



UNIVERSIDADE ESTADUAL PAULISTA “JÚLIO DE MESQUITA FILHO”

FACULDADE DE ENGENHARIA - CAMPUS DE ILHA SOLTEIRA

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Interface Gráfica no Contexto de Teoria dos Jogos sob a forma de *Java Applets*

Rosane Caldeira

Orientador: Prof. Dr. Sérgio Azevedo de Oliveira

Dissertação apresentada à Faculdade de Engenharia
- UNESP, Campus de Ilha Solteira, como parte dos
requisitos para obtenção do título de Mestre em En-
genharia Elétrica.

Área de Conhecimento: Automação.

Ilha Solteira - SP

Outubro/2010

FICHA CATALOGRÁFICA

Elaborada pela Seção Técnica de Aquisição e Tratamento da Informação
Serviço Técnico de Biblioteca e Documentação da UNESP - Ilha Solteira.

C146i Caldeira, Rosane.
Interface gráfica no contexto de teoria dos jogos sob a forma de Java
Applets / Rosane Caldeira. -- Ilha Solteira : [s.n.], 2010
118 f. : il.

Dissertação (mestrado) - Universidade Estadual Paulista. Faculdade de
Engenharia de Ilha Solteira. Área de conhecimento: Automação, 2010

Orientador: Sérgio Azevedo de Oliveira

1. Teoria dos jogos. 2. Jogos não-cooperativos. 3. Jogos cooperativos. 4.
Simulador. 5. Interface gráfica. 6. Java (Linguagem de programação de
computador).



UNIVERSIDADE ESTADUAL PAULISTA
CAMPUS DE ILHA SOLTEIRA
FACULDADE DE ENGENHARIA DE ILHA SOLTEIRA

CERTIFICADO DE APROVAÇÃO

TÍTULO: Interface Gráfica no Contexto de Teoria dos Jogos sob a forma de Java Applets

AUTORA: ROSANE CALDEIRA

ORIENTADOR: Prof. Dr. SERGIO AZEVEDO DE OLIVEIRA

Aprovada como parte das exigências para obtenção do Título de MESTRE em ENGENHARIA ELÉTRICA, Área: AUTOMAÇÃO, pela Comissão Examinadora:


Prof. Dr. SERGIO AZEVEDO DE OLIVEIRA
Departamento de Engenharia Elétrica / Faculdade de Engenharia de Ilha Solteira


Prof. Dr. CARLOS ROBERTO MINUSSI
Departamento de Engenharia Elétrica / Faculdade de Engenharia de Ilha Solteira


Prof. Dr. IVAN MATHIAS FILHO
Departamento de Informática / Pontifícia Universidade Católica do Rio de Janeiro - PUC-Rio

Data da realização: 07 de outubro de 2010.

Dedicatória

Dedico este trabalho de pesquisa, primeiramente a Deus, às minhas filhas Lidyane Pilar Caldeira Apaza Rodriguez e Luana Pilar Caldeira Apaza Rodriguez pelo amor, carinho e paciência mesmo sendo tão pequeninas (fonte de inspiração em todo tempo) e a todos aqueles que cooperaram acreditando que esse estudo poderia ser concretizado com seriedade e competência.

Dedico também à toda comunidade científica; pesquisadores, professores e estudantes que se encontram afogados em informação mas sedentos por conhecimento. E que esse conhecimento possa contribuir na utilização em aplicações práticas provendo soluções científicas para melhoria da vida acadêmica e em sociedade.

Agradecimentos

A Deus por me dar saúde, força e vitalidade.

À minha família, em especial a meus pais, Arthur Caldeira e Darcy Caldeira, que me apoiaram nos cuidados de minhas filhas.

Ao professor Dr. Sérgio Azevedo de Oliveira, que me oportunizou ter estudado o tema abordado, disponibilizou os recursos necessários para que o trabalho pudesse ser desenvolvido e teve muita paciência para que o mesmo pudesse ter sido concluído.

Ao professor Dr. Carlos Roberto Minussi, por suas valiosas observações.

Ao professor Dr. Ivan Mathias Filho por sua expressiva participação na banca.

À professora Dra. Karin Breitman do Departamento de Informática do Centro Técnico Científico da PUC (CTC/PUC-Rio) por ter me ensinado a gostar de pesquisar durante o período de graduação na PUC.

Aos amigos de perto e mais distante por terem me ajudado em diversas oportunidades, sem citar nomes para não deixar de prestigiar a todos.

À CAPES, empresa de fomento, pelo apoio à pesquisa durante este período.

E também agradeço a todos do DEE e do PPGEE que de uma forma ou de outra contribuíram para a conclusão deste trabalho.

A Deus, e à minha família.

**“Faça as coisas o mais simples que você puder,
porém não se restrinja ao mais simples.”**

Albert Einstein

Resumo

Na resolução de problemas de engenharia existem diversas metodologias que podem ser aplicadas. Normalmente, estas envolvem novos conceitos, às vezes inéditos em termos de aplicação como é o caso da teoria dos jogos. Por outro lado, pesquisadores vêm buscando novas técnicas para o desenvolvimento de ferramentas computacionais para o auxílio no processo ensino-aprendizado em diferentes níveis. Nesta dissertação apresenta-se um simulador computacional que apresenta uma interface gráfica, denominada ENGTJ, como proposta para o ensino de conceitos básicos da teoria dos jogos, via Web. Os conceitos relacionados com a teoria dos jogos são apresentados sob forma de exemplos, no contexto dos jogos não-cooperativos e cooperativos, com os módulos do Jogo Dilema do Prisioneiro (JDP) e o Jogo Coalizacional entre Empresas (JCE). Estes módulos foram desenvolvidos utilizando-se a linguagem de programação Java, sob a forma de *Java applets*, com o auxílio do ambiente de desenvolvimento Eclipse. O processo de desenvolvimento de software baseado nos padrões de engenharia de software foi utilizado para criar a interface da aplicação, e apresenta uma série de técnicas e atividades que procuram dar suporte a definição de processos de desenvolvimento, levantamento e especificação de requisitos, projeto e testes. Como resultado, tem-se uma interface gráfica que permite ao usuário interagir através de jogos modulares referentes a teoria dos jogos, permitindo ao mesmo inferir alguns conceitos básicos abordados nesta teoria, suportada por tutoriais, geral e específicos.

Palavras-chave: Teoria dos Jogos. Jogos Não-cooperativos. Jogos Cooperativos. Simulador. Interface Gráfica. Java.

Abstract

In solving engineering problems there are several methodologies that can be applied. Typically, these methods involve new concepts, sometimes unprecedented in terms of application as game theory. Moreover, researchers are seeking new techniques to develop computational tools to aid in the teaching-learning at different levels. This paper presents a computational simulator that presents a graphical interface, called ENGTJ, as proposed for teaching basic concepts of game theory, through Web. The game theory concepts are presented using examples, in the context of non-cooperative games and cooperative, especially the Prisoner's Dilemma Game (JDP) and the Coalitional Games between Companies (JCE). These modules were developed using the Java programming language, under "Java applets" form with the Eclipse developing environment. The software development process based on software engineering patterns was used to create an application interface combining several techniques and activities that could support the developing procedures, requirements, projects and tests. Therefore, there is a graphical interface that allows the user interacting with modular games referred to the game theory and the concepts approached in this work, supported by tutorials, general and specific.

Keywords: Game Theory. Non-cooperative Games. Cooperative Games. Simulator. GUI. Java.

Lista de Figuras

2.1 Tela do jogo de cartas “le her” [Extraído de Bortolossi, 2007]..	26
2.2 Jogo de combinar moedas.	27
2.3 Tela de ilustração do <i>core</i>	29
2.4 James Waldegrave (1684-1741)..	31
2.5 Ernst Zermelo (1871-1953) e Émile Borel (1871-1956).	32
2.6 John von Neumann (1903-1957) e Oskar Morgenstern (1902-1977).	33
2.7 John Forbes Nash Jr (1928-).	34
2.8 John Harsanyi (1920-2000) e Reinhard Selten (1930-).	35
2.9 Jogo “hawk-dove” na forma extensiva..	46
3.1 Arquitetura da Plataforma Eclipse [Extraído de Object Technology International, Inc., 2003].	58
3.2 Gerência de recursos.	59
3.3 Bancada de trabalho [Extraído de Eclipse Foundation, Inc., 2000].	60
3.4 Processo de compilação dependente de plataforma.	65
3.5 Processo de compilação independente de plataforma.	66
4.1 Tela do menu principal..	71
4.2 Fases de desenvolvimento.	72
4.3 Diagrama de caso de uso (JDP)..	79
4.4 Diagrama de comunicação (JDP)..	79
4.5 Diagrama de classes (JDP).	80

4.6 Diagrama de caso de uso (JCE)..	82
4.7 Diagrama de comunicação (JCE)..	82
4.8 Diagrama de classes (JCE).	83
4.9 Tela de boas vindas do Eclipse.	85
4.10 Ambiente de desenvolvimento Eclipse..	85
4.11 Tela para seleção de um “wizard” de criação de projeto..	86
4.12 Tela para criação de um projeto Java.	86
4.13 Tela para criação de uma nova classe Java..	87
4.14 Código da nova classe Java criada.	88
4.15 Tela de execução do programa Java..	89
4.16 Tela de edição de código do programa.	90
4.17 Tela de depuração do programa.	90
4.18 Tela de edição de código com erro de sintaxe..	91
4.19 Tela de documentação sobre a classe <i>Math</i> do Java.	91
5.1 Tela inicial do módulo JDP.	93
5.2 Tela do módulo JDP pré-configurado.	94
5.3 Tela de processamento do módulo JDP.	95
5.4 Tela de moral do jogo do módulo JDP.	95
5.5 Tela de finalização do módulo JDP..	96
5.6 Tela de final do módulo JDP..	96
5.7 Tela com as referências específicas do módulo JDP.	97
5.8 Tela inicial do módulo JCE.	97
5.9 Opções do menu JOGO.	98
5.10 Informações sobre os autores.	98

5.11 Tela com empresas fictícias.	98
5.12 Tela de área de atuação dos jogadores (empresas).	99
5.13 Tela com configurações iniciais.	99
5.14 Tela de registro das imputações.	100
5.15 Tela de resultado das imputações estáveis e instáveis.	100
5.16 Dados para o cálculo da estabilidade ou instabilidade das imputações.	101
5.17 Tela de reinicialização das imputações.	101
5.18 Tela de moral do jogo do módulo JCE.	102
5.19 Tela de finalização do módulo JCE.	102
5.20 Tela com as referências específicas do módulo JCE.	102

Lista de Tabelas

2.1 Matriz de ganhos do jogo “hawk-dove”	26
2.2 Matriz de ganhos da batalha dos sexos.	27
2.3 Matriz de ganhos do jogo par ou ímpar.	28
2.4 Matriz de ganhos do dilema do prisioneiro.	40
2.5 Concorrência de mercado.	43
4.1 Caso de uso: sentença final do crime.	78
4.2 Caso de uso: lado de atuação do jogador no triângulo.. . . .	81

Lista de Abreviaturas

- (API) “Application Programming Interface”
- (ASP) “Active Server Page”
- (AWT) “Abstract Widget Toolkit”
- (CAL) “Computer Aided Learning”
- (CASE) “Computer-Aided Software Engineering”
- (CGI) “Common Gateway Interface”
- (CPL) “Commom Public License”
- (CVS) “Concurrent Version System”
- (EEE) Estratégia Evolucionariamente Estável
- (EJB) “Enterprise JavaBeans”
- (FTP) “File Transfer Protocol”
- (GC) “Garbage Collection”
- (GUI) “Graphical User Interface”
- (HTML) “HyperText Markup Protocol”
- (HTTP) “HyperText Transfer Protocol”
- (IDE) “Integrated Development Environment”
- (IIS) “Internet Information Server”
- (J2EE) “Java 2 Enterprise Edition”
- (J2ME) “Java 2 Micro Edition”
- (J2SE) “Java 2 Standard Edition”

(JCE) Jogo Coalizacional entre Empresas

(JDP) Jogo Dilema do Prisioneiro

(JIT) “Just in Time”

(JRE) “Java Runtime Environment”

(JSP) “Java Server Pages”

(JVM) “Java Virtual Machine”

(LLPP) Laboratório de Linux e Processamento Paralelo

(OMG) “Object Management Group”

(OMT) “Object Modeling Technique”

(OOSE) “Object Oriented Software Engineering”

(OTI) “Object Technologies International”

(PHP) “HiperText Preprocessor”

(POO) Programação Orientada a Objetos

(PU) Processo Unificado

(RAD) “Rapid Application Development”

(SWT) “Standard Widget Toolkit”

(TCP/IP) “Transmission Control Protocol/ Internet Protocol”

(UML) “Unified Modeling Language”

(URL) “Uniform Resource Locator”

(VA4J) “Visual Age for Java”

(VM) “Virtual Machine”

(WWW) “World Wide Web”

Sumário

1 Introdução	19
1.1 Motivação.	19
1.2 Introdução	20
1.3 Organização dos Capítulos	22
2 Teoria dos Jogos - Fundamentos	24
2.1 Introdução	24
2.2 Descrição dos Jogos.	25
2.2.1 O jogo de cartas “le her”	25
2.2.2 O jogo “hawk-dove”	26
2.2.3 O jogo de combinar moedas	27
2.2.4 O jogo da batalha dos sexos	27
2.2.5 O jogo do par ou ímpar.	28
2.2.6 O jogo do dilema do prisioneiro.	28
2.2.7 O jogo coalizacional entre empresas	29
2.2.8 O equilíbrio de Nash	30
2.2.9 O jogo de xadrez	30
2.2.10 O jogo de pôquer.	31
2.3 Cenário Histórico	31
2.4 Usos da Teoria dos Jogos.	35
2.5 Conceitos Básicos	37

2.6 Teoria dos Jogos Não-cooperativos	38
2.6.1 A forma normal	38
2.6.1.1 jogos estratégicos	39
2.6.1.2 estratégia pura e estratégia mista	41
2.6.1.3 estratégia dominante e estratégia dominada.	42
2.6.2 A forma extensiva.	44
2.6.2.1 informação perfeita e informação imperfeita	45
2.7 Teoria dos Jogos Cooperativos.	47
2.7.1 Coalizões	47
2.7.2 Conceitos de solução de jogos cooperativos.	48
2.7.2.1 core	49
2.7.2.2 valor Shapley	50
2.7.2.3 valor Shapley bilateral	51
2.7.2.4 kernel	51
3 O Ambiente de Desenvolvimento e a Linguagem Utilizados.	53
3.1 Introdução	53
3.2 O Ambiente Eclipse.	56
3.2.1 A arquitetura Eclipse	57
3.2.1.1 “runtime”.	58
3.2.1.2 “workspace” - a gerência de recurso	59
3.2.1.3 “workbench” - a bancada de trabalho	59
3.2.1.4 “help” - o sistema de ajuda	61
3.2.1.5 CVS - a gerência de versões.	61
3.2.1.6 multiplataforma, linguagens de programação e idioma	62

3.2.2 O projeto Eclipse@Rio	62
3.3 A Linguagem de Programação Java	63
3.3.1 Histórico	63
3.3.2 Independência de plataforma	64
3.3.3 Principais características de Java	66
3.3.4 Vantagens e desvantagens	68
3.3.5 Princípios básicos de um ambiente Java típico	69
4 A Interface Gráfica Proposta (ENGTJ)	71
4.1 Introdução	71
4.2 O Processo de Desenvolvimento nos Padrões da Engenharia de Software	72
4.2.1 Definição do modelo conceitual.	73
4.2.2 Definição do diagrama de comunicação	74
4.2.3 Definição do diagrama de classes	74
4.2.4 A linguagem de modelagem unificada (UML)	74
4.2.5 A solução programada em Java	76
4.3 O Jogo do Dilema do Prisioneiro (JDP)	77
4.3.1 Especificação de requisitos.	77
4.3.2 Funções básicas	77
4.3.3 Caso de uso para o módulo JDP.	78
4.4 O Jogo Coalizacional entre Empresas (JCE).	80
4.4.1 Especificação de requisitos.	80
4.4.2 Funções básicas	81
4.4.3 Caso de uso para o módulo JCE.	81

4.5 A Fase de Construção.	83
4.5.1 Introdução	83
4.5.2 Uso no Eclipse.	84
4.5.2.1 criação do projeto	85
4.5.2.2 compilação do programa.	88
5 Resultados e Discussões	92
5.1 Introdução	92
5.2 O Módulo JDP.	93
5.3 O Módulo JCE.	97
6 Conclusões e Sugestões para Trabalhos Futuros.	103
6.1 Conclusões	103
6.2 Sugestões para Trabalhos Futuros.	104
Referências Bibliográficas	105
Apêndice A – Ajuda para o Jogo do Dilema do Prisioneiro	107
A.1 Desenvolvimento e Organização	107
A.1.1 O módulo JDP	107
Apêndice B – Ajuda para o Jogo Coalizacional entre Empresas	109
B.1 Desenvolvimento e Organização	109
B.1.1 O módulo JCE.	109
Anexo A – Evolução das Tecnologias de Conteúdo Dinâmico (FIELDS; KOLB, 2000)	111
A.1 Introdução	111
A.2 “Java Server Pages” - JSP	112
A.3 “Common Gateway Interface” - CGI	112

A.4 Linguagens de “Script”	113
A.4.1 “ColdFusion”	113
A.4.2 “Server-Side Javascript” - SSJS	114
A.4.3 “Hypertext Preprocessor” - PHP	114
A.5 “Java Servlets”	114
A.6 “Java Servlet Pages” - JSP/Servlet	115
A.6.1 Benefícios de JSP/Servlet	116
A.7 A Edição Corporativa de JSP/Servlet e Java 2	116
A.7.1 Edições de plataforma Java	116
A.7.2 Aplicações baseadas na Web	117

***1* Introdução**

1.1 Motivação

A proposta de elaborar um programa computacional para um sistema de auxílio ao ensino de conceitos da teoria dos jogos surgiu após o primeiro contato com a teoria, em um estudo especial, quando se fez uma revisão bibliográfica do tema. Como alternativa tecnológica optou-se por desenvolver algo em *Java applets*, que possui opções de multiprogramação e processamento para Web.

Ao reunir-se algumas *aplicações* e *Java applets* para os testes iniciais, foi possível constatar a versatilidade da ferramenta. A linguagem Java mostrou-se efetivamente portátil quando alguns dos trabalhos e práticas de laboratório que outrora haviam sido compilados e executados debaixo do sistema operacional Windows obtiveram excelente desempenho ao serem processados remotamente debaixo do sistema operacional Linux.

O modo como o desenvolvimento de um software, dito livre, ocorre é considerado pouco convencional. Software livre é um tipo de programa desenvolvido por um grupo de pessoas que disponibilizam o software e o código fonte gratuitamente para que estes sejam distribuídos e alterados livremente. O número de membros desta comunidade pode chegar à casa dos milhões; os mesmos encontram-se separados por grandes distâncias e se comunicam por ferramentas de uso comum na internet (como listas de discussão, bate papo e email).

Um fator interessante, é que durante o ciclo de vida de um projeto de software livre, este pode passar pelos cuidados de diferentes desenvolvedores e por diversos caminhos paralelos de desenvolvimento. Por essas razões, um processo deste tipo não atinge um final definitivo, pois mesmo que a equipe original pense que o software já atingiu o estágio de qualidade desejado, outra equipe pode continuar o desenvolvimento deste. Pode-se dizer, então, que um projeto de software livre se encontra sempre em um contínuo processo evolutivo. Este tipo de software ganhou muita repercussão em projetos como o Linux.

Assim, debaixo da plataforma Linux, sob o ambiente de desenvolvimento Eclipse e a lin-

guagem de programação Java além dos recursos computacionais existentes, no Laboratório de Linux e Processamento Paralelo do Departamento de Engenharia Elétrica da UNESP - Campus de Ilha Solteira, iniciou-se o desenvolvimento de um programa que auxiliasse o usuário no estudo dos conceitos básicos da teoria dos jogos.

1.2 Introdução

A partir dos trabalhos de John von Neumann e Oskar Morgenstern, a teoria dos jogos (TJ) vem sendo aperfeiçoada de forma a abranger situações mais próximas da realidade e apresentando soluções matemáticas para as mesmas. Em uma definição formal, a TJ é a análise econômica-matemática que estuda o comportamento dos indivíduos em situações que existe um problema de conflito de interesses, onde tais situações denominam-se jogos (DE VASCONCELLOS, 2003 apud VON NEUMANN; MORGENSTERN, 1944).

O termo jogo tem sido utilizado como uma metáfora para modelar situações onde agentes (jogadores) compartilham em ambientes de interação para a realização de seus objetivos sejam eles individuais ou coletivos. Diversos tipos de modelos para jogos têm sido sugeridos por diferentes áreas do conhecimento, tais como matemática, engenharia, ciência da computação, ciência política e social, entre outras.

Aparentemente, o nome teoria dos jogos pode levar à confusão, contudo abrange conceitos muito mais ambiciosos do que à primeira vista possa parecer. Não é difícil encontrar-se na literatura expressões como jogo da vida, entre outros, e, claro, todas essas situações envolvem um comportamento mais ou menos estratégico e não são considerados, em absoluto, triviais ou simplesmente recreativos, mas envolvem decisões de certa importância.

A teoria dos jogos tem sido bem sucedida a partir do ponto de vista literário, tendo como marco principal a publicação do livro de von Neumann e Morgenstern, “Theory of Games and Economic Behavior” em 1944, onde procurou-se utilizar o rigor matemático para estabelecer critérios de racionalidade aos agentes que interagem em um ambiente comum na busca por certos objetivos desejados.

Naquela época a utilização na economia não era amplamente aceita e, na verdade, era muito controversa. Nesta publicação, analisou-se e comparou-se a economia a outras ciências com respeito à utilização da matemática. Para von Neumann e Morgenstern tais ciências jamais teriam o devido desenvolvimento sem o uso da matemática, considerando-se que esta evolução se deu de forma lenta e gradual através da contribuição de vários trabalhos de pesquisa produzidos ao longo dos anos. Levando-se em consideração o sucesso das outras ciências e ressaltando-se,

também, que não existiria um caminho mais curto para o desenvolvimento da TJ, este processo seria feito de forma evolutiva, onde os modelos seriam descritos e aperfeiçoados durante o passar dos anos. O que se pretendia, aparentemente, nesta publicação era iniciar uma teoria, sem necessariamente ter objetivos imediatos de modelar o comportamento humano e econômico (VON NEUMANN; MORGENSTERN, 1944).

O que von Neumann e Morgenstern pareciam acreditar na época, de fato, vem se comprovando nos dias atuais. A teoria dos jogos evoluiu e a cada passo foram sendo incorporados novos conceitos que a tornam mais próxima dos problemas reais. Esta abordagem obteve relativo sucesso e tem sido empregada em diversas áreas do conhecimento, a exemplo da economia, relações internacionais e até mesmo da biologia. O prêmio Nobel de economia já foi concedido em duas oportunidades aos pesquisadores da área de teoria dos jogos John C. Harsanyi, John Fober Nash Jr. e Reinhard Selten em 1994; Robert Aumann e Thomas Schelling em 2005.

Neste trabalho, apresentamos um simulador que possui uma interface gráfica (ENGTJ) como ferramenta proposta com objetivos educacionais, de auxílio ao processo de ensino-aprendizagem, que pode ser executada via Web, objetivando maiores facilidades para os usuários aumentarem e melhorarem seu ritmo de aprendizado, no estudo introdutório da teoria dos jogos.

O simulador computacional, através da interface gráfica, apresenta diversos recursos computacionais que permitem ao usuário interagir com alguns dos conceitos da teoria dos jogos, de uma maneira amigável (via jogos) em que os conceitos são gradativamente incorporados ao conhecimento do mesmo através da possibilidade de se executar diversas instâncias em cada jogo.

O processo de desenvolvimento, baseado nos padrões de engenharia de software, foi utilizado para criar a interface da aplicação, reunindo uma série de técnicas e atividades que visam dar suporte à definição de processos de desenvolvimento, levantamento, especificação de requisitos, projeto e testes.

Um recurso adicional incorporado à interface foi o recurso de tutorial, onde se apresenta um histórico da evolução e dos conceitos fundamentais da teoria dos jogos. E em cada módulo, são apresentados conceitos mais específicos. São disponibilizados também um glossário com os termos mais utilizados, bem como referências da literatura, gerais e específicos para cada módulo.

Na interface gráfica ENGTJ, o usuário poderá executar os módulos: Jogo do Dilema do Prisioneiro (JDP) e Jogo Coalizacional entre Empresas (JCE), no contexto dos jogos não-cooperativos e cooperativos, respectivamente.

1.3 Organização dos Capítulos

O presente trabalho encontra-se dividido em seis capítulos, que se tratam de: motivação, introdução e organização dos capítulos; descrição dos jogos, aspectos fundamentais da teoria dos jogos subdivididos em cenário histórico, teoria dos jogos e conceituação sobre a teoria dos jogos não-cooperativos e cooperativos; o ambiente de desenvolvimento integrado e a linguagem Java destacando suas vantagens e desvantagens; o processo detalhado de desenvolvimento da interface ENGTJ; resultados e discussões e por último conclusão e sugestões para trabalhos futuros.

No capítulo 1 é mostrado, de forma introdutória, que a teoria dos jogos é um estudo utilizado em situações estratégicas na tomada de decisões em um ambiente onde o jogador (agente), figura que desempenha importante papel, interage com outros jogadores (agentes) para obter retorno favorável.

No capítulo 2 narra-se os acontecimentos históricos que marcaram época em torno da teoria dos jogos. Neste capítulo mostra-se uma breve descrição sobre os diferentes jogos que introduzem os conceitos relacionados à teoria dos jogos não-cooperativos e cooperativos. Apresenta-se também a importância dos trabalhos de pesquisa desenvolvidos por John von Neumann, Oskar Morgenstern, John Forbes Nash, entre outros. Na sequência mostra-se que a teoria dos jogos se baseia em quatro elementos importantes que são: os jogadores, o jogo, os resultados, e o ganho do jogo. Também é mostrado que os jogos não-cooperativos se subdividem em duas formas de representações, que são: a forma normal, a forma extensiva e os jogos cooperativos pela forma coalizacional de representação; bem como, os conceitos de solução *core*, valor Shapley, valor Shapley bilateral e kernel.

No capítulo 3 mostra-se a utilização do “Integrated Development Environment” (IDE) Eclipse, o qual é apresentado como um ambiente propício para implementação de programas de computador escritos nas mais diversas linguagens de programação. Empresas renomadas como a IBM, a Oracle e a Borland apoiam o uso do Eclipse na criação de aplicações como páginas Web, programas Java, programas em C++ e “Enterprise JavaBeans” (EJB). A arquitetura Eclipse é composta por: “runtime”, “workspace”, “workbench” e alguns “frameworks”. Apresenta-se também o Projeto Eclipse@Rio que impulsiona a disseminação do Eclipse no meio acadêmico. Na sequência, mostra-se o desenvolvimento de ferramentas de auxílio ao ensino, executadas em ambientes distribuídos, que são reutilizáveis e proporcionam elevado nível de integração apontando Java como uma das principais plataformas utilizadas, por suas vantagens intrínsecas.

No capítulo 4 descrevem-se todas as fases do processo de desenvolvimento da interface ENGTJ: a solução programada em Java, com critérios de mudança na fase de construção e criatividade do programa. O processo de desenvolvimento da interface proposta é apresentado detalhadamente para os dois módulos do programa: o Jogo do Dilema do Prisioneiro (JDP) e o Jogo Coalizacional entre Empresas (JCE).

No capítulo 5 mostram-se os resultados e discussões sobre a interface ENGTJ, ou seja, são detalhados todos os recursos do programa divididos nos módulos: JDP, proposto com o cálculo do total das penas, a média aritmética das penas e as penas em anos, meses e dias; e JCE, proposto para o registro das imputações estáveis e instáveis, com a inferência do conceito de solução *Core*.

No capítulo 6 mostram-se as conclusões sobre o trabalho realizado. Sugestões para trabalhos futuros também são apresentados no intuito de incorporar novas funcionalidades ao programa. No final do texto, consta ainda as referências utilizadas neste trabalho, bem como apêndices com o *Ajuda* dos módulos JDP e JCE e um anexo explicativo sobre as mais modernas tecnologias de conteúdo dinâmico que podem ser utilizadas futuramente.

2 Teoria dos Jogos - Fundamentos

2.1 Introdução

Teoria dos jogos é um ramo da matemática aplicada que estuda situações estratégicas na tomada de decisões onde jogadores escolhem diferentes ações na tentativa de maximizar seu desempenho. Tendo se tornado uma área de estudo proeminente da matemática nos anos 30 do século XX, a teoria dos jogos procura encontrar estratégias racionais para situações em que o resultado depende não só da estratégia própria de um único agente e das condições de mercado, mas também das estratégias escolhidas por outros agentes que possivelmente têm estratégias diferentes ou objetivos comuns.

Embora similar à teoria da decisão, a teoria dos jogos estuda decisões que são tomadas em um ambiente onde vários agentes interagem entre si. Um modelo de teoria dos jogos estuda as escolhas de comportamentos ótimos quando a relação custo/benefício de cada opção não é fixa, mas depende da escolha dos outros indivíduos. A teoria dos jogos pode ser entendida como a análise matemática de qualquer situação que envolva um conflito de interesses com o intuito de indicar as melhores opções (ações) que, sob determinadas restrições, conduzirão aos objetivos desejados. A intuição deste conceito em termos gerais, escolhe a melhor ação (jogada) possível considerando que os jogadores sabem todas as ações dos outros agentes e sabem que cada indivíduo vai tentar escolher a melhor ação observando o que conhece das opções dos outros (DE VASCONCELLOS, 2003 apud VON NEUMANN; MORGENSTERN, 1944).

Existem conceitos sobre teoria dos jogos que propõem situações de conflito e/ou cooperação, não só para a matemática, mas também para a economia, as ciências sociais e comportamentais como meio de analisar os conflitos. Para isso, utilizam-se modelos matemáticos que descrevem interações sujeitas a um conjunto de regras, onde a meta principal é determinar quais são as estratégias racionais ótimas (TÉLLEZ, 2004).

A teoria dos jogos é também conhecida como a ciência do conflito e a importante questão ao se escolher uma estratégia é tentar prever os ganhos e as perdas potenciais que existem em cada

alternativa. Grande parte do problema reside no fato de prever-se o que os outros participantes irão fazer ou estão fazendo. Como retorno, os jogadores sempre recebem pagamentos, representados por um valor. No entanto, o valor absoluto não é tão importante quanto a proporção entre as opções (ZUGMAN, 2007).

