustic designer in order to achieve more efficient designs that meet the acoustical requirements expected by the client. In this work, the design of a simulator that allows sound behavior simulation within a closed environment will be presented. The simulation consists of a sound representation as a particle that travels in a certain environment, identifying collision points and generating the respective sound absorption and reflections in order to simulate the reverberation event in the environment. This project aims to assist acoustic experts and / or designers, so they can monitor and evaluate if the parameters of the room are really suitable. It is a free software tool based on multi-agent paradigm, using some features of object orientation.

**Key-words:** acoustic simulator. reflection. absorption. reverberation. multi-agents.



# **Lista de ilustrações**

Figura 1 – Gráfico de frequência (SILVA, 1971) . . . . .	32
Figura 2 – Transmissão, absorção e reflexão do som (SILVA, 1971) . . . . .	34
Figura 3 – Reflexão. (SILVA, 1971) . . . . .	34
Figura 4 – Tela de simulação 3D do Odeon <sup>8</sup> . . . . .	42
Figura 5 – Telas de interface gráfica do software AcMus <sup>9</sup> . . . . .	43
Figura 6 – Processo TCC . . . . .	48
Figura 7 – Adaptação do Scrum para este TCC . . . . .	49
Figura 8 – Fonte Sonora . . . . .	61
Figura 9 – Ambiente bidimensional . . . . .	61
Figura 10 – JADE GUI . . . . .	62
Figura 11 – Arquitetura visão geral . . . . .	64
Figura 12 – Arquitetura da máquina de raciocínio do Simulador Acústico . . . . .	65
Figura 13 – Arquitetura JADE <sup>1</sup> . . . . .	66
Figura 14 – Ciclo de vida do agente JADE (SILVA, 2003) . . . . .	66
Figura 15 – Estrutura de pacotes do simulador acústico . . . . .	67
Figura 16 – Tela inicial do simulador acústico . . . . .	68
Figura 17 – Tela de configuração do ambiente . . . . .	68
Figura 18 – Tela de criação de obstáculos . . . . .	69
Figura 19 – Tabela de obstáculos . . . . .	69
Figura 20 – Tela de configuração da fonte sonora . . . . .	70
Figura 21 – Tabela de fontes sonoras . . . . .	71
Figura 22 – Tela de configurações da simulação . . . . .	71
Figura 23 – Barra de ferramentas . . . . .	71
Figura 24 – Visualização gráfica da simulação em andamento. . . . .	72
Figura 25 – Cobertura de código. . . . .	73
Figura 26 – Análise estática do código fonte. . . . .	74
Figura 27 – Tempo de reverberação dos simuladores por cenário. . . . .	76



# **Lista de tabelas**

Tabela 1 – Cronograma TCC 1 (2014) . . . . .	47
Tabela 2 – Cronograma TCC 2 (2015) . . . . .	47
Tabela 3 – Product Backlog Inicial . . . . .	50
Tabela 4 – Product Backlog Final . . . . .	51
Tabela 5 – Roadmap . . . . .	54
Tabela 6 – Percentual de conclusão das tarefas . . . . .	54
Tabela 7 – Cenários avaliados . . . . .	75
Tabela 8 – Resultados das simulações . . . . .	75



# **Lista de abreviaturas e siglas**

ACC	Acoplamento
ACCM	Complexidade Ciclomática
AMLOC	Média do tamanho dos métodos
AMS	Agent Management System
ANPM	Número de parâmetros por método
c.p.s	Ciclos por segundo
DF	Directory facilitator
DIT	Profundidade na árvore de herança
FGA	Faculdade do Gama
FIPA	Foundation for Intelligent Physical Agents
GPLv2	General Public License versão 2
GUI	Graphical User Interface
IDE	Integrated Development Environment
JADE	Java Agent Development Framework
JDK	Java Development Kit
LGPL	Lesser General Public License
LTS	Longo Tempo de Suporte
NOM	Número de métodos
NPA	Encapsulamento
RT60	Tempo de reverberação (Considerando o tempo em que o nível de intensidade sonora cai em 60dB).
SC	Coesão e acoplamento
US	User Story
TCC	Trabalho de Conclusão de Curso



# Listas de símbolos

$\gamma$	Letra grega Gama, símbolo da Faculdade do Gama (FGA).
Hz	Hertz
T	Período
f	Frequência
W	Energia sonora, medida em watt
S	Superfície, medida em $cm^2$
$\gamma$	Coeficiente de dissipação sonora.
$\varrho$	Coeficiente de reflexão.
$\tau$	Coeficiente de transmissão.
$\infty$	Coeficiente de absorção sonora.
$E_i$	Energia acústica incidente.
$E_d$	Energia acústica dissipada.
$E_r$	Energia acústica refletida.
$E_t$	Energia acústica transmitida.
dB	Decibél.
$\sigma$	Desvio padrão.
$\mu$	Média
LaTeX	Sistema de preparação de documentos utilizado na confecção deste trabalho.



# Sumário

<b>1</b>	<b>INTRODUÇÃO</b>	<b>27</b>
1.1	Contextualização	27
1.2	Questão de pesquisa	28
1.3	Justificativa	28
1.4	Objetivos	29
1.4.1	Objetivo geral	29
1.4.2	Objetivos específicos	29
1.5	Organização do documento	29
<b>2</b>	<b>REFERENCIAL TEÓRICO</b>	<b>31</b>
2.1	Acústica	31
2.1.1	Conceito do som	31
2.1.2	Frequência	31
2.1.3	Intensidade sonora	32
2.1.4	Reflexão	33
2.1.5	Absorção	35
2.2	Simulação	36
2.2.1	Simulação acústica	36
2.3	Sistemas Multiagentes	37
2.4	Considerações finais	38
<b>3</b>	<b>SUPORTE TECNOLÓGICO</b>	<b>39</b>
3.1	Ferramentas de desenvolvimento	39
3.1.1	JADE	39
3.1.2	Eclipse	39
3.1.3	JUnit	40
3.1.4	Ubuntu	40
3.1.5	Git	40
3.1.6	GitHub	41
3.2	Simuladores acústicos similares	41
3.2.1	Odeon	41
3.2.2	AcMus	42
3.3	Considerações finais	43
<b>4</b>	<b>METODOLOGIA</b>	<b>45</b>
4.1	Classificação da pesquisa	45

<b>4.2</b>	<b>Planejamento da pesquisa . . . . .</b>	<b>46</b>
<b>4.3</b>	<b>Cronograma da pesquisa . . . . .</b>	<b>46</b>
<b>4.4</b>	<b>Metodologia adotada no desenvolvimento do simulador . . . . .</b>	<b>47</b>
<b>4.5</b>	<b>Metodologia adotada na análise dos resultados . . . . .</b>	<b>56</b>
<b>4.6</b>	<b>Considerações finais . . . . .</b>	<b>56</b>
<b>5</b>	<b>RESULTADOS . . . . .</b>	<b>59</b>
<b>5.1</b>	<b>Prova de conceito . . . . .</b>	<b>59</b>
5.1.1	Especificações técnicas . . . . .	59
5.1.2	Descrição da prova de conceito . . . . .	59
5.1.3	Refinamento da proposta . . . . .	62
<b>5.2</b>	<b>O simulador acústico . . . . .</b>	<b>63</b>
5.2.1	Descrição do simulador . . . . .	63
5.2.2	Arquitetura . . . . .	63
5.2.3	Realizando uma simulação . . . . .	67
5.2.3.1	Tela inicial do simulador . . . . .	67
5.2.3.2	Adicionando obstáculos . . . . .	67
5.2.3.3	Adicionando fontes sonoras . . . . .	69
5.2.3.4	Configurando a velocidade da simulação . . . . .	70
5.2.3.5	Executando uma simulação . . . . .	71
5.2.4	Testes unitários e cobertura de código . . . . .	72
5.2.5	Métricas de qualidade de código fonte . . . . .	73
5.2.6	Comparação entre os cenários de uso avaliados . . . . .	74
<b>5.3</b>	<b>Considerações finais . . . . .</b>	<b>76</b>
<b>6</b>	<b>CONCLUSÃO . . . . .</b>	<b>79</b>
<b>6.1</b>	<b>Trabalhos futuros . . . . .</b>	<b>80</b>
	<b>Referências . . . . .</b>	<b>83</b>
	<b>APÊNDICES</b>	<b>87</b>
	<b>APÊNDICE A – CÓDIGO FONTE DA CLASSE AMBIENT . . . . .</b>	<b>89</b>
	<b>APÊNDICE B – CÓDIGO FONTE DA CLASSE SOUNDSOURCE . . . . .</b>	<b>93</b>
	<b>APÊNDICE C – CÓDIGO FONTE DA CLASSE SOUND . . . . .</b>	<b>97</b>
	<b>APÊNDICE D – CÓDIGO FONTE DA CLASSE OBSTACLE . . . . .</b>	<b>101</b>
	<b>APÊNDICE E – CÓDIGO FONTE DA CLASSE LINE . . . . .</b>	<b>103</b>

APÊNDICE F – CÓDIGO FONTE DA CLASSE NORMALLINE . .	107
APÊNDICE G – CÓDIGO FONTE DA CLASSE VERTICALLINE . .	109
APÊNDICE H – CÓDIGO FONTE DA CLASSE LOCATION . . . .	111
APÊNDICE I – CÓDIGO FONTE DA CLASSE SOUNDOBJECT .	113
APÊNDICE J – CÓDIGO FONTE DA CLASSE SOUNDSOURCE- OBJECT . . . . .	121
APÊNDICE K – CÓDIGO FONTE DA CLASSE AMBIENTOBJECT	127



# 1 Introdução

Segundo Silva (1971, pág. 17), uma boa acústica em um ambiente é consequência da aplicação, pelo arquiteto, dos princípios da Acústica Arquitetônica.

Um projeto acústico deve ser cuidadosamente estudado. O mesmo deve ser funcional, isto é, todos os detalhes deverão ter uma razão de ser, a fim de se evitar o emprego de materiais supérfluos. Nesse contexto, o uso de simulações computacionais torna-se cada vez mais frequente no dia a dia do projetista acústico, procurando garantir projetos mais eficientes e que atendam às necessidades acústicas esperadas pelo cliente. (SILVA, 1971).

## 1.1 Contextualização

Projetos de estúdios de gravação, salas de concerto, teatros, salas de aula ou locais que necessitem de qualidade acústica para realizar suas atividades requerem estudos sobre suas dimensões, volumetria, forma e composição de suas superfícies. É papel do projetista acústico definir esses parâmetros a fim de garantir os requisitos acústicos do projeto. (RAMIRES, 2011).

Em alguns casos, o volume e a forma desses ambientes já se encontram definidos, restando, portanto, ao projetista, definir quais materiais irão compor as superfícies desta sala, a fim de se obter um ambiente que cumpra com sua função acústica (RAMIRES, 2011).

Segundo Santos (2011, pág. 6), a adaptação do ambiente, de acordo com o uso em questão, é realizada a partir do direcionamento da energia sonora dentro do mesmo. Esta redistribuição pode ser tratada de diferentes maneiras, de acordo com o projeto ou com sua finalidade, seja ela música, fala ou mesmo palestras.

O estudo do comportamento acústico do campo sonoro em ambientes fechados foi desenvolvido na tentativa de explicar objetivamente como se dá o comportamento acústico de ambientes, bem como os vários parâmetros necessários para descrever o comportamento do som dentro de tais ambientes. (TORRES, 2008).

O estudo da acústica de salas data do início do século XX, porém, com a evolução da tecnologia, a simulação acústica de salas sofreu um grande avanço, sendo possível realizar simulações que requerem cada vez menos tempo e gastos. Devido a esse avanço, atualmente é possível prever, por exemplo, o comportamento sonoro de salas ainda não construídas, simular a acústica de construções antigas não mais existentes e ainda melhorar a acústica de salas já existentes (TORRES, 2008).

Segundo [Torres \(2008, pág. 1\)](#), as possibilidades trazidas pela simulação acústica são muitas. Com o uso dessa técnica, por exemplo, é possível projetar/simular uma sala construída de tijolo e concreto com detalhes e com uma rica qualidade sonora, o qual seria relativamente complexo sem o uso da simulação acústica.

Neste Trabalho de Conclusão de Curso, será tratado um sistema de simulação acústica, sendo esse uma ferramenta de software livre baseada em uma abordagem multiagentes. Até o momento, não foi possível encontrar na literatura investigada uma ferramenta que reúne simulação acústica e uma abordagem multiagentes dentro do contexto de software livre.

## 1.2 Questão de pesquisa

A questão motivadora desta pesquisa concentra-se em:

*É possível desenvolver um sistema que simule o comportamento do som dentro de um ambiente fechado utilizando uma abordagem multiagentes?*

## 1.3 Justificativa

A construção de salas, estúdios musicais, auditórios e ambientes, cuja propagação sonora é um fator importante, nem sempre é realizada de maneira adequada. A qualidade de som, a forma como este se propaga ao longo de uma sala e o isolamento acústico, são alguns dos fatores determinantes para um bom projeto. Adicionalmente, esses fatores podem influenciar diretamente na qualidade, nos custos e até mesmo no próprio comércio envolvido ([SANTOS, 2011](#)).

Em salas de aula, por exemplo, é importante saber se nas fileiras mais afastadas do professor é possível escutar com clareza as explicações. Além disso, é importante verificar se o projeto da sala foi eficiente o bastante para garantir que ruídos externos não interfiram na aula.

Já em estúdios musicais, é muito importante medir a qualidade do som, uma vez que esse fator está diretamente relacionado à sua atividade comercial. Além disso, verificar se o projeto atende ao requisito de isolamento sonoro do estúdio também requer especial atenção ([RAMIRES, 2011](#)).

Tendo em vista as dimensões do auditório, é importante verificar qual o melhor posicionamento dos alto-falantes, para que todos dentro do auditório possam ouvir o som com a mesma qualidade.

Atualmente, uma das dificuldades dos desenvolvedores de projetos de construção civil é predizer se o som se comporta adequadamente dentro das especificações e necessida-

des deste projeto bem como verificar possíveis melhorias de projeto, incluindo dimensões, disposições, materiais, os quais atendam aos requisitos de qualidade do projeto ([SANTOS, 2011](#)). Neste sentido, é proposto neste trabalho um software que apóie à visualização do comportamento dos elementos sonoros envolvidos em projetos de construção civil, auxiliando aqueles interessados na otimização acústica dentro de determinados ambientes.

## 1.4 Objetivos

### 1.4.1 Objetivo geral

Desenvolver um sistema que seja capaz de simular o comportamento do som dentro de um ambiente fechado, utilizando uma abordagem multiagentes, para que possa ser incorporado na implementação de novos simuladores acústicos, afim de potencializar o auxílio aos projetistas e/ou especialistas em acústica no que tange o acompanhamento e a avaliação dos parâmetros de seus projetos.

### 1.4.2 Objetivos específicos

1. Estudar o comportamento do som dentro de ambientes acústicos, identificando variáveis acústicas presentes dentro desses ambientes.
2. Identificar índices de absorção referentes aos materiais presentes nos ambientes em análise.
3. Propor um suporte tecnológico baseado em uma abordagem multiagentes, o qual será utilizado para a implementação da solução.
4. Explorar técnicas de programação, padrões de projeto e demais boas práticas da Engenharia de Software visando o desenvolvimento de um simulador manutenível e extensível.
5. Definir métricas de qualidade visando realizar a análise estática e a cobertura do código do simulador proposto, com base em uma abordagem de teste apropriada para o contexto, focada, principalmente, em testes de unidade.

## 1.5 Organização do documento

Os próximos capítulos estão organizados da seguinte forma:

- Referencial teórico - apresenta uma visão breve sobre os principais conceitos abordados neste trabalho, baseado na literatura disponível;

- Suporte tecnológico - apresenta as principais tecnologias utilizadas durante a condução deste trabalho.
- Metodologia - trata da descrição detalhada de todo o desenvolvimento do trabalho, tal como a explicação do tipo de pesquisa e a descrição da prova de conceito.
- Resultados - apresenta os produtos de trabalho desenvolvidos, assim como os resultados dos testes realizados na validação dos mesmos.
- Conclusão - conclui o trabalho, apresentando as principais contribuições do mesmo bem como acorda possíveis trabalhos relacionados.

## 2 Referencial Teórico

Este capítulo apresenta o referencial teórico estudado para que fosse possível realizar o desenvolvimento do simulador acústico proposto neste trabalho. São abordados neste capítulo, conceitos referentes a acústica, simulação, simulação acústica e sistemas multiagentes.

### 2.1 Acústica

Nesta seção serão apresentados alguns conceitos importantes sobre acústica que serão abordados neste projeto.

#### 2.1.1 Conceito do som

Ao falarmos sobre som, existem dois conceitos importantes a serem tratados, o som vibração, ou perturbação física, que percorre um meio qualquer de propagação e o som, sensação sonora, psico-fisiológico, que é captado pelo nosso ouvido. O conceito que mais interessa aos profissionais da Acústica Arquitetônica é o último, isto é, o do som, sensação sonora, captada pelo nosso aparelho auditivo. ([SILVA, 1971](#)).

Segundo [Bistafa \(2006, pág. 6\)](#), o som pode ser definido como uma variação da pressão ambiente detectável pelo sistema auditivo.

Para que haja a propagação do som, é necessário que ele esteja em um meio de propagação, provido de inércia e de elasticidade. O meio mais comum desta propagação, para chegar até ao nosso ouvido, é o ar que nos envolve. Se não houver gás preenchendo o espaço que nos circunda, os sons deixarão de ser ouvidos. ([SILVA, 1971](#))

#### 2.1.2 Frequência

Segundo [Silva \(1971, pág. 34\)](#), chama-se de frequência o número de oscilações completas por segundo, ou seja, o número de idas e voltas completas da partícula vibrante. A Frequência é medida em ciclos por segundo (c.p.s) ou em Hertz (Hz).

Se representarmos o diagrama das pressões para um ponto qualquer de um recinto, onde se propague o som, em função do tempo, encontraremos o gráfico da figura 1, onde as pressões foram registradas em ordenadas e o tempo em abcissas. As suas unidades são, respectivamente,  $dina/cm^2$  e segundo. ([SILVA, 1971](#))

O período T é o tempo necessário para se realizar uma oscilação completa. Portanto, sendo a frequência o número de períodos por segundo, a fórmula que exprime seu

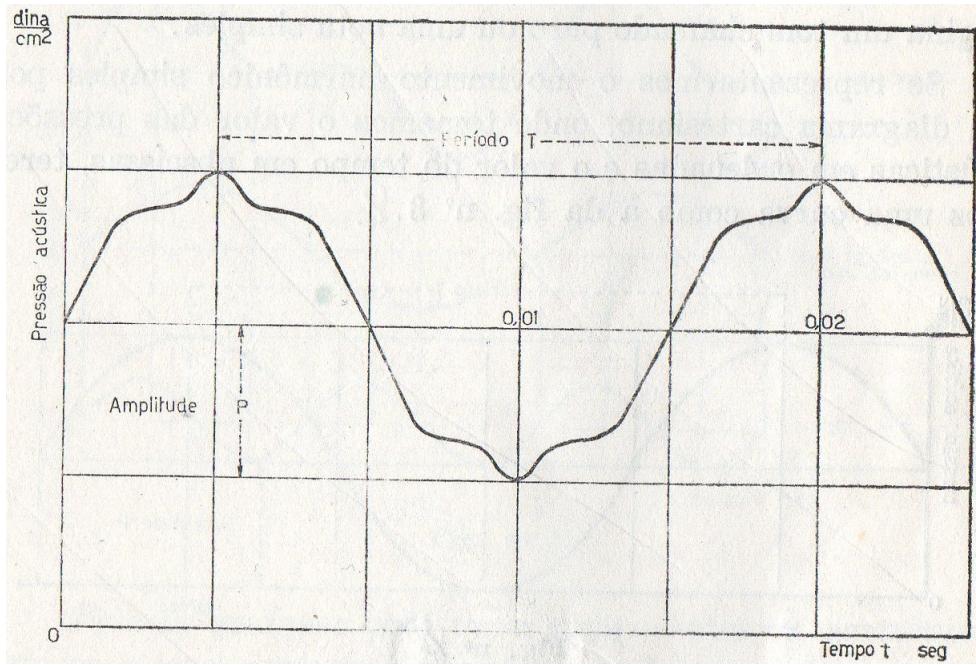


Figura 1 – Gráfico de frequência (SILVA, 1971)

valor será (BISTAFA, 2006, pág. 7):

$$f = \frac{1}{T}$$

"Os sons audíveis encontram-se no intervalo de  $\pm 16$  a  $\pm 20.00$  c.p.s. Aqueles, abaixo de 16 c.p.s, são chamados de infra-sons e aqueles, acima de 20.000 c.p.s, são designados como ultra-sons. Tanto uns como os outros não são captados pelo ouvido humano. (SILVA, 1971, pág. 35)."

Em música, o termo utilizado para a percepção da frequência sonora pelo ouvido humano é a altura. Quanto maior o número de oscilações por segundo, maior é a sua altura, portanto, as frequências baixas são percebidas como sons graves e as frequências mais altas como sons agudos.

### 2.1.3 Intensidade sonora

Para que possamos ter a sensação de audição, além da necessidade de o som estar entre o intervalo de 16 a 20.000 oscilações por segundo, isto é, ter determinada altura, é necessário que ele tenha uma certa intensidade sonora. (SILVA, 1971, pág 42)

Segundo Silva (1971, pág 42), a intensidade sonora  $I$ , medida em  $watt/cm^2$ , é a quantidade de energia sonora  $W$ , medida em  $watt$ , que atravessa um centímetro quadrado de área, perpendicular à direção em que o som se propaga. É calculada através da fórmula:

$$I = \frac{W}{S}$$

A menor variação de pressão ambiente detectável pelo sistema auditivo é da ordem de  $10^{-12} W/m^2$ , também conhecido como limiar de audibilidade. Já a variação de pressão ambiente capaz de provocar dor é o limiar de dor, que é equivalente à  $1 W/m^2$ . (BISTAFA, 2006, pág 6)

$$I_0 = 10^{-12} \text{ } W/m^2$$

$$I_{máx} = 1 \text{ } W/m^2$$

Relacionado a intensidade sonora, existe também o conceito de nível de intensidade sonora, o qual é representado pelo decibél (dB) e é equivalente a uma relação percentual. O nível de intensidade sonora precisa de um valor de referência para ser expresso. Em acústica, o 0 dB é equivalente ao limiar de audibilidade e, a cada 10 dB, a intensidade sonora aumenta em dez vezes e dobra a cada aproximadamente 3 db. A expressão utilizada para calcular o nível de intensidade sonora com base na intensidade sonora é:

$$\beta = 10 \log \frac{I}{I_0}$$

#### 2.1.4 Reflexão

Quando uma onda sonora atinge uma parede ou um obstáculo qualquer, parte da energia incidente é refletida, parte é dissipada pelo obstáculo, transformando-se em energia calorífica ou mecânica. O restante atravessa o referido obstáculo, passando para o outro lado, transmitindo-se através do meio adjacente. (SILVA, 1971).

