

Trabalho de Conclusão de Curso

Physimulation - Animação de Fenômenos

Físicos

Rafael Issao Miyagawa
rafael.miyagawa@usp.br

Alberto Hideki Ueda
alberto.ueda@usp.br

Orientadores: José Coelho de Pina Junior
João Pedro Kerr Catunda

Novembro de 2012

The most exciting phrase to hear in science, the one that heralds new discoveries, is not 'Eureka!' but 'That's funny ...'

Isaac Asimov

Conteúdo

I Parte objetiva	4
1 Introdução	4
1.1 Física no BCC	4
1.2 Problemas ao simular interações físicas	5
1.3 Simulador	5
1.4 Estrutura da Monografia	5
2 Plataforma computacional	7
2.1 Esquema de dependências	7
2.2 Ruby	7
2.3 Simulação com Chipmunk	9
2.4 Animação com Gosu	11
2.5 Interfaces com Glade e Visual Ruby	12
3 Discretização da simulação	14
3.1 Fixo	14
3.2 Variável	14
3.3 Tempo de Simulação no Physimulation	15
4 Colisões	16
4.1 <i>Broad phase</i> - Árvore AABB	16
4.2 <i>Broad phase</i> - 1D sweep and prune	18
4.3 <i>Broad phase</i> - Spatial hashing	18
4.4 <i>Narrow phase</i> - Separating Axis Theorem	19
5 Atividades realizadas	22
5.1 Estudo inicial	22
5.2 Ambientes de demonstração	23
5.3 Criador de Cenários	23
6 Animações produzidas	26
6.1 Descrição	27
6.2 Visualização das animações	33
7 Physimulation	34
7.1 A ferramenta	34
7.2 Criando um cenário	36
7.3 Outros exemplos	39

7.4	Arquivos de configuração	43
7.5	Pontos de modificação	45
8	Introdução à computação com animações	46
8.1	Configuração	46
8.2	Angry Bixos	46
8.3	Apolo	49
9	Conclusão	51
10	Instruções de instalação da plataforma	53
11	Apêndice	54
II	Parte subjetiva	56
11.1	Desafios e frustrações encontrados	56
11.2	Disciplinas mais relevantes - Alberto Ueda	57
11.3	Disciplinas mais relevantes - Rafael Issao Miyagawa	58
11.4	Estudos e trabalhos futuros	59
11.5	Desafios de simulação	60

Parte I

Parte objetiva

e-xer-cí-cios-pro-gra-mas

1 Introdução

O foco deste projeto é atrair a atenção dos alunos do IME em relação às disciplinas de física ministradas no curso de bacharelado em Ciência da Computação.

1.1 Física no BCC

Embora seja comum encontrar alunos da computação que tenham interesse em tópicos relacionados a física clássica e moderna - Leis de Newton, Leis de Kepler, Teoria da Relatividade de Einstein, entre outros - há considerável resistência do corpo discente em relação a grade curricular do BCC incluir as disciplinas de física. Isto não se aplica apenas às disciplinas de física, mas também a álgebra, cálculo, probabilidade e estatística.

Visando compreender melhor a situação e apresentar uma proposta de reformulação da grade ou das disciplinas, foi criado o Grupo de Apoio ao BCC, uma iniciativa da Comissão de Coordenação do BCC (CoC). Após as discussões e aplicação de questionários nos Encontros BCC, um dos consensos de ambas as partes - professores e alunos - foi o de que seria interessante uma maior contextualização destas disciplinas para os alunos em termos de computação. No apêndice da seção 11 há uma cópia de um dos textos produzidos pelo CoC, com referências para o documento original.

Tal contextualização pode ser feita de várias formas: por parte dos professores das disciplinas - mostrando aplicações dos tópicos estudadas para computação, trazendo professores ou especialistas em sala de aula que possam transmitir experiências relacionadas ao conteúdo aprendido - ou por parte dos próprios alunos - reunindo-se para palestras extra-classe, desenvolvendo bibliotecas que motivem os demais alunos a utilizar, pesquisar e até mesmo programar baseando-se nos conceitos adquiridos.

Nesse sentido, o Physimulation é uma ferramenta para simulação e animação de fenômenos físicos e tem como um dos principais objetivos integrar o conheci-

mento adquirido em FAP-126 (Física I) com um ambiente de programação.

Com este trabalho visamos contextualizar os assuntos abordados na disciplina com demonstrações de ambientes físicos, integrações com exercícios-programas (EP's) e a apresentação de desafios atuais de simulação no campo da física.

1.2 Problemas ao simular interações físicas

Há algumas dificuldades ao tentar simular um ambiente físico com a computação. No Physimulation, nos concentramos em duas delas.

A primeira é a escolha do tempo de simulação. Este é o tempo que fornecemos aos objetos físicos para interagirem entre si e comportar-se de modo a refletir a realidade. A segunda dificuldade refere-se aos algoritmos para detectar as colisões entre os objetos da simulação. Cada algoritmo tem suas vantagens e as desvantagens e é necessário compreendê-las para que os resultados da animação sejam satisfatórios.

1.3 Simulador

Com o Physimulation, o aluno da disciplina de física poderá simular comportamentos físicos estudados em sala de aula, por meio de criação de cenários e objetos com diferentes propriedades configuráveis. Exemplos de propriedades são: massa, tamanho, posição, velocidade inicial, coeficiente de atrito e de elasticidade.

Desde rampas, pequenas esferas e plataformas fixas a planetas orbitando em torno de um objeto de grande massa, tanto os alunos quanto o professor poderão utilizar o simulador para combinar tais objetos e observar resultados calculados em exercícios. Além disso, estes cenários podem ser guardados no sistema e carregados posteriormente para nova observação.

1.4 Estrutura da Monografia

Esta monografia está estruturada da seguinte forma: Na seção 2 descrevemos os conceitos e tecnologias estudadas para a realização do trabalho. São apresentadas as bibliotecas utilizadas, o papel de cada uma em nosso projeto e como elas interagem entre si. Nas seções 3 e 4 explicamos dois tópicos comuns e importantes ao realizar simulações físicas: são eles, respectivamente, a discretização da simulação - definição de tempo de simulação, problema do tamanho do passo,

granularidade - e o tratamento de colisões - algoritmos para filtragem de elementos, detecção de colisões e quais algoritmos são utilizados no Physimulation.

Já na seção 5 mostramos como foi o passo-a-passo de nosso trabalho, a metodologia utilizada e algumas dificuldades encontradas no caminho. A partir da seção 6, apresentamos um dos resultados obtidos com o trabalho: primeiramente, os ambientes físicos de demonstração e as razões por ter começado o trabalho por eles. Logo depois, na seção 7 descrevemos o Physimulation e as etapas para a criação de cenários físicos. Como aplicação do projeto, apresentamos duas integrações com exercícios-programas de disciplinas de introdução a computação na seção 8.

Concluímos o trabalho na seção 9, além de citar possíveis trabalhos futuros e mostrar desafios atuais da computação em simulações físicas que ainda necessitam de estudo e aprimoramento. Ao final, há um roteiro para instalação do ambiente necessário para executar o Physimulation, na seção 10.

2 Plataforma computacional

2.1 Esquema de dependências

O Physimulation depende de três bibliotecas: Chipmunk, Gosu e Glade. O programa começa com a leitura de dados com a interface criada com o Glade. Após a leitura, os dados lidos são convertidos em informações para que o Chipmunk possa processar e devolver o resultado da simulação. Os resultados das simulações são animadas com o Gosu. Toda interação entre as bibliotecas é feito com a linguagem Ruby.

A figura 1 representa um resumo do esquema de dependências entre as bibliotecas e o fluxo da dados da simulação.

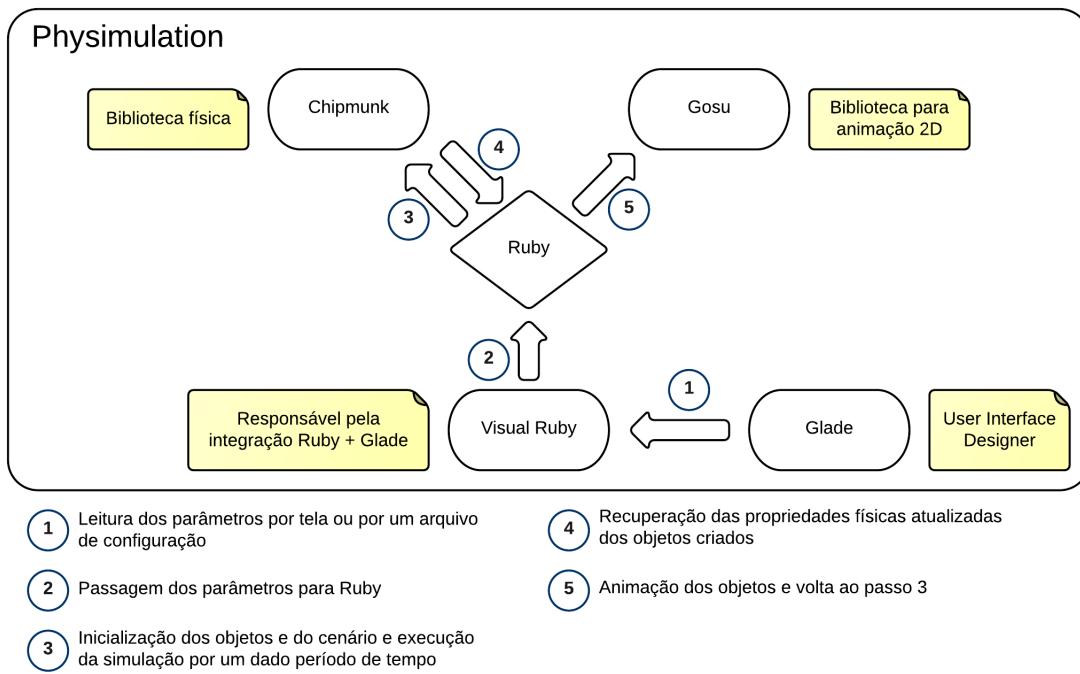


Figura 1: Esquema de dependências do Physimulation

Nas seções seguintes mostraremos algumas características das ferramentas utilizadas no Physimulation.

2.2 Ruby

É a linguagem de programação utilizada no Physimulation para integrar todas as bibliotecas.

Ruby possui uma sintaxe simples e direta e é puramente orientado a objetos, ou seja, tudo é objeto em ruby, até mesmo os tipos primitivos. TODO O Código 1 apresenta um exemplo de um método dos números, criado no Physimulation, onde o número é convertido de radianos para grau:

Código 1: Conversão de radianos em graus

```
def graus = 3.14.radians_to_degrees
```

Ruby também permite ao programador alterar partes da própria linguagem, tornado-a muito flexível. Podemos alterar comportamentos de classes ou objetos em tempo de execução. Um exemplo é listado no Código 2, onde adicionamos o método `radians_to_degrees` na classe `Numeric`:

Código 2: physics.rb

```
class Numeric
  def radians_to_degrees
    self * 180 / Math::PI
  end
end
```

Outro recurso interessante utilizado no Physimulation são os blocos de código. Os blocos podem ser anexados a qualquer método descrevendo como deve ser o comportamento da execução deste método. O Código 3 mostra um exemplo do uso de blocos em Ruby. Neste exemplo, passamos um bloco ao método `velocity_func` para atualizar a velocidade do objeto em relação a força gravitacional da Lua e da Terra:

Código 3: RocketSimulation.rb

```
self.body.velocity_func {
  |body, gravity, damping, dt|
  @earth_a = get_earth_acceleration(self)
  @moont_a = get_moon_acceleration(self)

  def self_a = vec2(@earth_a + @moon_a, 0)

  self.body.update_velocity(self_a, damping, dt)
}
```

2.3 Simulação com Chipmunk

Chipmunk é uma biblioteca física 2D escrita em C que permite a criação de objetos convexos e segmentos que interagem entre si em um ambiente físico. Para criar os objetos para simulação é preciso definir as propriedades físicas do corpo (`body`) e a forma do objeto (`shape`). Após a criação do objeto precisamos adicioná-lo a um ambiente físico (`space`) para que possamos iniciar a simulação.

Para criar o objeto é necessário primeiro definir o corpo. O corpo do objeto pode ser estático ou não. Um corpo estático é o corpo que não precisa de massa e momento de inércia pois não é influenciado por nenhuma força física, mantendo-se fixo no ambiente. A listagem do Código 4 mostra como criar o corpo de um objeto:

Código 4: physics.rb

```
if options[:static]
  @body = CP::StaticBody.new
else
  @body = CP::Body.new(massa, momento_inercia)
  @body.v = velocidade
  @body.w = velocidadeAngular
end

@body.p = vetor_posicao
@body.a = angulo
```

Após a definição das propriedades físicas de um objeto, é necessário definir a forma do objeto para o Chipmunk tratar as colisões. No Chipmunk é possível criar três formas primitivas: segmentos, círculos e polígonos convexos. No Physimulation criamos as formas de acordo com os parâmetros enviados pela tela inicial. É necessário passar o corpo associado à forma no momento da criação. O Código 5 mostra como são criadas as formas no Physimulation:

Código 5: physics.rb

```
def self.factory(body, params = {})  
    # Se existe um raio, criamos um círculo  
    if params.has_key? :radius  
        return Shape::Circle.new(body,  
            params[:radius],  
            Vec2::ZERO)  
    # Se existe uma espessura, criamos um segmento  
    if params.has_key? :thickness  
        return Shape::Segment.new(body,  
            params[:vectors][0],  
            params[:vectors][1],  
            params[:thickness])  
    # Criamos um polígono.  
    return Shape::Poly.new(body,  
        params[:vectors],  
        Vec2::ZERO)  
end
```

Após a criação do corpo e da forma, é preciso associá-los ao ambiente físico. Um ambiente físico é o espaço onde os objetos criados vão se interagir. É nele também que definimos propriedades como gravidade e amortecimento, como mostra o Código 6:

Código 6: physics.rb

```
@shape.add_to_space($space)  
$space.damping = 1.0  
$space.gravity = vec2(0.0, 5.0)
```

E finalmente, devemos passar um intervalo de tempo pré-determinado para o Chipmunk realizar toda simulação de física necessária:

Código 7: physics.rb

```
$space.step(@dt)
```

2.4 Animação com Gosu

A animação é feita com o Gosu, uma biblioteca para desenvolvimento de jogos em 2D em Ruby e C++. Fornece as seguintes funcionalidades:

- Criação de uma janela com um laço principal e callbacks;
- Criação de textos e imagens 2D;
- Som e música em vários formatos;
- Tratamento de eventos de entrada de teclado e mouse.