Vale ressaltar que essa teoria considera o fato dos jogadores serem racionais. Um jogador é tido como racional mediante a forma como ele toma suas decisões. Ele está ciente de suas possíveis ações, forma expectativas sobre as indefinições do problema, tem suas preferências bem claras e toma sua decisão após algum processo de otimização. Dentro de suas possibilidades e expectativas sobre o cenário, ele tomará a decisão que mais o beneficiará.

2.2 Descrição dos Jogos

Os jogos descritos nesta seção serão citados, ao longo do trabalho, para exemplificar os conceitos sobre a teoria dos jogos. Em particular, o jogo do dilema do prisioneiro e o jogo coalizacional entre empresas serão mostrados, com mais detalhes, sob a forma computacional.

2.2.1 O jogo de cartas “le her”

A versão original do jogo foi estudada por Bernoulli e Montmort, mas foi Waldegrave que forneceu uma solução usando o conceito de estratégias mistas.

Descrição do jogo: 13 cartas de um mesmo naipe são embaralhadas. No início do jogo, o jogador 1 recebe uma carta X (que apenas ele vê), o jogador 2 recebe uma carta Y (que apenas ele vê) e uma carta Z que é colocada sobre a mesa (que ninguém vê). O jogador 1 joga primeiro: ele deve decidir se mantém a sua carta X ou a troca com a carta Y do jogador 2 (no segundo caso, o jogador 2 não pode se recusar a fazer a troca). Depois é a vez do jogador 2: ele deve decidir se mantém a sua carta ou a troca com a carta Z. Ganha quem tiver a carta de maior valor. O jogo é mostrado na Figura 2.1.

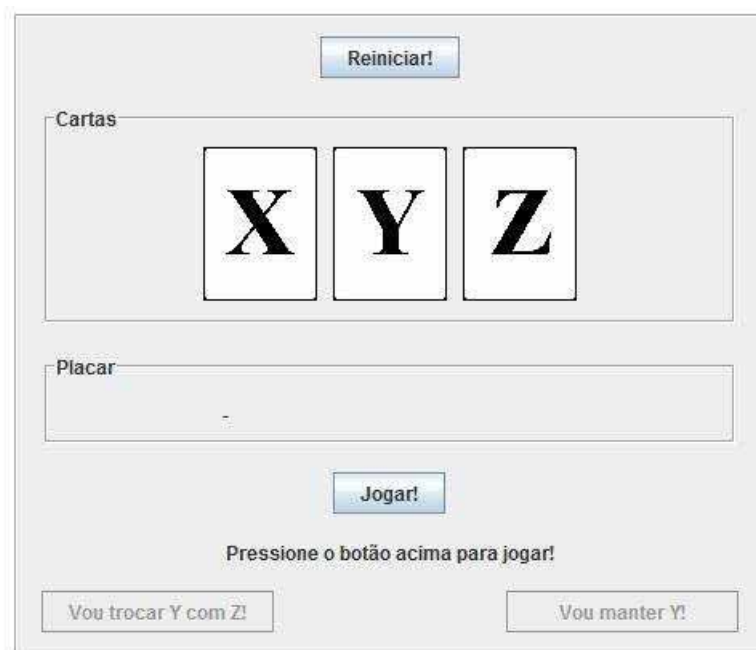


Figura 2.1: Tela do jogo de cartas “le her” [Extraído de Bortolossi, 2007].

2.2.2 O jogo “hawk-dove”

No jogo “hawk-dove” (SMITH; PRICE, 1973), os autores modelam o comportamento animal de geração em geração, onde dois jogadores (animais) disputam um recurso (uma presa) e podem escolher uma de duas estratégias: escolher (e) o conflito ou apresentar (d) suas armas (como dentes e chifres). Se os dois animais escolherem (e) o conflito continuará até que um deles fique seriamente ferido, ficando então o recurso com o vencedor. Se os dois escolherem (d) então um mecanismo aleatório (com chance de $1/2$ para cada animal) determinará com quem fica o recurso. Finalmente, se um deles escolher (e) e o outro (d) então o último fugirá do conflito e o recurso ficará com o primeiro. O valor do recurso é V e o dano causado por uma ferida seria igual a W , onde supõe-se que $W > V$. Essas informações sobre a estratégia do jogo encontram-se sistematizadas na Tabela 2.1 (ANDRADE, 2002).

Tabela 2.1: Matriz de ganhos do jogo “hawk-dove”.

		Animal 2	
		e	d
Animal 1	e	$(1/2(V - W), 1/2(V - W))$	$(V - 0)$
	d	$(V - 0)$	$(1/2V, 1/2V)$

2.2.3 O jogo de combinar moedas

No jogo de combinar moedas, dois jogadores exibem, ao mesmo tempo, a moeda que cada um esconde em sua mão. Se ambas as moedas apresentam cara ou coroa, o segundo jogador dá sua moeda para o primeiro. Se uma das moedas apresenta cara, enquanto a outra apresenta coroa, é a vez do primeiro jogador dar sua moeda para o segundo (ANDRADE, 2002). O jogo se encontra representado no grafo da Figura 2.2.

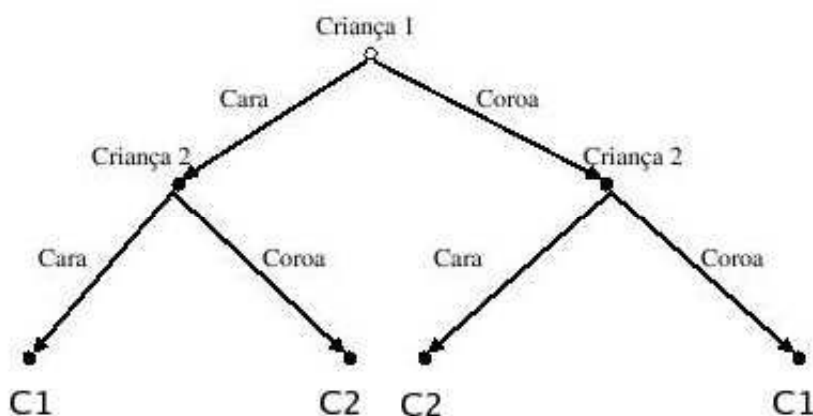


Figura 2.2: Jogo de combinar moedas.

2.2.4 O jogo da batalha dos sexos

Este jogo consiste de dois jogadores, um homem H e uma mulher M que combinaram de saírem juntos. O principal interesse deles é de se encontrar no mesmo lugar, mas o homem prefere assistir a um jogo de futebol enquanto que a mulher prefere ir ao cinema.

A Tabela 2.2 representa as preferências dos jogadores através de uma função *custo/benefício*. Nesta tabela, as linhas representam as ações possíveis para o jogador 1 (H) e as colunas são ações do jogador 2 (M). Uma entrada $(A1, A2)$ na tabela contém dois números (w_1, w_2) , onde cada um é a atribuição de valor de cada jogador dadas as ações $a1$ e $a2$. Este jogo modela uma situação onde os jogadores querem chegar num consenso, mas têm interesses conflitantes (ANDRADE, 2002).

Tabela 2.2: Matriz de ganhos da batalha dos sexos.

		Mulher	
		A	B
Homem	A	2, 1	0, 0
	B	0, 0	1, 2

2.2.5 O jogo do par ou ímpar

Dois jogadores estão jogando *par ou ímpar*, onde cada jogador pode optar por colocar um número par ou ímpar. Os jogadores mostram simultaneamente, um número par ou ímpar. Se a soma dos números for par, o jogador 1 vence. Se for ímpar, o jogador 2 vence. Cada jogador possui duas estratégias, mostrar um número par ou mostrar um número ímpar. O jogo é mostrado na Tabela 2.3. O jogador que escolhe par ganha quando os números forem dois pares ou dois ímpares. O outro ganha quando os números forem um par e um ímpar. Assim, o jogador que perder terá que pagar ao outro um real.

Tabela 2.3: Matriz de ganhos do jogo par ou ímpar.

		Jogador 2	
		Par	Ímpar
Jogador 1	Par	1, -1	-1, 1
	Ímpar	-1, 1	1, -1

2.2.6 O jogo do dilema do prisioneiro

O dilema do prisioneiro foi, originalmente, proposto por Melvin Dresher e Merrill Flood, em 1950. Tucker, (TUCKER, 1980), formalizou o jogo estipulando penas como pagamento e deu a ele o nome de dilema do prisioneiro. Este jogo acaba tendo muitas implicações no estudo da cooperação entre indivíduos.

O modelo clássico do dilema do prisioneiro consiste de dois suspeitos acusados de uma violação da lei que são mantidos separados pela polícia. À cada um é dito que se ele confessar e o outro não, ele obterá um prêmio enquanto que o outro irá para a prisão. Se ambos confessarem, ambos irão para a prisão. Se nenhum confessar, a polícia poderá sentenciar ambos por acusações menos graves (porte de armas) (ANDRADE, 2002).

As representações padrão do dilema do prisioneiro são jogos simétricos, ou seja, jogos

nos quais os pagamentos para os jogadores em uma estratégia particular dependem somente da estratégia escolhida, e não de quem está jogando. O dilema do prisioneiro apresenta um aparente conflito entre a moralidade e o auto-interesse, explicando porque a cooperação é requerida pelo auto-interesse, sendo importante componente. Esta estratégia comum é um componente da visão contrato social geral (GAUTHIER, 1987) (KAVKA, 1986). No exemplo da seção 2.6.1.1, página 39, o jogo do Dilema do Prisioneiro é apresentado com mais detalhes.

2.2.7 O jogo coalizacional entre empresas

A dinâmica do Jogo Coalizacional entre Empresas encontram-se descrita em (FERGUSON, 2009) no trabalho de estudo do professor e pesquisador Thomas S. Ferguson da Universidade da Califórnia. O Jogo Coalizacional entre Empresas é um jogo composto por três empresas que estão em negociação para juntas implementarem um projeto que trará benefícios comuns aos participantes. O jogo baseia-se nas coordenadas de um triângulo equilátero na forma coalizacional (de acordos) para descrever quais coalizões detém imputações estáveis e quais não detém, levando-se em consideração os critérios de conjunto vazio e superaditividade identificados no conceito de solução *core*, (Figura 2.3), presente na teoria dos jogos cooperativos.

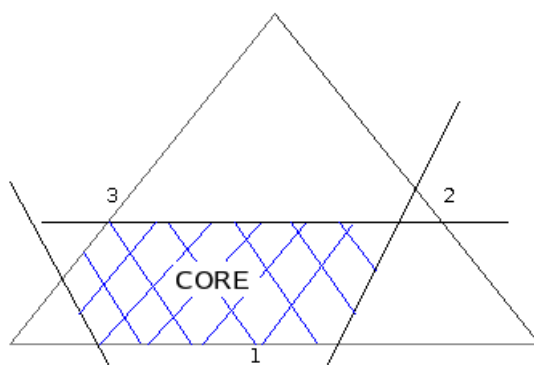


Figura 2.3: Tela de ilustração do *core*.

Para ilustrar o cálculo do *core* são atribuídos valores numéricos para o jogo com função característica v dado por (FERGUSON, 2009):

$$v(\emptyset) = 0, v(1) = 1, v(2) = 0, v(3) = 1, v(1, 2) = 4, v(1, 3) = 3, v(2, 3) = 5, v(1, 2, 3) = 8 \quad (2.1)$$

A função característica do jogo associa um valor para cada subconjunto de jogadores. Em um jogo de n jogadores, a função característica terá 2^n valores. As três empresas possuirão então

oito possíveis ($2^3 = 8$) valores. Em $v(\emptyset) = 0$, uma coalizão sem jogadores não gera ganho. Quando $v(1) = 1$, $v(2) = 0$ e $v(3) = 1$, significa que sozinhos os jogadores não são capazes de implementar o projeto. Em $v(1,2) = 4$, $v(2,3) = 5$ e $v(1,3) = 3$, os dois primeiros valores indicam que a associação do jogador 2 com o jogador 1 ou com o jogador 3 gera os valores mais altos. O último valor de $v(1,2,3) = 8$ representa o resultado caso haja a cooperação entre os jogadores simultaneamente. É no conjunto de soluções que se encontrará a distribuição final do ganho por intermédio de uma negociação entre as três empresas (FERGUSON, 2009). Na seção 2.7.2.1, página 49, são apresentados mais detalhes do Jogo Coalizacional entre Empresas.

2.2.8 O equilíbrio de Nash

No âmbito da teoria dos jogos, existem diversas formas de definir um equilíbrio. A definição mais empregada, devido as suas propriedades, é a denominada para o Equilíbrio de Nash (NASH, 1950).

O Equilíbrio de Nash consiste de um conjunto de estratégias, uma para cada jogador, tal que nenhum jogador pode beneficiar-se mudando unilateralmente a sua estratégia enquanto os demais mantêm as suas estratégias inalteradas.

Equilíbrio de Nash é um dos experimentos mais conhecidos e usados para analisar soluções em jogos. Essa noção captura um estado estacionário para o jogo estratégico, onde cada jogador tem a expectativa correta sobre o comportamento dos outros jogadores, e age racionalmente. Em exemplos bem simples, como a batalha dos sexos, o dilema do prisioneiro, o jogo do par-ou-ímpar, onde existem dois jogadores e duas ações, são analisados os equilíbrios de Nash em diferentes situações (ANDRADE, 2002).

A genialidade do Equilíbrio de Nash vem da sua estabilidade sem os jogadores estarem cooperando. A teoria dos jogos explica porquê nos grandes centros urbanos, farmácias, locadoras e outros competidores da mesma indústria tendem a ficar próximos uns aos outros. Sempre que um jogador se encontra em uma situação em que até poderia estar melhor, mas está fazendo o melhor possível dada a posição de seus competidores, existirá um equilíbrio de Nash (ZUGMAN, 2007).

O Equilíbrio de Nash é um conceito de solução de jogos entre duas ou mais pessoas em que nenhum jogador pode melhorar seus ganhos apenas alterando sua própria estratégia. Ele tenta prever o comportamento dos jogadores, englobando as ações que são as melhores respostas para cada um.

2.2.9 O jogo de xadrez

O jogo do xadrez consiste de dois jogadores adversários entre si que movimentam peças num tabuleiro quadrado chamado *tabuleiro de xadrez*. O jogador que possui as peças brancas começa o jogo. Diz-se *a vez* do jogador quando seu adversário completou o lance.

O objetivo de cada jogador é colocar o rei adversário *sob ataque* de tal forma que o adversário não tenha lance legal para evitar a captura de seu rei no lance seguinte. O jogador que alcançou tal objetivo ganhou a partida e diz-se que deu *mate* no adversário. O jogador que levou o *mate* então perdeu a partida. Se numa determinada posição nenhum dos jogadores pode dar *mate*, a partida está empatada.

2.2.10 O jogo de pôquer

Pôquer, ou “poker”, é um jogo de cartas jogado por duas ou mais pessoas. É o mais popular de uma classe de jogos nos quais os jogadores com as cartas total ou parcialmente escondidas fazem apostas (usando fichas) para um monte central, após o que o resultante das apostas é atribuído ao(s) jogador(es) que possuir(em) o melhor conjunto de cartas. Os jogadores apostam na força das suas cartas.

2.3 Cenário Histórico

Registros antigos sobre teoria dos jogos remontam ao século XVIII. Em correspondência dirigida a Nicholas Bernoulli, James Waldegrave (Figura 2.4) analisa um jogo de cartas chamado “le her” e fornece uma solução que é um equilíbrio de estratégia mista¹ (BORTOLOSSI; GARBAGIO; SARTINI, 2007). Em seu estudo, Waldegrave procurou encontrar uma estratégia que maximizasse a probabilidade de vitória do jogador, independentemente da escolha de estratégia de seu oponente. Este jogo foi discutido por Pierre Rémond de Montmort e por Nicholas Bernoulli em 1713. Os resultados foram publicados naquele ano por Montmort, que incluiu a solução de Waldegrave em um apêndice (BORTOLOSSI; GARBAGIO; SARTINI, 2007).

¹Uma estratégia pura é uma das escolhas que o jogador pode fazer. Uma estratégia mista é uma distribuição de probabilidades sobre o conjunto de estratégias puras.



Figura 2.4: James Waldegrave (1684-1741).

Em 1838, Antonie Augustin Cournot publicou sua obra “Recherches sur les Principes Mathematiques de la Theorie des Richesses”, na qual analisou um caso especial de duopólio. As empresas decidiam as quantidades a produzir e Cournot definiu o conceito de equilíbrio de mercado como sendo a situação em que ambas as empresas reagem de forma ótima à decisão da empresa concorrente. Este conceito de solução é uma versão do equilíbrio de Nash aplicado ao caso do duopólio (BORTOLOSSI; GARBAGIO; SARTINI, 2007).

Historicamente, pode-se dizer que a primeira contribuição à teoria dos jogos foi o trabalho de Zermelo, em 1913, sobre o xadrez. Naquele mesmo ano, Ernst Zermelo publicou um teorema sobre o jogo de xadrez no artigo “Uber eine Anwendung der Mengenlehre auf die Theorie des Schachspiels”, afirmando que, em cada etapa do jogo, pelo menos um dos jogadores possui uma estratégia que o levará a vitória ou ao empate (ZERMELO, 1913). Outro matemático que se interessou por jogos foi Émile Borel, tendo publicado uma série de artigos sobre jogos estratégicos. Assim, os primeiros resultados matemáticos em teoria dos jogos devem-se a Ernst Zermelo e a Émile Borel (Figura 2.5).



Figura 2.5: Ernst Zermelo (1871-1953) e Émile Borel (1871-1956).

Apesar da análise de Cournot ser mais geral do que a de Waldegrave, a teoria dos jogos não existia como um campo unificado até que John von Neumann publicou uma série de trabalhos em 1928 estabelecendo os primeiros esboços de uma teoria científica especializada em lidar com o conflito humano.

No artigo “Zur Theorie der Gesellschaftsspiele” de 1928, John von Neumann, demonstrou a existência de solução em estratégias mistas de um jogo finito de soma zero com dois jogadores e um número arbitrário de estratégias pura. Este artigo também introduziu a forma extensa de um jogo (BORTOLOSSI; GARBAGIO; SARTINI, 2007).

Até a década de 40, os artigos publicados sobre teoria dos jogos não tinham despertado muito o interesse dos cientistas sociais e de outras áreas que pesquisavam sobre conflitos de interesses. Talvez isto se deve ao fato de que os artigos eram escritos por matemáticos e publicados em revistas matemáticas (BORTOLOSSI; GARBAGIO; SARTINI, 2007).

John von Neumann pode, com justiça, ser creditado como o inventor da Teoria dos Jogos. O trabalho de von Neumann culminou no livro lançado em 1944, “The Theory of Games and Economic Behaviour” com a co-autoria de Oskar Morgenstern (Figura 2.6), onde a teoria dos jogos passou a ser aplicada na economia e também foi considerado o trabalho que estabeleceu a teoria dos jogos como campo de estudo (VON NEUMANN; MORGENSTERN, 1944).



Figura 2.6: John von Neumann (1903-1957) e Oskar Morgenstern (1902-1977).

Eles detalharam a formulação de problemas econômicos e mostraram várias possibilidades de aplicação da teoria dos jogos em economia, procurando apresentar as motivações, os raciocínios e as conclusões da teoria de forma acessível, atraindo assim a atenção de pesquisadores de diversas áreas. Na reedição de 1947, tomada como padrão, os autores estabeleceram os axiomas da teoria da utilidade. O livro foi republicado em 1953 e sua mais recente edição é de 1980 (BORTOLOSSI; GARBAGIO; SARTINI, 2007).

Durante a Segunda Guerra Mundial, a Marinha Britânica baseou-se em ideias diretamente relacionados com a teoria dos jogos para melhorar a sua capacidade de prever os movimentos dos submarinos alemães. Após a Segunda Guerra Mundial, o desenvolvimento da teoria dos jogos intensificou-se no período da Guerra Fria. John von Neumann demonstrou que todo jogo de soma zero com duas pessoas possui uma solução em estratégias mistas.

A teoria dos jogos experimentou uma atividade intensa nos anos 50, durante a qual conceitos de jogos na forma extensiva, jogador fictício, jogos repetidos e o valor Shapley foram desenvolvidos. Na Universidade de Princeton, John Forbes Nash Jr. (Figura 2.7) escreveu sua tese de doutorado, em 1949, sob o título “Non-Cooperative Games”. Ele definiu o conceito de *ponto de equilíbrio*, atualmente conhecido como *equilíbrio de Nash* de um jogo e provou sua existência para jogos não-cooperativos. Os resultados mais importantes de sua tese estão no artigo “Equilibrium Points in N-Person Games” de 1950. Mais detalhadamente, no artigo “Non-Cooperative Games” de 1951. Depois publicou “The Bargaining Problem” e, no ano de 1953 outro artigo sobre jogos cooperativos em “Two Person Cooperative Games”. Nestes, Nash definiu o conceito de *solução da barganha de Nash* em um jogo cooperativo com dois jogadores. Estabeleceu um sistema de axiomas que esta solução deveria satisfazer e provou a existência e unicidade desta solução (BORTOLOSSI; GARBAGIO; SARTINI, 2007).



Figura 2.7: John Forbes Nash Jr (1928-).

A partir dos trabalhos de Neumann, Morgenstern e Nash, a teoria dos jogos vem sendo aperfeiçoada de forma a abranger situações mais próximas da realidade e apresentando soluções matemáticas para as mesmas. John von Neumann se estabeleceu nos EUA, ajudou a desenvolver a bomba-A e, entre outras realizações, inventou o computador digital. Depois da Segunda Guerra, Neumann foi contratado pela RAND Corporation², onde aplicou a teoria dos jogos

²A RAND Corporation, com sede na Califórnia, é uma instituição sem fins lucrativos que realiza pesquisas para contribuir com a tomada de decisões e a implementação de políticas no setor público e privado.

mais produtivamente na estratégia da Guerra Fria. Ainda na década de 50, os resultados das pesquisas de Melvin Dresher e Merrill Flood na corporação RAND levaram à descoberta do jogo conhecido atualmente como o Dilema dos Prisioneiros (TÉLLEZ, 2004).

A partir de 1970, a teoria dos jogos passou a ser aplicada ao estudo do comportamento animal, incluindo evolução das espécies por seleção natural e sendo, então, extensivamente aplicada à biologia.

Em 1980, baseado em um memorando de 1950, Albert W. Tucker formalizou o modelo do dilema do prisioneiro que tem diversas implicações para o estudo da natureza da cooperação humana (TUCKER, 1980).

Em 1994, John Forbes Nash Jr. (Universidade de Princeton), John Harsanyi (Universidade de Berkeley, Califórnia) e Reinhard Selten (Universidade de Bonn, Alemanha) (Figura 2.8) receberam o Prêmio Nobel por suas contribuições para a teoria dos jogos (BORTOLOSSI; GARBAGIO; SARTINI, 2007).



Figura 2.8: John Harsanyi (1920-2000) e Reinhard Selten (1930-).

Em 2005, o cientista da teoria dos jogos Thomas C. Schelling (Universidade de Berkeley) e Robert J. Aumann (Universidade de Jerusalém) ganharam o Prêmio Nobel trabalhando com modelos dinâmicos.

Neste período de tempo até os dias atuais, o número de aplicações a situações reais onde aparecem conflitos de interesse tem sido muito importante. Recentemente, a teoria de jogos encontrou aplicações na resolução de problemas em sistemas elétricos de potência, como alocação de perdas, congestionamento, e especialmente em temas relacionados ao mercado elétrico (CONTRERAS, 1997).

2.4 Usos da Teoria dos Jogos

Inicialmente desenvolvida como ferramenta para compreender comportamento econômico e depois pela “RAND Corporation” para definir estratégias nucleares, a teoria dos jogos é, atualmente, usada em diversos campos acadêmicos. A teoria dos jogos vem sendo aplicada na economia, biologia, ciência política, ética, filosofia, jornalismo, psicologia, nas áreas de ciência da computação, lógica e engenharia. Os resultados da teoria dos jogos tanto podem ser aplicados aos jogos de entretenimento como aos aspectos significativos da vida em sociedade.

Os biólogos utilizam a teoria dos jogos para compreender e prever o desfecho da evolução de certas espécies. Esta aplicação à teoria da evolução produziu conceitos importantes como o da Estratégia Evolucionariamente Estável (EEE) (SMITH, 1982). Os biólogos têm usado teoria dos jogos evolucionários e a EEE para explicar o surgimento da comunicação entre animais.

Na economia, a teoria dos jogos tem sido usada para examinar a concorrência e a cooperação dentro de pequenos grupos de empresas. Os economistas têm usado a teoria dos jogos para analisar fenômenos econômicos incluindo leilões, barganhas, oligopólios, formação de rede social e sistemas de votação. A teoria dos jogos distingue-se na economia na medida em que procura encontrar estratégias racionais.

Na ciência da computação e lógica, a teoria dos jogos veio impulsionar importantes leis tanto na modelagem de computação interativa como na lógica. Existe também conexão entre a teoria dos jogos e as áreas matemáticas.

Em ciência política, uma explicação para a paz democrática é o debate público que transmite informações confiáveis a respeito da opinião pública em relação aos outros estados. Na filosofia, respondendo aos trabalhos de (QUINE, 1967) e (LEWIS, 1969) usou-se a teoria dos jogos para desenvolver uma explicação filosófica da convenção.

Na ética, alguns autores têm procurado impulsionar e derivar a moralidade do auto-interesse. No jornalismo, a teoria dos jogos tem aplicações na cooperação entre fonte anônima e repórter/veículo jornalístico.

Em relações internacionais, essa teoria desempenha um papel central em estratégia militar, no estudo de situações que envolvem tomada de decisões em questões de política internacional e no entendimento de vários aspectos das interações dinâmicas entre agentes em conflito.

Existem jogos psicológicos, jogados em um nível pessoal, em que palavras são as armas e os benefícios são sentimentos bons ou ruins.

A teoria dos jogos vem encontrando uma série de aplicações com êxito nos sistemas de

energia elétrica para problemas relativos a mercados de energia, expansão de sistemas de transmissão, alocação de perdas, entre outros.

Em complemento ao interesse acadêmico, a teoria dos jogos vem recebendo atenção da cultura popular. O pesquisador da teoria dos jogos e ganhador do prêmio Nobel de economia, John Forbes Nash Jr., foi sujeito, em 1998, de uma biografia escrita por Sylvia Nasar e de um filme em 2001 (*Uma mente brilhante*). A teoria dos jogos também foi tema do filme *Jogos de guerra* em 1983.

Desde as primeiras décadas do século XX até os dias atuais, vários resultados da teoria dos jogos têm contribuído para a elucidação de problemas econômicos, sociais e políticos. A teoria dos jogos tem contribuído substancialmente para uma abordagem satisfatória de inúmeras questões. A teoria dos jogos trouxe uma nova visão às ciências sociais. Seus conceitos e metodologia são facilmente aplicáveis. Assim, trabalha-se o mundo social a partir de modelos baseados em jogos de estratégias e na semântica dos jogos. Jogos de uma forma ou de outra são vastamente usados em diversas disciplinas acadêmicas. O uso da teoria dos jogos é para se conhecer, previamente, o melhor resultado para os jogadores diante das estratégias praticadas (WIKISLICE, 2008).

2.5 Conceitos Básicos

É possível pensar num jogo como um cenário onde os jogadores interagem. Esses jogadores possuem um conjunto de decisões (ou ações) passíveis de serem tomadas. A tomada de decisão é baseada nas preferências de cada jogador e na sua expectativa sobre as ações dos outros jogadores que pode ser um evento probabilístico. É justamente nessa dinâmica que a teoria de jogos foca seu estudo (ANDRADE, 2002).

Os jogos estudados pela teoria dos jogos constituem objetos matemáticos bem definidos. O elemento básico em um jogo é o conjunto de jogadores que dele participam. Cada jogador tem um conjunto de estratégias. Quando cada jogador escolhe sua estratégia, tem-se então uma situação ou perfil no espaço de todas as situações (perfis) possíveis. Cada jogador tem interesse ou preferência para cada situação no jogo. Em termos matemáticos, cada jogador tem uma função utilidade que atribui um número real (o ganho do jogador) a cada situação do jogo (BORTOLOSSI; GARBAGIO; SARTINI, 2007).

Ao ter-se que as preferências de cada jogador podem ser representadas numericamente por uma função de utilidade, dadas duas ações, um indivíduo preferirá uma delas somente se a sua utilidade esperada for maior do que para a outra. Num sentido mais amplo, um jogo é um

modelo teórico para uma situação definida por interesses competitivos, em que cada jogador procura maximizar seus ganhos. A teoria dos jogos pode ser vista como uma extensão da teoria da decisão para o caso de haver dois ou mais jogadores e onde busca-se entender o conflito e a cooperação através de modelos quantitativos e de exemplos hipotéticos (TÉLLEZ, 2004).

Para se descrever um jogo, existem cinco elementos básicos necessários que são (BERTRAND, 2005):

- *Jogadores*: É necessário definir quantos participantes existem no jogo e como eles se comportam. Um jogador joga por meio de ações que são escolhas que fazem um jogador de acordo com sua própria estratégia.
- *Conjunto de regras do jogo*: As regras do jogo indicam o que um participante pode ou não fazer. Também indicam a ordem em que os participantes jogam e a informação que tem antes de realizar suas ações.
- *Estratégia*: É o plano de ação completo num determinado jogo.
- *Resultado do jogo*: É necessário conhecer como transcorre e como acaba o jogo para qualquer combinação de ações que sejam feitas pelos jogadores. Este resultado depende das estratégias escolhidas por cada um dos jogadores.
- *Funções de Utilidade dos Jogadores*: Estas funções indicam as preferências que tem cada um dos jogadores com relação aos possíveis resultados do jogo.

Um jogo é uma representação formal de uma situação em que um número de indivíduos interage num conjunto de interdependência estratégica. Isso significa que a prosperidade de um indivíduo depende de suas próprias ações e das ações dos outros participantes do jogo (ZUGMAN, 2007).

De acordo com as circunstâncias do jogo e dependendo da forma como os jogadores se comportam, distingue-se dentro da teoria dos jogos duas classificações existentes, que são: *jogos não-cooperativos* e *jogos cooperativos*.

2.6 Teoria dos Jogos Não-cooperativos

Em jogos não-cooperativos, o conjunto de ações possíveis está associado a jogadores individuais (ANDRADE, 2002). O objetivo do jogo de xadrez, por exemplo, é capturar o rei do adversário. No poquer, cada jogador busca conseguir a melhor sequência de cartas e, assim,

vencer o jogo. O que esses jogos possuem em comum é que em ambos os casos cada jogador busca melhorar seu benefício individual. Estes são exemplos de jogos não-cooperativos, onde cada jogador defende seus próprios interesses, montando estratégias para conseguir alcançar o maior benefício possível.

