Fraction Bars Project Code

GitHub Copilot

September 29, 2025

Contents

1 Main HTML File

1.1 Fraction_Bars.html

```html
<!DOCTYPE html>
<!-- saved from url=(0033)https://educn101.sitehost.iu.edu/ -->
<html lang="en"><!-- test this will be a conflict --><head><meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

        <title>HTML 5: Fraction Bars</title>
        <link rel="stylesheet" href="./Fraction_Bars_files/fractionBars.css" type="text/css">
        <link rel="stylesheet" href="./Fraction_Bars_files/lang_eng.css" type="text/css">
        <link rel="stylesheet" href="./Fraction_Bars_files/deneme.css" type="text/css">
        <link rel="stylesheet" href="./Fraction_Bars_files/jquery-ui-1.10.3.custom.min.css" type="text/css">
        <script src="./Fraction_Bars_files/jquery-1.9.1.min.js" type="text/javascript" language="javascript"></script>
        <script src="./Fraction_Bars_files/jquery-ui-1.10.3.custom.min.js" type="text/javascript" language="javascript"></script>
        <script src="https://cdnjs.cloudflare.com/ajax/libs/jqueryui-touch-punch/0.2.3/jquery.ui.touch-punch.min.js"></script>
        <script src="./Fraction_Bars_files/cycle.js" type="text/javascript" language="javascript"></script>
        <script src="./Fraction_Bars_files/FileSaver.min.js" type="text/javascript" language="javascript"></script>
        <script src="./Fraction_Bars_files/Blob.js" type="text/javascript" language="javascript"></script>
        <script src="./Fraction_Bars_files/utilities.js" type="application/javascript" language="javascript"></script>
        <script src="./Fraction_Bars_files/Point.js" type="application/javascript" language="javascript"></script>
        <script src="./Fraction_Bars_files/Bar.js" type="application/javascript" language="javascript"></script>
        <script src="./Fraction_Bars_files/Mat.js" type="application/javascript" language="javascript"></script>
        <script src="./Fraction_Bars_files/Split.js" type="application/javascript" language="javascript"></script>
        <script src="./Fraction_Bars_files/SplitsWidget.js" type="application/javascript" language="javascript"></script>
        <script src="./Fraction_Bars_files/Line.js" type="application/javascript" language="javascript"></script>
        <script src="./Fraction_Bars_files/CanvasState.js" type="application/javascript" language="javascript"></script>
        <script src="./Fraction_Bars_files/FractionBarsCanvas.js" type="application/javascript" language="javascript"></script>
        <script src="./Fraction_Bars_files/fractionBars.js" type="application/javascript" language="javascript"></script>


</head>
<body data-new-gr-c-s-check-loaded="14.1231.0" data-gr-ext-installed="">
<a class="skip-link" href="https://educn101.sitehost.iu.edu/#main">Skip to main content</a>
        <h1 class="bar_titles" id="bar_titles">Fraction Bars</h1>
<div id="main">
        <div id="tools">
                <div class="toolGroup">
                        <a class="c_bar" id="tool_bar">&nbsp;</a>
                        <a class="c_mat" id="tool_mat">&nbsp;</a>
                         <!-- <a id="tool_cover">Cover</a> -->
                </div>
                <div class="toolGroup">
                        <a class="c_copy" id="action_copy">&nbsp;</a>
                        <a class="c_repeat" id="tool_repeat">&nbsp;</a>
                        <a class="c_iterate" id="window_iterate">&nbsp;</a>
                        <a class="c_join" id="action_join">&nbsp;</a>
                        <a class="c_delete" id="action_delete">&nbsp;</a>
                </div>
                <div class="toolGroup">
                        <a class="c_parts" id="window_split">&nbsp;</a>
                        <a class="c_pieces" id="tool_manualSplit">&nbsp;</a>
                        <a class="c_b_apart" id="action_breakApart">&nbsp;</a>
                        <a class="c_pullout" id="action_pullOutSplit">&nbsp;</a>
                        <a class="c_c_parts" id="action_clearSplits">&nbsp;</a>
                </div>
                <div class="toolGroup">
                        <a class="c_set_unit" id="action_setUnitBar">&nbsp;</a>
                        <a class="c_measure" id="action_measure">&nbsp;</a>
                        <a class="c_make" id="action_make">Make</a>
                        <a class="c_label" id="window_label">&nbsp;</a>
                </div>
                <div class="toolGroup">
                        <a class="c_undo" id="action_undo"></a>
                        <a class="c_redo" id="action_redo">&nbsp;</a>
                        <a class="c_save" id="action_save">&nbsp;</a>
                        <a class="c_open" id="action_open">&nbsp;</a>
                        <a class="c_new" id="action_clearAll">&nbsp;</a>
                        <a class="c_print" id="action_print">&nbsp;</a>
                        <a class="c_properties" id="window_properties">&nbsp;</a>
                </div>
                <div class="toolGroup" style="text-align:center">
                        <a class="colorBlock color10 colorSelected" id="setColor1">&nbsp;</a>
                        <a class="colorBlock color3" id="setColor2">&nbsp;</a>
                        <a class="colorBlock color7" id="setColor3">&nbsp;</a>
                        <a class="colorBlock color5" id="setColor4">&nbsp;</a>
                        <a class="colorBlock color12" id="setColor5">&nbsp;</a>
                        <a class="colorBlock color9" id="setColor6">&nbsp;</a>
                        <a class="colorBlock color13" id="setColor7">&nbsp;</a>
                        <a class="colorBlock color14" id="setColor8">&nbsp;</a>
                </div>
          <div class="toolGroup" style="text-align:center">&nbsp;
                <select class="c_filetext" id="id_filetext" multiple="multiple" style="display: none;"></select>
                        <a class="c_previous" id="action_previous" style="display: none;">&nbsp;</a>
                        <a class="c_next" id="action_next" style="display: none;">&nbsp;</a>
          </div>
                <div class="toolGroup" style="text-align:center">
                        <a class="hideShow c_hide" id="tool_hide">&nbsp;</a>
                        <a class="hideShow c_show" id="action_show">&nbsp;</a>
                </div>
        </div>

        <div hidden="" id="flags">
                <a id="marked-iterate" data-flag="true">&nbsp;</a>
        </div>

        <canvas id="fbCanvas" width="700" height="600"></canvas>

        <!--
                <p style="clear:both">Mouse: (<span id="mouseAction"> </span>) <span id="mouseLoc"> </span></p>
        -->

        <input name="labelField" id="labelInput" type="text" value="">











</div>


<div class="ui-dialog ui-widget ui-widget-content ui-corner-all ui-front ui-dialog-buttons ui-draggable" tabindex="-1" role="dialog" aria-describedby="dialog-splits" aria-labelledby="ui-id-1" style="display: none;"><div class="ui-dialog-titlebar ui-widget-header ui-corner-all ui-helper-clearfix"><span id="ui-id-1" class="ui-dialog-title"></span><button class="ui-button ui-widget ui-state-default ui-corner-all ui-button-icon-only ui-dialog-titlebar-close" role="button" aria-disabled="false" title="close"><span class="ui-button-icon-primary ui-icon ui-icon-closethick"></span><span class="ui-button-text">close</span></button></div><div class="c_dialog_splits ui-dialog-content ui-widget-content" id="dialog-splits">

                <p>
                        <canvas id="split-display" width="100" height="100"></canvas>
                </p>
                <div class="radio_vert_horz" id="radio_vert" style="DISPLAY: none">
                        <label class="c_vertical" for="vert">&nbsp;</label>
                        <input type="radio" name="vert_horiz" id="vert" value="Vertical" checked="">
                        <label class="c_horizontal" for="horiz">&nbsp;</label>
                        <input type="radio" name="vert_horiz" id="horiz" value="Horizontal">
                </div>

                <p>
                        <label class="c_number_part" for="split-slider-field">&nbsp;</label>
                        <input type="text" id="split-slider-field" style="border:0; color:#f6931f; font-weight:bold;" value="2" readonly="">
                </p>

                <div id="split-slider" class="ui-slider ui-slider-horizontal ui-widget ui-widget-content ui-corner-all" aria-disabled="false"><a class="ui-slider-handle ui-state-default ui-corner-all" href="https://educn101.sitehost.iu.edu/#" style="left: 0%;"></a></div>

                <div id="radio_whole">
                        <label class="c_part_whole" for="whole">&nbsp;</label>
                        <input type="radio" name="whole_part" id="whole" value="Whole" checked="">
                        <br>
                        <label class="c_part_part" for="part">&nbsp;</label>
                        <input type="radio" name="whole_part" id="part" value="Part">
                </div>

        </div><div class="ui-dialog-buttonpane ui-widget-content ui-helper-clearfix"><div class="ui-dialog-buttonset"><button type="button" class="ui-button ui-widget ui-state-default ui-corner-all ui-button-text-only" role="button" aria-disabled="false"><span class="ui-button-text">Ok</span></button><button type="button" class="ui-button ui-widget ui-state-default ui-corner-all ui-button-text-only" role="button" aria-disabled="false"><span class="ui-button-text">Cancel</span></button></div></div></div><div class="ui-dialog ui-widget ui-widget-content ui-corner-all ui-front ui-dialog-buttons ui-draggable" tabindex="-1" role="dialog" aria-describedby="dialog-properties" aria-labelledby="ui-id-2" style="display: none;"><div class="ui-dialog-titlebar ui-widget-header ui-corner-all ui-helper-clearfix"><span id="ui-id-2" class="ui-dialog-title"></span><button class="ui-button ui-widget ui-state-default ui-corner-all ui-button-icon-only ui-dialog-titlebar-close" role="button" aria-disabled="false" title="close"><span class="ui-button-icon-primary ui-icon ui-icon-closethick"></span><span class="ui-button-text">close</span></button></div><div class="c_dialog_properties ui-dialog-content ui-widget-content" id="dialog-properties">

                <label class="c_iterations" for="same">&nbsp;</label>
                        <div id="radio_iterate_p" title="Itterations">
                                <p>
                                        <label class="c_dont_create" for="same">&nbsp;</label>
                                        <input type="radio" name="create" id="same" value="Same">
                                        <br>
                                        <label class="c_create_new" for="new">&nbsp;</label>
                                        <input type="radio" name="create" id="new" value="New" checked="">
                                        <br>
                                        ---
                                        <br>

                                        <label class="c_two_way" for="two_way">&nbsp;</label>
                                        <input type="radio" name="two_ittr" id="two_way" value="Two_way">
                                        <br>
                                        <label class="c_one_way" for="one_way">&nbsp;</label>
                                        <input type="radio" name="two_ittr" id="one_way" value="One_way" checked="">
                                </p>
                        </div>
                        <br>
                        <label class="c_splits" for="same">&nbsp;</label>
                        <div id="radio_split_p" title="Split or Manual Split">
                                <p>
                                        <label class="c_vert_horiz" for="two_horiz">&nbsp;</label>
                                        <input type="radio" name="two_split" id="two_horiz" value="Two_horiz">
                                        <br>
                                        <label class="c_only_vert" for="one_horiz">&nbsp;</label>
                                        <input type="radio" name="two_split" id="one_horiz" value="One_horiz" checked="">
                                </p>
                                </div>

                                <!--
                        <label class="c_lang" for="lang">&nbsp;</label>
                     <div id="radio_lang_tr" title="Language">
                                <p>
                                        <fieldset class="elist">

                                            <ul>
                                            <li><input type="radio" name="lang" id="lang_tr" value="lang_tur" /><label class="c_lang_tur" for="lang_tr"></label></li>
                                            <li><input type="radio" name="lang" id="lang_en" value="lang_eng" checked /><label class="c_lang_eng" for="lang_en"></label></li>
                                        </ul>
                                    </fieldset>
                                </p>
                        </div>
                -->
                        <br>

                        <div id="tools2">
                                <div class="toolGroup" style="text-align:center">
                                        <p>
                                                <label class="c_color" for="color">&nbsp;</label>
                                        </p>
                                                        <a class="colorBlock1 color1" id="setColor1_2">&nbsp;</a>
                                                        <a class="colorBlock1 color2" id="setColor2_2">&nbsp;</a>
                                                        <a class="colorBlock1 color3" id="setColor3_2">&nbsp;</a>
                                                        <a class="colorBlock1 color4" id="setColor4_2">&nbsp;</a>
                                                        <a class="colorBlock1 color5" id="setColor5_2">&nbsp;</a>
                                                        <a class="colorBlock1 color6" id="setColor6_2">&nbsp;</a>
                                                        <a class="colorBlock1 color7" id="setColor7_2">&nbsp;</a>
                                                        <a class="colorBlock1 color8" id="setColor8_2">&nbsp;</a>
                                                        <a class="colorBlock1 color9" id="setColor9_2">&nbsp;</a>
                                                        <a class="colorBlock1 color10 colorSelected" id="setColor10">&nbsp;</a>
                                                        <a class="colorBlock1 color11" id="setColor11">&nbsp;</a>
                                                        <a class="colorBlock1 color12" id="setColor12">&nbsp;</a>
                                                        <a class="colorBlock1 color13" id="setColor13">&nbsp;</a>
                                                        <a class="colorBlock1 color14" id="setColor14">&nbsp;</a>
                                                        <a class="colorBlock1 color15" id="setColor15">&nbsp;</a>
                                                        <a class="colorBlock1 color16" id="setColor16">&nbsp;</a>
                                </div>
                        </div>
        </div><div class="ui-dialog-buttonpane ui-widget-content ui-helper-clearfix"><div class="ui-dialog-buttonset"><button type="button" class="ui-button ui-widget ui-state-default ui-corner-all ui-button-text-only" role="button" aria-disabled="false"><span class="ui-button-text">Ok</span></button><button type="button" class="ui-button ui-widget ui-state-default ui-corner-all ui-button-text-only" role="button" aria-disabled="false"><span class="ui-button-text">Cancel</span></button></div></div></div><div class="ui-dialog ui-widget ui-widget-content ui-corner-all ui-front ui-dialog-buttons ui-draggable" tabindex="-1" role="dialog" aria-describedby="dialog-iterate" aria-labelledby="ui-id-3" style="display: none;"><div class="ui-dialog-titlebar ui-widget-header ui-corner-all ui-helper-clearfix"><span id="ui-id-3" class="ui-dialog-title"></span><button class="ui-button ui-widget ui-state-default ui-corner-all ui-button-icon-only ui-dialog-titlebar-close" role="button" aria-disabled="false" title="close"><span class="ui-button-icon-primary ui-icon ui-icon-closethick"></span><span class="ui-button-text">close</span></button></div><div id="dialog-iterate" class="c_dialog_iterate ui-dialog-content ui-widget-content">

                <div class="radio_itterate_vert_horz" id="iterate_vert-horiz" style="DISPLAY: none">
```

```html
                        <label class="c_vertical" for="iterate_vert">&nbsp;</label>
                        <input type="radio" name="vert_horiz" id="iterate_vert" value="Vertical"
                         ↪  checked="">
                        <label class="c_horizontal" for="iterate_horiz">&nbsp;</label>
                        <input type="radio" name="vert_horiz" id="iterate_horiz" value="Horizontal">
                </div>

                <p>
                        <label class="c_number_iterations" for="iterate-field">&nbsp;</label>
                        <input type="text" id="iterate-field" style="border:1; color:#000000;
                         ↪  font-weight:bold;" value="2">
                </p>

        </div><div class="ui-dialog-buttonpane ui-widget-content ui-helper-clearfix"><div
        ↪  class="ui-dialog-buttonset"><button type="button" class="ui-button ui-widget ui-state-default
        ↪  ui-corner-all ui-button-text-only" role="button" aria-disabled="false"><span
        ↪  class="ui-button-text">Ok</span></button><button type="button" class="ui-button ui-widget
        ↪  ui-state-default ui-corner-all ui-button-text-only" role="button" aria-disabled="false"><span
        ↪  class="ui-button-text">Cancel</span></button></div></div></div><div class="ui-dialog ui-widget
        ↪  ui-widget-content ui-corner-all ui-front ui-dialog-buttons ui-draggable" tabindex="-1"
        ↪  role="dialog" aria-describedby="dialog-make" aria-labelledby="ui-id-4" style="display:
        ↪  none;"><div class="ui-dialog-titlebar ui-widget-header ui-corner-all ui-helper-clearfix"><span
        ↪  id="ui-id-4" class="ui-dialog-title"></span><button class="ui-button ui-widget
        ↪  ui-state-default ui-corner-all ui-button-icon-only ui-dialog-titlebar-close" role="button"
        ↪  aria-disabled="false" title="close"><span class="ui-button-icon-primary ui-icon
        ↪  ui-icon-closethick"></span><span class="ui-button-text">close</span></button></div><div
        ↪  id="dialog-make" class="c_dialog_make ui-dialog-content ui-widget-content">

                <p>
                        <label class="c_number_whole" for="whole-field">Write Fraction:</label>
                        </p><table style="width:10">

      <tbody>
        <tr>
          <td rowspan="2"><input type="text" id="whole-field" style="border:1; color:#000000;
           ↪  font-weight:bold; text-align:right;" size="4" value=" "></td>
          <td style="border-bottom:solid 1px"><input type="text" id="num-field" style="border:1;
           ↪  color:#000000; font-weight:bold; text-align:center;" size="4"></td>
          </tr>
          <tr>
            <td><input type="text" id="denum-field" style="border:1; color:#000000; font-weight:bold;
             ↪  text-align:center;" size="4"></td>
          </tr>
      </tbody>
   </table>
                        &nbsp;&nbsp;


                <p></p>

        </div><div class="ui-dialog-buttonpane ui-widget-content ui-helper-clearfix"><div
        ↪  class="ui-dialog-buttonset"><button type="button" class="ui-button ui-widget ui-state-default
        ↪  ui-corner-all ui-button-text-only" role="button" aria-disabled="false"><span
        ↪  class="ui-button-text">Ok</span></button><button type="button" class="ui-button ui-widget
        ↪  ui-state-default ui-corner-all ui-button-text-only" role="button" aria-disabled="false"><span
        ↪  class="ui-button-text">Cancel</span></button></div></div></div><div class="ui-dialog ui-widget
        ↪  ui-widget-content ui-corner-all ui-front ui-dialog-buttons ui-draggable ui-resizable"
        ↪  tabindex="-1" role="dialog" aria-describedby="dialog-file" aria-labelledby="ui-id-5"
        ↪  style="display: none; position: absolute;"><div class="ui-dialog-titlebar ui-widget-header
        ↪  ui-corner-all ui-helper-clearfix"><span id="ui-id-5" class="ui-dialog-title"></span><button
        ↪  class="ui-button ui-widget ui-state-default ui-corner-all ui-button-icon-only
        ↪  ui-dialog-titlebar-close" role="button" aria-disabled="false" title="close"><span
        ↪  class="ui-button-icon-primary ui-icon ui-icon-closethick"></span><span
        ↪  class="ui-button-text">close</span></button></div><div class="c_choose_file ui-dialog-content
        ↪  ui-widget-content" id="dialog-file">
                <p class="c_open_file">&nbsp;</p>
                <p></p>
                <p></p>
                <input type="file" id="files" name="files[]" multiple="">
<!--                    <output id="list"></output> -->
```

```
        </div><div class="ui-dialog-buttonpane ui-widget-content ui-helper-clearfix"><div
        ↪  class="ui-dialog-buttonset"><button type="button" class="ui-button ui-widget ui-state-default
        ↪  ui-corner-all ui-button-text-only" role="button" aria-disabled="false"><span
        ↪  class="ui-button-text">Cancel</span></button></div></div><div class="ui-resizable-handle
        ↪  ui-resizable-n" style="z-index: 90;"></div><div class="ui-resizable-handle ui-resizable-e"
        ↪  style="z-index: 90;"></div><div class="ui-resizable-handle ui-resizable-s" style="z-index:
        ↪  90;"></div><div class="ui-resizable-handle ui-resizable-w" style="z-index: 90;"></div><div
        ↪  class="ui-resizable-handle ui-resizable-se ui-icon ui-icon-gripsmall-diagonal-se"
        ↪  style="z-index: 90;"></div><div class="ui-resizable-handle ui-resizable-sw" style="z-index:
        ↪  90;"></div><div class="ui-resizable-handle ui-resizable-ne" style="z-index: 90;"></div><div
        ↪  class="ui-resizable-handle ui-resizable-nw" style="z-index:
        ↪  90;"></div></div></body><grammarly-desktop-integration
        ↪  data-grammarly-shadow-root="true"><template shadowrootmode="open"><style>
      div.grammarly-desktop-integration {
        position: absolute;
        width: 1px;
        height: 1px;
        padding: 0;
        margin: -1px;
        overflow: hidden;
        clip: rect(0, 0, 0, 0);
        white-space: nowrap;
        border: 0;
        -moz-user-select: none;
        -webkit-user-select: none;
        -ms-user-select:none;
        user-select:none;
      }

      div.grammarly-desktop-integration:before {
        content: attr(data-content);
      }
    </style><div aria-label="grammarly-integration" role="group" tabindex="-1"
    ↪  class="grammarly-desktop-integration" data-content="{&quot;mode&quot;:&quot;limited&quot;,&quot;i
    ↪  sActive&quot;:false,&quot;isUserDisabled&quot;:false,&quot;isAlwaysAvailableAssistantEnabled&quot
    ↪  ;:false}"></div></template></grammarly-desktop-integration></html>
```

2 JavaScript Files

2.1 Core Fraction Bars Logic

2.1.1 fractionBars.js

```
// Copyright University of Massachusetts Dartmouth 2014
//
// Designed and built by James P. Burke and Jason Orrill
// Modified and developed by Hakan Sandir
//
// This Javascript version of Fraction Bars is based on
// the Transparent Media desktop version of Fraction Bars,
// which in turn was based on the original TIMA Bars software
// by John Olive and Leslie Steffe.
// We thank them for allowing us to update that product.




/*
// pull in our other files

// TODO: figure out if this is really a desirable thing to do. I like it in
// that this approach feels more like other languages, but there are issues
// with the classes not being available when I expect them to be.

include_js('class/Point.js', 'js/');
include_js('class/Bar.js', 'js/');
include_js('class/Mat.js', 'js/');
include_js('class/Split.js', 'js/');
include_js('class/Line.js', 'js/');
include_js('class/FractionBarsCanvas.js', 'js/');