No Physimulation existem duas classes principais para a criação de animação. A primeira é a classe `PhysicObject` que é uma representação de um objeto do Chipmunk. Assim, qualquer objeto em uma animação que irá interagir fisicamente com outros objetos deverá ser uma instância do `PhysicObject`. Tanto as propriedades do corpo quanto da forma dos objetos estão nesta classe.

Código 8: physics.rb

```
class PhysicObject < Chingu::BasicGameObject
    # Adiciona informações do corpo e da forma.
    trait :physics
end
```

A segunda classe é a `PhysicWindow`, que é a janela da animação propriamente dita. Os principais métodos desta classe são o *update* e o *draw*. O método *update* chama o Chipmunk para realizar a simulação. Com a simulação finalizada, o método *draw* desenha na tela de animação todos os objetos com as posições atualizadas.

Código 9: physics.rb

```
class PhysicWindow < Chingu::Window

    # Configuracao da janela
    def setup
        self.caption = "TCC_Demos_-_Alberto_e_Issao"
        self.input = { esc: :exit, d: :toggle_lines }

        @dt = 1.0 / 40.0
        @substeps = 6

        @info_area = Chingu::Text.create("", :x => 300,
                                         :y => 30, :color => Gosu::Color::YELLOW)
        @feedbackMessage = ""
        TexPlay.set_options :caching => false
    end

    def update
        super
        @info_area.text = info
        $space.step(@dt)
    end

    def draw
        super
        @info_area.draw
    end
end
```

2.5 Interfaces com Glade e Visual Ruby

Glade é uma ferramenta que facilita o desenvolvimento de interfaces baseado no GTK+. Ela é independente de linguagem de programação e não produz código para eventos, como um clique de botão, mas sim um arquivo XML. Esse arquivo XML é aproveitado pelo Visual Ruby, uma ferramenta que simplifica o processo de criação de janelas baseadas em GTK+ e é completamente integrado com Glade. Para começar a utilizar, basta criar um arquivo .rb que carrega o arquivo XML do Glade:

Código 10: simulation.rb

```
class Simulation
  include GladeUI

  def show
    # Carrega o arquivo xml
    load_glade(__FILE__)
    # Disponibiliza as propriedades definidas no
    # xml em Ruby
    set_glade_all
    # Mostra a interface criada
    show_window
  end
end
```

3 Discretização da simulação

A simulação no Chipmunk ocorre em intervalos de tempo, ou seja, precisamos indicar o tempo em que os objetos vão interagir como vimos na seção 2. Esse intervalo que passamos para o Chipmunk é o tempo de simulação. É preciso definir com atenção esse valor para que a simulação não fique muito rápida ou muito lenta. Embora pareça trivial, há diferentes implementações que indicam como definir esse tempo de simulação. Nas próximas seções apresentaremos as implementações para tempo de simulação fixo e variável.

3.1 Fixo

A maneira simples é fixar o tempo de simulação, como por exemplo 1/60 de um segundo:

Código 11: Implementação de tempo de simulação fixo

```
def dt = 1/60

while (running) do
    $space.step(dt)
end
```

Na maioria das simulações o código acima é o ideal. Se a simulação coincidir com a taxa de atualização de quadro (*frame rate*) e ainda se a chamada `step(dt)` não demorar mais do que 1/60 de um segundo, então a solução é bastante satisfatória. Mas no mundo real não sabemos o valor da taxa de atualização do quadro. Por exemplo, a execução de uma simulação em uma máquina mais lenta, que não pode atualizar os quadros em 60 fps (*frames per second*), pode deixar a simulação mais lenta ou mais rápida.

Uma solução para evitar esse problema é o tempo de simulação variável.

3.2 Variável

A idéia do tempo de simulação variável é medir quanto tempo leva o quadro anterior e, em seguida, alimentar a simulação com esse tempo:

Código 12: Implementação de tempo de simulação variável

```
def current_time = Time.now

while (running) do
    def new_time = Time.now
    def dt = get_simulation_time(new_time - current_time)

    $space.step(dt)
end
```

A solução é simples e aparentemente razoável. Em um computador com taxa de atualização de quadro de 30 fps é considerado o tempo de 1/30 de um segundo para a simulação. Mas existe um problema com essa abordagem: a simulação não garante resultados corretos para todos os valores de tempo de simulação. Dependendo do valor do tempo e da velocidade dos objetos, poderemos observar objetos atravessando paredes ou outros objetos, diminuindo o realismo e o valor da animação.

3.3 Tempo de Simulação no Physimulation

Muitos estudos mostram que tempo de simulação fixo é indicado para simulações físicas e tempo de simulação variável para jogos, ou seja, quando a velocidade da animação for mais importante que a própria simulação. Porém, no Physimulation, tanto a simulação quanto a animação são importantes.

No nosso caso, o tempo de simulação fixo foi suficiente pois não afetou consideravelmente a animação. Consequentemente, o Physimulation pode ficar mais lento em computadores com menor poder de processamento.

4 Colisões

Uma das características das plataformas que suportam simulações físicas como o Chipmunk é a capacidade de detectar colisões entre os objetos. Para detectar colisões de forma eficiente essas plataformas utilizam um processo com duas fases: *broad phase* e a *narrow phase*.

A finalidade da *broad phase* é evitar a realização de cálculos caros para corpos distantes um dos outros. Para gerar os pares de objetos que devem passar pelo algoritmo de detecção de colisão, o Chipmunk suporta as seguintes estruturas de dados: árvores AABB e o *spatial hashing*. Na versão 5 do Chipmunk a estrutura de dados utilizada é o *spatial hashing* enquanto na versão 6 são as árvores AABB.

Narrow phase é a fase onde pares de objetos são verificados cuidadosamente em termos de colisão. O Chipmunk utiliza nesta fase o *Separating Axis Theorem*, que suporta somente polígonos convexos. Devido a esse motivo, não é possível criar objetos côncavos no Chipmunk.

Nas próximas seções serão explicadas resumidamente os algoritmos de cada fase.

4.1 *Broad phase* - Árvore AABB

Para entender a árvore AABB, é preciso primeiro entender o que é AABB. AABB é um acrônimo para *Axis Aligned Bounding Box*, ou seja, caixas delimitadoras de objetos em formas de retângulo alinhadas com os eixos x e y, como mostra a figura 2:



Figura 2: O retângulo azul representa uma caixa delimitadora do objeto

A árvore AABB é uma árvore binária que faz proveito da estrutura da AABB para armazenar e consultar objetos no espaço 2D. O nó da árvore pode guardar o próprio objeto ou possuir dois nós filhos. Quando um nó guarda um objeto, a caixa delimitadora deste nó deve conter a caixa delimitadora do objeto. Se o nó possui nós filhos, sua caixa delimitadora deve conter as caixas delimitadoras

dos nós filhos. Um ponto importante é que esta estrutura precisa de uma heurística para inserir os nós na árvore. No caso do Chipmunk existe uma heurística baseada na caixa delimitadora do objeto e também em relação à sua velocidade. Na figura 3 mostramos um exemplo de uma árvore AABB de três objetos A, B e C:

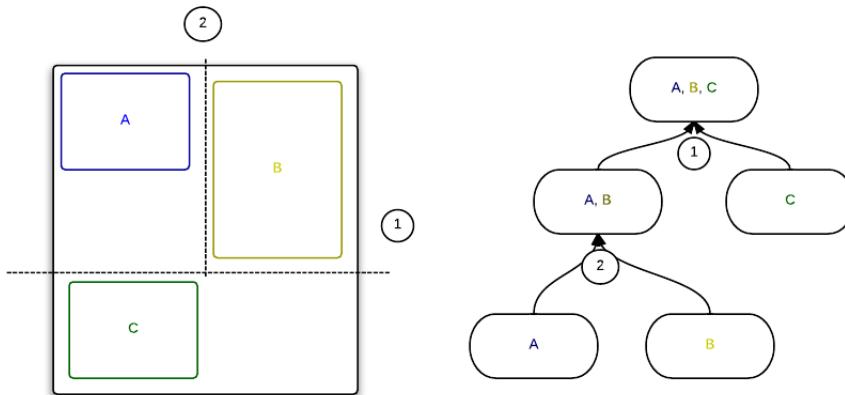


Figura 3: Exemplo de uma árvore AABB

A caixa delimitadora da raiz é o retângulo preto contendo os objetos A, B e C. As duas novas caixas delimitadoras criadas pela linha indicado pelo número 1 são agora filhas da raiz. E finalmente a linha indicado pelo número 2 criam mais duas caixas delimitadoras separando os objetos A e B.

A figura 4 mostra como são definidos os pares de objetos que passarão pela *narrow phase*. Começando pela raiz, a caixa delimitadora do objeto D intersecta somente com a caixa delimitadora do nó esquerdo. Em seguida, a caixa delimitadora do objeto D intersecta somente com a caixa delimitadora da folha contendo o objeto A. Então o par criado nessa busca é o par {A, D}.

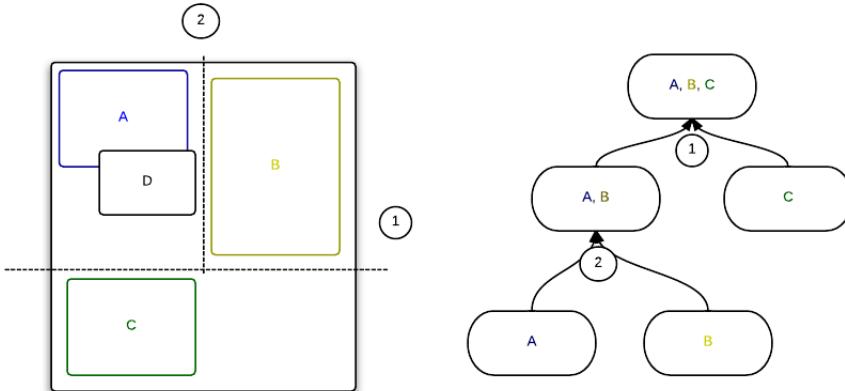


Figura 4: Exemplo de detecção de colisão. O objeto D é comparado somente com a folha contendo o objeto A.

4.2 Broad phase - 1D sweep and prune

A idéia deste algoritmo é varrer as caixas delimitadoras dos objetos criando assim os pares de objetos que deverão ser passados para a fase seguinte.

Seguindo o exemplo da figura 5, o algoritmo mantém uma lista de objetos que estão sendo varridos. Quando a varredura encontra o início do objeto, ela o inclui na lista. Quando a varredura encontra o final do objeto, ela o exclui da lista. No exemplo observamos que a lista [A, B, C] criam os pares {A, B}, {A, C} e {B, C} enquanto a lista [B, D] cria o par {B, D}.

Esse algoritmo, de acordo com a documentação do Chipmunk 6, pode ser muito eficiente em jogos voltados para dispositivos móveis se o seu mundo é muito comprido e plano como um jogo de corrida.

4.3 Broad phase - Spatial hashing

Spatial hashing é um processo onde o espaço de duas ou três dimensões é projetado em uma tabela *hash* de uma dimensão. Para isso o Chipmunk divide o espaço em células do mesmo tamanho, onde cada célula é mapeada a uma entrada na tabela *hash*. Um objeto sempre é mapeado nas células que ele está presente. Com esta estrutura é fácil de criar os pares de objetos. Dado um objeto precisamos verificar quais as células que ele está presente e criar o par para cada objeto

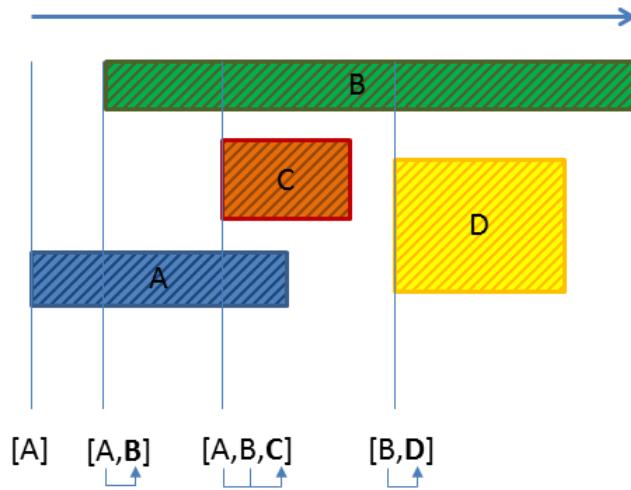


Figura 5: Exemplo de funcionamento do algoritmo *sweep and prune*

existente nas entradas da tabela destas células.

O tamanho das células é de suma importância para obter uma boa eficiência. Nas próximas figuras, as áreas cinzas representam as células. Quanto mais escu- ras, mais objetos estão mapeadas na célula. O ideal é que o tamanho das células não seja muito pequeno nem muito grande.

Esta estrutura de dados é a mais indicada quando a simulação possui um con- junto grande de objetos e de mesmo tamanho.

4.4 *Narrow phase - Separating Axis Theorem*

No Chipmunk existe somente um algoritmo para detecção de colisão, baseado no *Separating Axis Theorem*. A idéia do algoritmo é verificar se é possível desenhar um segmento de reta entre duas formas convexas. Se existe tal segmento, então não há colisão entre eles, caso contrário, as duas formas estão colidindo. No caso de formas côncavas isso não é verdade. Devido a essa restrição do algoritmo o Chipmunk impõe que todo polígono seja convexo.

É um algoritmo rápido que elimina a necessidade de ter um código de detecção de colisão para cada tipo de forma. A figura 9 ilustra a idéia do algoritmo.