Distingue-se dentro da teoria dos jogos diversas classes de jogos de acordo com o número de jogadores e com as circunstâncias do jogo. São jogos estratégicos (“strategic games”), aqueles em que os jogadores agem ao mesmo tempo e uma única vez; e jogos extensivos (“extensive games”), aqueles em que apenas um jogador joga a cada vez e ele conhece o histórico de jogadas anteriores. Desta maneira, consegue-se caracterizar melhor cada classe de jogo criando-se conceitos específicos sobre o comportamento dos jogadores (DE VASCONCELLOS, 2003). Existem duas formas de representação de jogos, comuns na literatura, e que são conhecidas como a *forma normal* e a *forma extensiva*.

2.6.1 A forma normal

O jogo na forma normal (ou modo estratégico) é uma matriz a qual mostra os jogadores, estratégias e pagamentos. Onde existem dois jogadores, um escolherá as linhas e o outro escolherá as colunas. Os pagamentos são registrados no seu interior. O primeiro número é o pagamento recebido pelo jogador da linha e o segundo é o pagamento para o jogador da coluna.

A forma normal é usada em situações onde os jogadores escolhem sua estratégia simultaneamente ou o fazem sem conhecer a estratégia dos outros jogadores. A representação de um jogo em forma normal, ou estratégica, é a maneira mais simples de descrever um jogo. Permite descrever jogos combinatórios finitos. Na forma normal, uma maneira compacta de se descrever os aspectos matemáticos de um jogo é permitindo a utilização de métodos simples em sua análise. Um jogo em forma normal é representado por uma matriz de pagamentos, onde são apresentadas as estratégias de cada jogador e os seus ganhos para cada perfil de estratégias. A estratégia no jogo de xadrez, por exemplo, é a descrição de quais os movimentos possíveis em cada situação que possa ocorrer.

2.6.1.1 jogos estratégicos

Um jogo estratégico é um modelo de tomada de decisão interativa no qual cada responsável pelas decisões escolhe sua ação uma única vez, e todos os jogadores fazem suas escolhas simultaneamente. Os conceitos introduzidos nos jogos estratégicos são bem simples, e ao mesmo

tempo, suficientes para ilustrar a ideia.

Um jogo estratégico consiste de um conjunto finito (o conjunto de jogadores), onde para cada jogador há um conjunto não vazio e uma relação de preferência sobre o conjunto (DE VASCONCELLOS, 2003).

Definição 1: *Jogos Estratégicos (“Strategic Games”)*

$$G = \{J, (A_i), (\succeq_i)\} \quad (2.2)$$

- um conjunto finito J (de jogadores);
- para cada jogador $i \in J$ um conjunto não-vazio A_i (de ações disponíveis para o jogador i);
- para cada jogador $i \in J$, uma relação de preferência (\succeq_i) sobre o conjunto $A = \prod A_i$.

Exemplo Prático do Dilema do Prisioneiro

Possivelmente, o exemplo mais conhecido na teoria dos jogos é o dilema do prisioneiro. Na situação em que dois ladrões, Al e Bob, são capturados e acusados de um mesmo crime. Presos em celas separadas e sem poderem se comunicar entre si, o delegado de plantão faz a seguinte proposta: cada um pode escolher entre confessar ou negar o crime. Se nenhum deles confessar, ambos serão submetidos a uma pena de 1 ano. Se os dois confessarem, então ambos terão pena de 5 anos. Mas se um confessar e o outro negar, então o que confessou será libertado e o outro será condenado a 10 anos de prisão. (BORTOLOSSI; GARBAGIO; SARTINI, 2007) Neste contexto, tem-se:

$$\begin{aligned} G &= \{Al, Bob\}, \\ S_{Al} &= \{confessar, negar\}, S_{Bob} = \{confessar, negar\}, \\ S &= \{(confessar, confessar), (confessar, negar), (negar, confessar), (negar, negar)\} \end{aligned}$$

As duas funções de utilidade são:

$$U_{Al} : S \rightarrow \mathcal{R} \text{ e } U_{Bob} : S \rightarrow \mathcal{R}$$

que são dadas por:

$$\begin{aligned} U_{Al}(confessar, confessar) &= 5, U_{Al}(confessar, negar) = 0 \\ U_{Al}(negar, confessar) &= 10, U_{Al}(negar, negar) = 1 \end{aligned}$$

que representam os ganhos de Al e

$$U_{Bob}(confessar, confessar) = 5, U_{Bob}(confessar, negar) = 10$$

$$U_{Bob}(negar, confessar) = 0, U_{Bob}(negar, negar) = 1$$

que representam os ganhos de Bob.

É uma prática representar os ganhos dos jogadores através de uma matriz de ganhos, como mostra a Tabela 2.4.

Tabela 2.4: Matriz de ganhos do dilema do prisioneiro.

		Bob	
		confessar	não-confessar
Al	confessar	5, 5	0, 10
	não-confessar	10, 0	1, 1

Nesta matriz, os números de cada célula representam, respectivamente, os ganhos de Al e Bob para suas escolhas.

2.6.1.2 estratégia pura e estratégia mista

Estratégia pura é aquela que não envolve experimentos aleatórios. Estratégia mista é aquela que envolve aleatoriedade (BORTOLOSSI; GARBAGIO; SARTINI, 2007).

No jogo do *par ou ímpar*, se o jogador 1 escolhe sempre mostrar um número par, independentemente da estratégia que o jogador 2 venha a escolher, diz-se que ele está jogando com uma estratégia pura. Caso contrário, se o jogador 1 escolhe arbitrariamente entre mostrar um número par ou ímpar, então ele está jogando com uma estratégia mista. Assim, é possível verificar neste jogo que não existe estratégia ótima (pura ou mista) independente do oponente. Se o jogador 1 mostra um número par, a estratégia ótima do jogador 2 é mostrar um número ímpar. Se o jogador 1 mostra um número ímpar, a estratégia ótima do jogador 2 é mostrar um número par. Este conceito é denominado “melhor resposta”.

Denomina-se *equilíbrio de Nash em estratégias puras*, aquele equilíbrio no qual a forma de jogar de todos os jogadores consiste em eleger sistematicamente uma certa estratégia concreta. Por outro lado, denomina-se *equilíbrio de Nash em estratégias mistas*, aquele equilíbrio no qual a estratégia de pelo menos um dos jogadores consiste em eleger entre um conjunto de estratégias

uma dada estratégia mediante atribuição de probabilidade a cada uma delas (BERTRAND, 2005).

Em geral, um jogo pode não ter nenhum equilíbrio de Nash em estratégias puras, pode ter um único equilíbrio de Nash em estratégias puras ou pode ter múltiplos equilíbrios de Nash em estratégias puras. Por outro lado, pode-se demonstrar que para uma ampla gama de jogos, sempre existe pelo menos um equilíbrio de Nash em estratégias mistas (BERTRAND, 2005).

Assim como no jogo de combinar moedas, existem jogos que não possuem equilíbrio de Nash em estratégias puras. Uma alternativa para estes casos é a idéia de considerar o jogo do ponto de vista probabilístico, ou seja, escolher uma distribuição de probabilidade sobre suas estratégias puras. Uma estratégia mista para o jogador é uma distribuição de probabilidades sobre o conjunto de estratégias puras do jogador (BORTOLOSSI; GARBAGIO; SARTINI, 2007).

Todos os critérios básicos para solução de jogos em estratégias puras podem ser estendidos para estratégias mistas. No dilema do prisioneiro, o perfil de estratégia mista é um equilíbrio de Nash. O único equilíbrio de Nash em estratégias mistas do jogo é o par (confessar, confessar). Na batalha dos sexos, o equilíbrio de Nash em estratégias mistas são correspondentes aos equilíbrios de Nash em estratégias puras. Neste jogo, os perfis de estratégia (futebol, futebol) e (cinema, cinema) são os únicos equilíbrios de Nash do jogo (BORTOLOSSI; GARBAGIO; SARTINI, 2007).

2.6.1.3 estratégia dominante e estratégia dominada

Quando, dentro do conjunto de estratégias de um jogador, existe uma estratégia que garante melhores resultados independentemente da estratégia escolhida pelos outros jogadores, então existe uma estratégia dominante. Se os benefícios oferecidos pela estratégia dominante forem sempre maiores do que os oferecidos pelas outras estratégias, diz-se que ela é estritamente dominante. Um exemplo clássico que ilustra este tipo de estratégia refere-se ao problema do dilema do prisioneiro. Neste jogo, confessar é uma estratégia dominante, pois independentemente da estratégia escolhida pelo outro prisioneiro, confessar garante melhores resultados do que não confessar, que é uma estratégia dominada. Baseado nisto e no conceito de melhor resposta, surge um dos conceitos de solução mais importantes da teoria dos jogos não-cooperativos, o *equilíbrio de Nash*.

Uma estratégia para um jogador é *fortemente dominada* se existe outra estratégia que proporciona ao jogador maior utilidade em todos os casos. Uma estratégia para um jogador é uma estratégia *debilmente dominada* se há outra estratégia que proporciona ao jogador uma utilidade

maior ou igual em todos os locais. Por último, uma estratégia é *dominante* se todas as demais estão dominadas por ela (BERTRAND, 2005).

Para jogos com dominância estratégica, quando os jogadores optam por jogar com a estratégia dominante, eles chegam simultaneamente ao equilíbrio de Nash. A solução de dominância estrita é o único equilíbrio de Nash para o jogo.

No dilema do prisioneiro, o perfil de estratégia (confessar, confessar) é um equilíbrio de Nash (BORTOLOSSI; GARBAGIO; SARTINI, 2007). Porém, observa-se que o equilíbrio de Nash não é a melhor solução para o jogo. Os jogadores poderiam conseguir melhores benefícios se ambos escolhessem não confessar. Contudo, seria tentador demais ficar calado quando confessar poderia livrá-los da pena.

Em termos da teoria dos jogos, diz-se que primeiro os dois jogadores possuem uma estratégia dominante, isto é, todas menos uma estratégia é estritamente dominada, segundo que o jogo é solucionado por dominância estrita e terceiro que o jogo termina em uma solução que é um equilíbrio de estratégia dominante (BORTOLOSSI; GARBAGIO; SARTINI, 2007).

Concorrência de Mercado

Na Tabela 2.5 apresenta-se uma situação em que dois jogadores concorrem no mesmo mercado. Ambos oferecem serviços similares e têm a opção de cobrar caro ou barato. Existem dois números dentro de cada quadrado e esses são os resultados que cada jogador recebe por sua estratégia. Tradicionalmente, o primeiro valor é quanto o jogador 1 recebe e o segundo valor, quanto o jogador 2 recebe.

Tabela 2.5: Concorrência de mercado.

		Jogador 2	
		Caro	Barato
Jogador 1	Caro	11, 10	2, 19
	Barato	20, 1	6, 5

Esse quadro pode representar os dois únicos dentistas de uma pequena cidade do interior e os números multiplicados por 1.000 u.m. os lucros ao final do mês. Há algum tempo, existia somente o jogador 1 na cidade e seus preços eram altos devido a falta de opções. Então, chega o jogador 2 e abre um consultório em frente ao do jogador 1. O jogador 2 agora deve definir quanto cobrar por seus serviços. Se ele se equiparar ao preço do concorrente, receberá um retorno de 10; o primeiro, por já estar estabelecido, fica com um retorno mais alto. O novo dentista

também tem a opção de cobrar um preço mais barato que o primeiro. Isso fará com que grande parte da clientela mude de dentista, e agora o lucro dele é bastante alto, enquanto o dentista inicial passa a viver com 2.000 u.m. mensais. Uma ação dessas não ficará sem reação, e o primeiro dentista pode também baixar seus preços altos logo no início, ou também os dois não entram em acordo e levantam seus preços juntos. Mas os dois são concorrentes e a motivação para qualquer um deles reduzir o preço é muito alta. O primeiro dentista pode resolver abaixar seus preços, atraído pela perspectiva de ter seus lucros quase dobrados, enquanto seu competidor fica com 1.000 u.m. por mês. O que ocorre nesse jogo é a dinâmica do dilema do prisioneiro. Pelo dilema do prisioneiro sugere-se que se tenha muito cuidado quando os concorrentes começam a baixar os preços. Sem referencial, corre-se o risco de ser forçado a uma guerra de preços (ZUGMAN, 2007).

O matemático e cientista político Robert Axelrod (1949-), investigando a cooperação na política internacional, durante o período da Guerra Fria, e baseando-se em resultados de simulações computacionais, propõe uma estratégia baseada na reciprocidade, segundo a qual o jogador deve cooperar no primeiro movimento e a partir de então fazer o que o outro jogador tiver feito na jogada precedente. A iteração do dilema do prisioneiro tem sido utilizada para analisar a evolução da cooperação em jogos experimentais, onde a probabilidade de cooperação ou deserção depende do histórico das interações. No caso do jogo ser repetido, os jogadores podem aprender a cooperar, ou seja, iniciar cooperando, copiar a última jogada do oponente, cooperar, retaliar, perdoar como metáfora para interação social (ajuda mútua versus exploração egoísta).

2.6.2 A forma extensiva

A forma extensiva de representação de um jogo é a maneira mais completa de se descrever um jogo. Um jogo representado nesta forma é especificado através de uma estrutura chamada árvore do jogo, que é um grafo direto com um número finito de nós N , conectados através de arcos que definem as ações dos jogadores. Esta forma de representação especifica todas as possíveis situações do jogo, o que cada jogador sabe e o que ele pode fazer na sua vez de jogar e, claro, os prêmios para cada jogador devido a cada combinação possível de movimentos.

Na forma extensiva, tenta-se representar jogos onde a ordem é importante. Os jogos são apresentados como árvores, onde cada vértice representa um ponto de decisão para um jogador. O jogador é especificado por um número listado no vértice. Os pagamentos são especificados na parte interior da árvore. A forma extensiva também pode representar jogos que se movem simultaneamente, podendo ser representado com uma lista tracejada ou um círculo no qual são desenhados dois diferentes vértices, isto é, os jogadores não sabem em qual ponto eles estão.

O jogo na forma extensiva descreve quem joga e quando, que ações estão disponíveis para cada jogador, quais são as informações em poder de um jogador quando é a sua vez de jogar, qual o resultado de uma série de ações seguidas pelos jogadores, e quais são os “payoffs” associados a cada resultado possível. Os jogos extensivos são muito utilizados em economia. Um jogo extensivo é um modelo de tomada de decisão interativa no qual cada jogador escolhe sua ação isoladamente (a cada jogada) conhecendo as ações anteriores (histórico de jogadas). O modelo consiste de um conjunto de jogadores J ; um conjunto de históricos de ações H , onde um histórico é uma sequência de ações $(a_k)_{k \in I}$ e $I \subseteq N$; uma função P , que retorna qual é o próximo jogador dado um histórico; e, para cada jogador i , uma relação de preferência (\succeq_i) sobre o conjunto de históricos terminais Z , onde um histórico $(a_k)_{k=1 \dots K} \in H$ é terminal se K é infinito ($k = \infty$) ou não existe a_{K+1} tal que a sequência $(a_k)_{k=1 \dots K+1} \in H$ (DE VASCONCELOS, 2003).

Definição 2: *Jogos Extensivos (“Extensive Games”)*

- um conjunto finito J (de jogadores);
- um conjunto H de sequências (finita ou infinita) que satisfaz três propriedades;
 - A sequência $\emptyset \in H$;
 - Se $(a_k)_{k=1 \dots K} \in H$ (onde $K \in H$ e $I \subseteq N$) e $\forall L < K$ então $(a_k)_{k=1 \dots L} \in H$;
 - Se uma sequência infinita $(a_k)_{k=1 \dots K}$ satisfaz $(a_k)_{k=1 \dots L}$ para todo inteiro positivo L então $(a_k)_{k=1 \dots K} \in H$;
- Uma função P que para cada histórico não-terminal (cada elemento de H/Z) retorna um elemento de J (P é a função jogador, $P(h)$ retorna qual é o próximo jogador);
- Para cada jogador $i \in J$ existe uma relação de preferência (\succeq_i) em Z .

Desta forma, cada jogador decide o seu plano de ação quando for a sua vez de jogar e a ordem dos acontecimentos encontra-se especificada.

2.6.2.1 informação perfeita e informação imperfeita

Os jogos podem ser de informação perfeita ou imperfeita, em função do tipo de informação com que contam os jogadores para tomar suas decisões. Um jogo tem informação perfeita se

cada jogador sabe exatamente o que ocorre cada vez que tem de tomar uma decisão. Se algum jogador não tem informação perfeita o jogo é de informação imperfeita (ANDRADE, 2002).

Um importante subconjunto dos jogos sequenciais consiste dos jogos de informação perfeita. Um jogo é de informação perfeita se todos os jogadores conhecem os movimentos prévios feitos por todos os outros jogadores. A maioria dos jogos estudados, na teoria dos jogos, na verdade, são de informação imperfeita, embora alguns jogos interessantes sejam de informação perfeita.

No xadrez, cada jogador possui conhecimento total do jogo na sua vez de jogar, ou seja, um enxadrista pode ver como seu oponente se movimentou na jogada anterior e, assim, sabe qual a situação em que se encontra na hora do seu movimento. Diz-se, então, que jogos do tipo xadrez são jogos de informação perfeita. No pôquer, um jogador não sabe quais as cartas de seus oponentes, assim, não sabe qual a estratégia adotada por eles e, consequentemente, não conhece qual o cenário do jogo na sua vez de jogar caracterizando um jogo de informação imperfeita. Outro jogo com informação imperfeita é o de combinação de moedas com moeda encoberta ou movimentos simultâneos.

O jogo “hawk-dove” é um jogo de informação imperfeita, enquanto o de combinação de moedas é de informação perfeita. Os termos apresentados seguem a seguinte definição:

- Um jogo é de informação perfeita se cada conjunto de informação contém um único nóculo de decisão. Caso isso não ocorra, é um jogo de informação imperfeita. Além disso, nos jogos com informação perfeita, a cada jogada todos os jogadores têm conhecimento das jogadas que já ocorreram, enquanto que nos jogos com informação imperfeita, o conhecimento é parcial (ANDRADE, 2002).

Sofisticando um pouco mais, o jogo “hawk-dove” é colocado na forma extensiva que descreve quem joga e quando, que ações estão disponíveis para cada jogador, quais são as informações em poder de um jogador quando é a sua vez de jogar, qual o resultado de um série de ações seguidas pelos jogadores e, quais são os ganhos associados a cada resultado possível, como mostrado na Figura 2.9.

Supondo que um dos animais seja o *dono* do território e que o outro seja um intruso que deseja conquistá-lo. O território pode até ser *bom* ou *ruim* (para a caça, por exemplo) e o valor do território depende de seu tipo. O valor de um território bom é oito, enquanto o de um território ruim é quatro. O dono do território conhece o seu valor mas o intruso não, sendo o dono do território escolhido aleatoriamente (com chances iguais para cada um). Para a árvore

que será apresentada a seguir, por exemplo, usa-se $W = 16$. Finalmente, a probabilidade do território ser ruim é igual a 0.6 e essa probabilidade é conhecida pelos participantes.

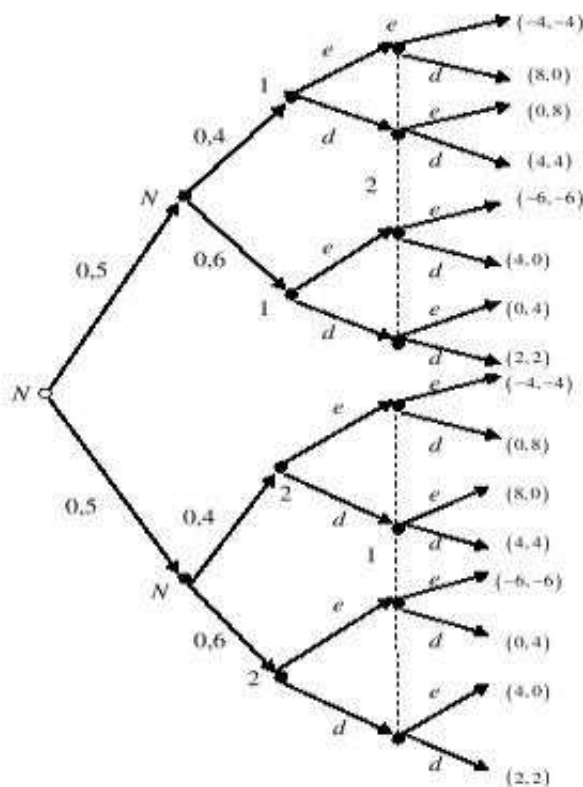


Figura 2.9: Jogo “hawk-dove” na forma extensiva.

2.7 Teoria dos Jogos Cooperativos

Em jogos cooperativos, o conjunto de ações possíveis está associado a grupos de jogadores (ou coalizões). A teoria dos jogos cooperativos apresenta maneiras sistemáticas de avaliar a situação e chegar a soluções ótimas (ou provar que não há solução ótima). Quando os participantes de um jogo decidem se juntar para cooperar em um certo negócio, o problema que aparece pode ser dividido entre todos. Se um ou mais jogadores perceberem que uma certa divisão proposta lhe é desvantajosa, eles podem decidir deixar a coalizão. Faria diferença se, no dilema do prisioneiro, os jogadores pudessem conversar antes de tomar suas decisões? Com certeza, sim. Ao invés de atuarem apenas por seus próprios interesses, eles escolheriam a opção com o melhor benefício para ambos, ou seja, nenhum deles confessaria. Esse fato reflete a principal diferença entre as teorias dos jogos cooperativos e não-cooperativos, que é a possibilidade dos jogadores atuarem em conjunto para melhorar seu desempenho.

2.7.1 Coalizões

No jogo cooperativo, os jogadores têm interesse em aumentar seus benefícios através de acordos, formando as chamadas *coalizões*. Matematicamente, para um jogo de n jogadores, uma coalizão é um subconjunto do conjunto de jogadores N , denominado S . Dentro de uma coalizão, os indivíduos atuam como uma única entidade coordenada. Assim, um jogador que pertence a uma coalizão não pode estabelecer acordos com outros jogadores que não esteja na coalizão. A grande coalizão, formada por todos os n jogadores, será denominada N . Existe, ainda, uma coalizão vazia, sem nenhum membro. Para um jogo com n jogadores é possível estabelecer 2^n coalizões. Considerando que sejam formadas m coalizões, representadas pelo conjunto $S = \{s_1, s_2, \dots, s_m\}$, três condições devem ser satisfeitas por esse conjunto, dadas por (FERGUSON, 2009):

$$S_j \neq \emptyset, j = 1, \dots, m \quad (2.3)$$

$$S_i \cap S_j = \emptyset, i \neq j \quad (2.4)$$

$$\cup S_j = N \quad (2.5)$$

Essas condições dizem que cada jogador pode pertencer apenas a uma coalizão não vazia das m existentes e não pode firmar acordos com jogadores fora da sua coalizão. Além disso, a união de todas as m coalizões forma a grande coalizão.

Para cada coalizão S , o maior benefício de ser obtido por seus membros, sem a ajuda de outros jogadores fora de S , é dada por sua função característica $v(S)$. Esse conceito foi introduzido por Von Neumann e Morgenstern, (VON NEUMANN; MORGENSTERN, 1944). Uma restrição a esta definição é que o valor da coalizão vazia seja zero, ou seja, $v(\emptyset) = 0$. Quando uma coalizão produz uma função característica em que configure o conceito de superaditividade (o jogador tem ganhos individuais maiores quando atua em uma coalizão do que quando atua isoladamente) diz-se que a função característica é uma imputação.

$$v(S \cup T) \geq v(S) + v(T); S, T \subset N; S \cap T = \emptyset \quad (2.6)$$

A superaditividade indica que o total da função de benefícios para a grande coalizão é coletivamente racional. Havendo superaditividade, e, portanto, coletividade racional, a grande coalizão se formará no final do jogo.

Porém, a divisão dos benefícios obtidos com a coalizão, representada pelo vetor de benefícios $\bar{x} = (x_1, x_2, \dots, x_n)$ não é trivial. É necessário que esse vetor satisfaça a certas exigências, dadas

por:

$$v(N) = \sum_{i=1}^n x_i \quad (\text{racionalidade coletiva}) \quad (2.7)$$

$$x_i \geq v_i \quad \forall i \in N \quad (\text{racionalidade individual}) \quad (2.8)$$

Se \bar{x} satisfaz a essas duas exigências, então, \bar{x} é chamado imputação. A racionalidade coletiva indica que a soma dos benefícios recebidos por cada membro da coalizão seja igual ao benefício total da coalizão. A racionalidade individual indica que o benefício de cada jogador deve ser, sempre, maior ou igual ao benefício que ele obteria atuando individualmente.

Em coalizões de votação, por exemplo, um jogador possui interesses específicos e se comporta como um todo. Enquanto cada votante pode ser visto como um jogador, eles se fortalecem ao formarem coalizões, votando em bloco. Existem, então, dois jogos, um dentro da coalizão, para escolher a decisão a ser tomada pelo grupo, e um entre a coalizão e os outros participantes do fórum.

2.7.2 Conceitos de solução de jogos cooperativos

Os jogos cooperativos descrevem os ganhos que podem ser obtidos através da formação de coalizões. Contudo, um jogador só irá aderir a uma coalizão se ele espera aumentar seus benefícios com isso. A seguir, apresenta-se alguns conceitos de solução de jogos cooperativos.

2.7.2.1 core

Em um jogo cooperativo, os jogadores podem formar coalizões para tentar maximizar conjuntamente seus benefícios. Além disso, em jogos cooperativos é possível dividir os benefícios obtidos com a cooperação através de pagamentos laterais, ou seja, transferências entre os membros da coalizão que mudam os benefícios. A cooperação é o ingrediente básico do modelo matemático de um jogo cooperativo, e é chamado de uma coalizão.

O *core* é um conceito de solução de jogos cooperativos introduzido por (GILLIES, 1953), e trata-se de um conjunto de imputações³ que atendem às definições de racionalidade individual, coletiva ou coalizacional. Define-se como racionalidade coalizacional a existência de racionalidade para todas as coalizões formadas no jogo.

$$\sum x_i \geq v(S); \forall i \in S; S \subset N \quad (2.9)$$

³Vetor de benefícios.

O core é, ainda, o conjunto de imputações que atende a um conjunto de restrições. Esse é o conceito mais simples dos conceitos de solução cooperativa. Contudo, existem muitos jogos sem core, ou com core vazio, basicamente por não atenderem às exigências de racionalidade coalizacional. Quando o jogo possui um core vazio, ao menos uma coalizão ficará insatisfeita.

Shapley e Shubik notaram que um jogo com core não vazio é sociologicamente neutro, ou seja, todas as demandas de todas as coalizões podem ser satisfeitas e não existe a necessidade de resolver conflitos. Também não é desejável um core com muitos elementos, pois apresentaria uma previsão imprecisa. Por outro lado, em um jogo sem core, as coalizões são muito exigentes para que sejam atendidas todas as demandas coalizacionais (CONTRERAS, 1997).

O conceito de core é útil como medida de estabilidade. Contudo, como conceito de solução, o core apresenta um conjunto de soluções sem distinção de quais são as melhores. O core fornece um intervalo de respostas que respeita as racionalidades dos agentes. Porém, o fato de existirem jogos que possuem core vazio ou mesmo de não possuírem core, acaba não sendo resolvido por este conceito. O conceito matemático do core fornece uma região na qual qualquer coalizão que seja formada atende ao objetivo do jogo, mas não se sabe qual das soluções é a melhor. Neste caso, se existe um core para o jogo, existe um conjunto de imputações que atende a todas as restrições das coalizões, mas não se sabe qual a melhor solução. Além disso, o core fornece um conjunto de soluções que são matematicamente encontradas, ou seja, se uma solução está dentro do core, ela com certeza está próxima do ótimo global, o que representa uma vantagem sobre as heurísticas, nas quais pode-se chegar a um ótimo local sem ter conhecimento disso. Usando-se a notação padrão da teoria dos jogos cooperativos, define-se a representação quantitativa como um “outcome” do jogador no final do jogo, que também é chamado de “pay-off” (CONTRERAS, 1997).

2.7.2.2 valor Shapley

Caso o core seja muito extenso ou não exista, é possível utilizar outras soluções para o jogo, tais como o *valor Shapley*. O valor Shapley provê uma única solução dada por:

$$x_i = \sum_{\forall S | i \notin S} P_n(S) [v(S \cup \{i\}) - v(S)]; \quad (2.10)$$

em que $P_n(S) = (|S|!(n - |S| - 1)!)/n!$ e $|S|$ é o número de jogadores em S . O valor Shapley pode ser considerado como uma média ponderada das contribuições marginais de um membro para todas as possíveis coalizões em que ele pode participar. Outra interpretação similar do valor Shapley pode ser vista como um somatório sobre todos os subconjuntos da coalizão S da

qual o jogador i é um membro, com um coeficiente multiplicando a diferença entre o valor de qualquer subconjunto S e o valor daquela coalizão sem o jogador i . Ele então quantifica para uma soma ponderada das adições incrementais feitas pelo jogador i , para todas as coalizões nas quais ele é um membro, que são subconjuntos da coalizão que ele pertenceu ultimamente. O valor Shapley não pertence ao core. O valor Shapley de um jogo u é o vetor de valores que satisfaz (CONTRERAS, 1997):

- o conjunto de jogadores recebe todos os possíveis recursos: $\sum_{i \in P} x_i[u] = v(P)$, onde P é a coalizão de todos os jogadores;
- se simulado, então $v(C) - v(C - i) = x_i$, onde C é uma determinada coalizão;
- o valor assinalado para o jogador i não depende da posição do jogador no conjunto de jogadores;
- e se u e v são dois jogos, $x_i[u + v] = x_i[u] + x_i[v]$.

O valor Shapley fornece um único vetor de pagamentos, sendo um conceito que calcula uma divisão justa dos benefícios comuns entre os membros de uma coalizão. Um dos principais atrativos do valor Shapley é o fato de fornecer um único ponto para qualquer jogo em estrutura de coalizão. Contudo, ele não reflete as possibilidades reais de cada jogador no jogo descentralizado (CONTRERAS, 1997).