```

```javascript
*/

var point1 = null ;
var point2 = null;
var fbContext = null ;
var splitWidgetContext = null;
var hiddenButtons = [];
var hiddenButtonsName=[];

var fracEvent = null;

splitWidgetObj = null;

$(document).ready(function() {
//first attempt
        hideButton("id_filetext");
        hideButton("action_previous");
        hideButton("action_next");



        fbContext = $('#fbCanvas')[0].getContext( '2d' ) ;
        fbCanvasObj = new FractionBarsCanvas(fbContext);
        splitWidgetContext = $('#split-display')[0].getContext('2d');
        var splitWidgetObj = new SplitsWidget(splitWidgetContext);

        // High-DPI/Retina support: scale canvas for crisp display
        var dpr = 3; // 3x for Retina/HiDPI
        var $canvas = $('#fbCanvas');
        var cssWidth = $canvas.attr('width');
        var cssHeight = $canvas.attr('height');
        $canvas[0].width = cssWidth * dpr;
        $canvas[0].height = cssHeight * dpr;
        $canvas.css({ width: cssWidth + 'px', height: cssHeight + 'px' });
        fbContext.setTransform(dpr, 0, 0, dpr, 0, 0);

        $("#split-slider").slider({
                change: function(event,ui) {
                        splitWidgetObj.handleSliderChange(event, ui);
                }
        });

        $("#vert,#horiz").change(handleVertHorizChange);

        function handleVertHorizChange(event) {
                splitWidgetObj.handleVertHorizChange(event);
        }




        $( "#files" ).change(handleFileSelect);
        FBFileReader = new FileReader();




//First attempt
        $( "#id_filetext" ).change(handleListSelect);
//


        $('#fbCanvas').dblclick(function() {
                var fbImg = fbContext.getImageData(0,0,1000,600) ;
                fbContext.clearRect(0,0,1000,600) ;
//                fbContext.restore() ;
                fbContext.putImageData(fbImg,0,0);
        });

        $('#fbCanvas').mousemove(function(e) {
                fracEvent = e;
                updateMouseLoc(e, $(this));
                updateMouseAction('mousemove');

                var p = Point.createFromMouseEvent(e, $(this)) ;

                if (fbCanvasObj.currentAction == "manualSplit") {
                        fbCanvasObj.manualSplitPoint = p;
                        fbCanvasObj.refreshCanvas();
                }

                if(fbCanvasObj.mouseDownLoc !== null) {
                        fbCanvasObj.updateCanvas(p);
                }

//                if (fbCanvasObj.currentAction == "manualSplit") {
//                        fbCanvasObj.manualSplitXORDraw(p);
//                }

        });

        $('#fbCanvas').mousedown(function(e) {

                fbCanvasObj.check_for_drag = true;
                fbCanvasObj.cacheUndoState();

                updateMouseLoc(e, $(this));
                updateMouseAction('mousedown');
                fbCanvasObj.mouseDownLoc = Point.createFromMouseEvent(e, $(this)) ;
                var b = fbCanvasObj.barClickedOn() ;
                var m = fbCanvasObj.matClickedOn() ;

                if( (fbCanvasObj.currentAction == 'bar') || (fbCanvasObj.currentAction == "mat")) {
                        fbCanvasObj.saveCanvas() ;
                } else if( fbCanvasObj.currentAction == 'repeat' ) {
                        fbCanvasObj.addUndoState();
                        b.repeat(fbCanvasObj.mouseDownLoc);
                        fbCanvasObj.refreshCanvas();
                } else {
                        // The click is being used to update the selected bars
                        if( b !== null ) {
                                if( $.inArray(b, fbCanvasObj.selectedBars) == -1) { // clicked on bar is
                                ↪  not already selected
                                        if( !Utilities.shiftKeyDown ) {
                                                fbCanvasObj.clearSelection();
                                        }
                                        $.each( fbCanvasObj.selectedBars, function(index, bar) {
                                                bar.clearSplitSelection();
                                        });
                                        fbCanvasObj.barToFront(b);
                                        fbCanvasObj.selectedBars.push(b);
                                        b.isSelected = true;
                                        b.selectSplit(fbCanvasObj.mouseDownLoc);
                                } else {
                                ↪                          // clicked bar is already selected
                                        $.each( fbCanvasObj.selectedBars, function(index, bar) {
                                                bar.clearSplitSelection();
                                        });
                                        if( !Utilities.shiftKeyDown ) {
                                                b.selectSplit(fbCanvasObj.mouseDownLoc);
                                        } else {
                                                fbCanvasObj.removeBarFromSelection(b);
                                        }
                                        fbCanvasObj.barToFront(b);
                                }
                                if (fbCanvasObj.currentAction == "manualSplit") {
                                        fbCanvasObj.clearSelection();
                                }
                        } else if( m !== null ) {
                                if( $.inArray(m, fbCanvasObj.selectedMats) == -1) { // clicked on mat is
                                ↪  not already selected
                                        if( !Utilities.shiftKeyDown ) {
                                                fbCanvasObj.clearSelection();
                                        }
                                        m.isSelected = true;
                                        fbCanvasObj.selectedMats.push(m);
                                } else {  // Clicked on mat is already selected
                                        if( Utilities.shiftKeyDown ) {
                                                fbCanvasObj.removeMatFromSelection(m);
                                        }
                                }
                        } else {
                                fbCanvasObj.clearSelection();
                        }
                        fbCanvasObj.refreshCanvas();
                }
        }) ;

        $('#fbCanvas').mouseup(function(e) {
                updateMouseLoc(e, $(this));
                updateMouseAction('mouseup');

                fbCanvasObj.mouseUpLoc = Point.createFromMouseEvent(e, $(this)) ;


                if( fbCanvasObj.currentAction == 'bar' ) {
                        fbCanvasObj.addUndoState();
                        fbCanvasObj.addBar() ;
                        fbCanvasObj.clear_selection_button ();

                } else if (fbCanvasObj.currentAction == 'mat') {
                        fbCanvasObj.addUndoState();
                        fbCanvasObj.addMat();
                        fbCanvasObj.clear_selection_button ();
                }


                if (fbCanvasObj.found_a_drag){
                        fbCanvasObj.finalizeCachedUndoState();
                        fbCanvasObj.check_for_drag = false;
                }

                fbCanvasObj.mouseUpLoc = null ;
                fbCanvasObj.mouseDownLoc = null ;
                fbCanvasObj.mouseLastLoc = null ;

        }) ;

        $('.colorBlock').click(function(e) {
                fbCanvasObj.setFillColor( $(this).css('background-color'));
                $('.colorBlock').removeClass('colorSelected');
                $(this).addClass('colorSelected');
                fbCanvasObj.updateColorsOfSelectedBars();
                fbCanvasObj.refreshCanvas();
        }) ;

//first attempt
        $('.colorBlock1').click(function(e) {
document.getElementById('fbCanvas').style.backgroundColor = $(this).css('background-color');
                $('.colorBlock1').removeClass('colorSelected');
                $(this).addClass('colorSelected');
        }) ;
//


        $('a').click(function(e) {

                var thisId = $(this).attr('id') ;
                if (thisId === null) { return; }
                var tool_on = false; // just temporarily keeps track of whether we're turning a tool on or
                ↪  off

//                First, handle any hiding, if we're in that mode
                if ((fbCanvasObj.currentAction == 'hide') && (thisId.indexOf('hide') == -1) ) {
                        $(this).hide();
                        hiddenButtonsName.push(thisId);
                        hiddenButtons.push($(this));
                        return;
                }

                if( thisId.indexOf('tool_') > -1 ) {

                        var toolName = thisId.substr(5,thisId.length);
                        if( toolName.toString() == fbCanvasObj.currentAction.toString() ) {
                                tool_on = false;
                                fbCanvasObj.clear_selection_button ();
                        } else {
                                fbCanvasObj.currentAction = thisId.substr(5,thisId.length) ;
                                tool_on = true;
                                $(this).addClass('toolSelected');
                        }
                        fbCanvasObj.handleToolUpdate(toolName, tool_on);
                        fbCanvasObj.refreshCanvas();
                }

                if( thisId.indexOf('action_') > -1 ) {
                fbCanvasObj.name=thisId.substr( 7, thisId.length );
                        switch( thisId.substr( 7, thisId.length )) {
                                case 'copy':
                                        fbCanvasObj.addUndoState();
                                        fbCanvasObj.copyBars() ;
                                        fbCanvasObj.refreshCanvas() ;
                                        break ;
                                case 'delete':
                                        fbCanvasObj.addUndoState();
                                        fbCanvasObj.deleteSelectedBars() ;
                                        fbCanvasObj.refreshCanvas() ;
                                        break ;
                                case 'join':
                                        fbCanvasObj.addUndoState();
                                        fbCanvasObj.joinSelected() ;
                                        fbCanvasObj.refreshCanvas() ;
                                        break ;
                                case 'setUnitBar':
                                        fbCanvasObj.addUndoState();
                                        fbCanvasObj.setUnitBar() ;
                                        fbCanvasObj.refreshCanvas() ;
                                        break ;
                                case 'measure':
                                        fbCanvasObj.addUndoState();
                                        fbCanvasObj.measureBars() ;
                                        fbCanvasObj.refreshCanvas() ;
                                        break ;
                                case 'make':
                                        fbCanvasObj.addUndoState();
                                        fbCanvasObj.make() ;
                                        fbCanvasObj.refreshCanvas() ;
                                        break ;
                                case 'breakApart':
                                        fbCanvasObj.addUndoState();
                                        fbCanvasObj.breakApartBars() ;
                                        fbCanvasObj.refreshCanvas() ;
                                        break ;
                                case 'clearSplits':
                                        fbCanvasObj.addUndoState();
                                        fbCanvasObj.clearSplits() ;
                                        fbCanvasObj.refreshCanvas();
                                        break ;
                                case 'pullOutSplit':
                                        fbCanvasObj.addUndoState();
                                        fbCanvasObj.pullOutSplit();
                                        fbCanvasObj.refreshCanvas();
                                        break ;
                                case 'undo':
                                        fbCanvasObj.undo();
                                        fbCanvasObj.refreshCanvas() ;
                                        break ;
                                case 'redo':
                                        fbCanvasObj.redo();
                                        fbCanvasObj.refreshCanvas();
                                        break;
                                case 'save':
                                        fbCanvasObj.save();
                                        break;
                                case 'open':
                                        SaveScreen();
                                        resetFormElement($("#files"));
                                        fbCanvasObj.openFileDialog();
                                        break;
                                        case 'print':
                                        fbCanvasObj.print_canvas();
                                        break ;
                                case 'clearAll':
                                    SaveScreen();
                                        location.reload();
                                        break;
                                case 'show':
                                        showAllButtons();
                                        break;
                                        case 'previous':
                                        previousSelectFile();
                                        break;
                                                case 'next':
                                                nextSelectFile();
                                                        break;

                        }

                }

                if( thisId.indexOf('window_') > -1 ) {
                        switch( thisId.substr( 7, thisId.length )) {
                                case 'label':
                                        fbCanvasObj.addUndoState();
                                        fbCanvasObj.editLabel() ;
                                        break ;
                                case 'split':
                                        fbCanvasObj.addUndoState();
                                        fbCanvasObj.split(splitWidgetObj) ;
                                        break ;
                                case 'iterate':
                                        fbCanvasObj.addUndoState();
                                        fbCanvasObj.iterate() ;
                                        break ;
                                case 'properties':
                                        fbCanvasObj.properties();
                                        break ;
                        }
                }

        }) ;


        $(document).keydown(function(e) {

                if( e.which == 16 ) {
                        Utilities.shiftKeyDown = true ;
                        fbCanvasObj.refreshCanvas();
                }
        });
        $(document).keyup(function(e) {
                if( e.which == 16 ) {
                        Utilities.shiftKeyDown = false ;
                        fbCanvasObj.refreshCanvas();
                }

                if( e.ctrlKey && e.keyCode==80) {
                        fbCanvasObj.properties();
                        fbCanvasObj.refreshCanvas();
                }

                if( e.ctrlKey && e.keyCode==83) {
                        fbCanvasObj.save();
                        fbCanvasObj.refreshCanvas();
                }

                if( e.ctrlKey && e.keyCode==72) {
                        //$( "#dialog-hidden" ).dialog('open');
                        if(Utilities.ctrlKeyDown){
                                showButton("tool_hide");
                                showButton("action_show");
                                Utilities.ctrlKeyDown=false;
                        } else {
                                Utilities.ctrlKeyDown =true;
                                hideButton("tool_hide");
                hideButton("action_show");
            }
            fbCanvasObj.clear_selection_button();
            fbCanvasObj.refreshCanvas();
        }
        if( e.ctrlKey && e.keyCode==46) {
            fbCanvasObj.addUndoState();
            fbCanvasObj.deleteSelectedBars() ;
            fbCanvasObj.refreshCanvas() ;
        }

    });

    $('#labelInput').keyup( function( e ) {
        if( e.which == 13 ) {
            fbCanvasObj.saveLabel( $('#labelInput').val(), Utilities.USE_CURRENT_SELECTION ) ;
            fbCanvasObj.hideEditLabel() ;
            fbCanvasObj.refreshCanvas();
        }
    }) ;

    // This gets triggered after we have already cleared out the selection,
    // so we need to have a way to be sure the LAST selection gets the label.
    $('#labelInput').blur( function() {
        fbCanvasObj.saveLabel( $('#labelInput').val(), Utilities.USE_LAST_SELECTION ) ;
        fbCanvasObj.hideEditLabel() ;
    }) ;



    $( "#dialog-splits" ).dialog({
            height: 300,
            width: 400,
            resizable: false,
            modal: true,
            buttons: [
                {
                    text: "Ok",
                    click: function() {
                        var num_splits = $("#split-slider-field").val();
                        var whole = $("input[type='radio'][name='whole_part']:checked").val();
                        var direction="Vertical";
                        if(Utilities.flag[1])
                        {
                            direction =  $("input[type='radio'][name='vert_horiz']:checked").val();
                        }

                        fbCanvasObj.makeSplits(num_splits, direction, whole);
                        $( this ).dialog( "close" );
                    }
                },
                {
                    text: "Cancel",
                    click: function() {
                        $( this ).dialog( "close" );
                    }
                }
            ],
            autoOpen: false
    });

    $( "#dialog-properties" ).dialog({
            height: 500,
            width: 400,
            resizable: false,
            modal: true,
            buttons: [
                {
                    text: "Ok",
                    click: function() {
                        var create_checked = $("input[type='radio'][name='create']:checked").val();
                        splitWidgetObj.vertical=true;
                        if (create_checked == "Same") {
                            Utilities.flag[0]= true;
                        } else if (create_checked == "New") {
                            Utilities.flag[0]= false;}

                        var horiz_checked = $("input[type='radio'][name='two_split']:checked").val();
                        if (horiz_checked == "One_horiz") {
                            Utilities.flag[1]= false;
                            document.getElementById("radio_vert").style.display = 'none';
                        } else if (horiz_checked == "Two_horiz") {
                            Utilities.flag[1]= true;
                            document.getElementById("radio_vert").style.display = 'block';
                        }

                        var itterate_way_checked = $("input[type='radio'][name='two_ittr']:checked").val();
                        if (itterate_way_checked == "One_way") {
                            Utilities.flag[2]= false;
                            document.getElementById("iterate_vert-horiz").style.display = 'none';
                        } else if (itterate_way_checked == "Two_way") {
                            Utilities.flag[2]= true;
                            document.getElementById("iterate_vert-horiz").style.display = 'block';
                        }

                        var language_checked = $("input[type='radio'][name='lang']:checked").val();
                        switch(language_checked) {
                        case 'lang_eng':
                            Utilities.flag[3]= false;
                            document.getElementById('stylesheet').href='css/lang_eng.css';
                            break ;
                        case 'lang_tur':
                            Utilities.flag[3]= true;
                            document.getElementById('stylesheet').href='css/lang_tur.css';
                            break ;
                        }

                        $( this ).dialog( "close" );
                    }
                },
                {
                    text: "Cancel",
                    click: function() {
                        $( this ).dialog( "close" );
                    }
                }
            ],
            autoOpen: false
    });



    $( "#dialog-iterate" ).dialog({
            height: 300,
            width: 400,
            resizable: false,
            modal: true,
            buttons: [
                {
                    text: "Ok",
                    click: function() {
                        var num_iterate = $("#iterate-field").val();
                        if(!Utilities.flag[2])
                        {
                            direction="Horizontal";
                        }
                        else
                        {
                            var direction =  $("input[type='radio'][name='vert_horiz']:checked").val();
                        }
                        fbCanvasObj.makeIterations(num_iterate, direction);
                        $( this ).dialog( "close" );
                    }
                },
                {
                    text: "Cancel",
                    click: function() {
                        $( this ).dialog( "close" );
                    }
                }
            ],
            autoOpen: false
    });

$( "#dialog-make" ).dialog({
            height: 300,
            width: 400,
            resizable: false,
            modal: true,
            buttons: [
                {
                    text: "Ok",
                    click: function() {
                        var num_whole = parseFloat($("#whole-field").val());
                        var num_num = parseFloat($("#num-field").val());
                        var num_denum = parseFloat($("#denum-field").val());

                        if(!num_whole)
                        {
                            num_whole=0;
                        }
                        if(!num_denum)
                        {
                            num_denum=1;
                        }
                        if(!num_num)
                        {
                            num_num=0;
                        }
                        num_frac=num_whole + (num_num/num_denum);
                        if (!num_frac)
                        {
                            alert("Please input fraction!");
                        }
                        else
                        {
                            fbCanvasObj.makeMake(num_frac);
                        }

                        document.getElementById('whole-field').value="";
                        document.getElementById('num-field').value="";
                        document.getElementById('denum-field').value="";
                        $( this ).dialog( "close" );
                    }
                },
                {
                    text: "Cancel",
                    click: function() {
                        $( this ).dialog( "close" );
                    }
                }
            ],
            autoOpen: false
    });

    $( "#split-slider" ).slider({
            value:2,
            min: 2,
            max: 20,
            step: 1,
            slide: function( event, ui ) {
```

```
                                $( "#split-slider-field" ).val( ui.value );
                        }
        });

        $( "#dialog-hidden" ).dialog({
                        height: 250,
                        width: 300,
                        modal: true,
                        buttons: [
                                {
                                        text: "Ok",
                                        click: function() {
//////////////////

                                                $( this ).dialog( "close" );
                                        }
                                },
                                {
                                        text: "Cancel",
                                        click: function() {
                                                $( this ).dialog( "close" );
                                        }
                                }
                        ],
                        autoOpen: false
        });

        $( "#dialog-file" ).dialog({
                        height: 250,
                        width: 300,
                        modal: true,
                        buttons: [
                                {
                                        text: "Cancel",
                                        click: function() {
                                                $( this ).dialog( "close" );
                                        }
                                }
                        ],
                        autoOpen: false
        });

        // --- Touch event helpers ---
        function getTouchPos(e, elem) {
                var touch = e.originalEvent.touches[0] || e.originalEvent.changedTouches[0];
                return {
                        x: touch.clientX - elem.position().left,
                        y: touch.clientY - elem.position().top
                };
        }
        function normalizeEvent(e, elem) {
                if (e.type.startsWith('touch')) {
                        var pos = getTouchPos(e, elem);
                        return {
                                clientX: pos.x + elem.position().left,
                                clientY: pos.y + elem.position().top,
                                which: 1,
                                ctrlKey: e.ctrlKey || false,
                                shiftKey: e.shiftKey || false,
                                preventDefault: function() { e.preventDefault(); }
                        };
                }
                return e;
        }

        // --- Canvas touch events ---
        $('#fbCanvas').on('touchstart', function(e) {
                e.preventDefault(); // Prevent scrolling/zooming
                var ne = normalizeEvent(e, $(this));
                // Simulate mousedown logic
                fbCanvasObj.check_for_drag = true;
                fbCanvasObj.cacheUndoState();
                updateMouseLoc(ne, $(this));
                updateMouseAction('mousedown');
                fbCanvasObj.mouseDownLoc = Point.createFromMouseEvent(ne, $(this));
```

```
            var b = fbCanvasObj.barClickedOn();
            var m = fbCanvasObj.matClickedOn();
            // Copy from the mouse handler above
            if( (fbCanvasObj.currentAction == 'bar') || (fbCanvasObj.currentAction == "mat")) {
                    fbCanvasObj.saveCanvas() ;
            } else if( fbCanvasObj.currentAction == 'repeat' ) {
                    fbCanvasObj.addUndoState();
                    b.repeat(fbCanvasObj.mouseDownLoc);
                    fbCanvasObj.refreshCanvas();
            } else {
                    if( b !== null ) {
                            if( $.inArray(b, fbCanvasObj.selectedBars) == -1) {
                                    if( !Utilities.shiftKeyDown ) {
                                            fbCanvasObj.clearSelection();
                                    }
                                    $.each( fbCanvasObj.selectedBars, function(index, bar) {
                                            bar.clearSplitSelection();
                                    });
                                    fbCanvasObj.barToFront(b);
                                    fbCanvasObj.selectedBars.push(b);
                                    b.isSelected = true;
                                    b.selectSplit(fbCanvasObj.mouseDownLoc);
                            } else {
                                    $.each( fbCanvasObj.selectedBars, function(index, bar) {
                                            bar.clearSplitSelection();
                                    });
                                    if( !Utilities.shiftKeyDown ) {
                                            b.selectSplit(fbCanvasObj.mouseDownLoc);
                                    } else {
                                            fbCanvasObj.removeBarFromSelection(b);
                                    }
                                    fbCanvasObj.barToFront(b);
                            }
                            if (fbCanvasObj.currentAction == "manualSplit") {
                                    fbCanvasObj.clearSelection();
                            }
                    } else if( m !== null ) {
                            if( $.inArray(m, fbCanvasObj.selectedMats) == -1) {
                                    if( !Utilities.shiftKeyDown ) {
                                            fbCanvasObj.clearSelection();
                                    }
                                    m.isSelected = true;
                                    fbCanvasObj.selectedMats.push(m);
                            } else {
                                    if( Utilities.shiftKeyDown ) {
                                            fbCanvasObj.removeMatFromSelection(m);
                                    }
                            }
                    } else {
                            fbCanvasObj.clearSelection();
                    }
                    fbCanvasObj.refreshCanvas();
            }
            ne.preventDefault();
    });

    $('#fbCanvas').on('touchmove', function(e) {
            e.preventDefault(); // Prevent scrolling/zooming
            var ne = normalizeEvent(e, $(this));
            fracEvent = ne;
            updateMouseLoc(ne, $(this));
            updateMouseAction('mousemove');
            var p = Point.createFromMouseEvent(ne, $(this));
            if (fbCanvasObj.currentAction == "manualSplit") {
                    fbCanvasObj.manualSplitPoint = p;
                    fbCanvasObj.refreshCanvas();
            }
            if(fbCanvasObj.mouseDownLoc !== null) {
                    fbCanvasObj.updateCanvas(p);
            }
            ne.preventDefault();
    });

    $('#fbCanvas').on('touchend', function(e) {
            e.preventDefault(); // Prevent scrolling/zooming
```

```
                var ne = normalizeEvent(e, $(this));
                updateMouseLoc(ne, $(this));
                updateMouseAction('mouseup');
                fbCanvasObj.mouseUpLoc = Point.createFromMouseEvent(ne, $(this));
                if( fbCanvasObj.currentAction == 'bar' ) {
                        fbCanvasObj.addUndoState();
                        fbCanvasObj.addBar() ;
                        fbCanvasObj.clear_selection_button ();
                } else if (fbCanvasObj.currentAction == 'mat') {
                        fbCanvasObj.addUndoState();
                        fbCanvasObj.addMat();
                        fbCanvasObj.clear_selection_button ();
                }
                if (fbCanvasObj.found_a_drag){
                        fbCanvasObj.finalizeCachedUndoState();
                        fbCanvasObj.check_for_drag = false;
                }
                fbCanvasObj.mouseUpLoc = null ;
                fbCanvasObj.mouseDownLoc = null ;
                fbCanvasObj.mouseLastLoc = null ;
                ne.preventDefault();
        });

        // --- Touch for color pickers and tool buttons ---
        $('.colorBlock, .colorBlock1').on('touchstart', function(e) {
                $(this).trigger('click');
                e.preventDefault();
        });
        $('a').on('touchstart', function(e) {
                $(this).trigger('click');
                e.preventDefault();
        });

});

function showAllButtons() {
        while(hiddenButtons.length >0) {
                thing = hiddenButtons.pop();
                thing.show();
        }
        hiddenButtons = [];
        hiddenButtonsName = [];
}

function SaveScreen() {
        var r=window.confirm("Do you want to save?");
        if (r==true)
        {
                fbCanvasObj.save();
        }
}

function showButton(item) {
    var cnt = 0;
    while(hiddenButtonsName.length >0) {
        if (hiddenButtonsName[cnt] === item) {
            var rem_but1=hiddenButtonsName.splice(cnt, 1);
            hiddenButtons.splice(cnt, 1);
        }
        else {
                cnt++;
        }
                if (hiddenButtonsName.length === cnt) {
                        $(document.getElementById(rem_but1)).show();
                        break;
                }
    }
}

function hideButton(item) {
        if (hiddenButtonsName.indexOf(item)<0) {
                hidden=document.getElementById(item) ;
    $(hidden).hide();
                 hiddenButtonsName.push(item);
                 hiddenButtons.push($(hidden));
        }
}

function handleFileSelect(event) {
        $( "#dialog-file" ).dialog("close");
        var files = event.target.files;
        if (files.length === 0) {return;}

//First attempt
        Utilities.file_list=event.target.files;
  Utilities.file_index=0;

        var aFile = files[0];
        readFileOpen(aFile);
  //
}

//First attempt
function handleListSelect(event) {
  Utilities.file_index= document.getElementById('id_filetext').selectedIndex;
        a_files = Utilities.file_list;

//        SaveScreen();
        fbCanvasObj.save();

        var aFileIndex=Utilities.file_index;
        var aFile = a_files[aFileIndex];
        readFileOpen(aFile);

}
//

function nextSelectFile(){
        //  SaveScreen();
                fbCanvasObj.save();

                var n_files = Utilities.file_list;
                Utilities.file_index = Utilities.file_index+1;
                document.getElementById('id_filetext').selectedIndex = Utilities.file_index;

                var nFileIndex=Utilities.file_index;
                var nFile = n_files[nFileIndex];
                readFileOpen(nFile);
}

function previousSelectFile(){
          //SaveScreen();
                fbCanvasObj.save();

                var p_files = Utilities.file_list;
                Utilities.file_index = Utilities.file_index-1;
                document.getElementById('id_filetext').selectedIndex = Utilities.file_index;

                var pFileIndex=Utilities.file_index;
                var pFile = p_files[pFileIndex];
                readFileOpen(pFile);
}

//First attempt
function readFileOpen(oFile){
        showAllButtons();

// reset undo and redo
        fbCanvasObj.mUndoArray = [];
        fbCanvasObj.mRedoArray = [];

        FBFileReader.readAsText(oFile);
        FBFileReader.onload = function (oFile) {
           fbCanvasObj.handleFileEvent(oFile);
        }
        showSelectList();
}
//