As paredes e divisões, geralmente rígidas, vibram no todo ou em parte, devido à energia das ondas sonoras. Essas paredes e divisões vibram como se fossem diafragmas e rerradiam a energia, que nelas incida. Sendo assim, as paredes ou divisões mais rígidas e mais pesadas são melhores isolantes sonoros, do que aquelas construídas de material leve e flexível. (SILVA, 1971, pág. 83).

Segundo Silva (1971, pág. 84), quando uma onda sonora pura ou livre, isto é, isenta de reflexões secundárias, atinge uma superfície uniforme e relativamente grande, em relação ao comprimento dessa onda, a reflexão do som assemelha-se muito com à da luz.

Se representarmos as ondas sonoras pelos seus raios sonoros, elas serão retas dirigidas segundo a direção em que as mesmas caminham. (SILVA, 1971, pág. 84)

O ângulo do raio incidente, com a normal à superfície refletora, é igual ao ângulo formado pelo raio refletido com aquela mesma linha e estão no mesmo plano, conforme representado na figura 3.

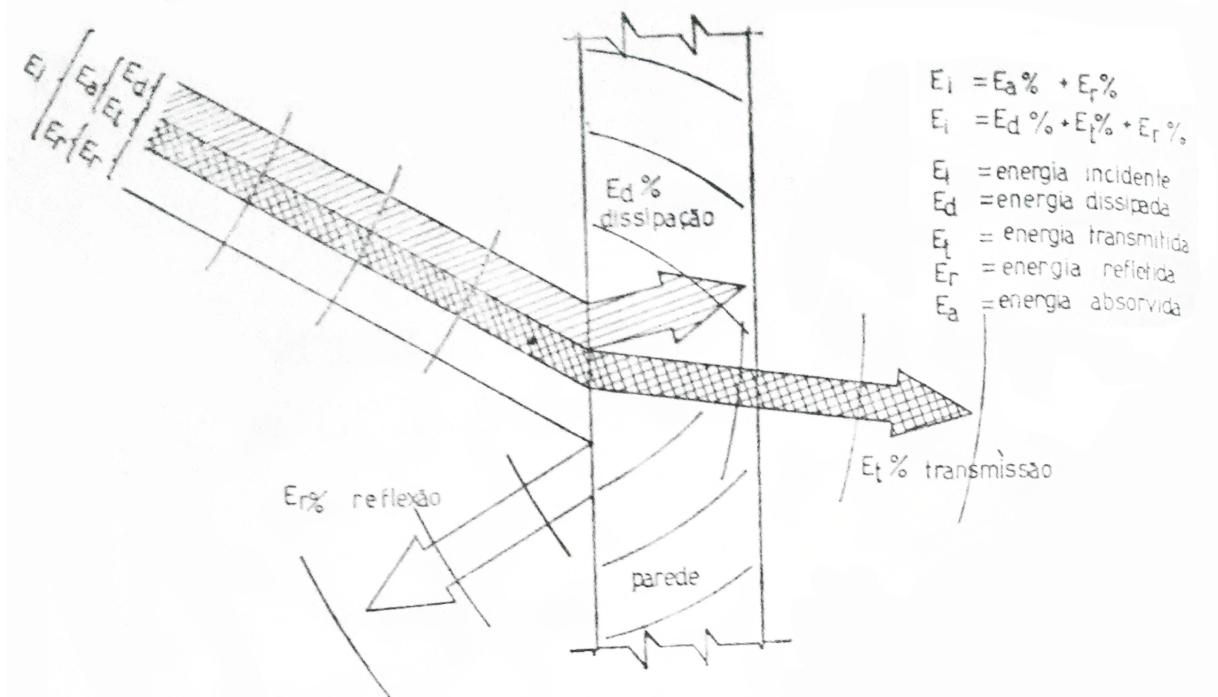


Figura 2 – Transmissão, absorção e reflexão do som (SILVA, 1971)

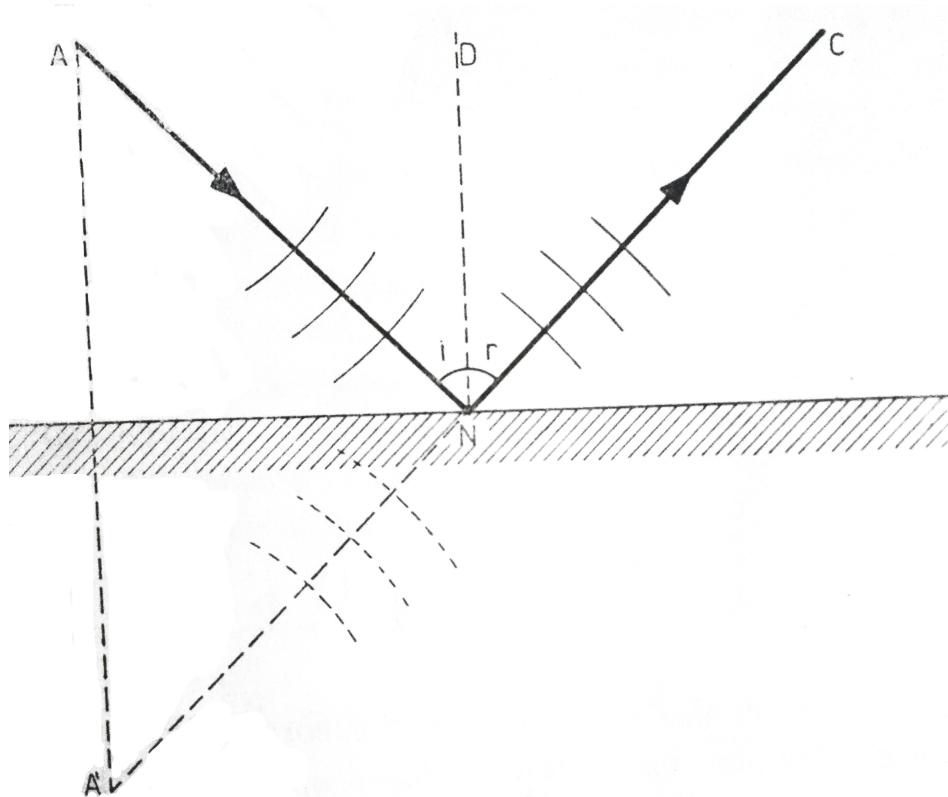


Figura 3 – Reflexão. (SILVA, 1971)

### 2.1.5 Absorção

Em ambientes fechados, a propagação da onda sonora, a partir da fonte, sofre interferência das ondas que são refletidas nas superfícies que delimitam o recinto como paredes, teto e piso. (BISTAFA, 2006, pág 231)

Conforme foi ilustrado na figura 2, quando o som incide sobre uma superfície, uma parte da energia sonora é refletida, enquanto o restante da energia incidente se compõe de duas outras parcelas: a energia sonora absorvida e a energia sonora transmitida para o outro lado da superfície. (BISTAFA, 2006, pág. 231)

Segundo (SILVA, 1971, pág. 126), um material é dito acústico absorvente quando uma grande percentagem da energia sonora que nele incide é retida, degradando-se em energia mecânica ou calorífica, ou ainda transmitindo-se para o outro lado, sendo dele refletida apenas uma pequena parcela.

Chama-se de coeficiente de dissipação sonora,  $\gamma$ , à porcentagem de energia sonora absorvida ou dissipada  $E_d$  em um material qualquer, em relação à quantidade de energia acústica incidente  $E_i$  na unidade de área. (SILVA, 1971, pág. 126).

$$\gamma = \frac{E_d}{E_i}$$

Coeficiente de reflexão,  $\varrho$ , é a relação entre o fluxo de energia acústica refletida,  $E_r$ , e o fluxo de energia acústica incidente,  $E_i$ , na unidade de área. (BISTAFA, 2006, pág. 232)

$$\varrho = \frac{E_r}{E_i}$$

Já o coeficiente de transmissão,  $\tau$ , é a relação entre o fluxo de energia acústica transmitida,  $E_t$ , e o fluxo de energia acústica incidente,  $E_i$ , também, na unidade de área. (SILVA, 1971, pág. 127).

$$\tau = \frac{E_t}{E_i}$$

O coeficiente de absorção acústica,  $\infty$ , é a soma dos coeficientes de dissipação,  $E_d$ , e de transmissão,  $E_t$ , isto é, a diferença entre a energia acústica incidente e a refletida. (BISTAFA, 2006, pág. 231).

$$\infty = \gamma + \tau = \frac{E_i - E_r}{E_i} = 1 - \frac{E_r}{E_i} \text{ ou } 1 - \varrho$$

A soma dos 3 coeficientes descritos acima é:

$$\gamma + \varrho + \tau = \frac{E_d}{E_i} + \frac{E_r}{E_i} + \frac{E_t}{E_i} = 1$$

Cada material tem um coeficiente de absorção e o seu valor não é constante, pois varia com a frequência do som incidente. Uma superfície, teórica, infinitamente rígida e polida, seria totalmente refletora e seu coeficiente de absorção seria nulo. Já uma janela aberta de um recinto qualquer, teria esse coeficiente igual a 1, o que significa que 100% da energia incidente passa para o outro lado, isto é, para o lado de fora. ([SILVA, 1971](#)).

## 2.2 Simulação

A simulação computacional possui diversos conceitos por diferentes autores, porém, em sua grande maioria, acabam por convergir para o fato de que a simulação contribui para a resolução de problemas complexos ([CARVALHO, 2003](#)). A seguir serão apresentadas as visões de alguns autores a respeito do conceito de simulação.

De acordo com ([SCHRIBER, 1974](#)), simulação pode ser definida da seguinte maneira: "simulação implica na modelagem de um processo ou sistema, de tal forma que o modelo imite as respostas do sistema real numa sucessão de eventos que ocorrem ao longo do tempo".

Segundo [Law e Kelton \(1991\)](#), simulação é a imitação de um sistema real, modelado em computador, para avaliação e melhoria de seu desempenho. Ou seja, simulação é a representação de um processo do mundo real para um ambiente controlado onde se pode estudar o comportamento do mesmo, sob diversas condições, sem riscos físicos e/ou grandes custos envolvidos. ([TORGA, 2007](#)).

Segundo [Torres \(2008\)](#), computadores são ótimas ferramentas para simular processos do mundo em que vivemos. De acordo com [Carvalho \(2003\)](#), atualmente, simulação é praticamente sinônimo de simulação computacional digital.

### 2.2.1 Simulação acústica

A acústica geométrica de salas é um modelo de representação acústica de ambientes fechados onde o conceito de onda sonora é substituído pelo conceito de raio sonoro. ([TORRES, 2008](#))

Dependendo da simplicidade do modelo, a utilização de ferramentas matemáticas como cálculo, álgebra ou probabilidade podem ser suficientes para se obter uma resposta exata a respeito de parâmetros como o tempo de reverberação do som em um ambiente. Essa resposta exata é conhecida como solução analítica do modelo. ([TORRES, 2008](#))

Apesar de ser possível obter resultados utilizando ferramentas matemáticas, é muito raro encontrar sistemas no mundo real que possam ser representados por mode-

los simples o suficiente para serem resolvidos de forma analítica, pois algumas equações clássicas da acústica de salas não são suficientes para oferecer bons resultados quando as salas assumem geometria complexa. (BASTOS; CARDOSO; MELO, 2010)

Segundo Torres (2008, pág. 5), para modelos mais complexos, a simulação é uma ferramenta poderosa que pode ajudar a obter mais informações a respeito de tais processos, como ocorre em simulação acústica, tornando possível prever a acústica de salas complexas.

"Para que os parâmetros acústicos de um ambiente possam estar de acordo com os padrões previstos para sua finalidade, é recomendável que se faça uma predição do seu comportamento acústico, através de modelos de simulação, para que sejam obtidos resultados com boa precisão em um tempo de processamento relativamente curto, reduzindo custos e aumentando a eficiência dos projetos, quando da seleção correta e quantidade certa de material a ser utilizada para se efetuar tratamentos acústicos nesse ambiente."(BASTOS; CARDOSO; MELO, 2010).

O avanço na capacidade de processamento dos computadores atuais os tornou capazes de calcular os mais diversos e complexos efeitos da propagação de ondas. Métodos numéricos são capazes de transportar a realidade física para a linguagem computacional. (CAMILO; TENENBAUM; COELHO, 2002).

## 2.3 Sistemas Multiagentes

Sistemas multiagentes são sistemas compostos de vários elementos de computação interagindo entre si, conhecidos como agentes. Agentes são sistemas computacionais com dois recursos importantes. Primeiramente, eles são pelo menos a certo ponto, capazes de realizar ações autônomas, ou seja, capazes de decidir por si mesmos o que eles precisam fazer a fim de satisfazer seus objetivos de projeto. Em segundo lugar, eles são capazes de interagir com outros agentes, não simplesmente pela troca de dados, mas por se envolver em análogos do tipo de atividade social que todos nós colocamos em nosso dia a dia: a cooperação, coordenação, negociação, dentre outros. (WOOLDRIDGE, 2009).

Segundo Weiss (1999), "Agentes" são entidades autônomas, computacionais. Dizer que os agentes são entidades computacionais simplesmente significa que existem na forma de programas que são executados em dispositivos de computação. Dizer que eles são autônomos significa que até certo ponto eles têm controle sobre seu comportamento e podem agir sem a intervenção de seres humanos ou outros sistemas. Agentes buscam atingir objetivos ou realizar tarefas, a fim de encontrar seus objetivos de projeto, e em geral, essas metas e tarefas podem ser tanto complementares, como conflitantes.

Dizer que os agentes são inteligentes não significa que eles são omnisciente ou onipotente, nem significa que eles nunca falham. Pelo contrário, significa que eles operam de forma flexível e racional em uma variedade de circunstâncias ambientais, dada a informação que eles têm e as suas capacidades perceptivas e eficazes. O foco principal está em processos como a resolução de problemas, planejamento, pesquisa, tomada de decisões e de aprendizagem que tornam possível para os agentes, mostrar flexibilidade e racionalidade em seu comportamento, e sobre a realização de tal processo em cenários multiagentes. ([WEISS, 1999](#)).

Agentes também interagem entre si, isto é, os agentes podem ser afetados por outros agentes em busca de atingir seus objetivos e executar suas tarefas. A interação pode ocorrer indiretamente, através do meio ambiente em que estão embutidas, ou diretamente, através de uma linguagem comum. ([WEISS, 1999](#)).

Um sistema multiagentes pode ser descrito como uma comunidade de agentes autônomos, onde estes interagem, se organizam e se comunicam, competindo ou cooperando uns com os outros, a fim de alcançar o objetivo proposto para esse sistema. ([THOMAZ, 2011](#)).

A área de sistemas multiagentes é um promissor domínio tecnológico para uso em performances musicais interativas. Essa tecnologia vem sendo utilizada para resolver problemas musicais de escopo específico e alcance limitado, como por exemplo, a simulação de instrumentos e o acompanhamento musical automático. ([THOMAZ, 2011](#)).

## 2.4 Considerações finais

Prever o comportamento do som dentro de um ambiente fechado vem se tornando uma necessidade cada vez mais notável, porém, não é trivial. Para isto, está sendo cada vez mais comum o uso de simuladores que auxiliem os profissionais da área da acústica arquitetônica na análise do comportamento sonoro em seus projetos.

O paradigma de programação multiagentes apresenta um grande potencial para tal finalidade, apesar de ter sido pouco explorado na área de simulação de ambientes acústicos. As capacidades de autonomia, paralelismo e cooperação dos agentes, também podem ser percebidas no comportamento sonoro em um ambiente, o que demonstra uma promissora capacidade de representação do som dentro de ambientes fechados..

# 3 Suporte Tecnológico

Este capítulo apresenta o suporte tecnológico utilizado para o desenvolvimento do simulador acústico. Durante o capítulo, serão abordadas ferramentas de desenvolvimento, o qual foram utilizados para a construção da ferramenta, e ferramentas de simulação acústica similares ao simulador proposto para este trabalho de conclusão de curso.

## 3.1 Ferramentas de desenvolvimento

### 3.1.1 JADE

O JADE (*Java Agent Development Framework*)<sup>1</sup> é um framework de software totalmente implementado na linguagem de programação Java. Ele simplifica a implementação de sistemas multiagentes através de um middleware que está em conformidade com as especificações FIPA (*Foundation for Intelligent Physical Agents*) e através de um conjunto de ferramentas gráficas que suportam as fases de depuração e implantação.

A especificação mínima do sistema para executar o JADE é a versão 5 do Java (o *run time environment* ou o JDK).

O framework JADE é um software livre e é distribuído pela Telecom Italia, titular dos direitos de autor, em código aberto sob os termos e condições da licença LGPL (*Lesser General Public License* versão 2).

Além da equipe do JADE, existe também, uma grande comunidade de desenvolvedores que começaram a participar no desenvolvimento do JADE nestes anos. Qualquer pessoa que está disposta a contribuir com esta Comunidade, relatando erros, fornecendo correções e contribuições ou simplesmente comentários e sugestões, pode participar do projeto.

### 3.1.2 Eclipse

O Eclipse<sup>2</sup> é uma IDE (*Integrated Development Environment*) *open source* para desenvolvimento Java, porém pode suportar outras linguagens a partir de plug-ins.

O Projeto Eclipse foi originalmente criado pela IBM em novembro de 2001 e apoiado por um consórcio de fornecedores de software. A Eclipse Foundation foi criada em janeiro de 2004 como uma corporação sem fins lucrativos para permitir que uma comunidade de fornecedores neutros, abertos e transparentes pudesse ser estabelecida em torno

---

<sup>1</sup> <<http://jade.tilab.com/>>

<sup>2</sup> <<https://www.eclipse.org/org/>>

do Eclipse. Hoje, a comunidade Eclipse é composta por indivíduos e organizações de uma seção transversal da indústria de software.

### 3.1.3 JUnit

JUnit<sup>3</sup> é um framework open-source simples para escrever testes repetíveis na linguagem de programação Java. É um exemplo da arquitetura xUnit para frameworks de testes de unidade.

### 3.1.4 Ubuntu

O Ubuntu<sup>4</sup> é um sistema operacional *open source*, construído a partir do núcleo Linux, baseado no Debian e patrocinado pela Canonical Ltd. Seu desenvolvimento visa segurança, sendo disponibilizadas atualizações relacionadas à segurança por pelo menos 18 meses para desktops e servidores e chegando à três anos na versão de Longo Tempo de Suporte (LTS).

O Linux já foi estabelecido como uma plataforma de servidor de empresa em 2004, mas o software livre não era uma parte da vida cotidiana para a maioria dos usuários de computador. É por isso que Mark Shuttleworth reuniu uma pequena equipe de desenvolvedores de um dos projetos Linux mais estabelecidos - Debian - e se propôs a criar um desktop Linux fácil de usar, o Ubuntu.

A visão para o Ubuntu é parte social e parte econômica: software livre, disponível a todos nas mesmas condições, e financiado por meio de um portfólio de serviços prestados pela Canonical.

### 3.1.5 Git

Git<sup>5</sup> é uma sistema de controle de versão distribuída de forma livre e de código aberto projetada para lidar com variados tipos de projeto, com ênfase na velocidade e na eficiência.

O Git é distribuído sob a licença GNU GPLv2 foi desenvolvido inicialmente para ser utilizado no desenvolvimento do Kernel Linux, porém, já foi adotado por muitos outros projetos.

---

<sup>3</sup> <<http://junit.org/index.html>>

<sup>4</sup> <<http://ubuntu-br.org/ubuntu>>

<sup>5</sup> <<http://git-scm.com/>>

### 3.1.6 GitHub

O GitHub<sup>6</sup> é um repositório web projetado para auxiliar projetos que utilizam o sistema de controle de versão Git. Essa ferramenta permite que um grupo de colaboradores ou até mesmo que equipes de pessoas em um projeto ou organização possam desenvolver, se comunicar e acompanhar o status do projeto com facilidade. Também é possível revisar mudanças, comentar linhas de código, reportar *issues* e planejar o futuro do projeto com ferramentas de discussão.

O GitHub é escrito em Ruby on Rails e possui planos comerciais e gratuitos no caso de projetos de código aberto.

## 3.2 Simuladores acústicos similares

### 3.2.1 Odeon

O Software ODEON<sup>7</sup> é desenvolvido para simular a acústica interior de construções. Dada a geometria e as propriedades das superfícies presentes no ambiente, a acústica pode ser prevista, ilustrada e ouvida. O ODEON utiliza o método de fontes virtuais combinado com o método de traçado de raios.

O projeto ODEON foi iniciado na cooperação entre a Universidade Técnica da Dinamarca (Departamento de Tecnologia Acústica) e um grupo de empresas de consultoria em 1984, com a finalidade de fornecer software de previsão confiável para acústica da sala. Os anos investidos no desenvolvimento do ODEON resultaram em um software de medição e previsão acústica de salas confiável e fácil de usar.

"O método do Traçado de Raios assume que o som é irradiado em forma de partículas na velocidade do som. As partículas viajam pelo ar e são refletidas quando encontram um obstáculo, i.e., as paredes da sala. Em cada reflexão são considerados os coeficientes de absorção e de espalhamento da parede. Quando a partícula atinge o receptor, sua energia e seu tempo são registrados, para que seja construída a Resposta Impulsiva do receptor. Este procedimento é repetido até que algum critério de parada seja alcançado, como por exemplo, quando a energia do raio for suficientemente pequena se comparada à energia inicial."(OFUGI; A; A, 2013).

"O Método das Fontes Virtuais pode ser comparado a uma questão óptica que envolve espelhos e suas imagens. É análogo a uma sala de espelhos onde os objetos são refletidos, e suas imagens também podem ser refletidas, e ambos podem ser observados. As imagens que são refletidas diretamente do objeto são chamadas imagens de primeira ordem, as imagens das

---

<sup>6</sup> <<https://github.com/>>

<sup>7</sup> <<http://www.odeon.dk/content/acoustics-simulation-software>>

imagens são chamadas de imagens de segunda ordem, e assim por diante. No caso da acústica, os objetos refletidos são as fontes sonoras e os observadores são os receptores."(OFUGI; A; A, 2013).