2.7.2.3 valor Shapley bilateral

Para evitar problemas de complexidade exponencial no cálculo do valor Shapley é que foi introduzido o *valor Shapley bilateral* (BSV), que pode ser adotado para um processo de negociação entre agentes racionais. Esse valor se define para duas coalizões dentro de um determinado conjunto de agentes, que se unem para formar uma coalizão maior. Numa estrutura de negociação baseada no valor Shapley, o BSV é introduzido para o processo de negociação bilateral e completamente descentralizado entre agentes racionais. Para que $S \subseteq P(A)$ seja uma estrutura de coalizão sobre o conjunto de jogadores $A = \{a_1, \dots, a_m\}$, tem-se $C = C_i \cup C_j \subseteq A$ e $C_i \cap C_j = \emptyset$; onde $P(A)$ é a coalizão de todos os jogadores, ou *grande coalizão*. Portanto, C é uma coalizão (bilateral) de coalizões disjuntas (n -jogadores) de C_i e C_j ($n \geq 0$). O BSV para a coalizão C_i , numa coalizão bilateral C , é definido por:

$$\varphi_C(C_i) = 1/2v(C_i) + 1/2(v(C) - v(C_j)). \quad (2.11)$$

Ambas coalizões C_i e C_j são adequadas para formar a coalizão C , se $v(C_i) \leq \varphi c(C_i)$ e $v(C_j) \leq \varphi c(C_j)$. De fato, um jogo cooperativo superaditivado é jogado entre C_i e C_j . Das equações acima, é possível perceber que os resultados conduzem a metade das contribuições locais de cada coalizão, e a outra metade obtida do trabalho cooperativo com a outra coalizão. O segundo termo da expressão BSV reflete a força de cada jogador baseado sobre suas contribuições. Então, dois jogadores formam uma coalizão se ambos obtêm mais valor do que atuando sozinho. O processo de formação de coalizão continua se com outros jogadores aumentar-se seus valores. Se o processo continua até o fim, a grande coalizão (todos os jogadores) formam uma equipe singular, desde que retorne benefício para todos (CONTRERAS, 1997).

2.7.2.4 kernel

O *kernel* é outro conceito de solução para jogos cooperativos. As configurações coalizacionais *kernel* são estáveis quando existir um equilíbrio entre pares de agentes individuais que encontram-se na mesma coalizão. Dois agentes A e B em uma coalizão C estão em equilíbrio se eles não puderem sobrecarregar um outro agente de C , com a coalizão comum deles. O agente A pode sobrecarregar B se A é mais potente do que B, onde a força refere-se ao potencial do agente A para exigir com sucesso uma parte do ganho do agente B. Em cada estágio do processo de formação de coalizão, os agentes estão em uma configuração coalizacional. Isto é, os agentes são organizados num conjunto de coalizões $C = \{C_i\}$. Durante a formação de coalizão, os agentes podem usar o conceito de solução *kernel* para objetar para a distribuição dos ganhos em que são atacados para suas configurações coalizacionais. As objeções que os agentes podem fazer são baseadas no conceito de *excesso*.

A teoria do *kernel* como um conceito de solução de jogos cooperativos, especificamente da teoria do excesso, foi introduzida em 1965 por Morton e Maschler (MORTON; MASCHLER, 1965). É uma teoria baseada na comparação entre todos os pares de jogadores de um jogo que utiliza o excesso como medida da força relativa que existe entre os agentes no processo formador de coalizões, entendendo-se que a maior força de um jogador é o maior excesso que ele possui.

Excesso

O excesso de uma coalizão é definido como o montante pelo qual uma coalizão excede seu pagamento preferencial, ou seja, o quanto seus membros recebem além do que esperavam receber inicialmente ao atuar em conjunto. A idéia do *kernel* é equiparar as forças relativas que existem entre os agentes que se coalizam, de maneira a estabelecer o vetor de pagamentos \bar{x} de maneira justa e sem violar racionalidades. Esta equiparidade é definida como *equilíbrio*

(CONTRERAS, 1997).

O excesso de uma coalizão C com relação a uma configuração coalizacional C é definido como:

$$e(C) = v(C) - \sum_{A_i \in C} u^i \quad (2.12)$$

em que u^i é o pagamento do agente A_i e $v(C)$ é o valor da função de coalizão C . O número de excessos é uma importante propriedade do conceito de solução kernel. Agentes usam os excessos como uma medida de suas *forças relativas*. Considerando o valor do excesso, o agente deve buscar o maior excesso que ele pode alcançar em uma coalizão. O máximo que ele pode alcançar é definido como *excedente*.

Excedente e Força Relativa

O máximo excedente S_{AB} do agente A sobre o agente B com relação a uma configuração é definido por:

$$S_{AB} = \max_{C|A \in C, B \notin C} e(C) \quad (2.13)$$

em que $e(C)$ são os excessos de todas as coalizões que incluem A e excluem B, e as coalizões C não estão nas configurações coalizacionais correntes. O agente A sobrepõe o agente B se $S_{AB} > S_{BA}$ e $u^B > v(B)$, em que $v(B)$ é o valor coalizacional do agente B numa coalizão de um único agente (CONTRERAS, 1997).

3 O Ambiente de Desenvolvimento e a Linguagem Utilizados

3.1 Introdução

Seria muito útil para um usuário de computador, seja ele desenvolvedor de software ou usuário final, se ele pudesse ter suas ferramentas de trabalho integradas em uma plataforma, de modo que ele também pudesse mover livremente seus dados entre aplicações.

Para o desenvolvedor de software, seria interessante se ele tivesse todas as suas ferramentas de desenvolvimento integradas em uma plataforma na qual ele possa trabalhar sem problemas. Dessa forma, ele ia se preocupar apenas com o projeto em desenvolvimento e não com a integração e incompatibilidade dos arquivos gerados pelas suas ferramentas.

O usuário final não quer perder seu tempo gerenciando o fluxo de arquivos e dados entre suas ferramentas para conseguir realizar o seu trabalho. Eles precisam de ferramentas que integrem com as demais ferramentas, que consigam resultados melhores e que possuem interface de usuário semelhantes para fazerem tarefas similares.

Uma solução para esse problema seria um nível de integração que combinem ferramentas de desenvolvimento separadas em um conjunto bem determinado; simples o suficiente para que tais ferramentas possam ser movidas entre plataformas. A plataforma deve ser aberta, de modo que usuários possam selecionar as suas melhores ferramentas, as quais eles já confiam e estão adaptados.

O usuário pode, se quiser, usar o bloco de notas (*notepad*), digitar um algoritmo nele e compilar num arquivo que pode ser executado no computador, como um executável, mas ele precisaria compilar usando linhas de comando no *prompt* de comando *shell*, do Linux, por exemplo, o que fica mais propenso a erros, tanto de sintaxe como de lógica.

De acordo com Hwang (1999), citado por (GONÇALVES; CANESIN, 2002), um dos alicerces de um país para a sustentação de seu desenvolvimento é a educação. Nas últimas

décadas, pesquisadores vêm tentando o refinamento de conceitos, técnicas e o desenvolvimento de novas ferramentas para auxiliar o aprimoramento do processo educacional em diversos níveis (HWANG, 1999). A forma tradicional de ensino apresentada nas salas de aula pode ser complementada com o emprego de sistemas de ajuda interativos baseados em computadores, envolvendo tanto o ensino quanto o treinamento (DEBEBE; RAJAGOPALAN, 1995). A evolução dos computadores, em conjunto com a disseminação cada vez maior da Internet (rede internacional de computadores), tornou fácil o acesso à informação, de quase todos os lugares, a qualquer hora e respeitando o ritmo do usuário. O advento das tecnologias de hipermídia acompanhado por hipertextos, simulações interativas, vídeos e sons, originaram um mecanismo novo e efetivo de entrega de informações com o poder de criar laboratórios virtuais e salas de aula virtuais (BENGU; SWART, 1996).

Logo, a inclusão da informática ao método tradicional de ensino, permite preparar os alunos do curso de engenharia para o mercado de trabalho atual, o qual é marcado por inovações tecnológicas e totalmente informatizado. Baseado nesta necessidade educacional, tem-se a motivação para o desenvolvimento de uma ferramenta computacional que tenha a finalidade de fortalecer o processo de aprendizagem do aluno, estimulado e orientado pelo professor.

Nestes últimos anos, diversos pesquisadores vem buscando o desenvolvimento de ferramentas de auxílio ao ensino que possam ser executadas em ambientes distribuídos, que sejam reutilizáveis e proporcionem um elevado nível de interação. Ou seja, uma ferramenta que permita um ensino efetivo “on-line” através da WWW. Neste sentido, uma das principais plataformas utilizadas, por suas vantagens intrínsecas, é a Java (WIE, 1998).

A plataforma Java foi introduzida pela Sun Microsystems Inc. e apresenta informações tecnológicas de arquiteturas para integração de diferentes plataformas. Alguns pesquisadores têm concentrado seus esforços no desenvolvimento de sistemas do tipo CAL (“Computer Aided Learning”), elaborados para funcionarem como um ambiente individual de aprendizado, onde o foco dos trabalhos se concentra nas interações entre o conjunto composto pelos estudantes e o computador.

Neste contexto, a linguagem de programação Java, na forma de *Java applets*, proporciona uma plataforma aberta, distribuída, e expansível para um sistema de auxílio ao ensino.

Por outro lado as principais características de Java que são: modelo seguro, acesso à Web, arquitetura neutra, desenvolvimento rápido, portabilidade e robustez, a tornam uma poderosa plataforma para o controle industrial, bem como, para o desenvolvimento de sistemas de controle em geral.

As *Java applets* são escritas na forma de programação orientada a objetos (POO) e podem ser facilmente incluídas em um documento HTML, em conjunto com outros elementos multimídias como figuras, vídeos e sons, permitindo uma fácil configuração de materiais de ensino multimídia dinâmicos (GONÇALVES; CANESIN, 2002).

Sendo as *Java applets* desenvolvidas para operar em ambientes distribuídos baseados na WWW e nas tecnologias da Internet, qualquer computador habilitado com Internet torna-se uma estação potencial de *ensino* ou, *controle*, introduzindo-se os novos conceitos de *professores virtuais* e *engenheiros virtuais*, respectivamente. As principais vantagens do sistema de ensino baseado na WWW são a independência de tempo e espaço, permitindo um grande número de possíveis atendimentos simultâneos, além de possuir uma interface simples e familiar, devido à metodologia estar baseada na utilização de um navegador de rede (“web browser”).

Um IDE (“Integrated Development Environment”) consiste em um software que contém um conjunto de funcionalidades embutidas, cuja finalidade é prover um modo mais fácil e interativo de construir e manipular os programas.

Geralmente os IDEs facilitam a técnica de RAD (“Rapid Application Development”), que visa a maior produtividade dos desenvolvedores. As características e ferramentas mais comuns encontradas nos IDEs são:

- *Editor de texto* - edita o código fonte do programa escrito na(s) linguagem(ns) suportada(s) pela IDE;
- *Compilador* - compila o código-fonte do programa, editado numa linguagem específica e a transforma em linguagem de máquina;
- *Depurador* - ferramenta especialmente feita para encontrar e corrigir erros;
- *Modelagem* - criação do modelo de classes, objetos, interfaces, associações e interações;
- *Geração de código* - característica mais explorada em ferramentas de caso de uso, a geração de código também é encontrada nas IDEs;
- *Distribuição* - auxilia no processo de criação do instalador do software, seja por CD, DVD ou internet;
- *Testes automatizados* - realiza testes no software de forma automatizada, com base em “scripts” ou programas de testes previamente especificados, gerando um relatório dos mesmos, assim auxiliando na análise do impacto das alterações no código-fonte;

- *Refatoração* - consiste na melhoria constante do código-fonte do software, seja na construção de código mais otimizado, mais limpo e/ou com melhor entendimento pelos envolvidos no desenvolvimento do software.

Assim, surge o ambiente de desenvolvimento de software que é distribuído como um componente principal (o Eclipse SDK) com o ambiente básico, com os recursos fundamentais para desenvolvimento padrão, mais diversos componentes adicionais distribuídos na forma de *plug-ins*. Estes *plug-ins* estendem a funcionalidade do ambiente e acrescentam suporte a recursos e tecnologias específicas. Diversos *plug-ins* são desenvolvidos como subprojetos da Fundação Eclipse em Eclipse.org, mas existe também uma enorme variedade de *plug-ins* desenvolvidos por terceiros, tanto livres quanto comerciais.

3.2 O Ambiente Eclipse

O Eclipse é um ambiente integrado de desenvolvimento de software, inicialmente desenvolvido pela IBM, que gastou milhões de dólares no seu desenvolvimento antes de ser transformado nessa ferramenta de código aberto para um consórcio, chamado Eclipse.org. Inicialmente incluiu a Borland, IBM, Merant, QNX Software Systems, Rational Software, Red Hat, SuSE, TogetherSoft e Webgain (ECLIPSE, 2007).

Outras companhias como Hewlett Packard, Fujitsu, Oracle e Sybase se uniram desde então. Entretanto, a IBM continua a frente do desenvolvimento do Eclipse por sua subsidiária, Object Technologies International (OTI), com a equipe que iniciou seu desenvolvimento. OTI é uma empresa que desenvolve ferramentas de programação orientada a objetos, com uma história que nos remete a 1988, quando a escolha da linguagens de orientação a objetos era Smalltalk. OTI foi adquirida pela IBM em 1996, onde a IBM agariou força em seu ambiente de desenvolvimento de orientação a objetos chamado *Visual Age* (ECLIPSE, 2007).

No Brasil, uma das ferramentas de desenvolvimento Java mais consagradas e usadas pelos desenvolvedores é mesmo a arquitetura Eclipse usada na construção de ambientes integrados de desenvolvimento (IDEs) de forma que esta possa ser usada na criação de aplicações diversas como páginas Web, programas Java, programas em C++ e Enterprise JavaBeans. Existe também um grande interesse na plataforma por parte de empresas de grande porte; um sinal claro deste fato é o número sempre crescente de empresas que fazem parte do consórcio. A licença utilizada é a CPL (“Common Public License”), que permite a livre distribuição do código fonte e trabalhos derivados do código original (CHRISTOPH, 2004).

3.2.1 A arquitetura Eclipse

O principal projeto desenvolvido pela comunidade do Eclipse é a “Eclipse Platform”, ou Plataforma Eclipse, que é a base para construir todas as outras ferramentas, como por exemplo a própria IDE Java. Eclipse, tecnicamente, é uma junção de diversos *plug-ins*, que são gerenciados por um núcleo interno sobre o qual todos os outros componentes são executados e, também, todos os serviços genéricos e independentes da linguagem usada.

Eclipse é uma plataforma que foi projetada para desenvolvimento Web e construção de aplicações integradas. Pelo projeto, a própria plataforma fornece liberdade de funcionalidade para o usuário final. O valor da plataforma é o que ela incentiva: desenvolvimento rápido de características integradas, tomando como base o modelo de arquitetura de *plug-ins*. Eclipse fornece também um modelo comum de interface com o usuário para trabalhar com as ferramentas. É projetado para funcionar em múltiplos sistemas operacionais, fornecendo integração robusta com cada um deles.

A plataforma Eclipse define uma arquitetura aberta, de modo que cada equipe de desenvolvimento de *plug-ins* possa focalizar em sua área de especialidade, onde especialistas em repositório constroem os bancos de dados e os especialistas em usabilidade constroem as ferramentas do usuário final. Se a plataforma for bem projetada, novas características significantes e níveis de integração podem ser adicionados sem impacto para outras ferramentas.

Os desenvolvedores de *plug-ins* se beneficiam bastante com esta arquitetura. Eles podem também se dedicar a uma tarefa específica em vez de se preocupar com problemas de integração. A plataforma controla a complexidade de diferentes ambientes “runtime”, tais como os diferentes sistemas operacionais ou ambientes servidores de “workgroup”.

A plataforma Eclipse é estruturada como subsistemas (Figura 3.1), os quais são implementados em um ou mais *plug-ins*. Além da pequena “runtime”, a plataforma do Eclipse consiste do “workbench” (bancada de trabalho), “workspace” (gerência de recurso), “help” (sistema de ajuda) e um conjunto de outros componentes.

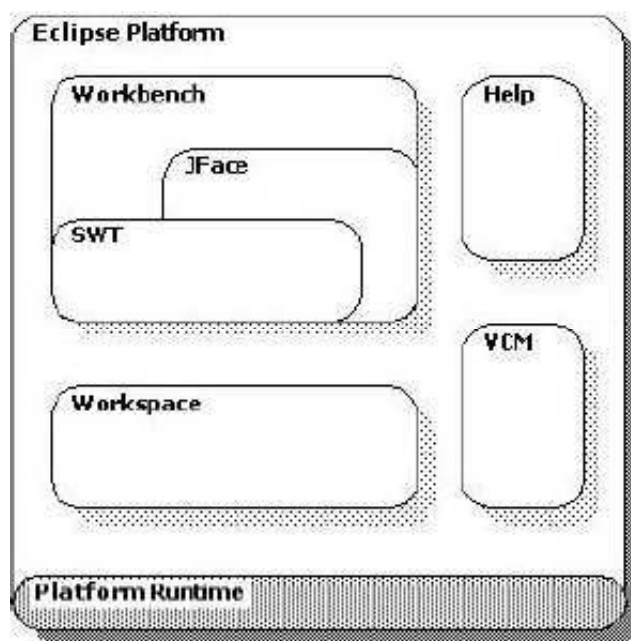


Figura 3.1: Arquitetura da Plataforma Eclipse [Extraído de Object Technology International, Inc., 2003].

3.2.1.1 “runtime”

O “runtime” é o *motor principal* sobre o qual todos os outros componentes são executados, lidando apenas com gerenciamento de *plug-ins*. O trabalho inicial da “runtime” é descobrir quais *plug-ins* estão disponíveis no diretório de *plug-ins* do Eclipse. Cada *plug-in* tem um arquivo XML, que lista as conexões que o *plug-in* exige. Essa extensão aponta para *plug-ins* que são requeridos. Por haver um número de *plug-ins* potencialmente grande, muitos não são carregados até que eles sejam requeridos de fato, minimizando assim tempo de inicialização e recursos do sistema.

No núcleo do Eclipse está uma arquitetura para a descoberta dinâmica dos *plug-ins*. A plataforma cuida do gerenciamento das funções do ambiente e fornece um modelo padrão de navegação do usuário. Cada *plug-in* pode então focalizar em fazer um número pequeno de tarefas com escopo bem definido.

O “runtime” usa um modelo de extensão para permitir que os desenvolvedores de *plug-ins* adicionem suporte para tipos de arquivos adicionais e instalações personalizadas, tais como servidores Web, servidores de “workgroup” e repositórios. Os artefatos para cada ferramenta, tal como arquivos e outros dados, são coordenados por um modelo de recurso comum da plataforma.

3.2.1.2 “workspace” - a gerência de recurso

O “workspace” gerencia diretórios e arquivos e é responsável por administrar os recursos do usuário que são organizados em um ou mais projetos. Cada projeto corresponde a um subdiretório do diretório de “workspace” do Eclipse. Cada projeto pode conter arquivos e subdiretórios. Normalmente cada diretório corresponde a um subdiretório do diretório do projeto, mas um subdiretório também pode ser unido a um diretório em qualquer lugar do sistema.

Ao iniciar o Eclipse pela primeira vez é apresentada a seguinte caixa de diálogo (Figura 3.2).

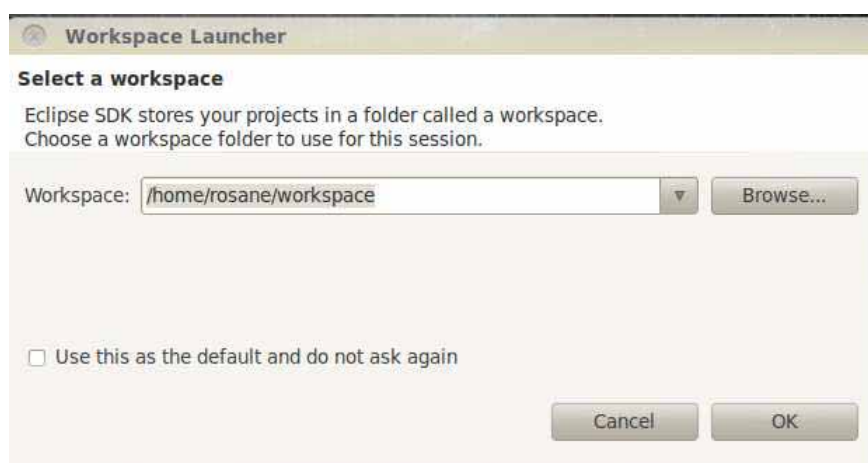


Figura 3.2: Gerência de recursos.

É possível alterar o espaço de trabalho, clicando em *Browser*, ou aceitando o local proposto pelo programa. Se não desejar mais essa caixa de diálogo na inicialização do Eclipse, basta marcar a caixa de checagem *Use this as the default and do not ask again*, antes de confirmar.

3.2.1.3 “workbench” - a bancada de trabalho

O “workbench” (Figura 3.3) é o *coração* da interface gráfica da IDE, que além de exibir menus e caixas de ferramentas, é organizado em perspectivas que contém visões e editores. Uma das características notáveis do “workbench”, diferente da maioria das aplicações Java, é a “look and feel” que emula o ambiente gráfico nativo. Isso ocorre porque ele foi construído usando “Standard Widget Toolkit” (SWT) e JFace, do próprio Eclipse, um “toolkit” de interface de usuário construída em cima do SWT. Diferente dos padrões gráficos existentes, o SWT mapeia diretamente os gráficos nativos do sistema operacional.

SWT é um dos aspectos mais controversos do Eclipse, porque o mesmo deve ser portado

para cada plataforma que suporta Eclipse. Isso não chega a ser um problema, porque SWT já foi portado às plataformas mais populares (incluindo Windows, Linux/Motif, Linux/GTK2, Solaris, QNX, AIX, HP-UX, e MacOS X).

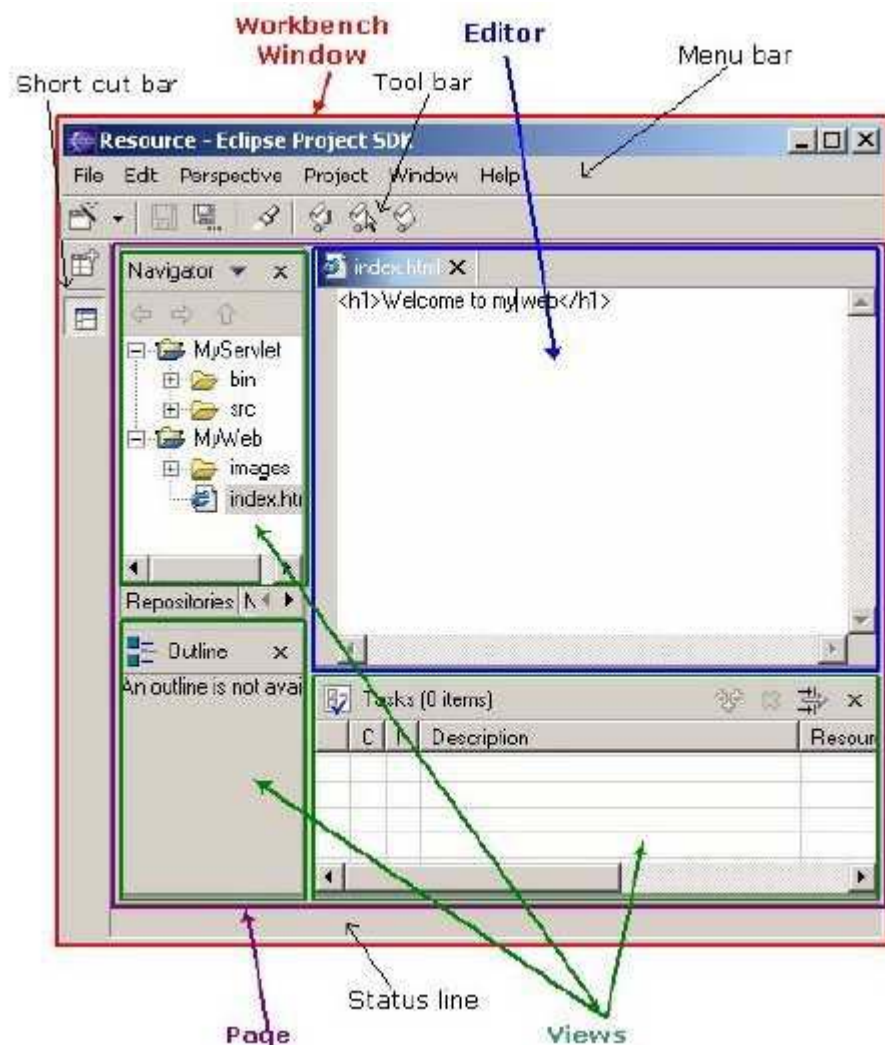


Figura 3.3: Bancada de trabalho [Extraído de Eclipse Foundation, Inc., 2000].

As barras de título, ferramentas, perspectivas e status tendem a ficar no mesmo local de uma perspectiva para outra. É possível personalizar o aparecimento e conteúdo das barras. A seguir, apresentam-se os detalhes do “workbench”, conforme apresentado na Figura 3.3.

- *Barra de Título* (“Title bar”): A barra de título sempre exibe o nome da perspectiva atual.
- *Barra de Menu* (“Menu bar”): Esta é a barra de menu global do Eclipse, que lhe permite fazer todas as tarefas gerais. As opções disponíveis em cada menu também mudarão dependendo do que estiver selecionado.

- *Barra de Ferramentas* (“Tool bar”): Esta é a barra de ferramentas global do Eclipse que também lhe permite executar tarefas, gerais e específicas para itens selecionados.
- *Barra de Status* (“Status bar”): Esta linha exibe tipicamente informação sobre o item selecionado, embora possa conter informações adicionais que dependem da tarefa que você está executando atualmente. Por exemplo, quando você estiver em um editor de texto, a barra de status poderá mostrar sua posição atual dentro do documento.
- *Painel do Editor* (“Panel Editor”): É exatamente o que o seu nome diz: é uma ferramenta para editar documentos. Esses documentos podem ser fontes de programas como configurações de sistemas.
- *Vistas* (“Views”): Mostram grupos de objetos relacionados.
- *Barra de atalho* (“Short cut bar”): É para abreviar o acesso aos componentes mais rapidamente.
- *Página* (“Page”): Lista a página da tarefa corrente.

Os “frameworks” que compõem a Plataforma Eclipse são o sistema de ajuda, para a ajuda ao usuário, a gerência de versão e de configuração para controle de versões e histórico de mudanças em recursos, o depurador para depuração genérica e o atualizador para atualizações automáticas.

O modelo de “workbench” comum integra as ferramentas do ponto de vista do usuário final. Novas ferramentas desenvolvidas podem ser adicionadas no “workbench” usando ligações bem definidas chamadas pontos de extensão.

3.2.1.4 “help” - o sistema de ajuda

Como a própria Plataforma do Eclipse, o componente de ajuda é um sistema de documentação extensível. Ferramentas podem adicionar documentação em formato HTML e, usando XML, define-se uma estrutura de navegação. Refletindo o modo como *plug-ins* se conectam com outros *plug-ins*, ferramentas de documentação podem inserir tópicos em uma árvore de tópico pré-existente.

3.2.1.5 CVS - a gerência de versões

O *plug-in* “team support” facilita o uso do controle da versão (ou configuração administrativa) do sistema para administrar os recursos no projeto do usuário e define a área de trabalho

necessária para salvar e recuperar um repositório. A Plataforma Eclipse também inclui um cliente para um Sistema de Versões Simultâneas (“Concurrent Versions System” - CVS).

3.2.1.6 multiplataforma, linguagens de programação e idioma

Embora a plataforma Eclipse seja escrita em linguagem Java e seu uso mais popular é como um IDE Java, o Eclipse é neutro em linguagem de desenvolvimento. O que o torna mais atrativo para desenvolvedores Java é devido aos seus inúmeros *plug-ins* voltados para essa linguagem, mas com *plug-ins* adicionais, pode-se programar em outras linguagens como C/C++, Cobol.

Eclipse também é neutro com respeito ao idioma. Da mesma forma que pode ser dada a capacidade de programar em outra linguagem de programação, com *plug-ins* também é possível alterar o idioma nativo. A IBM doou um pacote de idiomas que suporta o chinês, francês, alemão, italiano, japonês, coreano, português (brasileiro) e espanhol. No mesmo site da plataforma Eclipse é possível baixar o pacote de idiomas do Eclipse.

3.2.2 O projeto Eclipse@Rio

O projeto Eclipse@Rio, desenvolvido pelo Laboratório Teccomm da Pontifícia Universidade Católica do Rio de Janeiro, foi um dos vencedores do concurso “Eclipse Innovation Grants”, realizado pela IBM no final de 2002. Este projeto visa ajudar a disseminar a plataforma Eclipse a uma comunidade mais ampla de profissionais e pesquisadores. Essa disseminação é feita através de palestras e “workshops” gratuitos, a adoção da ferramenta em disciplinas obrigatórias e eletivas da graduação e pós-graduação e desenvolvimento de *plug-ins* gratuitos para a plataforma Eclipse (CHRISTOPH, 2004). As palestras e “workshops” servem como ponto de encontro para que profissionais interessados venham a conhecer ou aprimorar seus conhecimentos na plataforma Eclipse; já a adoção da plataforma nas disciplinas visa familiarizar o aluno com uma ferramenta largamente utilizada no mercado de trabalho e acadêmico. O desenvolvimento de *plug-ins* busca a criação de um conjunto maior e mais detalhado de ferramentas para a plataforma. Todos os *plug-ins* são software livres, seus códigos fontes são abertos.

3.3 A Linguagem de Programação Java

3.3.1 Histórico

Com a revolução dos microprocessadores no final da década de 80, a Sun Microsystems acreditou no desenvolvimeneto de dispositivos inteligentes destinados ao consumidor final. Desta forma, financiou, em 1991, pesquisa interna intitulada “Green”, resultando no desenvolvimento de uma linguagem de programação baseada em C e C++ que pudesse controlar tais dispositivos. Seu criador, James Gosling, a batizou como Oak (vocábulo do idioma inglês que significa carvalho no português), devido a uma árvore existente em frente a janela de seu escritório. Descobriu-se mais tarde que já havia uma linguagem de computador chamada Oak e o nome foi mudado para *Java*. Há uma certa curiosidade a respeito do nome dado a essa linguagem de programação. Java é o nome de uma ilha do Pacífico, onde se produz uma certa variedade de café. A inspiração bateu à equipe de desenvolvimento ao saborear esse café em uma lanchonete local (MAGALHÃES; LEITE, 2003)

O projeto “Green” atravessava dificuldades devido ao fato do mercado para dispositivos eletrônicos inteligentes não estar se desenvolvendo na velocidade esperada pela Sun. No ano de 1993 a WWW estava explodindo em popularidade e o pessoal da Sun viu o potencial de se empregar a linguagem Java para colocar conteúdo dinâmico nas páginas Web.