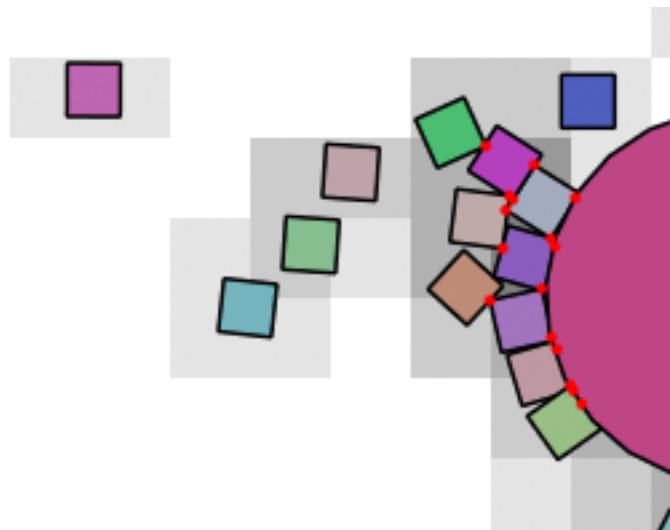


Figura 6: Cada célula contém poucos objetos. Estado ideal para *spatial hashing*

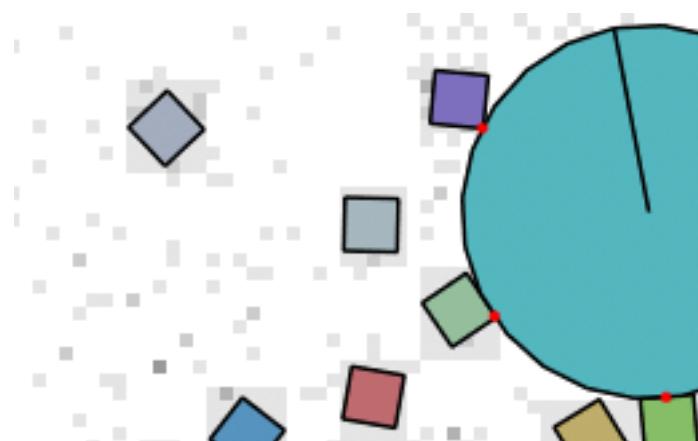


Figura 7: Cada objeto está mapeado em muitas células

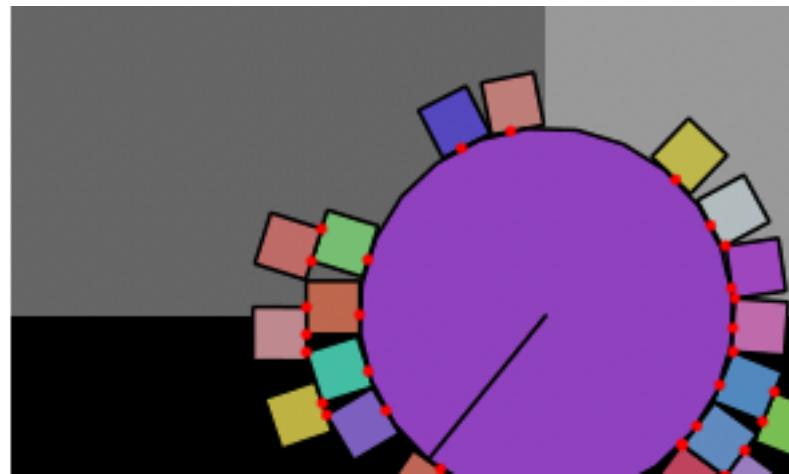


Figura 8: Cada célula contém muito objetos

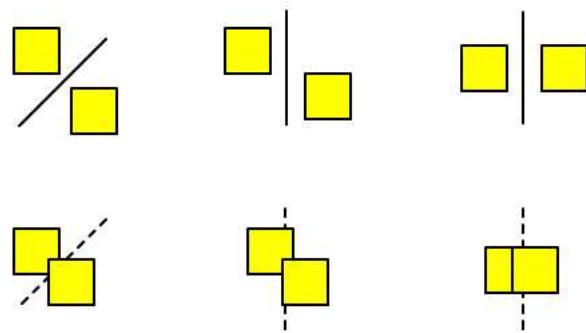


Figura 9: Se existe uma linha que separa os polígonos convexos, então eles não estão colidindo.

5 Atividades realizadas

Nesta seção explicamos o processo da realização do trabalho, desde o desenho do projeto no papel e ideias iniciais até a criação dos ambientes físicos de demonstração, interface de criação de cenários (Physimulation) e finalmente as integrações com exercícios-programas.

5.1 Estudo inicial

Após nosso orientador nos apresentar as discussões e ideias do Grupo de Apoio ao BCC (seções 1 e 11), decidimos planejar nosso trabalho de formatura levando em conta esta preocupação com os alunos do BCC. Inicialmente, nosso foco seria nas disciplinas de física (FAP-126) e estatística (MAE-121).

Partimos para a leitura de artigos e páginas da internet que tivessem relação com esta ideia. A seguir listamos algumas das referências mais interessantes neste processo:

- *An Introduction to Computer Simulation Methods: Applications to Physical Systems*, Gould, Tobochnik
- *Evaluation of real-time physics simulation systems*, Boeing e Bräunl
- *Esquema de detecção e resposta a colisões para animação física simplificada*, Temistocles, Atencio
- Box2D TODO
- Hotruby TODO

Percebemos que já existiam muitos estudos na área de simulações e animações físicas, motivados principalmente pela criação de jogos para computador, *video-games* e celulares. Neste último campo, com a recente expansão do uso de *smartphones*, havia um número considerável de bibliotecas voltadas para reprodução de ambientes físicos. Durante esta fase de pesquisa, o projeto começava a tomar mais forma e descartamos a possibilidade de trabalhar com a disciplina de estatística. A partir deste momento, concentraríamos esforços apenas em física.

Pesquisamos então as bibliotecas *open-source* disponíveis e adequadas ao nosso objetivo de integração com a disciplina do IME. Como já havíamos decidido trabalhar com a linguagem Ruby (seção 2), procuramos por *frameworks* nesta linguagem. Nesta fase de pesquisa, as bibliotecas que selecionamos para nosso trabalho foram o Chipmunk e Gosu (seção 2).

TODA Além dos *frameworks* vistos na seção anterior, outras bibliotecas foram consideradas para o projeto. Por exemplo...Porém Então

5.2 Ambientes de demonstração

Após definida a linguagem e as ferramentas que utilizaríamos no trabalho, partimos...

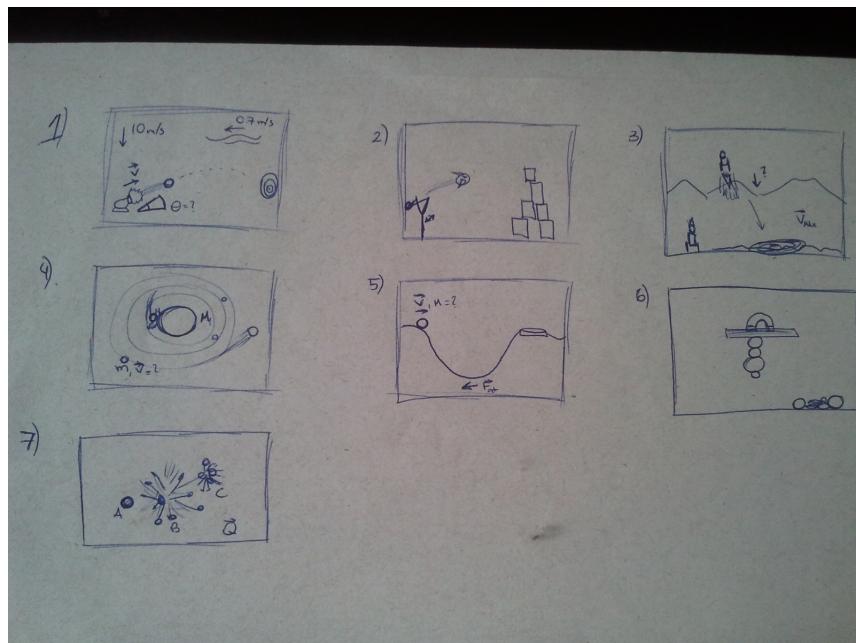


Figura 10: Primeiros rascunhos das demonstrações

5.3 Criador de Cenários

Com um conhecimento mais profundo das bibliotecas após a implementação dos ambientes de demonstração, começamos a montar a interface de criação de cenários físicos. Como já tínhamos esta ideia em mente desde o início do projeto, centralizamos boa parte do código que era importante em classes que facilitariam a criação de simulações genéricas. Esta estratégia nos facilitou bastante o trabalho desta etapa.

TODO Encontramos o glade...

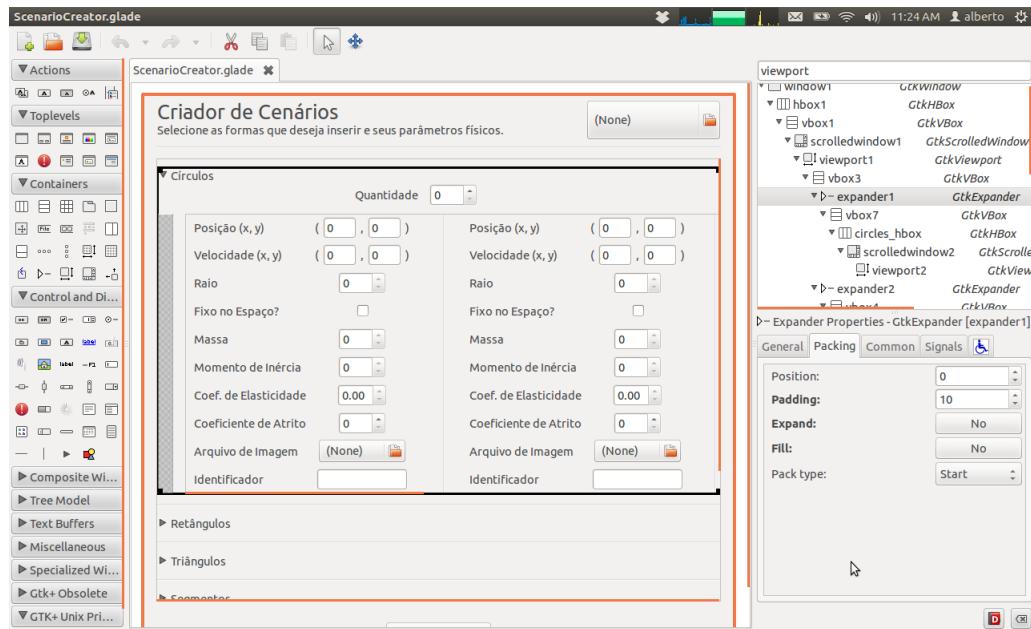


Figura 11: Gnome Glade

Discutimos durante as reuniões com o orientador e o João Kerr como seria a interface, as propriedades físicas que poderiam ou não poderiam ser alteradas...
TODO Falar da questão do momento de inércia...

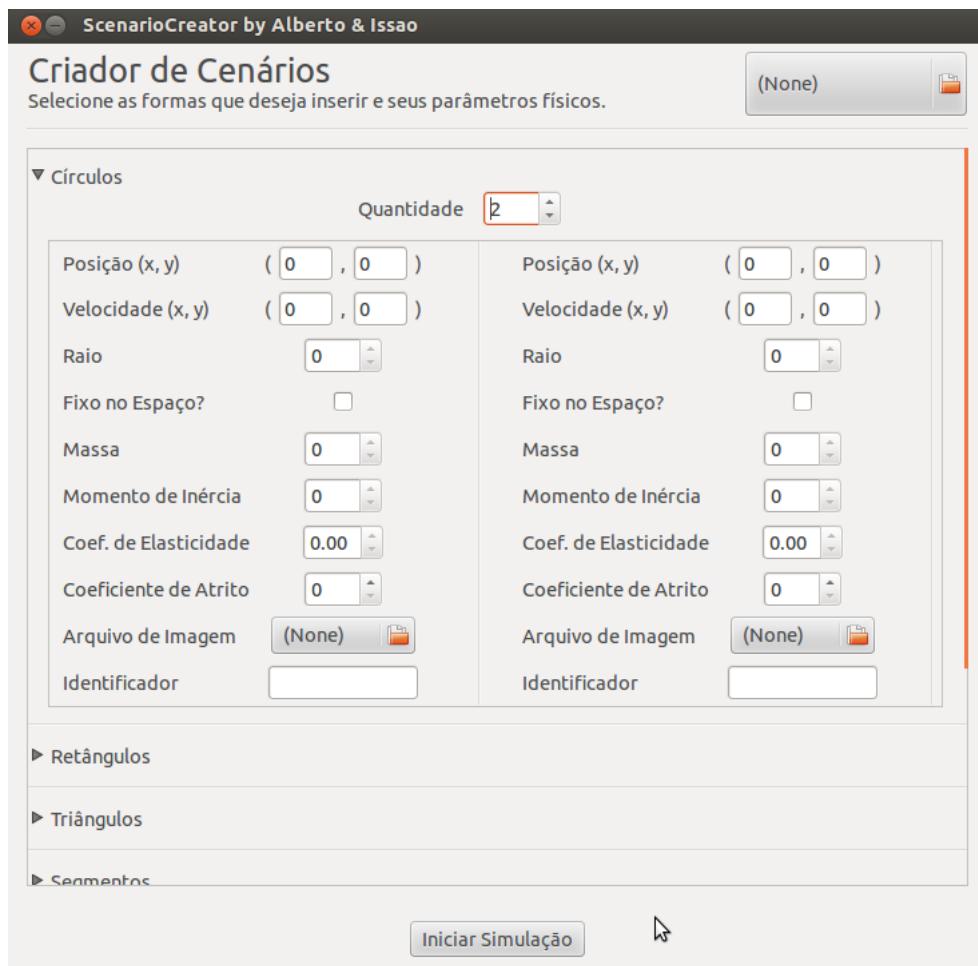


Figura 12: Interface ao final das discussões

6 Animações produzidas

As animações de diferentes tipos de ambientes físicos foram nossos primeiros resultados obtidos. Após desenhar alguns cenários simples que ilustravam tópicos da disciplina de física, como descrito na seção 5, partimos para implementação. Nossa principal objetivo nesta etapa era compreender melhor o funcionamento do Chipmunk: seus conceitos, seus recursos e suas limitações.

Além disso, como tínhamos em mente implementar um criador de cenários, a medida que implementávamos estas demonstrações extraímos todo código relevante para classes auxiliares que pudessem ser reutilizadas posteriormente. Atualmente, a classe `physics.rb` contém grande parte deste código extraído.