A figura 4 ilustra o módulo de simulação 3D da reflexão sonora dentro de um ambiente.



Figura 4 – Tela de simulação 3D do Odeon<sup>8</sup>

Este simulador foi utilizado como base e inspiração para o projeto proposto neste trabalho.

### 3.2.2 AcMus

O AcMus<sup>8</sup> é um projeto voltado para a pesquisa sobre acústica musical que visa desenvolver modelos e ferramentas computacionais para o estudo de ambientes destinados à escuta musical. Este pretende ser o primeiro projeto de um núcleo de pesquisa interdisciplinar voltado para o assunto da acústica musical, nascido de um trabalho conjunto entre os departamentos de Música e Ciência da Computação da Universidade de São Paulo.

O AcMus é uma ferramenta de software livre implementado como um arcabouço orientado a objetos utilizando a linguagem de programação Java. A ferramenta reúne, em um único ambiente integrado, medição, análise e simulação acústica de salas.

A interface gráfica da ferramenta é baseada no template da IDE Eclipse, descrita na seção 3.1.2, conforme pode ser visto na figura 5.

<sup>8</sup> <<http://gsd.ime.usp.br/acmus/menu.html>>

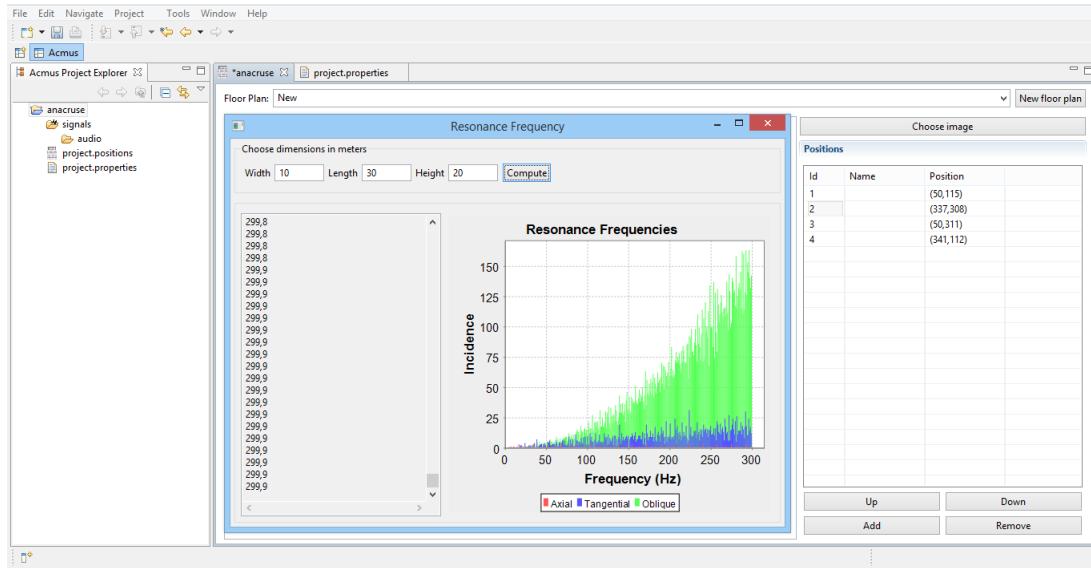


Figura 5 – Telas de interface gráfica do software AcMus<sup>9</sup>

### 3.3 Considerações finais

As ferramentas de desenvolvimento citadas neste capítulo são importantes tanto para o desenvolvimento da prova de conceito quanto do simulador em si. Em conjunto, essas ferramentas auxiliam o desenvolvimento de um sistema extensível e manutenível, visando qualidade e boas práticas de programação.

As ferramentas de simulação acústica descritas auxiliaram na compreensão do domínio do problema. Essas ferramentas foram investigadas com intuito de verificar quais são os sistemas existentes no mercado e como a simulação acústica é tratada nessas ferramentas. O estudo desses simuladores serviu de insumo para a elaboração da proposta do simulador acústico, produto de trabalho deste TCC.



## 4 Metodologia

Neste capítulo, será especificada qual a metodologia de pesquisa científica, dentre as pesquisadas, que mais se adequou ao tema coberto nesse Trabalho de Conclusão de Curso. Posteriormente, são descritos: (i) o planejamento da pesquisa seguido do cronograma que aborda suas atividades, (ii) a metodologia que conduziu o desenvolvimento do simulador em si, e (iii) a metodologia que orientou a análise dos resultados obtidos.

### 4.1 Classificação da pesquisa

Pesquisas são classificadas de acordo com seus objetivos gerais. O objetivo de uma pesquisa exploratória é a familiarização com o assunto que ainda não foi bem explorado e existem poucas informações acerca do mesmo. Em sua maioria, a pesquisa assume forma de planejamento bibliográfico ou estudo de caso. Essas pesquisas podem envolver levantamento bibliográfico, entrevistas com pessoas que tiveram experiências práticas com o problema pesquisado e também análise de exemplos que estimulem a compreensão (GIL, 2002).

Outro tipo de classificação é formulada com base nos procedimentos técnicos utilizados para seu desenvolvimento. A exemplo disto, existe a pesquisa experimental, que se trata de uma pesquisa envolvendo algum tipo de experimento. Consiste ainda em determinar um objeto de estudo, selecionar as variáveis capazes de influenciá-lo e definir as formas de controle e observação dos efeitos que a variável produz no objeto. Sendo assim, as pesquisas experimentais são um procedimento importante para os cientistas testarem hipóteses que estabelecem relações de causa e efeito entre as variáveis. Portanto, essa técnica garante ao pesquisador um caráter de agente ativo e não apenas de mero observador passivo (SILVA; TAFNER, 2006).

A pesquisa-ação é um tipo de pesquisa que em oposição ao modelo tradicional pode ser considerada como “independente” e “objetiva” (ROCHA, 2012). Segundo Thiollent (1986 apud ROCHA, 2012), caracteriza-se com pesquisa-ação:

"Um tipo de pesquisa social com base empírica, concebida e realizada em estreita associação com uma ação ou com a resolução de um problema coletivo no qual os pesquisadores e os participantes, representativos da situação e/ou do problema, estão envolvidos de forma cooperativa e participativa"(THIOLLENT, 1986 apud ROCHA, 2012).

Considerando os objetivos de estudo e o referencial bibliográfico levantado até o momento, este projeto se enquadra no modelo de pesquisa exploratório, com análise de

resultados baseada em pesquisa-ação.

## 4.2 Planejamento da pesquisa

O processo de pesquisa foi conduzido inicialmente, por um levantamento bibliográfico, porém, o mesmo foi revisado durante todo o projeto. Durante a fase inicial da pesquisa, algumas atividades tiveram um foco maior, como a definição do escopo, o estudo do domínio e o planejamento da abordagem. Esta última foi executada até o final do desenvolvimento da pesquisa.

Após a fase inicial da pesquisa, algumas outras atividades foram priorizadas, tais como a implementação da prova de conceito, o refinamento da proposta e a elaboração da parte escrita.

As considerações feitas pela banca de TCC 01 foram coletadas e registradas como sugestões de melhoria. A partir dessas sugestões, foram realizadas as devidas alterações no projeto. Com as melhorias implementadas, o desenvolvimento do Simulador Acústico se iniciou e foi conduzido utilizando conceitos de qualidade de código e técnicas de programação estudados durante o curso de graduação em Engenharia de Software.

Ao finalizar o desenvolvimento do Simulador Acústico, os resultados foram coletados e registrados, sendo então incorporados à parte escrita do projeto, que, por sua parte, teve um foco maior na fase final da pesquisa.

Durante esse processo, algumas metodologias de gerência e de desenvolvimento assim como ferramentas e tecnologias que pudessem proporcionar um melhor acompanhamento e suporte ao projeto foram estudadas.

Com o intuito de ilustrar como se deu a realização e condução deste projeto, segue a figura 6 que apresenta a modelagem do processo metodológico com base na metodologia descrita nesta seção.

## 4.3 Cronograma da pesquisa

Segue o cronograma nas tabelas 1 e 2 visando demonstrar como as principais atividades foram conduzidas ao longo deste trabalho.

	Ago	Set	Out	Nov	Dez
Realizar levantamento bibliográfico	X	X	X		
Estudar domínio do projeto	X				
Definir escopo do projeto	X	X			

Continua na página seguinte

**Tabela 1 – Continuação da página anterior**

	Ago	Set	Out	Nov	Dez
Realizar planejamento da abordagem	X	X			
Levantar suporte tecnológico	X	X			
Implementar da prova de conceito		X			
Refinar proposta			X		
Elaborar parte escrita do projeto		X	X		
Apresentação para a Banca de TCC 1				X	
Coletar sugestões de melhoria da banca				X	
Realizar mudanças					X

Tabela 1 – Cronograma TCC 1 (2014)

	Jun	Jul	Ago	Set	Out	Nov	Dez
Desenvolver simulador acústico	X	X	X				
Coletar primeiras impressões do simulador				X			
Elaborar parte escrita do projeto					X		
Apresentação para a Banca de TCC 2						X	
Refinar a escrita com base nas considerações da Banca						X	
Escrever Artigo							X

Tabela 2 – Cronograma TCC 2 (2015)

## 4.4 Metodologia adotada no desenvolvimento do simulador

Scrum<sup>1</sup> é uma estrutura de processo que tem sido usado para gerenciar o desenvolvimento de produtos complexos desde o início da década de 1990. Scrum não é um processo ou uma técnica para a construção de produtos, o Scrum é um quadro no qual você pode empregar diversos processos e técnicas. Scrum deixa clara a eficácia relativa das suas práticas de gestão de desenvolvimento de produto e de modo que você pode melhorar (SCHWABER; BEEDLE, 2002).

<sup>1</sup> <<https://www.scrum.org/>>

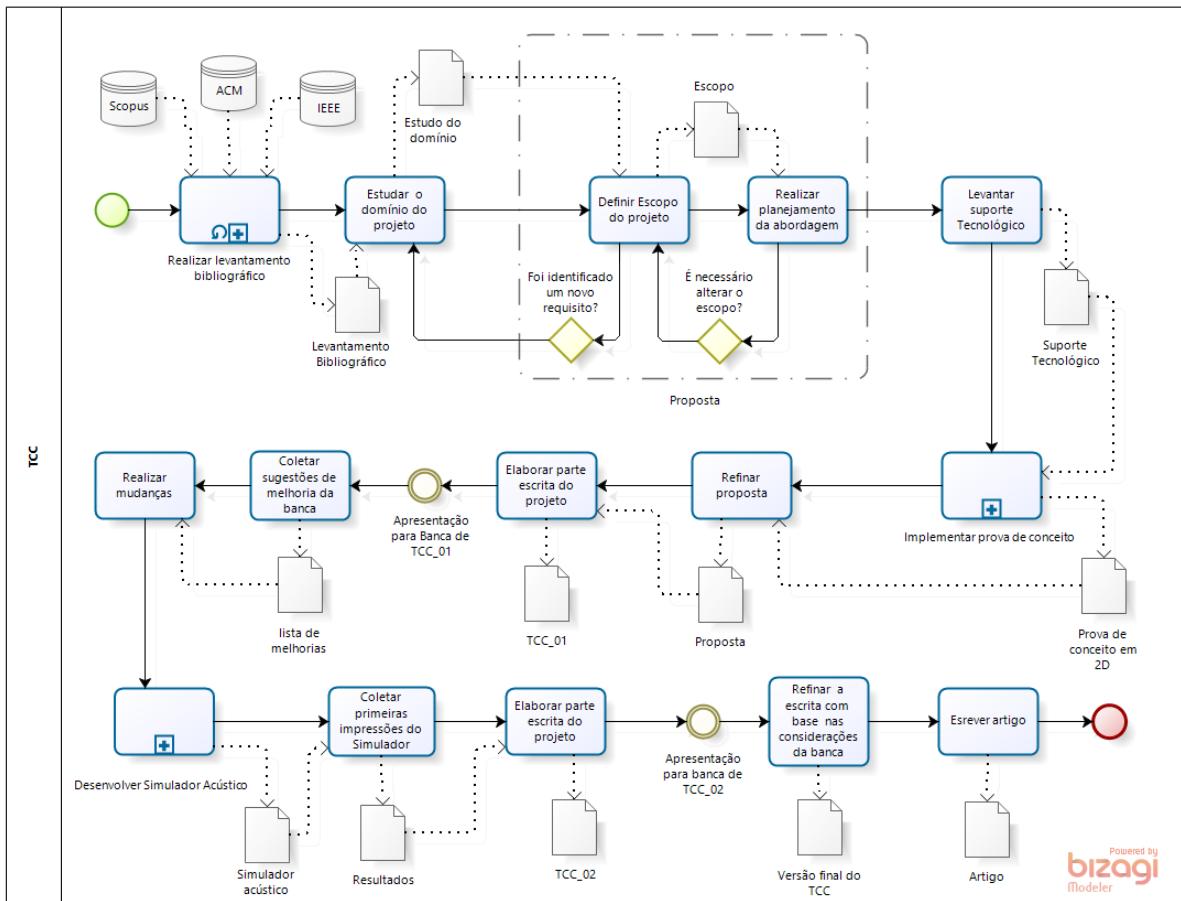


Figura 6 – Processo TCC

O desenvolvimento do simulador proposto neste trabalho foi conduzido considerando o modelo interativo incremental, onde foram realizadas algumas adaptações na metodologia ágil de desenvolvimento Scrum, figura 7. Para este projeto, foram definidas sprints com duração de uma semana.

Para a elicitação dos requisitos, foram levantadas histórias de usuário baseadas nos conceitos de acústica arquitetônica abordados no referencial teórico, assim como os conhecimentos adquiridos ao longo das disciplinas do curso de graduação em Engenharia de Software. O product backlog inicial deste trabalho está representado na tabela 3. Vale ressaltar que ocorreram algumas modificações durante o processo de desenvolvimento do simulador, assim como é previsto no Scrum.

Durante a execução deste trabalho, foi percebido que o uso de uma interface gráfica que pudesse auxiliar na configuração do ambiente assim como no controle e na visualização da simulação, poderia ser útil para nortear as atividades de desenvolvimento e tornar o simulador passível de uso a um usuário final, o que não era possível apenas com a máquina de raciocínio do simulador implementada.

Como a implementação da interface gráfica não estava prevista no planejamento

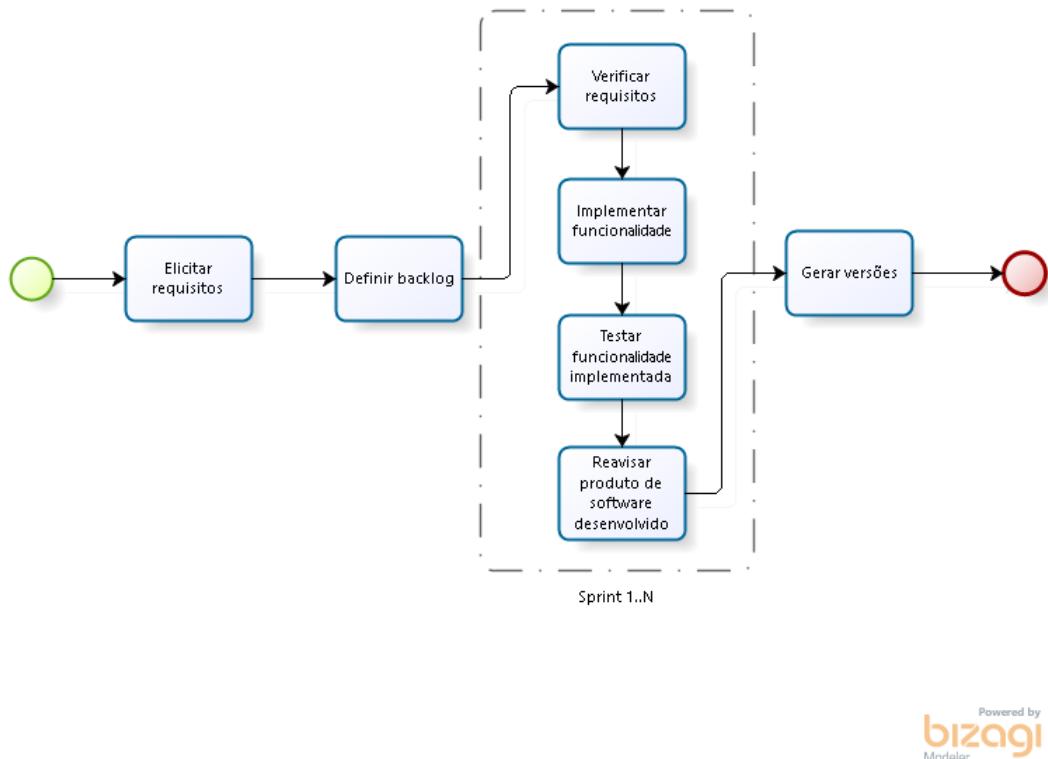


Figura 7 – Adaptação do Scrum para este TCC

inicial deste trabalho, foi preciso reduzir o escopo para conseguir completar essa atividade. Sendo assim, algumas alterações no product backlog inicial do projeto foram realizadas. Dentre essas alterações temos a modificação da US01: *"Como engenheiro, gostaria de definir as configurações de um ambiente tridimensional onde serão realizadas as simulações"*. O escopo dessa história foi reduzido, permitindo apenas a configuração de um ambiente bidimensional.

Também foi adicionada uma nova história para a implementação da interface gráfica com o usuário, onde será possível configurar, controlar e visualizar a simulação.

O product backlog final desse projeto pode ser visto na tabela 4.

Cada história foi dividida em tarefas antes de se iniciar sua implementação, essas tarefas foram úteis para entender melhor o problema e auxiliar a estimar a sua complexidade em relação ao seu desenvolvimento.

As tarefas relacionadas a cada história foram:

**US01:** Como engenheiro, gostaria de definir as configurações de um ambiente bidimensional onde serão realizadas as simulações.

- Implementar agente que irá controlar o ambiente de simulação.
- Implementar rotina para definição das dimensões do ambiente.

Identificador	História de usuário
US01	Como engenheiro, gostaria de definir as configurações de um ambiente tridimensional onde serão realizadas as simulações.
US02	Como engenheiro, gostaria de definir obstáculos dentro do ambiente de simulação assim como seus respectivos índices de absorção.
US03	Como engenheiro, gostaria de configurar uma fonte sonora e posicioná-la dentro do ambiente de simulação.
US04	Como engenheiro, gostaria de poder iniciar, pausar, retomar e parar uma simulação a qualquer momento após a configuração dos obstáculos e da fonte sonora.
US05	Como engenheiro, gostaria de poder remover um obstáculo ou fonte sonora da simulação.
US06	Como engenheiro, gostaria de poder definir a velocidade da simulação.
US07	Como engenheiro, gostaria de poder acompanhar o comportamento do som dentro do ambiente especificado durante a simulação.
US08	Como engenheiro, gostaria de acompanhar o nível de intensidade sonora dentro da sala durante a simulação.
US09	Como engenheiro, gostaria de saber qual foi o tempo de reverberação ao fim da simulação.
US10	Como engenheiro, quero um agente capaz de representar o som dentro de um ambiente fechado considerando sua propagação e suas reflexões.

Tabela 3 – Product Backlog Inicial

**US02:** Como engenheiro, gostaria de definir obstáculos dentro do ambiente de simulação assim como seus respectivos índices de absorção.

- Implementar classe Obstaculo.
- Implementar rotina para adição de obstáculos dentro do ambiente.
- Implementar rotina para configurar os parâmetros dos obstáculos, tais como dimensões, localização e índices de absorção.

**US03:** Como engenheiro, gostaria de configurar uma fonte sonora e posicioná-la dentro do ambiente de simulação.

- Implementar agente responsável por representar uma fonte sonora dentro do ambiente.
- Implementar comunicação entre os agentes Ambiente e FonteSonora.

Identificador	História de usuário
US01	Como engenheiro, gostaria de definir as configurações de um ambiente bidimensional onde serão realizadas as simulações.
US02	Como engenheiro, gostaria de definir obstáculos dentro do ambiente de simulação assim como seus respectivos índices de absorção.
US03	Como engenheiro, gostaria de configurar uma fonte sonora e posicioná-la dentro do ambiente de simulação.
US04	Como engenheiro, gostaria de poder iniciar, pausar, retomar e parar uma simulação a qualquer momento após a configuração dos obstáculos e da fonte sonora.
US05	Como engenheiro, gostaria de poder remover um obstáculo ou fonte sonora da simulação.
US06	Como engenheiro, gostaria de poder definir a velocidade da simulação.
US07	Como engenheiro, gostaria de poder acompanhar o comportamento do som dentro do ambiente especificado durante a simulação.
US08	Como engenheiro, gostaria de acompanhar o nível de intensidade sonora dentro da sala durante a simulação.
US09	Como engenheiro, gostaria de saber qual foi o tempo de reverberação ao fim da simulação.
US10	Como engenheiro, quero um agente capaz de representar o som dentro de um ambiente fechado considerando sua propagação e suas reflexões.
US11	Como engenheiro, gostaria de poder configurar, controlar e visualizar minha simulação por meio de uma interface gráfica.

Tabela 4 – Product Backlog Final

- Implementar rotina para emissão do pulso sonoro.

**US04:** Como engenheiro, gostaria de poder iniciar, pausar, retomar e parar uma simulação a qualquer momento após a configuração dos obstáculos e da fonte sonora.

- Implementar rotina para a verificação da existência obstáculos e fonte sonora.
- Implementar rotina de verificação de status da simulação (em execução, parada ou encerrada).
- Implementar rotina de execução da simulação.
- Implementar rotina de pausa da simulação em andamento.
- Implementar rotina para retomar uma simulação pausada.

- Implementar rotina para encerrar uma simulação.

**US05:** Como engenheiro, gostaria de poder remover um obstáculo ou fonte sonora da simulação.

- Implementar rotina de remoção de um obstáculo dentro do ambiente.
- Implementar rotina de remoção de uma fonte sonora dentro do ambiente.

**US06:** Como engenheiro, gostaria de poder definir a velocidade da simulação.

- Implementar rotina de configuração da velocidade da simulação.
- Implementar rotina de verificação da velocidade da simulação pelo agente sonoro.
- Implementar rotina de cálculo do tamanho do passo por tempo de atualização do agente sonoro.

**US07:** Como engenheiro, gostaria de poder acompanhar o comportamento do som dentro do ambiente especificado durante a simulação.

