CENTRO UNIVERSITÁRIO DE BRASÍLIA – UNICEUB

CURSO DE ENGENHARIA DA COMPUTAÇÃO

HENRIQUE GRANDE NETO

SIMULAÇÕES INTERATIVAS DE EXPERIMENTOS LABORATORIAIS DE FÍSICA E MECÂNICA

Orientador: Professora Maria Marony Sousa Farias

BRASÍLIA

DEZEMBRO, 2011

HENRIQUE GRANDE NETO

SIMULAÇÕES INTERATIVAS DE EXPERIMENTOS LABORATORIAIS DE FÍSICA E MECÂNICA

Trabalho apresentado ao Centro Universitário de Brasília (UniCEUB) como pré-requisito para a obtenção de Certificado de Conclusão de Curso de Engenharia de Computação.
Orientadora: Maria Marony Sousa Farias

Brasília
Dezembro, 2011

HENRIQUE GRANDE NETO

SIMULAÇÕES INTERATIVAS DE EXPERIMENTOS LABORATORIAIS DE FÍSICA E MECÂNICA

Trabalho apresentado ao Centro Universitário de Brasília (UniCEUB) como pré-requisito para a obtenção de Certificado de Conclusão de Curso de Engenharia de Computação.
Orientadora: Maria Marony Sousa Farias

Este Trabalho foi julgado adequado para a obtenção do Título de Engenheiro de Computação, e aprovado em sua forma final pela Faculdade de Tecnologia e Ciências Sociais Aplicadas - FATECS.

_____

Prof. Abiezer Amarilia Fernandez
Coordenador do Curso

Banca Examinadora:

_____

Prof.ª Maria Marony Souza Farias, mestrado.
Orientadora

_____

Prof. Cléber da Silva Pinheiro, doutorado.
Centro Universitário de Brasília – UniCEUB

_____

Prof. Flávio Antônio Klein, mestrado.
Centro Universitário de Brasília – UniCEUB

_____

Prof.ª Vera Lúcia Farini Alves Duarte, mestrado.
Centro Universitário de Brasília – UniCEUB

DEDICATÓRIA

À minha família, maiores mestres que tenho.

Aos meus amigos e professores, pela companhia, amizade e lições aprendidas.

AGRADECIMENTOS

Aos meus familiares e namorada, por todo carinho, educação e compreensão. Obrigado por todo o apoio e incentivo, que me fez chegar até aqui. Obrigado por me ensinarem todos os valores da vida e o significado da palavra “família”.

Ao meu primo e melhor amigo David Ricardo do Vale Pereira, que me ensinou e ensina tudo que sei de programação. Ensinou-me também a gostar de engenharia, física e matemática. Levou-me à minha primeira aula de engenharia, quando eu ainda estava no ensino fundamental. Obrigado pelos conselhos e paciência.

Ao professor de física, orientador de monitoria e grande amigo Thiago de Miranda Leão Toribio, que com tanta presteza colaborou nesta monografia, desde a ideia inicial, os aprimoramentos e, certamente, o futuro deste trabalho. Obrigado pelas aulas de física, cálculo, mecânica. Obrigado pela companhia e pelos saudosos momentos de monitoria, os melhores do tempo de faculdade.

Ao professor Roberto Ávila Paldês pelos ensinamentos que foram importantes na minha vida acadêmica, pelas aulas cativantes e pela ajuda, nas etapas finais do curso e deste trabalho.

Aos professores Thiago Toribio, Luís Cláudio, Roberto Paldês, José Julimá, Maurício Lyra, Marco Antônio, Flávio Klein, Francisco Javier e Leonardo Pol, pelas aulas ministradas com carisma e entusiasmo e por me ensinarem como ser, um dia, um grande professor.

À professora Maria Marony pela paciência na orientação, pelo convívio e pelo apoio.

Na sala de aula, agradeço o bom convívio, os aprendizados, a amizade e o companheirismo da Thaiany Guilherme Cardoso e Claudiney Alves Moreira, com quem dividi todos os semestres da faculdade.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Neste trabalho, é mostrado que o ensino de Física no Brasil passa por dificuldades e que novos métodos de ensino, mais modernos e com uma maior participação do aluno são necessários. Desta forma, é proposta a criação de uma aplicação virtual onde alunos e professores possam interagir e realizar dois experimentos de Física I: Plano Horizontal e Plano Inclinado, ambos, com e sem atrito. Fazem parte das etapas de desenvolvimento desta aplicação a criação dos diagramas de caso de uso, atividades e classes, projeto de telas e a programação. A aplicação é feita em linguagem Java, gratuita, e pode ser executada localmente em qualquer computador pessoal. A aplicação apresenta gráficos, tabelas e animações, de forma similar ao que é feito no laboratório de física, com equipamentos de precisão. De acordo com testes e comparativos feitos, a aplicação apresenta resultados muito próximos aos encontrados no ambiente de laboratório, com erro médio menor que 1%. Dentre as vantagens deste trabalho em comparação ao que já é feito atualmente nos laboratórios de física estão a economia, a possibilidade de realizar o experimento num ambiente acadêmico ou doméstico e a maior interatividade do usuário com o experimento. Ao final, são sugeridas atividades e perguntas que os professores podem fazer aos alunos.

Palavras Chave: física, experimento, java, ensino, aprendizagem, laboratório, atrito.

ABSTRACT

In this work, it is shown that the physics education in Brazil is struggling and that new teaching methods, more modern and with a greater participation from the students are needed. So, it is proposed the creation of a virtual application where students and teachers may interact and perform two Physics I experiments: Horizontal Plane and Inclined Plane, both, with or without friction. It is part of the development stages of this application the creation of the use case, activity and class diagrams, screen design and programming. The application is made in free Java language and can be run locally in any personal computer. The application shows graphics, tables and animations, similarly to what is made in the Physics laboratory, with precision devices. According to tests and comparisons made, the application show results that are very close to those found in the laboratory environment, with average error bellow 1%. Among the advantages of this work compared to what is already done today in Physics laboratories are the economy, the possibility of performing the experiment in an academic or domestic environment and the greater interactivity with the experiment from user. In the end, it have been suggested questions that teachers can ask their students.

Keywords: physics, experiment, java, teaching, learning, laboratory, friction.

CAPÍTULO 1 - INTRODUÇÃO

Neste trabalho, é proposta uma aplicação de computador que auxilie no processo de ensino-aprendizagem de Física I. A aplicação simula dois experimentos laboratoriais de cinemática e mecânica – plano horizontal (com e sem atrito) e plano inclinado (com e sem atrito). A relevância deste trabalho está na alternativa que ele oferece às dificuldades em realizar este experimento com equipamentos adequados, além da possibilidade de alterar parâmetros que não são possíveis no ambiente de laboratório.

A aplicação é feita em linguagem de programação *Java*, que é gratuita, utilizando conhecimentos de física e matemática ensinados em sala de aula. Posteriormente, são feitas comparações entre os resultados obtidos pela aplicação e os resultados obtidos no laboratório com instrumentos reais. Na figura 1.1 é mostrada a tela principal da aplicação, com os ajustes que o usuário pode manipular.

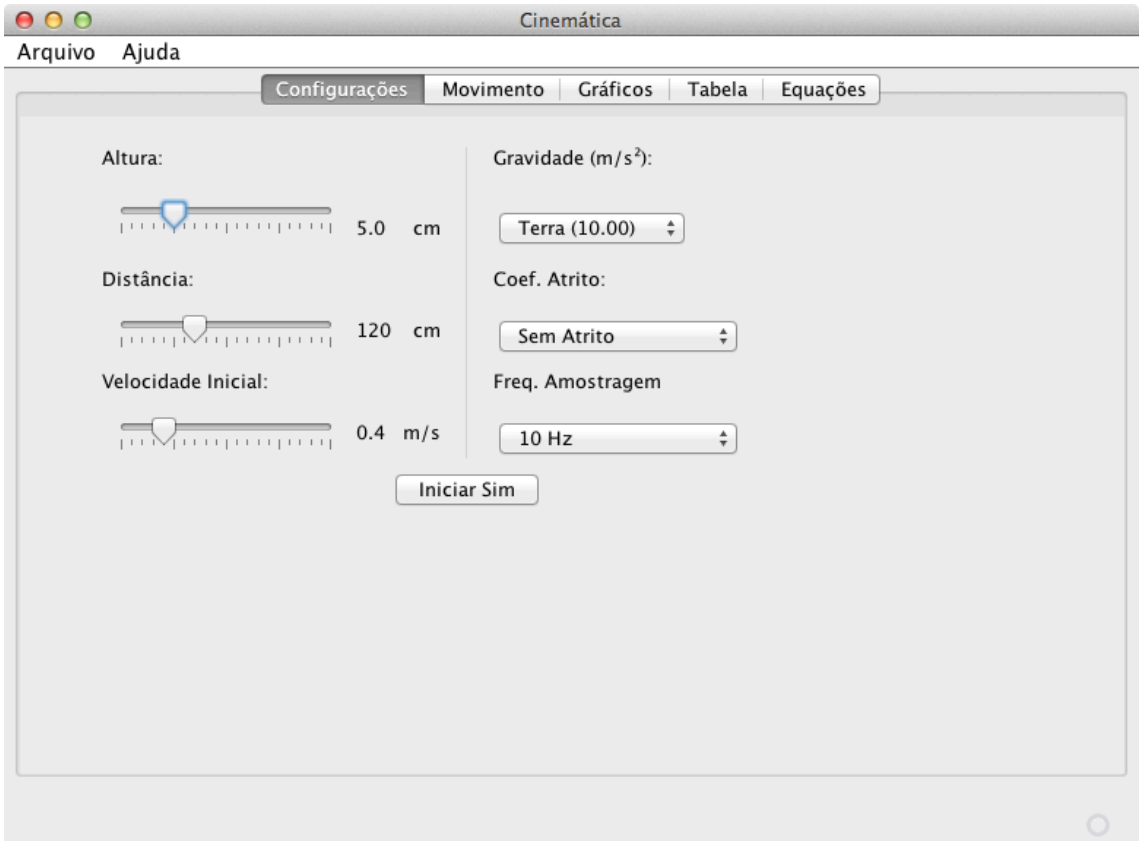

Figura 1.1 – Tela inicial da aplicação proposta neste trabalho
Fonte: Autor

1.1 Motivação

O ensino de Física no Brasil possui dificuldades que precisam ser encaradas e amparadas. Tanto em escolas de nível médio quanto no ensino superior, a disciplina apresenta alto índice de reprovação (BARROS, *et al.*, 2001). Pedagogos também indicam outro problema, voltado aos alunos do ensino superior, relacionado à dificuldade de ensinar adultos:

> Avançaremos mais se aprendermos a equilibrar o planejamento e criatividade, organização e adaptação a cada situação, a aceitar os imprevistos, a gerenciar o que podemos prever e a incorporar o novo, o inesperado. Planejamento aberto, que prevê, que está pronto para mudanças, para sugestões, adaptações. Criatividade, que envolve sinergia, pôr as diversas habilidades em comunhão, valorizar as contribuições de cada um, estimulando o clima de confiança, de apoio (MORAN, MASETTO e BEHRENS, 2007, p. 29).

A Revista Brasileira de Ensino de Física também discute com preocupação a questão da reprovação. Em artigo publicado na referida revista, Barros *et al* (2004) ressaltam que o ensino de Ciências e Matemática nas universidades brasileiras tem, em geral, um baixo rendimento que resulta em altos índices de reprovação, retenção e abandono. Uma das razões é o modelo passivo de aprendizado fomentado nos ambientes tradicionais de ensino em que alunos raramente interagem produtivamente e onde o estímulo é a nota e não o conhecimento. O autor destaca as estratégias utilizadas pelos alunos, no modelo passivo de aprendizagem: concentração em memorização, ao invés do entendimento, estudar nas vésperas de provas para obter notas, ao invés de conhecimentos, trabalhar sozinho, ao invés de articular ideias com seus colegas, solidificando-as (BARROS, *et al.*, 2004).

Nos cursos de Engenharia, adotar estratégias como as mencionadas anteriormente também é frequente. Machado (2009) expõe em artigo publicado para mestrado que na disciplina Física I dos cursos de Engenharia de Produção Mecânica e Engenharia de Produção, Controle e Automação, da Universidade Tecnológica Federal do Paraná, os alunos tiveram grande reprovação. Ele diz:

> Nesses cursos, ao final do primeiro ano de funcionamento, atingiu-se na disciplina de Física Geral I um alarmante índice de reprovação de aproximadamente 64%. Frente a esse problema, observou-se a necessidade de promover uma revisão na prática de ensino em Física junto às turmas de alunos recém-ingressos em Engenharia a fim de promover procedimentos de ensino na busca de despertar o aluno para uma nova realidade. Nessa nova realidade o aluno precisa conscientizar-se de que a qualidade da aprendizagem não deve ser

> refletida apenas em notas, mas, principalmente em significados. Se no ensino de Física a nível médio a aprendizagem priorizava a memorização de fórmulas e conceitos visando a aprovação em concurso vestibular, nos cursos de Engenharia essas fórmulas e conceitos só têm real significado ou importância se permitirem ao aluno o seu uso como base de conhecimento para a solução de problemas mais complexos (MACHADO, 2009, p. 3).

No artigo, Machado cita que as Diretrizes Curriculares Nacionais (DCNs) estabelecem que os cursos de Engenharias devam passar por mudanças, uma nova visão curricular, onde "[…]os professores de Engenharia precisam elaborar e aplicar atividades de ensino que passem a exigir do aluno uma participação mais ativa no processo de ensino e aprendizagem, realizando pesquisas, desenvolvendo projetos e realizando trabalhos acadêmicos" (MACHADO, 2009).

Em virtude do exposto, entende-se que uma nova abordagem, mais interativa e dinâmica, pode ser vantajosa para o ensino da disciplina de Física. Moran, Masetto e Behrens (2007) expõem princípios pedagógicos norteadores para uma melhor educação:

> Integrar tecnologias, metodologias, atividades. Integrar texto escrito, comunicação oral, escrita, hipertextual, multimídica. Aproximar as mídias, as atividades, possibilitando que transitem facilmente de um meio para o outro, de um formato para o outro. Experimentar as mesmas atividades em diversas mídias. Trazer o universo do audiovisual para dentro da escola.
>
> Variar a forma de dar aula, as técnicas usadas em sala de aula e fora dela, as atividades solicitadas, as dinâmicas propostas, o processo de avaliação. A previsibilidade do que o docente vai fazer pode tornar-se um obstáculo intransponível (MORAN, MASETTO e BEHRENS, 2007, p. 31).

No livro Educação Online, Silva (2006) corrobora com Moran, ao expor fundamentos da andragogia e da educação moderna. Ele diz:

> Em vez da transmissão unidirecional de informação, valoriza-se cada vez mais a interação e a troca de informação entre professor e aluno. No lugar da reprodução passiva de informações já existentes, deseja-se cada vez mais o estímulo à criatividade dos estudantes (SILVA, 2006, p. 27).

Tais mudanças educacionais não são fáceis de serem implementadas. O ensino de qualidade envolve uma organização inovadora, aberta e dinâmica, um projeto pedagógico flexível e infraestrutura adequada (MORAN, MASETTO e BEHRENS, 2007).

Especificamente acerca de um laboratório de física, os custos são elevados.

Computadores (e seus respectivos sistemas operacionais e aplicativos devidamente registrados e licenciados) e sensores de precisão são caros. Há a necessidade de se baratear custos em todos os aspectos (FERNANDES, 2007).

Pesquisas para a melhoria do ensino de Física tem sido feitas. Uma abordagem recente, que demonstrou ter despertado o interesse dos alunos, consiste na utilização de computadores como ferramentas para a resolução de problemas. Paralelamente à aprendizagem, o aluno aprofunda ou adquire conhecimentos de informática. A utilização do computador traz benefício no custo, especialmente quando comparado ao custo de um laboratório de física. Outro benefício a ser destacado é a mobilidade: Alunos que possuem computador em suas casas, podem refazer exercícios. Alunos que não possuem, fazem uso dos computadores do centro de estudos, tudo isso aliado à segurança, uma vez que não precisam manusear instrumentos perigosos (MORAES e RIBEIRO-TEIXEIRA, 2006).

O desenvolvimento de jogos também é objeto de estudos. A tentativa aqui é encontrar um equilíbrio entre os saberes e os fazeres, o aprender e o ensinar, de modo que o aprendizado seja evolutivo e considerável (PERRY, TIMM, *et al.*, 2007).

Um estudo recente aponta, qualitativa e quantitativamente, que trabalhos em grupo, onde os membros interagem uns com os outros, assumindo papéis diferentes (líder, anotador e cético), ajudam a diminuir o índice de reprovação, conforme é mostrado na Tabela 1. No trabalho, foram comparados os métodos de ensino tradicionais e o Engajamento Interativo (EI), proposto no artigo. O Engajamento Interativo consiste de apresentar o mesmo conteúdo sob diferentes óticas, ambientes e instrumentos de ensino (BARROS, REMOLD, *et al.*, 2004).

Tabela 1 – Índices de aprovação, para alunos frequentes em 2001/2 e 2001/1 em turmas de Física I utilizando métodos de ensino tradicionais e Engajamento Interativo (EI)

Ano/Semestre	Método	Aprovados
2001/2	Tradicional	36%
2001/2	EI	45%
2001/1	Tradicional	28%
2001/1	EI	59%

Fonte: BARROS, *et al*, 2004.

1.2 Objetivos

Diante do exposto, neste trabalho, é proposta a criação de uma aplicação computacional gratuita, que permita a interação aluno-aluno, aluno-professor e aluno-computador. A aplicação simula dois importantes experimentos de Física I, a saber, plano horizontal (com e sem atrito) e plano inclinado (com e sem atrito).

1.2.1 Objetivo Geral

Atuar como instrumento de ensino auxiliar, tanto para professores quanto para alunos, permitindo que o usuário faça as mesmas alterações de parâmetros que ele faria no ambiente do laboratório, além de outras, como mudança da aceleração local da gravidade e a mudança do coeficiente de atrito.

Em complemento, são apresentadas sugestões de roteiros e perguntas a serem discutidas pelos alunos, similarmente ao proposto por Barros, *et al.*, em artigo publicado na Revista Brasileira de Ensino de Física, visando melhoria no aprendizado efetivo (BARROS, MELO, *et al.*, 2001).

Com a criação deste trabalho, espera-se permitir ao aluno e ao professor interagir mais com estes experimentos de física e, possivelmente, levá-los a lugares antes não imaginados, como escolas e faculdades públicas que não possuem condições de montar um laboratório de física.

1.2.2 Objetivos Específicos

Fornecer ao usuário da aplicação proposta neste trabalho, a possibilidade de manipular os seguintes parâmetros do experimento: altura (para inclinar o percurso), distância do percurso, velocidade inicial do corpo, aceleração local da gravidade, coeficiente de atrito e frequência de amostragem. Variando a frequência de amostragem, é possível obter um gráfico com a precisão adequada àquela situação, ou seja, com uma grande densidade de pontos (maior precisão) ou pequena densidade de pontos (menor precisão).

Fornecer ao usuário da aplicação proposta neste trabalho, a possibilidade de visualizar os seguintes dados: prévia do movimento do corpo, gráficos (posição versus tempo, velocidade versus tempo e aceleração versus tempo), tabela (tempo e posição) e as equações do experimento.

1.3 Metodologia

Para o desenvolvimento deste projeto, foi realizada pesquisa bibliográfica para definir qual tipo de linguagem de computação seria a mais eficaz para a implementação do modelo pensado. Foram também realizados testes, comparações e simulações, com o propósito de assegurar que a aplicação desenvolvida está de acordo com os experimentos realizados em laboratório. Para isso, tabelas e estatísticas foram analisadas e algumas variáveis de erro da aplicação foram ajustadas.

Os seguintes componentes *Java* e computacionais são utilizados:

- Ambiente de desenvolvimento *NetBeans*;
- Bibliotecas *Java* padrão;
- Biblioteca *swing*, para elementos da interface do usuário;
- Biblioteca *awt*, para desenhos e gráficos;
- Objetos de tabela *JTable* e variáveis em ponto flutuante com o dobro de precisão (*double*);
- Ferramentas de desenho vetorial (*Adobe Illustrator* e *Adobe Photoshop*) e
- *Microsoft Excel* para comparativos entre valores reais e valores da aplicação.

Para demonstração prática da aplicação, são apresentados os recursos necessários para a realização dos experimentos em questão num laboratório e os resultados são comparados com aqueles obtidos na aplicação. A execução da aplicação não requer nenhum tipo de computador especial, sendo necessário apenas a máquina virtual do *Java* e executar a aplicação.

Para reproduzir este trabalho, deve-se primeiramente fazer um projeto da aplicação com artefatos da engenharia de programas, estabelecendo quem são os usuários e quais funcionalidades eles terão acesso (diagrama de caso de uso), quais são as atividades da aplicação (diagrama de atividades) e quais as ferramentas que suportam as atividades

(diagrama de classes).

Em seguida, deve ser feito um projeto de telas e seus elementos devem ser dispostos. Nesta etapa, as variáveis e constantes da aplicação devem ser conhecidas e seus valores escritos.

É então introduzida a programação da modelagem física e matemática. Esta modelagem deve ser comparada com o ambiente de laboratório, a fim de verificar possíveis erros de interpretação ou programação.

Por último, os parâmetros devem ser passados para as funções que geram os gráficos, animações e as tabelas desejadas.

1.4 Estrutura da Monografia

Este trabalho está dividido em cinco capítulos, iniciando por este, Introdução, onde é discutida a motivação para a realização do projeto, descrição dos objetivos e metodologia a ser utilizada. Este capítulo ainda traz esta seção, que descreve toda a estrutura da monografia.

O segundo capítulo, Referencial Teórico, aborda os principais conceitos de física e cálculo numérico envolvidos neste trabalho.