Código 13: Trecho de código do `physics.rb`

```
# @param [Hash] options : mapa com as opcoes de
#                      configuracao do objeto fisico.
def setup_trait(options = {})
  self.color = options[:color] || Gosu::Color::WHITE
  self.zorder = options[:zorder] || 100
  self.factor = options[:factor] || options[:scale]
  || $window.factor || 1.0
  @visible = true unless options[:visible] == false

  if options[:static]
    @body = CP::StaticBody.new
  else
    @body = CP::Body.new(options[:mass],
                        options[:moment_inertia])
    @body.v = options[:v] if options[:v]
    @body.w = options[:rotational_velocity] || 0
    @body.add_to_space($space)
  end
  (...)
```

6.1 Descrição

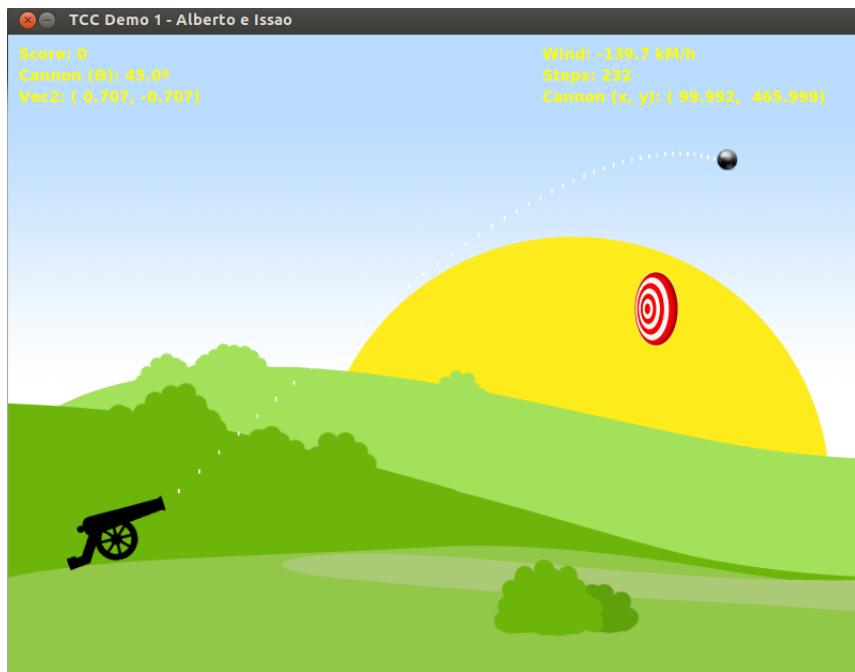


Figura 13: Cannonball

Cannonball foi nosso primeiro cenário físico. Baseado em jogos comuns de tiro ao alvo, como arco-e-flecha e catapultas, o objetivo é acertar uma bola de canhão em um alvo de posição escolhida aleatoriamente. O usuário pode modificar a angulação do canhão e atirar a bola pressionando a tecla de espaço. Há um vento de intensidade também calculada de forma aleatória, na direção horizontal e sentido contrário ao do trajeto da bola ao alvo. As informações relevantes ao usuário - ângulo do canhão, intensidade do vento, pontuação, entre outros - são exibidas no topo da janela.

Nossa intenção com este cenário era ilustrar o comportamento de lançamentos oblíquos e resistência do ar.

Posteriormente, utilizamos o código de *Cannonball* na integração com exercício-programa *Angry-Bixos*, descrita na seção 8.

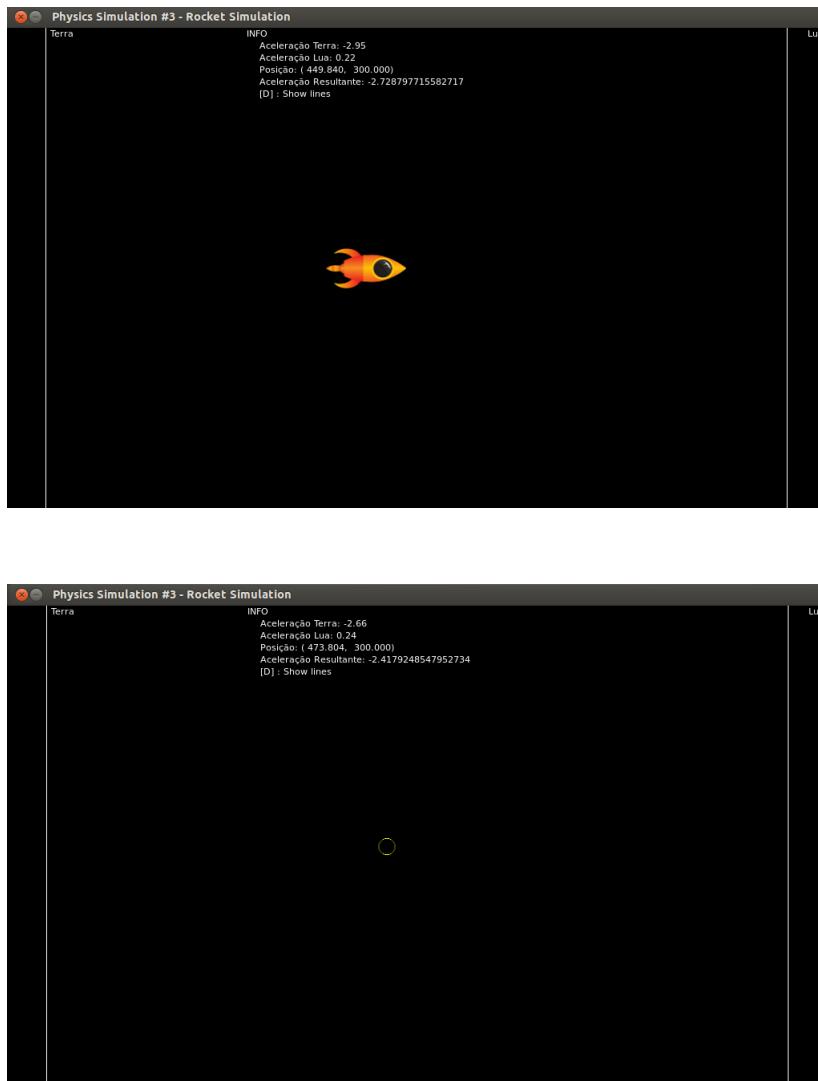


Figura 14: Rocket Simulation

A ação conjunta das gravidades da Terra e da Lua sobre um objeto é simulada em *Rocket Simulation*. Controlando um foguete, o usuário pode movimentar-se com as setas do teclado entre os dois corpos, representados por retas nas extremidades esquerda e direita da tela. Por ter uma massa bem maior, a atração exercida pela Terra é maior em boa parte da tela, exceto em regiões bem próximas à Lua.

Por motivos de simulação, não reproduzimos os valores reais de massa de ambos os corpos, pois a região em que a Lua teria maior efeito sobre a nave seria ínfima e não proporcionaria ao usuário a experiência que gostaríamos.

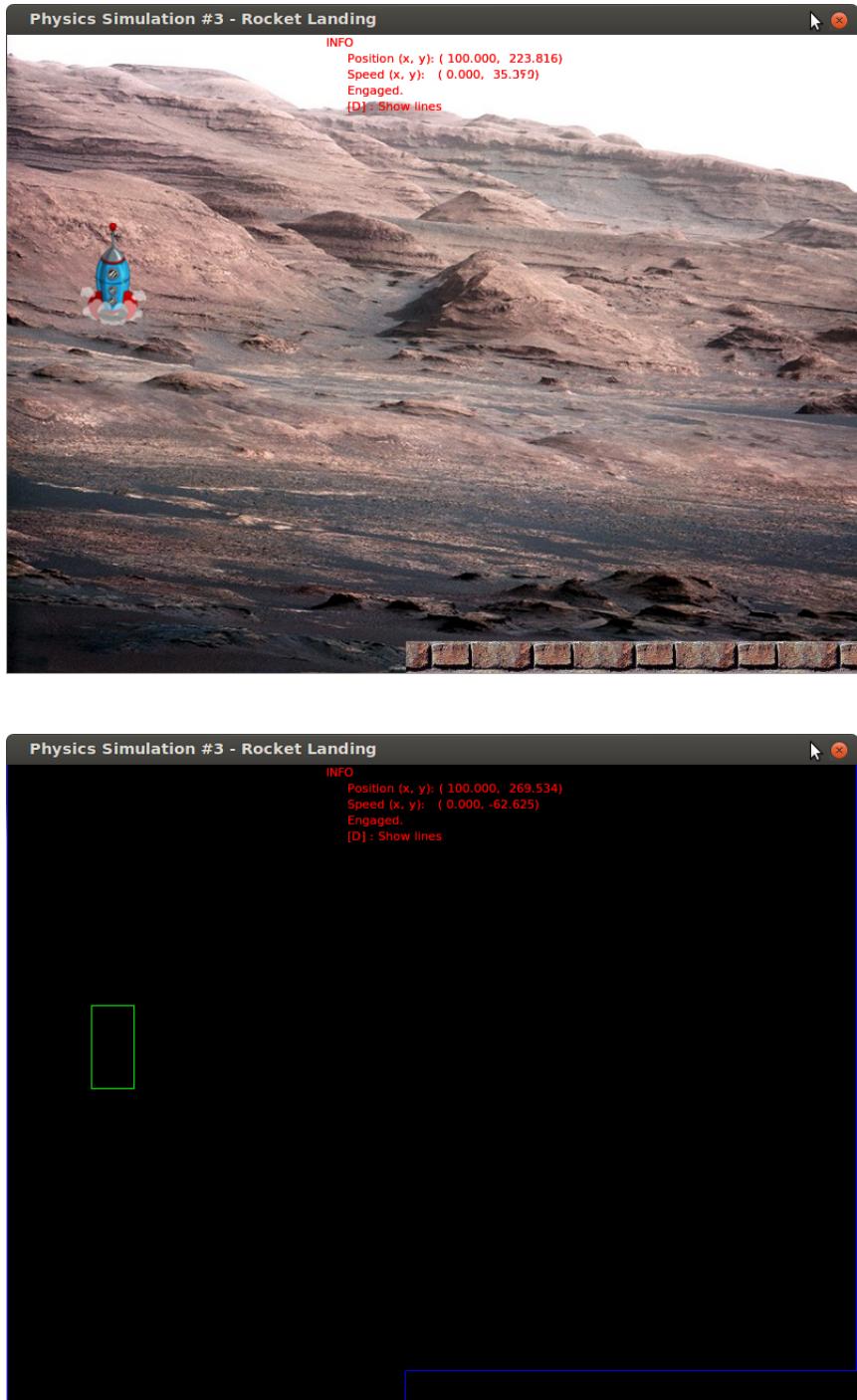


Figura 15: Lunar Landing

Baseado no jogo *Lander* TODO, o objetivo é aterrissar um foguete em uma superfície plana com uma velocidade bem pequena. Se o usuário não conseguir

diminuir adequadamente a velocidade antes de tocar a superfície, o foguete explode. Em caso de sucesso, ele pousa e uma mensagem de sucesso é exibida. As setas do teclado movimentam o foguete.

Com a implementação desta demonstração utilizamos em conjunto o código de detecção de colisão do *Cannonball* com a manipulação de objetos de *Rocket Simulation*.

Tanto neste e em outros cenários há a função "Raio-X" (tecla 'd'), que mostra ao usuário o esqueleto dos objetos do cenário. Em outras palavras, os *shapes* que estão ativos no *space* da simulação. Mais detalhes na seção 2.



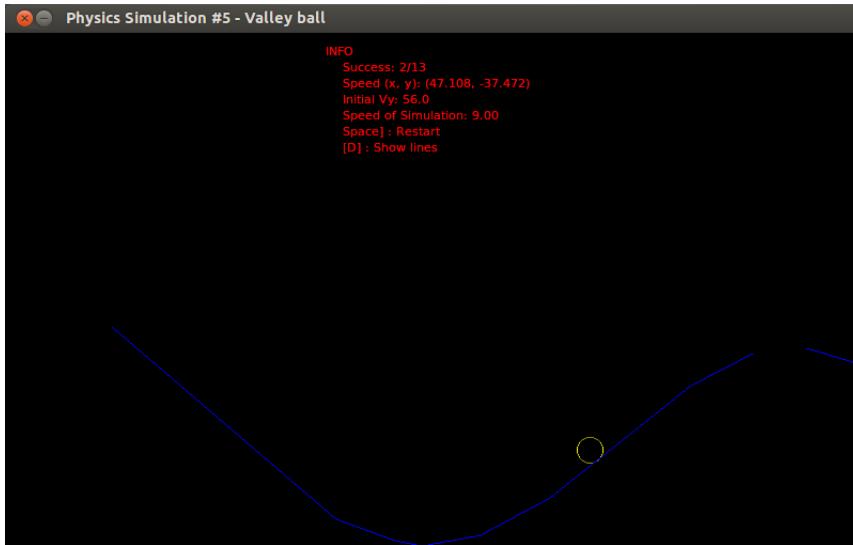


Figura 16: Valleyball

Em *Valleyball*, tentamos ilustrar o Princípio de Conservação da Energia Mecânica e a aplicação da força de atrito. Uma bola é lançada a uma certa altura do chão com velocidade vertical determinada aleatoriamente. Ela então desliza sobre um pequeno vale e dependendo da velocidade inicial ela pode: 1) voltar ao vale e perder velocidade até a situação de repouso; 2) cair em um buraco localizado próximo ao topo de uma montanha; ou 3) prosseguir seu movimento para fora do cenário. O que determina a situação resultante é a velocidade inicial da bola. A tecla de espaço reinicia a simulação com uma nova velocidade para a bola.

Nesta demonstração utilizamos pela primeira vez o controle de velocidade da simulação. Dentro de alguns limites, o usuário pode alterar o tamanho do passo da simulação (seção 3), tornando a animação mais rápida ou mais lenta. Isso é útil, por exemplo, para acelerar alguma etapa da simulação cujos resultados sejam previsíveis ou pouco interessantes.