- Implementar rotina de atualização dos parâmetros do agente sonoro a serem acompanhados.
- Implementar rotina de monitoramento dos parâmetros de cada agente sonoro presente no ambiente durante a simulação.
- Implementar rotina de renderização da interface gráfica com o usuário.

**US08:** Como engenheiro, gostaria de acompanhar o nível de intensidade sonora dentro da sala durante a simulação.

- Implementar rotina de leitura dos níveis de intensidade sonora de cada som dentro do ambiente.
- Implementar rotina de cálculo de nível de intensidade sonora total dentro do ambiente.
- Implementar rotina de monitoramento do nível de intensidade sonora total dentro do ambiente.

**US09:** Como engenheiro, gostaria de saber qual foi o tempo de reverberação ao fim da simulação.

- Implementar critério de parada da simulação com base na diminuição em 60dB do nível de intensidade sonora inicial da simulação.
- Implementar rotina de cálculo do tempo de simulação com base na distância percorrida pelos agentes sonoros.
- Implementar rotina de cálculo do tempo de reverberação baseado no critério de parada da simulação.
- Implementar rotina de monitoramento do tempo de reverberação dentro do ambiente.

**US10:** Como engenheiro, quero um agente capaz de representar o som dentro de um ambiente fechado considerando sua propagação e suas reflexões.

- Implementar agente Sonoro.
- Implementar comunicação entre o agente sonoro e sua fonte sonora.
- Implementar rotinas de cálculo da propagação sonora.
- Implementar rotinas de identificação de obstáculos dentro do ambiente.
- Implementar rotinas de colisão do agente sonoro com obstáculos dentro do ambiente.
- Implementar rotinas de reflexão sonora.

**US11:** Como engenheiro, gostaria de poder configurar, controlar e visualizar minha simulação por meio de uma interface gráfica.

- Implementar interface com o usuário.
- Implementar recursos de visualização gráfica da simulação.

No Roadmap da tabela 5, é possível visualizar quais atividades foram realizadas em cada sprint, assim como suas respectivas datas de início e de término. Durante o desenvolvimento do simulador, foram dedicadas algumas sprints para atividades de refatoração do código fonte.

A tabela 6 apresenta o percentual de conclusão das tarefas de cada história de usuário presente no Backlog final deste simulador.

Tabela 5 – Roadmap

Sprint	Atividade	Data de início	Data de Término
Sprint 1	Implementar US01	07/06/2015	14/06/2015
Sprint 2	Implementar US02	14/06/2015	21/06/2015
Sprint 3	Implementar US03	21/06/2015	28/07/2015
Sprint 4	Implementar US10	28/06/2015	05/07/2015
Sprint 5	Implementar US05	05/07/2015	12/07/2015
Sprint 6	Implementar US04	12/07/2015	19/07/2015
Sprint 7	Atividade de Refatoração	19/07/2015	26/07/2015
Sprint 8	Implementar US06	26/07/2015	02/08/2015
Sprint 9	Implementar US11	02/08/2015	09/08/2015
Sprint 10	Implementar US07	09/08/2015	16/08/2015
Sprint 11	Implementar US08	16/08/2015	23/08/2015
Sprint 12	Implementar US09	23/08/2015	30/08/2015
Sprint 13	Atividade de Refatoração	30/08/2015	06/09/2015

Tabela 6 – Percentual de conclusão das tarefas

Estória	Tarefa	Percentual
01	Implementar agente que irá controlar o ambiente de simulação.	100%
	Implementar rotina para definição das dimensões do ambiente.	100%
02	Implementar classe Obstaculo.	100%
	Implementar rotina para adição de obstáculos dentro do ambiente.	100%
	Implementar rotina para configurar os parâmetros dos obstáculos, tais como dimenções, localização e índices de absorção.	100%
03	Implementar agente responsável por representar uma fonte sonora dentro do ambiente.	100%
	Implementar comunicação entre os agentes Ambiente e FonteSonora.	100%
	Implementar rotina para emissão do pulso sonoro.	100%
04	Implementar rotina para a verificação da existência obstáculos e fonte sonora.	100%
	Implementar rotina de verificação de status da simulação (em execução, parada ou encerrada).	100%
	Implementar rotina de execução da simulação.	100%
	Implementar rotina de pausa da simulação em andamento.	100%
	Implementar rotina para retomar uma simulação pausada.	100%
05	Implementar rotina para encerrar uma simulação.	100%
	Implementar rotina de remoção de um obstáculo dentro do ambiente.	100%

Continua na página seguinte

**Tabela 6 – Continuação da página anterior**

Estória	Tarefa	Percentual
	Implementar rotina de remoção de uma fonte sonora dentro do ambiente.	100%
06	Implementar rotina de configuração da velocidade da simulação.	100%
	Implementar rotina de verificação da velocidade da simulação pelo agente sonoro.	100%
	Implementar rotina de cálculo do tamanho do passo por tempo de atualização do agente sonoro.	100%
07	Implementar rotina de atualização dos parâmetros do agente sonoro a serem acompanhados.	100%
	Implementar rotina de monitoramento dos parâmetros de cada agente sonoro presente no ambiente durante a simulação.	100%
	Implementar rotina de renderização da interface gráfica com o usuário.	100%
08	Implementar rotina de leitura dos níveis de intensidade sonora de cada som dentro do ambiente.	100%
	Implementar rotina de cálculo de nível de intensidade sonora total dentro do ambiente.	100%
	Implementar rotina de monitoramento do nível de intensidade sonora total dentro do ambiente.	100%
09	Implementar critério de parada da simulação com base na diminuição em 60dB do nível de intensidade sonora inicial da simulação.	100%
	Implementar rotina de cálculo do tempo de simulação com base na distância percorrida pelos agentes sonoros.	100%
	Implementar rotina de cálculo do tempo de reverberação baseado no critério de parada da simulação.	100%
	Implementar rotina de monitoramento do tempo de reverberação dentro do ambiente.	100%
10	Implementar agente Sonoro.	100%
	Implementar comunicação entre o agente sonoro e sua fonte sonora.	100%
	Implementar rotinas de cálculo da propagação sonora.	100%
	Implementar rotinas de identificação de obstáculos dentro do ambiente.	100%
	Implementar rotinas de colisão do agente sonoro com obstáculos dentro do ambiente.	100%

Continua na página seguinte

**Tabela 6 – Continuação da página anterior**

Estória	Tarefa	Percentual
	Implementar rotinas de reflexão sonora.	100%
11	Implementar interface com o usuário.	100%
	Implementar recursos de visualização gráfica da simulação.	100%

## 4.5 Metodologia adotada na análise dos resultados

Cenário de Uso tem sua origem, enquanto "modalidade de pesquisa", em outros contextos, tais como estudos de mercado e modelos estratégicos ([TONI, 2006](#)).

Outro tópico existente sobre Cenários de Uso, no contexto da computação, é dentro do escopo de Casos de Uso. Nesse caso, apesar de existirem semelhanças entre os Cenários de Uso como algo complementar às modalidades de pesquisa conhecidas e Cenários de Uso no contexto de Caso de Uso, foge da noção desejada para o objetivo neste trabalho ([REZENDE, 2003](#)).

Para a análise dos resultados neste trabalho, foi utilizada uma abordagem baseada em pesquisa-ação combinada com cenário de uso. O objetivo é identificar problemas relevantes em cenários de uso especificamente desenhados para testes no simulador. Esses cenários compreenderão descritivos em termos de hardware, configuração, questões atendidas, resultados obtidos e ações realizadas em termos de refinamento.

## 4.6 Considerações finais

Este trabalho consiste no desenvolvimento de um simulador acústico, baseado em sistemas multiagentes, onde os sons são representados como agentes autônomos dentro do ambiente, sendo capazes de interagir com outros sons e com os recursos presentes no ambiente através da troca de mensagens e de forma assíncrona. Trata-se de uma pesquisa exploratória, cujos resultados foram analisados via pesquisa-ação.

A fim de refinar a proposta inicial do trabalho, que antes era baseada apenas no referencial teórico estudado, foi desenvolvida uma prova de conceito, onde foi criada uma representação bidimensional de um ambiente, com obstáculos e seus respectivos coeficientes de absorção, fontes sonoras e um conjunto de sons percorrendo e interagindo com os elementos presentes no sistema.

Com a implementação da prova de conceito foi possível identificar alguns problemas de desempenho que puderam ser reduzidos com um modelo de solução multiparadigma. O uso dos paradigmas orientado a objetos e multiagentes, além de melhorar o

desempenho do sistema implementado na prova de conceito, tornou o modelo mais claro, sendo mais intuitiva a representação de obstáculos como objetos de software.

Uma proposta refinada do simulador acústico foi elaborada ao final desse processo metodológico a partir das evidências observadas com a prova de conceito, bem como de ciclos iterativos de pesquisa-ação. Como resultado final tem-se o modelo escrito deste capítulo.



# 5 Resultados

Neste capítulo, serão apresentados os principais resultados obtidos usando uma abordagem híbrida quantitativa e qualitativa, conduzida com cenários de uso e pesquisação. Este capítulo apresenta dados referentes ao desenvolvimento da prova de conceito e do próprio simulador, que são considerados resultados deste trabalho. As subseções "Testes Unitários e Cobertura de Código" e "Métricas de Qualidade de Código" apresentam dados obtidos através de um levantamento quantitativo visando uma análise qualitativa do código fonte do simulador desenvolvido. Por fim, foi realizada análise dos cenários de uso levantados.

## 5.1 Prova de conceito

A fim de entender melhor o domínio da aplicação e verificar a viabilidade do projeto, foi desenvolvida uma prova de conceito. Essa prova de conceito permitiu que alguns elementos que não puderam ser identificados apenas na fase inicial do projeto com o referencial bibliográfico estudado, fossem identificados, podendo assim, compor uma proposta mais refinada da solução.

### 5.1.1 Especificações técnicas

As especificações técnicas da máquina, a qual foi utilizada para a realização do desenvolvimento da prova de conceito foram, um processador Intel Core i7, modelo 2630QM de 2Ghz e uma memória RAM de 6Gb. O sistema operacional utilizado foi o Ubuntu, versão 14.10 de 64 bits.

### 5.1.2 Descrição da prova de conceito

Para a construção dos agentes, optou-se pelo uso do *framework* JADE. O principal motivo desta escolha foi o fato de ser uma ferramenta reconhecida pela comunidade que trabalha com sistemas multiagentes, e por seguir os padrões e normas estabelecidos pela FIPA<sup>1</sup>. Outro fator que foi levado em consideração na escolha do *framework* foi o fato de ser uma plataforma estabilizada no processo de criação de agentes além da sua boa documentação. (FRANÇA et al., 2012). O JADE é inteiramente implementado na linguagem de programação JAVA, portanto, as aplicações que fazem uso deste *framework* assim como sua integração com o mesmo, orientam-se por meio dessa linguagem.

---

<sup>1</sup> <<http://www.fipa.org/>>

Com base na literatura estudada, identificamos alguns elementos físicos importantes, candidatos a serem representados no sistema. Esses foram representados de forma simples para esta prova de conceito, tais como: paredes, obstáculos e fontes sonoras. Esses elementos físicos são utilizados para compor o ambiente a ser representado dentro do sistema. O som também deve ser representado, porém, o conceito de onda sonora foi substituído pelo de raio sonoro. O som representado como raio sonoro pode ser entendido como retas dirigidas segundo a direção em que as mesmas caminham, permitindo assim, tratar melhor o evento da reflexão sonora ([SILVA, 1971](#)). O conceito de raio sonoro já é utilizado por grande parte dos produtos de software na área de simulação acústica.

A princípio, todos os elementos identificados foram tratados e implementados como agentes de software. O ambiente foi desenvolvido como o agente principal, responsável pela criação de todos os outros agentes, os quais, ao final, acabam por compor a estrutura do ambiente, com exceção do som, que deve ser criado pela fonte sonora dentro do sistema.

Esse modelo apresentou alguns problemas de desempenho, no qual tornou inviável o uso de um volume maior de agentes. Portanto, um novo modelo foi proposto. O modelo consiste em uma solução híbrida, composta pelos paradigmas multiagentes e orientado a objetos, onde os obstáculos do ambiente são tratados como objetos, devido ao fato de não terem um papel significativo na comunicação dos agentes, pois são elementos estáticos dentro do sistema. Os demais elementos como o ambiente, a fonte sonora e o som se mantiveram como agentes de software, sendo o ambiente responsável por agrupar os demais elementos do sistema, que são, fontes sonoras (agentes) e obstáculos (objetos).

A fonte sonora é o agente responsável pela criação dos sons, simulando o comportamento de pulso sonoro. Já os sons, são os agentes mais ativos dentro do projeto, sendo responsáveis por exercerem seu papel e percorrerem todo o ambiente, identificando as devidas colisões, reflexões e alterando seus parâmetros de intensidade sonora, direção, dentre outros, até que o mesmo atinja uma intensidade inaudível, ocasionando assim o fim do mesmo.

O agente som, dentro do sistema, é tratado como um ponto dentro do mapa do ambiente. A fonte sonora é o agente responsável pela criação desses agentes sonoros, emitindo assim um arco com uma abertura de acordo com a configuração da fonte. Por exemplo, caso uma fonte sonora tenha sido configurada para emitir um pulso sonoro com uma abertura de 90º e os agentes sonoros sejam criados com um intervalo de um grau, essa fonte sonora irá lançar 91 agentes sonoros a partir de sua origem, que no caso será a fonte sonora. A figura 8 ilustra esse evento.

A fonte sonora lança cada agente em uma determinada direção. A potência na qual a fonte sonora emite os agentes pode ser configurada. Após o agente sonoro ser emitido pela fonte, ele seguirá na direção em que foi lançado até encontrar um obstáculo, onde então, irá recalcular sua intensidade sonora com base no índice de absorção do

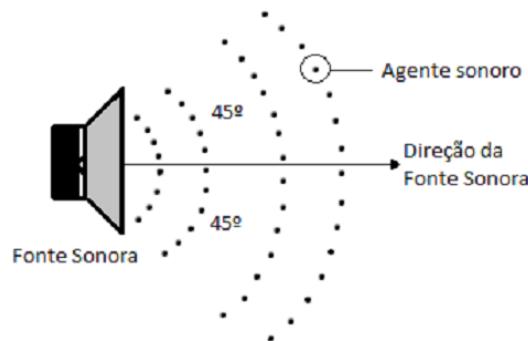


Figura 8 – Fonte Sonora

material do obstáculo encontrado, e irá alterar também sua direção com base no ângulo de inclinação do obstáculo. A cada reflexão, os agentes sonoros vão perdendo potência devido ao coeficiente de absorção sonora do material que reveste o obstáculo, até atingir uma intensidade sonora considerada inaudível ao ouvido humano.

Nesta prova de conceito, foram consideradas apenas duas dimensões para a representação simplificada do sistema, sendo assim, o som se desloca em um plano 2D e os obstáculos são definidos por linhas, conforme o exemplo de ambiente ilustrado na figura 9.

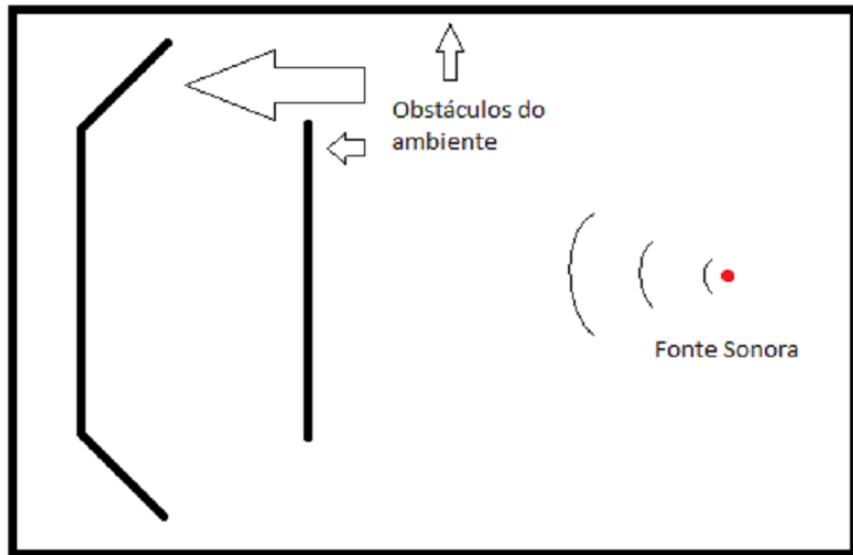


Figura 9 – Ambiente bidimensional

O comportamento dos agentes foi acompanhado a partir do JADE GUI, ferramenta gráfica nativa do JADE para acompanhamento dos agentes e suas trocas de mensagens. A figura 10 ilustra esta ferramenta.

Nesta abordagem híbrida para a solução da prova de conceito, o agente som se torna responsável por conhecer os obstáculos presentes no ambiente no qual está inserido,

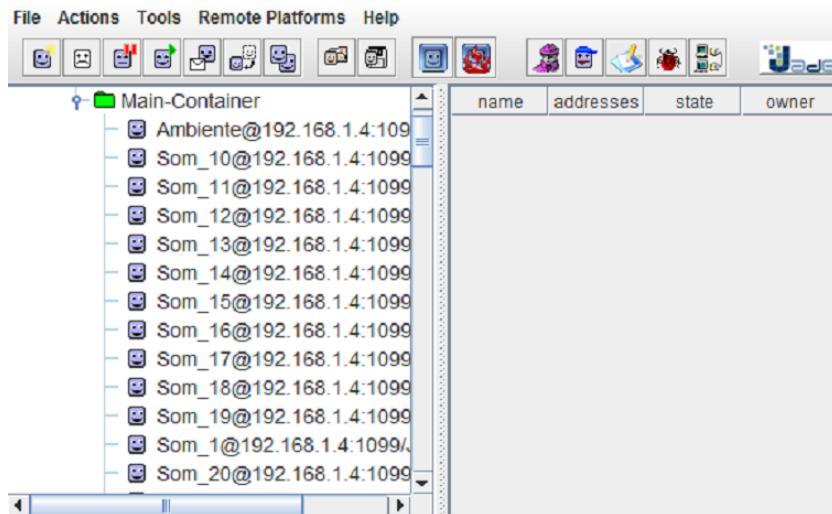


Figura 10 – JADE GUI

pois os obstáculos não participam ativamente do sistema, ou seja, não são capazes de realizar trocas de mensagens com os agentes. Como o ambiente é o agente que conhece todos os elementos presentes na sala, ele é responsável por informar à fonte sonora como estão mapeados os obstáculos dentro do ambiente. Dessa forma, quando a fonte sonora lança um pulso, ela é responsável por passar a cada agente sonoro essa informação no momento de sua criação.

### 5.1.3 Refinamento da proposta

Após a implementação da prova de conceito seguindo o modelo híbrido, onde os obstáculos são objetos e os demais elementos são agentes, foi possível trabalhar com um volume de até 447 agentes sonoros, ou seja, um agente a cada  $0,2^\circ$  considerando uma fonte com abertura de  $90^\circ$ , de forma segura, sem causar travamento do sistema e nem prejudicar a comunicação dos agentes. Porém, a velocidade da simulação cai significativamente, levando 3 minutos e 29 segundos até o final do último som no ambiente. Com um volume de 181 agentes, basicamente um agente a cada  $0,5^\circ$  em uma fonte sonora com abertura de  $90^\circ$ , foi possível realizar a simulação a um tempo de 1 minuto e 24 segundos. Ambos os resultados foram mais satisfatórios que a proposta inicial do sistema totalmente baseada em multiagentes, não apenas pelo tempo, mas também pelo fato de a primeira proposta não suportar um volume maior que 91 agentes sem apresentar problemas na troca de mensagens, ocasionando um erro associado à identificação de colisão dos agentes com os obstáculos.

Vale ressaltar que as configurações de máquina utilizadas nessa simulação não são ideais. Sistemas multiagentes são intrinsecamente distribuídos e para obter melhores resultados em termos de desempenho, seria necessário utilizar mais de uma máquina. Entretanto, tal recurso não foi viável durante a condução do experimento.

## 5.2 O simulador acústico

Esta seção trata de aspectos do simulador acústico desenvolvido neste trabalho, tais como, arquitetura, como realizar uma simulação dentro do sistema e as devidas verificações e validações realizadas no mesmo.

### 5.2.1 Descrição do simulador

A partir do referencial bibliográfico estudado neste trabalho e da prova de conceito desenvolvida no decorrer do TCC 01, foi possível definir que o simulador, objeto de estudo deste trabalho, consiste em um sistema que seja capaz de representar o comportamento sonoro dentro de um ambiente através de agentes de software, que sejam capazes de interagir de forma autônoma com os elementos do ambiente representados no sistema.

Com base na simulação realizada na prova de conceito, foi possível identificar que, por questões de desempenho, a solução do sistema final deveria ser composta por pelo menos dois paradigmas, o paradigma orientado a objetos e o de sistemas multiagentes.

Para a solução final, o sistema foi evoluído a fim de apresentar graficamente o andamento da simulação e o comportamento do som dentro do ambiente. O comportamento dos agentes sonoros também foi aprimorado. O nível de intensidade sonora no ambiente cai a medida que o som percorre o ambiente e sofre colisões com os obstáculos presentes, considerando uma taxa de decaimento em um ambiente tridimensional, mesmo a visualização da simulação sendo bidimensional. Tal contexto permite calcular o tempo de reverberação no ambiente considerando o parâmetro RT60, que representa o tempo em que o nível de intensidade sonora em um ambiente diminui em 60dB ([NETO, 2007](#)).

Tanto os agentes quanto os objetos do sistema foram escritos na linguagem de programação Java. A ferramenta utilizada para controle de versão é o Git, juntamente com o repositório GitHub, afim de promover a cultura de software livre e abrir espaço para a comunidade que deseja contribuir ou estudar sobre o assunto.

### 5.2.2 Arquitetura

Este simulador foi desenvolvido com base no modelo de arquitetura em camadas, onde temos duas camadas principais: apresentação e negócio. A camada de apresentação é responsável pela interface com o usuário. Nesta camada estão situados todos os componentes gráficos do sistema. Tais componentes foram desenvolvidos utilizando o pacote Swing do Java, juntamente com a biblioteca JFreeChart para plotagem dos gráficos.

Já a camada de negócio comprehende a máquina de raciocínio deste simulador, onde estão situados os agentes e os objetos que atuam nas principais funcionalidades do sistema. Toda essa camada foi desenvolvida utilizando a linguagem de programação Java.