O terceiro capítulo, Modelo Proposto, traz informações e trechos do código da aplicação. Constam ainda neste capítulo os artefatos da engenharia de programas, todas as telas da aplicação e um descritivo de todas as suas funcionalidades.

O quarto capítulo, Testes e Resultados, mostra comparativos entre o modelo proposto e o laboratório real, bem como os resultados obtidos ao longo da implementação deste projeto.

O quinto capítulo, Conclusão, traz as considerações finais sobre o trabalho e sugestões para trabalhos futuros.

Além dos cinco capítulos, este trabalho também apresenta, em sua parte final, as referências bibliográficas, apêndices e anexos.

CAPÍTULO 2 - REFERENCIAL TEÓRICO

O trabalho proposto nesta monografia permite que o leitor manipule todos os dados trabalhados num ambiente de laboratório, podendo, ainda, manipular parâmetros que não são usualmente intercambiáveis, como o coeficiente de atrito e a aceleração local da gravidade.

Determinar valores fixos para o coeficiente de atrito e para a aceleração local da gravidade não foi uma tarefa trivial. Na aplicação, o simulador utiliza valores com base num estudo estatístico destas duas variáveis, conforme é apresentado a seguir.

2.1 Forças de Atrito

As forças de atrito observadas no cotidiano possuem origem eletromagnéticas. Estas forças são completamente compreendidas no que diz respeito às suas leis básicas, entretanto, fenômenos que envolvem muitas partículas apresentam dificuldades matemáticas que não foram ainda superadas. Assim, para o estudo do atrito, foram propostos modelos e valores experimentais de referência (CHAVES, 2001).

As forças de atrito, no estudo da física mecânica, são usualmente divididas em força de atrito estático e força de atrito cinético. A força de atrito estático é a força que envolve duas superfícies paradas com tendência de movimento relativo antes da iminência de movimento(ou seja, quando elas estão próximas a começar a deslizar). A força de atrito cinético é a força entre duas superfícies em movimento, ou seja, quando a força de atrito estático entre as duas superfícies é ultrapassado (KELLER, GETTYS e SKOVE, 1997).

O módulo da força de atrito estático máximo é o produto do coeficiente de atrito estático pelo módulo da força normal do objeto, conforme mostrado na equação 2.1,

$$f_e = \mu_e N, \tag{2.1}$$

onde f_e é o módulo da força de atrito estático máximo, μ_e é o coeficiente de atrito estático da superfície em questão e N é o módulo da força normal ao objeto.

O módulo da força de atrito cinético é o produto do coeficiente de atrito cinético pelo

módulo da força normal do objeto, conforme mostrado na equação 2.2,

$$f_c = \mu_c N, \tag{2.2}$$

onde f_c é o módulo da força de atrito cinético e μ_c é o coeficiente de atrito cinético da superfície em questão.

Outrossim, existe o atrito de rolamento, aquele que as estradas exercem sobre as rodas, opondo-se ao movimento dos carros. Para manter o carro em velocidade constante, é necessário exercer uma força sobre a roda igual em magnitude e oposta em sentido à força de atrito de rolamento exercida sobre a roda pela estrada. Embora esta força seja considerada insignificante para a resolução de exercícios (TIPLER e MOSCA, 2009), devido a sua pequena ordem de grandeza (valores típicos para o coeficiente de atrito de rolamento variam entre 0,01 a 0,02 para pneus de borracha e 0,001 a 0,002 para rodas de aço sobre trilhos de aço), esta força não foi considerada no simulador proposto neste trabalho. Ainda, Keller, Gettys e Skove mencionam que a forma de estudar e compreender o coeficiente de atrito de rolamento é "praticamente a mesma" da forma de estudar o coeficiente de atrito cinético (KELLER, GETTYS e SKOVE, 1997).

Experimentos mostram que as forças de atrito estático e cinético dependem da natureza e da condição das duas superfícies em questão. Ambas as forças também são praticamente independentes da área de contato entre os corpos. O atrito cinético é aproximadamente independente da velocidade, quando consideradas velocidades relativas baixas entre as superfícies (KELLER, GETTYS e SKOVE, 1997).

Em vista disso, determinar os valores de coeficientes de atritos a serem utilizados na aplicação foi de primordial importância. Diversos autores apresentam tabelas de coeficientes de atrito estático e dinâmico, sendo que estes não possuem um valor padronizado. O atrito é um conceito estatístico, representado pela soma de um número de interações entre as moléculas dos dois corpos em questão, sendo impossível analisar tais interações individualmente (ALONSO e FINN, 2001).

Mediante o exposto, os coeficientes de atrito cinético utilizados nesta aplicação foram determinados como uma média dos valores das tabelas encontradas na literatura, conforme apresentado na Tabela 2 (FREEDMAN e YOUNG, 2003) (SERWAY e JEWETT, 2004) (TIPLER e MOSCA, 2009).

Tabela 2 – Valores tabelados de coeficientes de atrito cinético, de acordo com três autores, e suas médias simples

Materiais envolvidos	(SERWAY e JEWETT, 2004)	(TIPLER e MOSCA, 2009)	(FREEDMAN e YOUNG, 2003)	Média
Aço sobre aço	0,57	0,6	0,57	0,58
Alumínio sobre aço	0,47	–	0,47	0,47
Latão sobre aço	–	0,4	0,44	0,42
Vidro sobre vidro	0,4	0,4	0,4	0,4

Fontes: (SERWAY e JEWETT, 2004), (TIPLER e MOSCA, 2009) e (FREEDMAN e YOUNG, 2003).

2.2 Força Gravitacional

O segundo parâmetro da tomada de dados manipulável a ser tratado neste trabalho é a aceleração local da gravidade. Num ambiente de laboratório de escolas de ensino médio ou mesmo laboratórios de universidades, alterar o valor da gravidade local é inviável. Para a aplicação proposta neste trabalho, é possível que o usuário altere o valor da gravidade. Os valores carregados no simulador foram estabelecidos de acordo com a lei de Newton da gravitação universal, conforme mostrado na equação 2.3 (CHAVES, 2001), (KELLER, GETTYS e SKOVE, 1997) e (TIPLER e MOSCA, 2009),

$$g = \frac{Gm_c}{R_c^2}, \tag{2.3}$$

onde, g é o módulo da aceleração da gravidade num determinado ponto de observação, G é a constante gravitacional, m_c é a massa do corpo em questão e R_c é a distância do centro do corpo até um ponto de observação.

As bibliografias acima mencionadas divergem a partir do terceiro algarismo significativo do valor da constante gravitacional. Dessa forma, foi feita uma média aritmética entre os valores encontrados. Chaves apresenta o valor $6{,}67259\times10^{-11}\,N.m^2/Kg^2$, Keller, Gettys e Skove apresentam o valor $6{,}670\times10^{-11}\,N.m^2/Kg^2$ e, por último, Tipler e Mosca

apresentam o valor $6{,}6742\times10^{-11}\ N.m^2/Kg^2$.

Vale ressaltar que, devido à natureza extremamente fraca da força gravitacional, ainda hoje, duzentos anos após a primeira medição da constante de gravitação universal, uma das primeiras constantes físicas medidas, seu valor é ainda classificado como de baixa precisão (TIPLER e MOSCA, 2009).

A partir da equação 2.3 e tendo acesso a tabelas de raios e massas de outros planetas, foi possível estimar a ordem de grandeza de outras forças gravitacionais. Tais valores são apresentados na aplicação proposta neste trabalho.

2.3 Experimento 1 – Plano Horizontal

O primeiro experimento que a aplicação apresentada neste trabalho simula é o chamado "Plano Horizontal". Este experimento consiste em deslocar um corpo por um trilho horizontal, obtendo suas posições em função do tempo. É possível também obter seus valores de velocidade instantânea em função do tempo e ainda a sua aceleração em função do tempo.

Para a realização desta prática experimental com objetos reais, o laboratório de física do Centro Universitário de Brasília está equipado com: um sensor de movimento, um programa para coletar os dados do sensor em função do tempo e um local apropriado para o corpo deslizar, preferencialmente com o atrito conhecido.

Na figura 2.1 é mostrado um exemplo de montagem deste experimento. Um modelo em escala de um carro é posicionado num trilho horizontal, permitindo que o mesmo se movimente em apenas uma direção. O carro é impulsionado e o sensor de movimento (canto esquerdo do trilho) começa a captar seus movimentos, registrando no computador a posição do carro em função do tempo. O sensor funciona a uma taxa de amostragem fixa. Ao atingir o aparador (canto direito do trilho), o sensor de movimento é desligado, consequentemente, o programa de computador pára os registros. Ainda na figura 2.1, é possível ver um diagrama de forças atuantes no carro. O princípio fundamental da dinâmica é mostrado no conjunto de equações 2.4, e a consequente aceleração do móvel é dada pela equação 2.5. Considera-se, neste caso, o sentido de deslocamento do móvel da esquerda para a direita (TIPLER e MOSCA, 2009).

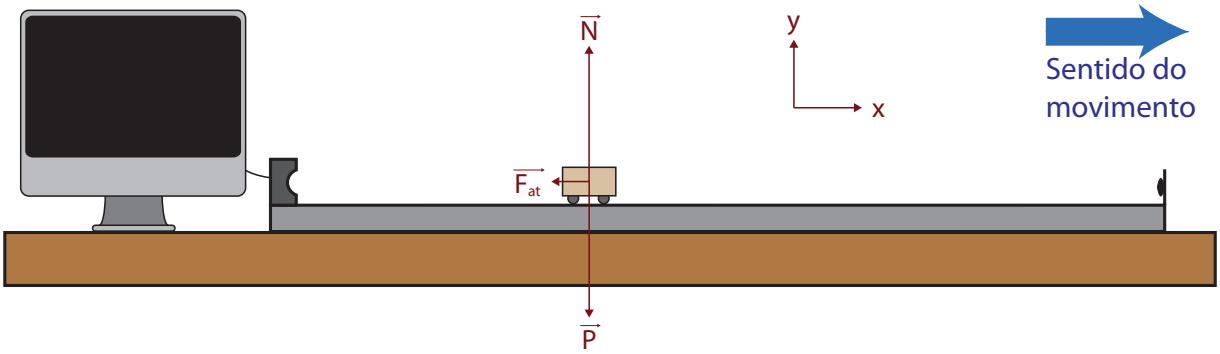

Figura 2.1 – Representação do experimento “Plano Horizontal”
Fonte: Autor

$$\sum F_x = ma; \tag{2.4}$$

$$-F_{at} = ma; \tag{2.4a}$$

$$-\mu_c N = ma; \tag{2.4b}$$

$$-\mu_c mg = ma; \tag{2.4c}$$

$$a = -\mu_c g. \tag{2.5}$$

Onde F_x são as forças do eixo cartesiano x, a é o módulo da aceleração do corpo em questão e F_{at} é a força de atrito.

Tendo obtido a aceleração do corpo em questão, foram obtidas as equações 2.6 e 2.6a, que descrevem a posição do corpo em questão, em função do tempo.

$$S(t) = S_0 + v_0 t + \frac{at^2}{2}; \tag{2.6}$$

$$S(t) = S_0 + v_0 t + \frac{(-\mu_c g)t^2}{2}. \tag{2.6a}$$

Onde $S(t)$ é a posição do corpo em questão, t é o tempo, S_0 é a posição inicial do corpo e v_0 é a velocidade inicial do corpo.

2.4 Experimento 2 – Plano Inclinado

O segundo experimento que a aplicação apresentada neste trabalho simula é o chamado “Plano Inclinado”. Este experimento consiste em deslocar um corpo por um trilho inclinado em um ângulo conhecido, obtendo sua posição em função do tempo. É possível

também obter seus valores de velocidade instantânea em função do tempo e ainda a sua aceleração em função do tempo.

Para a realização desta prática experimental com objetos reais, da forma como é feito no laboratório de física do Centro Universitário de Brasília, o laboratório precisa estar equipado com: um sensor de movimento, um programa para coletar os dados do sensor em função do tempo, um local apropriado para o corpo deslizar, preferencialmente com o atrito sendo conhecido e apoios para inclinar o plano de deslizamento do corpo.

Na figura 2.2 é mostrado um exemplo de montagem deste experimento. Um modelo em escala de um carro é posicionado num trilho inclinado, que permite que o mesmo se movimente em apenas uma direção. O carro é abandonado e o sensor de movimento (canto esquerdo do trilho) começa a captar seus movimentos e registrar no computador a posição do carro em função do tempo. O sensor funciona a uma taxa de amostragem fixa. Ao atingir o aparador (canto direito do trilho), o sensor de movimento é desligado, consequentemente, o programa de computador para os registros. Ainda na figura 2.2, é possível ver um diagrama de forças atuantes no carro. O princípio fundamental da dinâmica é mostrado no conjunto de equações 2.7 e a consequente aceleração do móvel é dada pela equação 2.8.

A inclinação do trilho desta figura foi exagerada propositalmente, para melhor visualização das forças atuantes no sistema (TIPLER e MOSCA, 2009).

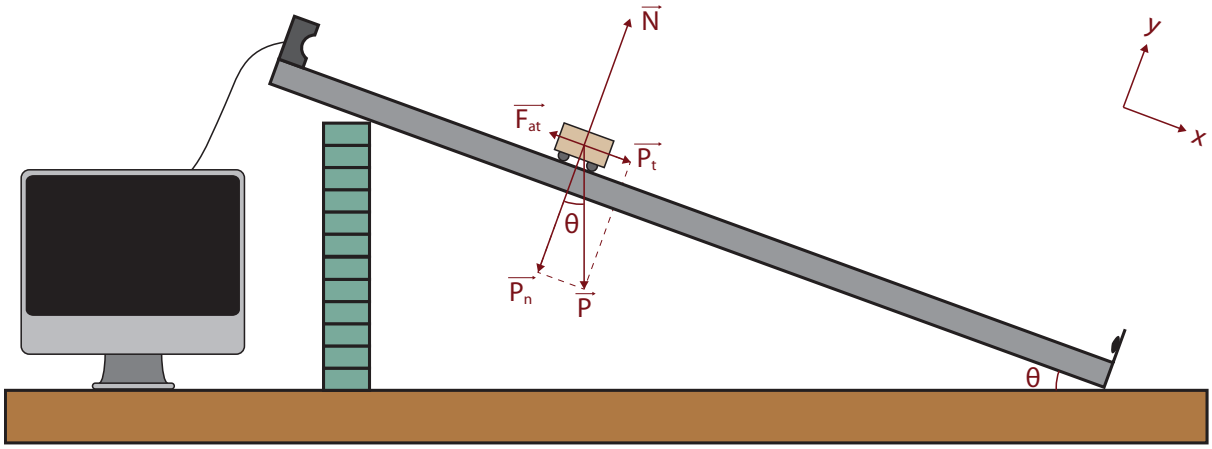

Figura 2.2 – Representação do experimento “Plano Inclinado”
Fonte: Autor

$$\sum F_x = P_t - F_{at} = ma; \tag{2.7}$$

$$P \operatorname{sen}\theta - \mu_c N = ma; \tag{2.7a}$$

$$P \operatorname{sen}\theta - \mu_c P \cos\theta = ma; \tag{2.7b}$$

$$mg(\text{sen}\,\theta - \mu_c \cos\theta) = ma; \tag{2.7c}$$

$$a = g(\text{sen}\,\theta - \mu_c \cos\theta). \tag{2.8}$$

Onde P é a força peso do corpo em questão, P_t é a componente tangencial da força peso do corpo em questão e P_n é a componente normal da força peso do corpo em questão.

Para provar que $P_t = P\,\text{sen}\,\theta$ e que $P_n = P\cos\theta$, basta analisar a trigonometria da figura 2.3, que expande as forças mencionadas, por questões didáticas.

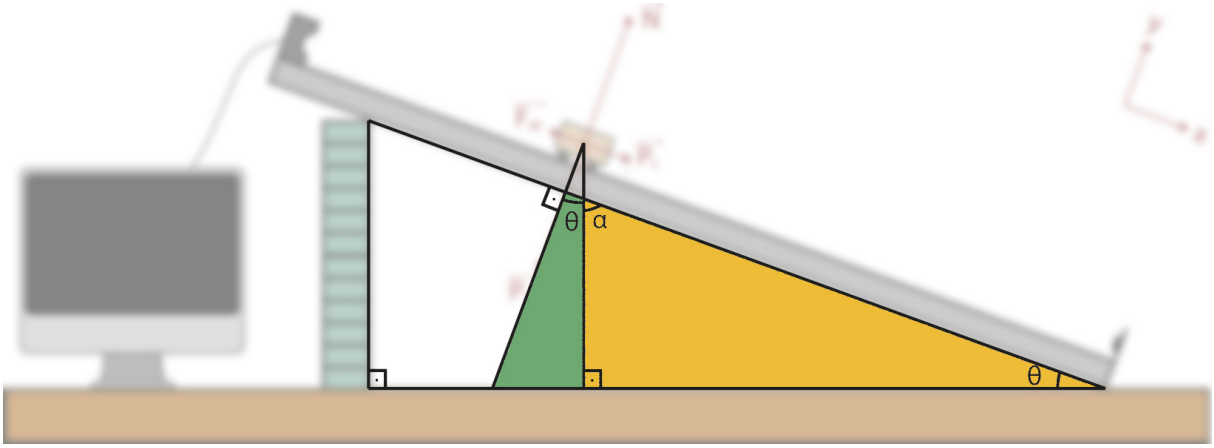

Figura 2.3 – Demonstração dos ângulos no plano inclinado
Fonte: Autor

O ângulo θ do plano inclinado (triângulo amarelo), somado ao ângulo reto que a força peso faz com a mesa e somado ainda com o ângulo α que a força peso faz com o percurso, totalizam 180 (cento e oitenta) graus. Ao analisar então o percurso, nota-se que que o mesmo ângulo α é somado a um ângulo reto que a componente P_n faz com o percurso. Para completar os mesmos 180 (cento e oitenta) graus da meia circunferência, resta o ângulo θ. Consequentemente, ao decompor o vetor P em suas componentes P_n e P_t, tem-se um triângulo retângulo, com hipotenusa P e catetos P_n e P_t. Uma vez que P_n é o cateto oposto ao ângulo θ, tem-se, finalmente, $\text{sen}\,\theta = P_n/P$.

Tendo obtido a aceleração do corpo em questão, esta aceleração foi substituída na equação 2.6 e então foi obtida a equação 2.9, que descreve a posição do corpo em questão, em função do tempo.

$$S(t) = S_0 + v_0 t + \frac{g(\text{sen}\,\theta - \mu_c \cos\theta)t^2}{2}. \tag{2.9}$$

2.5 Ajustes

No ambiente de laboratório, uma vez obtidas as tabelas de espaço (posição) em função do tempo, num experimento real, torna-se necessário fazer um ajuste dos pontos obtidos. Essa necessidade tem origem na própria obtenção dos dados experimentais, que possuem erros inerentes ao processo. São exemplos de fontes de erros experimentais a calibração de sensores, a precisão de sensores, a precisão no posicionamento de objetos, o tempo de execução da coleta de dados (SPERANDIO, MENDES e SILVA, 2003).

O tipo de ajuste necessário à primeira tabela (Espaço x Tempo) do primeiro experimento tratado na aplicação proposta neste trabalho ("Plano Horizontal") é o ajuste linear, já que o diagrama de dispersão proveniente deste experimento possui a forma geral de uma reta. Este ajuste também pode ser utilizado na segunda tabela (Velocidade x Tempo) do segundo experimento ("Plano Inclinado") tratado no programa. Já para a tabela de posições do segundo tipo de experimento tratado na aplicação proposta neste trabalho ("Plano Inclinado"), o tipo de ajuste necessário é o ajuste polinomial (PEDROSA, 2005).

2.5.1 Ajuste Linear

O ajuste linear é obtido através de uma equação de reta, de modo que o problema em questão está em ajustar os parâmetros β_0 e β_1 da reta, que melhor se ajustam ao diagrama de dispersão do experimento, conforme mostrado na equação 2.10 (SPERANDIO, MENDES e SILVA, 2003)

$$f(x) = \beta_0 + \beta_1 x, \tag{2.10}$$

onde $f(x)$ é a função em si, β_0 é o coeficiente linear da função, β_1 é o coeficiente angular da função e x é a variável independente.

Para ajustar os parâmetros β_0 e β_1, é utilizado o Método dos Mínimos Quadrados, cuja solução é descrita nas equações 2.11 e 2.12

$$\beta_1 = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{n\sum x_i^2 - (\sum x_i)^2}; \tag{2.11}$$

$$\beta_0 = \frac{\sum y_i - (\sum x_i)\beta_1}{n}, \tag{2.12}$$

onde n é a quantidade de medidas, x_i são valores da variável *x* e y_i representam os valores da variável dependente *y* (SPERANDIO, MENDES e SILVA, 2003).

