GUIA PRÁTICO DE PROGRAMAÇÃO DE SIMULAÇÕES PHET PARA O ENSINO DE CIÊNCIA E MATEMÁTICA
- básico e avançado -

Rafael João Ribeiro
Sani de Carvalho Rutz da Silva

www.fisicagames.com.br

PPGECT-UTFPR
Ponta Grossa – 2017

SUMÁRIO

GUIA PRÁTICO DE PROGRAMAÇÃO DE SIMULAÇÕES PHET PARA O ENSINO DE CIÊNCIA E MATEMÁTICA

PARTE I – PROGRAMAÇÃO BÁSICA

Este conteúdo é direcionado para professores, pesquisadores e interessados em criar simulações interativas para o ensino de Ciência e Matemática, que já possuam conhecimento de linguagem de programação Javascript.

Em caso de dúvidas, contate os autores do portal Física Games:
Prof. Me. Rafael João Ribeiro - rafael.ribeiro@ifpr.edu.br
Profa. Dra. Sani de Carvalho Rutz da Silva.

Introdução

O portal Física Games (www.fisicagames.com.br) foi criado para incentivar a divulgação e produção de novas simulações interativas para ensino de Ciência e Matemática. Este guia prático é uma introdução à programação de simulações interativas utilizando recursos de código aberto do projeto PhET (WIEMAN *et al.*, 2010). Este material foi desenvolvido em pesquisa de doutorado realizada no Programa de Pós-Graduação em Ensino de Ciência e Tecnologia da Universidade Federal do Paraná.

Para compreender melhor o uso das simulações PhET em sala de aula, recomenda-se a leitura da tese de doutorado Ribeiro (2017) que deu origem a este guia prático.

Para conhecer o projeto PhET, acesse: https://phet.colorado.edu/.

Informações sobre as licenças de uso das simulações do projeto PhET acesse: https://phet.colorado.edu/en/about/source-code.

PRÉ-REQUISITO: O conhecimento de lógica de programação é necessário para o desenvolvimento das simulações. As simulações do projeto PhET são programadas em Javascript (ZAKAS, 2011) em sistemas de módulos, ou seja, utiliza vários arquivos separados com funções específicas. Esses módulos são programados visando ao fácil reúso em novos projetos. Os módulos, ou *scripts*, são carregados de forma modular assíncrona (AMD) utilizando RequireJS (FRANKO, 2013). Assim, uma simulação consiste na combinação de vários pequenos módulos já existentes e em outros novos para formar um produto final.

CÓDIGO EM JAVASCRIPT: Exemplos básicos de códigos em Javascript utilizados nas simulações PhET podem ser acessados, para uma ideia prévia de como funciona a programação, no *link*:
http://phetsims.github.io/scenery/doc/a-tour-of-scenery.html

Os recursos de código aberto do projeto PhET estão disponíveis em um repositório na Internet de fácil acesso, para visualizar todo o conteúdo das simulações em HTML5 acesse o *link*: https://github.com/phetsims.

PRINCIPAL REFERÊNCIA DESSE GUIA: Os primeiros passos desse guia seguem as orientações dos próprios desenvolvedores do projeto PhET, conforme o documento *PhET Development Overview*, disponível em: https://phet.colorado.edu/pt/about/source-code

FÓRUM DE DISCUSSÃO: As dúvidas mais técnicas podem ser dirigidas diretamente para os desenvolvedores e demais colaboradores do projeto PhET no grupo: Developing Interactive Simulations in HTML5. Exemplo de discussão iniciada pelo autor desse guia: link.

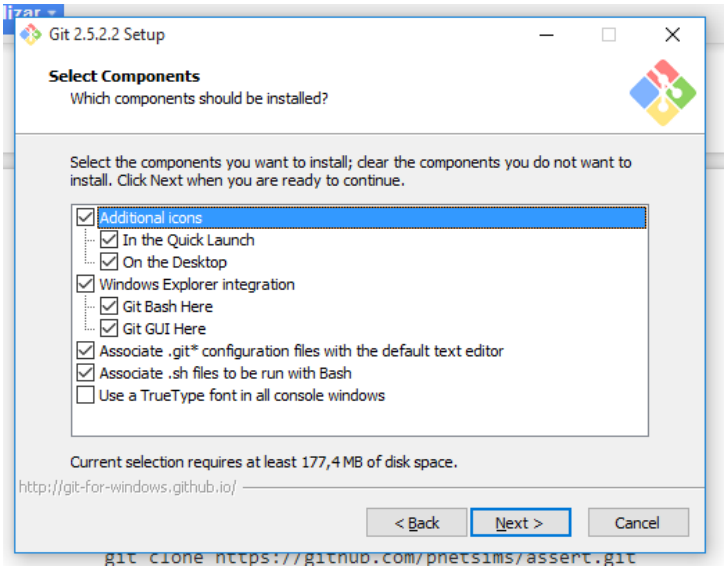
2. Preparação do ambiente de desenvolvimento

2.1 INSTALAÇÃO DO GIT

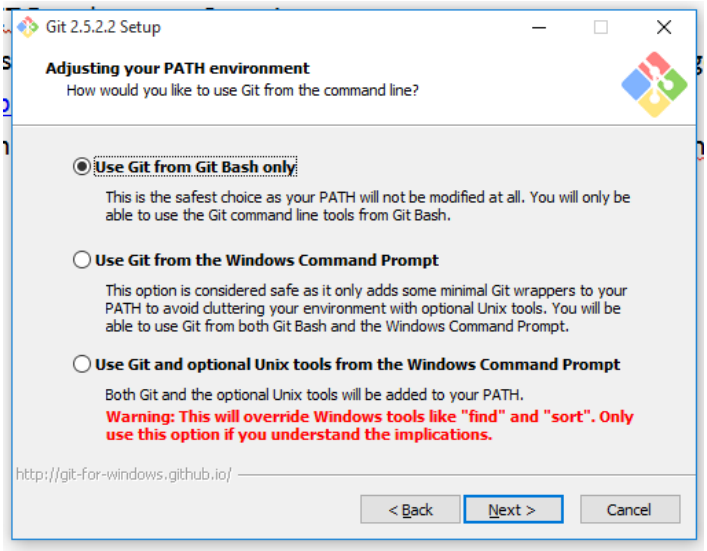
As simulações PhET estão hospedadas em repositório *online* GitHub (PIPINELLIS, 2015), que permite a clonagem dos códigos e a criação de novas versões. Porém, para testes de aprendizagem, é melhor utilizar cópias locais e não *online*. O *download* de cópias locais das simulações é feito utilizando o *software* GIT, que faz todo o trabalho de baixar os arquivos principais de uma simulação, e os demais arquivos necessários para a sua utilização de forma automática e rápida.

LINK PARA *DOWNLOAD* DO GIT: https://git-scm.com/downloads

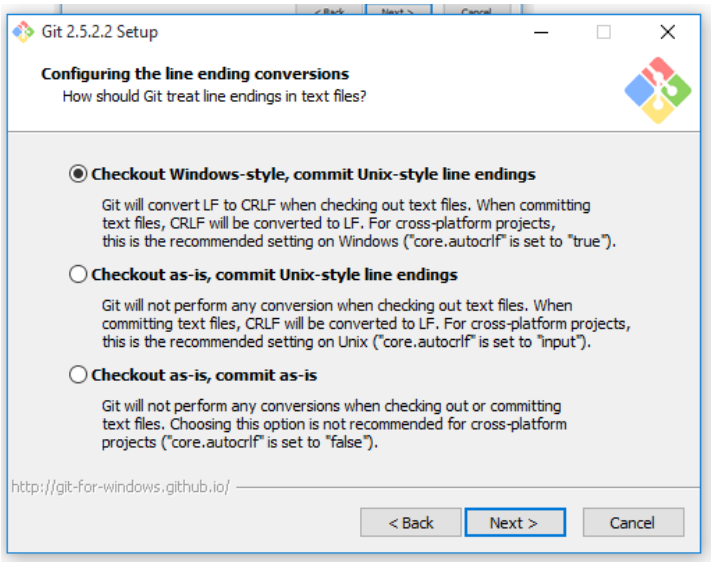
Durante a instalação do git, é aconselhável deixar marcada a opção: Additional Icons – On the desktop.