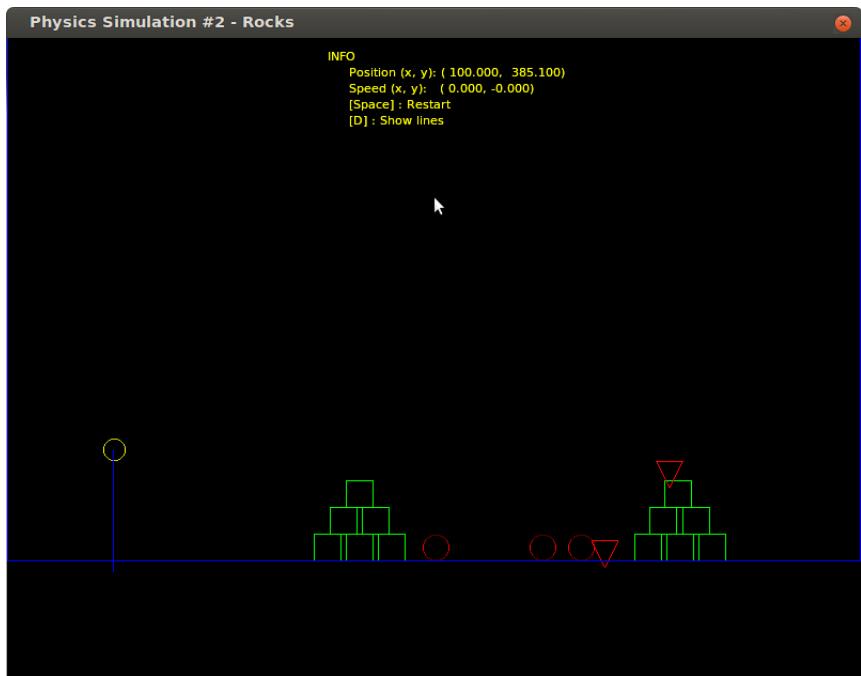


Figura 17: Rocks

Rocks foi nosso último cenário de demonstração. Parecido com *Cannonball*, o objetivo também é acertar alvos da tela - neste caso os objetos de cor vermelha.

Porém é o usuário que determina a força que a bola será atirada, dependendo de quanto ele "puxar" a bola segurando e arrastando o *mouse*, de modo semelhante ao comportamento de um estilingue. Com este cenário físico gostaríamos de simular um jogo que recentemente ficou bastante conhecido, o *Angry Birds*. Além dos alvos, há cubos empilhados na tela, sendo estes de dois tipos: de pequena massa (cor de madeira) e de grande massa (cor de pedra). O resultado das colisões entre os objetos da tela variam de acordo com a massa do objeto.

Além de ser o único cenário de demonstração que possui interação com o *mouse*, *Rocks* consolidou nosso conhecimento das bibliotecas utilizadas e das modificações que precisávamos fazer para implementar o Physimulation. Em termos de física, a novidade deste cenário é a utilização de Energia Potencial Elástica no lançamento do objeto.

6.2 Visualização das animações

Para visualizar as animações descritas nesta seção, o aluno deve executar o seguinte comando:

```
$ cd <<DIR_PHYSIMULATION>>/Simulation
$ ruby main.rb demos
```

onde `DIR_PHYSIMULATION` é o diretório em que o Physimulation foi instalado. Em seguida, deve escolher uma das animações listadas no programa e clicar em "Iniciar". Caso haja algum problema ao executar qualquer um dos comandos, verificar se os passos da seção 10 ("Instruções de instalação da plataforma") foram seguidos corretamente.

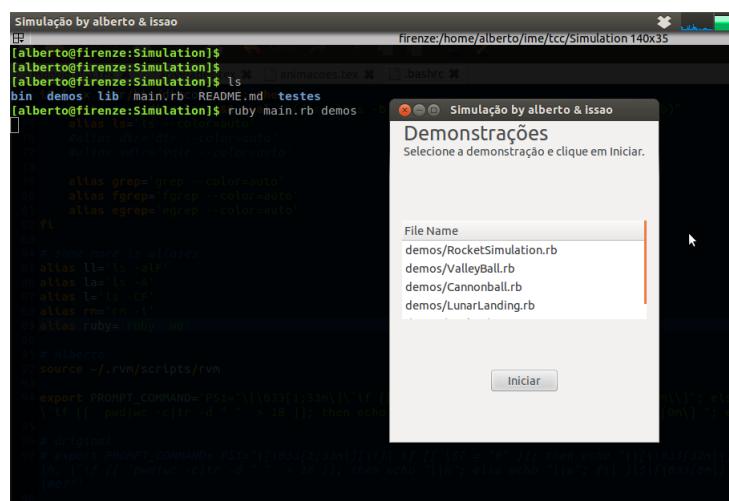


Figura 18: Menu de demonstrações

7 Physimulation

Em uma das reuniões com nossos orientadores, percebemos que com poucas primitivas - objetos simples como círculos, triângulos e retângulos - e com algumas propriedades físicas configuráveis poderíamos gerar uma grande quantidade de cenários diferentes entre si. Tais propriedades, como massa, gravidade, coeficiente de atrito e de elasticidade, já eram do nosso domínio de conhecimento pois as utilizamos ao implementar os cenários de demonstração.

O Physimulation é o resultado dessa ideia e é o principal produto do nosso trabalho.

7.1 A ferramenta

Com o Physimulation o usuário pode criar quatro tipos de objetos: círculos, triângulos, retângulos e segmentos, em qualquer posição da tela. Cada tipo de objeto possui algumas propriedades específicas à sua forma. Por exemplo, o círculo é a única das primitivas que possui o atributo "raio". Porém, a grande maioria dos parâmetros são comuns a todas as formas:

- Posição;
- Massa;
- Momento de Inércia;
- Coeficiente de Atrito;
- Coeficiente de Elasticidade;
- entre outros.

Ao posicionar o *mouse* nos campos editáveis são exibidas ao usuário as descrições de cada propriedade. Além disso, para cada atributo há um valor pré-definido por padrão, caso o aluno ou professor não esteja interessado em determinar seu valor.

Além de determinar as propriedades de cada objeto, há a possibilidade do usuário configurar parâmetros do próprio ambiente de simulação. Tais propriedades são:

- Gravidade: um vetor na forma (x, y) representando a gravidade do espaço;

- Coeficiente de Amortecimento: porcentagem que representa a velocidade final de um objeto em relação a velocidade inicial, em um passo da simulação. Por exemplo: com um amortecimento de valor 0.9, todos os objetos perderão 10% de suas velocidades a cada passo da simulação;
- Gravitação Pontual: se for selecionada, o simulador irá considerar a atração gravitacional entre os objetos da tela. Útil para criar cenários com movimentos orbitais;
- Espaço Limitado: se for selecionada, o simulador irá construir quatro segmentos que limitarão o espaço da simulação para o espaço visível da tela.

Para cada tipo de objeto há um campo "quantidade", que informa ao simulador o número de objetos do tipo escolhido. Este campo deve ser preenchido antes da criação dos objetos físicos e pode ser alterado posteriormente.

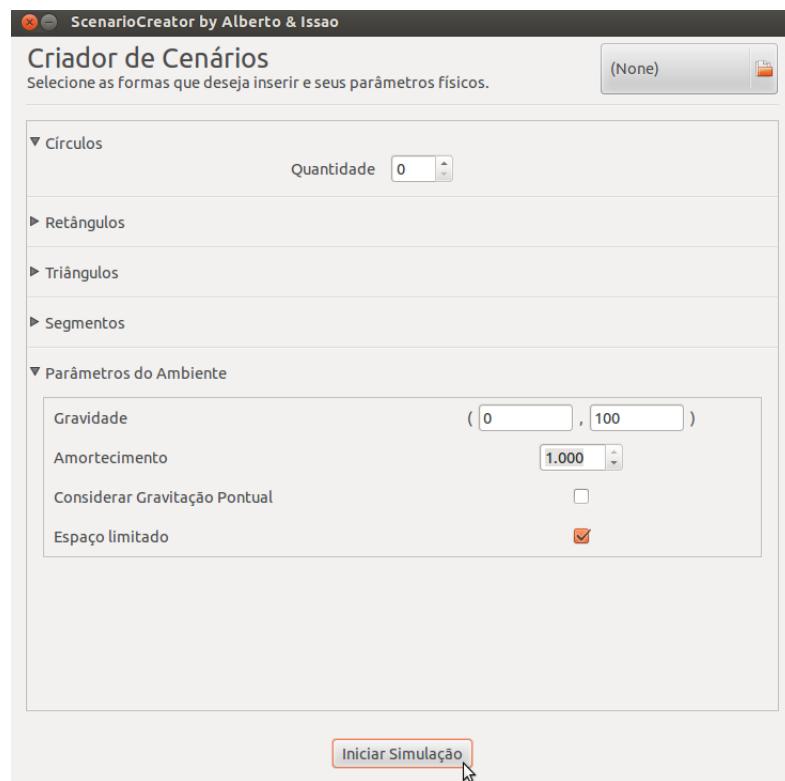


Figura 19: Tela inicial do Physimulation

7.2 Criando um cenário

Para mostrar um caso de uso do simulador, vamos utilizar a figura abaixo como referência para criar um cenário físico.

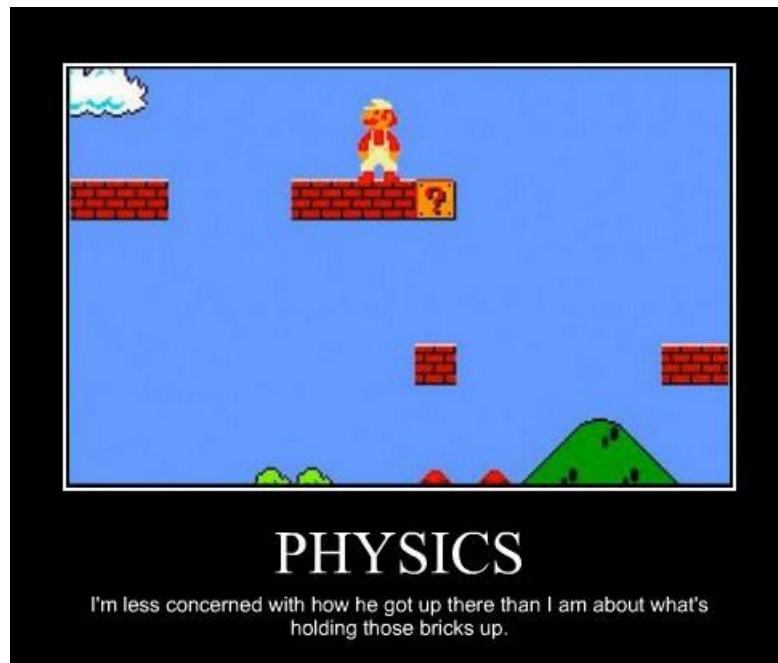


Figura 20: Objetos estáticos

Nesta imagem, há quatro blocos suspensos no ar e uma montanha verde com o formato parecido com um triângulo, além do personagem principal que está em cima de um dos blocos. Para esta simulação, faremos a seguinte correlação: os blocos suspensos serão retângulos do Physimulation, a montanha verde um triângulo e o personagem de pé uma pequena bola.

Para executar o physimulation, o usuário deve executar no terminal:

```
$ cd <<DIR_PHYSIMULATION>>/Simulation  
$ ruby main.rb
```

onde `DIR_PHYSIMULATION` é o diretório em que o Physimulation foi instalado. Novamente, caso haja algum problema ao executar qualquer um dos comandos, deve-se verificar o conteúdo da seção 10, "Instruções de instalação da plataforma".

A seguir descrevemos os passos para criar este cenário:

1. Expandir a aba "Círculos" e aumentar a quantidade para 1.

2. Determinar a posição do círculo = (350, 100), massa = 30, raio = 30.
Nota: a origem das coordenadas no Phisimulation é o canto superior esquerdo, com coordenadas positivas aquelas abaixo e à direita da origem.
3. Expandir a aba "Retângulos" e aumentar a quantidade para 4.
4. Preencher os atributos dos retângulos com os seguintes valores:
 - 1º: posição = (80, 200), altura = 60, largura = 120
 - 2º: posição = (350, 200), altura = 60, largura = 160
 - 3º: posição = (400, 400), altura = 60, largura = 60
 - 4º: posição = (720, 400), altura = 120, largura = 60
5. Em todos os retângulos anteriores, habilitar a opção "Fixo no espaço?".
Nota: ao habilitar esta opção, utilizaremos *static shapes* ao invés de *shapes* habituais. Na prática, estamos informando ao simulador que estes objetos não devem sofrer forças do ambiente físico (*space*), como a gravidade. Para mais detalhes, ver a seção 2.
6. Por último, criar um triângulo com os seguintes atributos: posição = (600, 520), ponto A = (100, 0), ponto B = (0, 100), ponto C = (200, 100).

Ao iniciar a simulação, temos este cenário:

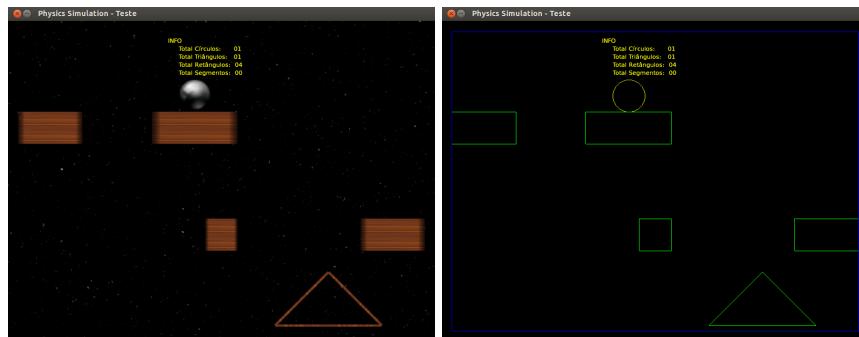


Figura 21: Cenário criado com o Phisimulation

Podemos fazer uma pequena modificação a este cenário e acrescentar uma velocidade horizontal positiva à bola, o que a fará despencar do bloco superior.

