

**Interactive X-Sheet\interactive xsheet working v1-06\_01.html**

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>Interactive Animation X-Sheet with Audio Waveform</title>
8   <script src="https://cdnjs.cloudflare.com/ajax/libs/jspdf/2.5.1/jspdf.umd.min.js">
9 </script>
10  <script
11    src="https://cdnjs.cloudflare.com/ajax/libs/html2canvas/1.4.1/html2canvas.min.js"></script>
12
13  <style>
14    /* General styling */
15    body {
16      font-family: Arial, sans-serif;
17      font-size: 10pt;
18      line-height: 1.2;
19      margin: 0;
20      padding: 10px;
21    }
22
23    .controls {
24      background-color: #f5f5f5;
25      padding: 10px;
26      margin-bottom: 15px;
27      border-radius: 5px;
28      display: flex;
29      flex-wrap: wrap;
30      gap: 10px;
31      align-items: center;
32    }
33
34    .controls select,
35    .controls input,
36    .controls button {
37      padding: 6px 10px;
38      border: 1px solid #ccc;
39      border-radius: 4px;
40    }
41
42    .controls button {
43      background-color: #4CAF50;
44      color: white;
45      border: none;
46      cursor: pointer;
47      transition: background-color 0.3s;
48    }
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
```

```
47     .controls button:hover {
48         background-color: #45a049;
49     }
50
51     #pdf-button {
52         background-color: #f44336;
53     }
54
55     #pdf-button:hover {
56         background-color: #d32f2f;
57     }
58
59     #print-button {
60         background-color: #2196F3;
61     }
62
63     #print-button:hover {
64         background-color: #0b7dda;
65     }
66
67     #audio-button {
68         background-color: #9c27b0;
69     }
70
71     #audio-button:hover {
72         background-color: #7b1fa2;
73     }
74
75     .header {
76         text-align: center;
77         margin-bottom: 5px;
78     }
79
80     .title {
81         font-size: 14pt;
82         font-weight: bold;
83         margin-bottom: 5px;
84     }
85
86     .metadata {
87         display: grid;
88         grid-template-columns: repeat(3, 1fr);
89         gap: 5px;
90         margin-bottom: 10px;
91     }
92
93     .metadata div {
94         border: 1px solid #000;
95         padding: 3px 5px;
96     }
```

```
97
98     .metadata span {
99         font-weight: bold;
100        margin-right: 5px;
101    }
102
103    .metadata input {
104        border: none;
105        width: 70%;
106        font-family: Arial, sans-serif;
107        font-size: 9pt;
108    }
109
110    table {
111        width: 100%;
112        border-collapse: collapse;
113        table-layout: fixed;
114    }
115
116    th,
117    td {
118        border: 1px solid #000;
119        padding: 2px 4px;
120        vertical-align: top;
121        height: 20px;
122        overflow: hidden;
123    }
124
125    th {
126        background-color: #eee;
127        font-weight: bold;
128        text-align: center;
129        font-size: 9pt;
130    }
131
132    .action-col {
133        width: 16%;
134    }
135
136    .frame-col {
137        width: 4%;
138        text-align: center;
139    }
140
141    .waveform-col {
142        width: 10%;
143        padding: 0;
144        position: relative;
145    }
146
```

```
147     .dialogue-col {
148         width: 10%;
149         text-align: center;
150     }
151
152     .sound-col {
153         width: 9%;
154         text-align: center;
155     }
156
157     .technical-col {
158         width: 9%;
159     }
160
161     .extra1-col {
162         width: 8%;
163     }
164
165     .extra2-col {
166         width: 8%;
167     }
168
169     .camera-col {
170         width: 12%;
171     }
172
173     .waveform-container {
174         position: absolute;
175         width: 100%;
176         top: 0;
177         left: 0;
178         z-index: 10;
179         pointer-events: none;
180     }
181
182     .waveform-overlay {
183         position: absolute;
184         width: 100%;
185         height: 100%;
186         top: 0;
187         left: 0;
188         z-index: 11;
189         pointer-events: auto;
190         cursor: crosshair;
191     }
192
193     .waveform-marker {
194         position: absolute;
195         width: 100%;
196         height: 20px;
```