```

```
//First attempt
function showSelectList() {
        f_files = Utilities.file_list;
  var first = document.getElementById('id_filetext');
  var b_title= document.getElementById('bar_titles');
        var file_length = f_files.length;
        var select_length = document.getElementById('id_filetext').selectedIndex;
        var s_files = Utilities.file_list[Utilities.file_index];
        select_length = select_length + 1;
        document.title =  s_files.name;
        b_title.innerHTML=": "+s_files.name;

  if(file_length===1){
                hideButton("id_filetext");
                hideButton("action_previous");
                hideButton("action_next");
        }
        else if (file_length===select_length){
                showButton("id_filetext");
                showButton("action_previous");
                hideButton("action_next");
        }
        else if (select_length===1 || select_length===0){
                showButton("id_filetext");
                hideButton("action_previous");
                showButton("action_next");
        }
        else {
                showButton("id_filetext");
          showButton("action_previous");
          showButton("action_next");
        }

        first.innerHTML='';
        for (var i=0, f1; f1=f_files[i]; i++) {
                        if (s_files.name !== f1.name ) {
                                        first.innerHTML=first.innerHTML+'<option value="' + f1.name +
                                        ↪  '">' + f1.name +'</option>';
                        }
                        else {
                                first.innerHTML=first.innerHTML+'<option value="' + f1.name +
                                ↪  '"selected>' + f1.name +'</option>';
                        }
        }
}
//


function resetFormElement(e) {
  e.wrap('<form>').closest('form').get(0).reset();
  e.unwrap();
}


// for debugging

function updateMouseLoc(e, elem) {
        x = e.clientX - elem.position().left ;
        y = e.clientY - elem.position().top ;
        offsetX = elem.offset().left;
        offsetY        = elem.offset().top;
        /*
        $('#mouseLoc').text(x + ', ' + y + ' | ' + offsetX  + ', ' + offsetY + ' | ' + window.pageXOffset
        ↪  + ', ' + window.pageYOffset );
        */
}

function updateMouseAction(actionName) {
        /*
        $('#mouseAction').text(actionName) ;
        */
}
```