Na segunda tela de opções, marque a opção *Git Bash only*:



As demais telas de opções não precisam ser modificadas, basta seguir a instalação padrão.



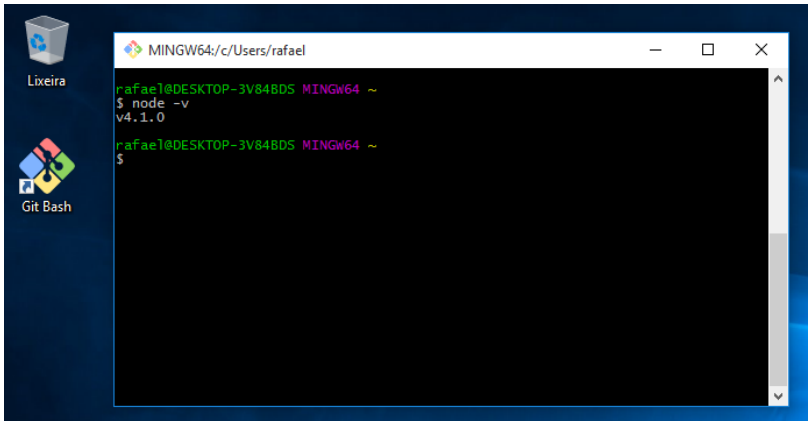
2.2 INSTALAÇÃO DO NPM do NODE.JS

As simulações PhET utilizam o gerenciador de pacotes NPM para fazer o *download* de bibliotecas e dependência externas necessárias para cada simulação. A instalação do gerenciador NPM é feita junto com a instalação do programa Node.js. Em um primeiro momento, não é necessário entender o funcionamento de aplicações Node.js, sendo importante entender que o NPM faz a busca e o *download* das bibliotecas externas de uso geral disponíveis no repositório NPMjs no endereço: https://www.npmjs.com.

Link para *download* do **Node.js,** o qual já possui o NPM: https://nodejs.org/en/download/.

Durante a instalação do Node.js, basta seguir as opções padrões.

TESTE DA INSTALAÇÃO DO NPM: O terminal de comandos Git Bash pode ser aberto por meio do ícone criado na tela de trabalho, conforme instalação anterior. Com o terminal aberto, a instalação do gerenciador NPM pode ser testada com comando de verificação de versão da instalação: **node –v.**



2.3 INSTALAÇÃO DO SERVIDOR LOCAL

A primeira aplicação a ser instalada com o NPM é o **http-server**, um pequeno, mas poderoso, servidor local para permitir o teste das simulações em qualquer navegador instalado no computador de desenvolvimento. Para saber mais sobre o http-server, acesse: https://www.npmjs.com/package/http-server.

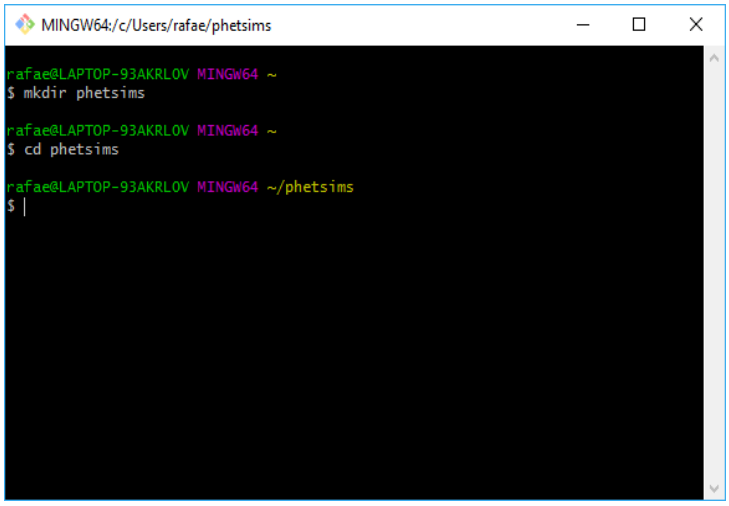
Comando para instalação do http-server: **npm install http-server -g.**

2.4 CRIAÇÃO DE UMA PASTA DE TRABALHO

Antes do processo de *download* das simulações, é necessário criar uma pasta de trabalho exclusiva para receber os arquivos.

Comando para criar uma pasta de trabalho: **mkdir phetsims.**

Comando para entrar na pasta de trabalho depois de criada: **cd phetsims.**

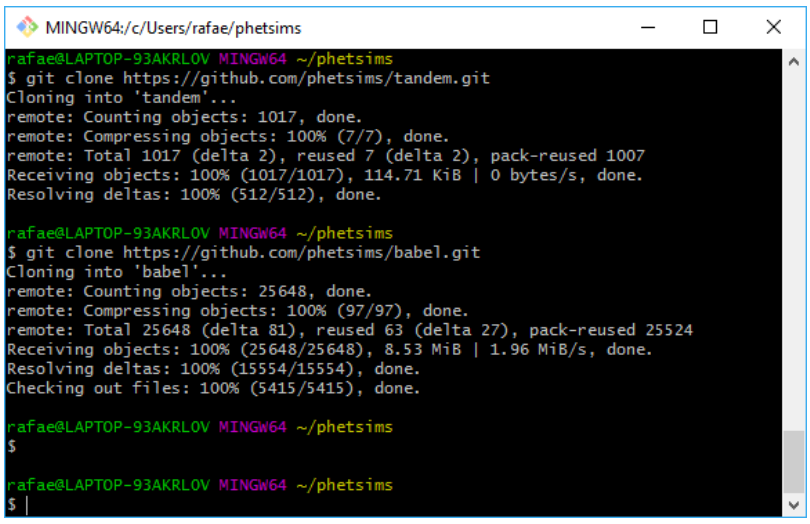


2.5 *DOWNLOAD* DOS ARQUIVOS PRINCIPAIS

O *download* dos arquivos das simulações presentes no repositório GitHub é feito utilizando o comando **git clone**, que cria clones dos arquivos na pasta de trabalho. Antes de executar o comando, é importante estar dentro da pasta de trabalho criada no passo anterior.

Para utilizar os comandos, basta copiar todas as linhas e colar utilizando o botão direito do *mouse* na tela do terminar GitBash e dar "enter", pode ser necessário mais um "enter" para rodar a última linha de comando. Esse processo pode demorar alguns minutos dependendo da velocidade de conexão com a Internet. A instalação é de, aproximadamente, 140 Mb.

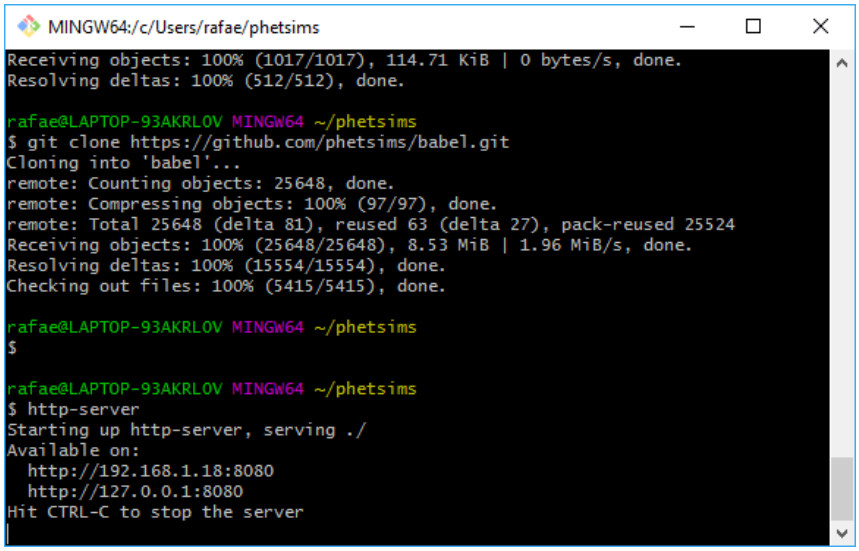
```
git clone https://github.com/phetsims/example-sim.git
git clone https://github.com/phetsims/assert.git
git clone https://github.com/phetsims/axon.git
git clone https://github.com/phetsims/brand.git
git clone https://github.com/phetsims/chipper.git
git clone https://github.com/phetsims/dot.git
git clone https://github.com/phetsims/joist.git
git clone https://github.com/phetsims/kite.git
git clone https://github.com/phetsims/phet-core.git
git clone https://github.com/phetsims/phetcommon.git
git clone https://github.com/phetsims/scenery.git
git clone https://github.com/phetsims/scenery-phet.git
git clone https://github.com/phetsims/sun.git
git clone https://github.com/phetsims/sherpa.git
git clone https://github.com/phetsims/tandem.git
git clone https://github.com/phetsims/babel.git
git clone https://github.com/phetsims/vegas.git
git clone https://github.com/phetsims/vibe.git
git clone https://github.com/phetsims/query-string-machine.git
```



Após este passo, o ambiente de desenvolvimento de simulações PhET já está pronto para a clonagem, modificação e criação de simulações interativas.