```
197     background-color: rgba(255, 255, 0, 0.2);  
198     pointer-events: none;  
199     text-align: center;  
200     font-size: 7pt;  
201     line-height: 18px;  
202     color: #666;  
203     z-index: 20;  
204 }  
205  
206 .waveform-canvas {  
207     position: absolute;  
208     top: 0;  
209     left: 0;  
210     width: 100%;  
211     z-index: 15;  
212     pointer-events: none;  
213 }  
214  
215 .waveform-col-container {  
216     position: relative;  
217 }  
218  
219 .phonetic-label {  
220     position: absolute;  
221     background-color: rgba(255, 255, 255, 0.8);  
222     border: 1px solid #ccc;  
223     border-radius: 2px;  
224     font-size: 7pt;  
225     padding: 1px 2px;  
226     z-index: 3;  
227     pointer-events: none;  
228 }  
229  
230 .footer {  
231     font-size: 8pt;  
232     text-align: center;  
233     margin-top: 5px;  
234     color: #333;  
235     font-style: italic;  
236 }  
237  
238 [contenteditable="true"] {  
239     min-height: 18px;  
240     cursor: text;  
241 }  
242  
243 [contenteditable="true"]:focus {  
244     background-color: #f0f7ff;  
245     outline: none;  
246 }
```

```
247
248     [contenteditable="true"]::empty:before {
249         content: attr(data-placeholder);
250         color: #888;
251         font-style: italic;
252     }
253
254     .frame-number {
255         background-color: #eee;
256         font-weight: bold;
257         text-align: center;
258     }
259
260     .modified {
261         background-color: #ffffacd;
262     }
263
264     .selected-cell {
265         background-color: rgba(0, 123, 255, 0.2) !important;
266         outline: 2px solid #0d6efd;
267     }
268
269     .status {
270         margin-top: 10px;
271         padding: 5px;
272         background-color: #f0f0f0;
273         border-radius: 4px;
274         font-style: italic;
275         color: #555;
276     }
277
278     #audio-controls {
279         display: flex;
280         flex-wrap: wrap;
281         gap: 5px;
282         align-items: center;
283         margin-top: 5px;
284         padding: 5px;
285         background-color: #f9f9f9;
286         border-radius: 4px;
287     }
288
289     #audio-controls button {
290         padding: 4px 8px;
291         background-color: #673ab7;
292         color: white;
293         border: none;
294         border-radius: 3px;
295         cursor: pointer;
296     }
```

```
297
298     #audio-controls button:hover {
299         background-color: #5e35b1;
300     }
301
302     #audio-info {
303         font-size: 8pt;
304         color: #333;
305     }
306
307     #phonetic-input {
308         position: absolute;
309         z-index: 100;
310         background: white;
311         border: 1px solid #ccc;
312         padding: 5px;
313         border-radius: 4px;
314         box-shadow: 0 2px 4px rgba(0, 0, 0, 0.2);
315         display: none;
316     }
317
318     #audio-upload {
319         display: none;
320     }
321
322     /* Drawing system specific styles */
323     .drawing-toolbar button {
324         transition: background-color 0.2s, color 0.2s;
325     }
326
327     .drawing-toolbar button:hover {
328         background-color: #e6e6e6 !important;
329     }
330
331     .drawing-toolbar button.active {
332         background-color: #4CAF50 !important;
333         color: white !important;
334     }
335
336     .drawing-layer-container {
337         pointer-events: none;
338     }
339
340     .drawing-layer-container canvas {
341         pointer-events: none;
342         touch-action: none;
343     }
344
345     /* Print specific styles */
346     @media print {
```