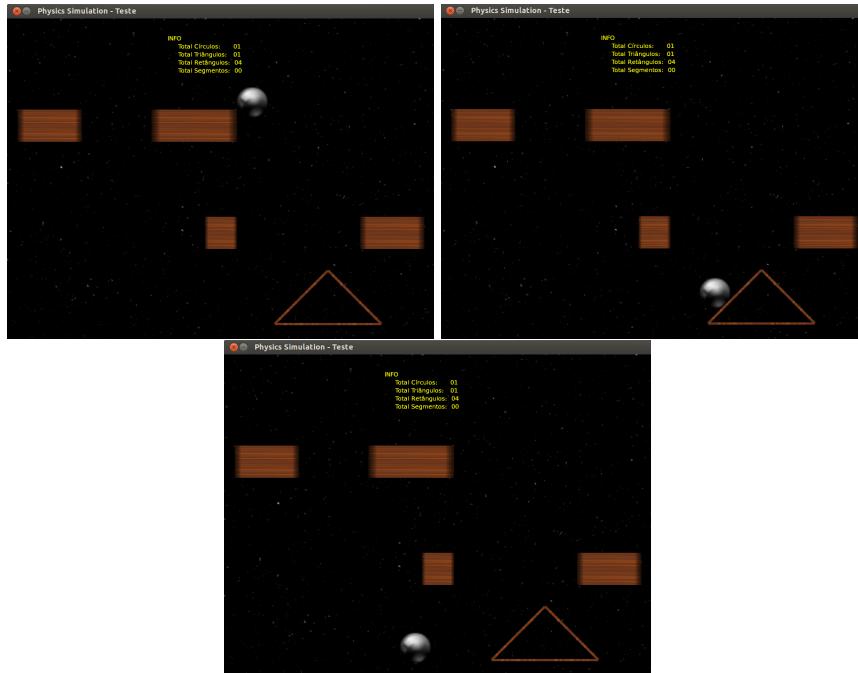


Figura 22: Mesmo cenário adicionando velocidade à bola

Há uma outra forma de iniciar uma simulação no Physimulation: utilizando um arquivo de configuração existente, com todas as propriedades dos objetos e do ambiente físico definidas. Para carregar um cenário, é necessário que o usuário use o botão no canto superior direito da interface, como mostrado na figura abaixo.

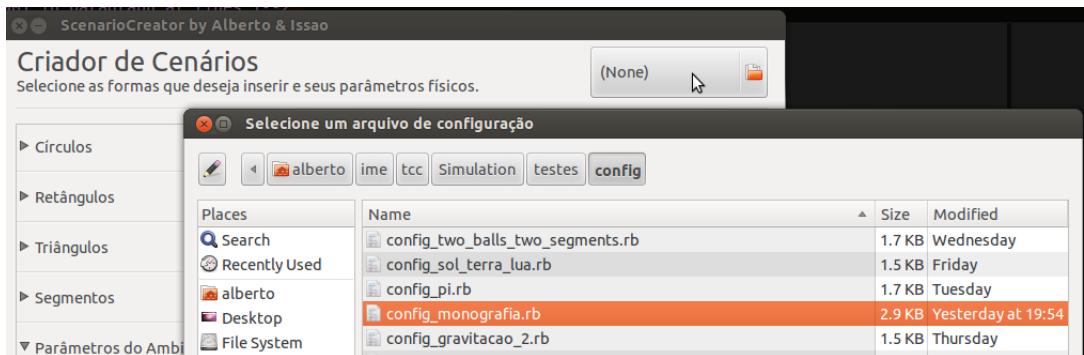


Figura 23: Carregando cenários

Ao final desta seção há mais detalhes sobre o armazenamento e carregamento de cenários físicos.

7.3 Outros exemplos

Para o aluno ou professor que deseja aprofundar seus conhecimentos sobre a ferramenta, recomendamos a realização de vários testes com o mesmo cenário gerado, com pequenas alterações de valores entre cada teste. É importante fixar certas propriedades e observar os resultados obtidos, ao invés de modificar muitos atributos de uma só vez.

Abaixo há uma lista com alguns cenários criados pelo Physimulation. Todos estão disponíveis no código fonte do projeto.

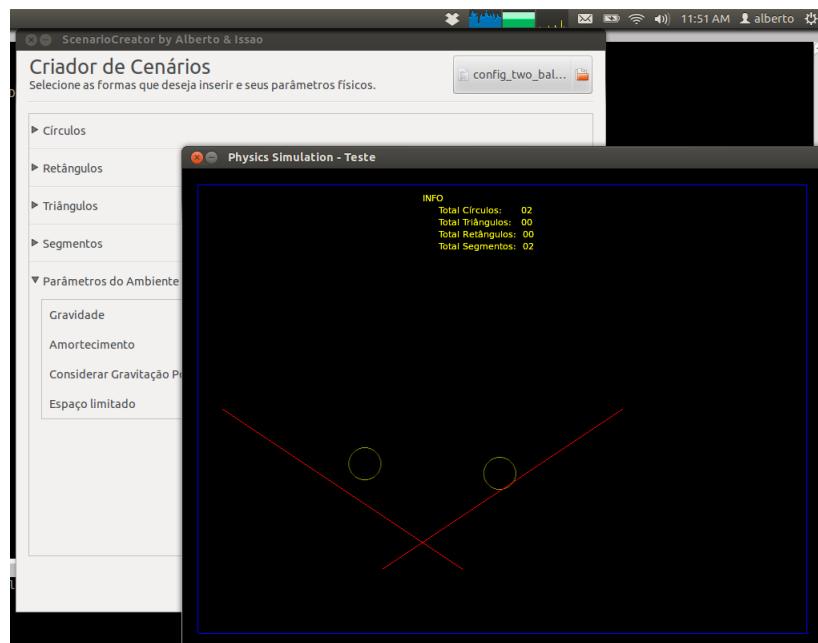


Figura 24: Cenário 1

No cenário 1, dois segmentos são utilizados como rampa para duas bolas. Embora pareçam idênticas, a rampa da esquerda tem um coeficiente de elasticidade superior ao da direita, o que faz uma das bolas quicar para outra rampa. Já no cenário 2, abaixo, é possível observar as quatro primitivas do Physimulation, sendo que para cada tipo há um objeto fixo e outro móvel.

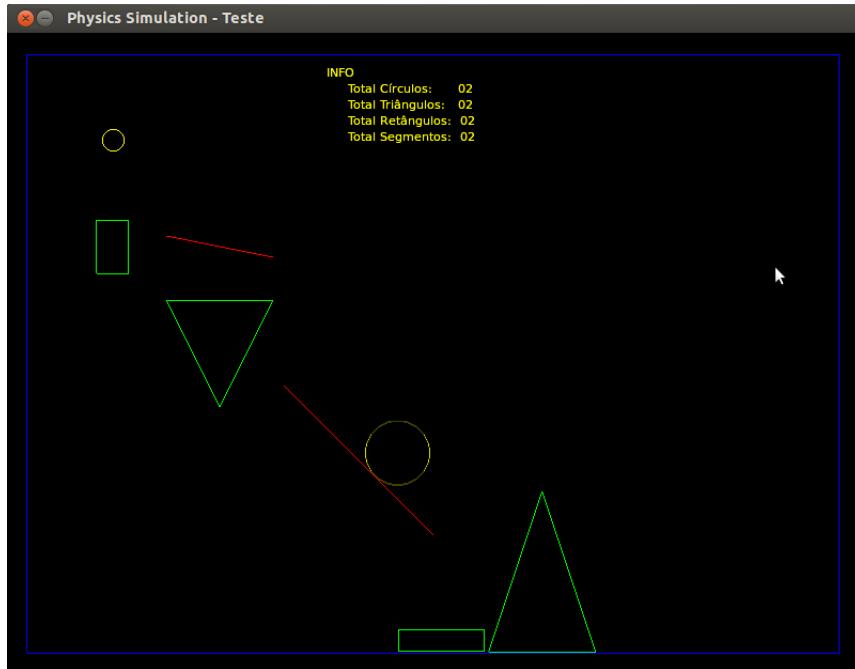
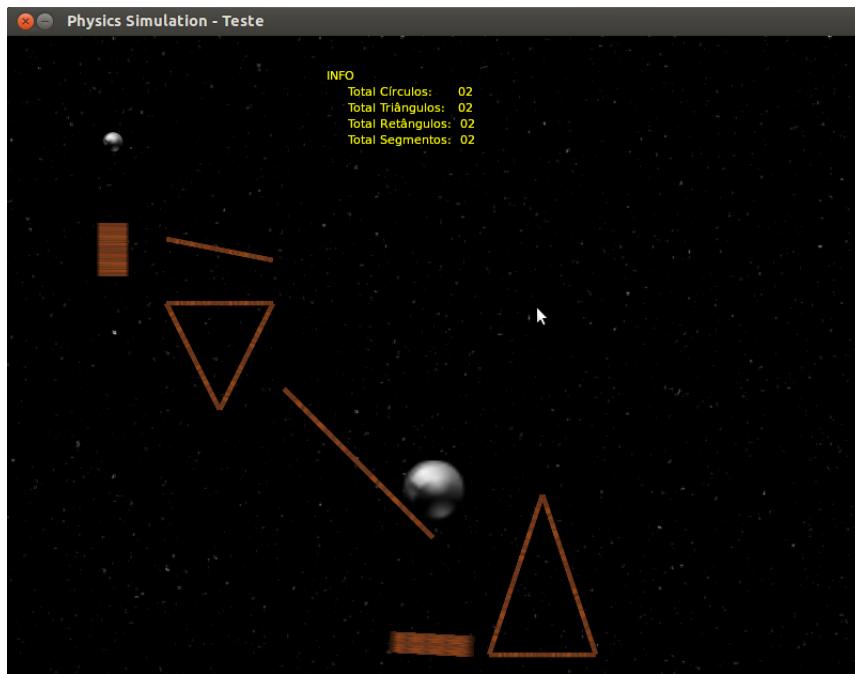


Figura 25: Cenário 2

Os cenários abaixo levam em conta a atração dos corpos gravitacionais entre si e são simulações de movimentos orbitais. Para criar animações deste tipo, nosso

orientador João Kerr nos sugeriu posicionar os corpos gravitacionais ao longo de um eixo imaginário; em seguida, adicionar aos corpos uma velocidade perpendicular a este eixo, variando o módulo conforme os resultados observados. Esta dica foi bastante útil para a construção destes cenários.

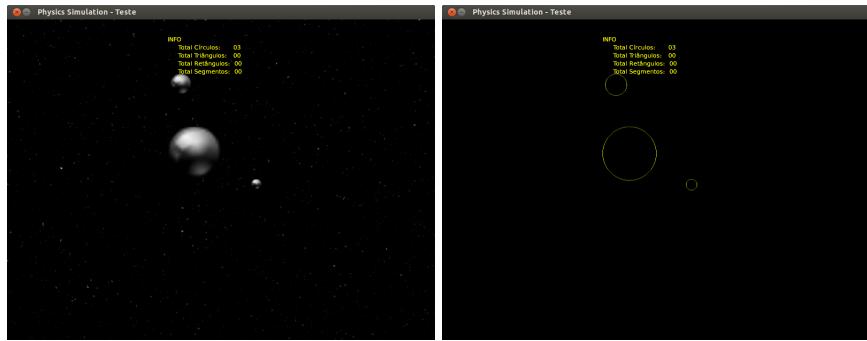


Figura 26: Cenário com Gravitação 1

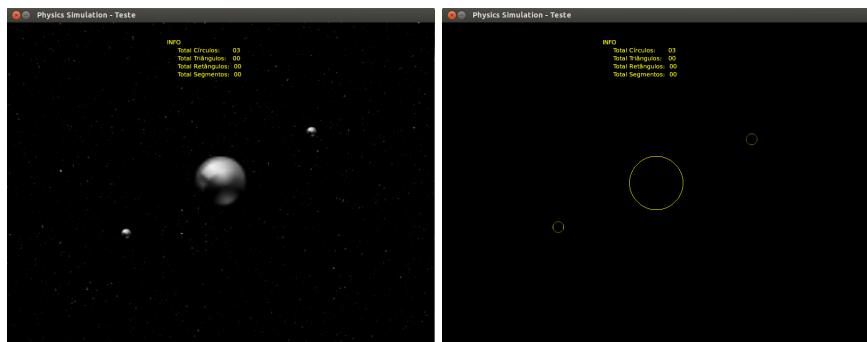


Figura 27: Cenário com Gravitação 2

No cenário a seguir há duas órbitas bem diferenciadas: das duas esferas menores em torno da maior e da esfera de menor tamanho em torno da esfera de tamanho intermediário.