2.1.2 FractionBarsCanvas.js

```
// Copyright University of Massachusetts Dartmouth 2013
//
// Designed and built by James P. Burke and Jason Orrill
// Modified and developed by Hakan Sandir
//
// This Javascript version of Fraction Bars is based on
// the Transparent Media desktop version of Fraction Bars,
// which in turn was based on the original TIMA Bars software
// by John Olive and Leslie Steffe.
// We thank them for allowing us to update that product.



function FractionBarsCanvas(canvasContext) {
        this.context = canvasContext ;
//        this.currentTool = '' ;
        this.currentAction = '' ;
        this.canvasState = null ;
        this.currentFill = '#FFFF66' ;
//        this.barFill = '#FFFF66' ;
        this.matFill = '#888888' ;
        this.mouseDownLoc = null ;
        this.mouseUpLoc = null ;
        this.mouseLastLoc = null ;

        this.bars = [] ;
        this.mats = [] ;
        this.selectedBars = [] ;
        this.selectedMats = [] ;
        this.lastSelectedBars = [] ;
        this.lastSelectedMats = [] ;
        this.unitBar = null ;
        this.context.fillStyle = this.currentFill ;
        this.context.font = '9pt Verdana' ;

        this.mUndoArray = [];
        this.mRedoArray = [];

        this.check_for_drag = false; // These two values are used to check for a drag so that we can
        this.found_a_drag = false;   // store an undo state before a drag, and register it when we know the
    ↪   drag happened

        this.manualSplitPoint = null;
}

FractionBarsCanvas.prototype.addBar = function(a_bar) {
        var b = null;
        if (a_bar === null | a_bar === undefined) {
                b = Bar.createFromMouse(this.mouseDownLoc, this.mouseUpLoc, 'bar', this.currentFill) ;
        } else {
                b = a_bar;
        }

        this.bars.push(b);
        this.clearSelection();
        this.updateSelectionFromState();
        this.updateCanvas(this.mouseUpLoc);
        // this.isSelected = true;
        this.refreshCanvas();

        // Utilities.Log(this.bars.length);
};

FractionBarsCanvas.prototype.addMat = function() {
        var m = Mat.createFromMouse(this.mouseDownLoc, this.mouseUpLoc, 'mat', this.matFill) ;
        this.mats.push(m);
        this.updateCanvas(this.mouseUpLoc);
        this.refreshCanvas();
        // Utilities.Log(this.bars.length);
};

// Also copy mats
```

```
FractionBarsCanvas.prototype.copyBars = function() {
        if( this.selectedBars.length > 0 ) {
                for( var i = this.selectedBars.length-1; i >= 0; i-- ) {
                        this.bars.push( this.selectedBars[i].copy(true) ) ;
                        this.selectedBars[i].isSelected = false ;
                }
        }
        if( this.selectedMats.length > 0 ) {
                for(var j = this.selectedMats.length-1; j >= 0; j-- ) {
                        this.mats.push( this.selectedMats[j].copy(true) ) ;
                        this.selectedMats[j].isSelected = false ;
                }
        }
        this.updateSelectionFromState();
};

FractionBarsCanvas.prototype.breakApartBars = function() {
        var newBars ;
        if( this.selectedBars.length > 0 ) {
                for( var i = 0; i < this.selectedBars.length; i++ ) {
                        newBars = this.selectedBars[i].breakApart() ;
                        for( var j = 0; j < newBars.length; j++ ) {
                                this.bars.push( newBars[j] ) ;
                        }
                }

                // all splits in bars copied...delete the original selection
                this.deleteSelectedBars() ;
        }
};

FractionBarsCanvas.prototype.pullOutSplit = function() {
        var sel_split = null;

        for (var i = 0; i < this.selectedBars.length; i++) {
                if (this.selectedBars[i].selectedSplit !== null) {
                        sel_split = this.selectedBars[i].selectedSplit;
                        var newbar = Bar.createFromSplit(sel_split, this.selectedBars[i].x,
                        ↪  this.selectedBars[i].y);
                        this.addBar(newbar);
                }
        }
};

FractionBarsCanvas.prototype.clearSplits = function() {
        if( this.selectedBars.length > 0 ) {
                for( var i = 0; i < this.selectedBars.length; i++ ) {
                        this.selectedBars[i].clearSplits() ;
                }
        }
};


FractionBarsCanvas.prototype.split = function(sw) {
        // This function opens the dialog, but doesn't actually perform the splits.
        // makesplits is called directly from the OK button handler code in the .dialog definition in
        ↪  fractionbars.js

        if ((this.selectedBars.length > 1) || (this.selectedBars.length === 0)) {

////////////////////////
        if (Utilities.flag[3]) {
                                                        alert("Lütfen ayrıştırılacak bir kesir şeridi
                                                        ↪  seçiniz.");
                                                } else {
                                                        alert("Please select a bar to partition.");
                                                }
        //alert("Please select a bar to partition.");
//        alert(window.getComputedStyle($('.c_split_alert')[0], ':before').getPropertyValue('content'));

//alert(getComputedStyle(document.querySelector('.c_split_alert'), ':before').content);


        } else {
                if( this.selectedBars.length > 0 ) {
                        // Show dialog
                        sw.color = this.selectedBars[0].color;
                        $( "#dialog-splits" ).dialog('open');
                        sw.refreshCanvas();
                        for( var i = 0; i < this.selectedBars.length; i++ ) {
                        // Do something to each bar
                        }
                }
        }
};

//değişecek
FractionBarsCanvas.prototype.properties = function() {
        if (Utilities.flag[0] ) {
                document.getElementById("new").checked = false;
                document.getElementById("same").checked = true;
        } else {
                document.getElementById("new").checked = true;
                document.getElementById("same").checked = false;
                }
        if (Utilities.flag[1] ) {
                document.getElementById("two_horiz").checked = true;
                document.getElementById("one_horiz").checked = false;
        } else {
                document.getElementById("two_horiz").checked = false;
                document.getElementById("one_horiz").checked = true;
                }
        document.getElementById("vert").checked=true;
        document.getElementById("horiz").checked=false;
        $( "#dialog-properties" ).dialog('open');
};


FractionBarsCanvas.prototype.makeSplits = function(num_splits, vert_horiz, whole_part) {
        var vert_truth = (vert_horiz === "Vertical");
        if( this.selectedBars.length > 0 ) {
                if (whole_part === "Whole") {
                        for( var i = 0; i < this.selectedBars.length; i++ ) {
                        // Do something to each bar
                                this_bar = this.selectedBars[i];
                                // alert(num_splits);
                                // this_bar.equalSplits(num_splits);


                                this_bar.wholeBarSplits(num_splits, vert_truth);
                        }
                } else {
                        if((this.selectedBars[0].splits.length === 0) ||
                        ↪  (this.selectedBars[0].selectedSplit === null)) {
                                // No splits, or no selected split, so treat this like a whole bar split

                                this.selectedBars[0].wholeBarSplits(num_splits, vert_truth);
                        } else {

                                this.selectedBars[0].splitSelectedSplit(num_splits, vert_truth);
                        }
                }
                this.refreshCanvas();
        }
};


FractionBarsCanvas.prototype.iterate = function(iw) {
        // This function opens the dialog, but doesn't actually perform the iteration.
        // makeIterations is called directly from the OK button handler code in the .dialog definition in
        ↪  fractionbars.js

        if ((this.selectedBars.length > 1) || (this.selectedBars.length === 0)) {
                if (Utilities.flag[3]) {
                                                        alert("Lütfen yineleme işlemi yapabilmek için bir
                                                        ↪  kesir şeridi seçiniz.");
                                                } else {
                                                        alert("Please select exactly one bar to
                                                        ↪  iterate.");
                                                }
                //alert("Please select exactly one bar to iterate.");
        } else {
                if( this.selectedBars.length > 0 ) {
                        // Show dialog
                        $( "#dialog-iterate" ).dialog('open');
                        //for( var i = 0; i < this.selectedBars.length; i++ ) {
                        // Do something to each bar
                        //}
                }
        }
};


FractionBarsCanvas.prototype.make = function(iw) {
        // This function opens the dialog, but doesn't actually perform the iteration.
        // makeIterations is called directly from the OK button handler code in the .dialog definition in fractionbars.js

        if ((this.selectedBars.length > 1) || (this.selectedBars.length === 0)) {
                if (Utilities.flag[3]) {
                                                        alert("Lütfen yeni bir şerit yapabilmek için bir kesir şeridi seçiniz.");
                                                } else {
                                                        alert("Please select exactly one bar to make new bar.");
                                                }
                //alert("Please select exactly one bar to iterate.");
        } else {
                if( this.selectedBars.length > 0 ) {
                        // Show dialog
                        $( "#dialog-make" ).dialog('open');

                }
        }
};



FractionBarsCanvas.prototype.makeIterations = function(num_iterations, vert_horiz) {
        var vert_truth = (vert_horiz === "Vertical");
        if( this.selectedBars.length > 0 ) {

                if(!Utilities.flag[0]){this.copyBars();}

                this.selectedBars[0].iterate(num_iterations, vert_truth);

                this.refreshCanvas();
        }
};


FractionBarsCanvas.prototype.makeMake = function(num_frac) {
        if( this.selectedBars.length > 0 ) {

                this.bars.push( this.selectedBars[0].makeNewCopy(num_frac) ) ;

                this.refreshCanvas();
        }
};

FractionBarsCanvas.prototype.measureBars = function() {
        if( this.selectedBars.length > 0 ) {
                for( var i = this.selectedBars.length-1; i >= 0; i-- ) {
                        this.selectedBars[i].fraction = Utilities.createFraction(this.selectedBars[i].size, this.unitBar.size) ;
                }
        }
};

FractionBarsCanvas.prototype.clearAllMeasurements = function() {
        for( var i = 0; i < this.bars.length; i++ ) {
                this.bars[i].isUnitBar = false ;
                this.bars[i].fraction = '' ;
        }
};
```

```


FractionBarsCanvas.prototype.setUnitBar = function() {
        this.clearAllMeasurements() ;
        if( this.selectedBars.length == 1 ) {
                this.selectedBars[0].isUnitBar = true ;
                this.selectedBars[0].fraction = '' ;
                this.unitBar = this.selectedBars[0] ;
        }
};

FractionBarsCanvas.prototype.editLabel = function() {
        var canvasPos = $('#fbCanvas').position() ;

        if( this.selectedBars.length == 1 ) {
                var labelDiv = $('#labelInput') ;
                $('#labelInput').css('position', 'absolute') ;
                $('#labelInput').css('width', this.selectedBars[0].w - 13) ;

                $('#labelInput').css('top', canvasPos.top + this.selectedBars[0].y +
                ↪  this.selectedBars[0].h - labelDiv.outerHeight() - 4) ;
                $('#labelInput').css('left', canvasPos.left + this.selectedBars[0].x + 5) ;
                $('#labelInput').val( this.selectedBars[0].label ) ;

                $('#labelInput').show() ;
                $('#labelInput').focus() ;

        }
};

FractionBarsCanvas.prototype.hideEditLabel = function() {
        $('#labelInput').hide() ;
};

FractionBarsCanvas.prototype.saveLabel = function(labelText, selectionType) {
        var barSelection = [] ;
        if( selectionType == Utilities.USE_CURRENT_SELECTION ) {
                barSelection = this.selectedBars ;
        } else {
                barSelection = this.lastSelectedBars ;
        }

        if( barSelection.length == 1 ) {
                barSelection[0].label = labelText ;
        }
        this.lastSelectedBars = [] ;
        this.refreshCanvas() ;
};

// Deletes both bars and mats that are selected
FractionBarsCanvas.prototype.deleteSelectedBars = function() {
        var newBars = [];
        var unitBarDeleted = false ;

        for( var i = 0; i < this.bars.length; i++ ) {
                if( !this.bars[i].isSelected ) {
                        newBars.push( this.bars[i] ) ;
                } else {
                        if( this.bars[i].isUnitBar ) {
                                unitBarDeleted = true ;
                        }
                }
        }
        this.bars = newBars ;
        if( unitBarDeleted ) {
                this.clearAllMeasurements() ;
        }
        var newMats = [];
        for (i = 0; i < this.mats.length; i++) {
                if( !this.mats[i].isSelected ) {
                        newMats.push( this.mats[i] ) ;
                }
        }
        this.mats = newMats;
};
```

```

// Works on bars and mats together
FractionBarsCanvas.prototype.updateSelectionFromState = function() {
        this.selectedBars = [];
        for( var i = 0; i < this.bars.length; i++ ) {
                if( this.bars[i].isSelected ) {
                        this.selectedBars.push( this.bars[i] ) ;
                }
        }
        this.selectedMats = [];
        for(i = 0; i < this.mats.length; i++ ) {
                if( this.mats[i].isSelected ) {
                        this.selectedMats.push( this.mats[i] ) ;
                }
        }
};

FractionBarsCanvas.prototype.findBarForPoint = function(p) {
        for( var i = this.bars.length-1; i >= 0; i-- ) {
                if( p.x > this.bars[i].x &&
                        p.x < this.bars[i].x + this.bars[i].w &&
                        p.y > this.bars[i].y &&
                        p.y < this.bars[i].y + this.bars[i].h) {

                        return(this.bars[i]);
                }
        }
        return null;
};

FractionBarsCanvas.prototype.findSplitForPoint = function(p) {
        var the_bar = this.findBarForPoint(p);
        if (the_bar !== null) {
                return (the_bar.findSplitForPoint(p));
        } else {
                return (null);
        }
};

FractionBarsCanvas.prototype.findSomethingForPoint = function(p) {
        // Returns either a bar or a split that matches the point. Or null if no match.
        var the_bar = this.findBarForPoint(p);
        if (the_bar !== null) {
                var the_split = the_bar.findSplitForPoint(p);
                if (the_split !== null) {
                        return (the_split);
                } else {
                        return (the_bar);
                }
        } else {
                return (null);
        }
};

FractionBarsCanvas.prototype.barClickedOn = function() {
        for( var i = this.bars.length-1; i >= 0; i-- ) {
                // Utilities.log(i);
                if( this.mouseDownLoc.x > this.bars[i].x &&
                        this.mouseDownLoc.x < this.bars[i].x + this.bars[i].w &&
                        this.mouseDownLoc.y > this.bars[i].y &&
                        this.mouseDownLoc.y < this.bars[i].y + this.bars[i].h)
                {
                        // this.bars[i].isSelected = true ;
                        if (this.currentAction == "manualSplit" ) {
                                this.addUndoState();
                                if (Utilities.flag[1] ) {
                                        split_key=Utilities.shiftKeyDown;
                                } else{
                                        split_key=false;
                                        }

                                this.bars[i].splitBarAtPoint(this.mouseDownLoc, split_key);

                        } else {
                                this.bars[i].selectSplit(this.mouseDownLoc);
                        }
                        return this.bars[i] ;
                }
        }
        return null ;
};

FractionBarsCanvas.prototype.barToFront = function(bar) {

        var new_list = [];

        for (var i = 0; i < this.bars.length; i++) {
                if (bar !== this.bars[i]) {
                        new_list.push(this.bars[i]);
                }
        }
        new_list.push(bar);
        this.bars = new_list;

};

FractionBarsCanvas.prototype.matClickedOn = function() {
        for( var i = this.mats.length-1; i >= 0; i-- ) {
                // Utilities.log(i);
                if( this.mouseDownLoc.x > this.mats[i].x &&
                        this.mouseDownLoc.x < this.mats[i].x + this.mats[i].w &&
                        this.mouseDownLoc.y > this.mats[i].y &&
                        this.mouseDownLoc.y < this.mats[i].y + this.mats[i].h)
                {
//                        this.mats[i].isSelected = true ;
                        return this.mats[i] ;
                }
        }
        return null ;
};

// CLear for bars and mats
FractionBarsCanvas.prototype.clearSelection = function() {
        $.each( this.bars, function(index, bar) {
                bar.isSelected = false ;
                bar.clearSplitSelection();
        });
        this.lastSelectedBars = this.selectedBars ;
        this.selectedBars = [] ;

        $.each( this.mats, function(index, mat) {
                mat.isSelected = false ;
        });
        this.lastSelectedMats = this.selectedMats ;
        this.selectedMats = [] ;
};

// CLear for bars and mats
FractionBarsCanvas.prototype.removeBarFromSelection = function(bar) {

        var new_list = [];

        for (var i = 0; i < this.selectedBars.length; i++) {
                if (bar !== this.selectedBars[i]) {
                        new_list.push(this.selectedBars[i]);
                }
        }

        this.selectedBars = new_list;
        bar.isSelected = false;
        bar.clearSplitSelection();

};

FractionBarsCanvas.prototype.removeMatFromSelection = function(mat) {

        var new_list = [];

        for (var i = 0; i < this.selectedMats.length; i++) {
                if (mat !== this.selectedMats[i]) {
                        new_list.push(this.selectedMats[i]);
                }
        }

        this.selectedMats = new_list;
        mat.isSelected = false;

};


FractionBarsCanvas.prototype.joinSelected = function() {
        // TODO: bulletproof this
        // TODO: update this to allow for more than two bars to be joined.
        if ((this.selectedBars.length > 2) || (this.selectedBars.length === 1) ||
        ↪  (this.selectedMats.length > 0)) {
                if (Utilities.flag[3]) {
                                                        alert("Birleştirme işlemi yapabilmek için lütfen
                                                        ↪  iki kesir şeridi seçiniz.");
                                                } else {
                                                        alert("Please select exactly two bars (and no
                                                        ↪  mats) before attempting to Join.");
                                                }
                //alert("Please select exactly two bars (and no mats) before attempting to Join.");
                {return;}
        }
        var success = this.selectedBars[0].join(this.selectedBars[1]);

        if (success) {
                this.selectedBars[0].isSelected = false ;
                this.deleteSelectedBars();
                this.updateSelectionFromState();
        }

};

FractionBarsCanvas.prototype.setupBarRepeats = function() {
        // For every bar, jsut set its repeatUnit. So that Repeat can work correctly.
        for (var i = this.bars.length - 1; i >= 0; i--) {
                this.bars[i].setRepeatUnit();
        }
};


FractionBarsCanvas.prototype.unsetBarRepeats = function() {
        // For every bar, jsut set its repeatUnit. So that Repeat can work correctly.
        for (var i = this.bars.length - 1; i >= 0; i--) {
                this.bars[i].repeatUnit = null;
        }
};


FractionBarsCanvas.prototype.handleToolUpdate = function(tool_name, tool_on) {
        // This is the Canvas' chance to do something when a tool switched on or off
        // We are given the name of the tool, and a Boolean value of whether it was turned on or off.

        switch(tool_name) {
                case 'repeat':
                        if (tool_on) {
                                this.setupBarRepeats();
                        } else {
                                this.unsetBarRepeats();
                        }
        }
};



FractionBarsCanvas.prototype.drawRect = function(p1, p2) {
        if (this.currentAction == "bar")
                this.context.fillStyle = this.currentFill;
        else if (this.currentAction == "mat")
                this.context.fillStyle = this.matFill;
        var w = Math.abs(p2.x - p1.x) ;
        var h = Math.abs(p2.y - p1.y) ;
        var p = Point.min( p1, p2 ) ;
        this.context.fillRect(p.x + 0.5, p.y + 0.5, w, h) ;
        this.context.strokeRect(p.x + 0.5, p.y + 0.5, w, h) ;
};

/*
FractionBarsCanvas.prototype.manualSplitXORDraw = function(the_point) {
        this.context.strokeStyle="#FF0000";
        this.context.globalCompositeOperation="xor";
        this.context.strokeRect(the_point.x-50, the_point.y-50, 100,100 ) ;
        this.context.strokeRect(the_point.x-50, the_point.y-50, 100,100 ) ;
        this.context.globalCompositeOperation="source-over";

}
*/

FractionBarsCanvas.prototype.drawBar = function(b) {

        this.context.fillStyle = b.color;
        this.context.fillRect(b.x + 0.5, b.y + 0.5, b.w, b.h) ;

        this.context.strokeStyle = '#FF0000' ;
        if( b.splits.length > 0 ) {
                for( i = 0; i < b.splits.length; i++ ) {
                        this.context.fillStyle = b.splits[i].color;
                        this.context.fillRect( b.x + b.splits[i].x + 0.5, b.y + b.splits[i].y + 0.5,
                        ↪  b.splits[i].w, b.splits[i].h ) ;
                        this.context.strokeRect( b.x + b.splits[i].x + 0.5, b.y + b.splits[i].y + 0.5,
                        ↪  b.splits[i].w, b.splits[i].h ) ;
                        if (b.splits[i].isSelected === true) {
                                var xcenter = b.splits[i].x+(b.splits[i].w /2);
                                var ycenter = b.splits[i].y+(b.splits[i].h /2);
                                this.context.strokeRect(b.x+xcenter-2, b.y+ycenter-2, 4, 4);
                        }
                }
        }

        this.context.fillStyle = b.color;

        this.context.strokeStyle = '#000000' ;
        if( b.isSelected ) {
                this.context.lineWidth = 2.5 ;
        }

        this.context.strokeRect(b.x + 0.5, b.y + 0.5, b.w, b.h) ;

        this.context.lineWidth = 1;
        this.context.fillStyle = '#000000' ;

        if( b.isUnitBar ) {
                this.context.fillText('Unit Bar', b.x, b.y + b.h + 15) ;
        }

        if ((this.currentAction == "manualSplit") && (this.manualSplitPoint !== null)) {
                var asplit = this.findSplitForPoint(this.manualSplitPoint);
                var abar = this.findBarForPoint(this.manualSplitPoint);
                var x_offset = 0;
                var y_offset = 0;
                var thing = null;

if (Utilities.flag[1] ) {
                                split_key=!Utilities.shiftKeyDown;
                        } else
                        { split_key=true;
                        }
                if (asplit !== null) {
                        x_offset = abar.x;
                        y_offset = abar.y;
                        thing = asplit;
                } else {
                        thing = abar;
                }
                if ((thing !== null) && !((asplit === null) && (abar !== null) && (abar.splits.length !==
                ↪  0))) {
                        // The above statement is complex because it checks for the condition where a user
                        ↪  can click
                        // exactly between existing splits.
                        var savestroke = this.context.strokeStyle;
                        this.context.strokeStyle = '#FF0000' ;

                        if (!split_key) {
                                this.context.strokeRect( thing.x+x_offset, this.manualSplitPoint.y,
                                ↪  thing.w,0 ) ;
                        } else {
                                this.context.strokeRect( this.manualSplitPoint.x, thing.y+y_offset, 0,
                                ↪  thing.h ) ;
                        }
                        this.context.strokeStyle = savestroke;
                }
        }


        var fractionStringMetrics = this.context.measureText( b.fraction ) ;
        this.context.fillText( b.fraction, b.x + b.w - fractionStringMetrics.width - 5, b.y - 5) ;

        var labelStringMetrics = this.context.measureText( b.label ) ;
        this.context.fillText( b.label, b.x + 5, b.y + b.h - 5) ;

        this.context.fillStyle = this.currentFill ;

};

FractionBarsCanvas.prototype.drawMat = function(b) {

        this.context.fillStyle = b.color;
        this.context.fillRect(b.x + 0.5, b.y + 0.5, b.w, b.h) ;

        this.context.strokeStyle = '#FF0000' ;

        this.context.strokeStyle = '#000000' ;
        if( b.isSelected ) {
                this.context.lineWidth = 2.5 ;
        }

        this.context.strokeRect(b.x + 0.5, b.y + 0.5, b.w, b.h) ;

        this.context.lineWidth = 1;
        this.context.fillStyle = '#000000' ;

        this.context.fillStyle = this.currentFill ;

};

FractionBarsCanvas.prototype.updateCanvas = function(currentMouseLoc) {

        if ((this.currentAction == 'bar') || (this.currentAction == 'mat')) {
                if( this.canvasState !== null ) {
                        this.context.putImageData(this.canvasState,0,0);
                }
                if( this.mouseDownLoc !== null ) {
                        this.drawRect(this.mouseDownLoc, currentMouseLoc) ;
                }
        } else if (this.currentAction == "manualSplit") {
                // this.calculateSplitLine(currentMouseLoc);
                this.manualSplitPoint = currentMouseLoc;
        } else {
                // we're dragging stuff around
                this.drag(currentMouseLoc);
        }
};

FractionBarsCanvas.prototype.saveCanvas = function() {
        this.canvasState = this.context.getImageData(0,0,1000,600) ;
};

FractionBarsCanvas.prototype.refreshCanvas = function() {
        this.context.clearRect(0,0,1000,600);
        for( var i = 0; i < this.mats.length; i++ ) {
                this.drawMat(this.mats[i]);
        }
        for( i = 0; i < this.bars.length; i++ ) {
                this.drawBar(this.bars[i]);
        }
};

FractionBarsCanvas.prototype.setFillColor = function(fillColor) {
        this.currentFill = fillColor ;
        this.context.fillStyle = this.currentFill ;
};

FractionBarsCanvas.prototype.updateColorsOfSelectedBars = function() {
        var i;
        if (this.selectedBars.length > 0) {
                this.addUndoState();
        }
        for (i in this.selectedBars) {
                if (this.selectedBars[i].hasSelectedSplit()) {
                        this.selectedBars[i].updateColorOfSelectedSplit(this.currentFill);
                } else {
                        this.selectedBars[i].color = this.currentFill;
                }
        }
        this.refreshCanvas();
};

FractionBarsCanvas.prototype.clearMouse = function() {
        this.mouseDownLoc = null ;
        this.mouseUpLoc = null ;
};

FractionBarsCanvas.prototype.drag = function(currentLoc) {
        if( this.mouseLastLoc === null || typeof(this.mouseLastLoc) == 'undefined') {
                this.mouseLastLoc = this.mouseDownLoc ;
        }

        for( var i = 0; i < this.selectedBars.length; i++ ) {
                this.selectedBars[i].x = this.selectedBars[i].x + currentLoc.x - this.mouseLastLoc.x ;
                this.selectedBars[i].y = this.selectedBars[i].y + currentLoc.y - this.mouseLastLoc.y ;

        }

        for(i = 0; i < this.selectedMats.length; i++ ) {
                this.selectedMats[i].x = this.selectedMats[i].x + currentLoc.x - this.mouseLastLoc.x ;
                this.selectedMats[i].y = this.selectedMats[i].y + currentLoc.y - this.mouseLastLoc.y ;

        }

        if(this.check_for_drag) {
                this.found_a_drag = true;
                this.check_for_drag = false;
        }

        this.mouseLastLoc = currentLoc ;

        this.refreshCanvas() ;

};

FractionBarsCanvas.prototype.addUndoState = function() {

        var newstate = new CanvasState(this);
        newstate.grabBarsAndMats();
        this.mUndoArray.push(newstate);  // Push new state onto the stack

        while (this.mUndoArray.length > 100) {
                this.mUndoArray.shift();  // Shift states off the bottom of the undo stack
        }

        this.mRedoArray = []; // When an undoable event happens, it clears the redo stack.

};

FractionBarsCanvas.prototype.clear_selection_button = function() {

                        fbCanvasObj.clearMouse();
                        fbCanvasObj.clearSelection();
                        $("[id^='tool_']").removeClass('toolSelected');
                        fbCanvasObj.currentAction = '' ;

};
FractionBarsCanvas.prototype.cacheUndoState = function() {

        this.CachedState = new CanvasState(this);
        this.CachedState.grabBarsAndMats();

};


FractionBarsCanvas.prototype.finalizeCachedUndoState = function() {

        if(this.CachedState !== null){
                this.mUndoArray.push(this.CachedState);  // Push new state onto the stack

                while (this.mUndoArray.length > 100) {
                        this.mUndoArray.shift();  // Shift states off the bottom of the undo stack
                }

                this.mRedoArray = []; // When an undoable event happens, it clears the redo stack.
        }

        this.check_for_drag = false;
        this.found_a_drag = false;

};

FractionBarsCanvas.prototype.undo = function() {
        // Store current state in Redo stack
        // Pop an undo state off the stack
        // Restore undo state
        if (this.mUndoArray.length > 0) {

                var newstate = new CanvasState(this);
                newstate.grabBarsAndMats();
                this.mRedoArray.push(newstate);  // Push new state onto the stack

                this.restoreAState(this.mUndoArray.pop());

        }
};

FractionBarsCanvas.prototype.redo = function() {
        if (this.mRedoArray.length > 0) {

                var newstate = new CanvasState(this);
                newstate.grabBarsAndMats();
                this.mUndoArray.push(newstate);  // Push new state onto the stack

                this.restoreAState(this.mRedoArray.pop());

        }
};

FractionBarsCanvas.prototype.restoreAState = function(a_new_state) {
        // clear the bars and mats
        // copy bars and mats from the new state
        // set the unit bar, if any.

        var temp_bar;

        this.bars = [];
        this.mats = [];
        this.selectedBars = [];
        this.selectedMats = [];


        while (a_new_state.mBars.length >0) {
                temp_bar = a_new_state.mBars.shift();
                this.bars.push(temp_bar);
        }

        while (a_new_state.mMats.length >0) {
```

```
                        this.mats.push(a_new_state.mMats.shift());
                }

                this.unitBar = a_new_state.mUnitBar;
                if (this.unitBar !== null ) {
                        this.unitBar.isUnitBar = true;
                        this.unitBar.fraction = '1/1' ;
                }
                //this.updateSelectionFromState();
                this.clearSelection();

};



FractionBarsCanvas.prototype.save = function() {

        var newstate = new CanvasState(this);
        newstate.grabBarsAndMats();

        newstate.mFBCanvas = null;

        var state_string = JSON.stringify(JSON.decycle(newstate));

        // alert(state_string);
        // Utilities.log(state_string);
        /*
        var new_win = window.open("","_blank", "resizable=yes, scrollbars=yes, titlebar=yes, width=1000, height=500, top=10, left=10");
        new_win.document.title = "Save this in a file on your hard drive.";
        new_win.document.writeln("** Save this text to your hard drive. Right-click here and use 'Save as...' or 'Save page as...'");
        new_win.document.writeln("**");
        new_win.document.writeln(state_string);
        new_win.document.close();
        returns false if user does not save
        */
        try {
                var blob = new Blob([state_string], {type: "text/plain;charset=utf-8"});
                //var filename = window.prompt("File name:","FractionBarsSave.txt");

// first attempt
                var select_length = document.getElementById('id_filetext').selectedIndex;
                if(select_length<0)
                {
                        var filename = window.prompt("File name:","FractionBarsSave.txt");
                }
                else
                {
                        var filename = Utilities.file_list[Utilities.file_index].name;
                }
//

                if (filename!=null)
                  {
                        saveAs(blob, filename);
                  }
                  else
                          {
                                return false;
                          }


        }
        catch(e){
                if (Utilities.flag[3]) {
                                                        alert("Bu tarayıcı kaydetmeyi desteklememektedir. Tarayıcının \nHTML5 destekli olması gereklidir. \n\nEn iyi sonuç için lütfen Firefox, \nChrome, Safari ya da Internet Explorer tarayıcılarından birini kullanınız.");
                                                } else {
                                                        alert("This browser does not support saving.
                                                        ↪  \nHTML5 support is needed. \n\nFor best
                                                        ↪  results use the most recent Firefox, \nChrome,
                                                        ↪  Safari, or Internet Explorer browser.");

                                                }
                        //alert("This browser does not support saving. \nHTML5 support is needed. \n\nFor best
                        ↪  results use the most recent Firefox, \nChrome, Safari, or Internet Explorer browser.");
        }
};

FractionBarsCanvas.prototype.openFileDialog = function() {
        // Show dialog
        $( "#dialog-file" ).dialog('open');
};

FractionBarsCanvas.prototype.openSaveDialog = function() {
        // Show dialog
        var r=window.confirm("Do you want to save?");
if (r==true)
        {
                /*var res=this.save();
                if (res==false)
                {
                        break;
                }*/
        }
};
FractionBarsCanvas.prototype.handleFileEvent = function(file_event) {


        var file_contents = file_event.target.result;
        // var lines = file_contents.split("**");
        // var text_state = lines[2].replace(/(\r\n|\n|\r)/gm,"");

        var text_state = "";
        var something = null;

        try {
                text_state = file_contents.replace(/(\r\n|\n|\r)/gm,"");
                something = JSON.retrocycle(JSON.parse(text_state));
        } catch (e) {
                var txt = "An error has occurred. \n\n";
                txt += "Fraction Bars cannot open this file. \n\n";
                txt += e.message;
                alert(txt);
                return;
        }


        this.restoreBarsAndMatsFromJSON(something);
};

FractionBarsCanvas.prototype.restoreBarsAndMatsFromJSON = function(JSON_obj) {

        this.bars = [];
        this.mats = [];
        this.selectedBars = [];
        this.selectedMats = [];
        this.unitBar = null;
        len = 0;

        if( JSON_obj.mBars.length > 0 ) {
                for( var i = 0; i < JSON_obj.mBars.length; i++ ) {
                        len = this.bars.push( Bar.copyFromJSON(JSON_obj.mBars[i]) ) ;
                        if (this.bars[len-1].isUnitBar) {
                                this.unitBar = this.bars[len-1];
                                this.bars[len-1].fraction = "1/1";
                        }
                }
        }
        if( JSON_obj.mMats.length > 0 ) {
                for( var j = 0; j < JSON_obj.mMats.length; j++ ) {
                        this.mats.push( Mat.copyFromJSON(JSON_obj.mMats[j]) ) ;
                }
        }

//First attempt
        var hiddenButtonsName1 = JSON_obj.mHidden.slice(0);
        for( var ii = 0; ii < hiddenButtonsName1.length; ii++ ) {
                if (hiddenButtonsName.indexOf(hiddenButtonsName1[ii])<0) {
                        hidden=document.getElementById(hiddenButtonsName1[ii]) ;

                        $(hidden).hide();
                        hiddenButtonsName.push(hiddenButtonsName1[ii]);
                        hiddenButtons.push($(hidden));
                }
        }
//

        Utilities.ctrlKeyDown=true;
        Utilities.ctrlKeyDown=true;
        this.clearSelection();
        this.refreshCanvas();
};

FractionBarsCanvas.prototype.print_canvas = function (){
    var canvas=document.getElementById("fbCanvas");
        //var ctx=canvas.canvasContext;
        var win=window.open();
    win.document.write("<html><br><img src='"+canvas.toDataURL()+"'/></html>");
    //win.print();
        win.self.print();
  win.location.reload();
}

FractionBarsCanvas.prototype.exportHighResPNG = function(filename) {
        var scale = 3;
        var canvas = document.getElementById('fbCanvas');
        var w = canvas.width;
        var h = canvas.height;
        // Create a high-res offscreen canvas
        var exportCanvas = document.createElement('canvas');
        exportCanvas.width = w * scale;
        exportCanvas.height = h * scale;
        var exportCtx = exportCanvas.getContext('2d');
        // Scale context
        exportCtx.setTransform(scale, 0, 0, scale, 0, 0);
        // Redraw everything at high-res
        this.drawAllToContext(exportCtx);
        // Export PNG
        var dataURL = exportCanvas.toDataURL('image/png');
        var link = document.createElement('a');
        link.href = dataURL;
        link.download = filename || 'FractionBars.png';
        document.body.appendChild(link);
        link.click();
        document.body.removeChild(link);
};

// Helper to draw all content to a given context (for export)
FractionBarsCanvas.prototype.drawAllToContext = function(ctx) {
        ctx.clearRect(0, 0, ctx.canvas.width, ctx.canvas.height);
        for (var i = 0; i < this.mats.length; i++) {
                this.drawMatToContext(ctx, this.mats[i]);
        }
        for (var i = 0; i < this.bars.length; i++) {
                this.drawBarToContext(ctx, this.bars[i]);
        }
};

FractionBarsCanvas.prototype.drawBarToContext = function(ctx, b) {
        ctx.save();
        ctx.fillStyle = b.color;
        ctx.fillRect(b.x + 0.5, b.y + 0.5, b.w, b.h);
        ctx.strokeStyle = '#FF0000';
        if (b.splits.length > 0) {
                for (var i = 0; i < b.splits.length; i++) {
                        ctx.fillStyle = b.splits[i].color;
                        ctx.fillRect(b.x + b.splits[i].x + 0.5, b.y + b.splits[i].y + 0.5, b.splits[i].w,
                        ↪  b.splits[i].h);
                        ctx.strokeStyle = '#000000';
                        ctx.strokeRect(b.x + b.splits[i].x + 0.5, b.y + b.splits[i].y + 0.5,
                        ↪  b.splits[i].w, b.splits[i].h);
                        if (b.splits[i].isSelected === true) {
                                var xcenter = b.splits[i].x + (b.splits[i].w / 2);
                                var ycenter = b.splits[i].y + (b.splits[i].h / 2);
                                ctx.strokeRect(b.x + xcenter - 2, b.y + ycenter - 2, 4, 4);
                        }
                }
        }
        ctx.fillStyle = b.color;
        ctx.strokeStyle = '#000000';
        if (b.isSelected) {
                ctx.lineWidth = 2.5;
        }
        ctx.strokeRect(b.x + 0.5, b.y + 0.5, b.w, b.h);
        ctx.lineWidth = 1;
        ctx.fillStyle = '#000000';
        ctx.font = '9pt Verdana';
        if (b.isUnitBar) {
                ctx.fillText('Unit Bar', b.x, b.y + b.h + 15);
        }
        var fractionStringMetrics = ctx.measureText(b.fraction);
        ctx.fillText(b.fraction, b.x + b.w - fractionStringMetrics.width - 5, b.y - 5);
        var labelStringMetrics = ctx.measureText(b.label);
        ctx.fillText(b.label, b.x + 5, b.y + b.h - 5);
        ctx.fillStyle = this.currentFill;
        ctx.restore();
};

FractionBarsCanvas.prototype.drawMatToContext = function(ctx, b) {
        ctx.save();
        ctx.fillStyle = b.color;
        ctx.fillRect(b.x + 0.5, b.y + 0.5, b.w, b.h);
        ctx.strokeStyle = '#000000';
        ctx.strokeRect(b.x + 0.5, b.y + 0.5, b.w, b.h);
        ctx.restore();
};
```