```
347     .controls,  
348     button,  
349     #frame-count-container,  
350     .status,  
351     #audio-controls,  
352     #phonetic-input,  
353     .drawing-toolbar {  
354         display: none !important;  
355     }  
356  
357     body {  
358         margin: 0;  
359         padding: 0;  
360     }  
361  
362     @page {  
363         size: auto;  
364         margin: 0.5cm;  
365     }  
366  
367     thead {  
368         display: table-header-group;  
369     }  
370  
371     tfoot {  
372         display: table-footer-group;  
373     }  
374  
375     tr {  
376         page-break-inside: avoid;  
377     }  
378  
379     /* Better waveform printing */  
380     .waveform-col {  
381         position: relative !important;  
382         overflow: hidden !important;  
383     }  
384  
385     .print-waveform-container {  
386         display: block !important;  
387         position: absolute !important;  
388         z-index: 1000 !important;  
389         pointer-events: none !important;  
390         overflow: hidden !important;  
391     }  
392  
393     .waveform-col * {  
394         page-break-inside: avoid !important;  
395     }  
396
```

```
397     /* ensure the waveform clone prints correctly */
398     .print-waveform-clone {
399         display: block !important;
400         position: absolute !important;
401         z-index: 1000 !important;
402         pointer-events: none !important;
403     }
404
405     .cell-waveform-window {
406         position: relative !important;
407         width: 100% !important;
408         height: 100% !important;
409         overflow: hidden !important;
410     }
411
412     /* Hide original waveform during print */
413     body>.waveform-container {
414         display: none !important;
415     }
416
417     .drawing-layer-container {
418         display: block !important;
419         position: absolute !important;
420         z-index: 1000 !important;
421         pointer-events: none !important;
422         page-break-inside: avoid !important;
423         page-break-after: avoid !important;
424     }
425
426     .drawing-layer-container.printing {
427         transform: translate(0, 0) !important; /* Prevent any transforms during print */
428     }
429
430     .drawing-layer-container canvas {
431         position: absolute !important;
432         display: block !important;
433         page-break-inside: avoid !important;
434     }
435     }
436     </style>
437 </head>
438
439 <body>
440     <div class="controls">
441         <div>
442             <label for="template-selector">Template:</label>
443             <select id="template-selector">
444                 <option value="large">11"x17" (96 Frames)</option>
445                 <option value="small">8"x10" (48 Frames)</option>
446             </select>
```

```
447     </div>
448
449     <div id="frame-count-container">
450         <label for="frame-count">Frames:</label>
451         <input type="number" id="frame-count" min="24" step="8" value="96">
452     </div>
453
454     <button id="audio-button">Import Audio</button>
455     <input type="file" id="audio-upload" accept="audio/*">
456
457     <button id="save-button">Save Project</button>
458     <button id="load-button">Load Project</button>
459     <button id="pdf-button">Export PDF</button>
460     <button id="print-button">Print</button>
461     <button id="add-rows-button">Add 8 Rows</button>
462     <button id="clear-button">Clear All</button>
463 </div>
464
465     <div id="audio-controls">
466         <button id="play-audio">Play/Pause</button>
467         <button id="stop-audio">Stop</button>
468         <input type="range" id="audio-scrubber" min="0" max="100" value="0" style="width: 200px;">
469         <span id="audio-info">No audio loaded</span>
470         <button id="add-phonetic">Add Phonetic Marker</button>
471         <div style="margin-left: 10px; color: #666; font-style: italic;">
472             ♦ TIP: Drag down the waveform column while holding the left mouse button to
473             scrub audio frame-by-frame<br>
474            💡 TIP: Click and drag to select multiple cells (use Ctrl+C to copy, Delete to
475             clear)
476         </div>
477     </div>
478
479     <div id="phonetic-input">
480         <input type="text" id="phonetic-text" placeholder="Enter phonetic sound">
481         <button id="save-phonetic">Save</button>
482         <button id="cancel-phonetic">Cancel</button>
483     </div>
484
485     <div id="printable-area">
486         <div class="header">
487             <div class="title">3D ANIMATION X-SHEET</div>
488         </div>
489
490         <div class="metadata">
491             <div><span>Project #:</span><input type="text" id="project-number"></div>
492             <div><span>DATE:</span><input type="date" id="project-date"></div>
493             <div><span>PAGE #:</span><input type="text" id="page-number"></div>
494             <div><span>ANIMATOR:</span><input type="text" id="animator-name"></div>
495             <div><span>VERSION:</span><input type="text" id="version-number"></div>
496             <div><span>Shot #:</span><input type="text" id="shot-number"></div>
```