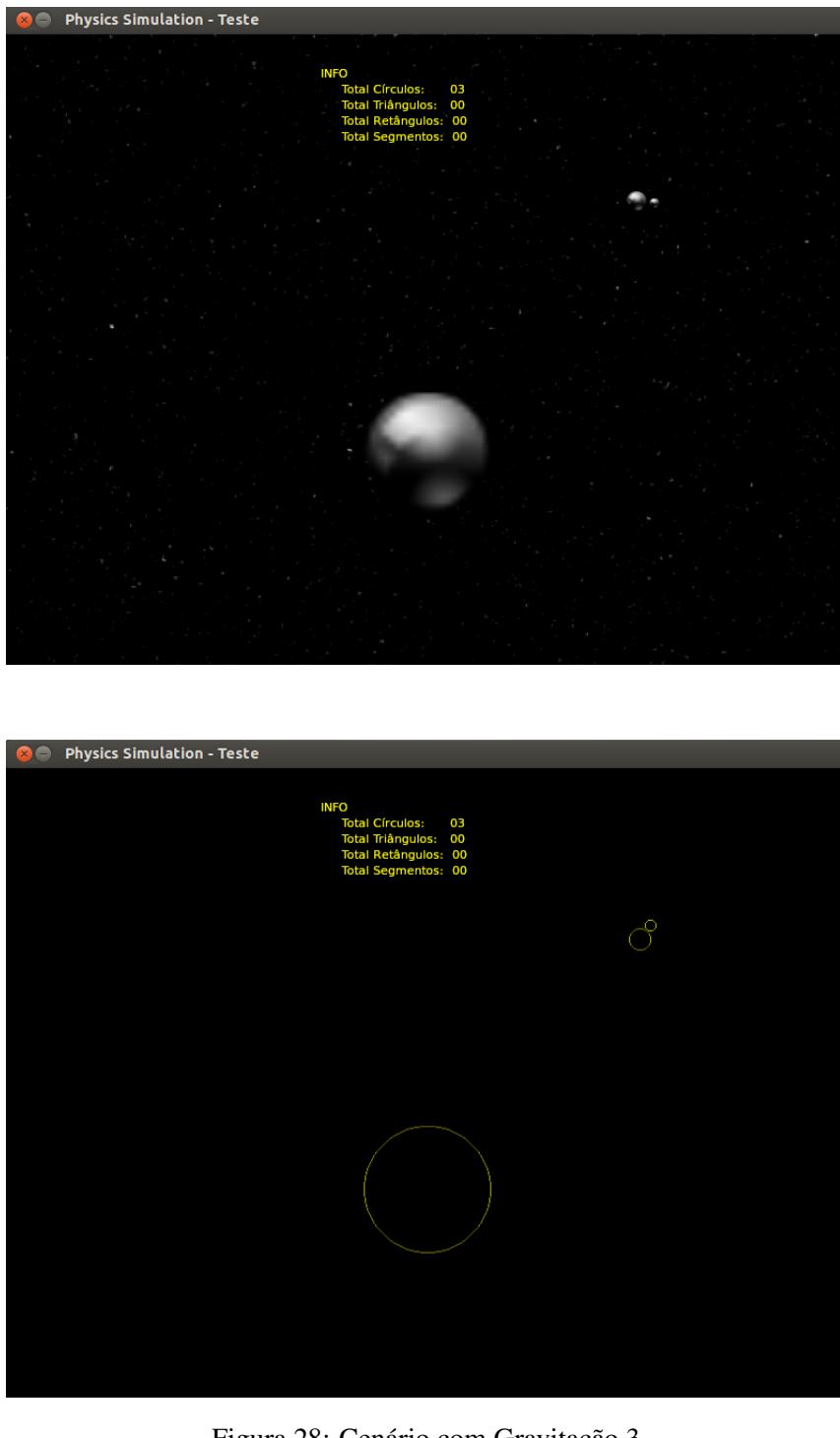


Figura 28: Cenário com Gravitação 3

7.4 Arquivos de configuração

As propriedades físicas do cenário e de cada objeto ficam guardadas em um arquivo `*.rb` que pode ser modificado diretamente, sem o uso do Physimulation. Observar e entender estes arquivos é importante para que o aluno do BCC entenda o funcionamento do programa como um todo.

O processo de animação do Physimulation divide-se basicamente em três etapas: 1) O preenchimento das propriedades físicas dos objetos no criador de cenários; 2) A geração do arquivo `config_gerado.rb`; 3) A leitura deste arquivo pela classe que irá executar de fato a simulação (`Test.rb`).

Ao alterar o arquivo de configuração manualmente e carregá-lo pelo Physimulation, o usuário estará partindo diretamente para a terceira etapa do processo, tornando-o mais rápido.

Para guardar no sistema um arquivo de configuração, basta copiar o arquivo gerado para outro com um nome diferente, para que o arquivo não seja sobreescrito. Por exemplo, para guardar a última configuração utilizada em um arquivo de nome "meu_config.rb", o usuário pode executar o seguinte comando:

```
$ cd <<DIR_PHYSIMULATION>>/Simulation/testes/config  
$ cp config_gerado.rb meu_config.rb
```

Abaixo um exemplo de arquivo de configuração:

Código 14: Exemplo de arquivo config.rb

```
module TestObjectConfig

  class Space
    attr_accessor :gravity, :damping,
                  :limited_space, :object_gravity

    def initialize
      @gravity = vec2(0, 100)
      @damping = 1.0
      @limited_space = true
      @object_gravity = false
    end

  end

end
```

```

Circles = [ {
    :mass => 100.0,
    :radius => 20,
    :factor_x => 2.0,
    :factor_y => 2.0,
    :x => 200,
    :y => 100,
    :v => vec2(0.0, 0.0),
    :moment_inertia => 100000,
    :elasticity => 1.0,
    :friction => 1.0,
    :zorder => 100,
    :collision_type => :undefined0,
    :image_name => 'cannonball2.png',
    :static => false
} ]

Segments = [ {
    :x => 200,
    :y => 400,
    :thickness => 1,
    :angle => 0.0,
    :vectors => [vec2(-150, -100),
                  vec2(150, 100)],
    :elasticity => 1.0,
    :friction => 0.0,
    :zorder => 100,
    :collision_type => :undefined0,
    :image_name => 'catapult-b.png',
    :static => true
} ]

Rectangles = []

Triangles = []

end

```

7.5 Pontos de modificação

O Physimulation possui pontos do código que podem ser modificados de acordo com o comportamento físico desejado, o que pode ser interessante para o aluno de computação.

Além de modificar o próprio código do projeto, o aluno pode também aproveitar os recursos da linguagem Ruby e adicionar métodos a classe já existentes. Por exemplo, no diretório `Simulation/ext/` há o arquivo:

```
pause_unpause.rb.example.
```

Ele implementa um método para a classe `TesteWindow`, utilizada para executar as simulações. Caso o aluno altere o nome desse arquivo para

```
pause_unpause.rb
```

, o interpretador irá reconhecer o novo método e ele já poderá ser utilizado na execução das animações.

Neste arquivo de exemplo, há uma implementação da função `pause` para simulações, associada a tecla 'P'. Assim, renomeando este arquivo e executando qualquer uma das animações, o usuário poderá utilizar automaticamente a função `pause` simplesmente apertando a tecla 'P'.

Baseando-se neste exemplo, o aluno de computação pode criar outros pontos de modificação no Physimulation, quando julgar apropriado.

8 Introdução à computação com animações

Nesta seção explicamos as duas integrações que fizemos com exercícios-programas que envolviam física, dados em disciplinas de introdução a computação na Poli. Os enunciados estão disponíveis no diretório deste projeto e também podem ser encontrados nas referências desta monografia.

8.1 Configuração

Para exemplificar uma possível utilização do Physimulation, fizemos duas integrações com EP's. A primeira foi com o Angry Bixos, dado no primeiro semestre deste ano, uma simulação de lançamento de objetos semelhante ao jogo *Angry Birds*. A segunda integração foi com EP Apollo 13, dado no primeiro semestre de 2011.

Em ambos os exercícios, são lidos arquivos `.txt` de entrada para que seja iniciada a simulação. Novamente em ambos os casos, a animação é exibida no próprio terminal. Na figura 29 há um exemplo válido de saída do EP Angry Bixos.

Nas duas integrações, utilizamos o Physimulation para:

1. Ler um arquivo no mesmo formato que o arquivo de entrada do EP;
2. Gerar um arquivo `config.rb` com as condições iniciais fornecidas no arquivo de entrada;
3. Mostrar uma animação do resultado obtido.

Por motivo de simulação, alguns dos valores de entrada foram ajustados para fazerem sentido no Physimulation.

8.2 Angry Bixos

O formato de arquivo de entrada a seguir foi extraído do próprio enunciado do EP Angry Bixos.

- yE , $vmax$: 2 reais que definem a posição do estilingue e a velocidade máxima com que um bixo pode ser arremessado;
- yA , hA : 2 reais que definem a posição do alvo e sua altura;
- $dist$: real positivo que define a distância $xA - xE$ entre o alvo e o estilingue.

Figura 29: Exemplo de saída do EP Angry Bixos para Poli

- nBix: inteiro que define o número de bixos que podem ser lançados;
- nLin, nCol: 2 inteiros que definem as dimensões do gráfico a ser impresso (ambos devem ser múltiplos de 5);
- nUni: real utilizado como fator de escala das alturas (número de unidades de altura por linha);
- g: real negativo que define a aceleração da gravidade.

Após ler o arquivo de entrada, o Physimulation pede ao usuário que informe a intensidade do lançamento e o ângulo que o Bixo deverá ser lançado, assim como no EP. Esse processo é repetido até que acabe o número total de lançamentos, definido no arquivo de entrada. Caso o usuário acerte o alvo, a simulação é encerrada. Tanto no caso de sucesso quanto no caso de falha, são exibidas mensagens informando o usuário do resultado. Para executar a integração:

```
$ cd <<DIR_PHYSIMULATION>>/Simulation/integracao-ep/angry-bixos
$ ruby integracao-angry-bixos.rb
```

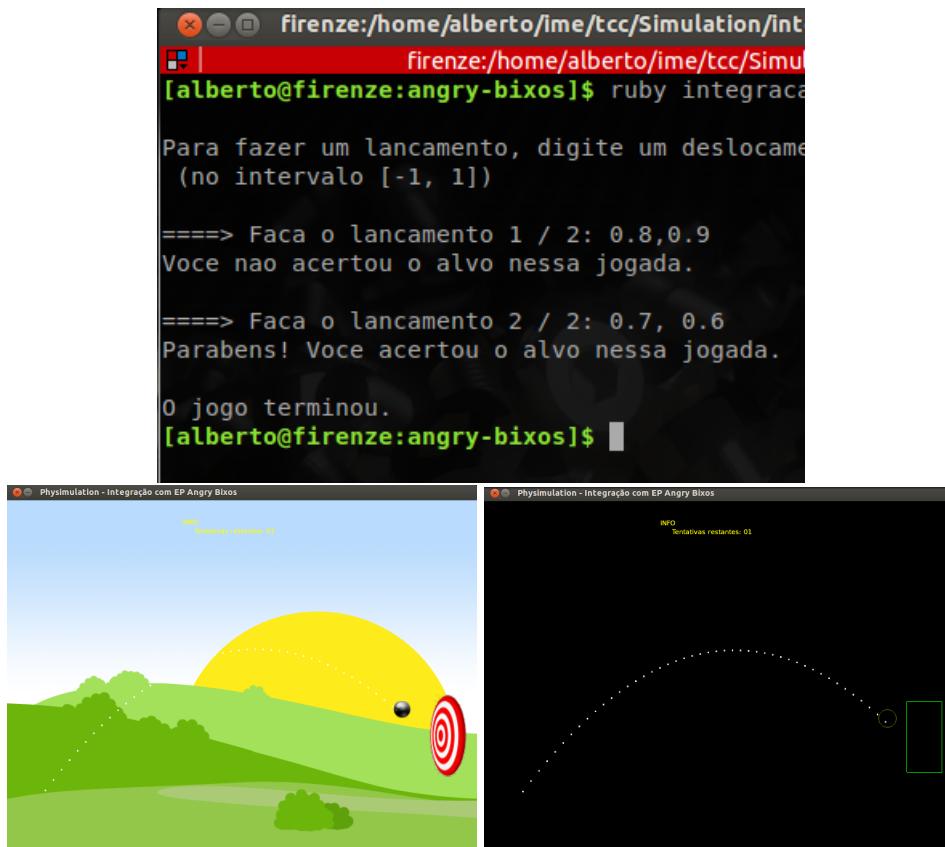


Figura 30: Integração com EP Angry Bixos

8.3 Apolo

O formato de arquivo de entrada a seguir também foi extraído do próprio enunciado do EP Angry Bixos.

- z1) posição inicial de uma nave em coordenadas cartesianas;
- z2) as componentes (V_X, V_Y) do vetor velocidade inicial da nave;
- z3) tempo máximo de simulação;
- z4) o intervalo dT entre um instante e o instante seguinte da simulação.

No caso do EP Apollo 13, integramos apenas com uma parte do exercício, em que é solicitado ao aluno mostrar a animação do movimento gravitacional no terminal.

Por padrão, o Physimulation irá executar a animação do arquivo `entrada.txt`, que corresponde à trajetória livre de retorno (efeito *slingshot*). Para executar a simulação:

```
$ cd <<DIR_PHYSIMULATION>>/Simulation/integracao-ep/apolo  
$ ruby integracao-apolo.rb
```

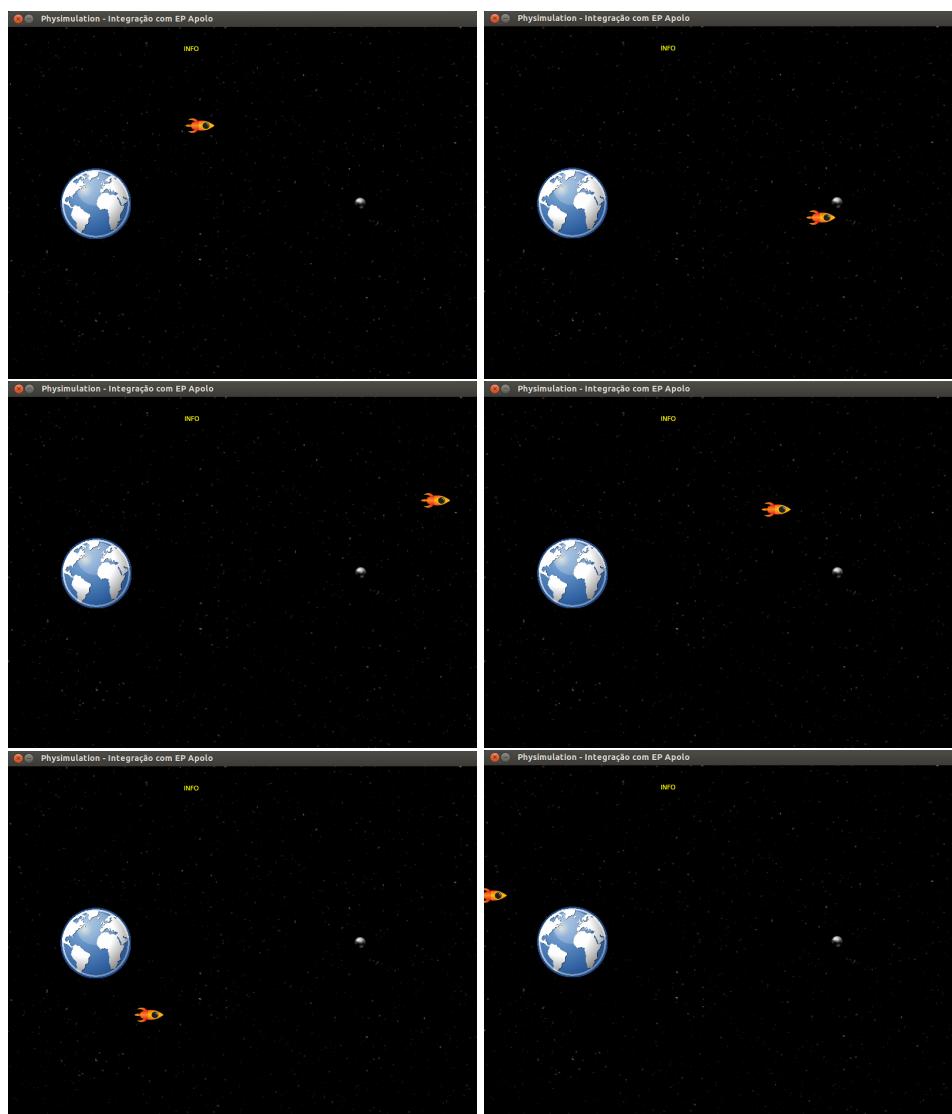


Figura 31: Efeito *slingshot* Integração com EP Apolo

9 Conclusão

Com o desenvolvimento do Physimulation e as integrações com exercícios-programas passamos por um grande aprendizado nas áreas de simulação e animação, programação - principalmente por utilizarmos Ruby, uma linguagem nova para nós - e física. Nossa intenção é que boa parte deste aprendizado seja repassado para os alunos de física e computação em geral, assim como a motivação para o estudo de disciplinas teóricas do IME.