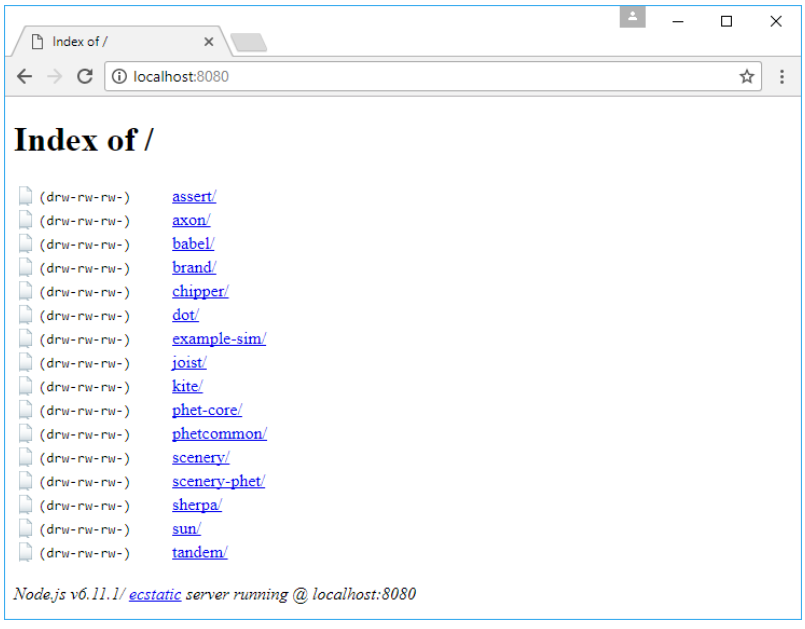
3. Teste em servidor local

Para testar uma simulação em servidor local, é necessário, primeiramente, iniciar o servidor com o comando: **http-server.**

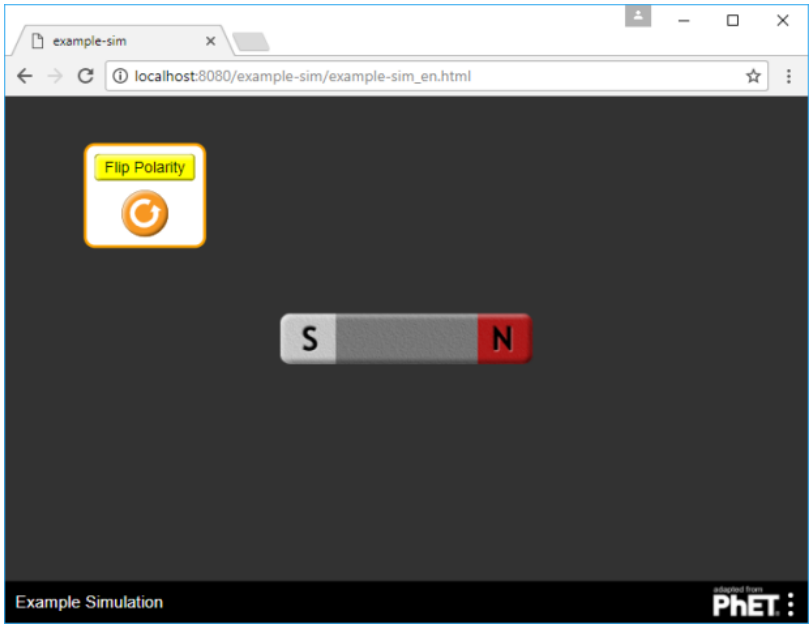


A combinação de teclas **ctrl+c** pode ser utilizada para interromper o servidor local.

Enquanto o http-server estiver sendo executado na tela do terminal, o servidor pode ser aberto em qualquer navegador pelo endereço eletrônico local: **http://localhost:8080/.**



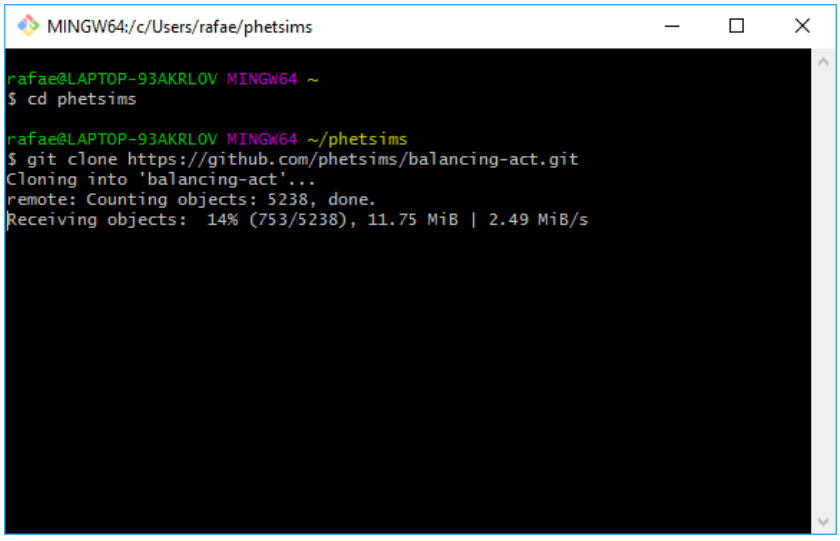
Para testar a simulação de exemplo, basta entrar na pasta example-sim e abrir o arquivo: **example-sim_en.html.**

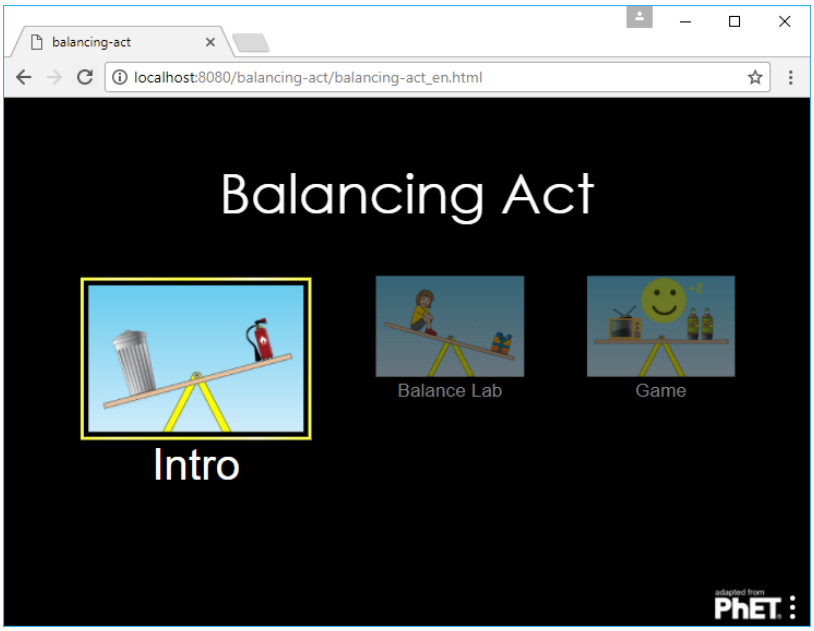


As simulações PhET são modificadas constantemente em tempo real no repositório GitHub, logo, é possível obter mensagens de erros na tela do terminal, principalmente referentes à ausência de arquivos. As mensagens de erros de falta de arquivos, geralmente, são resolvidas com o *download* de bibliotecas ausentes utilizando o comando git clone.