```
495     </div>
496
497     <table id="xsheet-table">
498         <thead>
499             <tr>
500                 <th class="action-col">Action/Description</th>
501                 <th class="frame-col">Fr</th>
502                 <th class="waveform-col">Audio Waveform</th>
503                 <th class="dialogue-col">Dialogue</th>
504                 <th class="sound-col">Sound FX</th>
505                 <th class="technical-col">Tech. Notes</th>
506                 <th class="extra1-col">Extra 1</th>
507                 <th class="extra2-col">Extra 2</th>
508                 <th class="frame-col">Fr</th>
509                 <th class="camera-col">Camera Moves</th>
510             </tr>
511         </thead>
512         <tbody id="xsheet-body">
513             <!-- Rows will be generated via JavaScript -->
514         </tbody>
515     </table>
516
517     <div class="footer">
518         Bold lines mark 8-frame intervals. Double lines mark 24-frame intervals (24fps).
519         Left columns track character actions, middle columns for technical notes, right
for camera moves.
520     </div>
521 </div>
522
523     <div class="status" id="status-message"></div>
524
525     <script>
526         /**
527         * INTERACTIVE X-SHEET DRAWING TOOLS IMPLEMENTATION
528         *
529         * This code adds comprehensive drawing capabilities to the animation X-Sheet.
530         * Features include:
531         * - Multiple drawing layers
532         * - Various drawing tools (pen, line, arrows, shapes, text, images, animation
symbols)
533         * - Grid-aware annotation that can span multiple frames
534         * - Object selection and manipulation
535         * - Integration with saving, loading, and printing
536         */
537
538         /**
539         * DRAWING LAYER SYSTEM
540         * Manages the canvas layers that contain drawing objects
541         */
542         class DrawingLayerSystem {
```

```
543     constructor(xsheetTable) {
544         this.xsheetTable = xsheetTable;
545         this.layers = [];
546         this.activeLayerIndex = 0;
547         this.container = null;
548
549         this.init();
550     }
551
552     init() {
553         // Create container aligned with table
554         const tableRect = this.xsheetTable.getBoundingClientRect();
555         this.container = document.createElement('div');
556         this.container.className = 'drawing-layer-container';
557         this.container.style.position = 'absolute';
558         this.container.style.left = `${tableRect.left}px`;
559         this.container.style.top = `${tableRect.top}px`;
560         this.container.style.width = `${tableRect.width}px`;
561         this.container.style.height = `${tableRect.height}px`;
562         this.container.style.pointerEvents = 'none'; // Initially pass events
563         through
564         this.container.style.zIndex = '5';
565         document.body.appendChild(this.container);
566
567         // Create default background and foreground layers
568         this.addLayer('background');
569         this.addLayer('foreground');
570         this.setActiveLayer(1); // Set foreground as active by default
571
572         // Handle window resize and table changes
573         this.setupResizeHandling();
574     }
575
576     addLayer(name) {
577         const canvas = document.createElement('canvas');
578         canvas.className = `drawing-layer-${name}`;
579         canvas.width = this.container.clientWidth;
580         canvas.height = this.container.clientHeight;
581         canvas.style.position = 'absolute';
582         canvas.style.left = '0';
583         canvas.style.top = '0';
584         canvas.style.pointerEvents = 'none';
585
586         this.container.appendChild(canvas);
587
588         const layer = {
589             name: name,
590             canvas: canvas,
591             context: canvas.getContext('2d'),
592             objects: []
593         }
594     }
595 }
```