Desta forma, o ajuste linear é feito no experimento do Plano Horizontal já que o tipo de movimento ali tratado é Movimento Uniforme, cuja equação é do primeiro grau, conforme mostrado na equação 2.13 (TIPLER e MOSCA, 2009)

$$S(t) = S_0 + vt. \tag{2.13}$$

2.5.2 Ajuste Polinomial

O ajuste polinomial é obtido através de uma curva, de modo que o problema está em ajustar os parâmetros β_0, β_1 e β_2 da curva, que melhor se ajustam ao diagrama de dispersão do experimento, conforme mostrado na equação 2.14 (SPERANDIO, MENDES e SILVA, 2003).

$$f(x) = \beta_0 + \beta_1 x + \beta_2 x^2 \tag{2.14}$$

Para determinar os parâmetros β_0, β_1 e β_2 da curva, foi montado o sistema descrito na equação 2.15, cuja solução é feita pelo Método de Gauss, que implica em transformar este sistema original em um sistema triangular, através de operações elementares (SPERANDIO, MENDES e SILVA, 2003).

$$\begin{bmatrix} n & \sum x_i & \sum x_i^2 \\ \sum x_i & \sum x_i^2 & \sum x_i^3 \\ \sum x_i^2 & \sum x_i^3 & \sum x_i^4 \end{bmatrix} . \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{bmatrix} = \begin{bmatrix} \sum y_i \\ \sum y_i x_i \\ \sum y_i x_i^2 \end{bmatrix}. \tag{2.15}$$

Assim, visto que o experimento de Plano Inclinado é um experimento de Movimento Uniformemente Variado, cuja equação é um polinômio de grau 2 (dois), conforme mostrado na equação 2.16, é feito, em seu conjunto de posições, o ajuste polinomial, a fim de se obter a equação do movimento (TIPLER e MOSCA, 2009).

$$S(t) = S_0 + v_0 t + \frac{at^2}{2}. \tag{2.16}$$

2.6 Programação

Além dos aspectos físicos e matemáticos, convém tratar da parte de programação utilizada no desenvolvimento do trabalho proposto. A linguagem de programação utilizada é o *Java*, em sua versão para desenvolvedores *Java 2 Platform, Standard Edition* (J2SE), versão esta para uso em computadores pessoais. O J2SE inclui a máquina virtual *Java*, o compilador *Java*, as APIs do *Java* (conjunto de padrões e rotinas pré-estabelecidas para utilização das funcionalidades do *Java* pelos aplicativos) e algumas ferramentas utilitárias (ORACLE TECHNOLOGY NETWORK, 2011).

A escolha da linguagem de programação *Java* é baseada em três pontos: em suas ferramentas básicas, suas bibliotecas prontas para uso e gratuidade. O *Java* é uma linguagem de programação poderosa e rápida o suficiente para as necessidades da aplicação proposta. É capaz de trabalhar com números em ponto flutuante com boa aproximação e sua programação é independente de plataforma. Isso quer dizer que o mesmo código *Java* pode gerar um programa executável para sistemas operacionais diferentes, como *Windows*, *Mac* e *Linux*.

CAPÍTULO 3 - MODELO PROPOSTO

Neste capítulo, são apresentados os projetos de telas, os diagramas da UML, trechos do código e das classes da aplicação proposta neste trabalho, seguidos de um breve comentário e explicações.

3.1 Projeto de Telas

Para a aplicação proposta neste trabalho, são projetadas cinco telas, posteriormente agrupadas em abas. Estas cinco telas consistem de: Configurações (mostradas na figura 3.1), Movimento (mostrado na figura 3.2), Gráficos (mostrados nas figuras 3.3a, 3.3b e 3.3c), Tabela (mostrada na figura 3.4) e Equações (mostradas na figura 3.5).

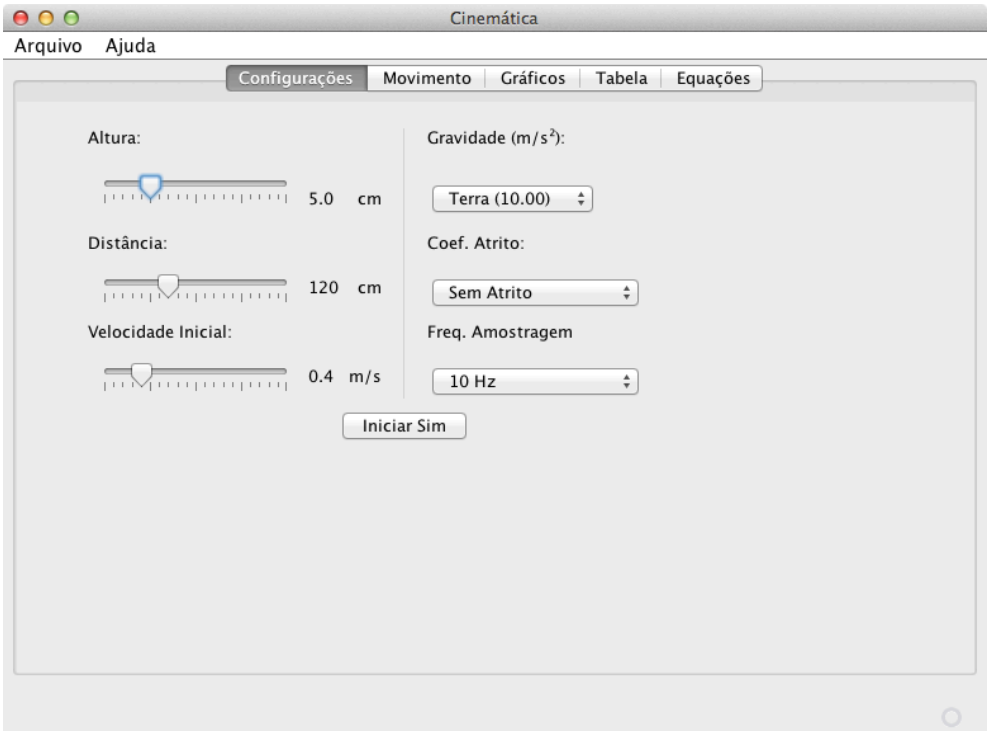

Figura 3.1 – Tela de configurações do experimento
Fonte: Autor

Dessa maneira, na figura 3.1 podem ser observadas as possibilidades de configurações e ajustes do experimento. É mostrada, na figura 3.2, uma prévia do movimento feito pelo carro em escala e do percurso que este carro fará.

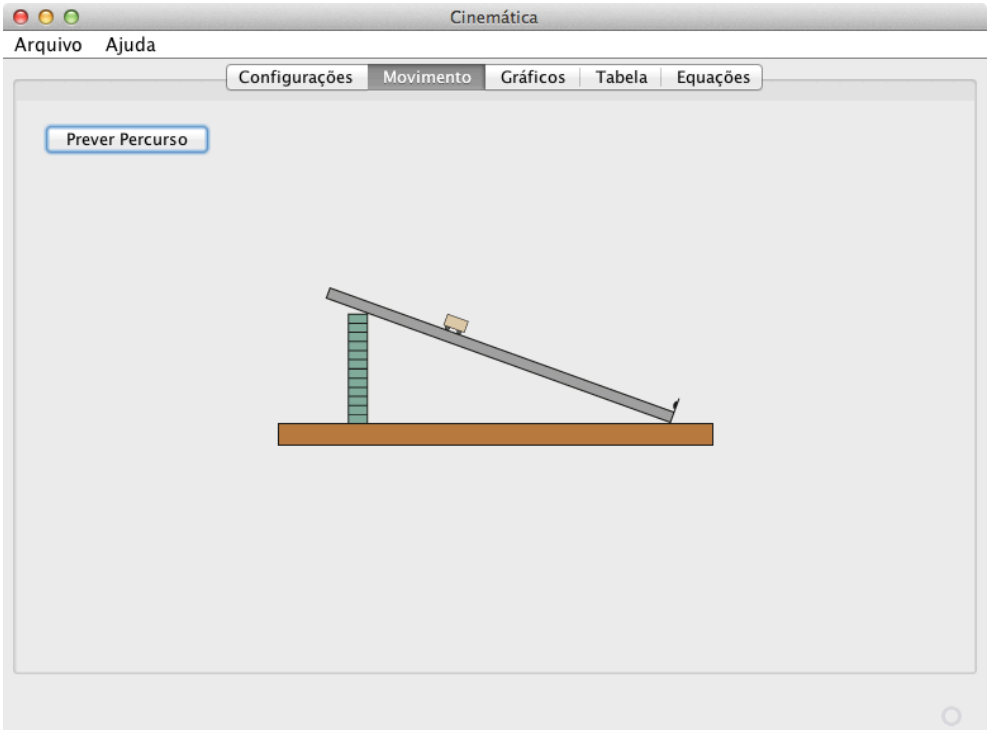

Figura 3.2 – Tela de prévia do movimento executado pelo carro em escala
Fonte: Autor

Nas figuras 3.3a, 3.3b e 3.3c, são mostrados os gráficos que a aplicação produz. Para melhor organização, todos os gráficos foram dispostos na mesma aba – Gráficos.

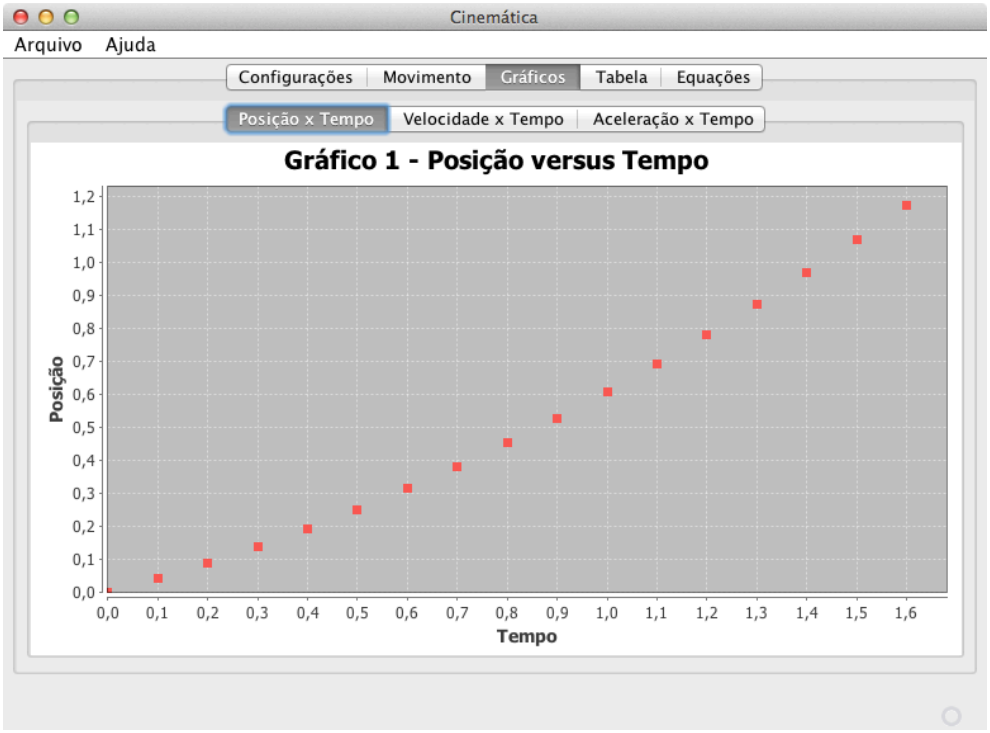

Figura 3.3a – Tela do gráfico da posição do carro em escala em função do tempo
Fonte: Autor

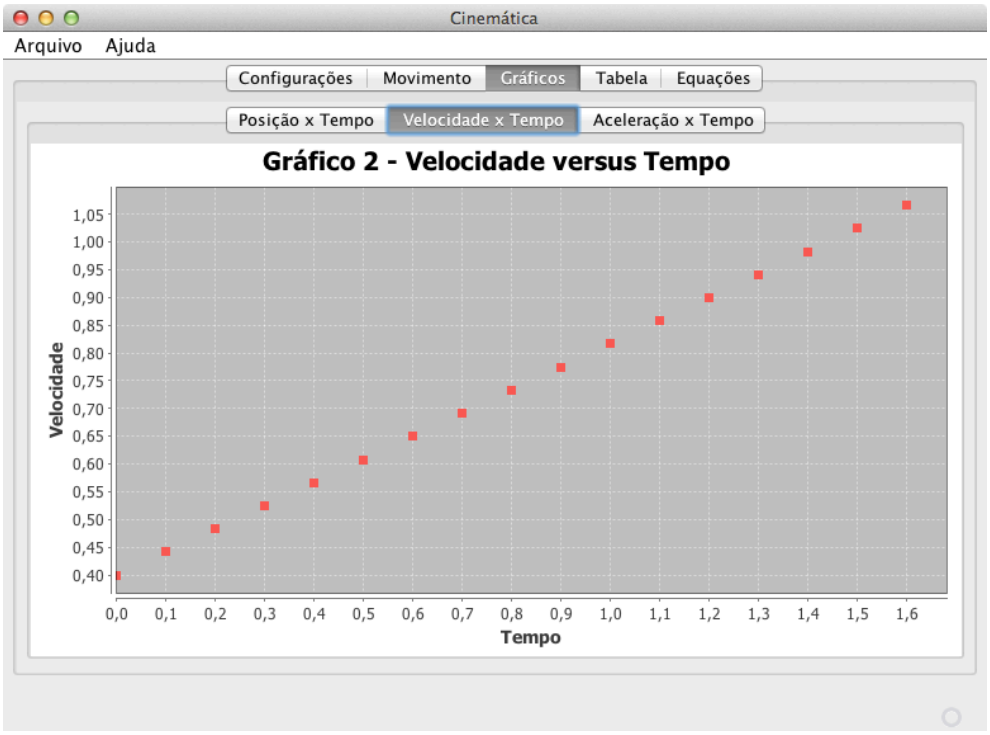

Figura 3.3b – Tela do gráfico da velocidade do carro em escala em função do tempo
Fonte: Autor

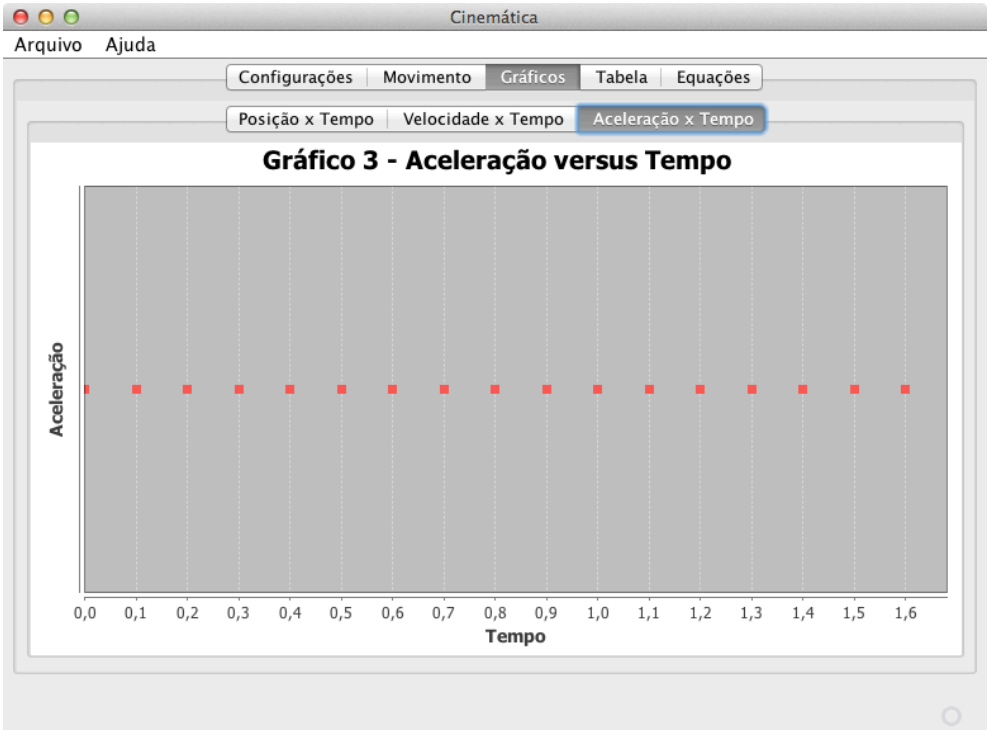

Figura 3.3c – Tela do gráfico da aceleração do carro em escala em função do tempo
Fonte: Autor

A quarta e quinta telas, denominadas Tabela e Equações, respectivamente, são mostradas nas figuras 3.4 e 3.5, respectivamente.

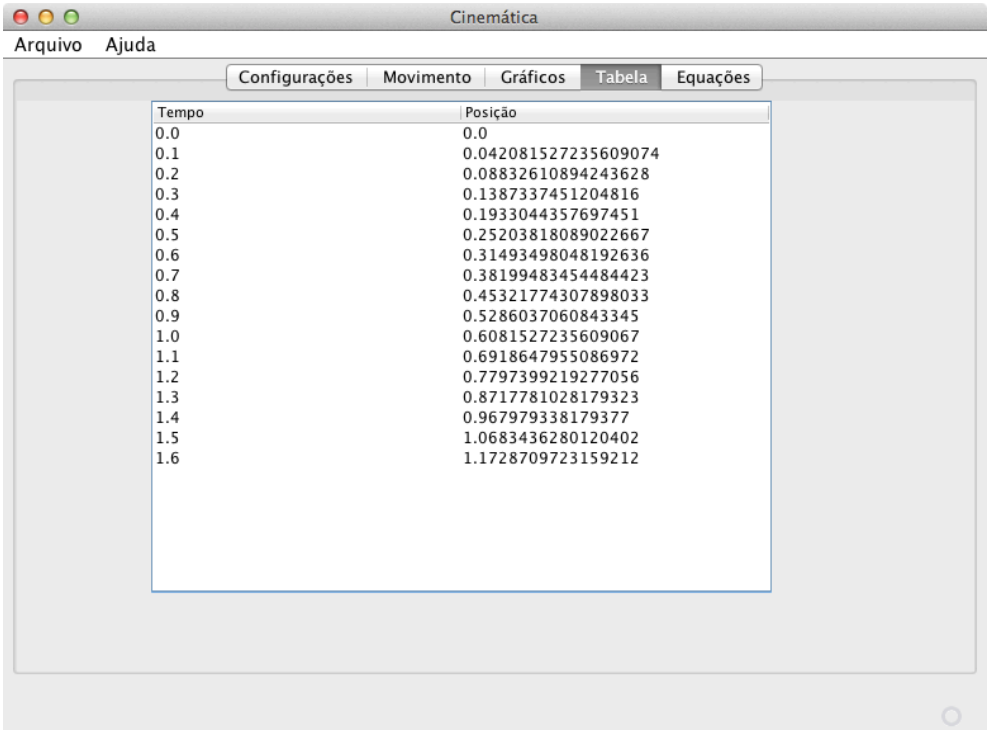

Figura 3.4 – Tela da tabela de tempo e posição
Fonte: Autor

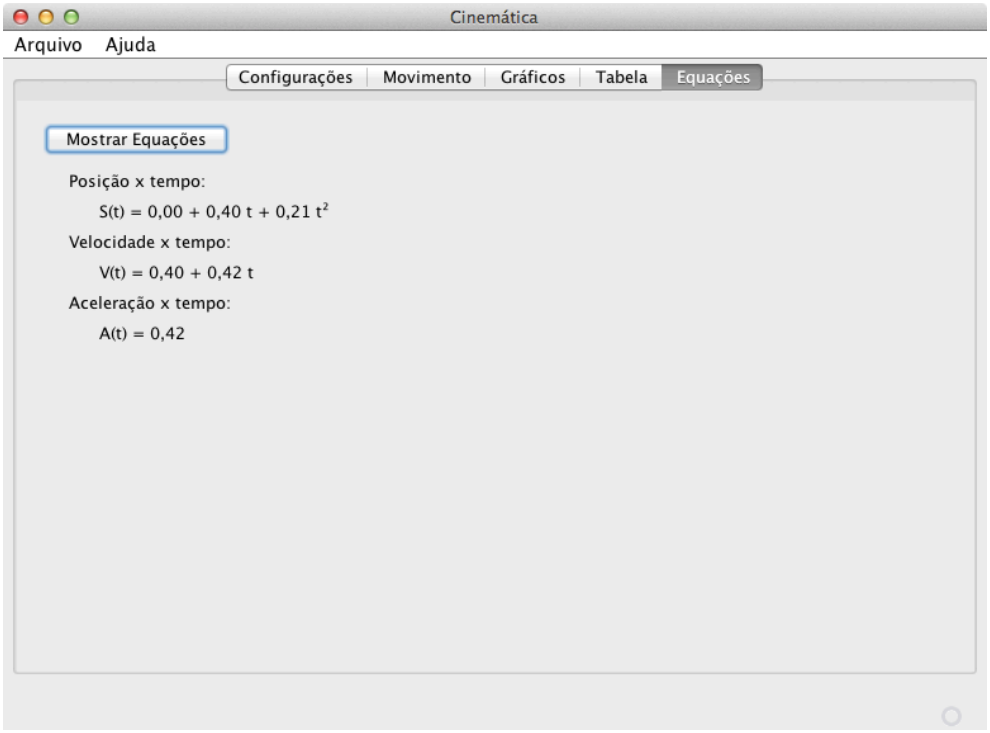

Figura 3.5 – Tela das equações da posição, velocidade e aceleração
Fonte: Autor

3.2 Diagramas da UML

A aplicação proposta neste trabalho possui quatro telas principais, onde podem ser acessadas as configurações do experimento, a visualização do movimento, os gráficos e as tabelas, conforme pode ser observado na figura 3.6.