4. Clonagem de uma simulação específica

A clonagem de simulações é realizada com o comando git clone seguido do endereço da simulação no repositório GitHub. O comando deve ser executado na tela do terminal dentro da pasta de trabalho. Como exemplo, segue o comando para clonar a simulação Balançando:
git clone https://github.com/phetsims/balancing-act.git





EXECUTANDO A SIMULAÇÃO NO IDIOMA PORTUGUÊS.

As simulações são executadas por padrão no idioma Inglês, para rodar as simulações em outro idioma, é preciso adicionar parâmetros nas url das simulações. Dessa maneira, a simulação busca na pasta “babel” o arquivo correspondente ao idioma requisitado. Por exemplo, para abrir a simulação Balançando em Português, é necessário utilizar a seguinte url acompanhada do parâmetro *locale*:
http://localhost:8080/balancing-act/balancing-act_en.html?&locale=pt_BR

5 Procedimentos para modificação das simulações

O processo de modificação das simulações consiste em, primeiramente, estudar o conteúdo do arquivo html e, principalmente, dos códigos presentes nos arquivos javascript presentes dentro da pasta js de cada simulação. Os elementos gráficos das simulações são criados com o uso da biblioteca Scenery, que possui uma documentação própria disponível na Internet no *link*: http://phetsims.github.io/scenery/.

Os arquivos de programação .js são organizados dentro das pastas em duas categorias: elementos visuais; e rotinas lógicas de modelagem. Cada cena, em uma simulação, possui uma pasta própria com um nome representativo dentro da pasta js, Dentro da pasta da cena, geralmente, existem outras duas pastas: model e view, que possuem os elementos visuais e as rotinas lógicas de modelagem.

Os arquivos .html e .js podem ser abertos e modificados utilizando editores que facilitam a compressão dos códigos, como exemplo os *softwares*: Sublime Text e Notepad++. Após a modificação de qualquer código .js, basta atualizar (F5) a página no navegador para visualizar o efeito da modificação.

MODIFICAÇÃO DA SIMULAÇÃO EXAMPLE-SIM:

A simulação que vem como exemplo possui apenas uma única cena. Os seus arquivos estão no caminho \example-sim\js\example\view. Na pasta view, é possível encontrar o arquivo **ExampleScreenView.js** responsável pela adição e pelo posicionamento dos elementos visuais da cena:

Como exemplo, na linha 38, existe a função responsável por adicionar o painel de controle na cena:

```
this.addChild( new ControlPanel( model, {
  x: 50,
  y: 50
} ) );
```

As variáveis x e y representam a posição do painel de controle na cena. Como teste, é possível alterar o valor da posição em x de 50 para 300 e salvar o arquivo. Ao testar a simulação, será possível notar o deslocamento em x do painel de controle na cena.

6 Compilação das simulações

Para rodar as simulações sem a necessidade do servidor local http-server, é necessário compilar as simulações. O procedimento de compilação da simulação em arquivo único é realizado de forma automática, todos os arquivos da pasta da simulação e as bibliotecas externas de dependência são compiladas em um único arquivo .html, O arquivo único, resultante do processo de compilação, pode ser facilmente distribuído ou hospedado em um servidor na Internet.

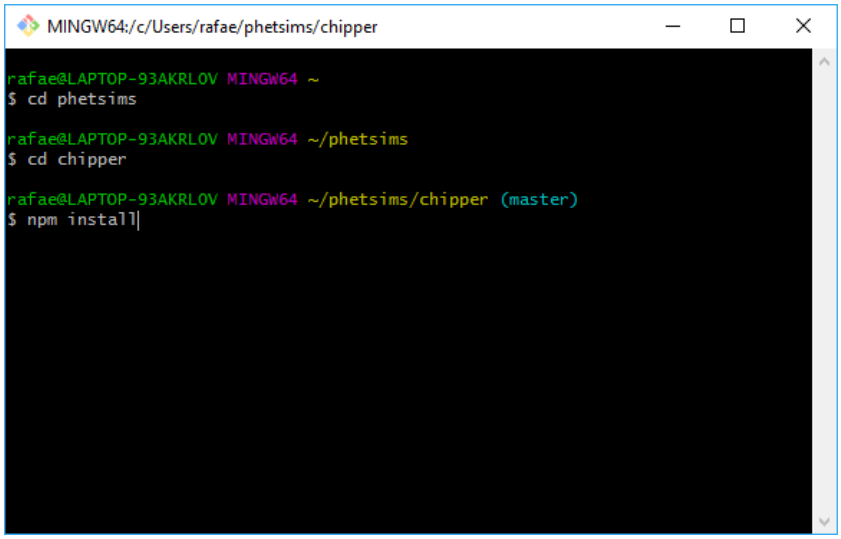
PREPARAÇÃO ANTES DO PROCEDIMENTO DE COMPILAÇÃO

A compilação das simulações é realizada com o apoio da biblioteca chipper, mas, antes de utilizá-la, é necessário instalar as suas dependências.

Comando para entrar na pasta chipper: **cd chipper** ou cd ../chipper.

Estando dentro da pasta chipper, executar o comando: npm install.

O comando npm install faz a leitura do conteúdo do arquivo package.json que possui uma lista com todas as dependências necessárias para que a biblioteca chipper funcione corretamente.



COMPILAÇÃO DE UMA SIMULAÇÃO

Para compilar a simulação Balançando, primeiramente, é necessário instalar as suas dependências externas executando o comando npm install dentro da pasta da simulação, da mesma maneira que foi feito para o chipper.

Comando para entrar na pasta da simulação: **cd balancing-act** ou **cd ../balancing-act.**

Estando dentro da pasta da simulação: **npm install.**

A compilação da simulação é feita utilizando a ferramenta de automação de tarefas Grunt.js, uma aplicação externa disponível no repositório NPMjs que deve ser instalada dentro da pasta da simulação com o comando: **npm install grunt-cli –g.**

O comando para iniciar as tarefas de compilação é: **grunt.**

Porém, para compilar a simulação com idioma Português como padrão, é necessário executar o comando com o seguinte parâmetro:

grunt --locales=pt_BR

Finalizado o procedimento de compilação, a simulação estará disponível dentro da pasta **build** na forma de arquivo único .html. Esse arquivo .html pode ser aberto com qualquer navegador instalado no computador, sem a necessidade do http-server. A compilação reúne todas as dependências e aplicações dentro de um único arquivo, não sendo mais necessário um servidor local para testar a simulação.

7 Atividade de aprendizagem

A simulação *exemplo-sim* é suficiente para testar o tour de códigos na documentação do projeto PhET, que possui códigos básicos de programação de elementos visuais. A documentação de ajuda do projeto PhET pode ser acessada no *link*: http://phetsims.github.io/scenery/doc/a-tour-of-scenery.html

Como proposta de aprendizagem, é interessante fazer o *tour* pelos códigos de programação, que aborda o básico para criação de caixas de texto, criação de figuras, animação e entrada de usuário. Todos esses códigos podem ser incorporados com finalidades de aprendizagem na simulação **example-sim**.

PARTE II – PROGRAMAÇÃO AVANÇADA

Introdução

Nesta segunda parte do guia, são apresentados os passos de programação para a adição de uma placar geral de pontuação nas simulações PhET. Esse processo requer conhecimentos mais avançados, como banco de dados MySQL (DELIDLE, 2010) e programação PHP, para poder fazer a integração da simulação com um banco de dados (SUEHRING; VALADE, 2013).