Os agentes de software foram implementados com o apoio do *framework* JADE. A figura 11 ilustra o modelo de arquitetura em camadas descrito.

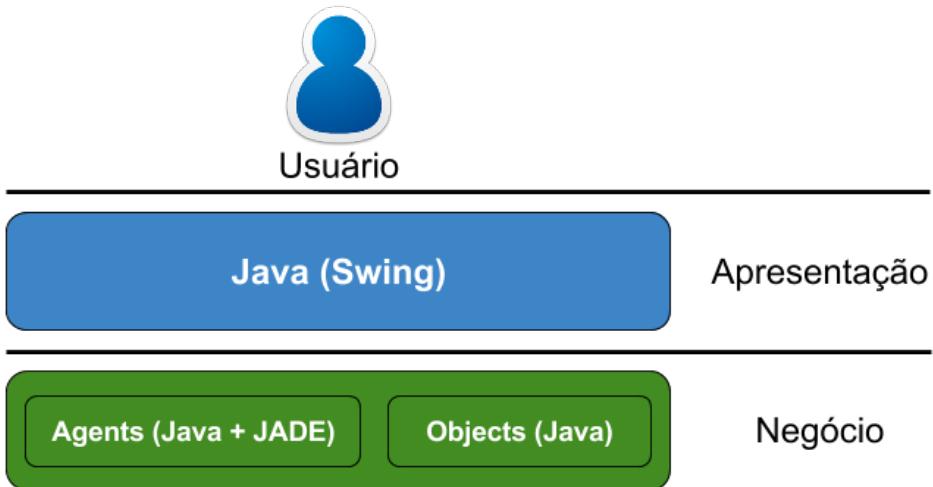


Figura 11 – Arquitetura visão geral

Dada a camada de negócio, o sistema é composto por três tipos de agentes, sendo eles, ambiente, fonte sonora e sons. Cada um possui um objeto correspondente, onde estão implementados os métodos que irão compor seus comportamentos. Outro elemento do sistema é o obstáculo, porém, este deixou de ser um agente, sendo tratado, no momento, como objeto também.

O Ambiente é o agente responsável pela criação das fontes sonoras que, por sua vez, são responsáveis pela criação dos sons. Cada um desses agentes possui um objeto relacionado, por exemplo, para criar um agente do tipo *SoundAgent*, é necessário passar um objeto do tipo *Sound* como argumento, assim como é preciso passar um objeto do tipo *SoundSource* para criar um agente do tipo *SoundSourceAgent*. Esses objetos são responsáveis por armazenar os valores dos atributos do agente correspondente e implementar os métodos que irão compor seus comportamentos.

Outro elemento importante para se realizar uma simulação é o obstáculo. A classe *Obstacle*, possui os atributos e métodos necessários para instanciar um objeto do tipo, como por exemplo, o índice de absorção, que por sua vez, será consultado toda vez que um agente sonoro colidir com o obstáculo. Este índice é importante para que o agente som possa calcular sua intensidade sonora após uma colisão, pois parte da sua energia será absorvida pelo obstáculo. O índice de absorção de um material varia de acordo com a frequência em que o som se encontra no momento da colisão. A classe *Obstacle* também é responsável por manter um *HashMap* com todos os obstáculos criados e seus respectivos IDs, para que os agentes possam consultá-los quando necessário.

O modelo da arquitetura dessa máquina de raciocínio está ilustrado na figura 12.

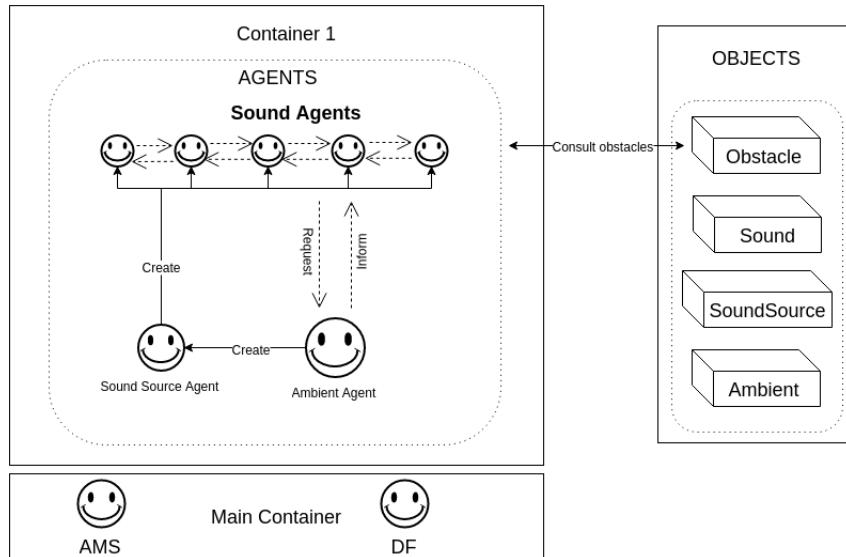


Figura 12 – Arquitetura da máquina de raciocínio do Simulador Acústico

Além da arquitetura base do sistema, também é implementada em uma camada transversal a todo o sistema, correspondente à arquitetura do *framework* JADE<sup>2</sup>. O JADE é utilizado para a implementação dos agentes neste projeto, provendo funcionalidades na camada de *middleware*. Este *framework* é interoperável, permitindo a comunicação entre agentes JADE e outros tipos de agentes fora da plataforma.

O JADE faz uso da abstração de *containers*, nos quais estão os agentes. Agentes podem ou não estar distribuídos na rede, sendo que cada *container* pode conter um ou mais agentes. Esta plataforma deve conter obrigatoriamente o *Main Container*, um *container* responsável pelo gerenciamento de agentes, composto pelos agentes AMS e DF. O AMS (Agent Management System) é único na plataforma e é responsável por conhecer todos os agentes presentes no sistema, pois é ele quem exerce o controle sobre o acesso e o uso da plataforma, além de manter a lista de identificadores dos agentes (AID). Portanto, todos os agentes criados devem ser registrados no AMS.

Já o agente DF (Directory Facilitator), oferece todas as interações básicas da plataforma, além de oferecer um conjunto de métodos para a implementação do comportamento do agente. Nem todos os agentes são registrados no DF, apenas aqueles que prestam algum serviço. O modelo da arquitetura do JADE pode ser visualizado na figura 13.

Os Agentes JADE possuem um ciclo de vida, conforme ilustrado na figura 14. O primeiro passo desse ciclo de vida é a criação do agente, seguido de sua inicialização. Ao ser inicializado o agente se torna ativo para execução de tarefas ou comportamentos, podendo ser suspenso e retornar ao estado de ativo posteriormente. Os agentes também podem entrar em modo de espera em busca de algum recurso que outro agente está

<sup>2</sup> <<http://jade.tilab.com/>>

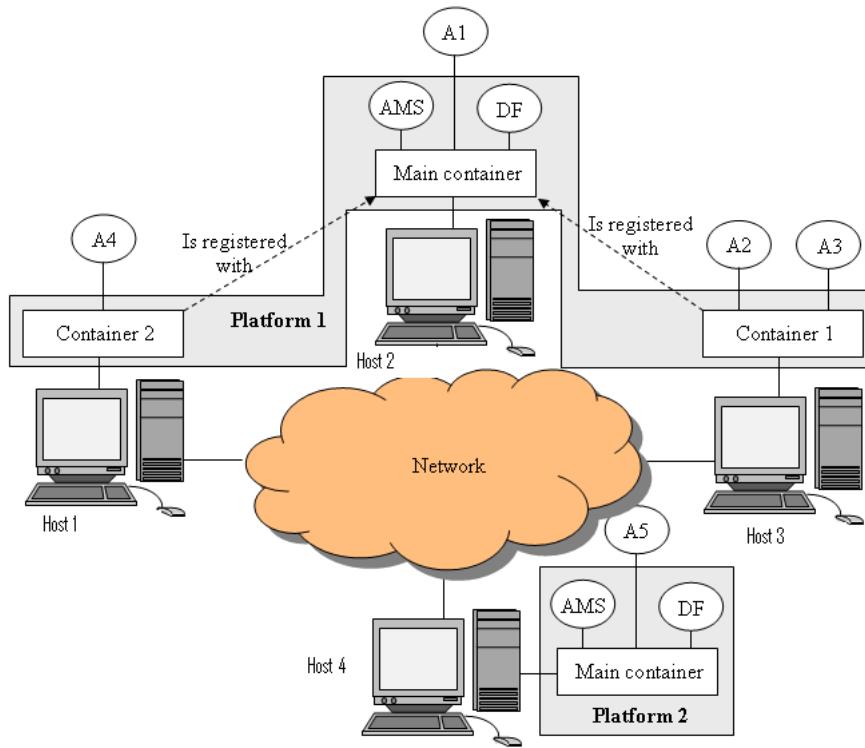


Figura 13 – Arquitetura JADE<sup>1</sup>

executando, e retornar quando este recurso for liberado. Outro estado é quando o agente está em trânsito, se movimentando pela rede, porém, independente do estado em que o agente esteja, ele pode ser destruído.

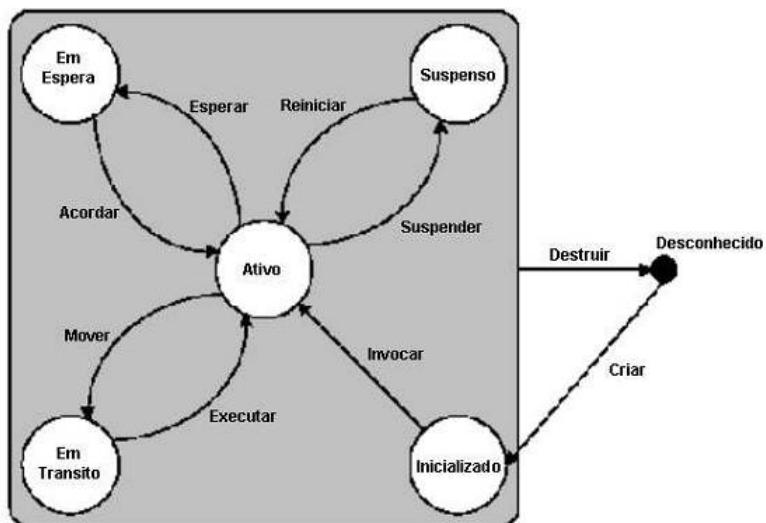


Figura 14 – Ciclo de vida do agente JADE ([SILVA, 2003](#))

A estrutura dos pacotes deste simulador pode ser visualizada na figura 15.

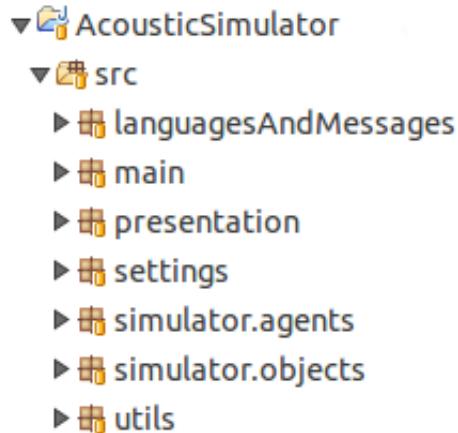


Figura 15 – Estrutura de pacotes do simulador acústico

### 5.2.3 Realizando uma simulação

Esta seção evidencia como utilizar o simulador acústico. O usuário pode definir um ambiente e suas dimensões, assim como inserir obstáculos e fontes sonoras dentro dele. Também é possível que o usuário inicie, pause, retome ou encerre uma simulação a qualquer momento.

O código do simulador acústico é aberto e está disponível no GitHub<sup>3</sup>.

#### 5.2.3.1 Tela inicial do simulador

Ao iniciar o simulador acústico, irá aparecer a tela principal do sistema, onde está localizada a maior parte de suas funcionalidades. No canto superior esquerdo, é possível observar o menu, com as opções *action* e *edit*, assim como a barra de ferramentas do sistema. Logo abaixo, na tela, é possível visualizar o espaço para as tabelas de obstáculos e fontes sonoras da simulação. À direita, o espaço para visualizar a simulação em andamento, conforme é evidenciado na figura 16.

#### 5.2.3.2 Adicionando obstáculos

O primeiro passo ao realizar uma simulação consiste em adicionar os obstáculos que irão compor o ambiente. Cada obstáculo possui um ponto inicial, um ponto final e um índice de absorção associado.

Há duas formas de inserir obstáculos. A primeira delas é informar o índice de absorção e as dimensões do ambiente. Após informar as dimensões, o software cria quatro obstáculos com o índice de absorção especificado compondo as delimitações do ambiente. Para utilizar este recurso, basta clicar no menu "Edit -> Define Ambient" ou utilizar

<sup>3</sup> <<https://github.com/gabriel-augusto/AcousticSimulator>>

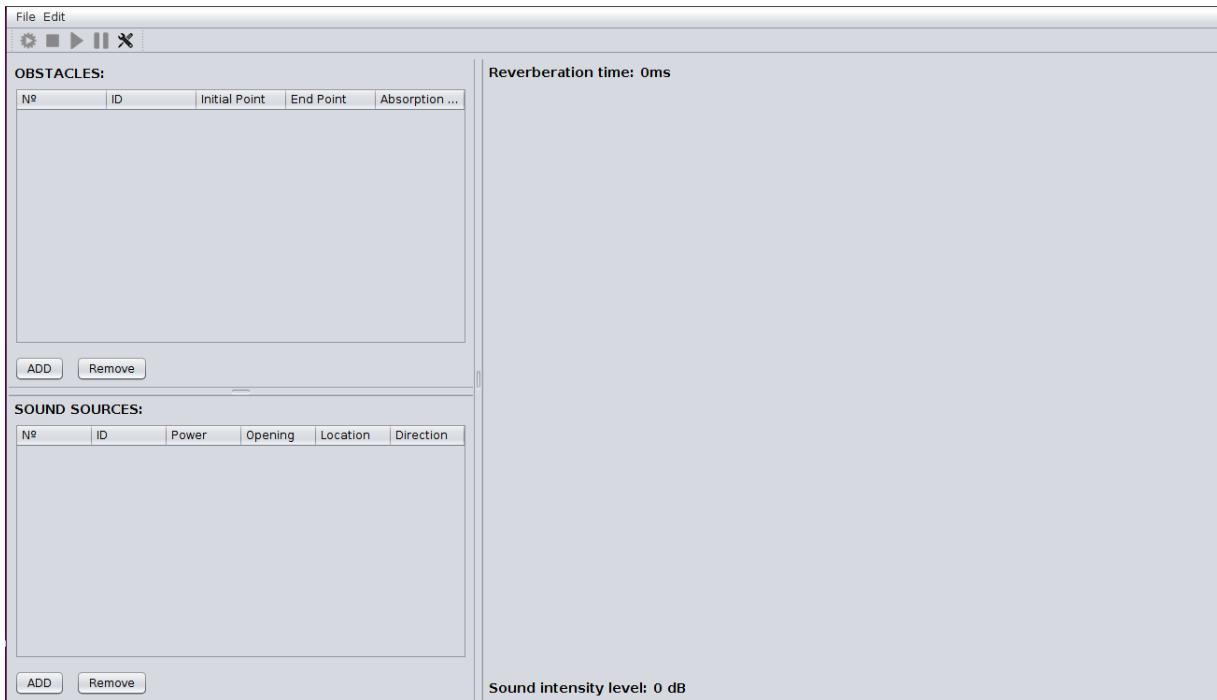


Figura 16 – Tela inicial do simulador acústico

as teclas de atalho "Ctrl+D" e informar os dados ao sistema na tela de configuração do ambiente que está evidenciada na figura 17.

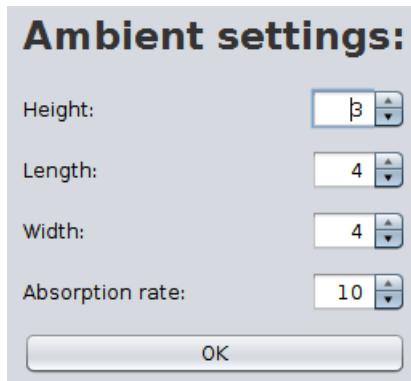


Figura 17 – Tela de configuração do ambiente

Outra forma de se adicionar obstáculos ao ambiente, é inseri-los individualmente. Para isto, basta clicar no menu "Edit -> Add Obstacle" ou clicar no botão "ADD" abaixo da tabela de obstáculos.

Para definir um obstáculo, é necessário informar as coordenadas do seu ponto inicial e final, assim como seu respectivo índice de absorção na tela de criação de obstáculos, evidenciada na figura 18. Dessa forma, o sistema irá criar um obstáculo em forma de linha reta dentro do seu ambiente. Vale ressaltar que, apesar da máquina de raciocínio do simulador estar preparada para trabalhar com diversos obstáculos, devido ao uso da biblioteca

de gráficos JFreeChart, não foi possível plotar esses obstáculos dentro do ambiente, uma vez que eles são representados como linhas retas e os sons são representados como pontos.

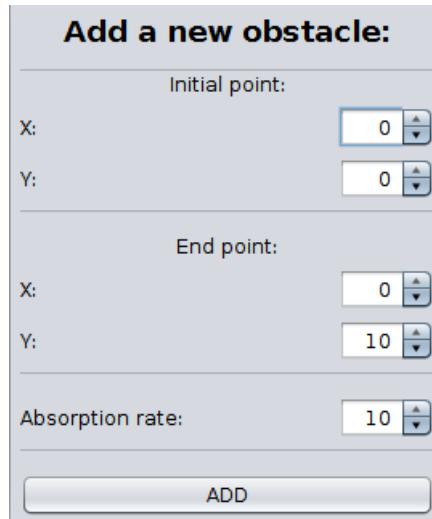


Figura 18 – Tela de criação de obstáculos

Todos os obstáculos criados, assim como seus respectivos parâmetros, podem ser visualizados na tabela de obstáculos na tela principal do sistema, a qual está evidenciada na figura 19. Abaixo desta tabela, há dois botões com as opções de adicionar e remover, permitindo realizar estas ações de forma rápida e prática. Para remover um obstáculo do ambiente, basta selecioná-lo na tabela de obstáculos e clicar no botão remover.

OBSTACLES:				
Nº	ID	Initial Point	End Point	Absorption Rate
1	Ob-1	(x: 0.0; y: 0.0)	(x: 0.0; y: 15.0)	10.0
2	Ob-2	(x: 0.0; y: 15.0)	(x: 20.0; y: 15.0)	10.0
3	Ob-3	(x: 20.0; y: 15.0)	(x: 20.0; y: 0.0)	10.0
4	Ob-4	(x: 20.0; y: 0.0)	(x: 0.0; y: 0.0)	10.0
5	Ob-5	(x: 4.0; y: 5.0)	(x: 17.0; y: 10.0)	20

ADD      Remove

Figura 19 – Tabela de obstáculos

### 5.2.3.3 Adicionando fontes sonoras

Após definir as delimitações do ambiente e seus obstáculos, é o momento de inserir uma ou mais fontes sonoras. Para realizar uma simulação, é necessário ter no mínimo uma

fonte sonora dentro do ambiente.

Existem três maneiras de abrir a tela de configurações da fonte sonora. Essa tela está ilustrada na figura 20. A primeira delas é através do menu "Edit -> Add Sound Source". Esta tela também pode ser acessada através do botão "ADD", abaixo da tabela de fontes sonoras, ou através das teclas de atalho "Ctrl + H".

As configurações de uma fonte sonora incluem seu ângulo de abertura em graus, suas coordenadas dentro do ambiente, a direção para a qual está apontada e o número de agentes sonoros que serão emitidos pela fonte. O número de agentes sonoros reflete na precisão da simulação, ou seja, quanto mais agentes sonoros são emitidos, mais fiel se torna o resultado da simulação. Entretanto, um número maior de agentes requer mais recursos computacionais.

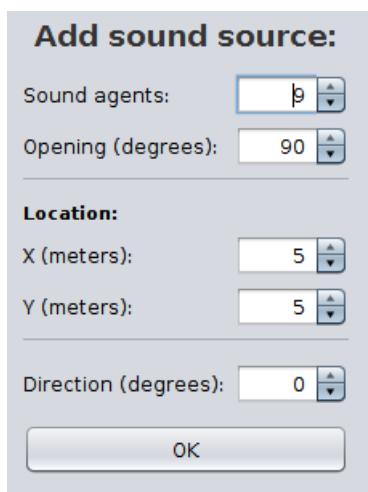


Figura 20 – Tela de configuração da fonte sonora

Igualmente aos obstáculos, todas as fontes sonoras criadas, assim como seus respectivos parâmetros, podem ser visualizados na tabela de fontes sonoras na tela principal do sistema, conforme ilustrado na figura 21. Esta tabela é bastante parecida com a tabela de obstáculos, contendo os dois botões com as opções de adicionar e remover, permitindo realizar estas ações de forma rápida e prática. Para remover uma fonte sonora do ambiente, basta selecioná-la na tabela de fontes sonoras e clicar no botão remover.

#### 5.2.3.4 Configurando a velocidade da simulação

A velocidade da simulação representa o tempo em que a posição dos agentes sonoros é atualizada. Para configurar essa velocidade, é necessário acessar a tela de configurações da simulação, evidenciada na figura 22, através do menu "edit -> simulation settings"ou através do ícone "Simulation Settings"na barra de ferramentas. As teclas de atalho para acessar essa tela são "Ctrl + I".

SOUND SOURCES:					
Nº	ID	Nº of Sounds	Opening	Location	Direction
1	SS-1	9	90	(x: 5.0; y: 5.0)	0
<input type="button" value="ADD"/>		<input type="button" value="Remove"/>			

Figura 21 – Tabela de fontes sonoras



Figura 22 – Tela de configurações da simulação

#### 5.2.3.5 Executando uma simulação

Após definir as dimensões do ambiente, inserir os obstáculos e fontes sonoras e configurar a velocidade, é possível executar a simulação. Com todos os itens configurados, há três maneiras de executar uma simulação. A primeira delas é através do menu "Action -> run simulation". Outra opção é através do ícone "Run simulation", na barra de ferramentas e por essa opção estar localizada diretamente na tela inicial do sistema, ela pode ser acessada de forma mais fácil e intuitiva. Há também a possibilidade de executar a simulação através das teclas de atalho "Ctrl + R".

Com a simulação em execução, é possível pausar, resumir ou encerrar a simulação à qualquer momento de sua execução. Todas essas opções estão disponíveis no menu "Action" ou na barra de ferramentas do sistema que está evidenciada na figura 23.