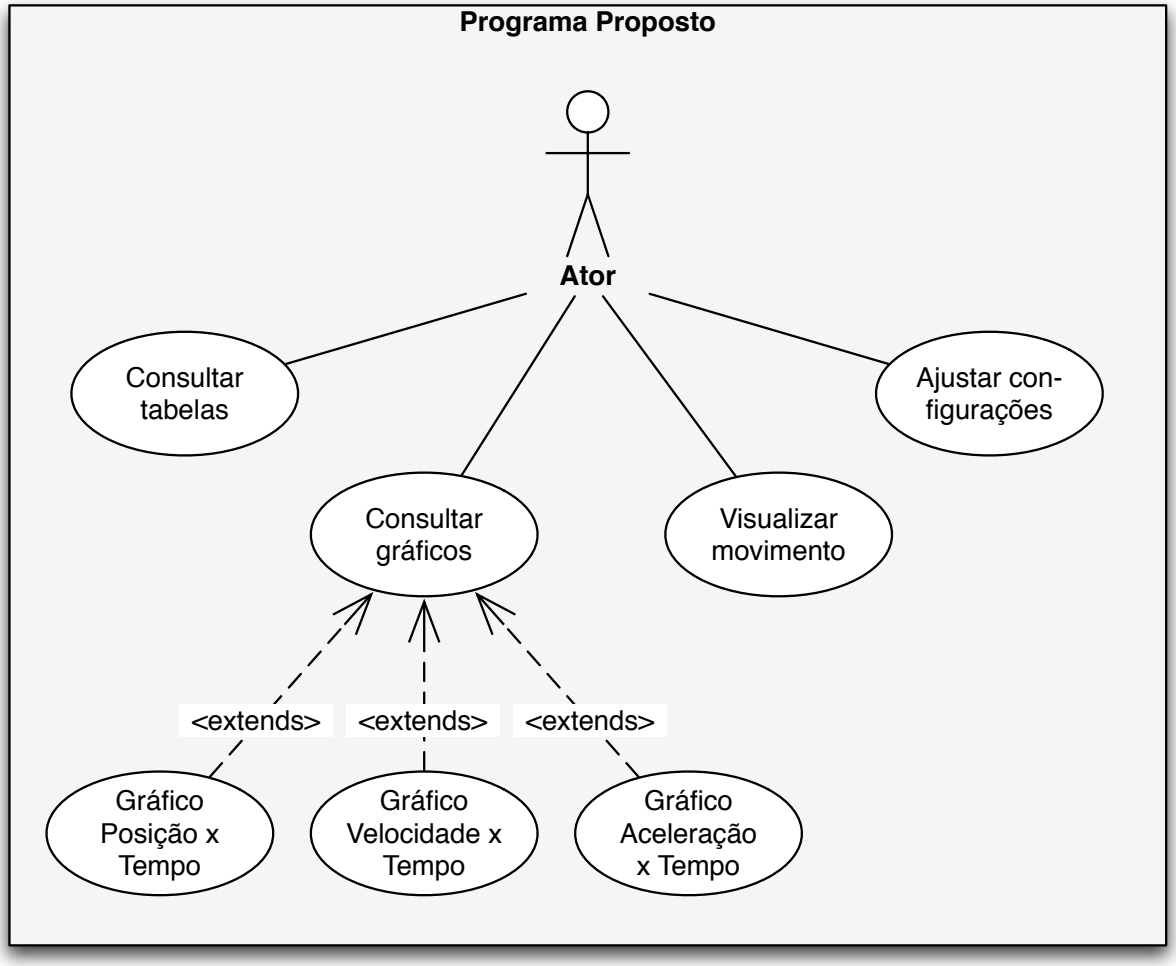

Figura 3.6 – Diagrama de caso de uso da aplicação proposta
Fonte: Autor

O diagrama representado na figura 3.6 é um diagrama de caso de uso, proposto pela *Unified Modeling Language* (Linguagem Unificada de Modelagem – UML). A UML é uma linguagem gráfica para visualização, especificação, construção e documentação de artefatos de programas de computador. Os casos de uso demonstram o comportamento pretendido do programa, sem especificar como este comportamento é implementado. É um diagrama de leitura simples, na visão do usuário, que comumente recebe o nome de "ator" (BOOCH, RUMBAUGH e JACOBSON, 2000).

Do mesmo modo, o diagrama de atividades da aplicação traz um panorama geral das

possibilidades de escolha do ator, bem como algumas funções implementadas pela aplicação proposta neste trabalho (PRESSMAN, 2006). Na figura 3.7 é mostrado o diagrama de fluxo desta aplicação.

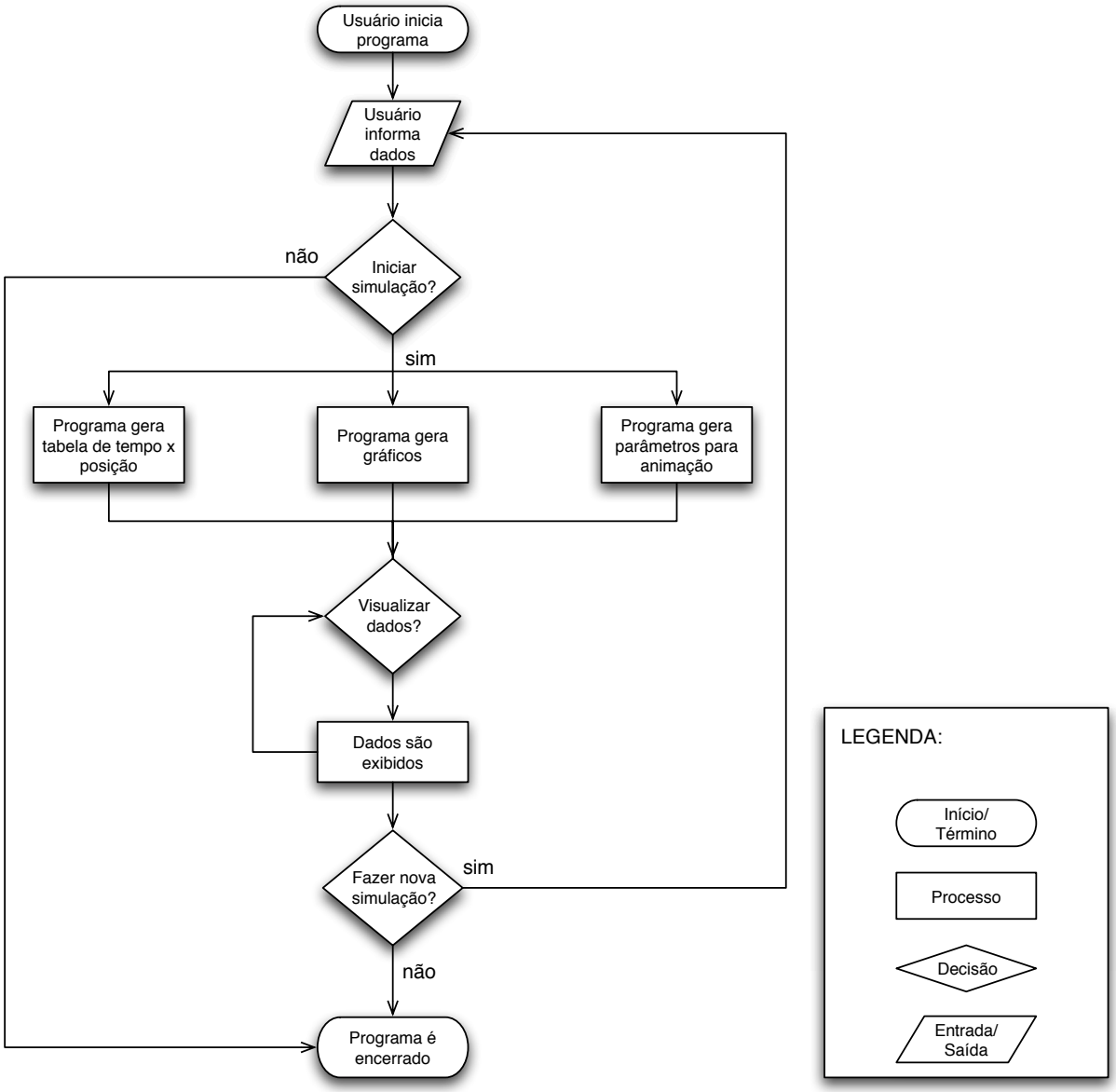

Figura 3.7 – Diagrama de atividades da aplicação proposta
Fonte: Autor

Não só o diagrama de caso de uso e o fluxograma são elementos importantes para a concepção de um programa orientado a objetos, mas também o diagrama de classes. Este, representa uma classe de objetos com um retângulo que possui três seções. A primeira seção contém o nome da classe, a segunda seção contém os atributos da classe e a terceira seção contém as operações associadas àquela classe (SOMMERVILLE, 2005).

Dessa forma, é apresentado na figura 3.8 o diagrama de classes da aplicação proposta

neste projeto. Na parte esquerda da figura, é observada a classe “Ator”, que possui “nome” e “titulação” como atributos. A mesma classe pode realizar a operação “definirParâmetros”. Cada ator pode realizar nenhum ou vários experimentos (conforme texto inserido acima da linha que conecta a classe “Ator” à classe “Experimento”). Cada “Experimento”, por sua vez, possui um “DetalheDoExperimento”. Por último, três gráficos são mantidos (incluídos, atualizados ou apagados) para cada “DetalheDoExperimento”.

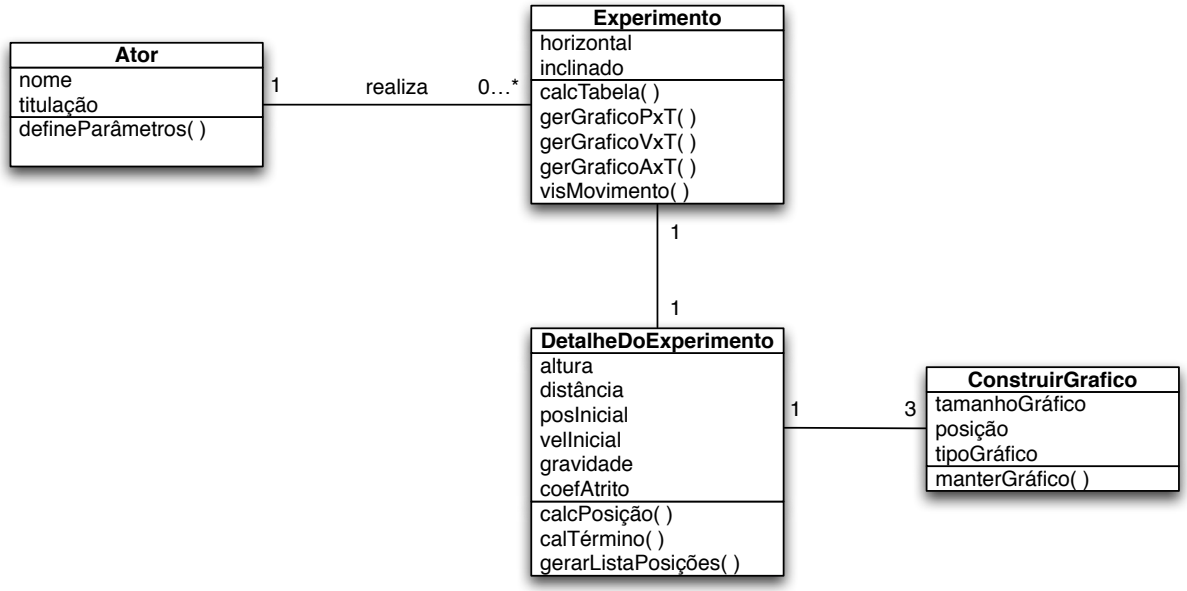

Figura 3.8 – Diagrama de classes da aplicação proposta
Fonte: Autor

No desenvolvimento de programas, é comum utilizar uma IDE (*Integrated Development Environment* – Ambiente de Desenvolvimento Integrado). A maioria das IDEs permitem escrever e editar códigos, ver erros enquanto o código é escrito, ver o código marcado em cores, automatizar tarefas repetitivas, compilar o código, entre outras tarefas e rotinas (ORACLE, 2011).

O ambiente de desenvolvimento utilizado na aplicação proposta neste trabalho foi o NetBeans 7.0.1, distribuído gratuitamente. A razão da escolha deste ambiente de desenvolvimento foi a presença de ferramentas de criação de interfaces de usuário, conforme mostrado na figura 3.9. Pode ser observado, na figura, que na parte direita do programa, são oferecidos diversos elementos de interface como painéis, botões, campos de preenchimento, listas de opções, entre outros (NETBEANS, 2011).

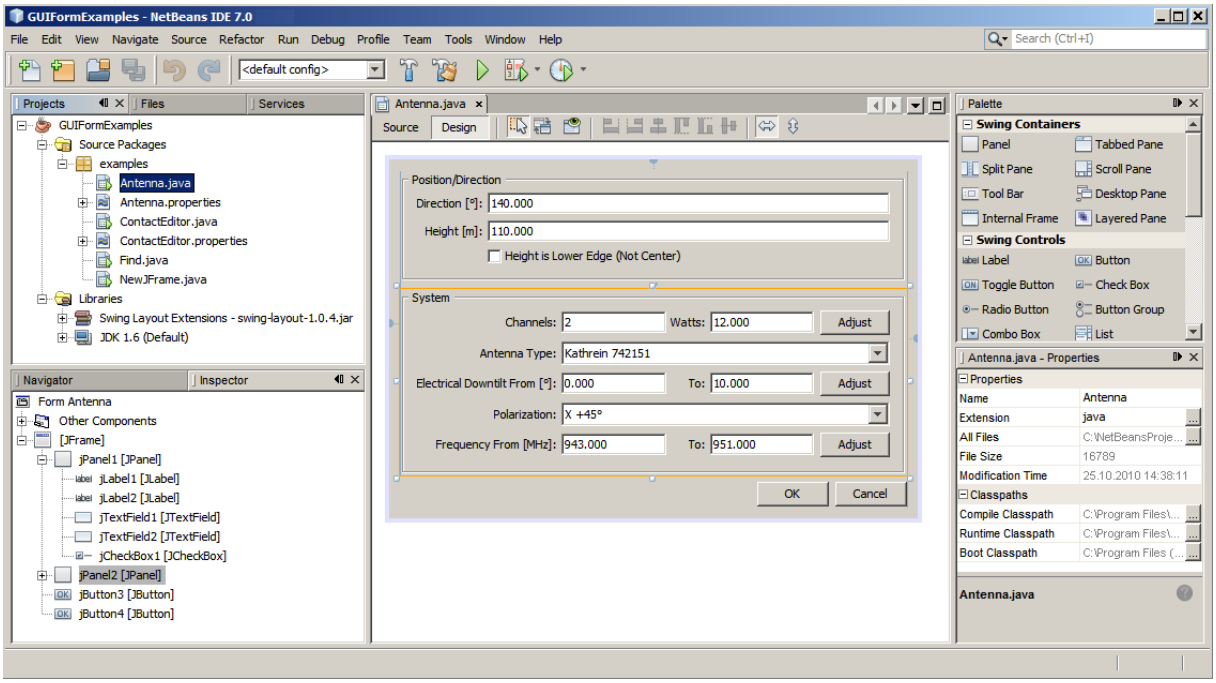

Figura 3.9 – Elementos de criação de interface do usuário do NetBeans
Fonte: NETBEANS, 2011

A aplicação proposta neste trabalho possui cinco classes principais, a saber, CinematicaApp, CinematicaView, Grafico, Logica, Ponto e Simulador e ainda uma sexta classe, CinematicaAboutBox, responsável pela janela "Sobre" da aplicação. As classes CinematicaApp, CinematicaView e CinematicaAboutBox foram criadas pelo próprio NetBeans, ao iniciar um novo projeto do tipo *Java Desktop Application* (Aplicação *Java* para *Desktop*). A classe CinematicaApp é a classe principal da aplicação (contém o "*main*") e é encarregada apenas de iniciar a aplicação.

3.3 Trechos do código

A classe CinematicaView possui duas ramificações: *Source* e *Design* (Código e Desenho). Na parte de desenho, foram arrastados os elementos necessários para ajustar os parâmetros do experimento, funcionalidade presente no NetBeans, conforme mostrado na figura 3.9 e tratado na seção 3.2. Na parte de código, foram escritos os códigos que controlam os elementos da parte de desenho. Dessa forma, na figura 3.10 é mostrada a criação da primeira tela, chamada "Configurações" e, em seguida, são mostrados alguns trechos de códigos para cada tipo de elemento desenhado.

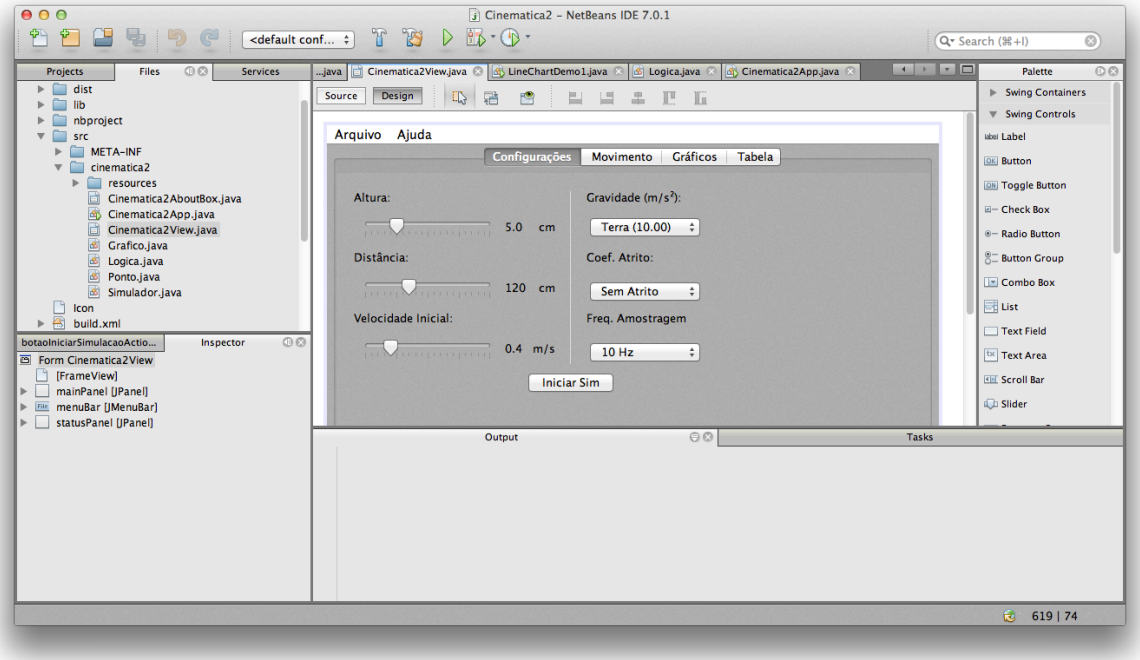

Figura 3.10 – Criação da interface de usuário no NetBeans
Fonte: Autor

Para criar o *slider* (botão deslizante) Altura, foi escrito o seguinte código:

```
1  private void sliderAlturaMouseReleased
2  (java.awt.event.MouseEvent evt) {
3
4        objLogica.altura = (double) sliderAltura.getValue();
5
6        //Faz o label do slider corresponder ao valor apontado
7        labelDinamicoAltura.setText(String.valueOf(objLogica.altura));
8        //Divide por 100 para transformar em centímetros
9        objLogica.altura = objLogica.altura/100.0;
10 }
```

Na linha 1 o próprio NetBeans criou o método, uma vez escolhida a ação para este elemento *slider* (soltar o botão do mouse). Na linha 4 é fornecido como valor para o atributo "altura" o valor do *slider*, ou seja, a posição do ponteiro do elemento. Em seguida, na linha 7, um *label* (texto) previamente escrito é alterado para refletir o valor do *slider*. Por último, na linha 9, o valor atribuído na linha 4 é dividido por 100, uma vez que será apresentado na tela na unidade de centímetros, mas usado nas equações na unidade de metros.

Códigos semelhantes foram escritos para os outros dois *sliders*, "Distância" e "Velocidade Inicial".

Para criar o *combobox* (lista de escolhas) Gravidade, foi escrito o seguinte código:

```
1  private void comboGravidadeActionPerformed
2  (java.awt.event.MouseEvent evt) {
3
4        switch (comboGravidade.getSelectedIndex()){
5            case 0:
6                //Gravidade: Terra (10.0)
7                objLogica.gravidade = 10.0;
8                break;
9            case 1:
10               //Gravidade: Terra (9.81)
11               objLogica.gravidade = 9.81;
12               break;
13           case 2:
14               //Gravidade: Lua (1/6 * Terra)
15               objLogica.gravidade = 9.81 / 6;
16               break;
17           case 3:
18               //Gravidade: Marte (0.38 * Terra)
19               objLogica.gravidade = 9.81 * 0.38;
20               break;
21       }
22 }
```

Na linha 1 o próprio NetBeans criou o método, uma vez escolhida a ação para este elemento *slider* (ação executada). Na linha 4 é apresentado um comando de tomada de decisão, que recebe o índice do item selecionado daquele *combobox*. Os itens são inseridos no *combobox* na parte de desenho do NetBeans, um a um. Então, a estrutura de tomada de decisão analisa qual foi o item escolhido (linhas 5, 9, 13 e 17) e, de acordo com sua posição, define um valor ao atributo gravidade (linhas 7, 11, 15 e 19), encerrando posteriormente a estrutura de decisão (comando *break*). Pensando no aspecto didático da aplicação, não é permitido ao usuário escolher um valor arbitrário para a gravidade. Porém, caso seja necessário, basta acrescentar mais casos a serem escolhidos (*case*) e seus respectivos valores.

Códigos semelhantes foram escritos para os outros dois *comboboxes*, “Coeficiente de Atrito” e “Frequência de Amostragem”.