```
592         visible: true
593     };
594
595     this.layers.push(layer);
596     return this.layers.length - 1; // Return index of new layer
597 }
598
599 setActiveLayer(index) {
600     if (index >= 0 && index < this.layers.length) {
601         this.activeLayerIndex = index;
602         return true;
603     }
604     return false;
605 }
606
607 getActiveLayer() {
608     return this.layers[this.activeLayerIndex];
609 }
610 updateLayoutSize() {
611     const tableRect = this.xsheetTable.getBoundingClientRect();
612
613     // Store current drawings from each layer
614     const tempCanvases = this.layers.map(layer => {
615         const tempCanvas = document.createElement('canvas');
616         tempCanvas.width = layer.canvas.width;
617         tempCanvas.height = layer.canvas.height;
618         const tempCtx = tempCanvas.getContext('2d');
619         tempCtx.drawImage(layer.canvas, 0, 0);
620         return tempCanvas;
621     });
622
623     // Update container position and dimensions
624     this.container.style.position = 'absolute';
625     this.container.style.left = `${tableRect.left}px`;
626     this.container.style.top = `${tableRect.top}px`;
627     this.container.style.width = `${tableRect.width}px`;
628     this.container.style.height = `${tableRect.height}px`;
629
630     // Update each layer canvas
631     this.layers.forEach((layer, i) => {
632         const scaleX = tableRect.width / layer.canvas.width;
633         const scaleY = tableRect.height / layer.canvas.height;
634
635         layer.canvas.width = tableRect.width;
636         layer.canvas.height = tableRect.height;
637
638         // Redraw with scaling
639         layer.context.save();
640         layer.context.scale(scaleX, scaleY);
641         layer.context.drawImage(tempCanvases[i], 0, 0);
```

```
642         layer.context.restore();
643     });
644
645     // Force redraw of all objects
646     this.redrawAll();
647
648     // Dispatch a custom event that the layout was updated
649     document.dispatchEvent(new Event('drawing-layout-updated'));
650 }
651
652 enableDrawing() {
653     this.layers.forEach(layer => {
654         layer.canvas.style.pointerEvents = 'auto';
655     });
656 }
657
658 disableDrawing() {
659     this.layers.forEach(layer => {
660         layer.canvas.style.pointerEvents = 'none';
661     });
662 }
663
664 clearLayer(layerIndex) {
665     if (layerIndex >= 0 && layerIndex < this.layers.length) {
666         const layer = this.layers[layerIndex];
667         layer.context.clearRect(0, 0, layer.canvas.width, layer.canvas.height);
668         layer.objects = [];
669     }
670 }
671
672 clearAllLayers() {
673     this.layers.forEach((layer, index) => {
674         this.clearLayer(index);
675     });
676 }
677
678 // Convert screen coordinates to canvas coordinates
679 screenToCanvas(screenX, screenY) {
680     const containerRect = this.container.getBoundingClientRect();
681     return {
682         x: screenX - containerRect.left,
683         y: screenY - containerRect.top
684     };
685 }
686
687 // Convert frame/column to canvas coordinates (for multi-frame spanning)
688 gridToCanvas(frame, column) {
689     const cell = document.querySelector(`tr.frame-${frame} td:nth-child(${column})`);
690     if (!cell) return null;
```