Em maio de 1995, a Sun anunciou a linguagem Java formalmente em uma conferência importante. Normalmente, um evento como esse não teria atraído muita atenção. Entretanto, Java gerou interesse imediato na comunidade comercial por causa do fenomenal interesse pela WWW. Java é agora utilizada para criar páginas Web com conteúdo interativo e dinâmico, para desenvolver aplicativos corporativos de larga escala, para aprimorar a funcionalidade de servidores da Web (os computadores que fornecem o conteúdo que se vê nos navegadores), fornecer aplicativos para dispositivos destinados ao consumidor final (como telefones celulares, “pagers” e PDAs) e assim por diante (DEITEL; DEITEL, 2005). Nos anos 90, a linguagem se popularizou, consistindo durante um bom tempo na única maneira de inserir conteúdo dinâmico em páginas Web.

Com o passar do tempo outras ferramentas surgiram, como JavaScript e Flash, para colocação de conteúdo dinâmico em páginas. A utilização da linguagem Java passou a sofrer restrições devido a velocidade reduzida de execução e a falta de padronização das máquinas virtuais existentes nos navegadores. Estes problemas se relacionam especificamente com a utilização de *applets*. A Sun, precebendo a forte concorrência das novas tecnologias, passou a dar ênfase na utilização de Java dentro dos servidores Web. Desta forma, no final da década de 90, a

linguagem tomou novo impulso, passando a ser utilizada dentro dos servidores para produzir páginas HTML dinamicamente. A utilização de *servlets* e JSP se popularizou rapidamente tornando-se fortes concorrentes do padrão ASP da Microsoft.

A utilização da linguagem Java na construção de *applets* serviu para popularizá-la embora, na maior parte das vezes, tenha sido empregada em aplicações amadoras. O público que utilizava a linguagem consistia de jovens estudantes procurando acrescentar algum detalhe em suas páginas Web que as diferenciasse das demais. Com a utilização em servidores Web a linguagem atinge sua maturidade, servindo agora como ferramenta séria de desenvolvimento para o mercado corporativo.

As *applets*, embora em menor número, continuaram sendo empregadas para aplicações específicas como: aplicações bancárias, gráficos de bolsas de valores e aplicações utilizando mapas em páginas Web. Nenhuma outra técnica obteria os resultados de uma linguagem de programação completa como Java.

Como as *applets* são normalmente aplicações muito simples, mascaram o poder da linguagem. Por ser totalmente orientada a objetos permite a modelagem e implementação de grandes sistemas de forma escalonável e organizada.

Atualmente a linguagem sofre forte expansão com a inclusão de um número cada vez maior de bibliotecas, migração para equipamentos portáteis (idéia original da Sun), componentes distribuídos (Enterprise JavaBeans) e utilização dentro de banco de dados (*stored procedures* escritos em Java).

Vale ressaltar o apoio dado por grandes corporações ao desenvolvimento e expansão da linguagem Java e que, recentemente, a Oracle adquiriu a Sun.

3.3.2 Independência de plataforma

A linguagem Java tem garantida a independência de plataforma devido a um componente chamado “Java Virtual Machine” ou JVM. Um programa fonte escrito em linguagem Java é traduzido pelo compilador para um arquivo com código binário proprietário Java. O compilador não traduz as instruções para código de máquina, mas sim para esse código neutro padronizado pela Sun, chamado *bytecode*. O *bytecode* pode ser interpretado e executado por uma máquina virtual Java.

A JVM é um programa capaz de interpretar os *bytecodes* produzidos pelo compilador. Com a utilização de *bytecode* há portabilidade em nível de código compilado, desde que haja uma

JVM para a plataforma onde o programa será executado. Com a difusão da linguagem Java, tem-se VMs para todos os sistemas operacionais amplamente utilizados.

Os navegadores mais populares já vêm com uma JVM. Novos avanços tem tornado o compilador dinâmico em muitos casos. Isso é possível devido a otimizações como a compilação especulativa, que aproveita o tempo ocioso do processador para pré-compilar *bytecode* para código nativo. Para melhorar o desempenho das aplicações Java, as VMs ainda trazem embutido elementos chamados Compiladores JIT (“Just in Time”). Quando um programa está sendo interpretado pela VM cada instrução encontrada no *bytecode* é traduzida para instrução de máquina. Quando uma mesma instrução é encontrada, ao invés de ser traduzida novamente o JIT aproveita a tradução anterior, disponibilizando o código de máquina. Outros mecanismos ainda mais elaborados guardam informações disponíveis somente em tempo de execução, como números de usuários, processamento usado, memória disponível, possibilitando que a JVM vá aprendendo e melhorando seu desempenho. Isto é uma realidade tão presente que hoje é fácil encontrar programas corporativos e de missão crítica usando tecnologia Java.

Na Figura 3.4 é apresentado o processo de compilação de um programa tradicional onde é gerado um código binário que pode ser usado somente em uma plataforma. E neste caso, quando o programa fonte tiver que ser alterado será necessária a sua recompilação para cada plataforma onde ele é executado.

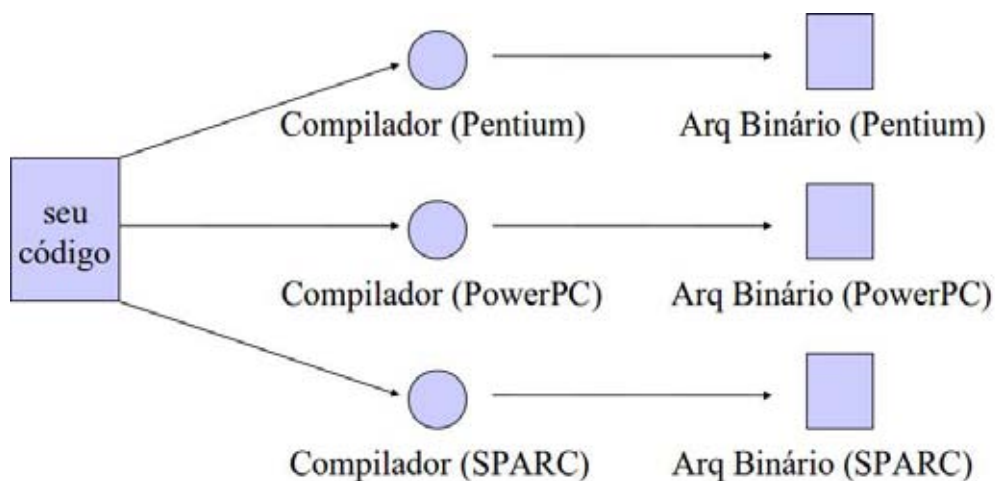


Figura 3.4: Processo de compilação dependente de plataforma.

No caso da compilação de programa Java (Figura 3.5), é gerado um código binário Java, denominado *Java bytecode*, que pode ser interpretado por qualquer plataforma de software sem a necessidade de compilação específica.

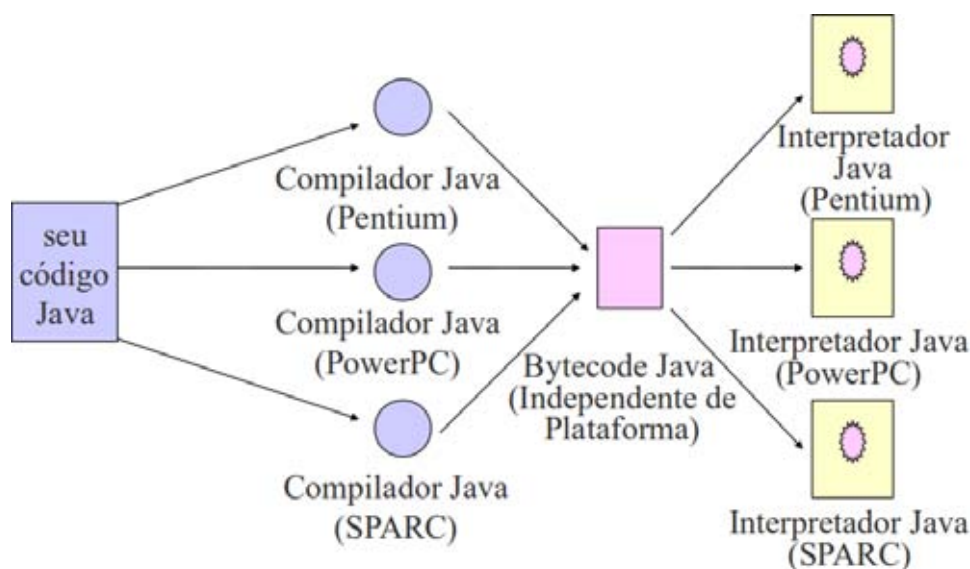


Figura 3.5: Processo de compilação independente de plataforma.

A cada ano a linguagem de programação Java tem se tornado mais rápida, na medida que se evolui o compilador dinâmico.

3.3.3 Principais características de Java

A linguagem Java é muito parecida com C++. Um programador C++ terá bastante facilidade de aprender Java pois a sintaxe é muito parecida. Esta característica é provavelmente uma das principais razões da popularidade da linguagem. Outras linguagens mais fáceis do que C++ foram inventadas nos últimos anos, tais como Delphi, mas não atraíram a mesma atenção. Apesar da semelhança, Java não herdou as complexidades da linguagem C++. Java não tem apontadores explícitos, arquivos de cabeçalho, pré-processadores, estruturas, uniões, matrizes multidimensionais, gabaritos nem sobrecarga de operadores.

A linguagem de programação Java foi projetada sob os seguintes objetivos:

1. **Orientação a Objeto** - Java é uma verdadeira linguagem de programação orientada a objetos. Tudo em Java é objeto. Mesmo os tipos simples e primitivos como números e variáveis booleanas, que não são objetos por questões de desempenho, podem ser encapsulados em objetos quando for necessário. Os programas escritos em Java são organizados em classes, que podem ser instanciadas para produzir objetos. As classes também podem herdar características (métodos e variáveis) de outras classes. Java não suporta herança de múltiplas classes mas permite que uma classe implemente mais de uma interface, que é um tipo especial de classe que não contém detalhes de implementação.

2. **Portabilidade** - Um programa escrito em Java precisa ser compilado antes de ser executado. O compilador traduz o código-fonte e gera arquivos objeto chamados arquivos de classe. Cada programa Java consiste da implementação de uma classe principal, que depois de compilado, pode ser executado em qualquer plataforma onde exista um sistema de tempo de execução Java. Com a portabilidade em nível de código compilado evita-se o trabalho de reescrever as aplicações desenvolvidas para as várias plataformas onde será executada. Atualmente, com a utilização de sistemas cada vez mais heterogêneos, esta vantagem passa a ser decisiva na escolha da linguagem de desenvolvimento.
3. **Robustez** - Java é uma linguagem que tem tipagem de dados forte: ela exige que os tipos de objetos e números sejam explicitamente definidos durante a compilação. O compilador não deixa passar qualquer indefinição em relação ao tipo de dados. Esta característica garante uma maior segurança do código e o torna menos sujeito a erros.

Em Java, a alocação de memória ocorre quando uma classe é instanciada com o comando *new*. A liberação de memória não precisa ser feita explicitamente. A linguagem fornece o “Garbage Colletion” (GC). Toda vez que um espaço de memória (objeto) não possui referências apontadas para ele, este mecanismo se encarrega de fazer a limpeza automaticamente. Não é preciso sair do programa para que o GC entre em operação, basta que não existam mais referências se referindo a um objeto para que ele seja retirado da memória. Quando uma classe é instanciada, a VM guarda uma cópia da referência feita ao programa. Através deste ponteiro oculto, o GC consegue uma referência para o objeto e libera a memória.

E quando ocorrem erros ou situações inesperadas, Java possui um meio de lidar com ele e se recuperar do erro, se possível. O controle de exceções é uma parte básica da linguagem e em muitos casos seu uso é obrigatório.

4. **Dinâmica** - A linguagem Java foi projetada para se adaptar a um ambiente dinâmico, em constante evolução. Possui uma representação de tempo de execução que permite que o programa saiba a classe a que pertence um objeto durante a execução. Isso permite a inclusão dinâmica de classes em qualquer aplicação.

Java também suporta a integração com métodos nativos de outras linguagens. Desta forma, podem surgir, em breve, aplicativos híbridos em Java e C++ aproveitando o grande volume de código já existente.

5. **Suporte a Rede** - Como a linguagem surgiu no ambiente Internet, possui amplo suporte a comunicação em rede e construção de aplicações cliente-servidor. Possui extensa

biblioteca de rotinas que facilitam a cooperação com protocolos TCP/IP, como HTTP e FTP. A linguagem Java possui suporte para programação com *Threads* e componentes distribuídos. Programas em Java podem ter mais de um linha de execução ocorrendo ao mesmo tempo. Os programadores podem definir prioridades para certas linhas de execução. Desta forma, os programas podem realizar tarefas concorrentes, ou seja, linhas de execução paralelas - é bom frisar que só ocorrerá processamento paralelo do programa se o computador possuir dois ou mais processadores, caso contrário será executado somente uma instrução a cada vez, mas para o usuário parecerá que essas instruções estão sendo executadas em paralelo dada a capacidade preemptiva do sistema operacional. Java ainda fornece meios de sincronizar essas linhas de execução. Quando um programa usa muitas delas e há o risco de utilizarem o mesmo conjunto de dados, é necessário sincronizar as ações para que não haja conflitos. Em Java, um método pode ser declarado sincronizado para garantir que o objeto no qual atua não possa ser alterado por outros métodos enquanto estiver operando sobre ele.

6. **Segurança** - Por ter uma tipagem de dados forte, só há permissão de acesso a campos pelo nome (e não por endereço). Por não ter aritmética de ponteiros nem qualquer tipo de acesso direto à posição de memória, um programa compilado em Java pode ser verificado antes de ser executado. A verificação dos *bytecodes* é realizada nos navegadores Web que suportam Java para garantir que as applets não estejam violando as restrições da linguagem e não possam provocar danos no computador do usuário local. Depois da verificação, os applets podem ser otimizados pelo sistema de execução, para garantir um melhor desempenho. A execução numa máquina cliente também é restrita, não permitindo a escrita no disco nem o acesso a outra localidade na rede a não ser àquela que enviou a applet. Como Java não dá acesso direto à memória, torna-se muito difícil o desenvolvimento de vírus. A segurança ocorre também no gerenciamento de memória feito pela VM e na proibição de aritmética de ponteiros.

3.3.4 Vantagens e desvantagens

Uma das grandes vantagens da linguagem Java é a disponibilidade de um enorme conjunto de bibliotecas de funções prontas que podem ser usadas pelo programador. A linguagem Java tem adquirido cada vez mais funcionalidade através de mudanças e incorporação de novas funcionalidades no conjunto de suas bibliotecas de funções. A “Sun Microsystems” tem investido bastante no desenvolvimento das bibliotecas de funções.

A linguagem Java tem sido muito usada no desenvolvimento de programas que realizam o

trabalho de conectar programas clientes com programas servidores, como por exemplo, servidores de banco de dados, servidores Web e outros tipos de servidores. Programas que realizam este tipo de tarefa são conhecidos por “middleware”, pois realizam o trabalho de conectar entidades de software.

Seguindo novas tendências de modelagem e desenvolvimento aplicada à maioria das linguagens modernas, Java permite a construção de grandes sistemas de forma escalonável e organizada. Atualmente, com a utilização de sistemas cada vez mais heterogêneos, esta vantagem passa a ser decisiva na escolha da linguagem de desenvolvimento.

Por outro lado, o desenvolvimento de aplicações escritas em Java exige do programador o conhecimento de orientação a objetos, o que é um paradigma de programação bem diferente do paradigma de programação tradicional baseado em procedimentos. O projeto de um sistema baseado em objetos é mais demorado que um projeto tradicional. Java exige um treinamento da equipe de projetistas e programadores no paradigma de orientação a objetos.

A pré-compilação exige um certo tempo, o que faz com que programas Java demorem um tempo significativo para começarem a funcionar. Soma-se a isso o tempo de carregamento da Máquina Virtual. Isso não é grande problema para programas que rodam em servidores e que deveriam ser inicializados apenas uma vez. No entanto, isso pode ser bastante indesejável para computadores pessoais onde o usuário deseja que o programa processe logo depois de abri-lo.

Os *bytecodes* produzidos pelos compiladores Java podem ser usados num processo de engenharia reversa para a recuperação do programa-fonte original. Esta é uma característica que atinge em menor grau todas as linguagens compiladas.

O padrão Java tem uma especificação rígida de como devem funcionar os tipos numéricos. Essa especificação não condiz com a implementação de pontos flutuantes na maioria dos processadores o que faz com que Java seja significativamente mais lento para estas aplicações quando comparado a outras linguagens.

3.3.5 Princípios básicos de um ambiente Java típico

Sistemas Java geralmente consistem em um ambiente, na linguagem, na interface de programas aplicativos Java e várias bibliotecas de classes. Os programas Java normalmente passam por cinco fases para serem executadas. Essas fases são: edição, compilação, carga, verificação e execução.

- *Fase 1:* Edição do arquivo. O programador digita um programa Java utilizando um editor

de texto e faz as correções se necessário. Os arquivos Java terminam com a extensão *.java*.

- *Fase 2:* O comando *javac* é utilizado para compilar o programa. O compilador Java traduz o programa Java para *bytecodes* - a linguagem entendida pelo interpretador Java. Ex.: *javac ENGTJ.java*. Se o programa compilar corretamente será gerado um arquivo chamado *ENGTJ.class*. Esse é o arquivo que contém os *bytecodes* que serão interpretados durante a fase de execução.
- *Fase 3:* É chamada de carga. O programa deve ser primeiramente colocado na memória antes de poder ser executado. Isso é feito pelo carregador de classe que pega o arquivo *.class* que contém os *bytecodes* e o transfere para a memória. O arquivo *.class* pode ser carregado a partir de um disco em seu sistema ou através de uma rede (rede local de universidade ou empresa, ou mesmo a Internet).
- *Fase 4:* Antes dos *bytecodes*, em uma *applet*, serem executados pelo interpretador Java embutido em um navegador ou pelo *appletviewer*, eles são verificados pelo *verificador de bytecode*. Isso assegura que os *bytecodes* para classes que são carregadas a partir da Internet sejam válidos e não violem as restrições de segurança de Java.
- *Fase 5:* Por fim, o computador, sob o controle de sua CPU, interpreta o programa, um *bytecode* por vez, realizando assim a ação especificada pelo programa.

4 A Interface Gráfica Proposta (ENGTJ)

4.1 Introdução

Neste capítulo apresenta-se uma proposta de simulação computacional, denominada ENGTJ, com o objetivo principal de se estabelecer os procedimentos necessários para o desenvolvimento de um programa, construído de forma modular e que utiliza recursos computacionais para uso via Web, para o auxílio ao ensino de conceitos básicos da teoria dos jogos. A Figura 4.1 mostra a tela inicial da interface.



Figura 4.1: Tela do menu principal.

A interface ENGTJ apresenta os módulos: Jogo do Dilema do Prisioneiro (JDP) e Jogo Coalizacional entre Empresas (JCE), escritos sob a forma de *Java applets*, para um sistema de

auxílio ao ensino baseado na WWW objetivando maiores facilidades para o aprendizado dos principais conceitos relacionados à teoria dos jogos.

O programa computacional simula a execução do Jogo do “Dilema do Prisioneiro”, bastante disseminado na teoria dos jogos não-cooperativos, bem como a execução do Jogo “Coalizacional entre Empresas”, no intuito de exemplificar o conceito de solução *core* presente na teoria dos jogos cooperativos.

Nas seções seguintes, todo o processo de desenvolvimento da interface proposta é relatado com base nos conceitos da engenharia de software, inicialmente apresentados. Por ser bastante exposto na literatura e, portanto, didático, o problema do dilema do prisioneiro será apresentado sob a forma de modelos por ser mais didático.

4.2 O Processo de Desenvolvimento nos Padrões da Engenharia de Software

Um processo de desenvolvimento de software é um método para organizar as atividades relacionadas com a criação, entrega e manutenção de sistemas de software.

Para criar o software de uma aplicação, é necessária uma descrição do problema e dos seus requisitos - o que é o problema e o que o sistema deve fazer. A análise enfatiza uma investigação do problema e como uma solução é definida.

Para desenvolver uma aplicação é necessário ter descrições de alto nível e detalhadas da solução lógica e de como ela atende os requisitos e as restrições. A essência da *análise* e do *projeto orientado a objetos* é enfatizar o domínio do problema e uma solução lógica, segundo a perspectiva de objetos.

Na Figura 4.2, apresenta-se uma visão geral das fases de desenvolvimento de software.



Figura 4.2: Fases de desenvolvimento.

Os projetos de software são complexos, e a decomposição *dividir para conquistar* é a estratégia básica para lidar com esta complexidade, ou seja, quebrar um problema em unidades que podem ser administradas. Antes da análise e do projeto orientados a objetos, a abordagem mais popular para a decomposição de um problema era a análise e projeto estruturados, nos quais a decomposição é feita basicamente através de funções e processos, resultando numa decomposição hierárquica de processos compostos por subprocessos. Contudo, outros critérios de decomposição são possíveis; a análise e o projeto orientados a objetos enfatizam a decomposição do espaço de um problema por objetos em vez de funções.

Um sistema (do mundo real ou de software) é, de modo geral, excessivamente complexo, portanto é necessário decompô-lo em pedaços que podem ser representados como modelos que descrevem e abstraem aspectos essenciais do sistema (RUMBAUGH, 1998).

Durante a *análise orientada a objetos*, há uma ênfase na descoberta e na descrição dos objetos - ou conceitos - do domínio do problema. No módulo JDP, alguns dos conceitos incluem aluno (usuário), prisioneiro e estratégia. Já no módulo JCE, os conceitos envolvidos são, além de aluno (usuário), coalizão, triângulo e reta.

Durante o *projeto orientado a objetos* existe uma ênfase na definição de elementos lógicos de software que serão implementados em uma linguagem de programação orientada a objetos. Estes objetos têm atributos e métodos. Ilustrando, na simulação do dilema do prisioneiro pode-se ter um atributo, o tipo da estratégia e o método “jogar”; e na simulação do jogo coalizacional pode haver atributo como “coordenada” e método como “registrar imputações”.

As atividades de análise e projeto existem ao longo de um contínuo, dentro de um processo iterativo e incremental. Entretanto, alguma distinção consistente entre investigação (análise) e solução (projeto), na prática, é útil, porque é vantajoso ter um passo bem-definido que enfatiza a pesquisa do que é o problema, antes de mergulhar em formas de criar uma solução.

4.2.1 Definição do modelo conceitual

A modelagem é uma parte central de todas as atividades que levam à implantação de um software de qualidade. Constróem-se modelos para comunicar a estrutura e o comportamento desejados do sistema, visualizar e controlar a arquitetura do sistema, compreender melhor o sistema que está sendo elaborado, expondo oportunidades de simplificação e reaproveitamento e gerência dos riscos.

Os modelos fornecem uma visão do projeto de um sistema. Cada modelo é, portanto, uma abstração semanticamente específica do sistema.

Quatro objetivos são atingidos com a modelagem:

- Os modelos ajudam a visualizar o sistema como ele é ou como deseja-se que seja;
- Os modelos permitem especificar a estrutura ou o comportamento de um sistema;
- Os modelos proporcionam um guia para a construção do sistema;
- Os modelos documentam as decisões tomadas.

4.2.2 Definição do diagrama de comunicação

O projeto orientado a objetos se preocupa com a definição de especificações lógicas de software que atendem os requisitos funcionais, baseado na decomposição por classes de objetos. Um passo essencial é a ilustração de como eles interagem através de mensagens expressas em diagramas de comunicação. Os diagramas de comunicação mostram o fluxo de mensagens entre instâncias e a invocação de métodos, ou seja, sugerem as conexões necessárias entre objetos e os métodos que cada classe de software deve definir. Ele ilustra o passo essencial do jogo, através do envio de mensagens a instâncias das classes, por exemplo, *Jogador* e *Estratégia*.

4.2.3 Definição do diagrama de classes

Uma das técnicas mais utilizadas no desenvolvimento orientado a objetos é o diagrama de classes. Para cada tipo de objeto o diagrama de classe descreve: a sua identidade, os relacionamentos, os atributos e as operações.

No jogo dilema do prisioneiro, uma inspeção do diagrama de colaboração conduz ao diagrama de classes de projeto. Uma vez que uma mensagem *jogar* é enviada para uma instância de *jogador*, *jogador* requer um método *jogar*, enquanto *estratégia* requer um método *escolher/decidir*.

O jogo dilema do prisioneiro é um problema apresentado com a intenção de focalizar alguns dos passos e artefatos da análise e do projeto orientado a objetos, em vez de focalizar-se no domínio do problema.

4.2.4 A linguagem de modelagem unificada (UML)

UML é a abreviação de “Unified Modeling Language”, uma notação para modelagem de sistemas usando conceitos orientados a objetos. O processo de desenvolvimento descreve uma

possível ordem de atividades e um ciclo de vida de desenvolvimento.

UML é um padrão consolidado, aceito pela indústria, para a modelagem orientada a objetos. Começou como um esforço conjunto de Grady Booch e Jim Rumbaugh em 1994, para combinar seus dois métodos populares - os métodos Booch e OMT (“Object Modeling Technique”). Mais tarde, Ivar Jacobson se juntou a eles (o criador do método OOSE, “Object Oriented Software Engineering”). Em resposta a uma solicitação do OMG (“Object Management Group”, uma entidade de padronização) para definir uma linguagem e notação de modelagem padronizada, a UML foi submetida como candidata em 1997 (OMG, 2007).

O OMG aceitou a UML, a qual também recebeu a aprovação pela indústria, uma vez que seus criadores apresentam métodos de análise e/ou projeto de primeira geração muito populares. Muitas organizações de desenvolvimento de software e fornecedores de ferramentas de casos de uso adotaram a UML, e já se tornou um padrão mundial utilizado por desenvolvedores, autores e fornecedores de ferramentas de caso de uso (OMG, 2007).

A UML fornece tecnologia necessária para apoiar a prática de engenharia de software orientada a objetos, mas não fornece o arcabouço de processo para guiar as equipes de projeto na aplicação da tecnologia. Ao longo dos últimos anos, pesquisadores como Jacobson, Rumbaugh e Booch desenvolveram o PU (“Processo Unificado”), um arcabouço para a engenharia de software orientada a objetos usando a UML. Hoje em dia, o *Processo Unificado* e a *UML* são amplamente usados em projetos orientados a objetos de todas as naturezas. O modelo iterativo e incremental proposto pelo PU pode, e deve, ser adaptado para satisfazer às necessidades específicas de projeto.

Um ciclo de vida iterativo se baseia no aumento e no refinamento sucessivo de um sistema através de múltiplos ciclos de desenvolvimento. Cada ciclo trata um conjunto relativamente pequeno de requisitos do mundo real. O sistema cresce incrementalmente, à medida que cada ciclo é completado. Isso contrasta com o clássico ciclo de vida em cascata, no qual cada atividade (análise, projeto e assim por diante) é executada uma única vez para o conjunto inteiro de requisitos do sistema.

Vantagens do ciclo de vida iterativo:

- A complexidade nunca se torna incontrolável;
- A realimentação é gerada cedo no processo, porque a implementação ocorre rapidamente para um pequeno subconjunto do sistema.

O processo unificado é um processo de software *orientado por casos de uso, centrado na*

arquitetura, iterativo e incremental, projetado como um arcabouço para métodos e ferramentas UML. O Processo Unificado é um modelo incremental no qual cinco fases são definidas (PRESSMAN, 2006):

- uma fase de *Concepção* que engloba tanto a comunicação com o cliente quanto atividades de planejamento, enfatiza o desenvolvimento e refinamento de casos de uso como o modelo principal;
- uma fase de *Elaboração* que engloba atividades de comunicação com o cliente e modelagem com foco na criação de modelos de análise e projeto com ênfase nas definições de classes e representações arquiteturais;
- uma fase de *Construção* que refina e então traduz modelo de projeto para componentes de software implementados;
- uma fase de *Transição* que transfere o software do desenvolvedor para usuário final para testes beta e aceitação;
- uma fase de *Produção* em que contínuo monitoramento e suporte são conduzidos.

É provável que, ao mesmo tempo em que as fases de construção, transição e produção estejam sendo conduzidas, o trabalho já tenha sido iniciado no incremento de software seguinte. Isso significa que as cinco fases do PU não ocorrem em sequência, mas em titubeante concorrência.

4.2.5 A solução programada em Java

A fase de *construção* de um projeto envolve repetidos ciclos de desenvolvimento, dentro dos quais o sistema é estendido. O objetivo final é um sistema de software em operação que atenda corretamente os requisitos do sistema. Para reduzir os riscos e aumentar as chances de desenvolver uma aplicação adequada, o desenvolvimento deve estar baseado em um volume significativo de modelagem em análise e em projeto, antes de começar a codificação.

As fases de análise e projeto foram interrompidas para fazer alguma programação exploratória, de maneira a descobrir o funcionamento da arquitetura Eclipse e, então, retomar à fase formal de projeto. Uma quantidade significativa de informações auxiliou a geração do código Java num processo de tradução relativamente direto.

Uma das forças de um processo de desenvolvimento iterativo e incremental está no fato de que os resultados de um ciclo anterior realimentam o início do ciclo seguinte.

Um volume significativo de decisões e de trabalho criativo é realizado durante as fases de análise e projeto para que a geração do código seja um processo de tradução relativamente direto. Contudo, em geral, a fase de programação não é um passo trivial de geração de código. Durante a fase de programação e teste são feitas muitas mudanças, descobertos e resolvidos problemas em detalhes. Modificações e desvios do projeto ocorrem também durante a fase de construção e de teste.

4.3 O Jogo do Dilema do Prisioneiro (JDP)

4.3.1 Especificação de requisitos

O objetivo deste módulo é criar um sistema de apoio ao ensino com base na teoria dos jogos não-cooperativos tomando como exemplo principal o problema do dilema do prisioneiro.

Esses objetivos incluem:

- *Identificação rápida do aluno (usuário) que vai interagir com o programa;*
- *Exibição das possíveis escolhas (estratégias) para o prisioneiro;*
- *Análise rápida e precisa do crime;*
- *Geração, exibição e gravação da sentença final do crime.*

4.3.2 Funções básicas

- *R1.1 Registrar usuário (aluno);*
- *R1.2 Exibir tabela de estratégias (escolhas);*
- *R1.3 Interrogar prisioneiros: confessar/não-confessar;*
- *R1.4 Registrar as respostas de cada prisioneiro num mecanismo de armazenamento físico;*
- *R1.5 Exibir a sentença final do crime.*

Mais importante que seguir um processo ou método oficial é o desenvolvedor adquirir a habilidade de criar um bom projeto, e que se sustente o desenvolvimento desse tipo de habilidade. A habilidade surge do domínio de um conjunto de princípios e de heurísticas relacionadas

com a identificação e a abstração de objetos adequados e da atribuição de funcionalidades aos mesmos.