Para criar o botão que inicia uma nova simulação, foi escrito o seguinte código:

```
1  private void botaoIniciarSimulacaoActionPerformed
2  (java.awt.event.MouseEvent evt) {
3
4        objLogica.tempo = 0;
5
6        if (objLogica.altura==0 && objLogica.velocidadeInicial < 0.3) {
7            JOptionPane.showMessageDialog(null,"Como o percurso é  "
8            + "horizontal (altura = 0), uma velocidade inicial mínima de "
9            + 0.3 m/s é exigida.", "Erro!", JOptionPane.ERROR_MESSAGE);
10       } else {
```

```
11
12      JFreeChart grafico = ChartFactory.createScatterPlot("Gráfico 1 -
13                                  Posição versus Tempo",
14                                  "Tempo",           //eixo X
15                                  "Posição",         //eixo Y
16                                  data,              //dados do XYSeries
17                                  PlotOrientation.VERTICAL,
18                                  false,                        //Legenda
19                                  true,                         //Tooltips
20                                  false                         //URL
21                                  );
22
23      Simulador objSimulador = new Simulador(model, serie1, data);
24      objSimulador.config = objLogica;
25
26      java.util.Timer timer = new java.util.Timer();
27      timer.schedule(objSimulador, 500, 500/objLogica.freqAmostragem);
28
29      }
30  }
```

Na linha 1 o próprio NetBeans criou o método, uma vez escolhida a ação para este elemento *slider* (ação executada). Na linha 4, é atribuído o valor 0 à variável tempo, uma vez que se trata de uma nova simulação. Em seguida, uma estrutura de decisão do tipo *if-else* examina se, para o percurso horizontal, o ator atribuiu uma velocidade inicial mínima de 0,3 m/s, para que o carro não fique parado. Caso esta condição seja aceita, um novo gráfico é preparado (linhas 12 a 21), com os parâmetros necessários (título do gráfico, título do eixo das abscissas, título do eixo das ordenadas, orientação de plotagem, legenda para o gráfico, legendas para os pontos e criação de URL para o gráfico) e uma nova instância do objeto Simulador é iniciada (linha 23), fornecido como parâmetros "model", que é o modelo de tabela previamente definido, "serie1", que é cada ponto do gráfico e "data", que é o encapsulamento dos pontos (serie1), passado mais a seguir como parâmetro para a criação do gráfico. São também passadas como configurações do simulador as variáveis da classe Logica (linha 24), que foram ajustados pelos *sliders*. Em seguida, um novo *timer* é instanciado na linha 26. Este *timer* é responsável por iniciar e cancelar (sem sair) a aplicação. Por fim, são atribuídos valores a um agendador (*schedule*). Os atributos deste agendador são: tarefa a ser executada, início da tarefa (em milissegundos) e repetição da tarefa (em milissegundos).

3.4 Classes da aplicação

Além das classes CinematicaApp, CinematicaView e CinematicaAboutBox, fornecidas pelo próprio NetBeans e tratadas na seção 3.3, a aplicação proposta neste trabalho teve ainda outras três classes: Logica, Ponto e Simulador.

Os códigos completos podem ser lidos nas seções Apêndice (propostos pelo autor da aplicação) e Anexos (gerados pelo NetBeans).

Um trecho do código da classe "Logica" é mostrado a diante. Nesta classe, estão os elementos que modelam o comportamento da aplicação, bem como os parâmetros físicos e matemáticos.

```
 1  package cinematica;
 2
 3  public class Logica {
 4        public double altura = 0.05;
 5        public double distancia = 1.2;
 6        public double posicaoInicial = 0.0;
 7        public double velocidadeInicial = 0.4;
 8        public double gravidade = 10.0;
 9        public double coeficienteAtrito = 0.0;
10        public double posicao() {
11            //Equação do espaço:
12            //S(t) = So + Vo t + (1/2) g (sen a - u cos a) t^2
13            double espaco = posicaoInicial + velocidadeInicial *
14                            tempoFatiado() + 0.5 * gravidade * (seno() -
15                            coeficienteAtrito * cosseno()) *
16                            Math.pow(tempoFatiado(), 2);
17            return espaco;
18        }
19
20        public double tempoFatiado() {
21            return tempo / freqAmostragem;
22        }
23
24        private double hipotenusa() {
25            return Math.sqrt(Math.pow(altura, 2) + Math.pow(distancia,2));
26        }
27
28        public boolean maximoIteracoes() {
29            if (posicao() <= hipotenusa())
30                return true;
31            else
32                return false;
33        }
34  }
```

No trecho apresentado, as variáveis da classe Logica são iniciadas (linhas 4 a 9), com valores que correspondem aos valores iniciais dos *sliders* (botões deslizantes) e das *combobox* (listas de opções) apresentadas na primeira aba, configurações. Em seguida, a equação do espaço (linha 13) é escrita. De forma similar, são escritas as equações da velocidade e aceleração. Por fim, algumas funções realizam cálculos auxiliares (tempoFatiado, seno, cosseno e hipotenusa) e, a última função (linha 28), verifica se o carro chegou ao fim do percurso.

A classe Ponto possui apenas o objeto de mesmo nome, e seu código é descrito a seguir:

```
1  package cinematica;
2
3  //Cria um objeto ponto, que armazenará cada tempo
4  //e sua respectiva posição
5  public class Ponto {
6      public double x;   //Aqui ele receberá o "tempoFatiado"
7      public double y;   //Aqui ele receberá a "posicaoAtual"
8      public double v;   //Aqui ele receberá a "velocidade"
9      public double a;   //Aqui ele receberá a "aceleração"
10 }
```

Por último, a classe Simulador. Esta classe, por sua vez, é responsável por preencher a tabela (conforme figura 3.4) e os gráficos (conforme figuras 3.3a, 3.3b e 3.3c). Um trecho do código da classe Simulador é descrito a seguir:

```
1  package cinematica;
2
3  import java.util.ArrayList;
4  import java.util.TimerTask;
5  import javax.swing.JOptionPane;
6  import javax.swing.table.DefaultTableModel;
7  import org.jfree.chart.JFreeChart;
8  import org.jfree.data.xy.XYSeries;
9  import org.jfree.data.xy.XYSeriesCollection;
10
11 public class Simulador extends TimerTask {
12
13     private DefaultTableModel model;
14
15     private XYSeries serie1;
16     private XYSeriesCollection data;
17     private JFreeChart grafico;
18
19     public Logica config;
20     private int iteracaoAtual = 0;
21     private int codErro = 0;
22     public ArrayList<Ponto> listaPosicoes = new ArrayList<Ponto>();
```

```
23
24      public Simulador(DefaultTableModel model, XYSeries serie1,
25  XYSeriesCollection data) {
26          this.model = model;
27          this.serie1 = serie1;
28          this.data = data;
29      }
30      public void run() {
31          iteracaoAtual++;
32
33          //Instancia um novo objeto Ponto
34          Ponto p = new Ponto();
35
36          //Atribui os valores do Ponto e adiciona-o à ArrayList
37          p.x = config.tempoFatiado();
38          p.y = config.posicao();
39          p.v = config.velocidade();
40          p.a = config.aceleracao();
41          listaPosicoes.add(p);
42
43          //Adiciona o Ponto atual à tabela (nova linha)
44          model.addRow(new Object[] {p.x, p.y});
45
46          //Adiciona o Ponto atual aos gráficos
47          serie1.add(p.x,p.y);
48          config.tempo++;
49          //Para a iteração.
50          if (config.maximoIteracoes() == false || codErro != 0) {
51              data.addSeries(serie1);
52              cancel();
53          }
54      }
55  }
```

Primeiramente, são importadas as bibliotecas necessárias para sua execução (linhas 3 a 9). Em seguida, é feito o construtor da classe Simulador, que estende uma função de uma das bibliotecas importadas, para uso dos métodos *run* e *cancel*. Depois, são feitas as declarações necessárias: modelo de tabela (model), séries, coleção de séries e gráfico (serie1, data e grafico), uma nova lógica, um contador de iterações, uma variável para controle de erros e, por último, uma pilha de pontos (linhas 13 a 22).

Adiante, na linha 24, a classe Simulador apresenta outro construtor, agora para variáveis que serão importadas pela classe CinematicaView para a criação dos gráficos. Na linha 30, a simulação começa a ser executada. A cada passo deste laço, um novo objeto ponto é criado. Cada ponto criado armazena, em seus atributos, o tempo, sua posição, sua velocidade e sua aceleração (linhas 37 a 40). Depois de calculadas estes atributos, aquele ponto é acrescentado à tabela (linha 44) e à série (linha 47). As séries, posteriormente, são

encapsuladas numa coleção de séries (linha 51) que será usada como parâmetro para a criação do gráfico (conforme tratado na seção 3.3). Este encapsulamento dos dados, feito somente após o término das iterações (linhas 50 e 51) é mais indicado pela documentação da biblioteca JFreeChart, porém, caso seja desejado, o encapsulamento pode ser feito a cada iteração, gerando, assim, um gráfico dinâmico.

CAPÍTULO 4 - TESTES E RESULTADOS

A fim de fazer testes e comparações entre a aplicação proposta e um experimento de laboratório, foram utilizadas ferramentas da companhia Pasco. A Pasco “fornece, a quarenta e cinco anos, soluções inovativas para o ensino de ciência”. Entre seus componentes fabricados, estão sensores, interfaces, modelos em escala de carros, programas de computador, entre outros (PASCO SCIENTIFICS, 2011).

Os laboratórios de física do Centro Universitário de Brasília – UniCEUB são equipados com equipamentos da referida companhia. Para realizar em laboratório os dois experimentos que a aplicação proposta neste trabalho faz (Plano Horizontal e Plano Inclinado), são necessários quatro equipamentos da Pasco: a interface que adquire os dados dos sensores, transforma estes dados em sinais digitais e os transfere ao computador; um sensor de movimento, que funciona por meio de ultrassom e coleta as posições de objetos colocados a sua frente; o trilho, que é o percurso e o carro dinâmico que, ao ser encaixado no trilho, funciona quase em condições ideais, ou seja, sem atrito.

Na figura 4.1 é mostrada a interface que a Pasco utiliza para conectar os diversos sensores e transmitir os dados para o aplicativo Data Studio (também da Pasco), no computador. A interface é conectada ao computador por cabo USB.

Figura 4.1 – Interface Science Workshop, da Pasco
Fonte: Autor

Na figura 4.2 é mostrado o sensor de movimento. Por se tratar de um sensor digital, o mesmo é conectado às portas 1 e 2 da interface.

Figura 4.2 – Sensor de movimento da Pasco
Fonte: Autor

Na figura 4.3 é mostrado o trilho com o carro dinâmico já montado sobre o mesmo. Ao fundo, é possível ver o sensor de movimento, também encaixado no trilho.

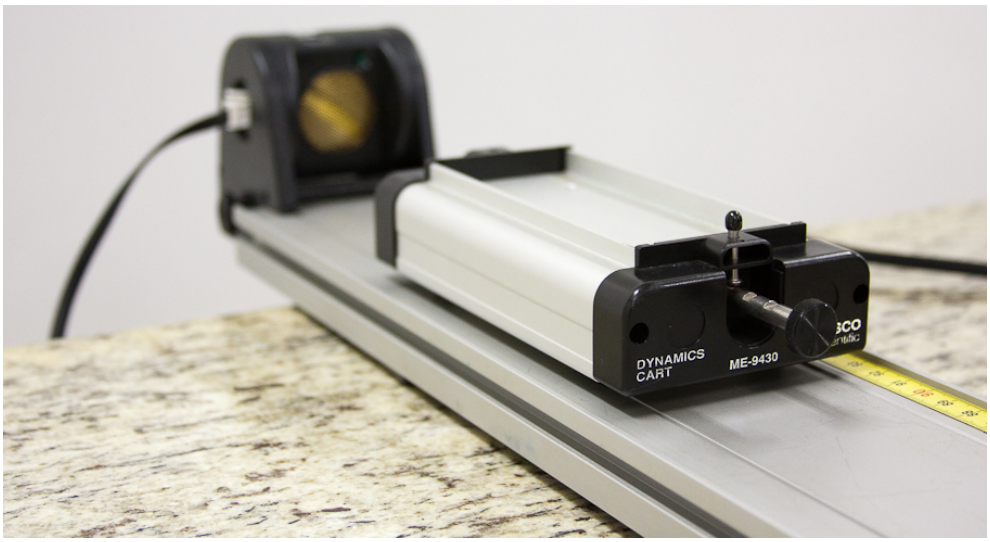

Figura 4.3 – Conjunto composto pelo trilho, carro e sensor de movimento
Fonte: Autor

Para o experimento “Plano Inclinado”, o roteiro de laboratório usado na disciplina de Física Experimental do Centro Universitário de Brasília recomenda adicionar blocos de madeira próximos ao sensor de movimento, alterando assim, a inclinação do trilho em relação à mesa, conforme mostrado na figura 4.4.

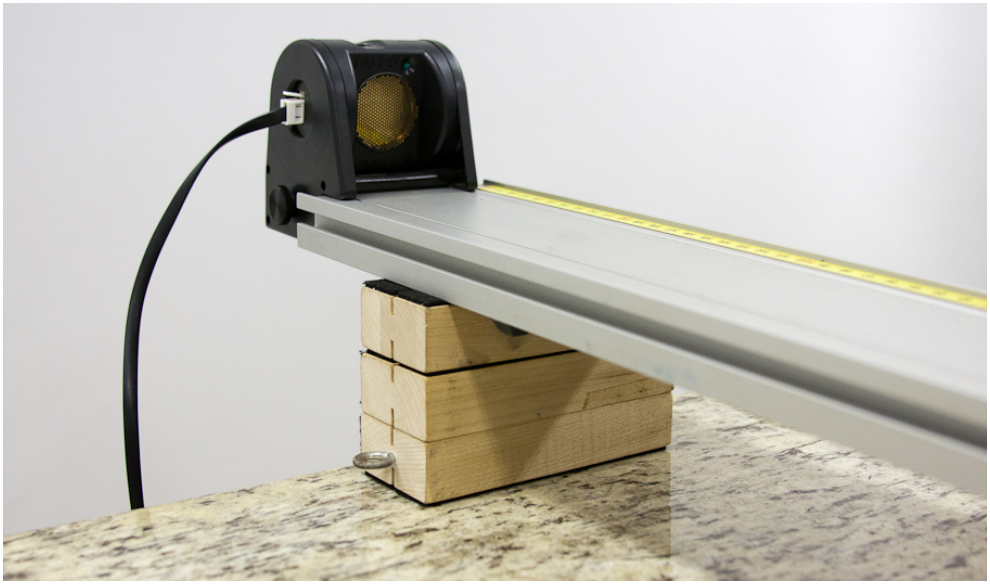

Figura 4.4 – Inclinação do trilho em relação à mesa
Fonte: Autor

Cada bloco de madeira adicionado aumenta a altura em 3 centímetros. Para efeitos de comparação, foram feitos, ao todo, doze experimentos: seis com o plano horizontal e outros seis com o plano inclinado. A íntegra das tabelas de comparativos pode ser encontrada na seção Apêndice. Duas tabelas comparativas, para cada experimento, são disponibilizadas nesta sessão, conforme mostrado nas tabelas 3, 4, 5 e 6. Por fim, é mostrada na figura 4.5 a tela inicial do Data Studio.

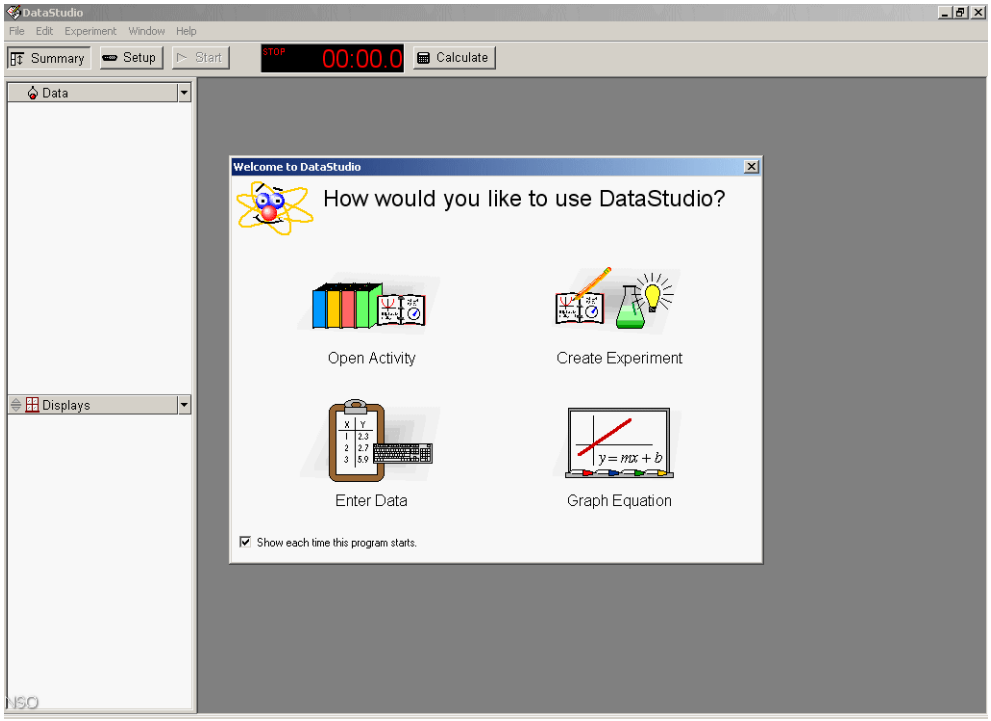

Figura 4.5 – Tela Inicial do Data Studio
Fonte: Autor

4.1 Comparativo – Plano Horizontal

A fim de comparar os resultados obtidos na aplicação proposta com os resultados obtidos num ambiente de laboratório, para o experimento de plano horizontal, foram feitas seis tomadas de dados, das quais duas estão aqui destacadas, nas tabelas 4 e 5. O fator que varia entre as referidas medições é a velocidade inicial do carro dinâmico e a frequência de amostragem.

A tabela 3, mostrada a seguir, compara os valores obtidos no laboratório e na aplicação proposta, segundo uma velocidade inicial de 0,603 metros por segundo e uma frequência de amostragem de 25 Hz.

De acordo com a teoria discutida no capítulo 2, seção 5.1, foi feito o ajuste linear para os dados obtidos no laboratório e, seus coeficientes, introduzidos na aplicação proposta.

Tabela 3 – Primeiro comparativo de dados obtidos no programa Data Studio (Pasco) e na aplicação proposta neste trabalho e a diferença percentual entre os dados, experimento de plano horizontal

Data Studio		Aplicação Proposta		Diferença
Tempo (s)	Posição (m)	Tempo (s)	Posição (m)	D (%)
0,2401	0,1639	0,24	0,16872	2,94
0,2801	0,1906	0,28	0,19284	1,18
0,32	0,216	0,32	0,21696	0,44
0,36	0,2403	0,36	0,24108	0,32
0,3999	0,2645	0,4	0,2652	0,26
0,4398	0,289	0,44	0,28932	0,11
0,4798	0,3132	0,48	0,31344	0,08
0,5197	0,3375	0,52	0,33756	0,02
0,5597	0,3615	0,56	0,36168	0,05
0,5996	0,3856	0,6	0,3858	0,05
0,6396	0,4099	0,64	0,40992	0,00
0,6795	0,434	0,68	0,43404	0,01
0,7194	0,4582	0,72	0,45816	0,01
0,7594	0,4825	0,76	0,48228	0,05
0,7993	0,5065	0,8	0,5064	0,02
0,8393	0,5308	0,84	0,53052	0,05
0,8792	0,5549	0,88	0,55464	0,05
0,9191	0,579	0,92	0,57876	0,04
0,9591	0,6035	0,96	0,60288	0,10
0,999	0,6276	1	0,627	0,10
1,039	0,6515	1,04	0,65112	0,06
1,0789	0,6756	1,08	0,67524	0,05
1,1188	0,6995	1,12	0,69936	0,02

1,1588	0,7234	1,16	0,72348	0,01
1,1987	0,7473	1,2	0,7476	0,04
1,2387	0,7712	1,24	0,77172	0,07
1,2786	0,795	1,28	0,79584	0,11
1,3185	0,8187	1,32	0,81996	0,15
1,3585	0,8426	1,36	0,84408	0,18
1,3984	0,8664	1,4	0,8682	0,21
1,4383	0,8899	1,44	0,89232	0,27
1,4783	0,9137	1,48	0,91644	0,30
1,5182	0,9374	1,52	0,94056	0,34
1,5582	0,961	1,56	0,96468	0,38
1,5981	0,985	1,6	0,9888	0,39
1,638	1,0084	1,64	1,0129	0,45
			Média:	0,28

Fonte: Autor

A figura 4.6, mostrada a seguir, fornece um comparativo gráfico (gráfico de dispersão) entre os valores de posição do Data Studio e da aplicação proposta neste trabalho.

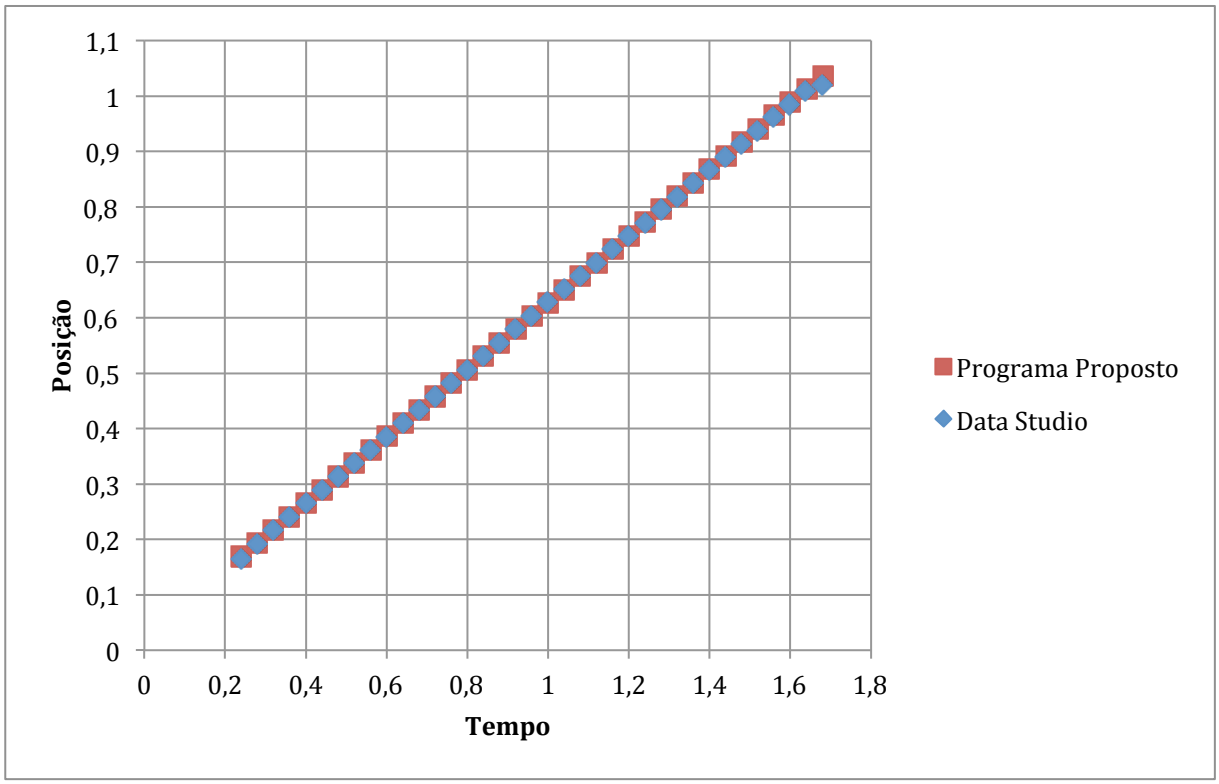

Figura 4.6 – Primeiro comparativo dos dados obtidos no Data Studio e na aplicação proposta (plano horizontal)
Fonte: Autor

A tabela 4, mostrada a seguir, compara os valores obtidos no laboratório e na aplicação proposta, segundo uma velocidade inicial de 0,743 metros por segundo e uma frequência de amostragem de 50 Hz.