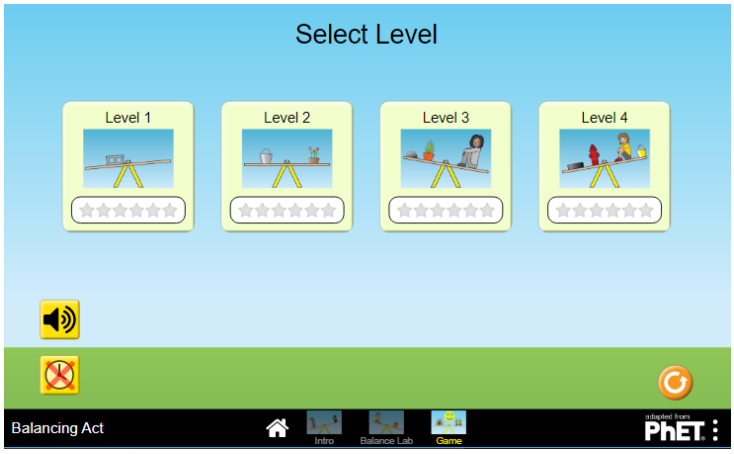
O placar de pontuação será criado na simulação Balançando, a qual já possui uma seção de jogo embutida que utiliza o módulo vegas.

Para conhecer as funções do módulo vegas, acesse o *link*: https://github.com/phetsims/vegas

Os pontos obtidos no jogo da simulação Balança serão associados à contagem de tempo na solução de cada nível de dificuldade do jogo. Assim, a pontuação será maior quanto maior a pontuação e menor o tempo para finalizar os desafios do jogo.

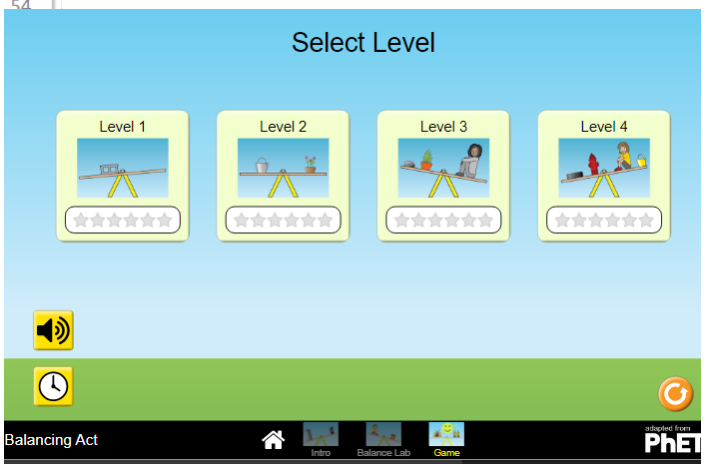
2. Contagem de tempo

A contagem de tempo no jogo da simulação Balança é desabilitado por padrão:



Para ativar a contagem de tempo no jogo, é preciso habilitar a variável timerEnabled presente no arquivo BalanceGameModel.js, que fica na pasta da simulação Balança:

```
36  var FULCRUM_HEIGHT = 0.85; // In meters.
37  var PLANK_HEIGHT = 0.75; // In meters.
38
39  function BalanceGameModel() {
40    var self = this;
41
42    this.soundEnabledProperty = new Property( true );
43    this.timerEnabledProperty = new Property( false );
44    this.levelProperty = new Property( 0 ); // Zero-based in the m
45    this.challengeIndexProperty = new Property( 0 );
46    this.scoreProperty = new Property( 0 );
47
48    // Valid values for gameState are 'choosingLevel', 'presenting
49    // 'showingIncorrectAnswerFeedbackTryAgain', 'showingIncorrect
50    // 'showingLevelResults'
51    this.gameStateProperty = new Property( 'choosingLevel' );
52    this.columnStateProperty = new Property( 'singleColumns' ); //
53    this.elapsedTimeProperty = new Property( 0 );
```

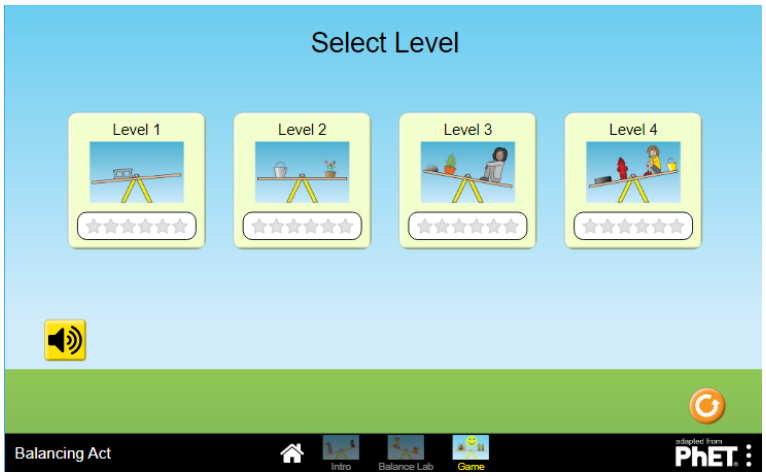


Também é necessário mudar o node do botão Timer para invisível, para impedir que o usuário desative a contagem de tempo. Os elementos visuais da

seção de jogo são programados no arquivo StartGameLevelNode.js, que fica na pasta: phetsims\balancing-act\js\game\view.

Para tornar invisível o botão de controle de tempo, é preciso adicionar no *javascript* a seguinte linha de comando: timerToggleButton.setVisible(false).

```
117
118     // The sound and timer buttons are positioned such that once is above the ground and the c
119     // wasn't space to put them both above or below.
120     var interButtonSpacing = 18;
121     soundToggleButton.left = options.controlsInset;
122     soundToggleButton.bottom = modelViewTransform.modelToViewY( 0 ) - interButtonSpacing / 2;
123     timerToggleButton.left = options.controlsInset;
124     timerToggleButton.top = soundToggleButton.bottom + interButtonSpacing;
125     timerToggleButton.setVisible(false);
126   }
127
128   balancingAct.register( 'StartGameLevelNode', StartGameLevelNode );
129
130   // Inherit from Node.
131   return inherit( Node, StartGameLevelNode );
132 } );
133
```



3. Adição de texto na cena do jogo

A programação que gerencia o sistema de pontuação no jogo é feita pelos códigos do módulo vegas. Então, a adição de um texto para mostrar a pontuação requer alterações no módulo vegas, sendo também necessário declarar novas variáveis no arquivo: LevelCompletedNode.js, que fica na pasta: \phetsims\vegas\js.

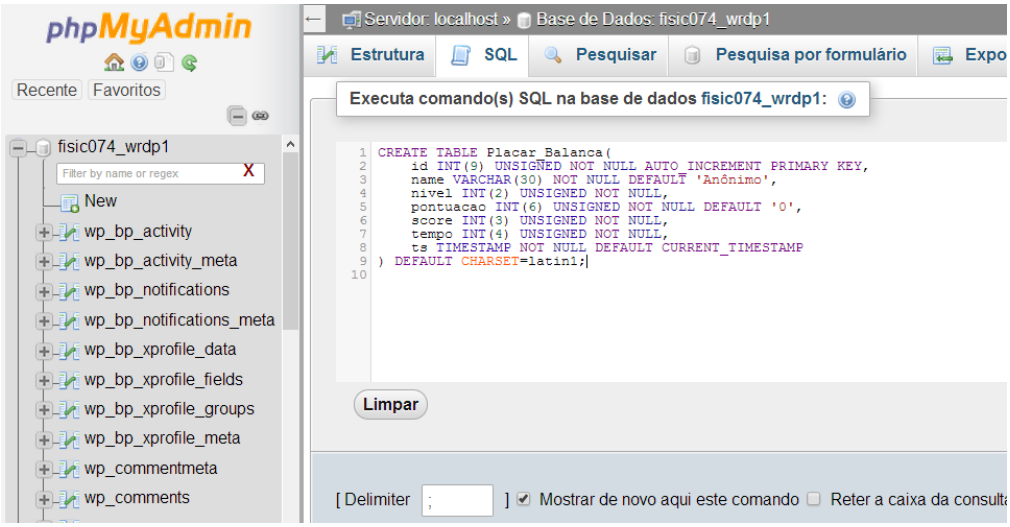
As dimensões da tela final de cada nível precisam ser adaptadas para mostrar a pontuação. Todas essas mudanças são mostradas no código completo no ANEXO, as linhas alteradas estão identificas com o comentário de marcação: //rafaeladd.