Uma visão das melhores práticas e dos melhores modelos empregados é o processo de desenvolvimento de sistemas orientados a objetos basear-se numa abordagem iterativa e incremental, dirigida por casos de uso. Ou seja, os processos e os requisitos de negócios são descobertos e depois expressos sob a forma de casos de uso.

4.3.3 Caso de uso para o módulo JDP

Um caso de uso é um documento narrativo que descreve a sequência de eventos de um agente externo que usa um sistema para completar um processo (RUMBAUGH, 1998). Casos de uso não são exatamente especificações de requisitos ou especificação funcional, mas ilustram e implicam requisitos na história que eles contam.

Na Tabela 4.1 é apresentado o caso de uso relacionado a exibição da sentença final do crime, para o módulo JDP. A identificação de processos e o seu registro em casos de uso não são realmente uma atividade de análise orientada a objeto, contudo, é um passo inicial importante e amplamente praticado naquilo que são denominados *método de análise* e *projeto orientado a objetos*.

Tabela 4.1: Caso de uso: sentença final do crime.

Caso de Uso	Exibir a sentença final do crime.
Atores	Sistema, Aluno.
Tipo	Primário.
Descrição	O sistema exibe numa caixa de diálogo de informações a sentença final do crime. O sistema disponibiliza um botão de execução para exibição dos resultados.
Função	R1.5

Compreender os requisitos inclui, em parte, compreender os processos do domínio e o ambiente externo. Casos de uso são um passo preliminar útil na descrição dos requisitos do sistema, como é mostrado na Figura 4.3. Na figura, o prisioneiro identifica-se com seus dados pessoais. Como resposta do sistema, o prisioneiro é interrogado a respeito do crime. O sistema registra o interrogatório do prisioneiro e em seguida a sentença final do crime é apresentada.

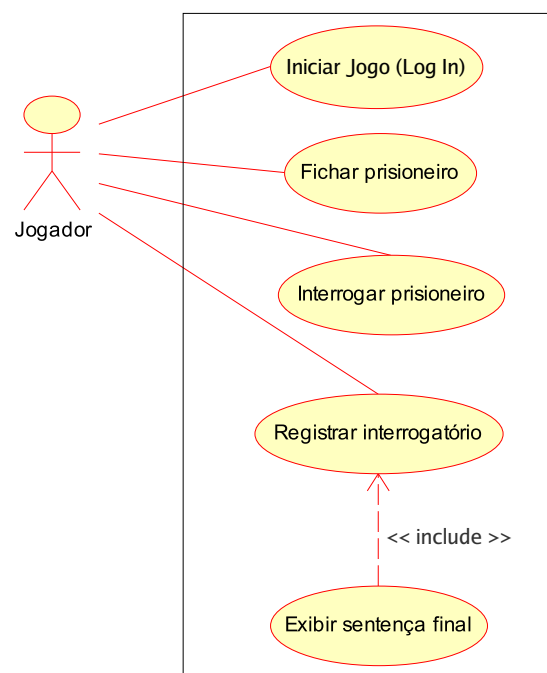


Figura 4.3: Diagrama de caso de uso (JDP).

No diagrama de comunicação do módulo JDP (Figura 4.4) é mostrado o fluxo de mensagens entre instâncias e a invocação dos métodos.

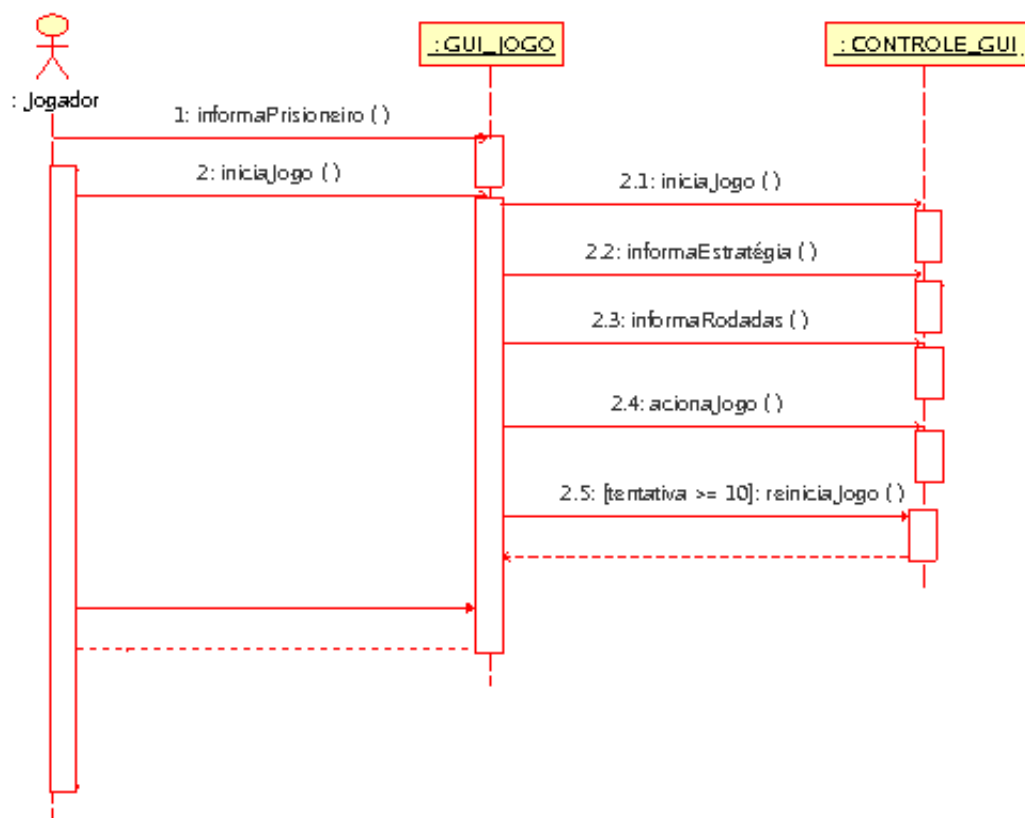


Figura 4.4: Diagrama de comunicação (JDP).

No diagrama de classes do módulo JDP (Figura 4.5) são mostrados os atributos, as operações e os relacionamentos entre os objetos.

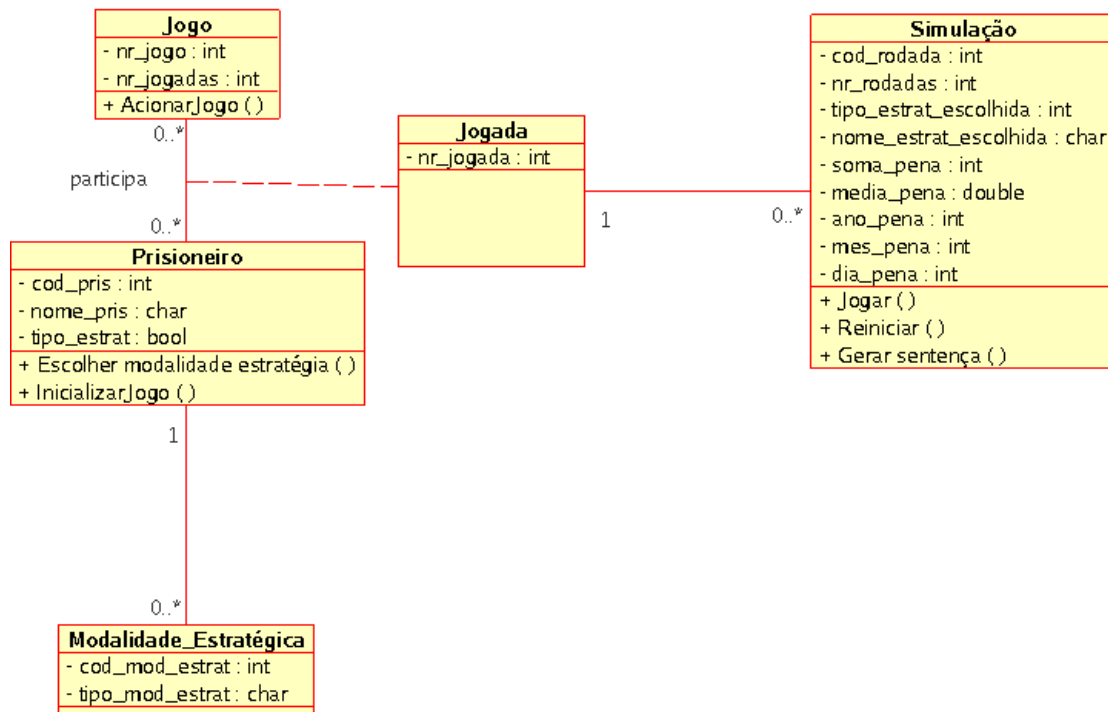


Figura 4.5: Diagrama de classes (JDP).

4.4 O Jogo Coalizacional entre Empresas (JCE)

O objetivo do jogo coalizacional entre empresas é a simulação de um jogo com base na teoria dos jogos cooperativos. A partir das coordenadas de um triângulo equilátero na forma de representação coalizacional (por acordos) é que se descreve o conjunto finito de jogadores, as opções disponíveis para cada jogador e os ganhos gerados em cada uma das opções de jogo.

4.4.1 Especificação de requisitos

Os principais objetivos do programa são:

- *Identificação rápida do aluno (usuário);*
- *Análise rápida e precisa do jogo;*
- *Exibição das possíveis escolhas para cada jogador (empresa);*
- *Geração e registro das imputações estáveis/instáveis do jogo.*

4.4.2 Funções básicas

- *R1.1 Registrar usuário (aluno);*
- *R1.2 Registrar lado de atuação do jogador no triângulo;*
- *R1.3 Registrar as imputações entre os jogadores (empresas);*
- *R1.4 Registrar a posição de cada coalizão;*
- *R1.5 Exibir o resultado final do jogo.*

4.4.3 Caso de uso para o módulo JCE

Na Tabela 4.2 é apresentado o caso de uso para o registro do lado de atuação do jogador no triângulo, para o módulo JCE.

Tabela 4.2: Caso de uso: lado de atuação do jogador no triângulo.

Caso de Uso	Registrar lado de atuação do jogador no triângulo.
Atores	Jogadores (empresas).
Tipo	Primário.
Descrição	O jogador (empresa) escolhe o seu lado de atuação no triângulo que pode ser a reta base verde, a reta esquerda vermelha ou a reta direita azul. O sistema disponibiliza três botões de execução para definição da área de atuação.
Função	R1.2

O caso de uso do jogo coalizacional é mostrado a seguir na Figura 4.6. Na figura, o jogador/empresa inicia o jogo. Em seguida gera as configurações iniciais da tabela de benefícios do jogo e atribui a reta de atuação do jogador no triângulo. Por meio de cada reta do triângulo são enviadas as coordenadas e registradas as devidas coalizões. A partir do registro das imputações do jogo, para cada coalizão, são geradas as imputações estáveis e instáveis como resposta do sistema.

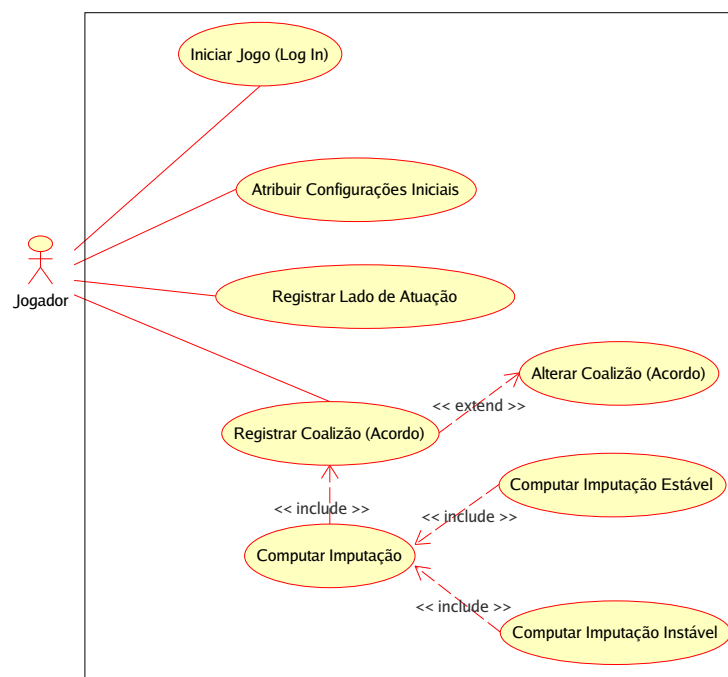


Figura 4.6: Diagrama de caso de uso (JCE).

No diagrama de comunicação do módulo JCE (Figura 4.7) é mostrado o fluxo de mensagens entre instâncias e a invocação dos métodos.

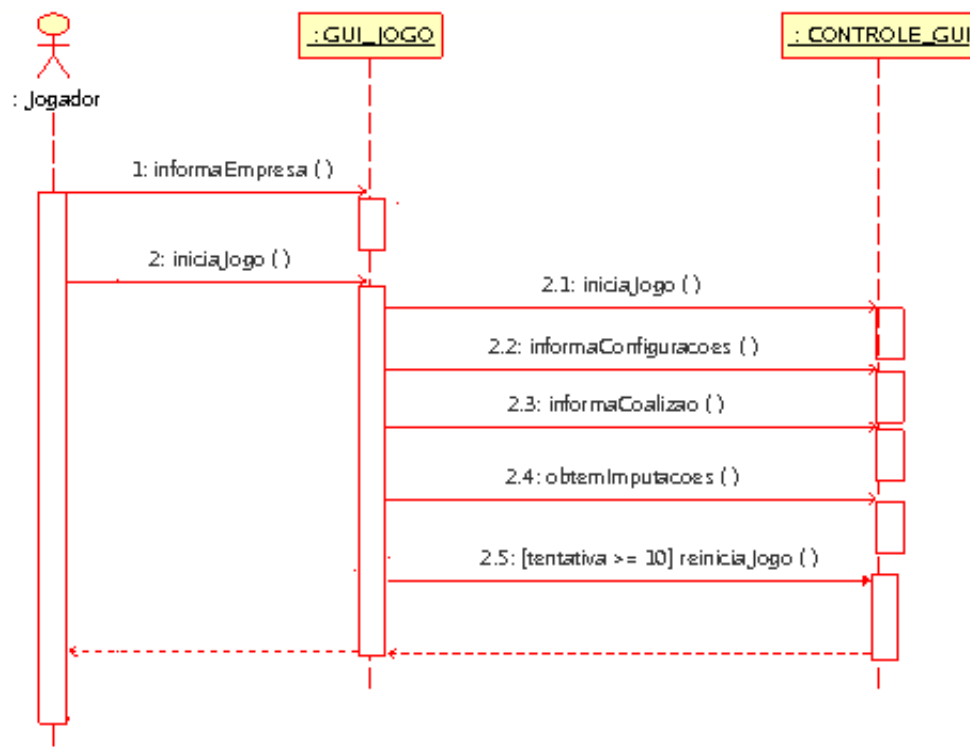


Figura 4.7: Diagrama de comunicação (JCE).

No diagrama de classes do módulo JCE (Figura 4.8) são mostrados os atributos, as operações e os relacionamentos entre os objetos.

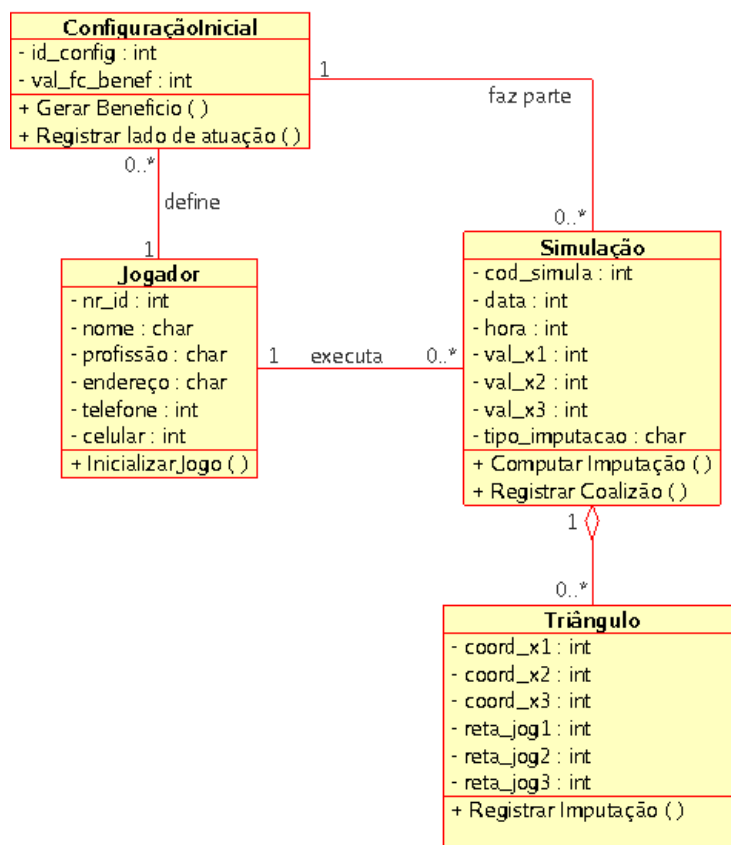


Figura 4.8: Diagrama de classes (JCE).

4.5 A Fase de Construção

4.5.1 Introdução

A fase de execução e testes do sistema ocorre quando o usuário se loga no sistema Linux, seguido pelo acesso ao ambiente de desenvolvimento Eclipse para execução dos programas escritos em Java. Existem duas maneiras distintas de se executar os jogos do dilema do prisioneiro (JDP) e coalizacional entre empresas (JCE), ou seja, existem dois tipos de programas para os quais o carregador de classe carrega arquivos *.class* - *aplicativos* e *applets*. A primeira é no próprio Eclipse sob a forma de aplicação e a outra forma é chamando pelo navegador sob a forma de *Java applet*. Um *aplicativo* é um programa semelhante a um programa processador de texto, um programa de planilha, um programa de desenho ou um programa de correio eletrônico, que normalmente são armazenados e executados a partir do computador local do usuário. Um *applet* é um programa que, normalmente, é armazenado e executado no navegador e descartado

quando se completa a execução.

Os aplicativos são carregados na memória e executados utilizando o interpretador Java através do comando *java*. Ao executar o jogo do dilema do prisioneiro, o comando *java* invoca o interpretador para os programas JDP e JCE e faz com que o *carregador de classes* carregue as informações utilizadas nos mesmos.

O carregador de classe também é executado quando os *applets* JDP e JCE são carregados em um navegador como Internet Explorer ou Mozilla Firefox. Os navegadores visualizam documentos na Web chamados via HTML. A linguagem HTML é utilizada para formatar um documento de maneira facilmente entendida pelo navegador.

Um documento em HTML pode referir-se a um *Java applet*. Quando o navegador vê os *applets* JDP e JCE referenciados em um documentos de HTML, como é o caso da interface ENGTJ, o navegador dispara o carregador de classe Java para carregar o *applet*. Cada navegador que suporta Java tem um interpretador Java embutido. Uma vez que o *applet* é carregado, o interpretador Java do navegador o executa.

Os *applets* também podem ser executados a partir da linha de comando usando o comando *appletviewer* fornecido com o J2SDK - o conjunto de ferramentas que inclui o compilador (*javac*), o interpretador (*java*), o visualizador (*appletviewer*) e outras ferramentas utilizadas por programadores Java. Como os navegadores, o *appletviewer* requer um documento HTML para invocar um *applet*. Se o arquivo ENGTJ.html refere-se aos *applets* ou *javascripts* JDP e JCE, o *appletviewer* é utilizado: / > *appletviewer ENGTJ.html*. Isso faz com que o carregador de classe carregue as informações utilizadas nos *applets* JDP e JCE. O *appletviewer* é referido como o navegador mínimo porque ele interpreta apenas *applets*.

4.5.2 Uso no Eclipse

Tanto a interface do módulo JDP quanto a interface do módulo JCE foram escritas no ambiente de desenvolvimento Eclipse. Ao abrir o Eclipse 3.1 será exibida uma janela de boas-vindas *Welcome*, conforme é mostrado na Figura 4.9. Basta fechar a aba *Welcome* para estar dentro do ambiente de desenvolvimento apresentado na Figura 4.10.

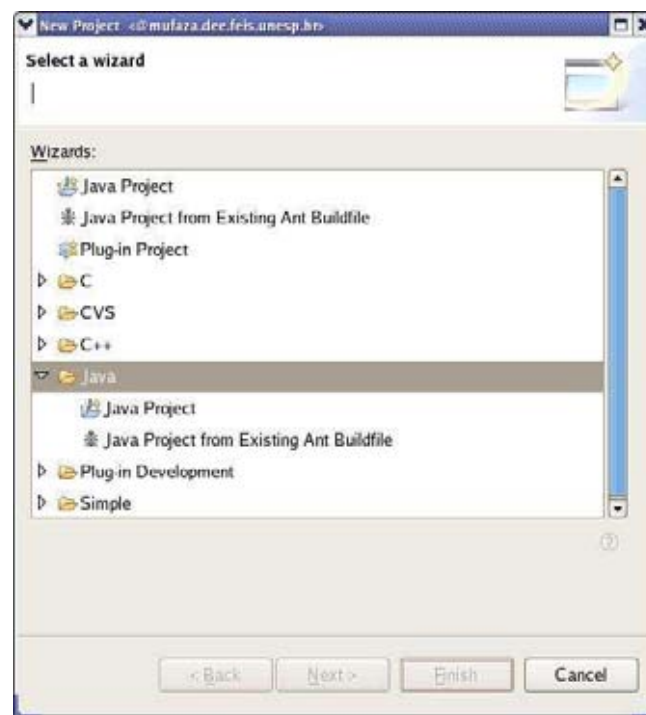


Figura 4.11: Tela para seleção de um “wizard” de criação de projeto.

Na sequência, aparecerá a tela apresentada na Figura 4.12 para a criação de um projeto Java.

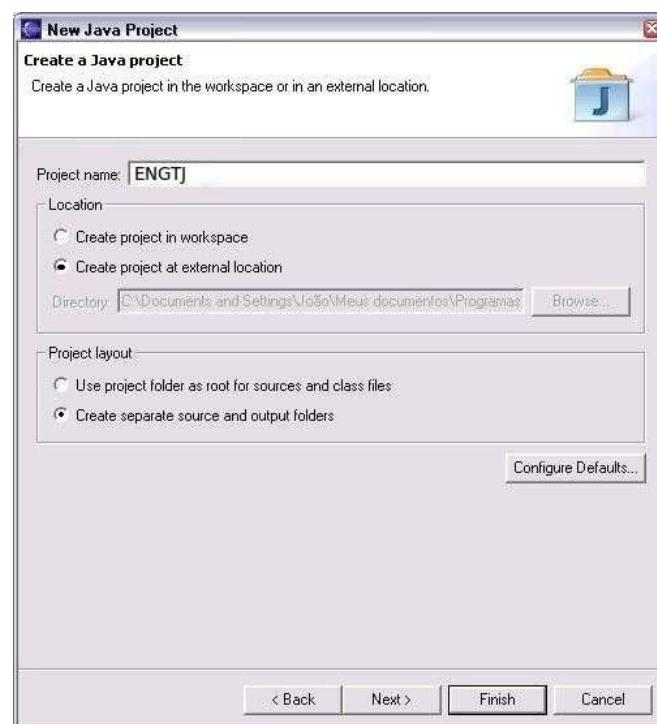


Figura 4.12: Tela para criação de um projeto Java.

- Digita-se um nome para o projeto e em seguida marca-se a opção “Create project at external location”, especificando o diretório onde o projeto será armazenado. Marca-se também a opção “Create separate source and output folders” na caixa *Project Layout*. Essa opção permite que não se misturem arquivos que contém o código-fonte com os arquivos gerados pelo compilador (“output”, ou saída), criando-se pastas diferentes para melhor organizar cada um desses tipos de arquivos.
- Aciona-se *Next*.
- Nesse passo é possível escolher os diretórios do source e de saída. Para manter os valores padrão clica-se no botão *Finish*, ou então o usuário pode alterar os valores de acordo com a conveniência de seu projeto. Se ainda aparecer uma janela pedindo para confirmar uma mudança de perspectiva, aciona-se *Yes*.

Nesse momento o projeto já está criado, mas ainda não possui nenhuma classe. Para criar uma classe basta acionar *File / > New / > Class*, a tela da Figura 4.13 aparecerá no vídeo.



Figura 4.13: Tela para criação de uma nova classe Java.

- Na parte superior da janela há algumas indicações sobre o preenchimento - como campos

obrigatórios, sugestão de usar inicial maiúscula para nomes de classes, etc.

- Na aba *Source Folder* determina-se a que projeto a classe fará parte.
- Seleciona-se o nome da sua classe no campo “Name”.
- Opcionalmente é possível determinar os modificadores de acesso da classe e deixar prontos os esqueletos de alguns métodos, no campo “Modifiers” e nos campos abaixo de “Which method stubs would you like to create?”. Para criar uma aplicação escolheu-se “public static void main(String[] args)”.
- Acionando o botão *Finish*, a classe será criada como mostrado na Figura 4.14.

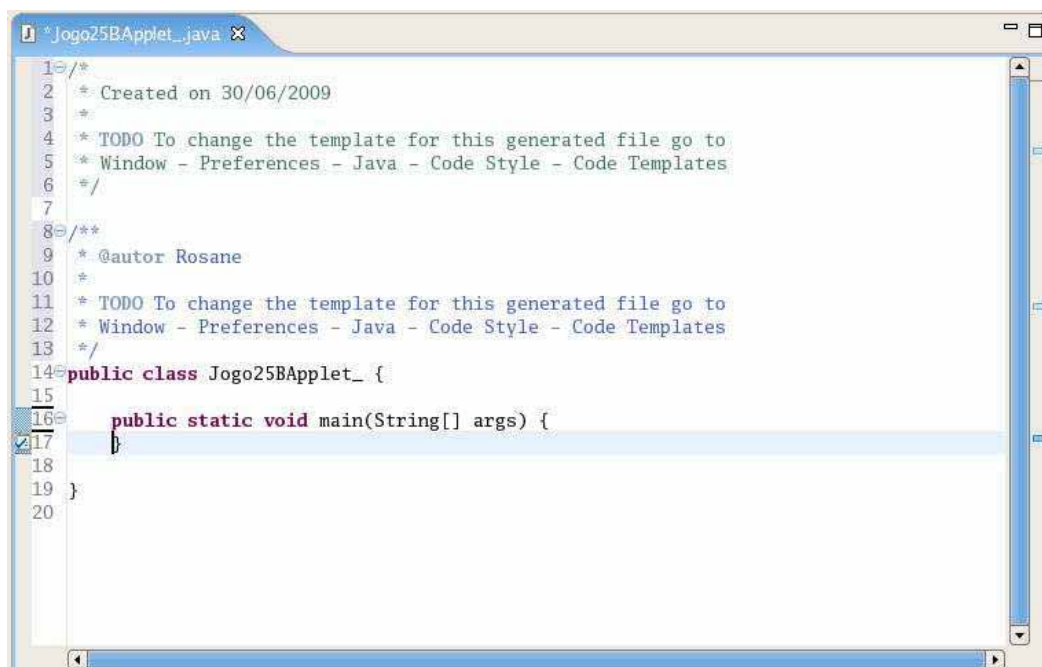


Figura 4.14: Código da nova classe Java criada.

- Percebe-se que um campo de comentários comum e um de comentários *javadoc* são automaticamente inseridos no início da classe. Para alterar os *templates* destes e outros comentários basta escolher as opções: *Window/ >Preferences/ >Java/ >Code Generation*, na aba *Code and Comments*.

4.5.2.2 compilação do programa

- Pode-se compilar o programa escolhendo as opções: *Project/ >Build Project*, ou como alternativa, acionando-se o botão direito sobre o projeto que se deseja compilar no *Package Explorer* e após escolher a opção *Build Project*.

- Para executar o programa, aciona-se com o botão direito sobre o arquivo que contém o código fonte principal que se deseja rodar e depois escolhe-se as opções: *Run/ >Java Application*. Automaticamente será criada uma configuração de execução para o projeto. Para gerenciar as configurações aciona-se *Run/ >Run* e a seguinte tela aparecerá, conforme mostra a Figura 4.15.

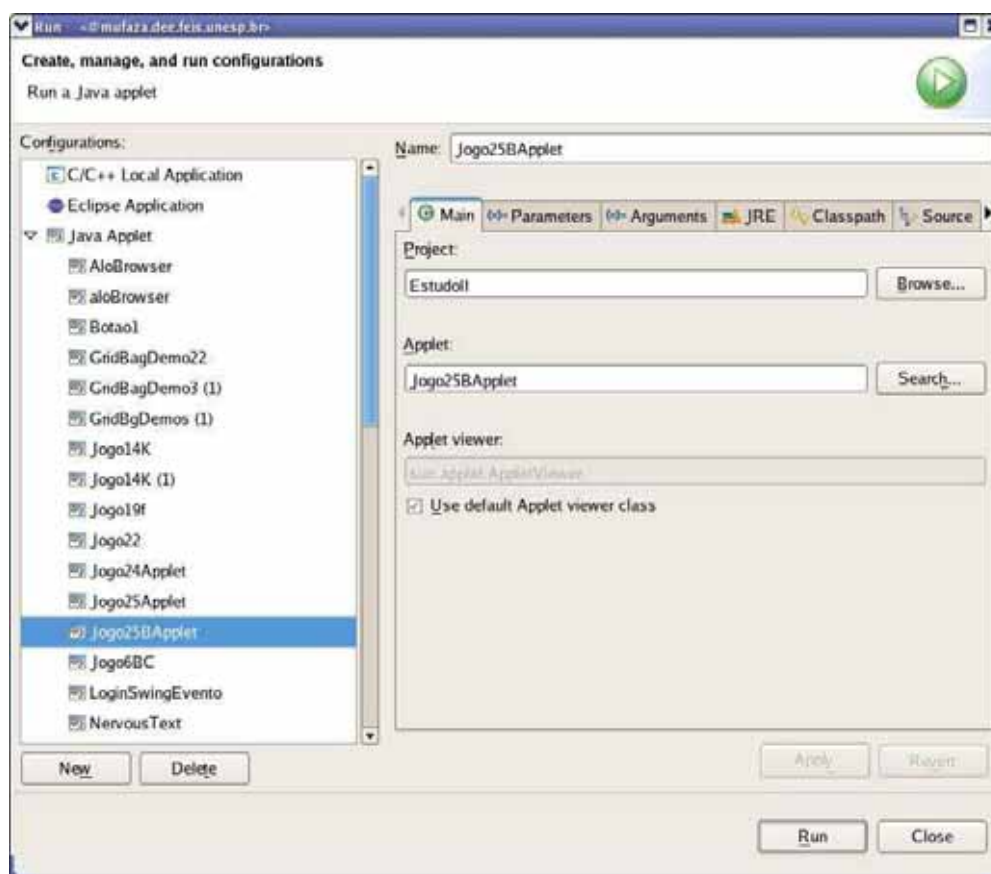


Figura 4.15: Tela de execução do programa Java.