```
691
692     const cellRect = cell.getBoundingClientRect();
693     const containerRect = this.container.getBoundingClientRect();
694
695     return {
696         x: cellRect.left - containerRect.left + cellRect.width / 2,
697         y: cellRect.top - containerRect.top + cellRect.height / 2
698     };
699 }
700
701 // Add a drawing object to the active layer
702 addObject(object) {
703     const layer = this.getActiveLayer();
704     layer.objects.push(object);
705     this.redrawLayer(this.activeLayerIndex);
706     return object;
707 }
708
709 // Redraw a specific layer
710 redrawLayer(layerIndex) {
711     if (layerIndex >= 0 && layerIndex < this.layers.length) {
712         const layer = this.layers[layerIndex];
713         layer.context.clearRect(0, 0, layer.canvas.width, layer.canvas.height);
714
715         // Draw all objects in this layer
716         layer.objects.forEach(obj => {
717             if (obj.visible) {
718                 obj.draw(layer.context);
719             }
720         });
721     }
722 }
723
724 // Redraw all layers
725 redrawAll() {
726     this.layers.forEach(_, index) => {
727         this.redrawLayer(index);
728     });
729 }
730
731 // Find object under point
732 findObjectAt(x, y) {
733     // Check active layer first, then others in reverse order (top to bottom)
734     const activeLayer = this.getActiveLayer();
735
736     // Check active layer
737     for (let i = activeLayer.objects.length - 1; i >= 0; i--) {
738         const obj = activeLayer.objects[i];
739         if (obj.containsPoint(x, y)) {
740             return { object: obj, layerIndex: this.activeLayerIndex };
```

```
741         }
742     }
743
744     // Check other layers from top to bottom
745     for (let l = this.layers.length - 1; l >= 0; l--) {
746         if (l === this.activeLayerIndex) continue; // Skip active layer (already
checked)
747
748         const layer = this.layers[l];
749         if (!layer.visible) continue;
750
751         for (let i = layer.objects.length - 1; i >= 0; i--) {
752             const obj = layer.objects[i];
753             if (obj.containsPoint(x, y)) {
754                 return { object: obj, layerIndex: l };
755             }
756         }
757     }
758
759     return null;
760 }
761
762 // Remove object
763 removeObject(object, layerIndex) {
764     const layer = layerIndex !== undefined ? this.layers[layerIndex] :
this.getActiveLayer();
765     const index = layer.objects.indexOf(object);
766     if (index !== -1) {
767         layer.objects.splice(index, 1);
768         this.redrawLayer(layerIndex !== undefined ? layerIndex :
this.activeLayerIndex);
769         return true;
770     }
771     return false;
772 }
773 }
774
775 /**
776 * DRAWING OBJECT MODEL
777 * Defines the object classes for different types of drawings
778 */
779
780 // Base class for all drawing objects
781 class DrawingObject {
782     constructor(props = {}) {
783         this.x = props.x || 0;
784         this.y = props.y || 0;
785         this.color = props.color || '#000000';
786         this.lineWidth = props.lineWidth || 2;
787         this.visible = props.visible !== undefined ? props.visible : true;
788         this.selected = false;
```

```
789         this.type = 'drawingObject'; // Base type
790     }
791
792     draw(context) {
793         // Base drawing functionality
794         if (this.selected) {
795             this.drawSelectionMarkers(context);
796         }
797     }
798
799     drawSelectionMarkers(context) {
800         // Draw selection handles (default implementation)
801         context.save();
802         context.strokeStyle = '#0099ff';
803         context.lineWidth = 1;
804         context.setLineDash([5, 3]);
805
806         // Default is to draw a box around the object
807         // This should be overridden by subclasses with specific bounds
808         const bounds = this.getBounds();
809         context.strokeRect(
810             bounds.x - 2,
811             bounds.y - 2,
812             bounds.width + 4,
813             bounds.height + 4
814         );
815
816         context.restore();
817     }
818
819     getBounds() {
820         // Default implementation - should be overridden
821         return { x: this.x, y: this.y, width: 0, height: 0 };
822     }
823
824     containsPoint(x, y) {
825         // Default implementation - should be overridden
826         return false;
827     }
828
829     move(dx, dy) {
830         this.x += dx;
831         this.y += dy;
832     }
833
834     toJSON() {
835         return {
836             type: this.type,
837             x: this.x,
838             y: this.y,
```