Figura 23 – Barra de ferramentas

Durante a execução da simulação, é possível visualizar o comportamento do som dentro do ambiente na tela inicial do sistema, conforme evidenciado na figura 24. Os sons são representados graficamente por pontos azuis, enquanto as fontes sonoras são representadas por pequenos quadrados verdes.

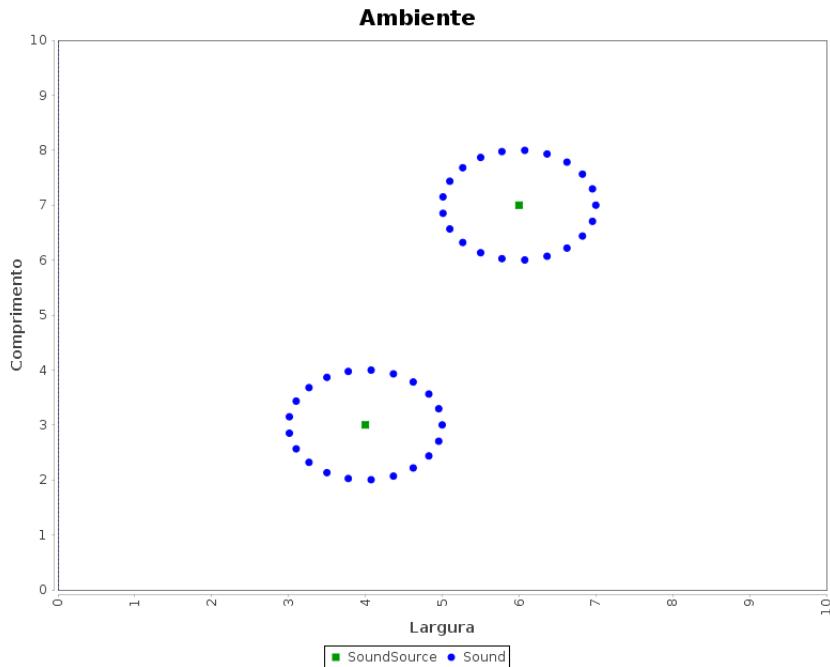


Figura 24 – Visualização gráfica da simulação em andamento.

Ao fim da simulação, o campo com o valor do tempo de reverberação, localizado acima do espaço destinado ao acompanhamento da simulação, é atualizado, apresentando então, o tempo de reverberação do ambiente configurado.

#### 5.2.4 Testes unitários e cobertura de código

Pra realização dos testes unitários e análise de cobertura de código fonte, foram adotadas as ferramentas JUnit<sup>4</sup> e EclEmma<sup>5</sup> respectivamente. Devido ao fato de os agentes comportamentais não possuirem uma ferramenta de validação comportamental adequada até o presente momento, foram criados objetos relacionados a cada um dos agentes, onde todas as rotinas utilizadas pelos comportamentos dos agentes foram migradas para seus respectivos objetos. Dessa forma, foi possível testar as rotinas utilizando as ferramentas adotadas. Porém, os agentes em si não puderam ser completamente testados.

Devido ao fato de não haver uma ferramenta de validação adequada para agentes comportamentais, foi estabelecida uma meta de 80% de cobertura de testes. O quadro com a cobertura de testes pode ser visualizado na figura 25. Como é possível observar, a cobertura esperada foi alcançada.

<sup>4</sup> <<http://junit.org/>>

<sup>5</sup> <<http://eclemma.org/>>

test (27/10/2015 17:00:13)				
Element	Coverage	Covered Instruc	Missed Instructi	Total Instructions
▼  AcousticSimulator	90,0 %	4.351	485	4.836
▼  src	82,3 %	2.259	485	2.744
►  languagesAndMessages	94,5 %	103	6	109
►  settings	100,0 %	27	0	27
►  simulator.agents	42,6 %	302	407	709
►  simulator.objects	96,6 %	1.667	58	1.725
►  utils	92,0 %	160	14	174

Figura 25 – Cobertura de código.

### 5.2.5 Métricas de qualidade de código fonte

Esta seção evidencia o resultado da qualidade do código fonte do simulador desenvolvido durante esse trabalho. Através da análise estática de código fonte foi possível obter métricas referentes ao código do simulador e realizar uma análise qualitativa dos resultados obtidos. A ferramenta adotada para realizar a análise foi a Analizo<sup>6</sup>

Segundo Filho (2013), medir e monitorar a qualidade do software é fundamental, independente da metodologia adotada para o desenvolvimento. Mesmo uma ótima bateria de testes pode produzir informações apenas sobre características externas, não refletindo qualidades como manutenibilidade, modularidade, flexibilidade e simplicidade (FILHO, 2013).

Para realizar a análise estática do código, foram utilizadas as métricas e seus respectivos valores de referência baseados na dissertação de Filho (2013). Tais valores foram obtidos baseando-se nos resultados da análise de um ou mais "projetos modelo", analisados no Capítulo 4 da tese de Meirelles (2013).

Dentre as métricas suportadas pelo Analizo, Meirelles (2013) seleciona um subconjunto representativo, pois foi constatado que muitas delas eram redundantes por medir prioridades muito correlacionadas. As métricas selecionadas abrangem diversos aspectos de uma classe:

1. Conexões aferentes, ou ACC, uma medida de acoplamento.
2. Média da complexidade ciclomática dos métodos, ou ACCM.
3. Média do tamanho dos métodos, ou AMLOC.
4. Média do número de parâmetros por método, ou ANPM.
5. Profundidade na árvore de herança, ou DIT.
6. Número de métodos, ou NOM.
7. Número de atributos públicos, ou NPA.

<sup>6</sup> <<http://www.analizo.org/>>

8. Complexidade estrutural, ou SC, uma medida que combina acoplamento (CBO) e coesão (LCOM4).

A figura 26 apresenta os resultados da análise estática realizada. É possível observar que todos os valores ficaram entre bom e ótimo, segundo os intervalos sugeridos por Filho (2013). Devido a esses resultados, e as atividades contínuas de refatoração durante o desenvolvimento do simulador, não foi realizada outra refatoração após a análise estática do código fonte.

	<b>ACC</b>	<b>ACCM</b>	<b>AMLOC</b>	<b>ANPM</b>	<b>DIT</b>	<b>NOM</b>	<b>NPA</b>	<b>SC</b>
<b>Excelente</b>	[0,2]	[0,3]	[0,8]	[0,2]	[0,2]	[0,10]	[0,1]	[0,12]
<b>Bom</b>	[2,7]	[3,5]	[8,19]	[2,3]	[2,4]	[10,17]	[1,2]	[12,28]
<b>Regular</b>	[7,15]	[5,7]	[19,37]	[3,5]	[4,6]	[17,27]	[2,3]	[28,51]
<b>Preocupante</b>	[15,∞]	[7,∞]	[37,∞]	[5,∞]	[6,∞]	[27,∞]	[3,∞]	[51,∞]
<b>Valores obtidos:</b>	1,64	1,63	12,07	0,68	0,42	8,15	0,79	17,24

Figura 26 – Análise estática do código fonte.

### 5.2.6 Comparação entre os cenários de uso avaliados

Para validar se os resultados gerados pelas simulações realizadas pelo simulador acústico desenvolvido neste trabalho estavam de acordo com o esperado, foram coletados os dados de mais dois simuladores que calculam o tempo de reverberação do ambiente através de seus parâmetros. Para que a comparação com o simulador acústico desenvolvido fosse possível, era necessário que o simulador a ser comparado permitisse a visualização dos coeficientes de absorção dos materiais que ele disponibiliza, pois qualquer diferença nesses coeficientes, implica em uma diferença considerável no resultado final da simulação. Este fato restringiu bastante a possibilidade de comparação com diversas ferramentas disponíveis no mercado.

Por fim, foram encontradas duas ferramentas que permitem visualizar os coeficientes de absorção de seus materiais ou a média final dos coeficientes de absorção adotados. A primeira delas é a ferramenta de cálculo de tempo de reverberação da hyperphysics<sup>7</sup>. Esta ferramenta permite editar manualmente os coeficientes de absorção de cada um dos obstáculos adicionados, possibilitando realizar a simulação com parâmetros completamente idênticos. A segunda é a rt60calc<sup>8</sup> da Belgi. Esta ferramenta, apesar de não permitir informar o valor dos coeficientes de absorção diretamente, permite inserir os materiais desejados e informa o coeficiente médio ao final, possibilitando atingir manipular os ma-

<sup>7</sup> <<http://hyperphysics.phy-astr.gsu.edu/>>

<sup>8</sup> <<http://www.belgi.net/audiocorp/techinfo/rt60calc.htm>>

Tabela 7 – Cenários avaliados

Dados do ambiente	Cenário 1	Cenário 2	Cenário 3
Largura	4 m	3 m	25 m
Comprimento	4 m	7 m	20 m
Altura	3 m	4 m	6 m
Absorção média	9 %	10 %	20 %

Tabela 8 – Resultados das simulações

Ferramenta	Cenário 1	Cenário 2	Cenário 3
Acoustic Simulator	1073 ms	1109 ms	1568 ms
belgi rt60calc	1070 ms	1106 ms	1563 ms
hyperphysics	1073 ms	1108 ms	1568 ms

teriais disponíveis e atingir valores bem próximos. Ambas as ferramentas foram adotadas para serem usadas nas comparações.

As especificações técnicas da máquina, a qual foi utilizada para a realização da comparação entre os cenários de uso levantados foram: um processador Intel Core i7, modelo 2630QM de 2Ghz e uma memória RAM de 6Gb. O sistema operacional utilizado foi o Ubuntu, versão 14.04.3 LTS de 64 bits e a versão do Java utilizada foi a "1.8.0\_66".

Para realizar as comparações, foram criados três cenários, o primeiro foi um ambiente com 4m de largura, 4m de comprimento e 3m de altura. Para o segundo cenário, foi especificado um ambiente um pouco maior, possuindo 3m de largura, 7m de comprimento e 4m de altura. Por fim, para o último cenário foi especificado um ambiente bem maior, possuindo 25m de largura, 20m de comprimento e 6m de altura. O coeficiente médio de absorção foi de 20% para todos os cenários. Essas configurações podem ser vistas na tabela 7.

Após especificar os cenários a serem analisados em cada uma das ferramentas, foi realizada a simulação em cada um delas. Após a realização da simulação, os dados referentes ao tempo de reverberação foram coletados. Os resultados podem ser visualizados na tabela 8.

Os resultados das simulações nos três cenários foram bem próximos. O gráfico da figura 27 demonstra a proximidade entre os tempos de reverberação entre as ferramentas para cada um dos cenários avaliados nessa comparação. Para afirmar tal proximidade, foi calculado o coeficiente de variação, medida de dispersão que representa a homogeneidade entre os dados coletados expressa pela seguinte equação:  $cv = 100 \frac{\sigma}{\mu}$ , onde  $cv$  é o coeficiente de variação,  $\sigma$  é o desvio padrão e  $\mu$  é a média.

Considerando os valores obtidos para o coeficiente de variação, foi possível concluir que a dispersão entre os valores dos simuladores é mínima, incluindo o simulador acústico desenvolvido neste trabalho, uma vez que todos os valores ficaram abiaixo de 1%. Sendo

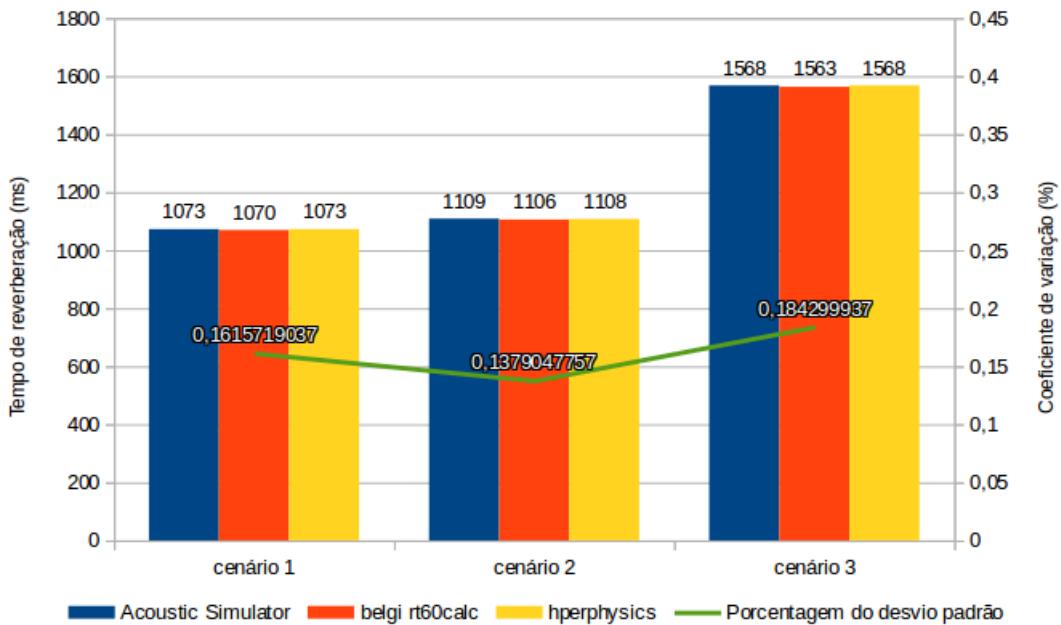


Figura 27 – Tempo de reverberação dos simuladores por cenário.

assim, os valores de tempo de reverberação obtidos pelo simulador acústico desenvolvido neste trabalho (i.e. Acoustic Simulator), estão dentro do esperado.

### 5.3 Considerações finais

A proposta deste sistema consiste em um sistema de simulação acústica capaz de representar um ambiente e, a partir deste sistema, por meio de uma fonte sonora, emitir agentes sonoros capazes de interagir com os elementos presentes no ambiente até que o nível de intensidade sonora dentro do ambiente caia em 60dB, onde então o som é caracterizado como finalizado e o agente é destruído. O parâmetro que simboliza o tempo de reverberação no ambiente, considerando essa redução do nível de intensidade sonora do ambiente em 60db, é denominado RT60.

A máquina de raciocínio deste sistema foi construída utilizando a linguagem de programação Java e é baseada na arquitetura de agentes de software, utilizando recursos como a troca de mensagens, assincronismo e autonomia. Porém, para que isto fosse possível, o sistema precisou rodar por cima da arquitetura do *framework* JADE. Trata-se de uma ferramenta escolhida para a implementação dos agentes deste simulador. Adicionalmente, a camada de apresentação do simulador foi implementada utilizando o pacote Swing do Java juntamente com a biblioteca JFreeChart, para a criação dos gráficos em 2D.

Quanto aos testes, foram realizados testes de unidade para validar os módulos do sistema, onde o nível de cobertura adotado foi de 80%. Afim de verificar a qualidade do

código fonte, foi utilizada a ferramenta Analizo para a coleta de métricas de código fonte. As métricas consideradas foram: ACC, ACCM, AMLOC, ANPM, DIT, NOM, NPA e SC. Tais métricas foram escolhidas com o intuito de evitar redundância entre as métricas suportadas pela ferramenta Analizo, segundo o estudo de Filho (2013).

Pra analizar os resultados finais obtidos pelas simulações do simulador desenvolvido neste trabalho, foram feitas comparações em três cenários diferentes com as ferramentas rt60calc e hyperphysics. A proximidade entre os valores da simulação foi analisada considerando o coeficiente de variação entre eles, onde todos os coeficientes ficaram abaixo de 0,2%, sendo então, considerados aceitáveis os resultados obtidos pelo simulador.

Todo o controle de versão é feito a partir do git e está disponível de forma livre no repositório do GitHub<sup>9</sup>, sendo possível acompanhar todo o desenvolvimento do simulador e contribuir com a comunidade de software livre.

---

<sup>9</sup> <<https://github.com/gabriel-augusto/AcousticSimulator>>



## 6 Conclusão

Este trabalho teve como objetivo o desenvolvimento do simulador acústico que, por sua vez, é uma ferramenta multiparadigma baseada nos paradigmas multiagentes e orientado a objetos. Esta ferramenta consiste em simular o comportamento do som em um ambiente fechado e prover o tempo de reverberação (RT60) ao final de cada simulação. Dessa forma, é possível que o usuário informe parâmetros como dimensões do ambiente, obstáculos e suas posições, coeficientes de absorção sonora de cada obstáculo e, por fim, fontes sonoras e suas devidas configurações, tais como, posição, ângulo de abertura e direção.

A partir da configuração desses itens, o usuário é capaz de executar e acompanhar uma simulação, podendo pausar, retomar ou encerrar a qualquer momento se assim desejar. Ao fim, o sistema informa o tempo de reverberação do ambiente com base nos parâmetros informados anteriormente pelo usuário. Neste trabalho, foi apresentado como o simulador acústico pode ser utilizado, desde a sua inicialização, inserções de obstáculos e fontes sonoras, até a realização da simulação de um ambiente acústico e a visualização do tempo de reverberação.

Para atingir o objetivo geral deste trabalho, foram definidos alguns objetivos específicos. A princípio, foi realizado um estudo a respeito dos conhecimentos de acústica, onde foi dada uma ênfase maior em ambientes acústicos e como se dava o comportamento do som dentro desses ambientes, a fim de identificar as variáveis a serem consideradas dentro do sistema, com o intuito de alcançar o primeiro objetivo específico deste trabalho, ou seja, estudar o comportamento do som dentro de ambientes acústicos, identificando variáveis acústicas presentes dentro desses ambientes.

Juntamente com o estudo sobre acústica, foi realizado também um estudo a respeito dos coeficientes de absorção dos materiais presentes no ambiente e como o som interagia com os mesmos, a fim de atender ao segundo objetivo específico deste trabalho. Desta forma, foi possível propor um suporte tecnológico adequado para a implementação do simulador, o qual foi o terceiro objetivo específico.

Durante a implementação da solução, foram exploradas técnicas de programação, bem como padrões de projeto e as demais boas práticas da engenharia de software, visando o desenvolvimento de um simulador manutenível e extensível. Para garantir a qualidade do código, foram realizados testes automatizados juntamente com a análise estática do código fonte, onde foi possível observar o nível de cobertura do código fonte e algumas métricas importantes referente ao código desenvolvido. Também houve uma preocupação em realizar atividades de refatoração de forma sistemática durante o desenvolvimento do

simulador. Todas essas atividades contribuíram para que os objetivos específicos 4 e 5 deste trabalho fossem atendidos.

Neste trabalho, foi proposta a seguinte questão de pesquisa: "*É possível desenvolver um sistema que simule o comportamento do som dentro de um ambiente fechado utilizando uma abordagem multiagentes?*". Com base nos resultados desse trabalho, oriundos dos objetivos específicos, pode-se concluir que a resposta à questão é: Sim, o simulador atendeu ao objetivo geral para o qual foi proposto.

A expectativa do autor é que o simulador possa ser evoluído e utilizado ou incorporado em novos simuladores e/ou suportes. O intuito é auxiliar projetistas e/ou especialistas em acústica no que tange o acompanhamento e a avaliação dos parâmetros de seus projetos.

## 6.1 Trabalhos futuros

O simulador desenvolvido neste trabalho de conclusão de curso aborda uma máquina de raciocínio capaz de representar o comportamento do som dentro de um ambiente. Com o intuito de permitir que este simulador fosse utilizado a nível de usuário, foi desenvolvida uma interface gráfica, possibilitando configurar um ambiente, obstáculos e fontes sonoras e então, realizar uma simulação. No entanto, a biblioteca de gráficos JFreeChart, limitou bastante a visualização da simulação, pois não era possível representar os obstáculos e os sons simultaneamente.

Um dos pontos de melhoria, seria investigar uma alternativa ao JFreeChart que possibilite a representação de todos os elementos dentro do ambiente. É preferível que essa nova alternativa possua recursos para a representação do ambiente em três dimensões, uma vez que o simulador já começa a apresentar recursos tridimensionais, os quais foram implementados e não podem ser visualizados adequadamente via JFreeChart..

A informação dos índices de absorção de cada material para cada faixa de frequência ocorre de forma manual no sistema atualmente. Uma potencial evolução no simulador seria investigar materiais comumente utilizados em obras que envolvem tratamento acústico e identificar seus índices de absorção para que possam então ser cadastrados no sistema, permitindo ao usuário escolher apenas o material que se aplica a cada obstáculo como uma alternativa à inserção do coeficiente de absorção de forma manual.

Por fim, outro ponto de extensibilidade do simulador é a implantação de novos parâmetros a serem coletados durante a simulação e apresentados ao usuário final. Existem diversos parâmetros adicionais que podem ser obtidos por meio da simulação como por exemplo: clareza, brilho, equilíbrio, ruído e distorção (FIGUEIREDO, 2005).

A arquitetura do simulador foi elaborada de modo a garantir uma ferramenta

manutenível e extensível. O código fonte está disponível para a comunidade de software livre no GitHub<sup>1</sup>, um site para compartilhamento de projetos que utilizam a ferramenta de controle de versionamento Git.

---

<sup>1</sup> <<https://github.com/gabriel-augusto/AcousticSimulator>>



## Referências

- ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. *NBR 12179: Tratamento acústico em recintos fechados — referências*. Rio de Janeiro, 1992. Citado na página 17.
- BASTOS, L. P.; CARDOSO, H. F. S.; MELO, G. d. S. V. de. Análise numérica e experimental dos parâmetros acústicos de uma sala de aula da universidade federal do pará. Campina Grande - Paraíba, 2010. Citado na página 37.
- BISTAFÁ, S. *Acústica aplicada ao controle do ruído*. [S.l.]: Edgard Blücher, 2006. ISBN 9788521203766. Citado 4 vezes nas páginas 31, 32, 33 e 35.
- CAMILO, T. S.; TENENBAUM, R. A.; COELHO, J. L. B. Engenharia acústica auxiliada por computador: um método híbrido para simulação de acústica de salas. *Anais do SEMEA*, 2002. Citado na página 37.
- CARVALHO, D. D. T. d. Metodologia de análise do desempenho da usina de concentração da samarco mineração sa baseada em simulação das operações. Programa de Pós-Graduação em Engenharia Mineral. Departamento de Engenharia de Minas, Escola de Minas, Universidade Federal de Ouro Preto., 2003. Citado na página 36.
- FIGUEIREDO, F. L. *Parâmetros acústicos subjetivos: Critérios para avaliação da qualidade acústica de salas de música*. Tese (Doutorado), 2005. Citado na página 80.
- FILHO, C. M. de O. *Kalibro: interpretação de métricas de código-fonte*. Tese (Doutorado) — Universidade de São Paulo, 2013. Citado 3 vezes nas páginas 73, 74 e 77.
- FRANÇA, R. M. et al. Análise comparativa entre as plataformas multiagentes jade e fipa-os. Acedido em Abril de 2012. Citado na página 59.
- GIL, A. C. *Como elaborar projetos de pesquisa*. São Paulo, Brasil: Atlas, 2002. Citado na página 45.
- LAW, A. M.; KELTON, W. D. *Simulation modeling and analysis*. [S.l.]: McGraw-Hill New York, 1991. Citado na página 36.
- MEIRELLES, P. R. M. *Monitoramento de métricas de código-fonte em projetos de software livre*. Tese (Doutorado) — Universidade de São Paulo, 2013. Citado na página 73.
- NETO, A. L. Análise do som transmitido por madeiras de diferentes densidades. Universidade Federal de Lavras, 2007. Citado na página 63.
- OFUGI, F. M. A.; A, C. J. E.; A, G. M. H. Simulação acústica em ambientes fechados utilizando o método do traçado de raios e fontes virtuais. Faculdade do Gama – Universidade de Brasília, 2013. Citado 2 vezes nas páginas 41 e 42.
- RAMIRES, F. Coeficiente de espalhamento sonoro de painéis perfurados. dissertação (mestrado). Faculdade de Engenharia Civil, Arquitetura e Urbanismo, Universidade Estadual de Campinas, São Paulo, 2011. Disponível em: <<http://www.bibliotecadigital.unicamp.br/document/?code=000811191&fd=y>>. Citado 2 vezes nas páginas 27 e 28.