Para os alunos interessados em discretização de simulações, seja para trabalhos acadêmicos ou para jogos que envolvam algum tipo de simulação física, recomendamos a leitura da seção 3, em que mostramos um pouco do que aprendemos em relação a tempo de simulação. Dependendo do objetivo do aluno, este tempo deverá ser fixo ou variável. Além disso, há a questão da simulação ser ou não em tempo real: há casos em que é mais importante ter uma animação com resultados precisos e realistas do que uma resposta imediata, mesmo que o processamento demande horas de cálculos.

O estudo de alguns conceitos e algoritmos de geometria computacional para detecção e tratamento de colisões também foi bastante interessante. A seção 4 possui referências para o aluno que desejar um conhecimento mais profundo desta área, que é inclusive uma das disciplinas eletivas oferecidas pelo IME atualmente.

Como visto na seção 7, o código do Physimulation foi construído com uma preocupação constante: ser fácil de entender e de modificar. Esperamos que com isso os alunos se sintam mais à vontade para utilizar nosso trabalho ou mesmo alterá-lo, dando continuidade ao projeto. Alguns trabalhos futuros estão descritos na parte subjetiva desta monografia.

Assim, esperamos que o Physimulation seja de fato uma contribuição, mesmo que pequena, à questão de contextualização das disciplinas do IME. Como foi dito na apresentação do trabalho, nossa expectativa é que esta iniciativa dê resultados a médio e longo prazo, tendo como horizonte a utilização do Physimulation em salas de aula pelo próprio professor e em conjunto com exercícios teóricos e EP's.

Referências

10 Instruções de instalação da plataforma

Passos para instalação das bibliotecas: ruby, chipmunk, gosu, chingu, etc.

Falar do warning do próprio chipmunk: gems/ruby-1.9.3-p286/gems/chipmunk-5.3.4.5/lib/chipmunk.rb:6: Use RbConfig instead of obsolete and deprecated Config

(Talvez seja um apêndice)

11 Apêndice

O texto a seguir foi extraído do site: <http://bcc.ime.usp.br>.

Material de Apoio ao BCC

A falta de contextualização das disciplinas básicas do BCC tem sido uma queixa recorrente dos alunos nas reuniões entre alunos e professores, no Encontro do BCC de 2010 e também no processo de avaliação semestral que é realizado pelo orientador pedagógico orientador pedagógico da Escola Politécnica (POLI), Giuliano Salcas Olguin.

A fim de motivar os alunos e ilustrar a relação entre ciência da computação e as disciplinas básicas de álgebra, cálculo, estatística, probabilidade e física presentes no currículo do BCC a CoC sugeriu que fossem produzidos documentos ilustrando aplicação de cada uma dessas disciplinas em ciência da computação e vice-versa. Esses documentos têm o objetivo de motivar os alunos do BCC:

1. ilustrando as relações entre as disciplinas básicas do curso e ciência da computação;
2. mostrando aos alunos quais das disciplinas mais avançadas do BCC que fazem uso dos conteúdos das disciplinas básicas;
3. fornecendo aos professores das disciplinas básicas do BCC exemplos de aplicações de suas especialidades em ciência da computação, que, eventualmente, podem ser mencionados em aulas ou ser temas de trabalhos.

Esses documentos poderão também ser usados pelas disciplinas de Introdução à Ciência da Computação que são oferecidas pelo DCC para várias unidades da USP. Nestas disciplinas, frequentemente, os chamados exercícios programas ilustram aplicações de métodos computacionais na solução de problemas em genética, física, economia, etc. Por exemplo, na última edição da disciplina

MAC2166 Introdução à Ciência da Computação para Engenharia

podemos ver um exercício programa em que é simulada a "trajetória livre de retorno" de uma nave sob a ação gravitacional da Terra e da Lua em <http://www.ime.usp.br/~mac2166/ep3/>. Já um exercício programa com aplicação em genética pode ser visto em <http://www.ime.usp.br/~mac2166/ep4/>.

Além de uma maior integração do curso este projeto pretende propor possíveis mudanças na grade curricular do BCC. Para isto pretendemos realizar uma pesquisa com o egresso do BCC e uma pesquisa das grades curriculares dos cursos de computação pelo mundo.

Parte II

Parte subjetiva

Nesta seção descreveremos a relação entre nosso projeto e a experiência adquirida no BCC.

Entregar este projeto como trabalho de formatura e disponibilizar seu código para os alunos do BCC foram duas das experiências mais gratificantes que já tivemos. Isto pois acreditamos que tal conteúdo poderá ser utilizado pelas próximas turmas do BCC como incentivo ao aprendizado da matéria de física. Além disso, tanto alunos do próprio Instituto de Física quanto da Engenharia Politécnica também poderão se interessar pelo conteúdo: o primeiro grupo (FIS) pela animação de fenômenos físicos estudados e o segundo (Poli) tanto pela animação quanto pela simulação de tais fenômenos.

Mas, ao mesmo tempo, por ser um trabalho que levou meses, certas dificuldades foram encontradas pelo caminho. Tivemos que tomar decisões às vezes frustrantes, porém necessárias.

11.1 Desafios e frustrações encontrados

Inicialmente, nossa motivação era entregar um sistema que utilize recursos do Nintendo Wii Remote TODO e que o professor pudesse utilizá-lo em sala de aula para realizar suas simulações e animações. Porém, chegamos a conclusão que esta tecnologia aumentaria consideravelmente o nível de complexidade de nosso trabalho e não tínhamos garantia de que utilizá-la acrescentaria da mesma forma ao resultado final. Assim descartamos esta possibilidade.

Como utilizamos algumas bibliotecas de terceiros em nosso projeto, tivemos que entender obrigatoriamente como eram feitas as principais chamadas de métodos destas bibliotecas, principalmente o Chipmunk e o Gosu. Um detalhe interessante que ocorreu no segundo mês de trabalho foi a necessidade de mudar o código da biblioteca e recompilá-la para que uma função simples de mensagem para o usuário funcionasse. Uma semana depois, utilizando uma versão mais nova da biblioteca, descobrimos que nossa alteração não era mais necessária, pois já havia sido feita pelos próprios programadores na mudança de versão.

Além disso, utilizamos um *binding* da versão original do Chipmunk. Isto tra-

zia duas dificuldades para nós: 1) o código original (em C++) sempre estava com uma versão mais recente e provia mais métodos; e 2) nem sempre o que víamos na documentação oficial possuía correspondente em nosso *binding*.

Por último, um desafio que tivemos foi encontrar um professor de física disponível para nos auxiliar na elaboração do protótipo do sistema. Ficamos muito felizes quando após algumas semanas o bacharel em física e aluno do BCC João Kerr veio a uma de nossas reuniões, a convite do professor Coelho.

11.2 Disciplinas mais relevantes - Alberto Ueda

- MAC0110 Introdução à Computação : Embora já tivesse contato com programação no ensino técnico, foi nesta disciplina que passei a conhecer e utilizar boas práticas de programação. Além disso, estudei algoritmos famosos e interessantes (por exemplo os de ordenação) que estimularam-me para as disciplinas que viriam a seguir.
- FAP0126 Física I : É a grande motivação deste trabalho. Os conceitos aprendidos nesta disciplina estão por todo nosso código e nas simulações produzidas. Com o Physimulation, tentamos unir o que vimos nesta disciplina com a computação.
- MAC0122 Princípios de Desenvolvimento de Algoritmos : O maior contato com algoritmos, dos mais simples e elegantes aos mais complexos, foi fundamental para minha formação. Primeiro porque me desafiou em certos momentos - e consequentemente me deu coragem para analisar ou implementar futuros algoritmos - e em segundo lugar pois deu-me a confiança de que gostaria de seguir carreira em computação.
- MAC0211 Laboratório de Programação I : Esta disciplina foi interessante por dois motivos: pelo estímulo ao trabalho em equipe e por nos apresentar conceitos e ferramentas relacionadas a qualidade de software, como o Doxygen para documentação de código. Foi nesta disciplina que aprendi o que era um Makefile!
- MAC0323 Estruturas de Dados : Essencial para minha formação como cientista da computação. As estruturas aprendidas nesta disciplina - como listas ligadas e árvores - são muito comuns na programação, mesmo no mercado. Possuem vantagens e desvantagens entre si e o conhecimento de suas propriedades assim como os algoritmos adequados para manipulá-las foram muito importantes para mim.

- MAC0420 Introdução a Computação Gráfica : Outro forte motivador para nosso trabalho. Nesta disciplina tivemos como exercício-programa a simulação de um jogo de bilhar em três dimensões. Foi uma das experiências mais gratificantes do meu BCC, pois minha dupla e eu aplicamos física em um código simples em C com algumas bibliotecas gráficas e de repente tínhamos uma simulação razoável do que ocorre na vida real. Foi quando percebi que com poucos conceitos de física podíamos reproduzir muitos fenômenos naturais, como colisões e dissipação de energia. Percebi também o quanto estes resultados me motivavam a estudar mais, tanto física quanto computação.
- FAP0137 Física II : Os tópicos desta disciplina não foram o foco deste projeto, mas foram grandes motivadores para nosso trabalho. Assim como em Física I, houve pouca contextualização do que foi estudado com o curso de computação. No futuro, temas como relatividade restrita poderão se tornar bem mais simples de se entender por meio de animações criadas pelo próprio usuário, utilizando nosso simulador.
- MAC0332 Engenharia de Software : na área de computação, um dos conceitos mais recorrentes em qualquer projeto de longo prazo é ciclo de vida de um *software*. Este era o tópico mais discutido na disciplina, tornando-a fundamental para o aluno de computação. Outro aspecto a destacar é a prática de trabalho em equipe.
- MAC0338 Análise de Algoritmos: difícil descrever em poucas linhas o quanto esta disciplina é importante para o aluno de computação. Além do desempenho ser uma preocupação constante e necessária a qualquer programador, o aprendizado nesta disciplina é uma das minhas bases sólidas como desenvolvedor. É uma das matérias que quero aprofundar meus conhecimentos durante o mestrado.
- MAC0446 Princípios de Interação Homem-computador : uma das disciplinas mais legais para o aluno que está preocupado com a usabilidade de seu sistema. Os conceitos aprendidos estão por todo o trabalho, assim como em outros projetos que participei ou fui responsável.

11.3 Disciplinas mais relevantes - Rafael Issao Miyagawa

- MAC0110 Introdução à Computação: Sem dúvida uma das disciplinas que foram importantes para mim. Quando cursei esta disciplina, ela foi ministrada com a linguagem Java que até então não tinha muita conhecimento

sobre linguagens orientado a objetos. Foi uma introdução à computação e a linguagem orientada a objetos.

- MAC0211 e MAC0242 Laboratório de Programação: Foi a disciplina que tivemos contato com ferramentas de desenvolvimento e trabalho em equipe. A importância de um código bem documentado, controle de versões para o projeto e documentação foram os tópicos que foram importantes para a minha formação como programador.
- Todas as disciplinas voltadas para algoritmos e estrutura de dados: Para mim é a base para poder desenvolver qualquer programa bom e eficiente. Entender as idéias envolvidas nos algoritmos e nas estruturas de dados ajudou muito para desenvolver um código de qualidade com eficiência.
- MAE0228 Noções de Probabilidade e Processos Estocásticos: Foi a disciplina que ajudou a formar a idéia do Physimulation. Cursei esta disciplina com a Professora Kira e ela conseguiu chamar minha atenção quando vi uma aplicação das teorias estudadas em um algoritmo para detectar a paridade de um número.
- MAC0332 Engenharia de Software: Foi o meu primeiro contato sobre processos de desenvolvimento. Realizar análises e estudos antes de começar a realmente escrever o código fonte foi uma tarefa difícil mas que vale muito a pena. Deveríamos ter mais disciplinas obrigatórias como esta.
- MAC0413 Tópicos de Programação Orientado à Objetos: Uma disciplina indispensável para desenvolver código de qualidade utilizando linguagem orientado à objetos.

11.4 Estudos e trabalhos futuros

Sem dúvida os tópicos de estudo mais importantes para a continuação deste trabalho são as disciplinas de Física I e II para o BCC. Quanto maior o conhecimento das leis e forças físicas presentes no mundo real, melhor serão as simulações e consequentemente as animações geradas.

Em segundo lugar, seria interessante uma análise de qual das alternativas a seguir tem uma melhor relação custo-benefício, visando a atualização do projeto com a versão mais nova do Chipmunk: A) migrar nosso projeto de Ruby para C++ e usar diretamente a versão original do Chipunk, sem *bindings*; ou B) atualizar o *binding* em Ruby adicionando os métodos e funcionalidades da versão mais recente em C++.

Também seria interessante dar continuidade ao projeto adicionando mais primitivas ao criador de cenários, como molas, pêndulos, segmentos móveis e polígonos de n lados.

Por último, mas não menos importante, um estudo de paradigmas que proporcionem mais usabilidade ao usuário, substituindo o preenchimento obrigatório de formulários para criação de objetos físicos. Ex: *drag-and-drop* do mouse para "arrastar" as formas geométricas, fornecendo os valores de massa, coeficientes de elasticidade e atrito *a posteriori* (após o objeto já estar na tela).

11.5 Desafios de simulação