A partir da primeira vez que se configura uma rotina de execução, basta acionar o botão *Run* na barra de ferramentas ou utilizar o atalho *Ctrl+F11* para executar o programa. É também possível rodar os programas com a combinação de teclas *Alt+Shift+X* que habilita um menu de escolha do tipo de programa a ser executado.

O *Package Explorer* é um visualizador simples e elegante de projetos. Um projeto representa toda a estrutura do seu programa, armazenando os arquivos-fonte (*.java*), os *bytecodes* (*.class*), as configurações gerais do ambiente para o projeto, eventuais arquivos de *backup* e outros arquivos inerentes ao escopo do programa. A navegação é semelhante à do Mozilla Firefox, bastante intuitiva e respeitando a hierarquia do projeto, mais facilmente familiarizável com a prática, ao longo do desenvolvimento do projeto.

O editor de textos do Eclipse denota as palavras-chave de Java em letras destacadas para facilitar a leitura do código. No entanto, esta é uma das funcionalidades mais triviais desta ferramenta, como é mostrado na Figura 4.16.

```
public int randomica(){
    return ((int) (Math.random()*2));
}

public float average ( int soma, int cont)
{
    return ((float)soma/(float)cont);
}
```

Figura 4.16: Tela de edição de código do programa.

Durante a implementação, quando for digitado o nome de alguma variável que denote um objeto e o ponto para chamada de método, o editor de textos do Eclipse exibe uma janela com uma relação completa de todos os métodos e atributos que este objeto pode acessar em seu contexto, e a medida em que se escreve as letras, será filtrado tudo o que puder ser destacado, como ilustrado na Figura 4.17.

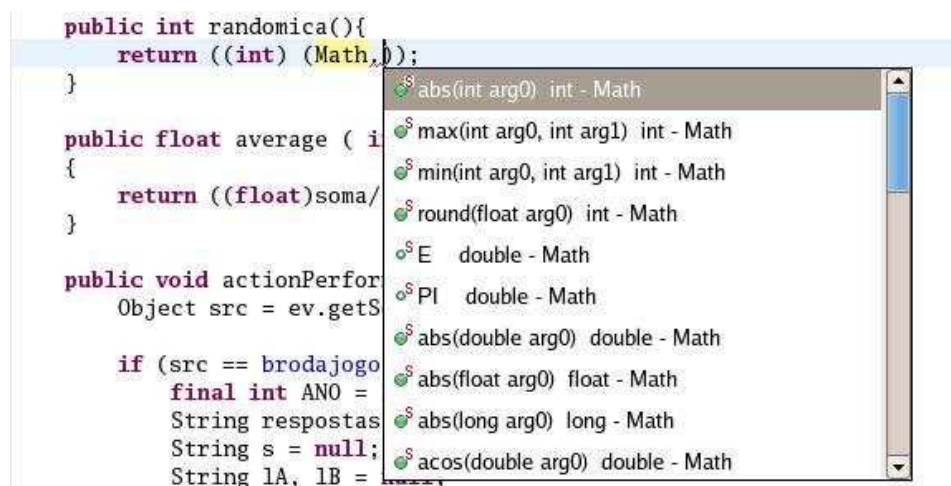


Figura 4.17: Tela de depuração do programa.

Uma grande funcionalidade das principais IDEs atuais é a detecção de erros de compilação em tempo de implementação. O editor de texto do Eclipse, ao perceber um erro de sintaxe e, inclusive, alguns poucos erros de lógica, imediatamente marca em vermelho o trecho que ele supõe estar errado, além de indicar as possíveis causas do erro e sugerir algumas soluções. Como na Figura 4.18, o trecho errado é sublinhado e as linhas que apresentam problema de sintaxe são marcadas. Ele também separa os trechos de código entre chaves, no lado esquerdo

da janela, em azul. É possível acionar os triângulos azuis para exibir/esconder trechos entre chaves, facilitando a visualização do arquivo e sua navegação.



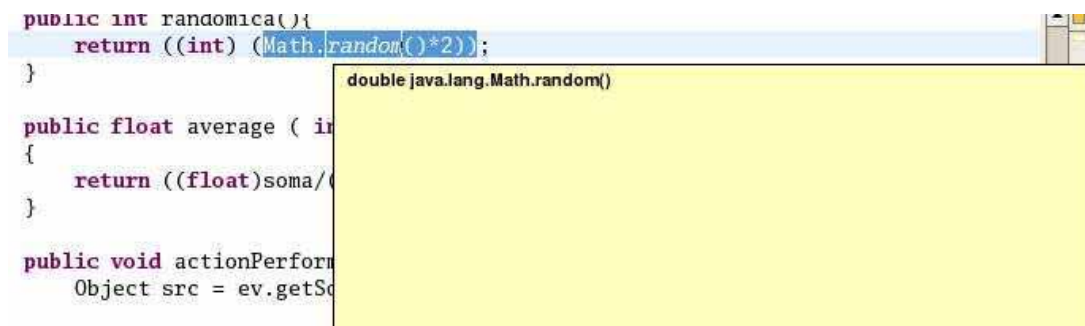
```

44 // JComboBox
45 private JComboBox listaA;
46 private String opcoesA[] = {"RANDÔMICO", "CONFESSAR", "NÃO-CONFESSAR"};
47 private String namesA[] = {"CONFESSAR ou NÃO-CONFESSAR ",
48                             "Sempre CONFESSAR           .",
49                             "Sempre NÃO-CONFESSAR        ."};
50 // JComboBox
51 private JComboBox listaB;
52 Syntax error, insert ";" to complete FieldDeclaration
53 private String namesB[] = {"CONFESSAR ou NÃO-CONFESSAR ",
54                             "Sempre CONFESSAR           .",
55                             "Sempre NÃO-CONFESSAR        ."};
56 private JLabel ljogadorA;
57 private JLabel lestrategiaA;
58 private JTextArea tareamsgA = new JTextArea(4,18);

```

Figura 4.18: Tela de edição de código com erro de sintaxe.

Por último, o editor do Eclipse (Figura 4.19) poupa bastante tempo de pesquisa em documentação de classes, nas APIs das mesmas, com um recurso muito mais simples e interessante. Ao fixar o mouse por cima de nomes de classe ou métodos, uma caixa de texto aparecerá na tela com toda a documentação da classe ou método em questão.



```

public int randomica(){
    return ((int) (Math.random()*2));
}

public float average ( int[] array ){
    {
        return ((float)soma/array.length);
    }
}

public void actionPerformed ( ActionEvent e ){
    Object src = e.getSource();
}

```

double java.lang.Math.random()

Figura 4.19: Tela de documentação sobre a classe *Math* do Java.

O Eclipse oferece um terminal de console para a entrada de dados do teclado pela “stream” de entrada padrão, e saída de dados que o programa escreve na “stream” de saída padrão. A janela *Outline* funciona semelhantemente ao *Package Explorer*, sendo que voltadas para a estrutura interna do arquivo *.java*, frequentemente a classe de uso. Ainda existem ícones diferentes para cada parte do arquivo.

5 Resultados e Discussões

5.1 Introdução

A Interface Gráfica ENG TJ foi desenvolvida no LLPP (Laboratório de Linux e Processamento Paralelo) do Departamento de Engenharia Elétrica da UNESP-Campus de Ilha Solteira, em linguagem Java, usando o Eclipse como ambiente de desenvolvimento, debaixo da plataforma Linux. Recentemente, foram feitas atualizações nos software computacionais do LLPP para o melhor aproveitamento dos recursos avançados Web; ou seja, para se extrair o que de mais vantajoso há na linguagem de programação Java - a portabilidade. O laboratório conta, hoje, com um acervo de pelo menos doze máquinas rodando Linux, e em algumas dessas máquinas já encontram-se instaladas algumas cópias do ambiente de desenvolvimento Eclipse e a linguagem de programação Java.

A proposta do trabalho é apresentar um simulador que apresenta uma interface gráfica de auxílio ao processo de ensino-aprendizado, disponibilizada na Web, que permite ao usuário, através de sucessivas interações, inferir sobre os principais conceitos teóricos abordados na resolução de problemas clássicos da teoria dos jogos aplicáveis para diferentes áreas do conhecimento. Nesta interface o usuário poderá executar os módulos: Jogo do Dilema do Prisioneiro (JDP) e Jogo Coalizacional entre Empresas (JCE), consultar os recursos de tutorial e bibliografia, gerais e específicos, disponíveis no topo e ao centro de cada uma das telas dos respectivos módulos, bem como a partir da tela inicial ter acesso a um glossário geral. A opção *Tutorial - Teoria dos Jogos* dá acesso a um texto com os principais conceitos de teoria dos jogos, a opção *Referências* exibe uma listagem das bibliografias mais gerais sobre a teoria dos jogos, e a opção *Glossário* explica os termos abordados no trabalho. Em cada módulo também é apresentado a opção *Ajuda* que serve como um manual prático de uso da ferramenta e a opção *Referências* que lista as bibliografias mais específicas sobre o assunto do respectivo módulo.

A princípio, o jogo do dilema do prisioneiro foi programado para ser executado remotamente no LLPP como aplicação *desktop* até ser convertido em *applet*. Atualmente este módulo é executado via Web juntamente com o módulo jogo coalizacional entre empresas, com todas

as funcionalidades implementadas anteriormente e encontram-se disponíveis para acesso em: <http://www.dee.feis.unesp.br/docentes/grilo/ENGTJ/ENGTJ.html>.

5.2 O Módulo JDP

O módulo JDP possui uma tela principal apresentada na Figura 5.1 na qual o usuário poderá simular a execução do jogo dilema do prisioneiro. Nesta tela, para cada jogador se pode escolher entre duas estratégias: confessar ou não-confessar o crime. Na caixa de combinações, localizada abaixo de cada um dos jogadores, Al e Bob, encontram-se as seguintes opções de modalidade estratégica: randômico, confessar, não-confessar e grim. Cada opção encontra-se vinculada à uma caixa de texto que descreve a função de cada modalidade estratégica. O número de interrogatórios para ambos os prisioneiros varia de 1 a 10. O histórico de processamento do jogo encontra-se na área de texto, parte central da tela, onde serão exibidos: o número total de rodadas do jogo, a lista de rodadas, a modalidade estratégica de cada jogador, a lista das penas em anos, o somatório das penas, a média das penas em anos e em anos, meses e dias. Por último, já no rodapé da tela estão os botões de *Jogar* e de *Reiniciar* o jogo.



Figura 5.1: Tela inicial do módulo JDP.

Antes de executar o jogo, é preciso configurá-lo com a estratégia preferida e também com o

número de rodadas (interrogatórios) de cada jogador (prisioneiro). Na Figura 5.2, o jogador Al escolheu a estratégia do tipo não-confessar (“Sempre NÃO-CONFESSAR”) e o jogador Bob escolheu o tipo grim (“Inicia CONFESSANDO depois muda para sempre NÃO-CONFESSAR”). E também o número de rodadas foi configurada para 8 (oito).



Figura 5.2: Tela do módulo JDP pré-configurado.

No rodapé da tela, o botão *Jogar* processa cada tentativa do jogo e o botão *Reiniciar* permite escolher outras estratégias até dez vezes. Ao selecionar o botão *Jogar* o jogo do dilema do prisioneiro será processado e então um histórico de processamento do jogo exibe uma lista numérica das rodadas com: a estratégia e o valor das penas para cada jogador, bem como a soma das penas, a média das penas e o valor médio das penas em anos, meses e dias de cada prisioneiro, conforme mostrado na Figura 5.3.



Figura 5.3: Tela de processamento do módulo JDP.

O usuário (aluno) poderá reiniciar o jogo até dez vezes, no máximo, selecionando o botão *Reiniciar*. Após cumprido o total de tentativas, os botões *Jogar* e *Reiniciar* serão desabilitados e o usuário poderá retornar a página principal. A qualquer momento do jogo o usuário poderá utilizar um tutorial com definições relativas aos jogos não-cooperativos para inferir sobre os conceitos de jogos não-cooperativos como: tomada de decisão racional, estratégia pura, estratégia mista, estratégia dominante e estratégia dominada. Contudo, ao final do jogo, o usuário poderá ler na caixa de diálogo *Explicação do Jogo* (Figura 5.4) a exibição da moral do jogo exibida em uma tela e logo depois selecionar o botão “OK” da caixa de diálogo *Fim do Jogo* (Figura 5.5) para terminar.

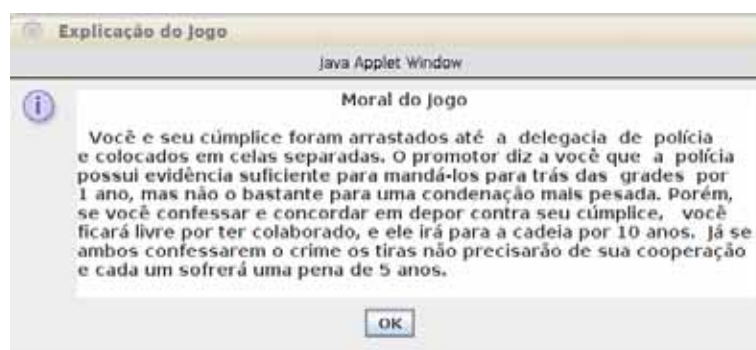


Figura 5.4: Tela de moral do jogo do módulo JDP.



Figura 5.5: Tela de finalização do módulo JDP.

Depois de finalizado o jogo, os botões de *Reiniciar* e *Jogar* são desabilitados e o usuário poderá pressionar a seta verde de retorno ao menu principal (no canto superior direito da tela), conforme mostrado na Figura 5.6 ou sair do programa pela finalização do próprio navegador.



Figura 5.6: Tela de final do módulo JDP.

A opção *Ajuda* (Apêndice A) disposta no centro da barra de menu da tela auxilia o usuário na execução do módulo JCE, assim como ter acesso ao recurso de referências específicas para jogos não-cooperativos, como mostrado na Figura 5.7.

Referências Específicas

FILOSOFIA&IDEIAS. Teoria do jogo. Disponível em: <<http://www.oocities.com/athens/4539/dilema.html>>. Acesso em 20 Nov 2008.

MYERSON, R. B. Game Theory: analysis of conflict. Cambridge: Harvard University Press, p.568, 1991.

POUNDSTONE, W. Prisoner's dilemma. Anchor Books. New York, 1992.

PRISONERS DILEMMA COMPETITION. Celebrating the 20TH anniversary. Disponível em: <<http://www.prisoners-dilemma.com/>>. Acesso em 07 Out 2008.

Figura 5.7: Tela com as referências específicas do módulo JDP.

5.3 O Módulo JCE

O módulo JCE possui uma tela principal, Figura 5.8, na qual o usuário poderá executar o jogo coalizacional entre empresas.



Figura 5.8: Tela inicial do módulo JCE.

No módulo JCE, a barra de menu *JOGO* descreve três itens de menu que são, respectivamente, *Descrição*, *Gerar Configurações Iniciais* e *Sobre*. O item de menu *Descrição* é para

informar ao usuário a finalidade geral do jogo. No item *Gerar Configurações Iniciais* descreve-se a tabela de benefícios dos jogadores. No item *Sobre* constam informações sobre os autores do trabalho. Os itens *Descrição* e *Sobre* são apresentados, respectivamente, nas figuras 5.9 e 5.10, a seguir.

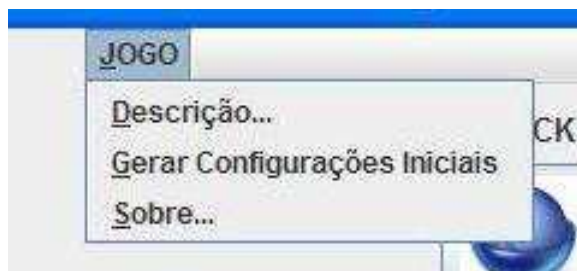


Figura 5.9: Opções do menu JOGO.

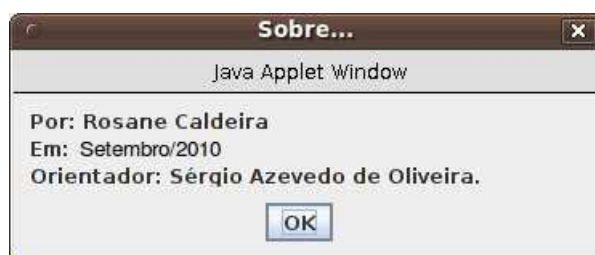


Figura 5.10: Informações sobre os autores.

A área de atuação de cada jogador, no triângulo equilátero, é definida acionando-se os botões correspondentes as seguintes empresas fictícias: Global Provider, Signal Internet e Brazilian Internet (Figura 5.11); onde a borda de cada botão torna-se enfatizada em cores distintas para identificação da reta de atuação que cada jogador ocupa no triângulo equilátero. Assim, o jogador 1 representa a base do triângulo na cor verde, o jogador 2 representa o lado direito do triângulo na cor azul e o jogador 3 representa o lado esquerdo do triângulo na cor vermelha (Figura 5.12).



Figura 5.11: Tela com empresas fictícias.

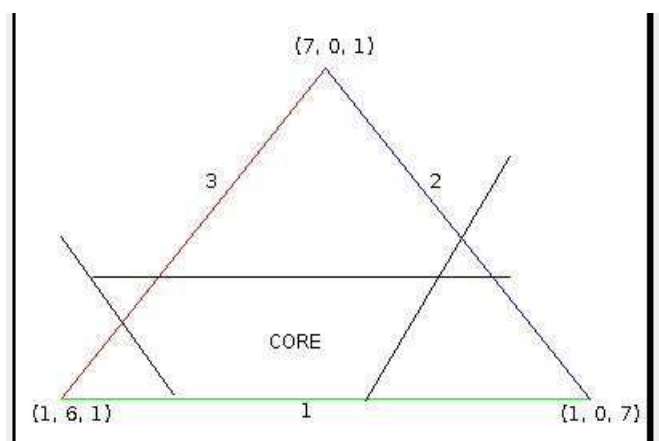


Figura 5.12: Tela de área de atuação dos jogadores (empresas).

No sub-item *Gerar Configurações Iniciais*, do menu *JOGO* é inicializada a tabela de benefícios dos jogadores, isto é, os pagamentos que eles recebem quando atuam sozinhos e quando atuam em acordos (coalizões). A Figura 5.13 mostra a tela com as configurações iniciais do programa, após ser acionado o respectivo sub-item.

HISTÓRICO DE			
Tabela de Benefícios: (\$)			
	{1}	{2}	{3}
v {1}	1	4	3
v {2}	4	0	5
v {3}	3	5	1

Figura 5.13: Tela com configurações iniciais.

No lado esquerdo da parte central da interface estão as opções de coalizão (acordos) entre os três participantes do jogo. Cada jogador do grupo formado envia a sua posição para o triângulo que futuramente será validada. Nas caixas de combinações, referentes aos jogadores que fazem acordos, estão os possíveis valores a serem escolhidos que posicionam o jogador no triângulo. A caixa de texto na cor amarela é calculada e representa o restante que falta para atingir o valor da grande coalizão. Acionando-se o botão *Registrar Imputações*, o programa é processado e as imputações armazenadas na memória, como é mostrado na Figura 5.14.

OPÇÕES DE COALIZÃO

Jogadores {1 e 2}:

Enviar posição para o jogador 1

x1: 1 x2: 5 x3: 2

Enviar posição para o jogador 2

x1: 2 x2: 2 x3: 4

Jogadores {1 e 3}:

Enviar posição para o jogador 1

x1: 1 x2: 6 x3: 1

Enviar posição para o jogador 3

x1: 1 x2: 6 x3: 1

Jogadores {2 e 3}:

Enviar posição para o jogador 2

x1: 5 x2: 2 x3: 1

Enviar posição para o jogador 3

x1: 2 x2: 5 x3: 1

Registrar Imputações

Figura 5.14: Tela de registro das imputações.

Após a configuração dos parâmetros do jogo, registro das opções de coalizão e registro das imputações, então é hora de jogar. Ao selecionar o botão *Jogar*, o jogo coalizacional será processado e então serão exibidos os seguintes resultados: o registro das imputações estáveis, ou seja, que atingiram a melhor região, a do Core, e o registro das imputações instáveis que correspondem a pior região escolhida, como é mostrado na Figura 5.15.

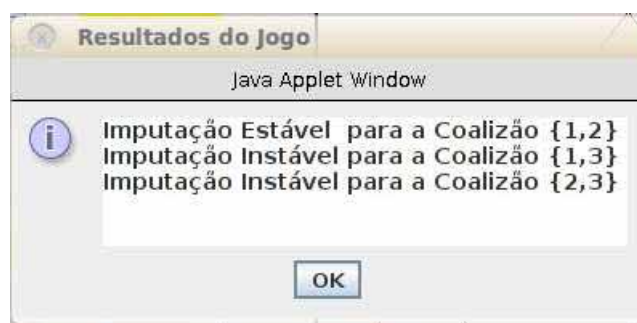


Figura 5.15: Tela de resultado das imputações estáveis e instáveis.

Os resultados para as imputações estáveis ou instáveis são calculados de acordo com os valores escolhidos, conforme é mostrado na Figura 5.16, comparados com a região do Core.

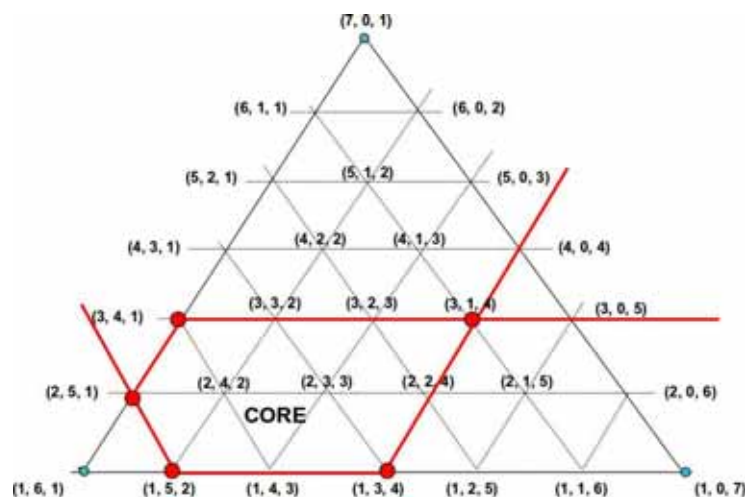


Figura 5.16: Dados para o cálculo da estabilidade ou instabilidade das imputações.

O usuário (aluno) poderá reiniciar o jogo até dez vezes no máximo selecionando o botão *Reiniciar*, (Figura 5.17), e depois de cumprido o total de dez tentativas os botões: *Registrar Imputações*, *Jogar* e *Reiniciar* serão desabilitados e o usuário poderá retornar a página principal.

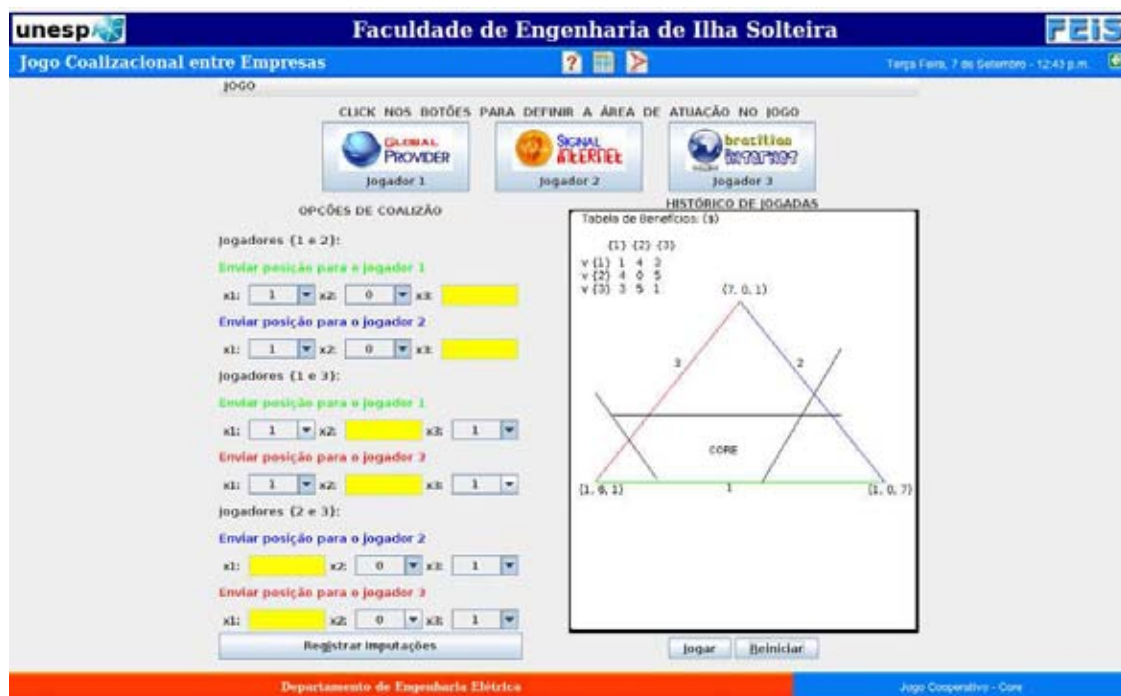


Figura 5.17: Tela de reinicialização das imputações.

Ao final do jogo, o usuário poderá ler na caixa de diálogo *Explicação do Jogo* a respeito da moral do jogo, como é mostrado na Figura 5.18. E a seguir o jogo é finalizado apresentando a tela da Figura 5.19.

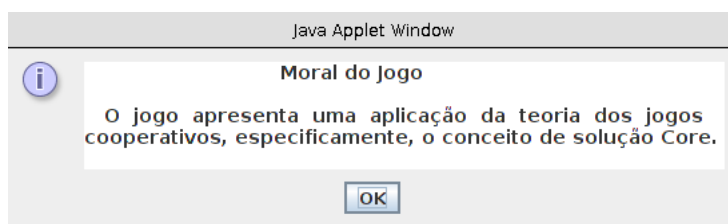


Figura 5.18: Tela de moral do jogo do módulo JCE.



Figura 5.19: Tela de finalização do módulo JCE.

A qualquer momento do jogo o usuário poderá utilizar um tutorial com os conceitos básicos relativos aos jogos cooperativos. Neste tutorial o usuário poderá inferir sobre os conceitos de jogos cooperativos, como: racionalidade individual, racionalidade coletiva, racionalidade coalizacional, imputações, imputações estáveis e instáveis, assim como o conceito de solução Core que é utilizado neste módulo.

O usuário tem a sua disposição ainda a opção *Ajuda* (Apêndice B) que o auxilia na execução do módulo JCE, bem como ainda a opção *Referências*, aonde o mesmo tem acesso as referências específicas para jogos cooperativos, como mostrado na Figura 5.20.

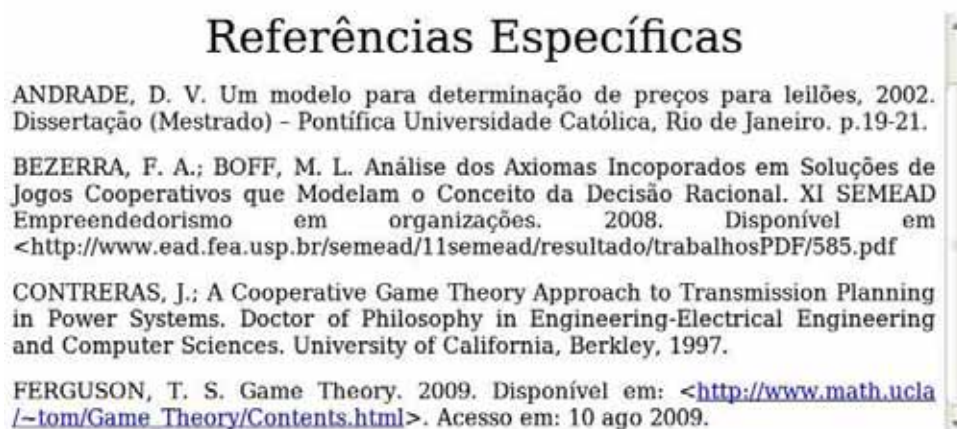


Figura 5.20: Tela com as referências específicas do módulo JCE.

6 Conclusões e Sugestões para Trabalhos Futuros

6.1 Conclusões

A revisão bibliográfica sobre a teoria dos jogos, descrevendo as aplicações e os conceitos relacionados a jogos não-cooperativos e cooperativos foi uma etapa inicial do presente estudo. Esta etapa do trabalho possibilitou a incorporação de tutoriais, geral e específicos, à interface desenvolvida, que apresenta ao usuário uma visão geral da teoria dos jogos.

Outra etapa, foi a escolha e análise de desempenho de uma linguagem apropriada para incorporar recursos necessários no desenvolvimento da interface com objetivos educacionais. Aqui, a escolha da linguagem Java mostrou-se eficaz para os objetivos pretendidos pois, com seus recursos e dentro do paradigma de programação orientada a objetos, possibilitou a criação de diversas funcionalidades para a interface proposta. Esta interface apresenta diversos recursos computacionais que permitem ao usuário final interagir com os conceitos da teoria dos jogos, de uma maneira amigável (via jogos), em que os conceitos teóricos são gradativamente incorporados ao conhecimento do mesmo através da possibilidade de diversas simulações em cada jogo.

Na sequência do trabalho, fez-se uma ampla abordagem das principais características que devem ter um programa para ser utilizado como um simulador computacional de auxílio ao usuário na compreensão e utilização de alguns dos conceitos teóricos abordados da teoria de jogos. Isto foi feito, via exemplos, dentro dos jogos não-cooperativos e cooperativos respectivamente, com o desenvolvimento dos módulos: Jogo do Dilema do Prisioneiro e Jogo Coalizacional entre Empresas. Desenvolveu-se assim, programas na forma de *Java applets* para um sistema de auxílio ao ensino baseado na Web, objetivando maiores facilidades para os estudantes aumentarem e melhorarem seu ritmo de aprendizado para aqueles conceitos em que normalmente têm maiores dificuldades. O processo de desenvolvimento como um todo, baseado nos padrões de engenharia de software, servirá como base para o desenvolvimento de novos

módulos para compor um *framework* de simulações computacionais, assim como a referência inicial para o desenvolvimento de outras interfaces que se pretende disponibilizar, via Web, nas páginas do LLPP-Laboratório de Linux e Processamento Paralelo.