De acordo com a teoria discutida no capítulo 2, seção 5.1, foi feito o ajuste linear para os dados obtidos no laboratório e, seus coeficientes, introduzidos na aplicação proposta.

Tabela 4 – Segundo comparativo de dados obtidos no programa Data Studio (Pasco) e na aplicação proposta neste trabalho e a diferença percentual entre os dados, experimento de plano horizontal

Data Studio		Aplicação Proposta		Diferença
Tempo (s)	Posição (m)	Tempo (s)	Posição (m)	D (%)
0,3381	0,1949	0,34	0,19752	1,34
0,358	0,2097	0,36	0,21238	1,28
0,3779	0,225	0,38	0,22724	1,00
0,3978	0,2423	0,4	0,2421	0,08
0,4176	0,2542	0,42	0,25696	1,09
0,4375	0,2695	0,44	0,27182	0,86
0,4574	0,2845	0,46	0,28668	0,77
0,4773	0,2991	0,48	0,30154	0,82
0,4972	0,3142	0,5	0,3164	0,70
0,5171	0,3287	0,52	0,33126	0,78
0,537	0,3437	0,54	0,34612	0,70
0,5569	0,3586	0,56	0,36098	0,66
0,5767	0,3732	0,58	0,37584	0,71
0,5966	0,3875	0,6	0,3907	0,83
0,6165	0,4027	0,62	0,40556	0,71
0,6364	0,4176	0,64	0,42042	0,68
0,6563	0,4326	0,66	0,43528	0,62
0,6762	0,4472	0,68	0,45014	0,66
0,6961	0,4622	0,7	0,465	0,61
0,7159	0,4773	0,72	0,47986	0,54
0,7358	0,4919	0,74	0,49472	0,57
0,7557	0,5069	0,76	0,50958	0,53
0,7756	0,5218	0,78	0,52444	0,51
0,7955	0,5365	0,8	0,5393	0,52
0,8154	0,5514	0,82	0,55416	0,50
0,8353	0,5659	0,84	0,56902	0,55
0,8551	0,5808	0,86	0,58388	0,53
0,875	0,5958	0,88	0,59874	0,49
0,8949	0,6103	0,9	0,6136	0,54
0,9148	0,6252	0,92	0,62846	0,52
0,9347	0,6402	0,94	0,64332	0,49
0,9546	0,6546	0,96	0,65818	0,55
0,9745	0,6696	0,98	0,67304	0,51
0,9943	0,684	1	0,6879	0,57
1,0142	0,6988	1,02	0,70276	0,57
1,0341	0,7138	1,04	0,71762	0,54
1,054	0,7282	1,06	0,73248	0,59
1,0739	0,743	1,08	0,74734	0,58

1,0938	0,758	1,1	0,7622	0,55
1,1137	0,7723	1,12	0,77706	0,62
1,1335	0,7872	1,14	0,79192	0,60
1,1534	0,8015	1,16	0,80678	0,66
1,1733	0,8163	1,18	0,82164	0,65
1,1932	0,8313	1,2	0,8365	0,63
1,2131	0,8456	1,22	0,85136	0,68
1,233	0,8603	1,24	0,86622	0,69
1,2529	0,8751	1,26	0,88108	0,68
1,2727	0,8896	1,28	0,89594	0,71
1,2926	0,9044	1,3	0,9108	0,71
1,3125	0,9187	1,32	0,92566	0,76
1,3324	0,9334	1,34	0,94052	0,76
			Média:	0,66

Fonte: Autor

A figura 4.7, mostrada a seguir, fornece um comparativo gráfico (gráfico de dispersão) entre os valores de posição do Data Studio e da aplicação proposta neste trabalho.

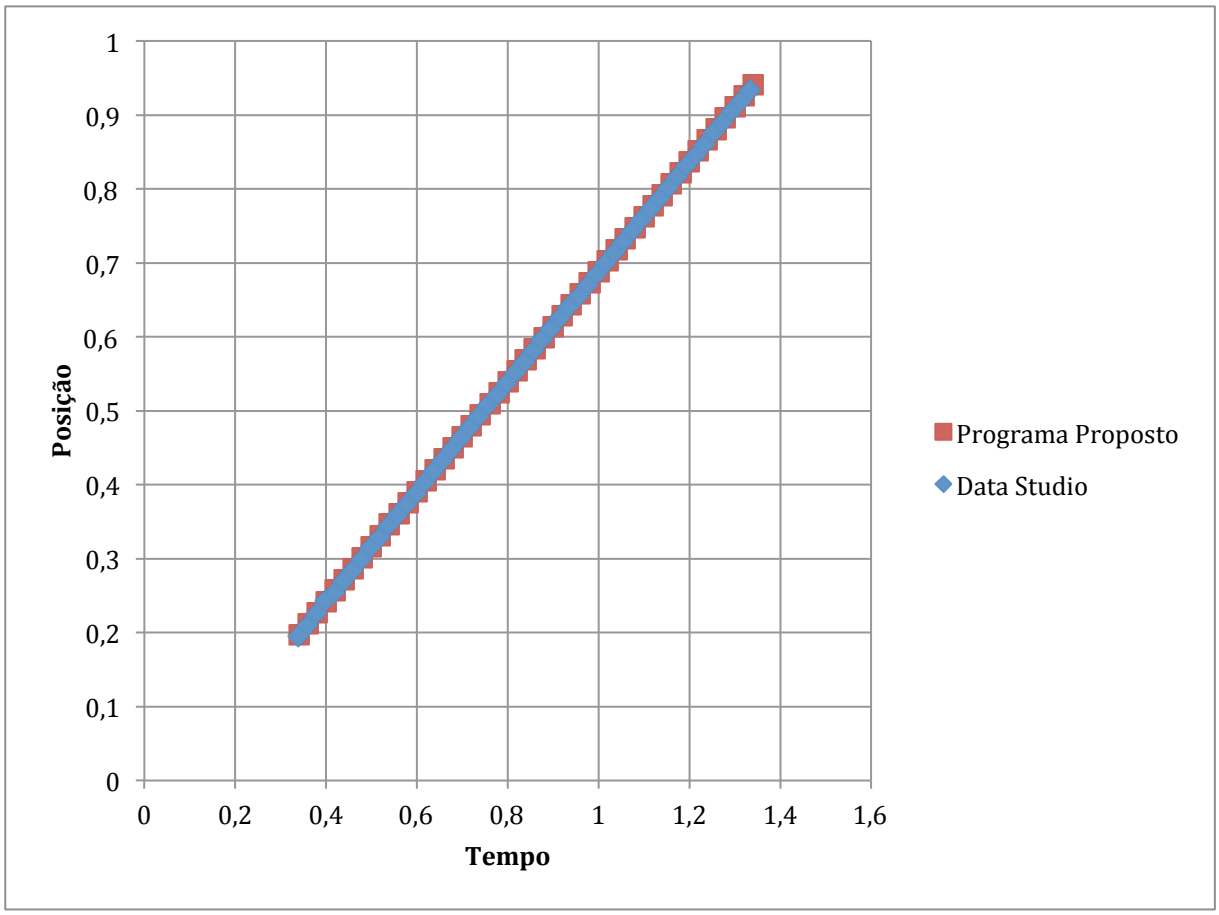

Figura 4.7 – Segundo comparativo dos dados obtidos no Data Studio e na aplicação proposta (plano horizontal)
Fonte: Autor

4.2 Comparativo – Plano Inclinado

A fim de comparar os resultados obtidos na aplicação proposta com os resultados obtidos num ambiente de laboratório, para o experimento de plano inclinado, foram feitas seis tomadas de dados, das quais duas estão aqui destacadas, nas tabelas 6 e 7. Os fatores que variam entre as referidas medições são a frequência de amostragem e a altura da inclinação da pista. A velocidade inicial é nula, uma vez que o carro foi abandonado.

A tabela 5, mostrada a seguir, compara os valores obtidos no laboratório e na aplicação proposta, segundo uma frequência de amostragem de 10 Hz e uma altura de 8,5 cm.

De acordo com a teoria discutida no capítulo 2, seção 5.2, foi feito o ajuste polinomial para os dados obtidos no laboratório e, seus coeficientes, introduzidos na aplicação proposta.

Tabela 5 – Primeiro comparativo de dados obtidos no programa Data Studio (Pasco) e na aplicação proposta neste trabalho e a diferença percentual entre os dados, experimento de plano inclinado

Data Studio		Aplicação Proposta		Diferença
Tempo (s)	Posição (m)	Tempo (s)	Posição (m)	D (%)
1,0993	0,2028	1,1	0,201494308	0,64
1,1994	0,2384	1,2	0,23999488	0,67
1,2994	0,2783	1,3	0,282625935	1,55
1,3995	0,3277	1,4	0,329387475	0,51
1,4996	0,3798	1,5	0,380279499	0,13
1,5997	0,4346	1,6	0,435302008	0,16
1,6998	0,4943	1,7	0,494455001	0,03
1,8	0,5557	1,8	0,557738479	0,37
1,9002	0,6233	1,9	0,625152441	0,30
2,0004	0,6952	2	0,696696888	0,22
2,1006	0,7707	2,1	0,772371818	0,22
2,2009	0,8509	2,2	0,852177234	0,15
2,3012	0,9341	2,3	0,936113134	0,22
			Média:	0,40

Fonte: Autor

A figura 4.8, mostrada a seguir, fornece um comparativo gráfico (gráfico de dispersão) entre os valores de posição do Data Studio e da aplicação proposta neste trabalho.

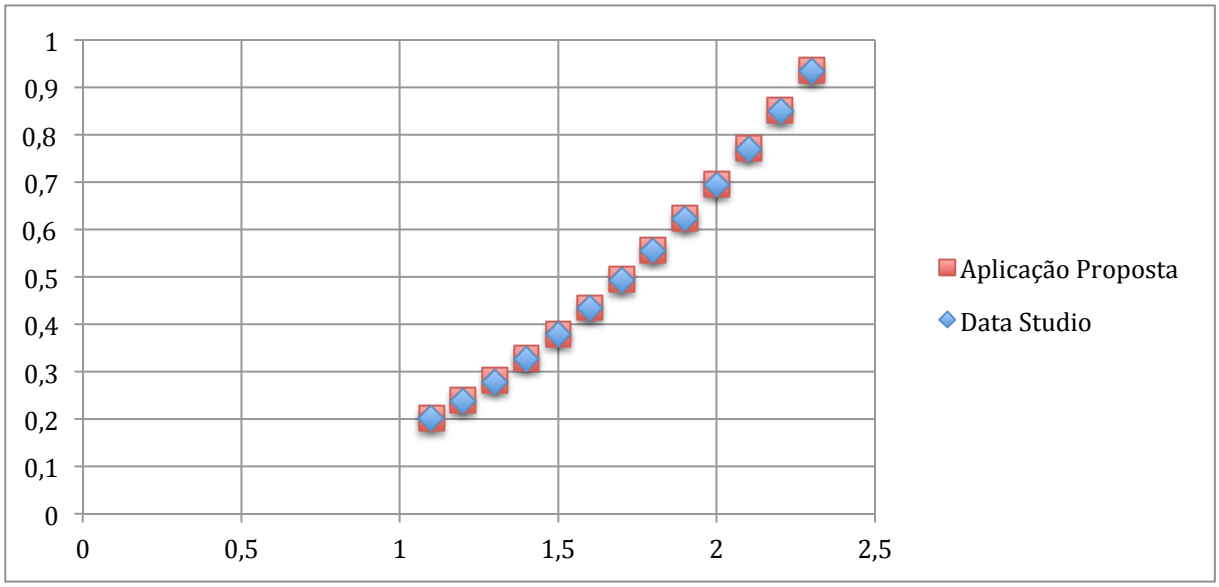

Figura 4.8 – Primeiro comparativo dos dados obtidos no Data Studio e na aplicação proposta (plano inclinado)
Fonte: Autor

A tabela 6, mostrada a seguir, compara os valores obtidos no laboratório e na aplicação proposta, segundo uma frequência de amostragem de 25 Hz e uma altura de 5,5 cm.

De acordo com a teoria discutida no capítulo 2, seção 5.2, foi feito o ajuste polinomial para os dados obtidos no laboratório e, seus coeficientes, introduzidos na aplicação proposta.

Tabela 6 – Segundo comparativo de dados obtidos no programa Data Studio (Pasco) e na aplicação proposta neste trabalho e a diferença percentual entre os dados, experimento de plano inclinado

Data Studio		Aplicação Proposta		Diferença
Tempo (s)	Posição (m)	Tempo (s)	Posição (m)	D (%)
0,9565	0,1608	0,96	0,164716691	2,44
0,9964	0,173	1	0,173940854	0,54
1,0362	0,1825	1,04	0,183435228	0,51
1,0761	0,1945	1,08	0,193199812	0,67
1,1159	0,2016	1,12	0,203234608	0,81
1,1558	0,2117	1,16	0,213539614	0,87
1,1957	0,2243	1,2	0,22411483	0,08
1,2355	0,2322	1,24	0,234960258	1,19
1,2754	0,2456	1,28	0,246075896	0,19
1,3153	0,2566	1,32	0,257461745	0,34
1,3551	0,2681	1,36	0,269117804	0,38
1,395	0,2798	1,4	0,281044074	0,44
1,4349	0,2921	1,44	0,293240556	0,39
1,4747	0,3046	1,48	0,305707247	0,36
1,5146	0,3173	1,52	0,31844415	0,36
1,5545	0,3301	1,56	0,331451263	0,41
1,5944	0,343	1,6	0,344728587	0,50

1,6342	0,3562	1,64	0,358276122	0,58
1,6741	0,3698	1,68	0,372093867	0,62
1,714	0,3836	1,72	0,386181823	0,67
1,7539	0,3978	1,76	0,40053999	0,69
1,7938	0,4123	1,8	0,415168368	0,70
1,8337	0,4271	1,84	0,430066956	0,69
1,8735	0,442	1,88	0,445235755	0,73
1,9134	0,4573	1,92	0,460674765	0,74
1,9533	0,4732	1,96	0,476383986	0,67
1,9932	0,489	2	0,492363417	0,69
2,0331	0,5052	2,04	0,508613059	0,68
2,073	0,5217	2,08	0,525132912	0,66
2,1129	0,5384	2,12	0,541922976	0,65
2,1528	0,5554	2,16	0,55898325	0,65
2,1927	0,5728	2,2	0,576313735	0,61
2,2326	0,5901	2,24	0,593914431	0,65
2,2725	0,608	2,28	0,611785337	0,62
2,3124	0,6259	2,32	0,629926454	0,64
2,3523	0,6441	2,36	0,648337782	0,66
2,3922	0,6627	2,4	0,667019321	0,65
2,4321	0,6815	2,44	0,68597107	0,66
2,472	0,7011	2,48	0,70519303	0,58
2,512	0,7205	2,52	0,724685201	0,58
2,5519	0,7401	2,56	0,744447583	0,59
2,5918	0,7601	2,6	0,764480175	0,58
2,6317	0,7802	2,64	0,784782978	0,59
2,6716	0,8005	2,68	0,805355992	0,61
2,7115	0,8213	2,72	0,826199217	0,60
2,7515	0,8421	2,76	0,847312652	0,62
2,7914	0,8634	2,8	0,868696298	0,61
2,8313	0,8849	2,84	0,890350155	0,62
2,8712	0,9068	2,88	0,912274222	0,60
2,9112	0,9288	2,92	0,9344685	0,61
2,9511	0,9515	2,96	0,956932989	0,57
2,991	0,9739	3	0,979667689	0,59
3,031	0,9967	3,04	1	0,60
			Média:	0,63

Fonte: Autor

A figura 4.9, mostrada a seguir, fornece um comparativo gráfico (gráfico de dispersão) entre os valores de posição do Data Studio e da aplicação proposta neste trabalho.

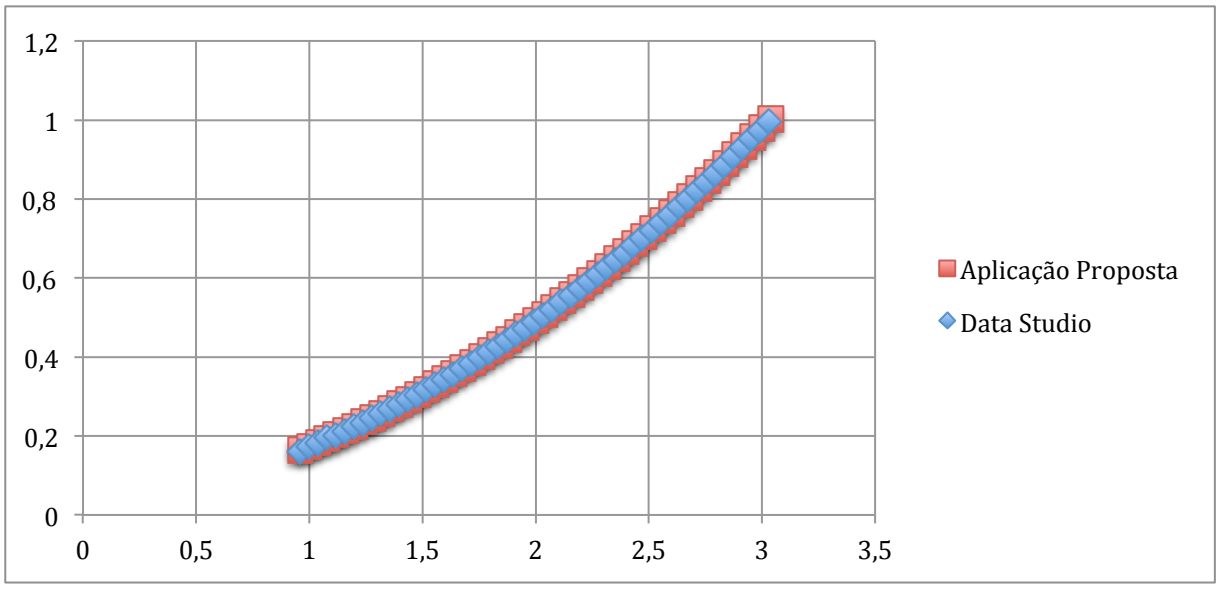

Figura 4.9 – Primeiro comparativo dos dados obtidos no Data Studio e na aplicação proposta (plano inclinado)
Fonte: Autor

CAPÍTULO 5 - CONCLUSÃO

Em face de tudo o que foi exposto anteriormente, principalmente com o que é defendido no quarto capítulo, conclui-se que as equações, gráficos e tabelas fornecidos pela aplicação tiveram êxito. A conclusão deste trabalho é feita sob três diferentes óticas: limitações da aplicação, facilidades da aplicação e sugestões de trabalhos futuros.

Quanto às limitações da aplicação, é mostrado no quarto capítulo deste trabalho, testes e comparativos, que os dados obtidos na aplicação proposta neste trabalho são muito próximos dos valores encontrados no ambiente laboratorial, usando equipamentos profissionais e específicos para este fim. Estes resultados, não só mostram que a aplicação é confiável, como corroboram e reafirmam a precisão dos equipamentos da Pasco. Em contrapartida, a aplicação proposta neste trabalho somente realiza uma classe de experimentos de física: a cinemática.

A respeito das facilidades da aplicação, tanto professores quanto alunos podem, a partir deste trabalho proposto, interagir com experimentos de cinemática, tornando o aprendizado de física mais moderno, incentivando a participação mais ativa do aluno e integrando conhecimentos teóricos e práticos.

Os equipamentos usados para comparativos, da marca Pasco, também permitem interação, porém, tais equipamentos são difíceis de serem adquiridos e só podem ser comprados no exterior. A tabela 7 mostra uma relação de preços, em libras esterlinas, para os respectivos equipamentos listados, de acordo com a lista de preço de um dos vendedores mundiais dos equipamentos Pasco. A lista de preços é válida até o dia 31 de dezembro de 2011 (FEEDBACK, 2011).

Tabela 7 – Lista de preços para montar um experimento de cinemática com equipamentos Pasco

Equipamento	Preço (£)
Interface 750 USB	640,00
Conjunto de trilho e carro dinâmico	378,00
Sensor de movimento	90,00
Total	1108,00

Fonte: (FEEDBACK, 2011)

O valor total, convertido em reais, na cotação média de 17 de outubro a 17 de novembro de 2011 (1 libra esterlina custa 2,793 reais), soma 3.194,64 (REUTERS, 2011).

Além do alto valor de compra, deve ainda ser levado em consideração o custo de transporte deste delicado equipamento e o imposto de importação. Ainda neste sentido, é preciso enfatizar que, o levantamento de preço feito adquire uma unidade deste conjunto experimental. Caso a instituição de ensino almeje montar um laboratório com mais bancadas, o valor seria ainda maior.