4. Criação de banco de dados

O registro das pontuações obtidas no jogo da simulação Balança pode ser feito com a utilização de um banco de dados, que requer o uso de um servidor com painel de gerenciamento de banco de dados, como o phpMyAdim.

No banco de dados, uma tabela deve ser criada utilizando programação SQL da seguinte maneira:

```
REATE TABLE Placar_Balanca(
    id INT(9) UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(30) NOT NULL DEFAULT 'Anônimo',
    nivel INT(2) UNSIGNED NOT NULL,
    pontuacao INT(6) UNSIGNED NOT NULL DEFAULT '0',
    score INT(3) UNSIGNED NOT NULL,
    tempo INT(4) UNSIGNED NOT NULL,
    ts TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
)
```



5. Script PHP para registro em banco de dados

O envio de dados da simulação para a tabela do bando de dados é feito com código PHP, o qual só funciona no servidor estando na mesma pasta da simulação. Porém, esse código em PHP pode ser testado via url, sem a simulação, para verificar a sua integração com a tabela MySQL.

Código do arquivo: placar_balanca_add.php

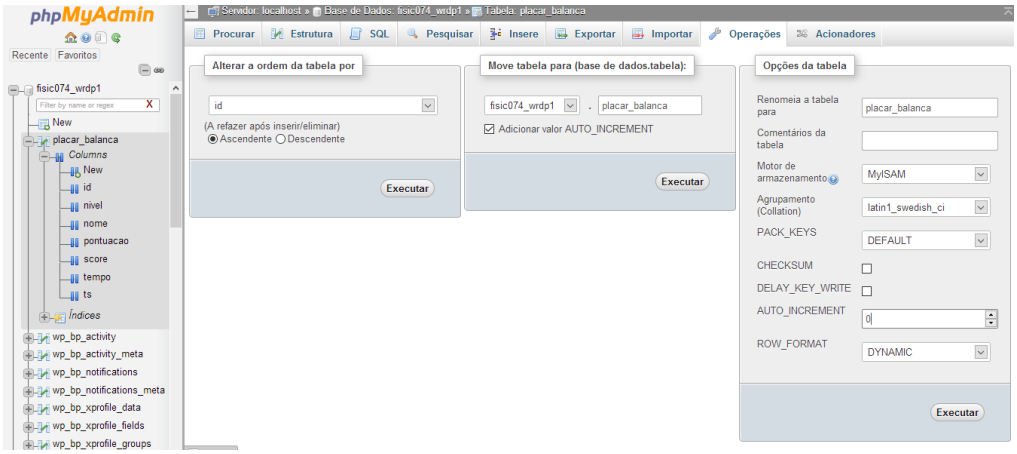
```
<?php
$servername = "localhost";
$username = "fisic074_placar";
$password = "xxxxxxxxxxxxx";
$dbname = "fisic074_wrdp1";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$nome = $_POST['enviarnome'];
$nivel = $_POST['nivel'];
$pontuacao = $_POST['pontuacao'];
$score = $_POST['score'];
```

```php
$tempo = $_POST['elapsedTime'];


$sql = "INSERT INTO placar_balanca (nome, nivel, pontuacao, score,
tempo)
VALUES ('$nome', '$nivel', '$pontuacao', '$score', '$tempo')";

if ($conn->query($sql) === TRUE) {
    echo "New record created successfully";
} else {
    echo "Error: " . $sql . "<br>" . $conn->error;
}

$conn->close();
?>
```

O teste desse código em PHP pode ser realizado diretamente no navegador passando parâmetros na própria url do arquivo:

http://".....”/placar_balanca_add.php?enviarnome=Rafael&nivel=5&pontuacao=1000&score=10&elapsedTime=20

Esse teste faz a inserção de dados na tabela de forma manual para fins de testes. Após fazer os testes, é possível deletar os dados enviados utilizando o próprio painel phpMyAdmin, além de poder zerar o id no campo AUTO_INCREMENT:



6. Função Javascript com jQuery.post()

A simulação precisa executar o código PHP passando os parâmetros necessários para o registro das variáveis no banco de dados. Segue exemplo de função para esse procedimento de integração entre a simulação e o banco MySQL usando jQuery.post() (FLANAGAN, 2010) e PHP:

```javascript
function EnviarPlacar(enviarnome, pontuacao, score, elapsedTime,
level) {
        var nivel = level + 1;
```

```
        $.post('http://"..caminho do arquivo no
servidor...."/placar_balanca_add.php',
        {enviarnome: enviarnome, pontuacao:pontuacao, score:score,
elapsedTime:elapsedTime, nivel:nivel});
        console.log(enviarnome, pontuacao, score, elapsedTime, nivel);
    }
```

7. Leitura do banco de dados em página HTML

Os dados de pontuações registrados no banco de dados podem ser lidos em uma página HTML utilizando script PHP. Segue um exemplo de código para esse procedimento:

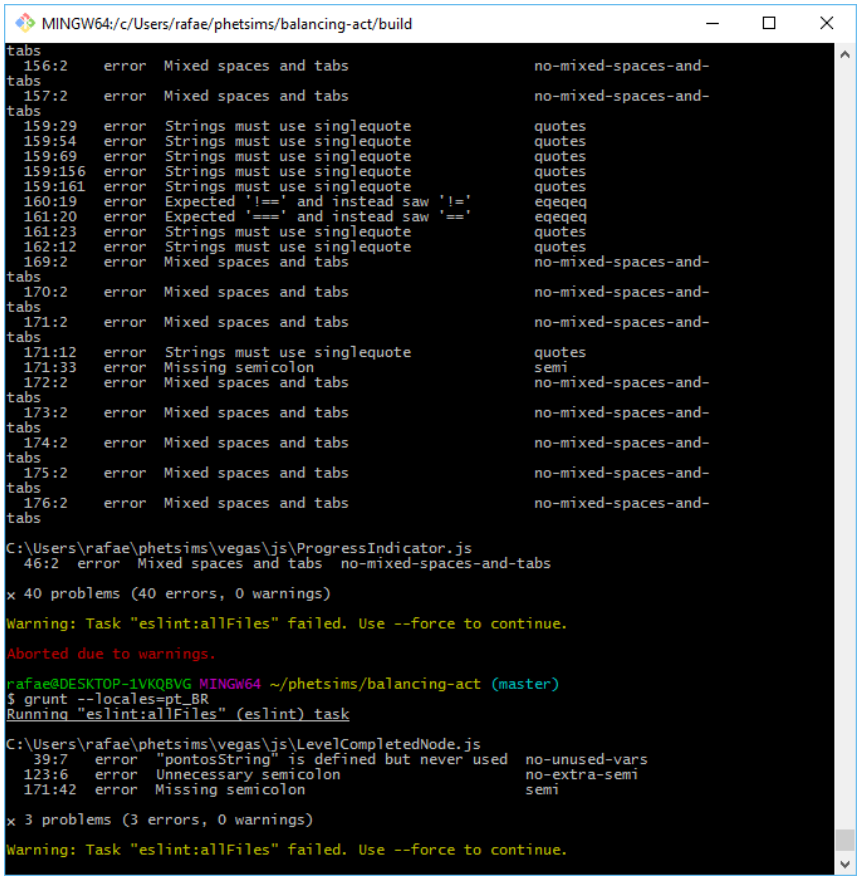
```
<?php
/* Template Name: Results */
?>
<?php get_header() ?>
  <?php include('play-game.php'); ?>
      <div id="content">
            <div class="padder">
                  <?php do_action( 'bp_before_blog_single_post' ) ?>
                  <div class="page" id="blog-single" role="main">
                  <?php if (have_posts()) : while (have_posts()) :
the_post(); ?>
                        <div id="post-<?php the_ID(); ?>" <?php
post_class(); ?>>
                              <div class="author-box">
                                    <?php echo get_avatar(
get_the_author_meta( 'user_email' ), '50' ); ?>
                                    <p><?php printf( _x( 'by %s', 'Post
written by...', 'buddypress' ), str_replace( '<a href=', '<a
rel="author" href=', bp_core_get_userlink( $post->post_author ) ) );
?></p>
                              </div>
                              <div class="post-content">
                                    <h2 class="posttitle"><?php
the_title(); ?></h2>
                                    <p class="date">
                                          <?php printf( __( '%1$s
<span>in %2$s</span>', 'buddypress' ), get_the_date(),
get_the_category_list( ', ' ) ); ?>
                                          <span class="post-utility
alignright"><?php edit_post_link( __( 'Edit this entry', 'buddypress'
) ); ?></span>
                                    </p>
                                    <div class="entry">
                                          <?php the_content( __( 'Read
the rest of this entry &rarr;', 'buddypress' ) ); ?>
                                          <?php
                                                global $wpdb;
                                                for ($x = 4; $x > 0;
$x--){
                                                      echo "<h5
style=\"text-align: center;\">Top 10 do Nível $x</h5>";

      $placar_balanca_nivel = $wpdb->get_results("SELECT * FROM
placar_balanca WHERE nivel=$x ORDER BY pontuacao DESC LIMIT 10;");
                                                      echo "<table
border=1 style=\"top:15px; left:10px;font-weight: bold;\">";
```

```
                                                    echo "<tr>";
                                                    echo "<th
style=\"background-color: #acf;padding: 4px;\">Pos.</th>";
                                                    echo "<th
style=\"background-color: #acf;padding: 4px;\">Nome</th>";
                                                    echo "<th
style=\"background-color: #acf;padding: 4px;\">Pontuação</th>";
                                                    //echo "<th
style=\"background-color: #acf;padding: 4px;\">Nível</th>";
                                                    echo "<th
style=\"background-color: #acf;padding: 4px;\">Data</th>";
                                                    echo "</tr>";
                                                    $i = 1;

      foreach($placar_balanca_nivel as $placar){
                                                    echo "<tr>";
                                                    echo "<td
style=\"padding: 2px;\">".$i."</td>";
                                                    $i++;
                                                    echo "<td
style=\"padding: 2px;\">".$placar->nome."</td>";
                                                    echo "<td
style=\"padding: 2px;\">".$placar->pontuacao."</td>";
                                                    //echo "<td
style=\"padding: 2px;\">".$placar->nivel."</td>";
                                                    $d = $placar->ts;
                                                    $d =
strtotime($d);
                                                    echo "<td
style=\"padding: 2px;\">".date("d/m/y",$d)."</td>";
                                                    echo "</tr>";
                                                    }
                                                    echo "</table>";
                                                    echo "<br>";
                                                }
                                            ?>
                                            <?php wp_link_pages( array(
'before' => '<div class="page-link"><p>' . __( 'Pages: ', 'buddypress'
), 'after' => '</p></div>', 'next_or_number' => 'number' ) ); ?>
                                        </div>
                                        <p class="postmetadata"><?php
the_tags( '<span class="tags">' . __( 'Tags: ', 'buddypress' ), ', ',
'</span>' ); ?> </p>
                                        <div class="alignleft"><?php
previous_post_link( '%link', '<span class="meta-nav">' . _x( '&larr;',
'Previous post link', 'buddypress' ) . '</span> %title' ); ?></div>
                                        <div class="alignright"><?php
next_post_link( '%link', '%title <span class="meta-nav">' . _x(
'&rarr;', 'Next post link', 'buddypress' ) . '</span>' ); ?></div>
                                    </div>
                                </div>
                    <?php comments_template(); ?>
                    <?php endwhile; else: ?>
                        <p><?php _e( 'Sorry, no posts matched your
criteria.', 'buddypress' ) ?></p>
                    <?php endif; ?>
            </div>
            <?php do_action( 'bp_after_blog_single_post' ) ?>
            </div><!-- .padder -->
      </div><!-- #content -->
      <?php get_sidebar() ?>
```

```
<?php get_footer() ?>
```

8. Problemas comuns na etapa de compilação

Após pronta a programação de uma simulação, o último passo é a compilação da simulação antes de ser enviada ao servidor. A aplicação “git”, que executa as tarefas de compilação, é bastante exigente com a formatação e organização dos códigos. Exemplo de erros comuns obtidos nesse processo:



Esses problemas são resolvidos revisando a formatação e a endentação do código Javascript. É importante seguir “boas maneiras” durante a escrita do código de programação.

PROBLEMA COM MOD_SECURITY

Outro erro comum ocorre durante o teste no servidor, dependendo do nível de segurança habilitado (MISCHEL, 2009), pode acontecer se o sistema bloquear a simulação no momento da integração com o banco de dados. Este problema resulta na seguinte resposta do servidor após chamada do código PHP:

```
Failed to load resource: the server responded with a status of 406 (Not Acceptable)
```

Este problema só pode ser resolvido entrando em contato com o suporte técnico responsável pelo servidor para solicitar que desabilitem a opção: **mod_security**.

PROBLEMA COM A CONCATENAÇÃO DE STRINGS

Os comandos $.post ou $.ajax type POST não passavam os dados de variáveis *string* corretamente para a url. Para contornar este problema, é preciso utilizar programação de concatenação das *strings* presentes nos parâmetros da url. Mesmo assim, é possível ter problemas com os caracteres diversos que os usuários podem digitar na hora de registrar seus nomes no placar. Para isso, a função escape() do Javascript é suficiente para limpar as *strings* antes do envio para o banco de dados MySQL. Exemplo:

```
var urlvar =
'placar_multiplicacao_add.php?enviarnome='+escape(enviarnome)+'&pontua
cao='+pontuacao+'&score='+score+'&elapsedTime='+elapsedTime+'&nivel='+
nivel+'&gamename='+gamename;
```

Neste caso, é aconselhável utilizar na configuração do banco de dados o formato de string do tipo: `utf8mb4_general_ci`.

9. Sugestão de trabalhos futuros de programação avançada

As simulações PhET de código aberto permitem a criação de aprimoramentos para atender melhor diferentes contextos de ensino e perfis de alunos. Como sugestão, ou desafio, seguem algumas ideias de elementos de *game design* que podem ser agregados às simulações PhET: opção de criação de avatar, opção de chat, salas de competição, painel de conquistas, registro de progressos, níveis de dificuldade adaptados ao usuário, e design responsivo para poder jogar em celular.

Todas as simulações PhET em HTML5 possuem código aberto, então, o limite de possibilidades de aprimoramentos fica por conta da criatividade e do conhecimento técnico. É importante lembrar que um bom recurso didático deve ser planejado com base em uma concepção de ensino e objetivos pedagógicos bem definidos (RIBEIRO *et al.*, 2015).

Conclusão

Este guia prático (básico e avançado) representa uma iniciativa de promover a criação de novas simulações interativas utilizando os recursos de código aberto do projeto PhET, que disponibiliza gratuitamente 133 simulações, em 36 idiomas, para o ensino de Ciência e Matemática, as quais já tiveram mais de 350 milhões de acessos.

As simulações são programadas com foco no reúso de códigos prontos. Os elementos visuais e as rotinas lógicas em uma simulação são armazenados

em pequenos módulos escritos em Javascript. Esse sistema modular permite o fácil reaproveitado de partes prontas na criação de novas simulações. Assim, o esforço inicial de aprendizagem da rotina de processo de programação das simulações é compensado, após certo conhecimento técnico, pela quantidade de módulos prontos já disponíveis no repositório do projeto PhET destinados à criação de novos laboratórios virtuais ou jogos digitais educacionais.

Referências

DELIDLE, M. **Mastering PhpMyAdmin 3.3.x for Effective MySQL Management: A Complete Guide to Getting Started with PhpMyAdmin 3.3 and Mastering Its Features**. Packt Pub., 2010.

FLANAGAN, D. **jQuery Pocket Reference: Read Less, Learn More**. O'Reilly Media, 2010. FRANKO, G. **Instant Dependency Management with Requirejs How-To**. Packt Publishing, 2013.

MISCHEL, M. **ModSecurity 2.5**. Packt Pub., 2009.