2.2 Object Classes

2.2.1 Bar.js

```
// Copyright University of Massachusetts Dartmouth 2013
//
// Designed and built by James P. Burke and Jason Orrill
// Modified and developed by Hakan Sandir
//
// This Javascript version of Fraction Bars is based on
// the Transparent Media desktop version of Fraction Bars,
// which in turn was based on the original TIMA Bars software
// by John Olive and Leslie Steffe.
// We thank them for allowing us to update that product.


function Bar() {
        //TODO: convert x, y to a Point?
        this.x = null ;
        this.y = null ;
        this.w = null ;
        this.h = null ;
        this.size = null ;
        this.color = null ;
        this.splits = [] ;
        this.label = '' ;
        this.isUnitBar = false ;
        this.fraction = '' ;
        this.type = null ;
        this.isSelected = false ;
        this.repeatUnit = null;    // This is a copy of whatever the bar looks like at the moment "repeat"
        ↪  mode is turned on.
        this.selectedSplit = null;
}

Bar.prototype.measure = function(targetBar) {
        this.fraction = Utilities.createFraction( this.size, targetBar.size ) ;
};
Bar.prototype.clearMeasurement = function() {
        this.fraction = '' ;
};
Bar.prototype.drawMeasurement = function() {};


Bar.prototype.addSplit = function(x, y, w, h, c) {
        this.addSplitToList(this.splits, x, y, w, h, c);
};

Bar.prototype.addSplitToList = function(list, x, y, w, h, c) {
        var split = new Split(x, y, w, h, c) ;
        if( this.splits.length > 0 ) {
                for( i = 0; i < this.splits.length; i++ ) {
                        if( split.equals( this.splits[i] )) {
                                return ;
                        }
                }
        }
        list.push( split ) ;
};

Bar.prototype.clearSplits = function() {
        this.splits = [] ;
};

Bar.prototype.copySplits = function() {
        var splitsCopy = [] ;

        for( var i = 0; i < this.splits.length; i++ ) {
                splitsCopy.push( this.splits[i].copy() ) ;
        }

        return splitsCopy ;
};

Bar.prototype.hasSelectedSplit = function() {
        for (var i = 0; i < this.splits.length; i++) {
                if (this.splits[i].isSelected) {
                        return true;
                }
        }
        return false;
};

Bar.prototype.updateColorOfSelectedSplit = function(in_color) {
                for (var i = 0; i < this.splits.length; i++) {
                if (this.splits[i].isSelected) {
                        this.splits[i].color = in_color;
                }
        }
        return false;
};

Bar.prototype.clearSplitSelection = function() {
        this.selectedSplit = null;
        for (var i = 0; i < this.splits.length; i++) {
                this.splits[i].isSelected = false;
        }
};


Bar.prototype.updateSplitSelectionFromState = function() {
        this.selectedSplit = null;
        for( var i = 0; i < this.splits.length; i++ ) {
                if( this.splits[i].isSelected ) {
                        this.selectedSplit = this.splits[i];
                }
        }
```

```
};


Bar.prototype.selectSplit = function(mouse_loc) {

        this.selectedSplit = this.splitClickedOn(mouse_loc);
};

Bar.prototype.findSplitForPoint = function(p)  {
        for( var i = this.splits.length-1; i >= 0; i-- ) {
                if( p.x > this.splits[i].x+this.x &&
                        p.x < this.splits[i].x+this.x + this.splits[i].w &&
                        p.y > this.splits[i].y+this.y &&
                        p.y < this.splits[i].y+this.y + this.splits[i].h) {

                        return(this.splits[i]);
                }
        }
        return null;
};

Bar.prototype.splitClickedOn = function(mouse_loc) {
        for( var i = this.splits.length-1; i >= 0; i-- ) {
                // Utilities.log(i);
                if( mouse_loc.x > this.x + this.splits[i].x &&
                        mouse_loc.x < this.x + this.splits[i].x + this.splits[i].w &&
                        mouse_loc.y > this.y + this.splits[i].y &&
                        mouse_loc.y < this.y + this.splits[i].y + this.splits[i].h)
                {
                        this.clearSplitSelection();
                        this.splits[i].isSelected = true ;
                        return this.splits[i] ;
                }
        }
        return null ;
};

Bar.prototype.removeASplit = function(split) {
        // If this bar has this split, remove it.
        var newsplits = [];
        for (var i = this.splits.length - 1; i >= 0; i--) {
                if (this.splits[i] !== split) {
                        newsplits.push(this.splits[i]);
                }
        }
        this.splits = newsplits;
};



/////////////////////////
Bar.prototype.splitBarAtPoint = function(split_point, vert_split) {
        var the_split = this.findSplitForPoint(split_point);

        if ((the_split !== null)) {

                // Splitting a single split
                if (!vert_split) {
                        this.addSplit(the_split.x, the_split.y, split_point.x-(this.x+the_split.x),
                        ↪  the_split.h, the_split.color);
                        this.addSplit(split_point.x-this.x, the_split.y,
                        ↪  the_split.w-(split_point.x-(this.x+the_split.x)), the_split.h,
                        ↪  the_split.color);
                } else {
                        this.addSplit(the_split.x, the_split.y, the_split.w,
                        ↪  split_point.y-(this.y+the_split.y), the_split.color);
                        this.addSplit(the_split.x, split_point.y-this.y, the_split.w,
                        ↪  the_split.h-(split_point.y-(this.y+the_split.y)),
                        ↪  the_split.color);
                }
                this.removeASplit(the_split);
        } else {
                // Adding a split to a bar with none
                if (this.splits.length == 0) {
                        // Make sure we really have no splits before doing this.
                        if (!vert_split) {
                                this.addSplit(0, 0, split_point.x-this.x, this.h, this.color);
                                this.addSplit(split_point.x-this.x, 0, this.x+this.w-split_point.x,
                                ↪  this.h, this.color);
                        } else {
                                this.addSplit(0, 0, this.w, split_point.y-this.y, this.color);
                                this.addSplit(0, split_point.y-this.y, this.w,
                                ↪  this.y+this.h-split_point.y, this.color);
                        }
                }
        }

};

Bar.prototype.initialSplits = function(num_splits, vert_direction) {
// Used for when there are no existing splits in a bar
        var split_interval = 0;
        var x = 0;
        var y = 0;
        var i = 0;

        if (vert_direction === true) {
                split_interval = this.w / num_splits;
                for ( i = 0; i < num_splits; i++) {
                        x = i*split_interval;
                        this.addSplit(x, y, split_interval, this.h, this.color);
                }
        } else {
                split_interval = this.h / num_splits;
                for ( i = 0; i < num_splits; i++) {
                        y = i*split_interval;
                        this.addSplit(x, y, this.w, split_interval, this.color);
                }
        }
};

Bar.prototype.splitSelectedSplit = function(num_splits, vert_direction) {

        this.updateSplitSelectionFromState();
        if (this.selectedSplit === null) {return;}

        var split_interval = 0;
        var x = this.selectedSplit.x;
        var y = this.selectedSplit.y;
        var i = 0;

        if (vert_direction === true) {
                split_interval = this.selectedSplit.w / num_splits;
                for ( i = 0; i < num_splits; i++) {
                        x = i*split_interval + this.selectedSplit.x;
                        this.addSplit(x, y, split_interval, this.selectedSplit.h,
                        ↪  this.selectedSplit.color);
                }
        } else {
                split_interval = this.selectedSplit.h / num_splits;
                for ( i = 0; i < num_splits; i++) {
                        y = i*split_interval +this.selectedSplit.y;
                        this.addSplit(x, y, this.selectedSplit.w, split_interval,
                        ↪  this.selectedSplit.color);
                }
        }
        var place = 0;
        while (this.splits[place] !== this.selectedSplit) {
                place++;
        }
        this.splits.splice(place, 1);
};



Bar.prototype.wholeBarSubSplit = function(a_split, vert_direction, subsplit_interval) {
        // Takes a single split and tries to subsplit it based on a whole bar fraction
        // If there is no subsplit, we just return a list containing the original split.
        // Otherwise we return the subsplits, but not the original split
        var new_subsplit_list = [];
        var i = 0;
        var split_hit = false; // Start out assuming we have not hit a split
        var lower_bound = 0; //storing the lower boundary of a new split
        var upper_bound = 0; // self explanatory
        var corrected_interval = 0; // storing the corrected width or height of a new split (in case it is cut off)

        if (vert_direction === true) {
                for (i = subsplit_interval; Math.floor(i)<= this.w; i = i + subsplit_interval) {
                        if (        ((i > a_split.x)                                         && (i < a_split.x + a_split.w)) ||
                                        ((i - subsplit_interval > a_split.x )        && (i - subsplit_interval < a_split.x + a_split.w)) ) {
                                split_hit = true;
                                lower_bound =  (a_split.x > (i - subsplit_interval)) ? a_split.x : (i - subsplit_interval);
                                upper_bound =  ((a_split.x + a_split.w) < i) ? a_split.x + a_split.w : i;
                                corrected_interval = upper_bound - lower_bound;
                                this.addSplitToList(new_subsplit_list, lower_bound, a_split.y, corrected_interval, a_split.h, a_split.color);
                        }
                }
        } else {
                for (i = subsplit_interval; Math.floor(i)<= this.h; i = i + subsplit_interval) {
                        if (        ((i > a_split.y)                                         && (i < a_split.y + a_split.h)) ||
                                        ((i - subsplit_interval > a_split.y )        && (i - subsplit_interval < a_split.y + a_split.h)) ) {
                                split_hit = true;
                                lower_bound =  (a_split.y > (i - subsplit_interval)) ? a_split.y : (i - subsplit_interval);
                                upper_bound =  ((a_split.y + a_split.h) < i) ? a_split.y + a_split.h : i;
                                corrected_interval = upper_bound - lower_bound;
                                this.addSplitToList(new_subsplit_list, a_split.x, lower_bound, a_split.w, corrected_interval, a_split.color);
                        }
                }
        }
        if (split_hit === false) {
                new_subsplit_list.push(a_split);
        }
        return new_subsplit_list;
};



Bar.prototype.wholeBarSplits = function(num_splits, vert_direction) {
        // Tries to split a whole bar, despite subsplits.
        var new_splits_list = [];
        var split_interval = 0;
        var list_passback = [];

        if (this.splits.length === 0) {
                // Doing initial splits because there are no existing splits
                this.initialSplits(num_splits, vert_direction);
        } else {
                // Doing subsequent splits because we already have some splits
                if (vert_direction === true) {
                        split_interval = this.w / num_splits;
                } else {
                        split_interval = this.h / num_splits;
                }
                // For every split
                for (var i = this.splits.length - 1; i >= 0; i--) {
                        // Attempt to subsplit it and concat the result into the new list
                        list_passback = this.wholeBarSubSplit(this.splits[i], vert_direction, split_interval);
                        new_splits_list = new_splits_list.concat(list_passback);
                }
                // When complete, use the new list to replace the old list.
                this.clearSplits();
                this.splits = new_splits_list;
        }
};

```

```


Bar.prototype.breakApart = function() {
        var newBars = [] ;
        var aBar ;
        if( this.splits.length === 0 ) {
                aBar = this.copy(false) ;
                aBar.isSelected = false ;
                newBars.push( aBar ) ;
        } else {
                for( var i = 0; i < this.splits.length; i++ ) {
                        newBars.push( Bar.create( this.x + this.splits[i].x, this.y + this.splits[i].y,
                        ↪  this.splits[i].w, this.splits[i].h, 'bar', this.splits[i].color )) ;
                }
        }
        return newBars ;
};

Bar.prototype.copy = function(with_offset) {
        var offset = 10 ;
        var b = new Bar() ;

        if (with_offset === false) {
                offset = 0;
        }
        if (fbCanvasObj.currentAction == "repeat") {
                offset=0;
        }
        b.x = this.x + offset ;
        b.y = this.y + offset ;
        b.w = this.w ;
        b.h = this.h ;
        b.size = this.size ;
        b.color = this.color ;
        b.splits = this.copySplits() ;
        b.label = this.label ;
        b.isUnitBar = false ;
        if (this.isUnitBar === true) {
                b.fraction = "" ;
        } else {
                b.fraction = this.fraction ;
        }
        b.type = this.type ;
        b.isSelected = true ;
        b.repeatUnit = this.repeatUnit;

        return b ;
};


Bar.prototype.makeCopy = function() {

        // This version of the copy routine does not set isSelected to true.
        // I a using this to make a copy that is just stored, so there is no reason for
        // the bar to think it is selected.

        var offset = 10 ;
        var b = new Bar() ;

        b.x = this.x + offset ;
        b.y = this.y + offset ;
        b.w = this.w ;
        b.h = this.h ;
        b.size = this.size ;
        b.color = this.color ;
        b.splits = this.copySplits() ;
        b.label = this.label ;
        b.isUnitBar = false ;
        b.fraction = this.fraction ;
        b.type = this.type ;
        b.isSelected = false ;

        return b ;
};

```

```
Bar.prototype.makeNewCopy = function(with_height) {

        // This version of the copy routine does not set isSelected to true.
        // I a using this to make a copy that is just stored, so there is no reason for
        // the bar to think it is selected.

        var offset = 10 ;

        var b = new Bar() ;

        b.x = this.x ;
        b.y = this.y +this.h + offset ;
        b.w = this.w * with_height;
        b.h = this.h  ;
        b.size = this.size * with_height;
        b.color = this.color ;
        b.isUnitBar = false ;
        b.type = this.type ;
        b.isSelected = false ;
        this.isSelected = false ;
        return b ;
};

Bar.prototype.repeat = function(clickLoc) {

//        alert(clickLoc.x +", "+clickLoc.y);

        govert = false;
/*        local_x = clickLoc.x - this.x;
        local_y = clickLoc.y - this.y;
        diag_slope = this.h / this.w;
        // modified by hsandir
        if (local_y > (local_x * diag_slope)) {
                govert = true;
        }*/

        if (this.repeatUnit !== null) {
                if (govert) {
                        this.repeatUnit.x -= 5;
                } else {
                        this.repeatUnit.y -= 5;
                }
                this.join(this.repeatUnit);
                if ((this.splits.length === 2) && (this.repeatUnit.splits.length === 0) &&
                ↪  Utilities.getMarkedIterateFlag()) {
                        this.splits[1].color = this.splits[0].color;
////////////////////////////
//                        this.splits[1].color = Utilities.colorLuminance(this.splits[0].color.toString(),
↪  -0.1);
                }
        } else {
                alert("Tried to Repeat when no repeatUnit was set.");
        }

};


Bar.prototype.iterate = function(iterate_num, vert) {

        offset = 3;
        i_iter = 0;

        iterate_unit = this.makeCopy();
        if (vert === true) {
                iterate_unit.y += offset;
        } else {
                iterate_unit.x += offset;
        }

        start_split_num = this.splits.length;

        for (i_iter = 1; i_iter < iterate_num; i_iter++) {
                this.join(iterate_unit);
        }

        if((start_split_num === 0) && (this.splits.length >0) && Utilities.getMarkedIterateFlag()) {
                //this.splits[0].color = Utilities.colorLuminance(this.splits[0].color.toString(), -0.1);
        }
};

Bar.prototype.join = function(b) {
        var gap = Bar.distanceBetween(this,b);
        gap.x = Math.abs(gap.x);
        gap.y = Math.abs(gap.y);
        var b1, b2 ;
        var originalBar = this.copy(true) ;
        var joinDimension ='' ;

        // TODO: add check for matching dimensions

        var vertmatch = this.h == b.h;
        var horizmatch = this.w == b.w;

        if (!vertmatch && !horizmatch) {
                alert("To Join, bars must have a matching dimension in height or width.");
                return(false);
        }


//          this.x = Math.min(this.x, b.x) ;
//          this.y = Math.min(this.y, b.y) ;

        if (vertmatch && horizmatch) { // since both match, determine join dimension
//                alert("Both match!");

                if( Math.abs(gap.x) < Math.abs(gap.y) ) {
                        this.h = this.h + b.h ;
                        joinDimension = 'w' ;
                } else {
                        this.w = this.w + b.w ;
                        joinDimension = 'h' ;
                }
        } else { // just one matched
                if (vertmatch) {
//                        alert("Only h matched!");
                        this.w = this.w + b.w ;
                        joinDimension = 'h';
                } else {
//                        alert("Only w matched!");
                        this.h = this.h + b.h ;
                        joinDimension = 'w';
                }
        }

        this.size = this.w * this.h ;

        var i = 0;

        this.clearSplits();

        // handling will be different for vertical/horizontal joins
        if( joinDimension == 'w' ) {
//                alert("Joining along width");
                if( originalBar.y < b.y ) {
                        b1 = originalBar ;
                        b2 = b ;
                } else {
                        b1 = b ;
                        b2 = originalBar ;
                }

                this.x = b1.x;
                this.y = b1.y;

                if (b1.splits.length === 0) {
                        this.addSplit(0, 0, b1.w, b1.h, b1.color) ;
                }
                if (b2.splits.length === 0) {
                        this.addSplit(0, b1.h, b2.w, b2.h, b2.color ) ;
                }

                if( b1.splits.length > 0 ) {
                        for(i = 0; i < b1.splits.length; i++ ) {
                                this.addSplit(b1.splits[i].x, b1.splits[i].y, b1.splits[i].w,
                                ↪  b1.splits[i].h, b1.splits[i].color ) ;
                        }
                }
                if( b2.splits.length > 0 ) {
                        for(i = 0; i < b2.splits.length; i++ ) {
                                this.addSplit(b2.splits[i].x, b2.splits[i].y + b1.h, b2.splits[i].w,
                                ↪  b2.splits[i].h, b2.splits[i].color ) ;
                        }
                }


        } else {
//              alert("Joining along height");
                if( originalBar.x < b.x ) {
                        b1 = originalBar ;
                        b2 = b ;
                } else {
                        b1 = b ;
                        b2 = originalBar ;
                }

                this.x = b1.x;
                this.y = b1.y;

                if (b1.splits.length === 0) {
                        this.addSplit(0, 0, b1.w, b1.h, b1.color) ;
                }
//              this.addSplit(0, b1.h, originalBar.w, originalBar.h, b2.c) ;
//              this.addSplit(0, b1.h, b2.w, b2.h, b2.c) ;
                if (b2.splits.length === 0) {
                        this.addSplit(b1.w, 0, b2.w, b2.h, b2.color) ;
                }

                if( b1.splits.length > 0 ) {
                        for(i = 0; i < b1.splits.length; i++ ) {
                                this.addSplit(b1.splits[i].x, b1.splits[i].y, b1.splits[i].w,
                                ↪  b1.splits[i].h, b1.splits[i].color ) ;
                        }
                }
                if( b2.splits.length > 0 ) {
                        for(i = 0; i < b2.splits.length; i++ ) {
                                this.addSplit(b2.splits[i].x + b1.w, b2.splits[i].y, b2.splits[i].w,
                                ↪  b2.splits[i].h, b2.splits[i].color ) ;
                        }
                }

        }

        // this.purgeOverlappingSplits() ;


        this.clearMeasurement() ;

        return(true);
};

Bar.prototype.nearestEdge = function(p) {
        // return a string indicating which edge is the closest one to the given point (p)
        var closestEdge = 'bottom' ;
        var dl = p.x - this.x ;
        var dr = this.w - dl ;
        var dt = p.y - this.y ;
        var db = this.h - dt ;

        if (dl <= dr && dl <= dt && dl <= db ) {
                closestEdge = "left" ;
        } else if ( dr <= dl && dr <= dt && dr <= db ) {
                closestEdge = "right" ;
        } else if ( dt <= dl && dt <= dr && dt <= db ) {
                closestEdge = "top" ;
        }
        return closestEdge ;

};

Bar.prototype.toggleSelection = function() {};

Bar.prototype.setRepeatUnit = function() {
        this.repeatUnit = this.makeCopy(true);
        this.repeatUnit.unPastel();
};

Bar.prototype.unPastel = function() {

}

// static methods

Bar.create = function(x, y, w, h, type, color) {
        var b = new Bar() ;
        b.x = x ;
        b.y = y ;
        b.w = w ;
        b.h = h ;
        b.size = w * h ;
        b.color = color ;
        b.type = type ;
        return b ;
};

Bar.createFromMouse = function(p1, p2, type, color) {
        var w = Math.abs(p2.x - p1.x) ;
        var h = Math.abs(p2.y - p1.y) ;
        var p = Point.min( p1, p2 ) ;
        var b = Bar.create(p.x, p.y, w, h, type, color) ;
        return b ;
};


Bar.createFromSplit = function(s, inx, iny) {
        var b = Bar.create(inx+s.x+10, iny+s.y+10, s.w, s.h, this.type, s.color) ;
        return b ;
};

Bar.distanceBetween = function(b1, b2) {
        // Returns the distance vertically and horizontally between the centers
        // of two bars.
        // Given as separate dimensions in a Point object, think of the return value as the amount needed
        // to translate b1 so that the center would be precisely over b2.
        var p = new Point() ;
//        var totalDistance = Math.max(b1.x + b1.w, b2.x + b2.w) - Math.min(b1.x, b2.x) ;
//        p.x = totalDistance - b1.w - b2.w ;
//        totalDistance = Math.max(b1.y + b1.h, b2.y + b2.h) - Math.min(b1.y, b2.y) ;
//        p.y = totalDistance - b1.h - b2.h ;
        p.x = b2.x - b1.x;
        p.y = b2.y - b1.y;
        return p ;
};


Bar.copyFromJSON = function(JSON_Bar) {
        var b = new Bar() ;


        b.x = JSON_Bar.x ;
        b.y = JSON_Bar.y ;
        b.w = JSON_Bar.w ;
        b.h = JSON_Bar.h ;
        b.size = JSON_Bar.size ;
        b.color = JSON_Bar.color ;
        b.makeSplitsFromJSON(JSON_Bar.splits) ;
        b.label = JSON_Bar.label ;
        b.isUnitBar = JSON_Bar.isUnitBar ;
        b.fraction = JSON_Bar.fraction ;
        b.type = JSON_Bar.type ;
        b.isSelected = false ;

        return b ;
};

Bar.prototype.makeSplitsFromJSON = function(JSON_splits) {

        this.clearSplits();
        for (var i = 0; i < JSON_splits.length; i++) {
                this.addSplit(JSON_splits[i].x,JSON_splits[i].y,JSON_splits[i].w,JSON_splits[i].h,JSON_sp
                ↪  lits[i].color);
        }
};