Assim, em se tratando da questão da facilidade, a aplicação proposta atinge o objetivo estabelecido inicialmente, de ampliar o conhecimento de Física I e incentivar uma maior participação do aluno em sala de aula. Além disso, por se tratar de uma aplicação virtual, não é necessária a presença de um monitor ou técnico de laboratório e não há riscos de quebrar ou danificar equipamentos.

Nos apêndices deste trabalho, são fornecidas algumas sugestões de roteiros de experimento utilizando a aplicação proposta neste trabalho, bem como algumas perguntas que levarão o aluno a desenvolver fórmulas e aplicar conhecimentos de cálculo, física e cálculo numérico.

Por último, algumas sugestões de trabalhos futuros são listados. A primeira sugestão seria dar continuidade aos experimentos da própria Física I e poder acrescentar molas e polias, criando um sistema dinâmico. Outra sugestão é partir diretamente para as outras áreas da física, como Física II e III, por exemplo, e criar aplicações que simulem experimentos de hidrostática, hidrodinâmica, circuitos elétricos, entre outros. Por último, pode ser criada uma grande aplicação, onde cada instituição de ensino adquire aqueles experimentos que forem convenientes. Esta última opção pode ser uma boa estratégia para a área comercial, onde alguns experimentos são disponibilizados gratuitamente, para que a instituição de ensino conheça a aplicação e, posteriormente, faça opções de compras individuais de novos experimentos.

REFERÊNCIAS

ALONSO, M.; FINN, E. J. **Física - Um Curso Universitário**. 2ª. ed. São Paulo: Editora Edgard Blücher, v. 1, 2001.

BARROS, A. D. et al. Engajamento interativo no curso de Física I da UFJF. **Revista Brasileira de Ensino de Física**, São Paulo, Volume 26, 2004.

BARROS, J. A. D. et al. Dep. de Física UFJF. **Departamento de Física**, 2001. Disponivel em: <http://www.fisica.ufjf.br/pesquisa/ensino.html>. Acesso em: 22 setembro 2011.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **UML - Guia do Usuário**. Rio de Janeiro: Editora Campos, 2000.

CHAVES, A. **Física**. Rio de Janeiro: Reichmann & Affonso Editores, v. 1, 2001.

FEEDBACK. Feedback Group. **Education Equipament, Time and Attendance and Access Control products**, 2011. Disponivel em: <http://vendorad.ecom2.graphicalliance.co.uk/Vendors/bin/623ce1edac40661eb81d763055d75e7a/media/pdf/original_filenames/PASCO%20price%20list%20vers.11C.pdf>. Acesso em: 13 de novembro de 2011.

FERNANDES, K. C. **Construção de um Radiotelescópio Amador em Microondas 12 GHz, Dotado de um Sistema Automático de Aquisição de Dados**. Universidade Católica de Brasília. Brasília. 2007.

FREEDMAN, R. A.; YOUNG, H. D. **Física I**. 10ª. ed. São Paulo: Pearson, v. 1, 2003.

KELLER, F. J.; GETTYS, W. E.; SKOVE, M. J. **Física**. 1ª. ed. São Paulo: Makron Books, v. 1, 1997.

MACHADO, V. **Problemas Geradores de Discussões: Uma Metodologia Para o Ensino de Física em Engenharia**. Universidade Tecnológica Federal Do Paraná - Campus Ponta Grossa. Ponta Grossa. 2009.

MORAES, M. B. D. S. A.; RIBEIRO-TEIXEIRA, R. M. **Circuitos Elétricos: Novas e Velhas Tecnologias Como Facilitadoras de uma Aprendizagem Significativa no Nível Médio**. Universidade Federal do Rio Grande do Sul. Porto Alegre. 2006.

MORAN, J. M.; MASETTO, M. T.; BEHRENS, M. A. **Novas Tecnologias e Mediação Pedagógica**. 13ª. ed. São Paulo: Papirus, 2007.

NETBEANS. **NetBeans**, 2011. Disponivel em: <http://www.netbeans.com/index.html>. Acesso em: 4 de agosto de 2011.

ORACLE. Getting Started with an Integrated Development Environment (IDE). **Sun Developer Network**, 2011. Disponivel em: <http://java.sun.com/developer/technicalArticles/tools/intro.html>. Acesso em: 4 de novembro de 2011.

ORACLE TECHNOLOGY NETWORK. Java Platform, Standard Edition. **Oracle Technology Network for Java Developers**, 2011. Disponivel em: <http://www.oracle.com/technetwork/java/index.html>. Acesso em: 18 de outubro de 2011.

PASCO SCIENTIFICS. **Pasco**, 2011. Disponivel em: <http://www.pasco.com/home.cfm>.

Acesso em: 27 de outubro de 2011.

PEDROSA, D. P. F. Métodos Computacionais em Engenharia. **Departamento de Engenharia de Computação e Automação**, 2005. Disponivel em: <http://www.dca.ufrn.br/~diogo/FTP/dca0304/ajustedecurvas.pdf>. Acesso em: 19 de outubro de 2011.

PERRY, G. T. et al. Necessidades específicas do design de jogos educacionais. **Sociedade Brasileira de Computação - SBGames**, Porto Alegre, 2007.

PRESSMAN, R. S. **Engenharia de Software**. 6ª. ed. São Paulo: McGraw-Hill, 2006.

REUTERS. **Currency Quote**, 2011. Disponivel em: <http://www.reuters.com/finance/currencies/quote?srcAmt=1.00&srcCurr=GBP&destAmt=&destCurr=BRL&historicalDate=>. Acesso em: 17 de novembro de 2011.

SERWAY, R. A.; JEWETT, J. W. **Princípios de Física**. 1ª. ed. São Paulo: Thomson, v. 1, 2004.

SILVA, M. **Educação Online**. 2ª. ed. São Paulo: Edições Loyola, 2006.

SOMMERVILLE, I. **Engenharia de Software**. 6ª. ed. São Paulo: Pearson Education, 2005.

SPERANDIO, D.; MENDES, J. T.; SILVA, L. H. M. **Cálculo Numérico - Características Matemáticas e Computacioais dos Métodos Numéricos**. 4ª. ed. São Paulo: Pearson, 2003.

TIPLER, P. A.; MOSCA, G. **Física Para Cientistas e Engenheiros**. 6ª. ed. Rio de Janeiro: LTC, v. 1, 2009.

APÊNDICE

1 Sugestões de perguntas e roteiros de experimentos

Nesta seção, são apresentadas, em forma de listas, algumas sugestões de experimentos e perguntas que professores podem desenvolver com os alunos, num ambiente de sala de aula ou de laboratório.

Roteiro 1: Ajuste as configurações do experimento para um percurso sem inclinação, e com atrito. Inicie uma nova simulação. Em posse do gráfico de posição versus tempo, responda as seguintes questões:

1) O movimento feito pelo carro é progressivo ou retrógrado? Explique.
2) Qual é a velocidade média do carro, a contar do tempo 0 até metade do percurso?
3) Qual é a aceleração média do carro no mesmo período de tempo?
4) De acordo com a velocidade média, obtida na questão 2, quanto tempo o carro levaria para percorrer 10 metros?
5) Observe o gráfico da posição versus tempo. Escolha dois pontos convenientes (de fácil leitura) e, a partir deles, determine a equação desta reta. O que acontece quando você deriva esta equação?

Roteiro 2: Ajuste as configurações do experimento para o atrito do tipo “Simulador”, distância de 120 centímetros, velocidade inicial nula, gravidade de 10 m/s^2, frequência de amostragem de 10 Hz e altura de 5 centímetros.

1) Faça um diagrama ilustrativo das forças atuantes no carro.
2) A partir do diagrama de forças, aplique o princípio fundamental da dinâmica e encontre a equação que expressa a aceleração do carro.
3) Uma vez que o atrito escolhido foi “Simulador”, que representa um valor praticamente nulo, reescreva a equação que expressa a aceleração do carro, desprezando o atrito.
4) Observe o gráfico da velocidade versus tempo. Escolha dois pontos convenientes (de fácil leitura) e, a partir deles, determine a equação desta reta. O que o coeficiente angular desta reta indica?

5) Preencha a primeira linha da Tabela 8.
6) Repita o experimento, para as alturas 10, 15 e 20 centímetros, refazendo, para cada repetição, os itens 4 e 5 mencionados anteriormente.

Tabela 8 – Tabela do roteiro sugerido do experimento de plano inclinado

Medida	Altura	Ângulo θ	a (m/s^2)	g (m/s^2)
1	5 cm			
2	10 cm			
3	15 cm			
4	20 cm			

Fonte: Autor

7) Calcule a média do módulo da aceleração da gravidade.
8) Qual é a diferença, em percentagem, entre o valor da sua medida para a gravidade e da medida indicada no aplicativo? O que justifica esta diferença?
9) Como você relaciona a altura com o valor da gravidade?

Pergunta complementar, para ambos os experimentos:

Faria diferença realizar este experimento no 10º andar de um prédio e no 3º subsolo do mesmo? A diferença da aceleração gravitacional alteraria os resultados? Considere que cada andar do prédio possua 3 metros de altura.

2 Códigos da aplicação

É apresentada, nesta seção, a íntegra dos códigos desenvolvidos pelo autor da aplicação proposta neste trabalho.

Na classe CinematicaView.java:

```
36    Logica objLogica = new Logica();
37    Percurso objPercurso = new Percurso();
38
39    DefaultTableModel model = new DefaultTableModel (new Object[] {"Tempo",
"Posição"}, 0);
40    XYSeries serie1 = new XYSeries("Posições");
41    XYSeries serie2 = new XYSeries("Velocidades");
42    XYSeries serie3 = new XYSeries("Acelerações");
43    XYSeriesCollection data = new XYSeriesCollection();
```

```
44    XYSeriesCollection data2 = new XYSeriesCollection();
45    XYSeriesCollection data3 = new XYSeriesCollection();
```

Entre as linhas 46 a 652, está um código gerado automaticamente pelo NetBeans. O código é encontrado na seção anexos, deste trabalho.

```
648 private void sliderAlturaMouseReleased(java.awt.event.MouseEvent evt) {
649
650     objLogica.altura = (double) sliderAltura.getValue();
651
652     //Faz o label do slider corresponder ao valor apontado
653     labelDinamicoAltura.setText( String.valueOf(objLogica.altura) );
654     //Divide por 100 para transformar em centimetro
655     objLogica.altura = objLogica.altura/100.0;
656
657     System.out.println("Altura: " + objLogica.altura);
658
659 }
660
661 private void sliderDistanciaMouseReleased(java.awt.event.MouseEvent evt) {
662
663     objLogica.distancia = (double) sliderDistancia.getValue();
664
665     //Faz o label do slider corresponder ao valor apontado
666     labelDinamicoDistancia.setText( String.valueOf(objLogica.distancia) );
667     //Divide por 100 para transformar em centimetro
668     objLogica.distancia = objLogica.distancia/100.0;
669
670     System.out.println("Distancia: " + objLogica.distancia);
671 }
672
673 private void sliderVelocidadeInicialMouseReleased(java.awt.event.MouseEvent evt){
674
675     objLogica.velocidadeInicial = (double) sliderVelocidadeInicial.getValue() / 10.0;
676
677     //Faz o label do slider corresponder ao valor apontado
678     labelDinamicoVelocidadeInicial.setText(String.valueOf(objLogica.velocidadeInicial));
679
680     System.out.println("Vel Inicial: " + objLogica.velocidadeInicial);
681 }
682
683 private void botaoIniciarSimulacaoActionPerformed(java.awt.event.ActionEvent evt) {
684
685         objLogica.tempo = 0;
686
687         if (objLogica.altura==0 && objLogica.velocidadeInicial < 0.3) {
688             JOptionPane.showMessageDialog ( null, "Como o percurso é horizontal "
689                     + "(altura = 0), uma velocidade inicial mínima de 0.3 m/s é"
690                     + "exigida.", "Erro!", JOptionPane.ERROR_MESSAGE);
691         } else {
692
```

```
693     JFreeChart grafico = ChartFactory.createScatterPlot("Gráfico 1 - "
694                             + "Posição versus Tempo", //Título
695                 "Tempo",
696                 "Posição",
697                 data, //dados do XYDataset
698                 PlotOrientation.VERTICAL,
699                 false,                  //Legenda
700                 true,                   //Tooltips
701                 false                    //URL
702                 );
703
704     JFreeChart grafico2 = ChartFactory.createScatterPlot("Gráfico 2 - "
705                             + "Velocidade versus Tempo", //Título
706                 "Tempo",
707                 "Velocidade",
708                 data2, //dados do XYDataset
709                 PlotOrientation.VERTICAL,
710                 false,                  //Legenda
711                 true,                   //Tooltips
712                 false                    //URL
713                 );
714
715     JFreeChart grafico3 = ChartFactory.createScatterPlot("Gráfico 3 - "
716                             + "Aceleração versus Tempo", //Título
717                 "Tempo",
718                 "Aceleração",
719                 data3, //dados do XYDataset
720                 PlotOrientation.VERTICAL,
721                 false,                  //Legenda
722                 true,                   //Tooltips
723                 false                   //URL
724                 );
725
726     Simulador objSimulador = new Simulador(model, serie1, serie2, serie3,
727           data, data2, data3);
728     objSimulador.config = objLogica;
729
730     java.util.Timer timer = new java.util.Timer();
731     timer.schedule(objSimulador, 500,2000/(int)objLogica.freqAmostragem);
732     // Parâmetros: tarefa, início (em ms), repetição (em ms)
733
734     ChartPanel frame = new ChartPanel(grafico);
735     abaGraficoPosicao.removeAll();
736     abaGraficoPosicao.setLayout(new GridLayout(1,1));
737     abaGraficoPosicao.add(frame);
738
739     ChartPanel frame2 = new ChartPanel(grafico2);
740     abaGraficoVelocidade.removeAll();
741     abaGraficoVelocidade.setLayout(new GridLayout(1,1));
742     abaGraficoVelocidade.add(frame2);
```

```
743         if (objLogica.altura == 0){
744             XYPlot plot = grafico2.getXYPlot();
745             plot.getRangeAxis().setRange(-4.0,4.0);
746         }
747 
748         ChartPanel frame3 = new ChartPanel(grafico3);
749         abaGraficoAceleracao.removeAll();
750         abaGraficoAceleracao.setLayout(new GridLayout(1,1));
751         abaGraficoAceleracao.add(frame3);
752 
753         abaMovimento.removeAll();
754         abaMovimento.setLayout(new GridLayout(1,1));
755         abaMovimento.add(new JLabel ( objPercurso.criarImagem() ));
756 
757       }
758 
759 }
760 
761 private void comboGravidadeActionPerformed(java.awt.event.ActionEvent evt) {
762 
763         switch (comboGravidade.getSelectedIndex()){
764             case 0:
765                 //Gravidade: Terra (10.0)
766                 objLogica.gravidade = 10.0;
767                 break;
768             case 1:
769                 //Gravidade: Terra (9.81)
770                 objLogica.gravidade = 9.81;
771                 break;
772             case 2:
773                 //Gravidade: Lua (1/6 * Terra)
774                 objLogica.gravidade = 9.81 / 6;
775                 break;
776             case 3:
777                 //Gravidade: Marte (0.38 * Terra)
778                 objLogica.gravidade = 9.81 * 0.38;
779                 break;
780         }
781 
782         System.out.println("Gravidade: " + objLogica.gravidade);
783 }
784 
785 private void comboCoefAtritoActionPerformed(java.awt.event.ActionEvent evt) {
786 
787         switch (comboCoefAtrito.getSelectedIndex()){
788             case 0:
789                 //Coef Atrito: Zero
790                 objLogica.coeficienteAtrito = 0.0;
791                 break;
792             case 1:
```

```
793             //Coef Atrito: Aço - Aço (0.58)
794             objLogica.coeficienteAtrito = 0.58;
795             break;
796         case 2:
797             //Coef Atrito: Alum. - Aço (0.47)
798             objLogica.coeficienteAtrito = 0.47;
799             break;
800         case 3:
801             //Coef Atrito: Vidro - Vidro (0.4)
802             objLogica.coeficienteAtrito = 0.4;
803             break;
804         case 4:
805             //Coef Atrito: Latao - Aço (0.42)
806             objLogica.coeficienteAtrito = 0.42;
807             break;
808     }
809
810     System.out.println("Coef Atrito: " + objLogica.coeficienteAtrito);
811 }
812
813 private void comboFreqAmostragemActionPerformed(java.awt.event.ActionEvent evt) {
814
815     switch (comboFreqAmostragem.getSelectedIndex()){
816         case 0:
817             //Freq Amostragem: 10 Hz
818             objLogica.freqAmostragem = 10.0;
819             break;
820         case 1:
821             //Freq Amostragem: 25 Hz
822             objLogica.freqAmostragem = 25.0;
823             break;
824         case 2:
825             //Freq Amostragem: 50 Hz
826             objLogica.freqAmostragem = 50.0;
827             break;
828         case 3:
829             //Freq Amostragem: 100 Hz
830             objLogica.freqAmostragem = 100.0;
831             break;
832     }
833
834     System.out.println("Freq Amostragem: " + objLogica.freqAmostragem);
835 }
836
837   private void botaoMostrarEquacoesActionPerformed(java.awt.event.ActionEvent evt){
838
839     labelDinamicoEqPosicao.setText( "S(t) = " +
840             String.valueOf(new DecimalFormat ("0.00").format
841             (objLogica.posicaoInicial)) + " + "+
842             String.valueOf(new DecimalFormat ("0.00").format
```

```
843             (objLogica.velocidadeInicial)) + " t + "+
844             String.valueOf(new DecimalFormat ("0.00").format
845             (objLogica.aceleracao()/2)) + " t\u00B2"
846             );
847       labelDinamicoEqVelocidade.setText( "V(t) = " +
848             String.valueOf(new DecimalFormat ("0.00").format
849             (objLogica.velocidadeInicial)) + " + " +
850             String.valueOf(new DecimalFormat ("0.00").format
851             (objLogica.aceleracao())) + " t"
852             );
853       labelDinamicoEqAceleracao.setText( "A(t) = " +
854             String.valueOf(new DecimalFormat ("0.00").format
855             (objLogica.aceleracao())));
856
857    }
```

Na classe Logica.java:

```
1  package cinematica;
2
3  public class Logica {
4        public double altura = 0.05;
5        public double distancia = 1.2;
6        public double posicaoInicial = 0.0;
7        public double velocidadeInicial = 0.4;
8        public double gravidade = 10.0;
9        public double coeficienteAtrito = 0.002;
10
11       public double tempo;
12       public double freqAmostragem = 10.0;
13       public double maximoIteracoes;
14
15       public double posicao() {
16          //Equação do espaço:
17          //S(t) = So + Vo t + (1/2) g (sen a - u cos a) t^2
18
19          double espaco = posicaoInicial + velocidadeInicial * tempoFatiado()
20                      + 0.5 * gravidade * (seno() - coeficienteAtrito *
21                      cosseno()) * Math.pow(tempoFatiado(), 2);
22
23          return espaco;
24       }
25
26       public double velocidade() {
27          //Equação da velocidade:
28          //V(t) = Vo + g (sen a - u cos a) t
29
30          double velocidade = velocidadeInicial + gravidade * (seno() –
31                      coeficienteAtrito * cosseno()) * tempoFatiado();
32
33          return velocidade;
```

```
34        }
35
36        public double aceleracao() {
37           //Equação da aceleração:
38           //A(t) = g (sen a - u cos a)
39
40           double aceleracao = gravidade * (seno() - coeficienteAtrito *
41                 cosseno());
42
43           return aceleracao;
44        }
45
46        public double tempoFatiado() {
47              return tempo / freqAmostragem;
48        }
49
50        private double hipotenusa() {
51              return Math.sqrt(Math.pow(altura, 2) + Math.pow(distancia,2));
52        }
53
54        private double seno() {
55              return altura/hipotenusa();
56        }
57
58        private double cosseno() {
59              return distancia/hipotenusa();
60        }
61
62        public boolean maximoIteracoes() {
63              if (posicao() <= hipotenusa())
64                 return true;
65              else
66                 return false;
67        }
68
69 }
```

Na classe Ponto.java:

```
1 package cinematica;
2
3 //Cria um objeto ponto, que armazenará cada tempo
4 //e sua respectiva posição
5
6 public class Ponto {
7    public double x;   //Aqui ele receberá o "tempoFatiado"
8    public double y;   //Aqui ele receberá a "posicaoAtual"
9    public double v;   //Aqui ele receberá a "velocidade"
10    public double a;   //Aqui ele receberá a "aceleração"
11 }
```