PIPINELLIS, A. **GitHub Essentials**. Packt Publishing, 2015.

RIBEIRO, Rafael João. ***Game design* aplicado em simulações interativas educacionais**. 2017. 185f. Tese (Doutorado em Ensino de Ciência e Tecnologia) - Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2017.

RIBEIRO, Rafael João *et al.* Teorias de Aprendizagem em Jogos Digitais Educacionais: um Panorama Brasileiro. **Novas Tecnologias na Educação**, v. 13, n. 1, p. 1–10, 2015.

RIBEIRO, Rafael João. ***Game design* aplicado em simulações interativas educacionais**. 2017. 187f. Tese (Doutorado em Ensino de Ciência e Tecnologia) - Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2017.

SUEHRING, S; VALADE, J. **PHP, MySQL, JavaScript & HTML5 All-in-One For Dummies**. Wiley, 2013.

WIEMAN, C. E. *et al.* Teaching Physics Using PhET Simulations. **The Physics Teacher**, v. 48, n. 4, p. 225, 2010.

ZAKAS, N C. **Professional JavaScript for Web Developers**. Wiley, 2011.

ANEXOS

Código modificado do LevelCompletedNode.js, que fica na pasta: \phetsims\vegas\js:

```
// Copyright 2013-2015, University of Colorado Boulder

/**
 * This node is used to display a user's results when they complete a
level.
 *
 * @author original John Blanco
 * @author original Chris Malley (PixelZoom, Inc.)
 * @author of this adapted version Rafael João Ribeiro
 */
define( function( require ) {
  'use strict';

  // modules
  var Color = require( 'SCENERY/util/Color' );
  var ProgressIndicator = require( 'VEGAS/ProgressIndicator' );
  var GameTimer = require( 'VEGAS/GameTimer' );
  var inherit = require( 'PHET_CORE/inherit' );
  var MultiLineText = require( 'SCENERY_PHET/MultiLineText' );
  var Panel = require( 'SUN/Panel' );
  var PhetFont = require( 'SCENERY_PHET/PhetFont' );
  var Property = require( 'AXON/Property' );
  var StringUtils = require( 'PHETCOMMON/util/StringUtils' );
  var Text = require( 'SCENERY/nodes/Text' );
  var TextPushButton = require( 'SUN/buttons/TextPushButton' );
  var VBox = require( 'SCENERY/nodes/VBox' );
  var vegas = require( 'VEGAS/vegas' );
  var Tandem = require( 'TANDEM/Tandem' );

  // strings
  var keepTryingString = require( 'string!VEGAS/keepTrying' );
  var goodString = require( 'string!VEGAS/good' );
  var greatString = require( 'string!VEGAS/great' );
  var excellentString = require( 'string!VEGAS/excellent' );
  var labelScoreMaxString = require( 'string!VEGAS/label.score.max' );
  var labelTimeString = require( 'string!VEGAS/label.time' );
  var yourNewBestString = require( 'string!VEGAS/yourNewBest' );
  var pattern0YourBestString = require(
'string!VEGAS/pattern.0yourBest' );
  var continueString = require( 'string!VEGAS/continue' );
  var labelLevelString = require( 'string!VEGAS/label.level' );
  var pontosString = "Pontos: "; //rafaeladd

  var pontuacao = 0; //rafaeladd
  var placargatilho = true; //rafaeladd
  var nivel = 0; //rafaeladd

  /**
   * @param {number} level starting from zero, 1 added to this when
displayed
   * @param {number} score
   * @param {number} perfectScore
   * @param {number} numStars
```

```
   * @param {boolean} timerEnabled
   * @param {number} elapsedTime (in seconds)
   * @param {number} bestTimeAtThisLevel (in seconds), null indicates
no best time
   * @param {boolean} isNewBestTime
   * @param {function} continueFunction Function to call when the user
presses the 'Continue' button.
   * @param {Object} [options]
   * @constructor
   */
  function LevelCompletedNode( level, score, perfectScore, numStars,
timerEnabled, elapsedTime, bestTimeAtThisLevel, isNewBestTime,
continueFunction, options ) {

    options = _.extend( {
      levelVisible: true, // display the level number?
      fill: new Color( 180, 205, 255 ),
      stroke: 'black',
      lineWidth: 2,
      cornerRadius: 35,
      xMargin: 20,
      yMargin: 20,
      ySpacing: 15,//rafaeladd 30 para 15
      titleFont: new PhetFont( { size: 28, weight: 'bold' } ),
      infoFont: new PhetFont( { size: 22, weight: 'bold' } ),
      buttonFont: new PhetFont( 26 ),
      buttonFill: new Color( 255, 255, 0 ),
      starDiameter: 62,
      tandem: null
    }, options );
    Tandem.validateOptions( options ); // The tandem is required when
brand==='phet-io'

    // nodes to be added to the panel
    var children = [];

    // Title, which changes based on how the user did.
    var proportionCorrect = score / perfectScore;
    var titleText = keepTryingString;
    if ( proportionCorrect > 0.95 ) {
      titleText = excellentString;
    }
    else if ( proportionCorrect > 0.75 ) {
      titleText = greatString;
    }
    else if ( proportionCorrect >= 0.5 ) {
      titleText = goodString;
    }
    var title = new Text( titleText, { font: options.titleFont } );
    children.push( title );

    // Progress indicator
    children.push( new ProgressIndicator( numStars, new Property(
score ), perfectScore, {
      starInnerRadius: options.starDiameter / 4,
      starOuterRadius: options.starDiameter / 2
    } ) );

    // Level (optional)
    if ( options.levelVisible ) {
```

```
      children.push( new Text( StringUtils.format( labelLevelString,
level + 1 ), { font: options.infoFont } ) );
    }

    // Score
    children.push( new Text( StringUtils.format( labelScoreMaxString,
score, perfectScore ), { font: options.infoFont } ) );

    // Time (optional)
    if ( timerEnabled ) {
      var time = new MultiLineText( StringUtils.format(
labelTimeString, GameTimer.formatTime( elapsedTime ) ), {
        font: options.infoFont,
        align: 'center'
      } );
      if ( isNewBestTime ) {
        time.text = time.text + '\n' + yourNewBestString;
      }
      else if ( bestTimeAtThisLevel !== null ) {
        time.text = time.text + '\n' + StringUtils.format(
pattern0YourBestString, GameTimer.formatTime( bestTimeAtThisLevel ) );
      }
      children.push( time );
    };
      //rafaeladd if..
      /*
      if (elapsedTime < 288) {
            pontuacao = score*24 + (288-elapsedTime)*score;//rafaeladd;
      }
      else {
            pontuacao = score*24;
      }
      */
      pontuacao = Math.round(100 * score * score * score /
elapsedTime);

      // Continue button
    children.push( new TextPushButton( continueString, {
      listener: continueFunction,
      font: options.buttonFont,
      baseColor: options.buttonFill,
      tandem: options.tandem && options.tandem.createTandem(
'continueButton' )
    } ) );
      //Botão placar //inicia rafaeladd
      children.push(new Text("Pontuação (score e tempo): " +
pontuacao,{ font: options.infoFont } ) );//rafaeladd
      placargatilho = true;
      children.push( new TextPushButton("Enviar para o placar!", {
        listener: function() {
              if (placargatilho) {
                  nivel = parseInt(level + 1);
                  prompt("Você fez "+ pontuacao + " pontos.\n" + "Entre
com o seu nome para mostrar\nno placar geral do nível "+ nivel +".",
"Digite aqui o seu nome.");//rafaeladd
                  placargatilho = false;
              }
              else {
                    alert("Placar já enviado.")
              }
        } ,
```

```
        font: options.buttonFont,
        baseColor: options.buttonFill,
        tandem: options.tandem && options.tandem.createTandem(
'continueButton' )
      } ) );
      //termina aqui rafaeladd

    // Panel
    Panel.call( this, new VBox( { children: children, spacing:
options.ySpacing } ), options );
  }

  vegas.register( 'LevelCompletedNode', LevelCompletedNode );

  return inherit( Panel, LevelCompletedNode );
} );
```