A interface ENGTJ, assim denominada pois pretende-se futuramente incorporar a mesma, programas específicos de engenharia que utilizam conceitos da teoria dos jogos. É uma proposta de trabalho que agrega novos paradigmas no desenvolvimento de recursos computacionais voltados para o ensino de teorias mais complexas, como é o caso da teoria dos jogos. Assim, pretende-se que a interface, desenvolvida neste trabalho, possa ajudar o usuário no processo de ensino-aprendizagem de alguns dos conceitos básicos da teoria dos jogos, em particular.

6.2 Sugestões para Trabalhos Futuros

Para continuidade deste trabalho, com relação aos aspectos de desenvolvimento de novos recursos computacionais, sugere-se, por exemplo, o uso de tecnologias de conteúdo dinâmico. Estas técnicas permitem uma criação de páginas Web que poderiam ser atualizadas dependendo do tipo de usuário (jogador); poderia haver também armazenamento dinâmico de dados, e com isso pode-se implementar recursos de atualização destes dados em *runtime*, além de outras funcionalidades. Todas estas sugestões trariam, uma vez implementadas, novos recursos para a interface gráfica proposta. No Anexo A, um texto explicativo, apresenta a evolução destas tecnologias.

Referências Bibliográficas

- ANDRADE, D. V. *Um modelo para determinação de preços para leilões*. 2002. 69f. Dissertação (Mestrado) — Pontifícia Universidade Católica, Rio de Janeiro, 2002.
- BENGU, G.; SWART, W. A computer-aided, total quality approach to manufacturing education in engineering. *IEEE Transactions on Education*, New York, v.39, n. 3, p. 415–422, 1996.
- BERTRAND, R. G. *Equilíbrio de Nash en mercados eléctricos: técnicas de modelado y análisis en ingeniería*. Ciudad Real: Universidad Castilha - La Mancha, 2005.
- BORTOLOSSI, H. J.; GARBAGIO, G.; SARTINI, B. Uma introdução à teoria econômica dos jogos. COLÓQUIO BRASILEIRO DE MATEMÁTICA, 26, 2007, Rio de Janeiro. *Colóquio Brasileiro de Matemática*. Rio de Janeiro: IMPA, 2007.
- CHRISTOPH, R. *Engenharia de software para software livre*. 2004. 118f. Dissertação (Mestrado) — Pontifícia Universidade Católica, Rio de Janeiro, 2004.
- CONTRERAS, J. A *Cooperative Game Theory Approach to Transmission Planning in Power Systems*. Tese (Doutorado) — Berkley: University of California, 1997.
- DE VASCONCELLOS, D. R. *Análise de estratégia utilizando verificação formal de modelos*. 2003. 128f. Dissertação (Mestrado) — Pontifícia Universidade Católica, Rio de Janeiro, 2003.
- DEBEBE, K.; RAJAGOPALAN, V. A learning aid for power eletrronics with knowledge based components. *IEEE Transactions on Education*, New York, v. 38, n. 2, p. 171–176, 1995.
- DEITEL, H.; DEITEL, P. *Java: como programar*. Porto Alegre: Pearson Education Documents, 2005.
- ECLIPSE. 2007. Disponível em: <<http://www.eclipse.org/>>. Acesso em: 14 março 2009.
- FERGUSON, T. S. *Game theory*. 2009. Disponível em: <http://www.math.ucla/~tom/Game_Theory/Content.html/>. Acesso em: 10 ago. 2009.
- FIELDS, D. K.; KOLB, M. A. *Desenvolvendo na Web com Java Server Pages*. Rio de Janeiro: Ciência Moderna, 2000.
- GAUTHIER, D. *Morals by agreement*. Oxford: Oxford University Press, 1987.
- GILLIES, D. B. *Some theorems on n-person games*. Tese (Doutorado) — Princeton University Press, New Jersey, 1953.
- GONÇALVES, F. A. S.; CANESIN, C. A. Java applets para um software educacional distribuído em eletrônica de potência. *SBA: Controle e Automação*, Campinas, v. 13, n. 3, p. 314–316, 2002.

HWANG, G. A tutorial strategy supporting system for distance learning on computer networks. *IEEE Transactions on Education*, New York, v. 42, n. 4, p. 343–343, 1999.

KAVKA, G. *Hobbesian moral and political theory*. New York: Princeton University Press, 1986.

LEWIS, D. *Convention: a philosophical study*. [S.l.]: Willey-Blackwell, 1969.

MAGALHÃES, J.; LEITE, M. Orientação a objetos. In: SIMPÓSIO TECNOLOGIA JAVA NO SERPRO, 1, 2003, Rio de Janeiro. *Curso Básico de Programação JAVA*. Rio de Janeiro: SERPRO, 2003.

MORTON, D.; MASCHLER, M. *The kernel of a cooperative game*. [S.l.]: Naval Research Logistics Quarterly, 1965.

NASH, J. F. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, United States of America, v. 36, n. 1, p. 48–49, 1950.

OMG. *Object Management Group*. 2007. Disponível em: <<http://www.omg.org/>>. Acesso em: 8 outubro 2010.

PRESSMAN, R. S. *Engenharia de Software*. São Paulo: McGraw-Hill, 2006.

QUINE, W. V. O. *Truth by convention*. [S.l.]: Philosophica Essays for A.N. Whitehead Russel and Russel Publishers, 1967.

RUMBAUGH, B. J. *Visual modeling with rational rose and UML*. New York: Addison Wesley Longman, Inc., 1998.

SMITH, J. M. *Evolution and the theory of games*. Cambridge: Cambridge University Press, 1982.

SMITH, J. M.; PRICE, G. R. The logical of animal conflict. *Nature*, London, n. 246, p. 15–18, 1973.

TUCKER, A. W. On jargon: the prisoner's dilemma. *UMAP Journal*, v. 1, p. 101, 1980.

TÉLLEZ, C. A. Teoria dos jogos e relações internacionais. *Observatório Tempo Presente*, Rio de Janeiro, n. 1, p. 10, 2004.

VON NEUMANN, J.; MORGENSTERN, O. *Theory of games and economics behaviour*. Cambridge: Harvard University Press, 1944.

WIE, C. R. Educational java applets in solid-state material. *IEEE Transactions on Education*, New York, v. 46, n. 1, p. 12–13, 1998.

WIKISLICE. *The free encyclopedia*. 2008. Disponível em: <http://en.wikipedia.org/wiki/Wikipedia:WikiProject_wikislice>. Acesso em: 12 jul. 2008.

ZERMELO, E. *Über eine anwendung der mengenlehre auf die theorie des schachpiels*, Atas do Congresso Internacional de Matemáticos, v. 2, p. 501–504, 1913.

ZUGMAN, F. *Uma introdução à ciência que vê a vida como uma sequência de jogos*. 2007. Disponível em: <<http://www.iced.org.br/artigos/>>. Acesso em: 18 jul. 2007.

APÊNDICE A – Ajuda para o Jogo do Dilema do Prisioneiro

A.1 Desenvolvimento e Organização

O jogo do dilema do prisioneiro é executado quando o arquivo *Jdp.class* é carregado no navegador da Web e visualizado como documento HTML. O navegador executa o arquivo *engdile.html* e dispara o carregador de classe para carregar as informações utilizadas no módulo JDP.

A.1.1 O módulo JDP

A opção *Jogo do Dilema do Prisioneiro (Não-cooperativo)* conduz o usuário à interface gráfica JDP, na qual ele poderá executar o jogo dilema do prisioneiro.

No lado superior esquerdo da tela está o jogador Al, que pode escolher entre duas estratégias que são: confessar ou não-confessar o crime. No lado superior direito estão as mesmas estratégias para o jogador Bob. As estratégias de cada um dos jogadores encontram-se na caixa de combinação divididas entre as seguintes opções de modalidade estratégica, e são:

- randômico: aleatoriamente faz movimentos randômicos alternando entre “CONFESSAR” ou “NÃO-CONFESSAR” com 50 por cento de probabilidade;*
- confessar: sempre joga “CONFESSAR” o crime;*
- não-confessar: sempre joga “NÃO-CONFESSAR” o crime;*
- grim: inicia “CONFESSANDO”, mas depois da primeira que “NÃO-CONFESSA” então muda sempre para “NÃO-CONFESSAR”.*

Todas as vezes que o jogo é iniciado a modalidade estratégica “randômico” aparece como padrão (“default”). A caixa de combinação das opções de modalidade estratégica encontra-se vinculada a uma caixa de texto que descreve a função de cada opção. O número de interrogatórios para ambos os jogadores (prisioneiros) varia de 1 a 10. E o histórico de processamento do jogo encontra-se na área de texto na parte central da tela, onde serão exibidos: o número total de rodadas do jogo, a lista de rodadas, a modalidade estratégica de cada jogador, a lista das penas mediante o tipo de estratégia adotada, o somatório das penas, a média das penas, a média das penas em anos, meses e dias. Por último, já no rodapé da tela estão os botões de *Jogar* e de *Reiniciar* o jogo.

Antes de executar o jogo, é preciso configurá-lo com a estratégia desejada e também com o número de rodadas (interrogatórios) pelos quais os prisioneiros irão passar. Após a configuração dos parâmetros do jogo, então é hora de jogar. Ao selecionar o botão *Jogar* o jogo do dilema do prisioneiro será processado e então serão exibidos os seguintes resultados: o cálculo de soma das penas, a média das penas e as penas em anos, meses e dias de cada prisioneiro. O usuário poderá reiniciar o jogo até dez vezes no máximo selecionando o botão *Reiniciar*. Após cumprido o total de tentativas, os botões *Jogar* e *Reiniciar* serão desabilitados e o jogo será encerrado. Contudo, o usuário poderá ler na caixa de diálogo *Explicação do Jogo* a respeito da moral do jogo podendo assim inferir sobre os conceitos apresentados na simulação e em seguida retornar a página principal ou sair do navegador.

A qualquer momento do jogo o usuário poderá utilizar um tutorial com os conceitos básicos relativos aos jogos não-cooperativos.

APÊNDICE B – Ajuda para o Jogo Coalizacional entre Empresas

B.1 Desenvolvimento e Organização

O jogo coalizacional entre empresas é executado quando o arquivo *Jce.class* é carregado no navegador da Web e visualizado como documento HTML. O navegador executa o arquivo *engcoal.html* e dispara o carregador de classe para carregar as informações utilizadas no módulo JCE.

B.1.1 O módulo JCE

A opção *Jogo Coalizacional entre Empresas (Cooperativo)* permite ao usuário executar a execução do jogo coalizacional entre empresas através da interface gráfica JCE. Na barra de menu *JOGO*, há três sub-itens de menu que são: *Descrição*, *Gerar Configurações Iniciais* e *Sobre*. O sub-item *Descrição* é para informar ao usuário a finalidade geral do jogo. Através do sub-item *Gerar Configurações Iniciais*, são geradas as configurações da função benefício do jogo. E o sub-item de menu *Sobre* contém informações sobre os autores. A área de atuação de cada jogador no triângulo equilátero é definida acionando-se os botões das seguintes empresas fictícias: Global Provider [Jogador 1], Signal Internet [Jogador 2] e Brazilian Internet [Jogador 3], respectivamente, onde a borda de cada botão é colorida de acordo com a reta de cada jogador no jogo. Assim, o Jogador 1 representa a base do triângulo na cor verde, o Jogador 2 representa o lado direito do triângulo na cor azul e o Jogador 3 representa o lado esquerdo do triângulo na cor vermelha. No lado esquerdo da parte central da interface estão as opções de coalizão (acordos) entre as empresas participantes do jogo. Cada jogador da coalizão formada envia sua posição para o triângulo que futuramente terá sua posição validada. Nas caixas de combinações, referentes aos jogadores que fazem acordos, estão os possíveis valores a serem escolhidos que posicionam o jogador no triângulo.

A caixa de texto na cor amarela é calculada e representa o restante que falta para a coalizão formada atingir o valor da grande coalizão. Feitas as escolhas de posição então é hora de jogar. Quando o usuário acionar no botão *Registrar Imputações* o programa será processado e as imputações computadas e armazenadas na memória para futura validação. A cada registro de imputações pode-se acionar o botão *Reiniciar* até no máximo dez vezes. O botão *Jogar* pode ser acionado a qualquer momento e faz com que seja gerado o resultado das imputações estáveis e instáveis do jogo.

O usuário poderá recomeçar o jogo até dez vezes no máximo selecionando o botão *Reiniciar*. Depois de atingido o número total de tentativas os botões de comando do jogo serão desabilitados e o usuário não poderá interagir mais com o programa. Contudo, o usuário poderá ler na caixa de diálogo *Explicação do Jogo* a respeito da moral do jogo podendo inferir sobre os conceitos apresentados na simulação e retornar a página principal ou sair do navegador.

A qualquer momento do jogo o usuário poderá utilizar um tutorial com os conceitos básicos relativos aos jogos cooperativos, tais como: racionalidade individual e coletiva, racionalidade coalizacional, imputações estáveis e instáveis, assim como o conceito de solução *core*.

ANEXO A – Evolução das Tecnologias de Conteúdo Dinâmico (FIELDS; KOLB, 2000)

A.1 Introdução

Para as solicitações mais simples da Web, um navegador solicita um documento HTML e o servidor da Web encontra o arquivo correspondente e o devolve. Se o documento HTML incluir qualquer imagem, o navegador irá submeter solicitações para os documentos de imagem também. Todas estas solicitações são para arquivos estáticos, ou seja, os documentos que são solicitados nunca mudam dependendo de quem os solicitou, não se altera também quando os mesmos são solicitados, ou ainda que parâmetros adicionais são incluídos com a solicitação. Novas versões do documento podem ser colocadas no servidor, mas a qualquer momento cada solicitação por aqueles documentos retorna exatamente os mesmos resultados. Em tais situações, o servidor da Web precisa apenas localizar o arquivo correspondente ao documento solicitado e responder ao navegador com o conteúdo daquele arquivo.

No entanto, a maioria dos dados fornecidos através da Web hoje tem uma natureza dinâmica. Os preços mais atualizados de ações da bolsa de valores e as mais recentes notícias meteorológicas podem ser visualizados. As mensagens pessoais de e-mail de um usuário e a agenda de compromissos podem ser gerenciados. Os consumidores podem adicionar conteúdo àquele elemento principal de *e-commerce*, o carrinho de compras “on-line”, ao clicando numa imagem do item que se deseja adquirir. Todos estes dados são dinâmicos por natureza, porque a informação está baseada em constante mudança, ou porque devem ser personalizados para cada visualizador individual, ou ambos.

O conteúdo dinâmico da Web, então, exige que o servidor da Web faça algum processamento adicional da solicitação correspondente, a fim de gerar uma resposta personalizada. Além do URL da solicitação, o formulário desta resposta personalizada pode depender de valores de parâmetros adicionais incluídos na solicitação. Alternadamente, ele pode se basear na data e na hora, o local na rede a partir do qual a solicitação foi feita, ou em alguma representação da

identidade do usuário que está fazendo a solicitação. Na verdade, os detalhes exatos da resposta podem depender da combinação de alguns ou de todos estes fatores.

A.2 “Java Server Pages” - JSP

“Java Server Pages” - JSP - é uma tecnologia baseada em Java que simplifica o processo de desenvolvimento de sites dinâmicos. Com JSP, os projetistas e os programadores podem rapidamente incorporar elementos dinâmicos nas páginas usando Java embutido e algumas “tags” de marcação simples. Estas “tag(s)” fornecem ao projetista de HTML um meio de acessar dados e lógica de negócios armazenados em objetos Java sem que seja preciso dominar as complexidades do desenvolvimento de aplicações.

Considere a JSP como um tipo de linguagem de criação de “scripts” no servidor, embora ela opere de maneira bem diferente. JSP são arquivos de texto, normalmente com a extensão *.jsp*, que substituem as páginas HTML tradicionais. Os arquivos JSP contém HTML tradicional junto com código embutido que permitem que o projetista de páginas acesse dados do código de Java rodando no servidor. Quando a página é solicitada por um usuário e processada pelo servidor de “HyperText Transport Protocol” (HTTP), a parte HTML da página é transmitida. No entanto, as partes de código da página são executadas no momento em que a solicitação é recebida e o conteúdo dinâmico gerado por este código é unido na página antes de ser enviado para lógica de programação contida no código.

A.3 “Common Gateway Interface” - CGI

Os servidores de HTTP mais antigos não incluíam qualquer mecanismo embutido para gerar respostas dinamicamente. Ao invés disso, foram fornecidas interfaces para chamar outros programas para traduzir solicitações no conteúdo de “runtime”. O primeiro padrão para conteúdo dinâmico da Web se baseava na “Common Gateway Interface”, ou CGI, que especificava um mecanismo para o servidor da Web passar as informações da solicitação para programas externos, que eram então processados pelo servidor para gerar respostas no tempo de execução. A linguagem Perl é uma linguagem popular para escrever programas de CGI, mas os códigos de CGI podem ser escritos em qualquer linguagem que possa ser chamada como um programa independente pelo servidor de HTTP. Um programa de CGI poderia ser escrito em qualquer linguagem de criação de “scripts” aceita pelo sistema operacional local. Alternativamente, ele poderia ser escrito em C e compilado no código de objetos nativos. Os programas de CGI

poderiam até ser escritos como aplicações de Java.

Quando a Sun Microsystems introduziu pela primeira vez a tecnologia de Java à comunidade de computação, foi no contexto de pequenos programas, chamados de *applets*, que podem ser transmitidos pela internet e executados dentro de navegadores da Web. Desde o início, Java podia também ser usado para escrever aplicações independentes, mas os programas interativos rodando dentro de navegador certamente receberam a maioria da atenção inicial.

Os programas de CGI baseados em Java apareceram pela primeira vez logo depois que Java foi disponibilizado para o público em 1995. Foi eventualmente reconhecido que os benefícios da plataforma de Java se aplicavam igualmente ao servidor e ao cliente, e hoje Java no servidor tem um papel importante na evolução contínua da plataforma de Java.

A.4 Linguagens de “Script”

A geração de conteúdo dinâmico exige que o servidor processe solicitações no tempo de execução, a fim de construir uma resposta apropriada específica da solicitação. As instruções são necessárias a fim de realizar este processamento, logo, em um nível ou outro, alguma programação é necessária. Como resultado, muitos dos sistemas de conteúdo dinâmico mais populares, como “ColdFusion” da Allaire, “Active Server Pages” da Microsoft, “Server-Side JavaScript” da Netscape e PHP (pré-processador de hipertexto de código aberto) permitem que conteúdo dinâmico seja especificado usando-se linguagens de criação de “scripts”.

O uso das linguagens de criação de “scripts” é uma escolha particularmente apropriada porque os programadores estão acostumados a mudanças rápidas ao testar suas páginas: assim que o HTML em uma página estática é modificado, os resultados daquela modificação podem ser visualizados em um navegador. Ao se basear em linguagens de criação de “scripts” que não necessitam de um ciclo longo de editar e compilar “links” antes que qualquer código possa ser rodado, tais ferramentas de conteúdo dinâmico fornecem o mesmo “feedback” imediato com o qual os programadores se acostumaram com a HTML.

A.4.1 “ColdFusion”

As diferenças principais entre os sistemas modelo, residem nas suas linguagens de criação de “scripts” e as capacidades fornecidas nas mesmas. O “ColdFusion” fornece um conjunto de “tags” do tipo HTML que inicialmente visava embutir consultas de banco de dados em páginas da Web; mas desde então, foram estendidas para suportar uma ampla variedade de fontes de

dados para geração de conteúdo dinâmico. A vantagem da adoção de “tags” do tipo HTML é a existência de um único estilo consistente de sintaxe ao longo da página; as “tags ColdFusion” são confortáveis para os projetistas porque elas se parecem com outras “tags” presentes no documento. O “ColdFusion” suporta tanto as plataformas UNIX quanto Microsoft Windows.

A.4.2 “Server-Side Javascript” - SSJS

“Server-Side JavaScript” (SSJS) é como uma linguagem de criação de “scripts”. SSJS é uma extensão da linguagem JavaScript básica que também é a base para a linguagem JavaScript no cliente popular, usada para criar “scripts” para navegadores da web. A SSJS adiciona recursos embutidos para suporte de e-mail e banco de dados, gerenciamento de sessão e interoperabilidade com classes Java no servidor. Assim como o código Java compilado, SSJS compilado não é específico de plataforma em relação a hardware ou sistema operacional.

A.4.3 “Hypertext Preprocessor” - PHP

Assim como JavaScript, PHP emprega uma sintaxe do tipo C e fornece forte suporte para acesso de banco de dados e correspondência de padrão. Extensões para se comunicar com outros recursos de rede, como servidores de diretório e correspondência eletrônica, também estão disponíveis.

No entanto, diferentemente da maioria dos outros sistemas de conteúdo dinâmico agora disponíveis, o PHP é um produto de código aberto. Como acontece com os outros produtos do gênero, como o sistema operacional Linux e o servidor de HTTP Apache, o PHP não é um produto comercial. Ao invés disso, ele é o resultado de contribuições de uma comunidade de programadores interessados, contribuindo gratuitamente e suportando sua base de código. Um resultado importante desta natureza de código aberto é que o mesmo é utilizado no Windows NT, em diversos sistemas operacionais UNIX e com diversos servidores de HTTP como: Apache, “Internet Information Server” (IIS) e Netscape Enterprise Server.

A.5 “Java Servlets”

Pela importância da geração de conteúdo dinâmico para o desenvolvimento da Web, foi natural que a Sun propusesse extensões para Java em seu domínio. Da mesma forma que a Sun introduziu *applets* como pequenas aplicações baseadas em Java para adicionar funcionalidade interativa aos navegadores, em 1996 a Sun introduziu *servlets* como pequenas aplicações

baseadas em Java para adicionar funcionalidade dinâmica a servidores da Web. Os *servlets* de Java têm um modelo de programação similar aos “scripts” de CGI, a medida em que eles recebem uma solicitação de HTTP de um navegador como “input” e espera-se que localizem e/ou construam o conteúdo apropriado para a resposta do servidor.

Diferentemente dos programas tradicionais de CGI que necessitam da criação de um novo processo para tratar de cada nova solicitação, todos os *servlets* associados com um servidor da Web rodam dentro de um único processo. Este processo roda uma “Java Virtual Machine” (JVM), que é o programa específico de plataforma para rodar programas de Java compilados (compatível com várias plataformas).

Já que a JVM persiste além da vida de uma única solicitação, os *servlets* também podem evitar muitas operações demoradas, como conexão a um banco de dados, ao compartilhá-los entre todas as solicitações. Ao mesmo tempo, pelo fato dos *servlets* serem escritos em Java, eles se aproveitam de todos os benefícios da plataforma de Java básica: um modelo de programação orientado a objetos, gerenciamento automático de memória, portabilidade compatível com várias plataformas e acessam a rica coleção de APIs de Java agora disponíveis para acessar banco de dados, servidores de diretório, recursos de rede e assim por diante.

A.6 “Java Servlet Pages” - JSP/Servlet

Incorporar conteúdo dinâmico deve envolver algum tipo de programação para descrever como determinado conteúdo é gerado. Minimizar a necessidade de programação é frequentemente um objetivo desejável. Combinar este objetivo com o objetivo da SUN por um suporte robusto e repleto de recursos para Java no servidor, um sistema modelo baseado em Java, é um resultado natural.

JSP/Servlet é um tanto quanto híbrido entre os sistemas modelo, porque suporta dois estilos diferentes para adicionar conteúdo dinâmico a páginas da Web. “Scripts” podem ser embutidos em páginas JSP contendo o código de programação propriamente dito. No caso de JSP/Servlet, este código de programação é tipicamente Java.

JSP/Servlet aceita um conjunto de “tags” do tipo HTML que interagem com objetos Java no servidor, sem necessidade que código de Java bruto apareça na página. Na verdade, a especificação JSP/Servlet leva esta capacidade um passo a frente, ao fornecer um mecanismo de extensão de “tags” que permite que os programadores criem bibliotecas de “tags” personalizadas, que podem ser carregadas em uma página JSP/Servlet.

A.6.1 Benefícios de JSP/Servlet

JSP/Servlet oferece diversos benefícios como um sistema para a geração de conteúdo dinâmico. Sendo uma tecnologia baseada em Java, ela se aproveita de todas as vantagens que a linguagem Java fornece em relação a desenvolvimento e acionamento. Como uma linguagem orientada a objetos com forte digitação, encapsulamento, tratamento de exceções e gerenciamento de memória automática, o uso de Java conduz a uma produtividade aumentada do programador e a código mais robusto. Pelo fato de haver uma API padrão publicada para JSP/Servlet, e pelo fato do “bytecode” de Java compilado ser portátil através de todas as plataformas que aceitam uma JVM, o uso de JSP/Servlet não restringe ao uso de uma plataforma de hardware, sistema operacional ou software de servidor específico.

A.7 A Edição Corporativa de JSP/Servlet e Java 2

JSP/Servlet é agora uma parte integrante do desenvolvimento de aplicações baseadas na Web usando-se Java. Pela sua habilidade em separar a lógica de implementação da apresentação, ao combinar texto de marcação padrão com elementos de criação de “scripts” e componentes orientados a objetos, a JSP/Servlet fornece uma excelente tecnologia “front-end” para aplicações que são lançadas na Web.

A.7.1 Edições de plataforma Java

Em junho de 1999, a Sun Microsystems anunciou que a plataforma de software Java 2 seria dividida em três edições, visando diferentes categorias de plataformas de acionamento de hardware. O “Java Runtime Environment” tradicional, ou JRE, que contém todas as classes básicas na especificação de linguagem formal, incluindo as classes de interface gráfica com o usuário, utilitários e rede padrão foi renomeado como “Java 2 Standard Edition” ou J2SE. A J2SE é indicada para plataformas de processamento de área de trabalho tradicionais.

Um subconjunto das classes básicas, indicadas para dispositivos de mão, processadores embutidos e os aparelhos de informações compreende a “Java 2 Micro Edition” ou J2ME. O objetivo da J2ME é fornecer um ambiente de Java como um mínimo e área de cobertura que, no entanto, aceite a visão de Java de código de programa “Write Once, Run Anywhere”.

No extremo oposto da J2ME, encontra-se a “Java 2 Enterprise Edition” (Edição corporativa de Java 2), ou J2EE. Ao invés de subtrair do núcleo de Java 2, como a micro edição, a J2EE empacota as classes de Java básicas com extensões projetadas para o desenvolvimento que fornece

um conjunto de abstrações padrão para acessar depósitos de dados corporativos, como banco de dados e servidores de diretórios, com apoio automático para gerenciamento de transações e “pool” de recursos.

No entanto, dada a complexidade inerente envolvida no projeto, construção e manutenção de aplicações corporativas de grande escala, a especificação da Sun de J2EE inclui um conjunto de diretrizes para desenvolvimento de software usando-se a plataforma J2EE. Estas diretrizes tornam a forma de um arquitetura de software base recomendada chamada “J2EE Application Model” (Modelo de Aplicação J2EE).

A.7.2 Aplicações baseadas na Web

O uso da Web como mecanismo preferencial para entrega de dados entre a aplicação e o usuário final é um elemento chave do Modelo de Aplicação de J2EE, que se utiliza do navegador como uma interface principal com o usuário para o software corporativo. A vantagem desta abordagem é que o navegador, nos poucos anos desde o nascimento da World Wide Web, estabeleceu um padrão de *facto* compatível com diversas plataformas e onipresente para acessar dados na rede. Quando uma aplicação se baseia no navegador para ser sua interface com o usuário, não é necessário que as aplicações sejam desenvolvidas e acionadas no servidor, os usuários finais automaticamente começam a usar a nova versão.

Para facilitar aplicações baseadas na Web, tanto *servlet* quanto JSP/Servlet são elementos necessários da especificação J2EE. Embora ambas as tecnologias possam ser usadas para gerar HTML dinamicamente para ser enviada para o navegador de um usuário final, apenas a JSP/Servlet realiza uma forte separação entre a lógica de apresentação envolvida na exibição de dados e a lógica de programação ou negócios usada para gerar aqueles dados. Esta separação significa que o projeto da interface com o usuário, incorporado em um conjunto de páginas JSP/Servlet, pode ser realizado independentemente do projeto do outro código que roda no servidor. Esta independência a aplicações é muito mais robusta, já que modificações em uma parte da aplicação geralmente não necessitam de modificações correspondentes em outras partes.

Classificação de Segurança Livre		Documento no.	
Data (mês e ano) Outubro/2010		Projeto no.	
Título e subtítulo Interface Gráfica no Contexto de Teoria dos Jogos sob a forma de Java Applets		No. do volume Único	
		No. da parte	
Entidade Executora (autor coletivo)		Autor(es) Rosane Caldeira	
Entidade patrocinada (cliente ou destinatário principal) PPGEE - UNESP Campus de Ilha Solteira			
Resumo Na resolução de problemas de engenharia existem diversas metodologias que podem ser aplicadas. Normalmente, estas envolvem novos conceitos, às vezes inéditos em termos de aplicação como é o caso da teoria dos jogos. Por outro lado, pesquisadores vêm buscando novas técnicas para o desenvolvimento de ferramentas computacionais para o auxílio no processo ensino-aprendizado em diferentes níveis. Nesta dissertação apresenta-se um simulador computacional que apresenta uma interface gráfica, denominada ENG TJ, como proposta para o ensino de conceitos básicos da teoria dos jogos, via Web. Os conceitos relacionados com a teoria dos jogos são apresentados sob forma de exemplos, no contexto dos jogos não-cooperativos e cooperativos, com os módulos do Jogo Dilema do Prisioneiro (JDP) e o Jogo Coalizacional entre Empresas (JCE). Estes módulos foram desenvolvidos utilizando-se a linguagem de programação Java, sob a forma de <i>Java applets</i> , com o auxílio do ambiente de desenvolvimento Eclipse. O processo de desenvolvimento de software baseado nos padrões de engenharia de software foi utilizado para criar a interface da aplicação, e apresenta uma série de técnicas e atividades que procuram dar suporte a definição de processos de desenvolvimento, levantamento e especificação de requisitos, projeto e testes. Como resultado, tem-se uma interface gráfica que permite ao usuário interagir através de jogos modulares referentes a teoria dos jogos, permitindo ao mesmo inferir alguns conceitos básicos abordados nesta teoria, suportada por tutoriais, geral e específicos.			
Palavras-chave Teoria dos Jogos. Jogos Cooperativos. Jogos Não-cooperativos. Simulador. Interface Gráfica. Java.			
No. da edição	No. de páginas 118	ISSN (para relatórios publicados)	Classificação (CDC ou CDD)
Distribuidor		Número de exemplares 5 (cinco)	Preço -
Observações Dissertação apresentada à Faculdade de Engenharia - UNESP, Campus de Ilha Solteira, como parte dos requisitos para obtenção do título de Mestre em Engenharia Elétrica.			