Na classe Simulador.java:

```
1 package cinematica;
2
3 import java.util.ArrayList;
4 import java.util.TimerTask;
5 import javax.swing.JOptionPane;
6 import javax.swing.table.DefaultTableModel;
7 import org.jfree.chart.JFreeChart;
8 import org.jfree.data.xy.XYSeries;
9 import org.jfree.data.xy.XYSeriesCollection;
10
11 public class Simulador extends TimerTask {
12
13     private DefaultTableModel model;
14
15     private XYSeries serie1;
16     private XYSeries serie2;
17     private XYSeries serie3;
18     private XYSeriesCollection data;
19     private XYSeriesCollection data2;
20     private XYSeriesCollection data3;
21     private JFreeChart grafico;
22
23     public Logica config;
24     private int iteracaoAtual = 0;
25     private int codErro = 0;
26     private int i = 0;
27     public ArrayList<Ponto> listaPosicoes = new ArrayList<Ponto>();
28
29     public Simulador(DefaultTableModel model, XYSeries serie1, XYSeries serie2,
30             XYSeries serie3, XYSeriesCollection data, XYSeriesCollection data2,
31             XYSeriesCollection data3) {
32         this.model = model;
33         this.serie1 = serie1;
34         this.serie2 = serie2;
35         this.serie3 = serie3;
36         this.data = data;
37         this.data2 = data2;
38         this.data3 = data3;
39     }
40
41     public void run() {
42         iteracaoAtual++;
43
44         //Instanciando um novo objeto Ponto
45         Ponto p = new Ponto();
46
47         //Atribuindo os valores do Ponto
48         p.x = config.tempoFatiado();
49         p.y = config.posicao();
```

```
50          p.v = config.velocidade();
51          p.a = config.aceleracao();
52
53          listaPosicoes.add(p);
54
55
56          if (i >= 1 && listaPosicoes.get(i).y <
57                  listaPosicoes.get(i - 1).y == true){
58
59              //Tratamento para caso o carro tenha parado antes de chegar ao final
60              //do percurso
61
62              p.x = config.tempoFatiado();
63              p.y = listaPosicoes.get(i - 1).y;
64              p.v = 0.0;
65              p.a = listaPosicoes.get(i - 1).a;
66              //listaPosicoes.set(i, listaPosicoes.get(i - 1));
67
68              //mensagem de erro pois o carro está voltando
69              JOptionPane.showMessageDialog ( null, "Carro parou antes de chegar "
70                      + "ao final da pista!");
71
72              codErro = 1;
73          }
74
75          //Adiciona o Ponto atual à tabela (nova linha)
76          model.addRow(new Object[] {p.x, p.y});
77
78          //Adiciona o Ponto atual aos gráficos
79          serie1.add(p.x,p.y);
80          serie2.add(p.x,p.v);
81          serie3.add(p.x,p.a);
82
83          config.tempo++;
84
85          i++;
86
87          //Parar a iteração.
88          if (config.maximoIteracoes() == false || codErro != 0) {
89              data.addSeries(serie1);
90              data2.addSeries(serie2);
91              data3.addSeries(serie3);
92              cancel();
93          }
94
95      }
96  }
```

3 Tabelas comparativas

Tabela 9 – Terceiro comparativo de dados obtidos no programa Data Studio (Pasco) e na aplicação proposta neste trabalho e a diferença percentual entre os dados, experimento de plano horizontal

Data Studio		Programa Proposto		Diferença
Tempo	Posição	Tempo	Posição	D (%)
0,4006	0,1705	0,4	0,1749	2,58
0,5007	0,2207	0,5	0,2215	0,36
0,6007	0,268	0,6	0,2681	0,04
0,7008	0,3156	0,7	0,3147	0,29
0,8009	0,3629	0,8	0,3613	0,44
0,9009	0,4095	0,9	0,4079	0,39
1,001	0,4568	1	0,4545	0,50
1,1011	0,5036	1,1	0,5011	0,50
1,2012	0,5504	1,2	0,5477	0,49
1,3012	0,5975	1,3	0,5943	0,54
1,4013	0,644	1,4	0,6409	0,48
1,5014	0,6901	1,5	0,6875	0,38
1,6015	0,7367	1,6	0,7341	0,35
1,7015	0,7826	1,7	0,7807	0,24
1,8016	0,8284	1,8	0,8273	0,13
1,9017	0,8744	1,9	0,8739	0,06
2,0017	0,9199	2	0,9205	0,07
2,1018	0,9651	2,1	0,9671	0,21
			Média:	0,45

Fonte: Autor

Tabela 10 – Quarto comparativo de dados obtidos no programa Data Studio (Pasco) e na aplicação proposta neste trabalho e a diferença percentual entre os dados, experimento de plano horizontal

Data Studio		Programa Proposto		Diferença
Tempo	Posição	Tempo	Posição	D (%)
0,3199	0,1995	0,32	0,20112	0,81
0,3599	0,2346	0,36	0,23576	0,49
0,3999	0,2695	0,4	0,2704	0,33
0,4399	0,3044	0,44	0,30504	0,21
0,4799	0,3392	0,48	0,33968	0,14
0,5199	0,3739	0,52	0,37432	0,11
0,5599	0,4087	0,56	0,40896	0,06
0,5999	0,4434	0,6	0,4436	0,05
0,64	0,4788	0,64	0,47824	0,12
0,68	0,5136	0,68	0,51288	0,14
0,72	0,5485	0,72	0,54752	0,18
0,76	0,5833	0,76	0,58216	0,20
0,8	0,6178	0,8	0,6168	0,16

0,84	0,6524	0,84	0,65144	0,15
0,88	0,687	0,88	0,68608	0,13
0,92	0,7215	0,92	0,72072	0,11
0,96	0,7561	0,96	0,75536	0,10
1	0,7905	1	0,79	0,06
1,04	0,8249	1,04	0,82464	0,03
1,08	0,8593	1,08	0,85928	0,00
1,12	0,8935	1,12	0,89392	0,05
1,16	0,9278	1,16	0,92856	0,08
1,2	0,962	1,2	0,9632	0,12
1,24	0,9966	1,24	0,99784	0,12
1,2799	1,0253	1,28	1,032480	0,70
			Média:	0,19

Fonte: Autor

Tabela 11 – Quinto comparativo de dados obtidos no programa Data Studio (Pasco) e na aplicação proposta neste trabalho e a diferença percentual entre os dados, experimento de plano horizontal

Data Studio		Programa Proposto		Diferença
Tempo	Posição	Tempo	Posição	D (%)
0,3381	0,1949	0,34	0,19752	1,34
0,358	0,2097	0,36	0,21238	1,28
0,3779	0,225	0,38	0,22724	1,00
0,3978	0,2423	0,4	0,2421	0,08
0,4176	0,2542	0,42	0,25696	1,09
0,4375	0,2695	0,44	0,27182	0,86
0,4574	0,2845	0,46	0,28668	0,77
0,4773	0,2991	0,48	0,30154	0,82
0,4972	0,3142	0,5	0,3164	0,70
0,5171	0,3287	0,52	0,33126	0,78
0,537	0,3437	0,54	0,34612	0,70
0,5569	0,3586	0,56	0,36098	0,66
0,5767	0,3732	0,58	0,37584	0,71
0,5966	0,3875	0,6	0,3907	0,83
0,6165	0,4027	0,62	0,40556	0,71
0,6364	0,4176	0,64	0,42042	0,68
0,6563	0,4326	0,66	0,43528	0,62
0,6762	0,4472	0,68	0,45014	0,66
0,6961	0,4622	0,7	0,465	0,61
0,7159	0,4773	0,72	0,47986	0,54
0,7358	0,4919	0,74	0,49472	0,57
0,7557	0,5069	0,76	0,50958	0,53
0,7756	0,5218	0,78	0,52444	0,51
0,7955	0,5365	0,8	0,5393	0,52
0,8154	0,5514	0,82	0,55416	0,50
0,8353	0,5659	0,84	0,56902	0,55

0,8551	0,5808	0,86	0,58388	0,53
0,875	0,5958	0,88	0,59874	0,49
0,8949	0,6103	0,9	0,6136	0,54
0,9148	0,6252	0,92	0,62846	0,52
0,9347	0,6402	0,94	0,64332	0,49
0,9546	0,6546	0,96	0,65818	0,55
0,9745	0,6696	0,98	0,67304	0,51
0,9943	0,684	1	0,6879	0,57
1,0142	0,6988	1,02	0,70276	0,57
1,0341	0,7138	1,04	0,71762	0,54
1,054	0,7282	1,06	0,73248	0,59
1,0739	0,743	1,08	0,74734	0,58
1,0938	0,758	1,1	0,7622	0,55
1,1137	0,7723	1,12	0,77706	0,62
1,1335	0,7872	1,14	0,79192	0,60
1,1534	0,8015	1,16	0,80678	0,66
1,1733	0,8163	1,18	0,82164	0,65
1,1932	0,8313	1,2	0,8365	0,63
1,2131	0,8456	1,22	0,85136	0,68
1,233	0,8603	1,24	0,86622	0,69
1,2529	0,8751	1,26	0,88108	0,68
1,2727	0,8896	1,28	0,89594	0,71
1,2926	0,9044	1,3	0,9108	0,71
1,3125	0,9187	1,32	0,92566	0,76
1,3324	0,9334	1,34	0,94052	0,76
1,3523	0,9481	1,36	0,95538	0,77
1,3722	0,9625	1,38	0,97024	0,80
1,392	0,9771	1,4	0,9851	0,82
1,4119	0,9919	1,42	0,99996	0,81
1,4318	1,0062	1,44	1,01482	0,86
			Média:	0,68

Fonte: Autor

Tabela 12 – Sexto comparativo de dados obtidos no programa Data Studio (Pasco) e na aplicação proposta neste trabalho e a diferença percentual entre os dados, experimento de plano horizontal

Data Studio		Programa Proposto		Diferença
Tempo	Posição	Tempo	Posição	D (%)
0,3382	0,2164	0,34	0,21902	1,21
0,3581	0,2312	0,36	0,23128	0,03
0,378	0,2413	0,38	0,24354	0,93
0,3978	0,2532	0,4	0,2558	1,03
0,4177	0,2657	0,42	0,26806	0,89
0,4376	0,2783	0,44	0,28032	0,73
0,4575	0,2903	0,46	0,29258	0,79
0,4773	0,3029	0,48	0,30484	0,64

0,4972	0,3149	0,5	0,3171	0,70
0,5171	0,3273	0,52	0,32936	0,63
0,5369	0,3397	0,54	0,34162	0,57
0,5568	0,3516	0,56	0,35388	0,65
0,5767	0,364	0,58	0,36614	0,59
0,5966	0,3765	0,6	0,3784	0,50
0,6164	0,3884	0,62	0,39066	0,58
0,6363	0,4008	0,64	0,40292	0,53
0,6562	0,4128	0,66	0,41518	0,58
0,676	0,4252	0,68	0,42744	0,53
0,6959	0,4377	0,7	0,4397	0,46
0,7158	0,4498	0,72	0,45196	0,48
0,7357	0,4622	0,74	0,46422	0,44
0,7555	0,4745	0,76	0,47648	0,42
0,7754	0,4862	0,78	0,48874	0,52
0,7953	0,499	0,8	0,501	0,40
0,8151	0,5108	0,82	0,51326	0,48
0,835	0,5232	0,84	0,52552	0,44
0,8549	0,5356	0,86	0,53778	0,41
0,8748	0,5476	0,88	0,55004	0,45
0,8946	0,56	0,9	0,5623	0,41
0,9145	0,5724	0,92	0,57456	0,38
0,9344	0,5843	0,94	0,58682	0,43
0,9542	0,5967	0,96	0,59908	0,40
0,9741	0,6087	0,98	0,61134	0,43
0,994	0,6211	1	0,6236	0,40
1,0139	0,6333	1,02	0,63586	0,40
1,0337	0,6453	1,04	0,64812	0,44
1,0536	0,6576	1,06	0,66038	0,42
1,0735	0,6698	1,08	0,67264	0,42
1,0933	0,6816	1,1	0,6849	0,48
1,1132	0,694	1,12	0,69716	0,46
1,1331	0,7059	1,14	0,70942	0,50
1,1529	0,7181	1,16	0,72168	0,50
1,1728	0,7305	1,18	0,73394	0,47
1,1927	0,7424	1,2	0,7462	0,51
1,2126	0,7546	1,22	0,75846	0,51
1,2324	0,7668	1,24	0,77072	0,51
1,2523	0,7786	1,26	0,78298	0,56
1,2722	0,7909	1,28	0,79524	0,55
1,292	0,8027	1,3	0,8075	0,60
1,3119	0,8149	1,32	0,81976	0,60
1,3318	0,8271	1,34	0,83202	0,59
1,3516	0,8388	1,36	0,84428	0,65
1,3715	0,8511	1,38	0,85654	0,64
1,3914	0,8633	1,4	0,8688	0,64

1,4113	0,875	1,42	0,88106	0,69
1,4311	0,8872	1,44	0,89332	0,69
1,451	0,8989	1,46	0,90558	0,74
1,4709	0,9111	1,48	0,91784	0,74
1,4907	0,9233	1,5	0,9301	0,74
1,5106	0,935	1,52	0,94236	0,79
1,5305	0,9472	1,54	0,95462	0,78
1,5503	0,9592	1,56	0,96688	0,80
1,5702	0,9709	1,58	0,97914	0,85
1,5901	0,9832	1,6	0,9914	0,83
1,61	0,9947	1,62	1,0037	0,90
1,6298	1,0069	1,64	1,0159	0,90
			Média:	0,59

Fonte: Autor

ANEXOS

É apresentada, nesta seção, a íntegra dos códigos gerados pelo NetBeans no desenvolvimento da aplicação proposta neste trabalho.

Na classe CinematicaAboutBox.java:

```
 1 /*
 2  * CinematicaAboutBox.java
 3  */
 4
 5 package cinematica;
 6
 7 import org.jdesktop.application.Action;
 8
 9 public class CinematicaAboutBox extends javax.swing.JDialog {
10
11     public CinematicaAboutBox(java.awt.Frame parent) {
12         super(parent);
13         initComponents();
14         getRootPane().setDefaultButton(closeButton);
15     }
16
17     @Action public void closeAboutBox() {
18         dispose();
19     }
20
21     /** This method is called from within the constructor to
22      * initialize the form.
23      * WARNING: Do NOT modify this code. The content of this method is
24      * always regenerated by the Form Editor.
25      */
26     Generated Code
132
133     // Variables declaration - do not modify
134     private javax.swing.JButton closeButton;
135     // End of variables declaration
136
137 }
```

Na classe CinematicaApp:

```
1 /*
2  * CinematicaApp.java
3  */
4
5 package cinematica;
6
```

```
7 import org.jdesktop.application.Application;
8 import org.jdesktop.application.SingleFrameApplication;
9
10 /**
11  * The main class of the application.
12  */
13 public class CinematicaApp extends SingleFrameApplication {
14
15     /**
16      * At startup create and show the main frame of the application.
17      */
18     @Override protected void startup() {
19         show(new CinematicaView(this));
20     }
21
22     /**
23      * This method is to initialize the specified window by injecting resources.
24      * Windows shown in our application come fully initialized from the GUI
25      * builder, so this additional configuration is not needed.
26      */
27     @Override protected void configureWindow(java.awt.Window root) {
28     }
29
30     /**
31      * A convenient static getter for the application instance.
32      * @return the instance of CinematicaApp
33      */
34     public static CinematicaApp getApplication() {
35         return Application.getInstance(CinematicaApp.class);
36     }
37
38     /**
39      * Main method launching the application.
40      */
41     public static void main(String[] args) {
42         launch(CinematicaApp.class, args);
43     }
44 }
```

Na classe CinematicaView.java:

```
1 /*
2  * CinematicaView.java
3  */
4
5 package cinematica;
6
7 import java.awt.GridLayout;
8 import org.jdesktop.application.Action;
9 import org.jdesktop.application.ResourceMap;
10 import org.jdesktop.application.SingleFrameApplication;
```

```
11 import org.jdesktop.application.FrameView;
12 import org.jdesktop.application.TaskMonitor;
13 import java.awt.event.ActionEvent;
14 import java.awt.event.ActionListener;
15 import java.text.DecimalFormat;
16 import javax.swing.*;
17 import javax.swing.table.DefaultTableModel;
18 import org.jfree.chart.ChartFactory;
19 import org.jfree.chart.ChartPanel;
20 import org.jfree.chart.JFreeChart;
21 import org.jfree.chart.plot.PlotOrientation;
22 import org.jfree.chart.plot.XYPlot;
23 import org.jfree.data.xy.XYSeries;
24 import org.jfree.data.xy.XYSeriesCollection;
25
26
27 /**
28  * The application's main frame.
29  */
30 public class CinematicaView extends FrameView {
31
32     // Códigos do autor – Conforme Apêndices
47     public CinematicaView(SingleFrameApplication app) {
48
49         super(app);
50
51         initComponents();
52
53         // status bar initialization - message timeout, idle icon and busy animation, etc
54         ResourceMap resourceMap = getResourceMap();
55         int messageTimeout = resourceMap.getInteger("StatusBar.messageTimeout");
56         messageTimer = new Timer(messageTimeout, new ActionListener() {
57             public void actionPerformed(ActionEvent e) {
58                 statusMessageLabel.setText("");
59             }
60         });
61         messageTimer.setRepeats(false);
62         int busyAnimationRate = resourceMap.getInteger("StatusBar.busyAnimationRate");
63         for (int i = 0; i < busyIcons.length; i++) {
64             busyIcons[i] = resourceMap.getIcon("StatusBar.busyIcons[" + i + "]"); 65         }
66         busyIconTimer = new Timer(busyAnimationRate, new ActionListener() {
67             public void actionPerformed(ActionEvent e) {
68                 busyIconIndex = (busyIconIndex + 1) % busyIcons.length;
69                 statusAnimationLabel.setIcon(busyIcons[busyIconIndex]);
70             }
71         });
72         idleIcon = resourceMap.getIcon("StatusBar.idleIcon");
73         statusAnimationLabel.setIcon(idleIcon);
74         progressBar.setVisible(false);
75
```

```
76      // connecting action tasks to status bar via TaskMonitor
77      TaskMonitor taskMonitor = new TaskMonitor(getApplication().getContext());
78      taskMonitor.addPropertyChangeListener(new java.beans.PropertyChangeListener() {
79          public void propertyChange(java.beans.PropertyChangeEvent evt) {
80              String propertyName = evt.getPropertyName();
81              if ("started".equals(propertyName)) {
82                  if (!busyIconTimer.isRunning()) {
83                      statusAnimationLabel.setIcon(busyIcons[0]);
84                      busyIconIndex = 0;
85                      busyIconTimer.start();
86                  }
87                  progressBar.setVisible(true);
88                  progressBar.setIndeterminate(true);
89              } else if ("done".equals(propertyName)) {
90                  busyIconTimer.stop();
91                  statusAnimationLabel.setIcon(idleIcon);
92                  progressBar.setVisible(false);
93                  progressBar.setValue(0);
94              } else if ("message".equals(propertyName)) {
95                  String text = (String)(evt.getNewValue());
96                  statusMessageLabel.setText((text == null) ? "" : text);
97                  messageTimer.restart();
98              } else if ("progress".equals(propertyName)) {
99                  int value = (Integer)(evt.getNewValue());
100                   progressBar.setVisible(true);
101                   progressBar.setIndeterminate(false);
102                   progressBar.setValue(value);
103               }
104           }
105       });
106   }
107
108   @Action
109   public void showAboutBox() {
110       if (aboutBox == null) {
111           JFrame mainFrame = CinematicaApp.getApplication().getMainFrame();
112           aboutBox = new CinematicaAboutBox(mainFrame);
113           aboutBox.setLocationRelativeTo(mainFrame);
114       }
115       CinematicaApp.getApplication().show(aboutBox);
116   }
117
118   /** This method is called from within the constructor to
119    * initialize the form.
120    * WARNING: Do NOT modify this code. The content of this method is
121    * always regenerated by the Form Editor.
122    */
123   @SuppressWarnings("unchecked")
124   Generated Code
647   // Códigos do autor – Conforme Apêndices
```

```
867     // Variables declaration - do not modify
868     private javax.swing.JPanel abaConfiguracoes;
869     private javax.swing.JPanel abaEquacoes;
870     private javax.swing.JPanel abaGraficoAceleracao;
871     private javax.swing.JPanel abaGraficoPosicao;
872     private javax.swing.JPanel abaGraficoVelocidade;
873     private javax.swing.JPanel abaGraficos;
874     private javax.swing.JPanel abaMovimento;
875     private javax.swing.JPanel abaTabela;
876     private javax.swing.JButton botaoIniciarSimulacao;
877     private javax.swing.JButton botaoMostrarEquacoes;
878     private javax.swing.JComboBox comboCoefAtrito;
879     private javax.swing.JComboBox comboFreqAmostragem;
880     private javax.swing.JComboBox comboGravidade;
881     private javax.swing.JScrollPane jScrollPane1;
882     private javax.swing.JTable jTable1;
883     private javax.swing.JLabel labelAltura;
884     private javax.swing.JLabel labelCoefAtrito;
885     private javax.swing.JLabel labelDinamicoAltura;
886     private javax.swing.JLabel labelDinamicoDistancia;
887     private javax.swing.JLabel labelDinamicoEqAceleracao;
888     private javax.swing.JLabel labelDinamicoEqPosicao;
889     private javax.swing.JLabel labelDinamicoEqVelocidade;
890     private javax.swing.JLabel labelDinamicoUnidadeAltura;
891     private javax.swing.JLabel labelDinamicoUnidadeDistancia;
892     private javax.swing.JLabel labelDinamicoUnidadeVelocidade;
893     private javax.swing.JLabel labelDinamicoVelocidadeInicial;
894     private javax.swing.JLabel labelDistancia;
895     private javax.swing.JLabel labelEqAceleracao;
896     private javax.swing.JLabel labelEqPosicao;
897     private javax.swing.JLabel labelEqVelocidade;
898     private javax.swing.JLabel labelFreqAmostragem;
899     private javax.swing.JLabel labelGravidade;
900     private javax.swing.JLabel labelVelocidadeInicial;
901     private javax.swing.JPanel mainPanel;
902     private javax.swing.JMenuBar menuBar;
903     private javax.swing.JProgressBar progressBar;
904     private javax.swing.JSeparator separador;
905     private javax.swing.JSlider sliderAltura;
906     private javax.swing.JSlider sliderDistancia;
907     private javax.swing.JSlider sliderVelocidadeInicial;
908     private javax.swing.JLabel statusAnimationLabel;
909     private javax.swing.JLabel statusMessageLabel;
910     private javax.swing.JPanel statusPanel;
911     private javax.swing.JTabbedPane tabGeral;
912     private javax.swing.JTabbedPane tabGraficos;
913     // End of variables declaration
914
915     private final Timer messageTimer;
916     private final Timer busyIconTimer;
```

```
917     private final Icon idleIcon;
918     private final Icon[] busyIcons = new Icon[15];
919     private int busyIconIndex = 0;
920
921     private JDialog aboutBox;
922 }
```