```

2.2.2 Mat.js

```
// Copyright University of Massachusetts Dartmouth 2013
//
// Designed and built by James P. Burke and Jason Orrill
// Modified and developed by Hakan Sandir
//
// This Javascript version of Fraction Bars is based on
// the Transparent Media desktop version of Fraction Bars,
// which in turn was based on the original TIMA Bars software
// by John Olive and Leslie Steffe.
// We thank them for allowing us to update that product.



function Mat() {
        //TODO: convert x, y to a Point?
        this.x = null ;
        this.y = null ;
        this.w = null ;
        this.h = null ;
        this.size = null ;
        this.color = null ;
//        this.splits = [] ;
//        this.label = '' ;
//        this.fraction = '' ;
        this.type = null ;
        this.isSelected = false ;
}



Mat.prototype.copy = function(with_offset) {
        var offset = 10 ;
        var b = new Bar() ;

        if (with_offset === false) {
                offset = 0;
        }

        b.x = this.x + offset ;
        b.y = this.y + offset ;
        b.w = this.w ;
        b.h = this.h ;
        b.size = this.size ;
        b.color = this.color ;
//        b.splits = this.copySplits() ;
//        b.label = this.label ;
//        b.isUnitBar = false ;
//        b.fraction = this.fraction ;
        b.type = this.type ;
        b.isSelected = true ;

        return b ;
};


```

```
Mat.prototype.nearestEdge = function(p) {
        // return a string indicating which edge is the closest one to the given point (p)
        var closestEdge = 'bottom' ;
        var dl = p.x - this.x ;
        var dr = this.w - dl ;
        var dt = p.y - this.y ;
        var db = this.h - dt ;

        if (dl <= dr && dl <= dt && dl <= db ) {
                closestEdge = "left" ;
        } else if ( dr <= dl && dr <= dt && dr <= db ) {
                closestEdge = "right" ;
        } else if ( dt <= dl && dt <= dr && dt <= db ) {
                closestEdge = "top" ;
        }
        return closestEdge ;
};

Mat.prototype.toggleSelection = function() {};

// static methods

Mat.create = function(x, y, w, h, type, color) {
        var b = new Mat() ;
        b.x = x ;
        b.y = y ;
        b.w = w ;
        b.h = h ;
        b.size = w * h ;
        b.color = color ;
        b.type = type ;
        return b ;
};

Mat.createFromMouse = function(p1, p2, type, color) {
        var w = Math.abs(p2.x - p1.x) ;
        var h = Math.abs(p2.y - p1.y) ;
        var p = Point.min( p1, p2 ) ;
        var b = Mat.create(p.x, p.y, w, h, type, color) ;
        return b ;
};

Mat.distanceBetween = function(b1, b2) {
        var p = new Point() ;
        var totalDistance = Math.max(b1.x + b1.w, b2.x + b2.w) - Math.min(b1.x, b2.x) ;
        p.x = totalDistance - b1.w - b2.w ;
        totalDistance = Math.max(b1.y + b1.h, b2.y + b2.h) - Math.min(b1.y, b2.y) ;
        p.y = totalDistance - b1.h - b2.h ;
        return p ;
};

Mat.copyFromJSON = function(JSON_Mat) {
        var b = new Mat() ;


        b.x = JSON_Mat.x ;
        b.y = JSON_Mat.y ;
        b.w = JSON_Mat.w ;
        b.h = JSON_Mat.h ;
        b.size = JSON_Mat.size ;
        b.color = JSON_Mat.color ;

        b.type = JSON_Mat.type ;
        b.isSelected = false ;

        return b ;
};
```

2.2.3 Point.js

```
// Copyright University of Massachusetts Dartmouth 2013
//
// Designed and built by James P. Burke and Jason Orrill
```

```javascript
// Modified and developed by Hakan Sandir
//
// This Javascript version of Fraction Bars is based on
// the Transparent Media desktop version of Fraction Bars,
// which in turn was based on the original TIMA Bars software
// by John Olive and Leslie Steffe.
// We thank them for allowing us to update that product.



function Point() {
        this.x = null ;
        this.y = null ;
}

Point.prototype.equals = function(p) {
        var _output = false ;
        if( p ) {
                _output = (p.x == this.x && p.y == this.y) ;
        }
        return _output ;
};

Point.prototype.isOnLine = function(line) {
        var onLine ;
        if (line.x1 == line.x2) {
                isOnLine = (this.x == line.x1) && (this.y >= Math.min(line.y1, line.y2)) && (this.y <=
                ↪  Math.max(line.y1, line.y2)) ;
        } else {
                isOnLine = (this.y == line.y1) && (this.x >= Math.min(line.x1, line.x2)) && (this.x <=
                ↪  Math.max(line.x1, line.x2)) ;
        }
        return isOnLine ;
};

// static methods

Point.createFromMouseEvent = function(e, elem) {
        var p = new Point();
        // Support both mouse and normalized touch events
        var x = (typeof e.clientX !== 'undefined') ? e.clientX : (e.touches && e.touches[0] ?
        ↪  e.touches[0].clientX : 0);
        var y = (typeof e.clientY !== 'undefined') ? e.clientY : (e.touches && e.touches[0] ?
        ↪  e.touches[0].clientY : 0);
        p.x = Math.round((x - elem.position().left) + window.pageXOffset);
        p.y = Math.round((y - elem.position().top) + window.pageYOffset);
        return p;
}

Point.subtract = function(p1, p2) {
        var p = new Point() ;
        p.x = p1.x - p2.x ;
        p.y = p1.y - p2.y ;
        return p ;
}

Point.add = function(p1, p2) {
        var p = new Point() ;
        p.x = p1.x + p2.x ;
        p.y = p1.y + p2.y ;
        return p ;

}

Point.multiply = function(p1, p2) {
        var p = new Point() ;
        p.x = p1.x * p2.x ;
        p.y = p1.y * p2.y ;
        return p ;
}

Point.min = function( p1, p2 ) {
        var p = new Point() ;
        p.x = Math.min(p1.x, p2.x) ;
        p.y = Math.min(p1.y, p2.y) ;
        return p ;
}
```

2.2.4 Line.js

```
// Copyright University of Massachusetts Dartmouth 2013
//
// Designed and built by James P. Burke and Jason Orrill
// Modified and developed by Hakan Sandir
//
// This Javascript version of Fraction Bars is based on
// the Transparent Media desktop version of Fraction Bars,
// which in turn was based on the original TIMA Bars software
// by John Olive and Leslie Steffe.
// We thank them for allowing us to update that product.


function Line(x1, y1, x2, y2) {
        this.x1 = x1 ;
        this.y1 = y1 ;
        this.x2 = x2 ;
        this.y2 = y2 ;
}

Line.prototype.equals = function(line) {
        var _output ;
        if( line ) {
                _output = (this.x1 == line.x1 && this.y1 == line.y1 && this.x2 == line.x2 && this.y2 ==
                ↪  line.y2) ;
        }
        return _output ;
}
```

2.2.5 Split.js

```
// Copyright University of Massachusetts Dartmouth 2013
//
// Designed and built by James P. Burke and Jason Orrill
// Modified and developed by Hakan Sandir
//
// This Javascript version of Fraction Bars is based on
// the Transparent Media desktop version of Fraction Bars,
// which in turn was based on the original TIMA Bars software
// by John Olive and Leslie Steffe.
// We thank them for allowing us to update that product.


function Split(x, y, w, h, c) {
        this.x = x ;
        this.y = y ;
        this.w = w ;
        this.h = h ;
        this.color = c ;
        this.isSelected = false;
}

Split.prototype.equals = function(s) {
        var _output = false ;
        if( s ) {
                _output = (s.x == this.x && s.y == this.y && s.w == this.w && s.h == this.h) ;
        }
        return _output ;
};

Split.prototype.copy = function() {
        newsplit = new Split(this.x, this.y, this.w, this.h, this.color);
        return newsplit;
};
```

2.2.6 Blob.js

```javascript
/* Blob.js
 * A Blob implementation.
 * 2013-06-20
 *
 * By Eli Grey, http://eligrey.com
 * By Devin Samarin, https://github.com/eboyjr
 * License: X11/MIT
 *   See LICENSE.md
 */

/*global self, unescape */
/*jslint bitwise: true, regexp: true, confusion: true, es5: true, vars: true, white: true,
  plusplus: true */

/*! @source http://purl.eligrey.com/github/Blob.js/blob/master/Blob.js */

if (typeof Blob !== "function" || typeof URL === "undefined")
if (typeof Blob === "function" && typeof webkitURL !== "undefined") self.URL = webkitURL;
else var Blob = (function (view) {
        "use strict";

        var BlobBuilder = view.BlobBuilder || view.WebKitBlobBuilder || view.MozBlobBuilder || view.MSBlobBuilder || (function(view) {
                var
                          get_class = function(object) {
                                return Object.prototype.toString.call(object).match(/^\[object\s(.*)\]$/)[1];
                        }
                        , FakeBlobBuilder = function BlobBuilder() {
                                this.data = [];
                        }
                        , FakeBlob = function Blob(data, type, encoding) {
                                this.data = data;
                                this.size = data.length;
                                this.type = type;
                                this.encoding = encoding;
                        }
                        , FBB_proto = FakeBlobBuilder.prototype
                        , FB_proto = FakeBlob.prototype
                        , FileReaderSync = view.FileReaderSync
                        , FileException = function(type) {
                                this.code = this[this.name = type];
                        }
                        , file_ex_codes = (
                                  "NOT_FOUND_ERR SECURITY_ERR ABORT_ERR NOT_READABLE_ERR ENCODING_ERR "
                                + "NO_MODIFICATION_ALLOWED_ERR INVALID_STATE_ERR SYNTAX_ERR"
                        ).split(" ")
                        , file_ex_code = file_ex_codes.length
                        , real_URL = view.URL || view.webkitURL || view
                        , real_create_object_URL = real_URL.createObjectURL
                        , real_revoke_object_URL = real_URL.revokeObjectURL
                        , URL = real_URL
                        , btoa = view.btoa
                        , atob = view.atob

                        , ArrayBuffer = view.ArrayBuffer
                        , Uint8Array = view.Uint8Array
                ;
                FakeBlob.fake = FB_proto.fake = true;
                while (file_ex_code--) {
                        FileException.prototype[file_ex_codes[file_ex_code]] = file_ex_code + 1;
                }
                if (!real_URL.createObjectURL) {
                        URL = view.URL = {};
                }
                URL.createObjectURL = function(blob) {
                        var
                                  type = blob.type
                                , data_URI_header
                        ;
                        if (type === null) {
                                type = "application/octet-stream";
```

```
				}
				if (blob instanceof FakeBlob) {
					data_URI_header = "data:" + type;
					if (blob.encoding === "base64") {
						return data_URI_header + ";base64," + blob.data;
					} else if (blob.encoding === "URI") {
						return data_URI_header + "," + decodeURIComponent(blob.data);
					} if (btoa) {
						return data_URI_header + ";base64," + btoa(blob.data);
					} else {
						return data_URI_header + "," + encodeURIComponent(blob.data);
					}
				} else if (real_create_object_URL) {
					return real_create_object_URL.call(real_URL, blob);
				}
			};
			URL.revokeObjectURL = function(object_URL) {
				if (object_URL.substring(0, 5) !== "data:" && real_revoke_object_URL) {
					real_revoke_object_URL.call(real_URL, object_URL);
				}
			};
			FBB_proto.append = function(data/*, endings*/) {
				var bb = this.data;
				// decode data to a binary string
				if (Uint8Array && (data instanceof ArrayBuffer || data instanceof Uint8Array)) {
					var
						  str = ""
						, buf = new Uint8Array(data)
						, i = 0
						, buf_len = buf.length
					;
					for (; i < buf_len; i++) {
						str += String.fromCharCode(buf[i]);
					}
					bb.push(str);
				} else if (get_class(data) === "Blob" || get_class(data) === "File") {
					if (FileReaderSync) {
						var fr = new FileReaderSync;
						bb.push(fr.readAsBinaryString(data));
					} else {
						// async FileReader won't work as BlobBuilder is sync
						throw new FileException("NOT_READABLE_ERR");
					}
				} else if (data instanceof FakeBlob) {
					if (data.encoding === "base64" && atob) {
						bb.push(atob(data.data));
					} else if (data.encoding === "URI") {
						bb.push(decodeURIComponent(data.data));
					} else if (data.encoding === "raw") {
						bb.push(data.data);
					}
				} else {
					if (typeof data !== "string") {
						data += ""; // convert unsupported types to strings
					}
					// decode UTF-16 to binary string
					bb.push(unescape(encodeURIComponent(data)));
				}
			};
			FBB_proto.getBlob = function(type) {
				if (!arguments.length) {
					type = null;
				}
				return new FakeBlob(this.data.join(""), type, "raw");
			};
			FBB_proto.toString = function() {
				return "[object BlobBuilder]";
			};
			FB_proto.slice = function(start, end, type) {
				var args = arguments.length;
				if (args < 3) {
					type = null;
				}
				return new FakeBlob(
					  this.data.slice(start, args > 1 ? end : this.data.length)
```

```
                                , type
                                , this.encoding
                        );
                };
                FB_proto.toString = function() {
                        return "[object Blob]";
                };
                return FakeBlobBuilder;
        }(view));

        return function Blob(blobParts, options) {
                var type = options ? (options.type || "") : "";
                var builder = new BlobBuilder();
                if (blobParts) {
                        for (var i = 0, len = blobParts.length; i < len; i++) {
                                builder.append(blobParts[i]);
                        }
                }
                return builder.getBlob(type);
        };
}(self));
```

2.3 UI Components

2.3.1 CanvasState.js

```
// A CanvasState object is used to contain a copy of the hierarchy of all the contents of the
// FractionBars Canvas. Essentially, it is kind of an epty husk of a canvas with just the bars,
// mats, and minimum required noformation for performing an "undo" or a "redo"

// Copyright University of Massachusetts Dartmouth 2013
//
// Designed and built by James P. Burke and Jason Orrill
// Modified and developed by Hakan Sandir
//
// This Javascript version of Fraction Bars is based on
// the Transparent Media desktop version of Fraction Bars,
// which in turn was based on the original TIMA Bars software
// by John Olive and Leslie Steffe.
// We thank them for allowing us to update that product.



function CanvasState(FBCanvas) {
        this.mFBCanvas = FBCanvas ;
        this.canvasState = null ;

        this.mBars = [] ;
        this.mMats = [] ;
        this.mUnitBar = null ;
        this.mHidden= [] ;
}



// Also copy mats
CanvasState.prototype.grabBarsAndMats = function() {

        var mBars = [];
        var mMats = [];
        var aBar = null;
        var mHidden=[];

        for( var i = 0; i < this.mFBCanvas.bars.length; i++ ) {
                aBar = this.mFBCanvas.bars[i].copy(false);
                this.mBars.push( aBar ) ;
                if (this.mFBCanvas.bars[i] === this.mFBCanvas.unitBar) {
                        this.mUnitBar = aBar; // Remember which copy is a copy of the unit bar, if any.
                }
                if (this.mFBCanvas.bars[i].isSelected) {
                        aBar.isSelected = true;
                } else {
                        aBar.isSelected = false;
                }
                if (this.mFBCanvas.bars[i].isUnitBar) {
                        aBar.isUnitBar = true;
                }
//                aBar.clearSplitSelection();
        }

        for(var j = 0; j <this.mFBCanvas.mats.length; j++ ) {
                this.mMats.push( this.mFBCanvas.mats[j].copy(false) ) ;
        }
        this.mHidden=hiddenButtonsName.slice(0);
        if (hiddenButtonsName.indexOf("tool_hide")<0) {
        this.mHidden.push('tool_hide') ;
        }
        if (hiddenButtonsName.indexOf("action_show")<0) {
        this.mHidden.push('action_show') ;
        }

};


```

2.3.2 SplitsWidget.js

```
// Copyright University of Massachusetts Dartmouth 2013
//
// Designed and built by James P. Burke and Jason Orrill
// Modified and developed by Hakan Sandir
//
// This Javascript version of Fraction Bars is based on
// the Transparent Media desktop version of Fraction Bars,
// which in turn was based on the original TIMA Bars software
// by John Olive and Leslie Steffe.
// We thank them for allowing us to update that product.


function SplitsWidget(canvasContext) {
        this.context = canvasContext ;
        this.images = [];
        this.vertical = true;
        this.num_splits = 2;
        this.color = "yellow";

}



SplitsWidget.prototype.handleSliderChange = function(event, ui) {
        // var aslider = event.target;
        // this.num_splits = aslider.slider( "value" );
        // alert(this.num_splits);


//        this.num_splits = $("#split-slider").slider("value");
//        this.num_splits = $("#split-slider-field").val();

        this.num_splits = ui.value;
        this.refreshCanvas();

};


SplitsWidget.prototype.handleVertHorizChange = function(event) {


        var the_checked = $("input:checked").val();
        if (the_checked == "Vertical") {
                this.vertical = true;
        } else {
                if(Utilities.flag[1]){

                        this.vertical = false;}
```

```
		else {
			this.vertical = true;}
	}
	this.refreshCanvas();
};

SplitsWidget.prototype.refreshCanvas = function() {

	this.context.strokeStyle = "#FF3333";
	this.context.fillStyle = this.color;

//	  this.num_splits = $("#split-slider").slider("value");
//	  this.num_splits = $("#split-slider-field").val();
//	  this.num_splits = document.getElementById("split-slider").slider();

	var width = $("#split-display").attr("width");
	var height =  $("#split-display").attr("height");

	this.context.fillRect(0,0,width,height);

//	  this.context.strokeText(this.num_splits,10,10);

	if (this.vertical) {
		width = width / this.num_splits;
		for (var i = 0; i < this.num_splits; i++) {
			this.context.strokeRect(i*width,0,width,height);
		}
	} else {
		height = height / this.num_splits;
		for (var j = 0; j < this.num_splits; j++) {
			this.context.strokeRect(0,j*height,width,height);
		}
	}

	this.refreshed = true;
};
```

2.4 Utility Functions

2.4.1 utilities.js

```
// Copyright University of Massachusetts Dartmouth 2013
//
// Designed and built by James P. Burke and Jason Orrill
// Modified and developed by Hakan Sandir
//
// This Javascript version of Fraction Bars is based on
// the Transparent Media desktop version of Fraction Bars,
// which in turn was based on the original TIMA Bars software
// by John Olive and Leslie Steffe.
// We thank them for allowing us to update that product.



function Utilities() {
	this.shiftKeyDown = false ;
	this.ctrlKeyDown = false ;
}

//First attempt
Utilities.file_list="";
Utilities.file_index=0;
//

Utilities.flag=['it,sp,rpt,lng'];
Utilities.flag[0]=false;
Utilities.flag[1]=false;
Utilities.flag[2]=false;
Utilities.flag[3]=false;
Utilities.USE_CURRENT_SELECTION = 'useCurrent' ;
Utilities.USE_LAST_SELECTION = 'useLast' ;


Utilities.include_js = function (file,path) {
        if (typeof(path) !== 'undefined' && path !== null) {
                file = path + file ;
        }
        var include_file = document.createElement('script');
        include_file.type = 'text/javascript';
        include_file.src = file;
        document.getElementsByTagName('head')[0].appendChild(include_file);
};

Utilities.createFraction = function(numerator, denominator) {
  // Calculate the (approximate) fraction for this measurement.
  // Basic algorigm taken from Dr. Math at the Math Forum...
  // http://mathforum.org/library/drmath/view/51910.html

  var max_terms = 30 ;
  var min_divisor = 0.000001 ;
  var max_error = 0.00001 ;

  var v = numerator / denominator ;
  var f = v ;

  var n1 = 1 ;
  var d1 = 0 ;
  var n2 = 0 ;
  var d2 = 1 ;

  var a ;

  for (i = 0; i < max_terms; i++) {
    a = Math.round(f) ;
    f = f - a ;
    n = n1 * a + n2 ;
    d = d1 * a + d2 ;

    n2 = n1 ;
    d2 = d1 ;

    n1 = n ;
    d1 = d ;

    if (f < min_divisor && Math.abs(v-n/d) < max_error) {
      break ;
    }

    f = 1/f ;
  }

  if (Math.floor(v) == v) {
          return v ;
  }
  else{
          return Math.abs(n) + "/" + Math.abs(d) ;
  }
};

Utilities.log = function(msg) {
        if( window.console ) {
                console.log( msg ) ;
        }
};

Utilities.colorLuminance = function(hex, lum) {

  // validate hex string
  hex = String(hex).replace(/[^0-9a-f]/gi, '');
  if (hex.length < 6) {
    hex = hex[0]+hex[0]+hex[1]+hex[1]+hex[2]+hex[2];
  }
  lum = lum || 0;

  // convert to decimal and change luminosity
  var rgb = "#", c, i;
  for (i = 0; i < 3; i++) {
    c = parseInt(hex.substr(i*2,2), 16);
    c = Math.round(Math.min(Math.max(0, c + (c * lum)), 255)).toString(16);
    rgb += ("00"+c).substr(c.length);
  }

  return rgb;
};

Utilities.getMarkedIterateFlag = function() {
  // Returns false by default
  state = ($('#marked-iterate').attr('data-flag') === "true");
  return state;
};
```