REZENDE, J. L. *Aplicando técnicas de comunicação para a facilitação de debates no ambiente AulaNet*. Tese (Doutorado) — Dissertação de Mestrado, PUC-Rio, 2003. Citado na página 56.

ROCHA, T. L. Viabilidade da utilização da pesquisa-ação em situações de ensino-aprendizagem. *Cadernos da FUCAMP*, v. 11, n. 14, 2012. Citado na página 45.

SANTOS, C. dos. Influência do espalhamento acústico na percepção auditiva de espaços: métodos e desenvolvimentos. Campinas, SP, 2011. Citado 3 vezes nas páginas 27, 28 e 29.

SCHRIBER, T. J. *Simulation using GPSS*. [S.l.], 1974. Citado na página 36.

SCHWABER, K.; BEEDLE, M. gilé software development with scrum. 2002. Citado na página 47.

SILVA, L. d. M. Estudo e desenvolvimento de sistemas multiagentes usando jade: Java agent development framework. *Trabalho de Conclusão de Curso (Graduação em Ciências da Computação)–Centro Tecnológico. Universidade Federal de Pernambuco, Pernambuco*, 2003. Citado 2 vezes nas páginas 15 e 66.

SILVA, P. *Acústica arquitetônica*. Belo Horizonte, MG: Edições Engenharia e Arquitetura, 1971. (Publicação (Universidade Federal de Minas Gerais)). Citado 9 vezes nas páginas 15, 27, 31, 32, 33, 34, 35, 36 e 60.

SILVA, R.; TAFNER, E. P. Apostila de metodologia científica. Associação Educacional do Vale do Itajaí-Mirim, 2006. Disponível em: <<http://www.teses.usp.br/teses/disponiveis/18/18149/tde-2592012-204752/>>. Citado na página 45.

THIOLLENT, M. Metodologia da pesquisa-ação. In: *Metodologia da pesquisa-ação*. [S.l.]: Cortez, 1986. Citado na página 45.

THOMAZ, L. F. *Um arcabouço para construção de sistemas multiagentes musicais*. Tese (Doutorado) — Universidade de São Paulo, 2011. Disponível em: <<http://www.teses.usp.br/teses/disponiveis/45/45134/tde-13082011-113109/>>. Citado na página 38.

TONI, J. Cenários e análise estratégica: questões metodológicas. *Rev. Espaço*, 2006. Citado na página 56.

TORGA, B. L. M. *Modelagem, simulação e otimização em sistemas puxados de manufatura*. Tese (Doutorado) — UNIVERSIDADE FEDERAL DE ITAJUBÁ, 2007. Citado na página 36.

TORRES, M. H. C. Simulação acústica no ambiente acmus. dissertação (mestrado em ciência da computação). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2008. Disponível em: <<http://www.teses.usp.br/teses/disponiveis/45/45134/tde-09092009-103134/>>. Citado 4 vezes nas páginas 27, 28, 36 e 37.

WEISS, G. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, 1999. (Intelligent Robotics and Autonomous Agents Series). ISBN 9780262731317. Disponível em: <<http://books.google.com.br/books?id=JYcznFCN3xcC>>. Citado 2 vezes nas páginas 37 e 38.

WOOLDRIDGE, M. *An Introduction to MultiAgent Systems*. Wiley, 2009. ISBN 9780470519462. Disponível em: <<http://books.google.com.br/books?id=X3ZQ7yeDn2IC>>. Citado na página 37.



## Apêndices



# APÊNDICE A – Código fonte da classe Ambient

```

1 package simulator.agents;
2
3 import jade.core.AID;
4 import jade.core.Agent;
5 import jade.core.behaviours.CyclicBehaviour;
6 import jade.lang.acl.ACLMessage;
7 import jade.wrapper.ContainerController;
8 import languagesAndMessages.Message;
9 import presentation.HomeFrame;
10 import simulator.objects.AmbientObject;
11 import simulator.objects.UIController;
12
13 public class Ambient extends Agent{
14
15     private static final long serialVersionUID = 3849717343310509053L;
16
17     private AmbientObject ambient = AmbientObject.getInstance();
18
19     private int countSoundSourcesFinished = 0;
20
21     @Override
22     protected void setup() {
23         defineAmbient();
24         addBehaviour(new GetMessageBehaviour(this));
25         addBehaviour(new GetEventBehaviour(this));
26         ambient.createAgent(null, ambient.getContainer(), ambient.GUI);
27     }
28
29     public void defineAmbient() {
30         ambient.setAmbientAID(this.getAID());
31         ambient.setContainer(getContainerController());
32         ambient.setCc((ContainerController) ambient.getContainer());
33     }
34
35     /*----- COMPORTAMENTOS -----*/
36     private class GetMessageBehaviour extends CyclicBehaviour {
37
38         private static final long serialVersionUID = 1L;
39
40         public GetMessageBehaviour(Agent agent) {

```



```

87         break;
88     case Message.PAUSE:
89         pauseSimulation();
90         break;
91     case Message.RESUME:
92         resumeSimulation();
93         break;
94     case Message.RUN:
95         runSimulation();
96         break;
97     case Message.CREATE_SOUND_SOURCE:
98         ambient.defineSoundSource();
99         break;
100    default:
101        break;
102    }
103  }
104 }
105 private void runSimulation(){
106     send(Message.prepareMessage(ACLMessage.INFORM, null, Message.RUN,
107         ambient.getSoundSources()));
108 }
109 private void stopSimulation(String status) {
110     send(Message.prepareMessage(ACLMessage.INFORM, null, status, ambient.
111         getSoundSources()));
112 }
113 private void pauseSimulation(){
114     send(Message.prepareMessage(ACLMessage.INFORM, null, Message.PAUSE,
115         ambient.getSoundSources()));
116 }
117 private void resumeSimulation() {
118     send(Message.prepareMessage(ACLMessage.INFORM, null, Message.RESUME,
119         ambient.getSoundSources()));
120 }
121 }

```

src/simulator/agents/Ambient.java



# APÊNDICE B – Código fonte da classe SoundSource

```

1 package simulator.agents;
2
3 import simulator.objects.SoundSourceObject;
4 import jade.core.AID;
5 import jade.core.Agent;
6 import jade.core.behaviours.CyclicBehaviour;
7 import jade.domain.DFService;
8 import jade.domain.FIPAException;
9 import jade.domain.FIPAAgentManagement.DFAgentDescription;
10 import jade.lang.acl.ACLMessage;
11 import languagesAndMessages.Language;
12 import languagesAndMessages.Message;
13
14 public class SoundSource extends Agent{
15
16     private static final long serialVersionUID = 3414148236747846279L;
17
18     private SoundSourceObject soundSource;
19
20     @Override
21     protected void setup() {
22         getParameters();
23         registerSoundSource();
24         soundSource.setSelfAID(getAID());
25         soundSource.setContainer(getContainerController());
26         soundSource.setCc(getContainerController());
27         addBehaviours();
28     }
29
30     private void addBehaviours() {
31         addBehaviour(new GetMessageBehaviour(this));
32     }
33
34     private void registerSoundSource(){
35         try {
36             DFAgentDescription dfd = new DFAgentDescription();
37             dfd.setName(getAID());
38             DFService.register(this, dfd);
39         } catch (FIPAException e) {
40         }
41     }

```

```

41    }
42
43    private void getParameters() {
44        Object [] args = getArguments();
45        soundSource = ( SoundSourceObject ) args [0];
46    }
47
48    /*----- COMPORTAMENTS -----*/
49
50    private class GetMessageBehaviour extends CyclicBehaviour {
51
52        private static final long serialVersionUID = 1L;
53
54        public GetMessageBehaviour(Agent agent) {
55            super(agent);
56        }
57
58        @Override
59        public void action() {
60            ACLMessage message = receive();
61
62            if ( message != null && message.getPerformative() == ACLMessage.INFORM
63                ) {
64                AID sender = message.getSender();
65
66                if ( message.getContent() . equals( Message.STOP_RESUMED ) ){
67                    send( soundSource.stopSimulation( Message.STOP_RESUMED ) );
68                    System.out.println( "Simulation stoped." );
69                }
70                else if ( message.getContent() . equals( Message.STOP_PAUSED ) ){
71                    send( soundSource.stopSimulation( Message.STOP_PAUSED ) );
72                    System.out.println( "Simulation stoped." );
73                }
74                else if( message.getContent() . equals( Message.PAUSE ) ){
75                    send( soundSource.suspendAllSounds() );
76                }
77                else if( message.getContent() . equals( Message.RESUME ) ){
78                    soundSource.resumeAllSounds();
79                }
80                else if( message.getContent() . equals( Message.RUN ) ){
81                    soundSource.emitSoundPulse();
82                }
83                else if( message.getLanguage() . equals( Language.FINISH ) ){
84                    soundSource.getSounds() . remove( message.getContent() );
85                    if ( soundSource.getSounds() . isEmpty() ){
send( Message.prepareMessage(ACLMessage.INFORM, null, Message.
FINISH_SIMULATION, soundSource.getAmbient() ) );

```

```
86         }
87     }
88     else {
89         send( Message .getAnswerOfANotUnderstoodMessage( sender ) );
90     }
91 }
92 }
93 }
94 }
95
96 private static int id = 0;
97
98 public static String nextId() {
99     return "sound_source" + (++id);
100 }
101 }
```

src/simulator/agents/SoundSource.java



# APÊNDICE C – Código fonte da classe Sound

```
1 package simulator.agents;
2
3 import settings.ProjectSettings;
4 import simulator.objects.SoundObject;
5 import jade.core.Agent;
6 import jade.core.behaviours.CyclicBehaviour;
7 import jade.core.behaviours.TickerBehaviour;
8 import jade.domain.DFService;
9 import jade.domain.FIPAException;
10 import jade.domain.FIPAAgentManagement.DFAgentDescription;
11 import jade.lang.acl.ACLMessage;
12 import languagesAndMessages.Language;
13 import languagesAndMessages.Message;
14
15
16 public class Sound extends Agent{
17
18     private static final long serialVersionUID = 3789435534023352353L;
19     private SoundObject sound;
20
21     @Override
22     protected void setup() {
23         getParameters();
24         soundRegister();
25         sound.findNextObstacle();
26         addBehavior();
27     }
28
29     private void addBehavior() {
30         addBehaviour(new UpdateSoundBehavior(this, ProjectSettings.
31             getProjectSettings().getSimulationSpeed()));
32         addBehaviour(new GetMessageBehaviour(this));
33     }
34
35     private void soundRegister(){
36         try{
37             DFAgentDescription sound = new DFAgentDescription();
38             sound.setName(getAID());
39             DFService.register(this, sound);
40         } catch (FIPAException e) {
```

```
40         e.printStackTrace();
41     }
42 }
43
44 private void getParameters() {
45     sound = (SoundObject) getArguments()[0];
46 }
47
48 private void killSound() {
49     send(Message.prepareMessage(ACLMessage.INFORM, Language.FINISH, sound.
50         getIdentifier(), sound.getSoundSource()));
51     SoundObject.getSounds().remove(sound.getIdentifier());
52     this.doDelete();
53     System.out.println("END OF SOUND!!!");
54 }
55
56 private String getActualState(){
57     return this.getAID().getName() + ":" + sound.getState();
58 }
59
60 private static int id = 0;
61
62 public static String nextId() {
63     return "Sound_" + (++id);
64 }
65 /*----- COMPORTAMENTS ----- */
66
67 private class UpdateSoundBehavior extends TickerBehaviour {
68
69     private static final long serialVersionUID = 5631501784835798992L;
70
71     public UpdateSoundBehavior(Agent a, long period) {
72         super(a, period);
73     }
74
75     @Override
76     protected void onTick() {
77         try{
78             sound.update();
79         }catch(NullPointerException e){
80             killSound();
81         }
82         System.out.println("\n"+getActualState());
83         if(sound.getDecibel() < 0){
84             killSound();
85         }
86     }
87 }
```

```
86     }
87
88 }
89
90 private class GetMessageBehaviour extends CyclicBehaviour {
91
92     private static final long serialVersionUID = 1L;
93
94     public GetMessageBehaviour(Agent agent) {
95         super(agent);
96     }
97
98     @Override
99     public void action() {
100        ACLMessage message = receive();
101
102        if (message != null && message.getPerformative() == ACLMessage.INFORM)
103            {
104                if (message.getContent().equals(Message.STOP_RESUMED))
105                    killSound();
106                else if (message.getContent().equals(Message.PAUSE))
107                    doSuspend();
108            }
109        }
110    }
111 }
112 }
```

src/simulator/agents/Sound.java



# APÊNDICE D – Código fonte da classe Obstacle

```

1 package simulator.objects;
2
3 import java.util.HashMap;
4
5
6 public class Obstacle {
7
8     private Line line;
9     private double absorptionRate;
10    private static HashMap <String , Obstacle> obstacles = new HashMap<>();
11
12    private Obstacle(Line line , double absorbtionRate){
13        this.setLine(line);
14        this.absorptionRate = absorbtionRate;
15    }
16
17    public static Obstacle createObstacle(String id , Line line , double
18        absorptionRate){
19        Obstacle obstacle = new Obstacle(line , absorptionRate);
20        getObstacles().put(id , obstacle);
21        return obstacle;
22    }
23
24    public double getAbsortionRate() {
25        return absorptionRate;
26    }
27
28    public void setAbsortionRate(double absorbtionRate) {
29        this.absorptionRate = absorbtionRate;
30    }
31
32    public Line getLine() {
33        return line;
34    }
35
36    public void setLine(Line line) {
37        this.line = line;
38    }
39
40    public double getSize(){

```

```
40     return this.getLine().getInitialPoint().distance(this.getLine() .
41             getFinalPoint());
42
43     public static HashMap <String , Obstacle> getObstacles() {
44         return obstacles;
45     }
46 }
```

src/simulator/objects/Obstacle.java

# APÊNDICE E – Código fonte da classe Line

```

1 package simulator.objects;
2
3
4 public abstract class Line {
5
6     private Location initialPoint = null;
7     private Location finalPoint = null;
8     private double directionAngle = 0;
9     private int type; // 0 para normal e 1 para vertical
10    public static final int VERTICAL = 1;
11    public static final int NORMAL = 0;
12
13    //Constructors methods
14
15    protected Line(Location initialPoint, Location finalPoint, int tipo){
16        this.initialPoint = initialPoint;
17        this.finalPoint = finalPoint;
18        this.type = tipo;
19    }
20
21    protected Line(Location initialPoint, double direction, int tipo){
22        this.initialPoint = initialPoint;
23        this.directionAngle = direction;
24        this.type = tipo;
25    }
26
27    //Static methods
28
29    public static Line getLine(Location initialPoint, Location finalPoint){
30        if(isVertical(initialPoint, finalPoint)){
31            return new VerticalLine(initialPoint, finalPoint);
32        }
33        return new NormalLine(initialPoint, finalPoint);
34    }
35
36    public static Line getLine(Location initialPoint, double direction){
37        if(isVertical(direction)){
38            return new VerticalLine(initialPoint, direction);
39        }
40        return new NormalLine(initialPoint, direction);
41    }
42

```

```
43     private static boolean isVertical(Location initialPoint, Location
44         finalPoint){
45         if(initialPoint.getX() == finalPoint.getX())
46             return true;
47         return false;
48     }
49
50     public static boolean isVertical(double direction){
51         if(direction == 90 || direction == 270)
52             return true;
53         return false;
54     }
55
56     //Abstract methods
57
58     protected abstract double getConstant();
59     public abstract double getSlope();
60     public abstract Location searchIntersectionPoint(NormalLine line);
61     public abstract Location searchIntersectionPoint(VerticalLine line);
62
63     //Public methods
64
65     public Location searchIntersectionPoint(Line line){
66         Location intersectionPoint = null;
67         switch(line.getType()){
68             case 0:
69                 intersectionPoint = searchIntersectionPoint((NormalLine)line);
70                 break;
71             case 1:
72                 intersectionPoint = searchIntersectionPoint((VerticalLine)line);
73                 break;
74         }
75         return intersectionPoint;
76     }
77
78     //Gets and Sets
79
80     public Location getInitialPoint() {
81         return initialPoint;
82     }
83
84     public void setInitialPoint(Location initialPoint) {
85         this.initialPoint = initialPoint;
86     }
87
88     public Location getFinalPoint() {
89         return finalPoint;
```

```
89  }
90
91 public void setFinalPoint(Location finalPoint) {
92     this.finalPoint = finalPoint;
93 }
94
95 public double getDirection() {
96     return directionAngle;
97 }
98
99 public void setDirection(double direction) {
100    this.directionAngle = direction;
101 }
102
103 public int getType() {
104     return type;
105 }
106
107 public void setType(int vertical) {
108     this.type = vertical;
109 }
110 }
```

src/simulator/objects/Line.java



# APÊNDICE F – Código fonte da classe NormalLine

```
1 package simulator.objects;
2
3 public class NormalLine extends Line{
4
5     protected NormalLine(Location initialPoint, Location finalPoint) {
6         super(initialPoint, finalPoint, Line.NORMAL);
7     }
8
9     protected NormalLine(Location initialPoint, double direction) {
10        super(initialPoint, direction, Line.NORMAL);
11    }
12
13    public Location searchIntersectionPoint(NormalLine line) {
14        Location intersectionPoint;
15        double x;
16        double y;
17
18        if((this.getSlope() - line.getSlope()) != 0){
19            x = (line.getConstant() - this.getConstant())/(this.getSlope() - line
19                .getSlope());
20            y = line.getSlope() * x + line.getConstant();
21            intersectionPoint = new Location(x,y);
22            if(intersectionPoint != null && this.belongsToTheInterval(
22                intersectionPoint.getX()) && line.belongsToTheInterval(
22                intersectionPoint.getX()))
23                return intersectionPoint;
24        }
25        return null;
26    }
27
28    public Location searchIntersectionPoint(VerticalLine line) {
29        double y;
30        y = this.getY(line.getConstant());
31
32        if(line.belongsToTheInterval(y) && this.belongsToTheInterval(line
32            .getConstant()))
33            return new Location(line.getConstant(), y);
34        return null;
35    }
36
```

```

57     protected boolean belongsToTheInterval(double x){
58         if(this.getFinalPoint() != null){
59             if(x <= Math.max(this.getInitialPoint().getX(), this.getFinalPoint().
60                 getX()) && x >= Math.min(this.getInitialPoint().getX(), this.
61                 getFinalPoint().getX())){
62                 return true;
63             }
64         } else{
65             if(this.getDirection() > 90 && this.getDirection() < 270 && x <= this
66                 .getInitialPoint().getX()){
67                 return true;
68             } else{
69                 if(x >= this.getInitialPoint().getX()){
70                     return true;
71                 }
72             }
73         }
74         return false;
75     }
76
77     public double getSlope() {
78         if(this.getFinalPoint() != null){
79             double deltaX = this.getFinalPoint().getX() - this.getInitialPoint().
80                 getX();
81             double deltaY = this.getFinalPoint().getY() - this.getInitialPoint().
82                 getY();
83             return deltaY/deltaX;
84         }
85         return Math.tan(
86             Math.toRadians(this.getDirection()));
87     }
88
89     protected double getConstant() {
90         return this.getInitialPoint().getY() - this.getSlope() * this.
91             getInitialPoint().getX();
92     }
93
94     protected Double getY(double x){
95         return this.getSlope() * x + this.getConstant();
96     }
97 }
```

src/simulator/objects/NormalLine.java

# APÊNDICE G – Código fonte da classe VerticalLine

```

1 package simulator.objects;
2
3 import utils.Util;
4
5 public class VerticalLine extends Line{
6
7     private static final double TOLERANCE = 0.000001;
8
9     protected VerticalLine(Location initialPoint, Location finalPoint) {
10         super(initialPoint, finalPoint, Line.VERTICAL);
11     }
12
13     protected VerticalLine(Location initialPoint, double direction){
14         super(initialPoint, direction, Line.VERTICAL);
15     }
16
17     public Location searchIntersectionPoint(VerticalLine line){
18         return null;
19     }
20
21     public Location searchIntersectionPoint(NormalLine line){
22         double y;
23         y = line.getY(this.getConstant());
24
25         if(this.belongsToTheInterval(y) && line.belongsToTheInterval(this.
26             getConstant()))
27             return new Location(this.getConstant(), y);
28         return null;
29     }
30
31     protected boolean belongsToTheInterval(double y){
32         if(this.getFinalPoint() != null){
33             if(y <= Math.max(this.getInitialPoint().getY(), this.getFinalPoint().
34                 getY()) && y >= Math.min(this.getInitialPoint().getY(), this.
35                 getFinalPoint().getY())){
36                 return true;
37             }
38         }else{
39             if(Util.compareDouble(this.getDirection(), 270, TOLERANCE) && y <=
40                 this.getInitialPoint().getY())
41             return true;
42         }
43     }
44 }
```

```
37     return true;
38
39     if( Util.compareDouble( this .getDirection() , 90 , TOLERANCE) && y >=
40         this .getInitialPoint().getY())
41     return true;
42
43     return false;
44 }
45
46 public double getSlope() {
47     return Double.POSITIVE_INFINITY;
48 }
49
50 public double getConstant() {
51     return super.getInitialPoint().getX();
52 }
53
54 }
```

src/simulator/objects/VerticalLine.java

# APÊNDICE H – Código fonte da classe Location

```

1 package simulator.objects;
2
3 public class Location {
4
5     private double x;
6     private double y;
7
8     public Location(double x, double y) {
9         this.x = x;
10        this.y = y;
11    }
12
13    public Location(Location l) {
14        this(l.x, l.y);
15    }
16
17    public Location() {
18        this(-1,-1);
19    }
20
21    public double getX() {
22        return x;
23    }
24
25    public void setX(double x) {
26        this.x = x;
27    }
28
29    public double getY() {
30        return y;
31    }
32
33    public void setY(double y) {
34        this.y = y;
35    }
36
37    public double distance(Location location){
38        double dx = Math.abs(this.getX() - location.getX());
39        double dy = Math.abs(this.getY() - location.getY());
40        return Math.sqrt((dx*dx) + (dy*dy));