2.4.2 cycle.js

```
/*
    cycle.js
    2013-02-19

    Public Domain.

    NO WARRANTY EXPRESSED OR IMPLIED. USE AT YOUR OWN RISK.

    This code should be minified before deployment.
    See http://javascript.crockford.com/jsmin.html

    USE YOUR OWN COPY. IT IS EXTREMELY UNWISE TO LOAD CODE FROM SERVERS YOU DO
    NOT CONTROL.
*/

/*jslint evil: true, regexp: true */

/*members $ref, apply, call, decycle, hasOwnProperty, length, prototype, push,
    retrocycle, stringify, test, toString
*/

if (typeof JSON.decycle !== 'function') {
    JSON.decycle = function decycle(object) {
        'use strict';

// Make a deep copy of an object or array, assuring that there is at most
// one instance of each object or array in the resulting structure. The
// duplicate references (which might be forming cycles) are replaced with
// an object of the form
//      {$ref: PATH}
// where the PATH is a JSONPath string that locates the first occurance.
// So,
//      var a = [];
//      a[0] = a;
//      return JSON.stringify(JSON.decycle(a));
// produces the string '[{"$ref":"$"}]'.

// JSONPath is used to locate the unique object. $ indicates the top level of
// the object or array. [NUMBER] or [STRING] indicates a child member or
// property.

        var objects = [],   // Keep a reference to each unique object or array
            paths = [];     // Keep the path to each unique object or array

        return (function derez(value, path) {

// The derez recurses through the object, producing the deep copy.

            var i,          // The loop counter
                name,       // Property name
                nu;         // The new object or array

// typeof null === 'object', so go on if this value is really an object but not
// one of the weird builtin objects.

            if (typeof value === 'object' && value !== null &&
                    !(value instanceof Boolean) &&
                    !(value instanceof Date)    &&
                    !(value instanceof Number)  &&
                    !(value instanceof RegExp)  &&
                    !(value instanceof String)) {

// If the value is an object or array, look to see if we have already
// encountered it. If so, return a $ref/path object. This is a hard way,
// linear search that will get slower as the number of unique objects grows.

                for (i = 0; i < objects.length; i += 1) {
                    if (objects[i] === value) {
                        return {$ref: paths[i]};
                    }
                }

// Otherwise, accumulate the unique value and its path.

                objects.push(value);
                paths.push(path);

// If it is an array, replicate the array.

                if (Object.prototype.toString.apply(value) === '[object Array]') {
                    nu = [];
                    for (i = 0; i < value.length; i += 1) {
                        nu[i] = derez(value[i], path + '[' + i + ']');
                    }
                } else {

// If it is an object, replicate the object.

                    nu = {};
                    for (name in value) {
                        if (Object.prototype.hasOwnProperty.call(value, name)) {
                            nu[name] = derez(value[name],
                                path + '[' + JSON.stringify(name) + ']');
                        }
                    }
                }
                return nu;
            }
            return value;
        }(object, '$'));
    };
}


if (typeof JSON.retrocycle !== 'function') {
    JSON.retrocycle = function retrocycle($) {
        'use strict';

// Restore an object that was reduced by decycle. Members whose values are
// objects of the form
//      {$ref: PATH}
// are replaced with references to the value found by the PATH. This will
// restore cycles. The object will be mutated.

// The eval function is used to locate the values described by a PATH. The
// root object is kept in a $ variable. A regular expression is used to
// assure that the PATH is extremely well formed. The regexp contains nested
// * quantifiers. That has been known to have extremely bad performance
// problems on some browsers for very long strings. A PATH is expected to be
// reasonably short. A PATH is allowed to belong to a very restricted subset of
// Goessner's JSONPath.

// So,
//      var s = '[{"$ref":"$"}]';
//      return JSON.retrocycle(JSON.parse(s));
// produces an array containing a single element which is the array itself.

        var px =
            /^\$(?:\[(?:\d+|\"(?:[^\\\"\u0000-\u001f]|\\([\\\"\/bfnrt]|u[0-9a-zA-Z]{4}))*\")\])*$/;

        (function rez(value) {

```

```
// The rez function walks recursively through the object looking for $ref
// properties. When it finds one that has a value that is a path, then it
// replaces the $ref object with a reference to the value that is found by
// the path.

            var i, item, name, path;

            if (value && typeof value === 'object') {
                if (Object.prototype.toString.apply(value) === '[object Array]') {
                    for (i = 0; i < value.length; i += 1) {
                        item = value[i];
                        if (item && typeof item === 'object') {
                            path = item.$ref;
                            if (typeof path === 'string' && px.test(path)) {
                                value[i] = eval(path);
                            } else {
                                rez(item);
                            }
                        }
                    }
                } else {
                    for (name in value) {
                        if (typeof value[name] === 'object') {
                            item = value[name];
                            if (item) {
                                path = item.$ref;
                                if (typeof path === 'string' && px.test(path)) {
                                    value[name] = eval(path);
                                } else {
                                    rez(item);
                                }
                            }
                        }
                    }
                }
            }
        }($));
        return $;
    };
}
```

3 CSS Files

3.1 Main Styles

3.1.1 fractionBars.css

```

/*
// Copyright University of Massachusetts Dartmouth 2013
//
// Designed and built by James P. Burke and Jason Orrill
//
// This Javascript version of Fraction Bars is based on
// the Transparent Media desktop version of Fraction Bars,
// which in turn was based on the original TIMA Bars software
// by John Olive and Leslie Steffe.
// We thank them for allowing us to update that product.
*/


body, p, td {
        font-family: Helvetica, Arial, sans-serif ;
        font-size: 12px ;
}
a {
        cursor: pointer;
}

.skip-link {display:none;}

```

```css
#labelInput {
        display: none ;
}

#fbCanvas {
        background-color: #EFEFEF;
        border: 1px gray groove;
}
#tools {
        float:left ;
        width: 100px ;
}
#tools .toolGroup {
        margin-bottom: 12px ;
}
#tools .toolGroup a {
        border: 1px solid #999999 ;
        display: block ;
        width: 80px ;
        text-align: center ;
        margin: 4px ;
        padding: 2px ;
}

#tools .toolGroup a.colorBlock {
        float: left;
        border: 1px solid gray;
        width: 10px;
        height: 10px;
        display: inline;
        margin: 3px 3px 4px 4px;
}

#tools .toolGroup a.colorBlock1 {
        float: left;
        border: 1px solid gray;
        width: 10px;
        height: 10px;
        display: inline;
        margin: 3px 3px 4px 4px;
}

#tools .toolGroup a.hideShow {
        float: left;
        border: 1px solid gray;
        width: 35px;
        display: inline;
        margin: 3px 3px 4px 4px;
}

#tools .toolGroup a.colorSelected {
        border:1px solid black ;
}

#tools .toolGroup a.toolSelected {
        border:1px solid black ;
        background-color: #EEEEEE ;
}

#split-display {
        float: right;
        background-color: #EFEFEF;
        border: 1px gray groove;
}


#split-slider {
        width: 250px;
}

.color10 {background-color: #FFFF66;}
.color3 {background-color: #ACBEFF;}
.color7 {background-color: #E6E6E6;}
.color5 {background-color: #FFFFFF;}
.color12 {background-color: #CCFF66;}
```

```
.color9 {background-color: #FFCC66;}
.color13 {background-color: #DD99FF;}
.color14 {background-color: #FF92DA;}
.color1 {background-color: #A8FFF4;}
.color6 {background-color: #EFEFEF;}
.color2 {background-color: #DDFFF0;}
.color4 {background-color: #707EFF;}
.color8 {background-color: #D6D6D6;}
.color16 {background-color: #FF8C8C;}
.color11 {background-color: #E9FF66;}
.color15 {background-color: #F56ED0;}
```

3.1.2 deneme.css

```
fieldset {
  border: 2px #aa3333 ;
  border-radius: 10px;
}

/* Editable [pseudo]select (i.e. fieldsets with [class=elist]) */

fieldset.elist {
  display: block;
  position: relative;
  vertical-align: bottom;
  overflow: visible;
    width: 200px;
  padding: 0;
  margin: 0;
  border: none;
}

fieldset.elist ul {
  position: absolute;
  width: 100%;
  max-height: 320px;
  padding: 0;
  margin: 0;
  overflow: hidden;
  background-color: #ffffff;
}

fieldset.elist:hover ul {
  background-color: #ffffff;
  border: 2px #aaaaaa solid;
  left: 2px;
  overflow: auto;
}

fieldset.elist ul > li {
  list-style-type: none;
  background-color: transparent;
}

fieldset.elist label {
  display: none;
  width: 100%;
}

fieldset.elist ul input[type="radio"] {
  display: none;

}

fieldset.elist input[type="radio"]:checked ~ label {
  display: block;
  width: 292px;
  background-color: #ffffff;

}

fieldset.elist input[type="radio"]:checked ~ label:after {
  content: " \2335";
}

fieldset.elist:hover label {
  display: block;
  height: 100%;
}


fieldset.elist:hover input[type="radio"]:checked ~ label {
  background-color: #aaaaaa;
}
```

3.2 Language and UI Styles

3.2.1 lang_eng.css

```
.bar_title:before{content: "Fraction Bar";}
.c_bar:before{content: "Bar";}
.c_mat:before{content: "Mat";}
.c_copy:before{content: "Copy";}
.c_repeat:before{content: "Repeat";}
.c_iterate:before{content: "Iterate";}
.c_join:before{content: "Join";}
.c_delete:before{content: "Delete";}
.c_parts:before{content: "Parts";}
.c_pieces:before{content: "Line";}
.c_b_apart:before{content: "Break Apart";}
.c_pullout:before{content: "Pull Out Parts";}
.c_c_parts:before{content: "Clear Parts";}
.c_set_unit:before{content: "Set Unit Bar";}
.c_measure:before{content: "Measure";}
.c_label:before{content: "Label";}
.c_undo:before{content: "Undo";}
.c_redo:before{content: "Redo";}
.c_save:before{content: "Save";}
.c_open:before{content: "Open";}
.c_new:before{content: "New";}
.c_print:before{content: "Print";}
.c_properties:before{content: "Properties";}
.c_hide:before{content: "Hide";}
.c_show:before{content: "Show";}
.c_previous:before{content: "Previous";}
.c_next:before{content: "Next";}
.c_dialog_splits title:before {content: "Partition part of bar";}
.c_vertical:before{content: "Vertical";}
.c_horizontal:before{content: "Horizontal";}
.c_number_part:before{content: "Number of parts:";}
.c_part_whole:before{content: "Partition whole bar";}
.c_part_part:before{content: "Partition part of bar";}
.c_dialog_properties title:before {content: "Partition part of bar";}
.c_iterations:before{content: "Iterations";}
.c_dont_create:before{content: "Don't Create New Bar";}
.c_create_new:before{content: "Create New Bar";}
.c_two_way:before{content: "Two way Iterate";}
.c_one_way:before{content: "Only one way Iterate";}
.c_splits:before{content: "Splits (Parts)";}
.c_vert_horiz:before{content: "Vertical or Horizontal Split";}
.c_only_vert:before{content: "Only vertical split";}
.c_lang:before{content: "Language";}
.c_lang_tur:before{content: "Turkish";}
.c_lang_eng:before{content: "English";}
.c_color:before{content: "Background Color";}
.c_dialog_iterate title:before {content: "Iterate";}
.c_number_iterations:before{content: "Number of iterations:";}
.c_c_choose_file title:before {content: "Choose a File";}
.c_c_open_file:before{content: "Use the button below to choose a FractionBars file to open.";}
.c_split_alert:before{content: "Please select a bar to partition.";}
```

4 AceofBases Program

4.1 HTML Files

4.1.1 index.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Ace of Base Arithmetic</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
            margin: 0;
            background-color: #f9f9f9;
        }

        .container {
            text-align: center;
            border: 1px solid #ccc;
            padding: 20px;
            border-radius: 10px;
            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
            background-color: white;
        }

        canvas {
            border: 1px solid #000;
        }

        .controls {
            margin-top: 20px;
        }

        button {
            margin: 5px;
        }

        .results {
            margin-top: 20px;
        }

        .popup {
            display: none;
            position: fixed;
            left: 50%;
            top: 50%;
            transform: translate(-50%, -50%);
            padding: 20px;
            background-color: white;
            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
            border-radius: 10px;
            z-index: 1000;
        }

        .popup .close {
            cursor: pointer;
            background-color: red;
            color: white;
            border: none;
            border-radius: 5px;
            padding: 5px 10px;
            margin-top: 10px;
        }
    </style>
</head>
<body>
    <div class="container">
```

```html
        <h1>Ace of Base Arithmetic</h1>
        <p>Circle the number of squares you want to be your grouping unit:</p>
        <div class="instructions">
            <p>Drag to select between 2 and 15 cubes:</p>
            <canvas id="cubeCanvas" width="600" height="400"></canvas>
        </div>
        <div class="controls">
            <p id="selectedUnits">Selected Units: 0</p>
            <button id="composeButton">Compose</button>
            <button id="decomposeButton">Decompose</button>
            <button id="addCubeButton">Add Cube</button>
            <button id="removeCubeButton">Remove Cube</button>
        </div>
        <div class="results">
            <p id="baseConversion"></p>
            <p id="baseTenCount"></p>
        </div>
    </div>
    <div id="explanationPopup" class="popup">
        <p>With bases larger than ten, we need different symbols for the numbers called ten, eleven, twelve, thirteen, fourteen, etc., in base ten. Here is how they are represented:</p>
        <ul>
            <li>What is called ten in base ten will be represented with the digit T.</li>
            <li>What is called eleven in base ten will be represented with the digit E.</li>
            <li>What is called twelve in base ten will be represented with the digit D.</li>
            <li>What is called thirteen in base ten will be represented with the digit R.</li>
            <li>What is called fourteen in base ten will be represented with the digit F.</li>
        </ul>
        <button class="close" onclick="document.getElementById('explanationPopup').style.display='none'">Close</button>
    </div>
    <div id="overflowPopup" class="popup">
        <p>Please choose a larger grouping unit.</p>
        <button class="close" onclick="document.getElementById('overflowPopup').style.display='none'">Close</button>
    </div>
    <script src="script.js"></script>
</body>
</html>
```

4.1.2 index_ace_of_bases.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Ace of Base Arithmetic</title>
    <link rel="stylesheet" href="styles_ace_of_bases.css">
</head>
<body>
    <div class="container">
        <h1>Ace of Base Arithmetic</h1>
        <p>Circle the number of squares you want to be your grouping unit:</p>
        <div class="instructions">
            <p>Drag to select between 2 and 15 cubes:</p>
            <canvas id="cubeCanvas" width="600" height="400"></canvas>
        </div>
        <div class="controls">
            <p id="selectedUnits">Selected Units: 0</p>
            <button id="composeButton">Compose</button>
            <button id="decomposeButton">Decompose</button>
            <button id="addCubeButton">Add Cube</button>
            <button id="removeCubeButton">Remove Cube</button>
        </div>
        <div class="results">
            <p id="baseConversion"></p>
            <p id="baseTenCount"></p>
        </div>
    </div>
    <div id="explanationPopup" class="popup">
        <p>With bases larger than ten, we need different symbols for the numbers called ten, eleven, twelve, thirteen, fourteen, etc., in base ten. Here is how they are represented:</p>
        <ul>
            <li>What is called ten in base ten will be represented with the digit T.</li>
            <li>What is called eleven in base ten will be represented with the digit E.</li>
            <li>What is called twelve in base ten will be represented with the digit D.</li>
            <li>What is called thirteen in base ten will be represented with the digit R.</li>
            <li>What is called fourteen in base ten will be represented with the digit F.</li>
        </ul>
        <button class="close"
        ↪  onclick="document.getElementById('explanationPopup').style.display='none'">Close</button>
    </div>
    <div id="overflowPopup" class="popup">
        <p>Please choose a larger grouping unit.</p>
        <button class="close"
        ↪  onclick="document.getElementById('overflowPopup').style.display='none'">Close</button>
    </div>
    <script src="script_ace_of_bases.js"></script>
</body>
</html>
```

4.2 JavaScript Files

4.2.1 script.js

```
document.addEventListener('DOMContentLoaded', () => {
    const canvas = document.getElementById('cubeCanvas');
    const ctx = canvas.getContext('2d');
    const composeButton = document.getElementById('composeButton');
    const decomposeButton = document.getElementById('decomposeButton');
    const addCubeButton = document.getElementById('addCubeButton');
    const removeCubeButton = document.getElementById('removeCubeButton');
    const selectedUnitsDisplay = document.getElementById('selectedUnits');
    const baseConversionDisplay = document.getElementById('baseConversion');
    const baseTenCountDisplay = document.getElementById('baseTenCount');
    const explanationPopup = document.getElementById('explanationPopup');
    const overflowPopup = document.getElementById('overflowPopup');

    let cubes = [];
    let selectedUnits = 0;
    let modulus = 0;
    let isDragging = false;
    let dragOffsetX, dragOffsetY;
    let draggedCube = null;
    let isSelecting = false;
    let selectionStartX, selectionStartY;

    const generateRandomCubes = () => {
        const count = Math.floor(Math.random() * 14) + 2;
        cubes = Array.from({ length: count }, (_, i) => ({
            x: Math.random() * (canvas.width - 20),
            y: Math.random() * (canvas.height - 20),
            size: 20,
        }));
        drawCubes();
    };

    const drawCubes = () => {
        ctx.clearRect(0, 0, canvas.width, canvas.height);
        cubes.forEach(cube => {
            ctx.fillStyle = 'blue';
            ctx.fillRect(cube.x, cube.y, cube.size, cube.size);
        });
    };

    const handleMouseDown = (e) => {
        const rect = canvas.getBoundingClientRect();
        const startX = e.clientX - rect.left;
        const startY = e.clientY - rect.top;

        draggedCube = cubes.find(cube => (
            startX >= cube.x && startX <= cube.x + cube.size &&
            startY >= cube.y && startY <= cube.y + cube.size
        ));

        if (draggedCube) {
            isDragging = true;
            dragOffsetX = startX - draggedCube.x;
            dragOffsetY = startY - draggedCube.y;
        } else {
            isSelecting = true;
            selectionStartX = startX;
            selectionStartY = startY;
            canvas.addEventListener('mousemove', handleMouseMove);
            canvas.addEventListener('mouseup', handleMouseUp);
        }
    };

    const handleMouseMove = (e) => {
        if (isSelecting) {
            const rect = canvas.getBoundingClientRect();
            const currentX = e.clientX - rect.left;
            const currentY = e.clientY - rect.top;
            ctx.clearRect(0, 0, canvas.width, canvas.height);
            drawCubes();
            ctx.strokeStyle = 'red';
            ctx.strokeRect(selectionStartX, selectionStartY, currentX - selectionStartX, currentY -
            ↪  selectionStartY);
        }
    };

    const handleMouseUp = (e) => {
        if (isSelecting) {
            const rect = canvas.getBoundingClientRect();
            const endX = e.clientX - rect.left;
            const endY = e.clientY - rect.top;
            const selected = cubes.filter(cube => (
                cube.x >= Math.min(selectionStartX, endX) && cube.x <= Math.max(selectionStartX, endX) &&
                cube.y >= Math.min(selectionStartY, endY) && cube.y <= Math.max(selectionStartY, endY)
            ));
            selectedUnits = Math.min(selected.length, 15);
            selectedUnitsDisplay.textContent = `Selected Units: ${selectedUnits}`;
            modulus = selectedUnits;

            if (selectedUnits > 10) {
                explanationPopup.style.display = 'block';
            } else {
                explanationPopup.style.display = 'none';
            }

            isSelecting = false;
            canvas.removeEventListener('mousemove', handleMouseMove);
            canvas.removeEventListener('mouseup', handleMouseUp);
        }
    };

    const handleTouchStart = (e) => {
        const touch = e.touches[0];
        const rect = canvas.getBoundingClientRect();
        const startX = touch.clientX - rect.left;
        const startY = touch.clientY - rect.top;

        draggedCube = cubes.find(cube => (
            startX >= cube.x && startX <= cube.x + cube.size &&
            startY >= cube.y && startY <= cube.y + cube.size
        ));

        if (draggedCube) {
            isDragging = true;
            dragOffsetX = startX - draggedCube.x;
            dragOffsetY = startY - draggedCube.y;
        } else {
            isSelecting = true;
            selectionStartX = startX;
            selectionStartY = startY;
            canvas.addEventListener('touchmove', handleTouchMove);
            canvas.addEventListener('touchend', handleTouchEnd);
        }
    };

    const handleTouchMove = (e) => {
        if (isSelecting) {
            const touch = e.touches[0];
            const rect = canvas.getBoundingClientRect();
            const currentX = touch.clientX - rect.left;
            const currentY = touch.clientY - rect.top;
            ctx.clearRect(0, 0, canvas.width, canvas.height);
            drawCubes();
            ctx.strokeStyle = 'red';
            ctx.strokeRect(selectionStartX, selectionStartY, currentX - selectionStartX, currentY -
            ↪  selectionStartY);
        } else if (isDragging && draggedCube) {
            const touch = e.touches[0];
            const rect = canvas.getBoundingClientRect();
            draggedCube.x = touch.clientX - rect.left - dragOffsetX;
            draggedCube.y = touch.clientY - rect.top - dragOffsetY;
            drawCubes();
        }
    };

    const handleTouchEnd = (e) => {
        if (isSelecting) {
            const rect = canvas.getBoundingClientRect();
            const touch = e.changedTouches[0];
            const endX = touch.clientX - rect.left;
            const endY = touch.clientY - rect.top;
            const selected = cubes.filter(cube => (
                cube.x >= Math.min(selectionStartX, endX) && cube.x <= Math.max(selectionStartX, endX) &&
                cube.y >= Math.min(selectionStartY, endY) && cube.y <= Math.max(selectionStartY, endY)
            ));
            selectedUnits = Math.min(selected.length, 15);
            selectedUnitsDisplay.textContent = `Selected Units: ${selectedUnits}`;
            modulus = selectedUnits;

            if (selectedUnits > 10) {
                explanationPopup.style.display = 'block';
            } else {
                explanationPopup.style.display = 'none';
            }

            isSelecting = false;
            canvas.removeEventListener('touchmove', handleTouchMove);
            canvas.removeEventListener('touchend', handleTouchEnd);
        }
    };

    const handleDrag = (e) => {
        if (isDragging && draggedCube) {
            const rect = canvas.getBoundingClientRect();
            draggedCube.x = e.clientX - rect.left - dragOffsetX;
            draggedCube.y = e.clientY - rect.top - dragOffsetY;
            drawCubes();
        }
    };

    const handleDragEnd = (e) => {
        isDragging = false;
        draggedCube = null;
    };

    const updateBaseConversion = () => {
        if (modulus > 1) {
            const base = modulus;
            const baseStr = convertToBase(cubes.length, base);

            if (baseStr.length > 4) {
                overflowPopup.style.display = 'block';
                return;
            }

            const baseComponents = { rods: 0, flats: 0, cubes3D: 0, units: 0 };
            let count = cubes.length;

            while (count > 0) {
                if (count >= base * base * base) {
                    baseComponents.cubes3D++;
                    count -= base * base * base;
                } else if (count >= base * base) {
                    baseComponents.flats++;
                    count -= base * base;
                } else if (count >= base) {
                    baseComponents.rods++;
                    count -= base;
                } else {
                    baseComponents.units++;
                    count--;
                }
            }

            baseConversionDisplay.textContent = `Base ${base}: ${baseStr}`;
            baseTenCountDisplay.textContent = `Base 10: ${cubes.length}`;
            drawBaseComponents(base, baseComponents);
        }
    };

    const convertToBase = (number, base) => {
        const digitMap = { 10: 'T', 11: 'E', 12: 'D', 13: 'R', 14: 'F' };
        let result = '';
        while (number > 0) {
            let digit = number % base;
            if (digit >= 10 && digit <= 14) {
                digit = digitMap[digit];
            }
            result = digit.toString() + result;
            number = Math.floor(number / base);
        }
        return result;
    };

    const drawBaseComponents = (base, baseComponents) => {
        ctx.clearRect(0, 0, canvas.width, canvas.height);
        let xOffset = 0;
        let yOffset = 0;

        // Draw units
        for (let i = 0; i < baseComponents.units; i++) {
            ctx.fillStyle = 'blue';
            ctx.fillRect(xOffset, yOffset, 20, 20);
            xOffset += 25;
            if (xOffset > canvas.width - 20) {
                xOffset = 0;
                yOffset += 25;
            }
        }

        // Draw rods
        for (let i = 0; i < baseComponents.rods; i++) {
            ctx.fillStyle = 'green';
            ctx.fillRect(xOffset, yOffset, 20 * base, 20);
            for (let j = 0; j < base; j++) {
                ctx.strokeStyle = 'black';
                ctx.strokeRect(xOffset + j * 20, yOffset, 20, 20);
            }
            xOffset += 20 * base + 5;
            if (xOffset > canvas.width - 20 * base) {
                xOffset = 0;
                yOffset += 25;
            }
        }

        // Draw flats
        for (let i = 0; i < baseComponents.flats; i++) {
            ctx.fillStyle = 'yellow';
            ctx.fillRect(xOffset, yOffset, 20 * base, 20 * base);
            for (let j = 0; j < base; j++) {
                for (let k = 0; k < base; k++) {
                    ctx.strokeStyle = 'black';
                    ctx.strokeRect(xOffset + j * 20, yOffset + k * 20, 20, 20);
                }
            }
            xOffset += 20 * base + 5;
            if (xOffset > canvas.width - 20 * base) {
                xOffset = 0;
                yOffset += 20 * base + 5;
            }
        }

        // Draw 3D cubes
        for (let i = 0; i < baseComponents.cubes3D; i++) {
            ctx.fillStyle = 'red';
            const cubeSize = 20 * base;
            const depth = cubeSize / 3;

            // Draw front face
            ctx.fillRect(xOffset, yOffset, cubeSize, cubeSize);
            ctx.strokeStyle = 'black';
            for (let j = 0; j < base; j++) {
                for (let k = 0; k < base; k++) {
                    ctx.strokeRect(xOffset + j * 20, yOffset + k * 20, 20, 20);
                }
            }

            // Draw top face
            ctx.beginPath();
            ctx.moveTo(xOffset, yOffset);
            ctx.lineTo(xOffset + depth, yOffset - depth);
            ctx.lineTo(xOffset + cubeSize + depth, yOffset - depth);
            ctx.lineTo(xOffset + cubeSize, yOffset);
            ctx.closePath();
            ctx.fillStyle = 'rgba(255, 0, 0, 0.8)';
            ctx.fill();
            ctx.stroke();

            // Draw right face
            ctx.beginPath();
            ctx.moveTo(xOffset + cubeSize, yOffset);
            ctx.lineTo(xOffset + cubeSize + depth, yOffset - depth);
            ctx.lineTo(xOffset + cubeSize + depth, yOffset + cubeSize - depth);
            ctx.lineTo(xOffset + cubeSize, yOffset + cubeSize);
            ctx.closePath();
            ctx.fillStyle = 'rgba(255, 0, 0, 0.6)';
            ctx.fill();
            ctx.stroke();

            xOffset += cubeSize + depth + 5;
            if (xOffset > canvas.width - cubeSize) {
                xOffset = 0;
                yOffset += cubeSize + depth + 5;
            }
        }
    };

    const decomposeCubes = () => {
        cubes.forEach(cube => {
            cube.x = Math.random() * (canvas.width - 20);
            cube.y = Math.random() * (canvas.height - 20);
        });
        drawCubes();
        selectedUnits = 0;
        selectedUnitsDisplay.textContent = `Selected Units: ${selectedUnits}`;
        baseConversionDisplay.textContent = '';
        baseTenCountDisplay.textContent = '';
    };

    const addCube = () => {
        cubes.push({
            x: Math.random() * (canvas.width - 20),
            y: Math.random() * (canvas.height - 20),
            size: 20,
        });
        drawCubes();
    };

    const removeCube = () => {
        if (cubes.length > 0) {
            cubes.pop();
            drawCubes();
        }
    };

    canvas.addEventListener('mousedown', handleMouseDown);
    canvas.addEventListener('touchstart', handleTouchStart);
    canvas.addEventListener('mousemove', handleDrag);
    canvas.addEventListener('mouseup', handleDragEnd);
    canvas.addEventListener('touchmove', handleTouchMove);
    canvas.addEventListener('touchend', handleDragEnd);

    composeButton.addEventListener('click', updateBaseConversion);
    decomposeButton.addEventListener('click', decomposeCubes);
    addCubeButton.addEventListener('click', addCube);
    removeCubeButton.addEventListener('click', removeCube);

    generateRandomCubes();
});
```

4.2.2 script_ace_of_bases.js

```
document.addEventListener('DOMContentLoaded', () => {
    const canvas = document.getElementById('cubeCanvas');
    const ctx = canvas.getContext('2d');
    const composeButton = document.getElementById('composeButton');
    const decomposeButton = document.getElementById('decomposeButton');
    const addCubeButton = document.getElementById('addCubeButton');
    const removeCubeButton = document.getElementById('removeCubeButton');
    const selectedUnitsDisplay = document.getElementById('selectedUnits');
    const baseConversionDisplay = document.getElementById('baseConversion');
    const baseTenCountDisplay = document.getElementById('baseTenCount');
    const explanationPopup = document.getElementById('explanationPopup');
    const overflowPopup = document.getElementById('overflowPopup');

    let cubes = [];
    let selectedUnits = 0;
    let modulus = 0;
    let isDragging = false;
    let dragOffsetX, dragOffsetY;
    let draggedCube = null;
    let isSelecting = false;
    let selectionStartX, selectionStartY;

    const generateRandomCubes = () => {
        const count = Math.floor(Math.random() * 14) + 2;
        cubes = Array.from({ length: count }, (_, i) => ({
            x: Math.random() * (canvas.width - 20),
            y: Math.random() * (canvas.height - 20),
            size: 20,
        }));
        drawCubes();
    };

    const drawCubes = () => {
        ctx.clearRect(0, 0, canvas.width, canvas.height);
        cubes.forEach(cube => {
            ctx.fillStyle = 'blue';
            ctx.fillRect(cube.x, cube.y, cube.size, cube.size);
        });
    };

    const handleMouseDown = (e) => {
        const rect = canvas.getBoundingClientRect();
        const startX = e.clientX - rect.left;
        const startY = e.clientY - rect.top;

        draggedCube = cubes.find(cube => (
            startX >= cube.x && startX <= cube.x + cube.size &&
            startY >= cube.y && startY <= cube.y + cube.size
        ));

        if (draggedCube) {
            isDragging = true;
            dragOffsetX = startX - draggedCube.x;
            dragOffsetY = startY - draggedCube.y;
        } else {
            isSelecting = true;
            selectionStartX = startX;
            selectionStartY = startY;
            canvas.addEventListener('mousemove', handleMouseMove);
            canvas.addEventListener('mouseup', handleMouseUp);
        }
    };

    const handleMouseMove = (e) => {
        if (isSelecting) {
            const rect = canvas.getBoundingClientRect();
            const currentX = e.clientX - rect.left;
            const currentY = e.clientY - rect.top;
            ctx.clearRect(0, 0, canvas.width, canvas.height);
            drawCubes();
            ctx.strokeStyle = 'red';
            ctx.strokeRect(selectionStartX, selectionStartY, currentX - selectionStartX, currentY -
            ↪  selectionStartY);
        }
    };

    const handleMouseUp = (e) => {
        if (isSelecting) {
            const rect = canvas.getBoundingClientRect();
            const endX = e.clientX - rect.left;
            const endY = e.clientY - rect.top;
            const selected = cubes.filter(cube => (
                cube.x >= Math.min(selectionStartX, endX) && cube.x <= Math.max(selectionStartX, endX) &&
                cube.y >= Math.min(selectionStartY, endY) && cube.y <= Math.max(selectionStartY, endY)
            ));
            selectedUnits = Math.min(selected.length, 15);
            selectedUnitsDisplay.textContent = `Selected Units: ${selectedUnits}`;
            modulus = selectedUnits;

            if (selectedUnits > 10) {
                explanationPopup.style.display = 'block';
            } else {
                explanationPopup.style.display = 'none';
            }

            isSelecting = false;
            canvas.removeEventListener('mousemove', handleMouseMove);
            canvas.removeEventListener('mouseup', handleMouseUp);
        }
    };

    const handleTouchStart = (e) => {
        const touch = e.touches[0];
        const rect = canvas.getBoundingClientRect();
        const startX = touch.clientX - rect.left;
        const startY = touch.clientY - rect.top;

        draggedCube = cubes.find(cube => (
            startX >= cube.x && startX <= cube.x + cube.size &&
            startY >= cube.y && startY <= cube.y + cube.size
        ));

        if (draggedCube) {
            isDragging = true;
            dragOffsetX = startX - draggedCube.x;
            dragOffsetY = startY - draggedCube.y;
        } else {
            isSelecting = true;
            selectionStartX = startX;
            selectionStartY = startY;
            canvas.addEventListener('touchmove', handleTouchMove);
            canvas.addEventListener('touchend', handleTouchEnd);
        }
    };

    const handleTouchMove = (e) => {
        if (isSelecting) {
            const touch = e.touches[0];
            const rect = canvas.getBoundingClientRect();
            const currentX = touch.clientX - rect.left;
            const currentY = touch.clientY - rect.top;
            ctx.clearRect(0, 0, canvas.width, canvas.height);
            drawCubes();
            ctx.strokeStyle = 'red';
            ctx.strokeRect(selectionStartX, selectionStartY, currentX - selectionStartX, currentY -
            ↪  selectionStartY);
        } else if (isDragging && draggedCube) {
            const touch = e.touches[0];
            const rect = canvas.getBoundingClientRect();
            draggedCube.x = touch.clientX - rect.left - dragOffsetX;
            draggedCube.y = touch.clientY - rect.top - dragOffsetY;
            drawCubes();
        }
    };

    const handleTouchEnd = (e) => {
        if (isSelecting) {
            const rect = canvas.getBoundingClientRect();
            const touch = e.changedTouches[0];
            const endX = touch.clientX - rect.left;
            const endY = touch.clientY - rect.top;
            const selected = cubes.filter(cube => (
                cube.x >= Math.min(selectionStartX, endX) && cube.x <= Math.max(selectionStartX, endX) &&
                cube.y >= Math.min(selectionStartY, endY) && cube.y <= Math.max(selectionStartY, endY)
            ));
            selectedUnits = Math.min(selected.length, 15);
            selectedUnitsDisplay.textContent = `Selected Units: ${selectedUnits}`;
            modulus = selectedUnits;

            if (selectedUnits > 10) {
                explanationPopup.style.display = 'block';
            } else {
                explanationPopup.style.display = 'none';
            }

            isSelecting = false;
            canvas.removeEventListener('touchmove', handleTouchMove);
            canvas.removeEventListener('touchend', handleTouchEnd);
        }
    };

    const handleDrag = (e) => {
        if (isDragging && draggedCube) {
            const rect = canvas.getBoundingClientRect();
            draggedCube.x = e.clientX - rect.left - dragOffsetX;
            draggedCube.y = e.clientY - rect.top - dragOffsetY;
            drawCubes();
        }
    };

    const handleDragEnd = (e) => {
        isDragging = false;
        draggedCube = null;
    };

    const updateBaseConversion = () => {
        if (modulus > 1) {
            const base = modulus;
            const baseStr = convertToBase(cubes.length, base);

            if (baseStr.length > 4) {
                overflowPopup.style.display = 'block';
                return;
            }

            const baseComponents = { rods: 0, flats: 0, cubes3D: 0, units: 0 };
            let count = cubes.length;

            while (count > 0) {
                if (count >= base * base * base) {
```

```
                baseComponents.cubes3D++;
                count -= base * base * base;
            } else if (count >= base * base) {
                baseComponents.flats++;
                count -= base * base;
            } else if (count >= base) {
                baseComponents.rods++;
                count -= base;
            } else {
                baseComponents.units++;
                count--;
            }
        }

        baseConversionDisplay.textContent = `Base ${base}: ${baseStr}`;
        baseTenCountDisplay.textContent = `Base 10: ${cubes.length}`;
        drawBaseComponents(base, baseComponents);
    }
};

const convertToBase = (number, base) => {
    const digitMap = { 10: 'T', 11: 'E', 12: 'D', 13: 'R', 14: 'F' };
    let result = '';
    while (number > 0) {
        let digit = number % base;
        if (digit >= 10 && digit <= 14) {
            digit = digitMap[digit];
        }
        result = digit.toString() + result;
        number = Math.floor(number / base);
    }
    return result;
};

const drawBaseComponents = (base, baseComponents) => {
    ctx.clearRect(0, 0, canvas.width, canvas.height);
    let xOffset = 0;
    let yOffset = 0;

    // Draw units
    for (let i = 0; i < baseComponents.units; i++) {
        ctx.fillStyle = 'blue';
        ctx.fillRect(xOffset, yOffset, 20, 20);
        xOffset += 25;
        if (xOffset > canvas.width - 20) {
            xOffset = 0;
            yOffset += 25;
        }
    }

    // Draw rods
    for (let i = 0; i < baseComponents.rods; i++) {
        ctx.fillStyle = 'green';
        ctx.fillRect(xOffset, yOffset, 20 * base, 20);
        for (let j = 0; j < base; j++) {
            ctx.strokeStyle = 'black';
            ctx.strokeRect(xOffset + j * 20, yOffset, 20, 20);
        }
        xOffset += 20 * base + 5;
        if (xOffset > canvas.width - 20 * base) {
            xOffset = 0;
            yOffset += 25;
        }
    }

    // Draw flats
    for (let i = 0; i < baseComponents.flats; i++) {
        ctx.fillStyle = 'yellow';
        ctx.fillRect(xOffset, yOffset, 20 * base, 20 * base);
        for (let j = 0; j < base; j++) {
            for (let k = 0; k < base; k++) {
                ctx.strokeStyle = 'black';
                ctx.strokeRect(xOffset + j * 20, yOffset + k * 20, 20, 20);
            }
        }
```

```
            xOffset += 20 * base + 5;
            if (xOffset > canvas.width - 20 * base) {
                xOffset = 0;
                yOffset += 20 * base + 5;
            }
        }

        // Draw 3D cubes
        for (let i = 0; i < baseComponents.cubes3D; i++) {
            ctx.fillStyle = 'red';
            const cubeSize = 20 * base;
            const depth = cubeSize / 3;

            // Draw front face
            ctx.fillRect(xOffset, yOffset, cubeSize, cubeSize);
            ctx.strokeStyle = 'black';
            for (let j = 0; j < base; j++) {
                for (let k = 0; k < base; k++) {
                    ctx.strokeRect(xOffset + j * 20, yOffset + k * 20, 20, 20);
                }
            }

            // Draw top face
            ctx.beginPath();
            ctx.moveTo(xOffset, yOffset);
            ctx.lineTo(xOffset + depth, yOffset - depth);
            ctx.lineTo(xOffset + cubeSize + depth, yOffset - depth);
            ctx.lineTo(xOffset + cubeSize, yOffset);
            ctx.closePath();
            ctx.fillStyle = 'rgba(255, 0, 0, 0.8)';
            ctx.fill();
            ctx.stroke();

            // Draw right face
            ctx.beginPath();
            ctx.moveTo(xOffset + cubeSize, yOffset);
            ctx.lineTo(xOffset + cubeSize + depth, yOffset - depth);
            ctx.lineTo(xOffset + cubeSize + depth, yOffset + cubeSize - depth);
            ctx.lineTo(xOffset + cubeSize, yOffset + cubeSize);
            ctx.closePath();
            ctx.fillStyle = 'rgba(255, 0, 0, 0.6)';
            ctx.fill();
            ctx.stroke();

            xOffset += cubeSize + depth + 5;
            if (xOffset > canvas.width - cubeSize) {
                xOffset = 0;
                yOffset += cubeSize + depth + 5;
            }
        }
    };

    const decomposeCubes = () => {
        cubes.forEach(cube => {
            cube.x = Math.random() * (canvas.width - 20);
            cube.y = Math.random() * (canvas.height - 20);
        });
        drawCubes();
        selectedUnits = 0;
        selectedUnitsDisplay.textContent = `Selected Units: ${selectedUnits}`;
        baseConversionDisplay.textContent = '';
        baseTenCountDisplay.textContent = '';
    };

    const addCube = () => {
        cubes.push({
            x: Math.random() * (canvas.width - 20),
            y: Math.random() * (canvas.height - 20),
            size: 20,
        });
        drawCubes();
    };

    const removeCube = () => {
        if (cubes.length > 0) {
            cubes.pop();
            drawCubes();
        }
    };

    canvas.addEventListener('mousedown', handleMouseDown);
    canvas.addEventListener('touchstart', handleTouchStart);
    canvas.addEventListener('mousemove', handleDrag);
    canvas.addEventListener('mouseup', handleDragEnd);
    canvas.addEventListener('touchmove', handleTouchMove);
    canvas.addEventListener('touchend', handleDragEnd);

    composeButton.addEventListener('click', updateBaseConversion);
    decomposeButton.addEventListener('click', decomposeCubes);
    addCubeButton.addEventListener('click', addCube);
    removeCubeButton.addEventListener('click', removeCube);

    generateRandomCubes();
});
```

4.3 CSS Files

4.3.1 styles.css

```css
body {
    font-family: Arial, sans-serif;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
    margin: 0;
    background-color: #f9f9f9;
}

.container {
    text-align: center;
    border: 1px solid #ccc;
    padding: 20px;
    border-radius: 10px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    background-color: white;
}

canvas {
    border: 1px solid #000;
}

.controls {
    margin-top: 20px;
}

button {
    margin: 5px;
}

.results {
    margin-top: 20px;
}
```

4.3.2 styles_ace_of_bases.css

```css
body {
    font-family: Arial, sans-serif;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
    margin: 0;
    background-color: #f9f9f9;
}
```

```

.container {
    text-align: center;
    border: 1px solid #ccc;
    padding: 20px;
    border-radius: 10px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    background-color: white;
}

canvas {
    border: 1px solid #000;
}

.controls,
button,
.results {
    margin-top: 20px;
}

button {
    margin: 5px;
}
```

5 Configuration Files

5.1 Fraction_Bars.code-workspace

```
{
	"folders": [
		{
			"path": "."
		},
		{
			"path": "../../../Desktop/Spring 2025/GPT4_1/Fraction_Bars_files"
		}
	],
	"settings": {
		"liveServer.settings.multiRootWorkspaceName": "Fraction_Bars"
	}
}
```