```

```

41    }
42
43    @Override
44    public String toString() {
45        return "(x: "+x+"; y: "+y+ ")";
46    }
47
48    public boolean equals(Location location, double precision){
49        if(this.distance(location) < precision)
50            return true;
51        return false;
52    }
53
54    public static Location valueOf(String loc){
55        StringBuilder x = new StringBuilder();
56        StringBuilder y = new StringBuilder();
57        char coordinate = 'x';
58        boolean foundANumber = false;
59
60        for(int i = 0; i < loc.length(); i++){
61            if((Character.isDigit(loc.charAt(i)) || loc.charAt(i) == '.' || loc.
62                charAt(i) == '-') && coordinate=='x'){
63                foundANumber = true;
64                x.append(loc.charAt(i));
65            }
66
67            if((Character.isDigit(loc.charAt(i)) || loc.charAt(i) == '.' || loc.
68                charAt(i) == '-') && coordinate=='y'){
69                y.append(loc.charAt(i));
70            }
71
72            if(Character.isLetter(loc.charAt(i)) && foundANumber){
73                coordinate = 'y';
74            }
75        }
76        return new Location(Double.valueOf(x.toString()), Double.valueOf(y.
77            toString()));
78    }
79
80}

```

src/simulator/objects/Location.java

# APÊNDICE I – Código fonte da classe SoundObject

```

1 package simulator.objects;
2
3 import java.util.HashMap;
4
5 import utils.Util;
6 import jade.core.AID;
7 import settings.ProjectSettings;
8
9 public class SoundObject {
10
11     private double sizeOfStep;
12     private double error;
13
14     private String identifier;
15     private String state;
16     private AID soundSource;
17
18     private Line rote;
19     private Location collisionPoint;
20     private Obstacle collisionObstacle;
21
22     private Location initialLocation;
23     private Location actualLocation;
24     private double direction;
25     private double power;
26     private double actualVirtualSoundSourcePower;
27     private int opening;
28     private double intensity;
29     private double distanceOfPreviousColisionPoint = 0;
30     private double distanceTraveled = 0;
31
32     private static HashMap <String , SoundObject> sounds = new HashMap<>();
33
34     public static HashMap <String , SoundObject> getSounds() {
35         return sounds;
36     }
37
38     public static SoundObject createSound(Location location , double direction
39             , double power , int opening , AID ambient , AID soundSource , String id){
ProjectSettings settings = ProjectSettings.getProjectSettings();

```

```

40     SoundObject sound = new SoundObject(location, direction, power, opening
41         , ambient, soundSource, id);
42     sound.setSizeOfStep((settings.getWidth() + settings.getLength()) / 50);
43     sound.setError(sound.getSizeOfStep());
44     getSounds().put(id, sound);
45     return sound;
46 }
47
48 private SoundObject(Location location, double direction, double power,
49     int opening, AID ambient, AID soundSource, String id){
50     this.initialLocation = new Location(location);
51     this.actualLocation = new Location(location);
52     System.out.println("\nInitial Location: " + initialLocation + "\nActual
53         Location: " + actualLocation);
54     this.direction = direction;
55     this.power = power;
56     this.setActualVirtualSoundSourcePower(power);
57     this.intensity = power;
58     this.opening = opening;
59     this.soundSource = soundSource;
60     this.identifier = id;
61     this.rote = Line.getLine(this.getInitialLocation(), direction);
62 }
63
64 public void findNextObstacle(){
65     this.setCollisionPoint(null);
66     this.setCollisionObstacle(null);
67     Location intersectionPoint = null;
68     for(Obstacle obstacle : Obstacle.getObstacles().values()){
69         intersectionPoint = this.getRote().searchIntersectionPoint(obstacle.
70             getLine());
71         if(intersectionPoint != null && !this.getActualLocation().equals(
72             intersectionPoint, getError())){
73             if(this.getCollisionPoint() == null || this.getActualLocation().
74                 distance(intersectionPoint) < this.getActualLocation().distance(
75                 this.getCollisionPoint())){
76                 this.setCollisionObstacle(obstacle);
77                 this.setCollisionPoint(intersectionPoint);
78                 System.out.println("Obstacle found in "+this.getActualLocation().
79                     distance(this.getCollisionPoint())+" meters:: \nindex: " +
80                     obstacle.getAbsortionRate() + "\ncollision point: " + this.
81                     getCollisionPoint().toString());
82             }
83         }
84     }
85 }
86

```

```
77 public void update(){
78     this.setDistanceOfPreviousColisionPoint(this.
79         getDistanceOfPreviousColisionPoint() + getSizeOfStep());
80     this.setDistanceTraveled(this.getDistanceTraveled() + getSizeOfStep());
81     this.setIntensity(calculateIntensityBySoundPropagation(this.
82         getActualVirtualSoundSourcePower(), this.getOpening(), this.
83         getDistanceTraveled()));
84     updateLocation();
85
86     if(isCollisionPoint()){
87         updateParameters();
88         System.out.println("\nCOLLIDED!!!!!");
89         findNextObstacle();
90     }
91     this.setState(this.getActualState());
92 }
93
94 public void updateLocation() {
95     this.getActualLocation().setX(Util.calculateX(this.getDirection(), this.
96         .getDistanceOfPreviousColisionPoint()) + this.getInitialLocation().
97         getX());
98     this.getActualLocation().setY(Util.calculateY(this.getDirection(), this.
99         .getDistanceOfPreviousColisionPoint()) + this.getInitialLocation().
100        getY());
101 }
102
103 public double calculateIntensityBySoundPropagation(double power, double
104     angle, double radius){
105     return (power / (4 * Math.PI * radius * radius)) * (360 / angle);
106 }
107
108 public boolean isCollisionPoint() {
109     return this.getActualLocation().equals(this.getCollisionPoint(),
110         getError());
111 }
112
113 public void updateParameters(){
114     this.setDistanceOfPreviousColisionPoint(0);
115     this.setInitialLocation(this.getCollisionPoint());
116     this.setDirection(calculateNewDirection());
117     this.setIntensity(calculateIntencityAfterColisionPoint(this.
118         getCollisionObstacle().getAbsortionRate()));
119     this.setActualVirtualSoundSourcePower(this.
120         calculateNewVirtualSoundSourcePower(this.getIntensity(), this.
121             getDistanceTraveled(), this.getOpening()));
122     this.setRote(Line.getLine(this.getInitialLocation(), this.getDirection(
123         )));
124 }
```

```
111    }
112
113    public double calculateNewVirtualSoundSourcePower(double intensity ,
114        double distance , double angle){
115        return (intensity * angle * 4 * Math.PI * distance * distance)/360;
116    }
117
118    public double calculateIntensityAfterColisionPoint(double absorptionRate)
119    {
120        return this.getIntensity() - (this.getIntensity()*(absorptionRate/100))
121        ;
122    }
123
124    public double calculateNewDirection() {
125        double obstacleSlopeAngle = Math.toDegrees(Math.atan(this.
126            getCollisionObstacle().getLine().getSlope()));
127        double newDirection = 2 * obstacleSlopeAngle - this.getDirection();
128        return Util.normalizeAngle(newDirection);
129    }
130
131
132    public String getActualState(){
133        return "\ndecibel: " + this.getDecibel()
134        + "\ndirection: " + this.getDirection() + " degrees" + "\ndistance
135        of origin: "
136        + this.getDistanceOfPreviousColisionPoint() + "\nnitial location:
137        "
138        + this.getInitialLocation() + "\nlocation: " + this.
139            getActualLocation();
140    }
141
142    /* ----- Gets and Setters ----- */
143
144    public String getIdentifier() {
145        return identifier;
146    }
147
148    public void setIdentifier(String identifier) {
149        this.identifier = identifier;
150    }
151
152    public AID getSoundSource() {
153        return soundSource;
154    }
155
156    public void setSoundSource(AID soundSource) {
157        this.soundSource = soundSource;
158    }
159
160    public Line getRote() {
161        return rote;
```

```
151     }
152
153     public void setRote(Line rote) {
154         this.rote = rote;
155     }
156
157     public Location getCollisionPoint() {
158         return collisionPoint;
159     }
160
161     public void setCollisionPoint(Location collisionPoint) {
162         this.collisionPoint = collisionPoint;
163     }
164
165     public Obstacle getCollisionObstacle() {
166         return collisionObstacle;
167     }
168
169     public void setCollisionObstacle(Obstacle collisionObstacle) {
170         this.collisionObstacle = collisionObstacle;
171     }
172
173     public Location getInitialLocation() {
174         return initialLocation;
175     }
176
177     public void setInitialLocation(Location initialLocation) {
178         this.initialLocation = new Location(initialLocation);
179     }
180
181     public Location getActualLocation() {
182         return actualLocation;
183     }
184
185     public void setActualLocation(Location actualLocation) {
186         this.actualLocation = new Location(actualLocation);
187     }
188
189     public double getDirection() {
190         return direction;
191     }
192
193     public void setDirection(double direction) {
194         this.direction = direction;
195     }
196
197     public double getPower() {
```

```
198     return power;
199 }
200
201 public void setPower(double power) {
202     this.power = power;
203 }
204
205 public int getOpening() {
206     return opening;
207 }
208
209 public void setOpening(int opening) {
210     this.opening = opening;
211 }
212
213 public double getIntensity() {
214     return intensity;
215 }
216
217 public void setIntensity(double intensity) {
218     this.intensity = intensity;
219 }
220
221 public double getDistanceOfPreviousColisionPoint() {
222     return distanceOfPreviousColisionPoint;
223 }
224
225 public void setDistanceOfPreviousColisionPoint(double
226     distanceOfPreviousColisionPoint) {
227     this.distanceOfPreviousColisionPoint = distanceOfPreviousColisionPoint;
228 }
229
230 public double getDistanceTraveled() {
231     return distanceTraveled;
232 }
233
234 public void setDistanceTraveled(double distanceTraveled) {
235     this.distanceTraveled = distanceTraveled;
236 }
237
238 public String getState() {
239     return state;
240 }
241
242 public void setState(String state) {
243     this.state = state;
244 }
```

```
244
245     public double getActualVirtualSoundSourcePower() {
246         return actualVirtualSoundSourcePower;
247     }
248
249     public void setActualVirtualSoundSourcePower(double
250             actualVirtualSoundSourcePower) {
251         this.actualVirtualSoundSourcePower = actualVirtualSoundSourcePower;
252     }
253
254     public double getDecibel(){
255         return 10 * Math.log10(this.getIntensity() / Math.pow(10.0, -12.0));
256     }
257
258     public double getReverberationTime(){
259         return this.distanceTraveled * (1/0.34029);
260     }
261
262     public double getSizeOfStep() {
263         return sizeOfStep;
264     }
265
266     public void setSizeOfStep(double sizeOfStep) {
267         this.sizeOfStep = sizeOfStep;
268     }
269
270     public double getError() {
271         return error;
272     }
273
274     public void setError(double error) {
275         this.error = error;
276     }
```

src/simulator/objects/SoundObject.java



# APÊNDICE J – Código fonte da classe SoundSourceObject

```

1 package simulator.objects;
2
3 import java.util.HashMap;
4
5 import jade.core.AID;
6 import jade.lang.acl.ACLMessage;
7 import jade.wrapper.AgentController;
8 import jade.wrapper.ContainerController;
9 import jade.wrapper.ControllerException;
10 import jade.wrapper.PlatformController;
11 import languagesAndMessages.Message;
12 import simulator.agents.Sound;
13 import utils.Util;
14
15 public class SoundSourceObject {
16     private AID ambientAID;
17     private AID selfAID;
18     private Location location;
19     private int opening;
20     private double power;
21     private int direction;
22     private PlatformController container;
23     private ContainerController cc;
24     private double soundSeparation;
25
26     private static HashMap<String, SoundSourceObject> soundSources = new
27         HashMap<>();
28     private HashMap<String, AID> sounds = new HashMap<>();
29
30     public static void createSoundSource(String id, AID ambient, Location
31         location, int opening, int numberOfSounds,
32         int direction) {
33         getSoundSources().put(id, new SoundSourceObject(ambient, location,
34             opening, numberOfSounds, direction));
35     }
36
37     private SoundSourceObject(AID ambient, Location location, int opening,
38         int numberOfSounds, int direction) {
39         this.setAmbient(ambient);
40         this.setLocation(location);
41     }
42
43     public void setAmbient(AID ambient) {
44         this.ambientAID = ambient;
45     }
46
47     public void setLocation(Location location) {
48         this.location = location;
49     }
50
51     public void setPower(double power) {
52         this.power = power;
53     }
54
55     public void setOpening(int opening) {
56         this.opening = opening;
57     }
58
59     public void setDirection(int direction) {
60         this.direction = direction;
61     }
62
63     public void setContainer(PlatformController container) {
64         this.container = container;
65     }
66
67     public void setCC(ContainerController cc) {
68         this.cc = cc;
69     }
70
71     public void setSoundSeparation(double soundSeparation) {
72         this.soundSeparation = soundSeparation;
73     }
74
75     public AID getAmbient() {
76         return ambientAID;
77     }
78
79     public Location getLocation() {
80         return location;
81     }
82
83     public int getOpening() {
84         return opening;
85     }
86
87     public int getDirection() {
88         return direction;
89     }
90
91     public PlatformController getContainer() {
92         return container;
93     }
94
95     public ContainerController getCC() {
96         return cc;
97     }
98
99     public double getPower() {
100        return power;
101    }
102
103    public String toString() {
104        return "SoundSourceObject [ambientAID=" + ambientAID + ", location=" +
105            location + ", opening=" + opening + ", direction=" + direction +
106            ", power=" + power + "]";
107    }
108}
```

```

37     this.setOpening(opening);
38     this.setPower(Math.pow(10.0, 6.0) * Math.pow(10.0, -13));
39     this.setSoundSeparation((double) opening / (double) numberOfSounds);
40     this.setDirection(direction);
41 }
42
43 public void emitSoundPulse() {
44     double angle;
45     createSound(location, direction, power, opening);
46     for (double i = soundSeparation; i <= opening / 2; i += soundSeparation
47         ) {
48         angle = Util.normalizeAngle(direction + i);
49         createSound(this.getLocation(), angle, power, opening);
50         angle = Util.normalizeAngle(direction - i);
51         createSound(this.getLocation(), angle, power, opening);
52     }
53
54     public AID createSound(Location location, double direction, double
55         potency, int opening) {
56         final String identifier = Sound.nextId();
57         SoundObject.createSound(new Location(location), direction, potency,
58             opening, this.getAmbient(),
59             this.getSelfAID(), identifier);
60         Object[] args = { SoundObject.getSounds().get(identifier) };
61         Util.initAgent(getContainer(), args, "simulator.agents.Sound",
62             identifier);
63
64         System.out.println(identifier + " created at: " + location);
65         AID sound = new AID(identifier, AID.ISLOCALNAME);
66         getSounds().put(identifier, sound);
67         return sound;
68     }
69
70
71     public ACLMessage suspendAllSounds() {
72         return Message.prepareMessage(ACLMessages.INFORM, null, Message.PAUSE,
73             this.getSounds());
74     }
75
76     public boolean resumeAllSounds() {
77         AgentController ac;
78         boolean resumed = false;
79         for (AID sound : this.getSounds().values()) {
80             try {
81                 ac = cc.getAgent(sound.getLocalName());
82                 ac.activate();
83                 resumed = true;
84             }
85         }
86     }
87 }
```

```

79         } catch (ControllerException e) {
80     }
81 }
82 return resumed;
83 }

84

85 public ACLMessage stopSimulation(String status) {
86     ACLMessage message = null;
87     if (status.equals(Message.STOP_RESUMED)) {
88         message = Message.prepareMessage(ACLMessages.INFORM, null, Message.
89             STOP_RESUMED, this.getSounds());
90     } else {
91         AgentController ac;
92         for (AID sound : this.getSounds().values()) {
93             try {
94                 ac = cc.getAgent(sound.getLocalName());
95                 ac.kill();
96             } catch (ControllerException e) {
97             }
98         }
99         message = Message.prepareMessage(ACLMessages.INFORM, null, Message.
100            ALREADY_STOPED, this.getSounds());
101    }
102    return message;
103 }

104 public static HashMap<String, SoundSourceObject> getSoundSources() {
105     return soundSources;
106 }

107 public AID getAmbient() {
108     return ambientAID;
109 }

110

111 public void setAmbient(AID ambient) {
112     this.ambientAID = ambient;
113 }

114

115 public Location getLocation() {
116     return location;
117 }

118

119 public void setLocation(Location location) {
120     this.location = location;
121 }

122

123 public int getOpening() {

```

```
124     return opening;
125 }
126
127 public void setOpening(int opening) {
128     this.opening = opening;
129 }
130
131 public double getPower() {
132     return power;
133 }
134
135 public void setPower(double power) {
136     this.power = power;
137 }
138
139 public int getDirection() {
140     return direction;
141 }
142
143 public void setDirection(int direction) {
144     this.direction = direction;
145 }
146
147 public AID getSelfAID() {
148     return selfAID;
149 }
150
151 public void setSelfAID(AID selfAID) {
152     this.selfAID = selfAID;
153 }
154
155 public PlatformController getContainer() {
156     return container;
157 }
158
159 public void setContainer(PlatformController container) {
160     this.container = container;
161 }
162
163 public HashMap<String, AID> getSounds() {
164     return sounds;
165 }
166
167 public void setSounds(HashMap<String, AID> sounds) {
168     this.sounds = sounds;
169 }
170
```

```
171     public ContainerController getCc() {
172         return cc;
173     }
174
175     public void setCc(ContainerController cc) {
176         this.cc = cc;
177     }
178
179     public double getSoundSeparation() {
180         return soundSeparation;
181     }
182
183     public void setSoundSeparation(double soundSeparation) {
184         this.soundSeparation = soundSeparation;
185     }
186 }
```

src/simulator/objects/SoundSourceObject.java



# APÊNDICE K – Código fonte da classe AmbientObject

```

1 package simulator.objects;
2
3 import java.util.HashMap;
4
5 import jade.core.AID;
6 import jade.wrapper.AgentController;
7 import jade.wrapper.ContainerController;
8 import jade.wrapper.ControllerException;
9 import jade.wrapper.PlatformController;
10 import simulator.agents.GUIAgentController;
11 import simulator.agents.SoundSource;
12 import utils.Util;
13
14 public class AmbientObject {
15
16     private static AmbientObject ambient = null;
17
18     public final String SOUNDSOURCE = "SoundSource";
19     public final String GUI = "GUIAgentController";
20
21     private AID ambientAID;
22     private Object[] soundSourceParameters;
23     private HashMap <String , AID> soundSources = new HashMap<>();
24     private ContainerController cc = null;
25     private PlatformController container = null;
26
27     private AmbientObject(){
28
29     }
30
31     public static AmbientObject getInstance(){
32         if(ambient == null){
33             ambient = new AmbientObject();
34         }
35         return ambient;
36     }
37
38     public void defineSoundSource(){
39         Location location = (Location) getSoundSourceParameters() [0];
40         int numberofsounds = (int) getSoundSourceParameters() [1];

```

```

41     int opening = (int) getSoundSourceParameters() [2];
42     int direction = (int) getSoundSourceParameters() [3];
43     String id = (String) getSoundSourceParameters() [4];
44
45     SoundSourceObject.createSoundSource(id, ambientAID, location, opening,
46                                         numberOfSounds, direction);
46     Object [] argsSoundSource = {SoundSourceObject.getSoundSources().get(id)
47                                 };
47
48     getSoundSources().put(id, createAgent(argsSoundSource, getContainer(),
49                             this.SOUNDSOURCE));
49 }
50
51 public AID createAgent(Object [] args, PlatformController container,
52                         String type) {
52     String id = getNextId(type);
53     Util.initAgent(container, args, "simulator.agents." + type, id);
54     if (type.equals(this.SOUNDSOURCE)){
55         System.out.println(type + " criado(a) em: " + args[0]);
56     }
57     return new AID(id, AID.ISLOCALNAME);
58 }
59
60 public String getNextId(String type) {
61     String id = new String();
62     switch(type){
63         case SOUNDSOURCE:
64             id = SoundSource.nextId();
65             break;
66         case GUI:
67             id = GUIAgentController.nextId();
68             break;
69     }
70     return id;
71 }
72
73 public void killSoundSource(String id){
74     AgentController ac;
75     try {
76         ac = getCc().getAgent(soundSources.get(id).getLocalName());
77         ac.kill();
78     } catch (ControllerException e) {
79         e.printStackTrace();
80     }
81 }
82
83 public HashMap <String, AID> getSoundSources() {

```

```
84     return soundSources;
85 }
86
87 public void setSoundSources(HashMap <String , AID> soundSources) {
88     this.soundSources = soundSources;
89 }
90
91 public AID getAmbientAID() {
92     return ambientAID;
93 }
94
95 public void setAmbientAID(AID ambientAID) {
96     this.ambientAID = ambientAID;
97 }
98
99 public PlatformController getContainer() {
100    return container;
101 }
102
103 public void setContainer(PlatformController container) {
104    this.container = container;
105 }
106
107 public ContainerController getCc() {
108    return cc;
109 }
110
111 public void setCc(ContainerController cc) {
112    this.cc = cc;
113 }
114
115 public Object [] getSoundSourceParameters() {
116    return soundSourceParameters;
117 }
118
119 public void setSoundSourceParameters(Object [] soundSourceParameters) {
120    this.soundSourceParameters = soundSourceParameters;
121 }
122 }
```

src/simulator/objects/AmbientObject.java