```
839         color: this.color,
840         lineWidth: this.lineWidth,
841         visible: this.visible
842     );
843 }
844
845     static fromJSON(data) {
846         // Factory method to create objects from JSON
847         // This will be overridden by subclasses
848         return new DrawingObject(data);
849     }
850 }
851
852 // Line object
853 class LineObject extends DrawingObject {
854     constructor(props = {}) {
855         super(props);
856         this.x2 = props.x2 || 0;
857         this.y2 = props.y2 || 0;
858         this.type = 'line';
859         this.dashPattern = props.dashPattern || [];
860     }
861
862     draw(context) {
863         context.save();
864         context.beginPath();
865         context.strokeStyle = this.color;
866         context.lineWidth = this.lineWidth;
867
868         if (this.dashPattern.length > 0) {
869             context.setLineDash(this.dashPattern);
870         }
871
872         context.moveTo(this.x, this.y);
873         context.lineTo(this.x2, this.y2);
874         context.stroke();
875         context.restore();
876
877         super.draw(context);
878     }
879
880     getBounds() {
881         const minX = Math.min(this.x, this.x2);
882         const minY = Math.min(this.y, this.y2);
883         const width = Math.abs(this.x2 - this.x);
884         const height = Math.abs(this.y2 - this.y);
885
886         return { x: minX, y: minY, width, height };
887     }
888 }
```

```
889 |     containsPoint(x, y) {
890 |         // Check if point is near the line
891 |         const lineLength = Math.sqrt(
892 |             Math.pow(this.x2 - this.x, 2) + Math.pow(this.y2 - this.y, 2)
893 |         );
894 |
895 |         // If line is too short, use a minimum distance
896 |         if (lineLength < 1) {
897 |             const dx = x - this.x;
898 |             const dy = y - this.y;
899 |             return Math.sqrt(dx * dx + dy * dy) <= 5;
900 |         }
901 |
902 |         // Calculate distance from point to line segment
903 |         const t = ((x - this.x) * (this.x2 - this.x) + (y - this.y) * (this.y2 -
904 |             this.y)) / (lineLength * lineLength);
905 |
906 |         if (t < 0) {
907 |             // Point is beyond start point
908 |             const dx = x - this.x;
909 |             const dy = y - this.y;
910 |             return Math.sqrt(dx * dx + dy * dy) <= 5;
911 |         }
912 |
913 |         if (t > 1) {
914 |             // Point is beyond end point
915 |             const dx = x - this.x2;
916 |             const dy = y - this.y2;
917 |             return Math.sqrt(dx * dx + dy * dy) <= 5;
918 |
919 |         // Calculate perpendicular distance
920 |         const px = this.x + t * (this.x2 - this.x);
921 |         const py = this.y + t * (this.y2 - this.y);
922 |         const dx = x - px;
923 |         const dy = y - py;
924 |         return Math.sqrt(dx * dx + dy * dy) <= 5;
925 |     }
926 |
927 |     move(dx, dy) {
928 |         super.move(dx, dy);
929 |         this.x2 += dx;
930 |         this.y2 += dy;
931 |     }
932 |
933 |     toJSON() {
934 |         const json = super.toJSON();
935 |         return {
936 |             ...json,
937 |             x2: this.x2,
```

```
938     y2: this.y2,
939     dashPattern: this.dashPattern
940   );
941 }
942
943   static fromJSON(data) {
944     return new LineObject(data);
945   }
946 }
947
948 // Arrow object (extends Line)
949 class ArrowObject extends LineObject {
950   constructor(props = {}) {
951     super(props);
952     this.arrowSize = props.arrowSize || 10;
953     this.type = 'arrow';
954   }
955
956   draw(context) {
957     // Draw the line part
958     super.draw(context);
959
960     // Draw the arrowhead
961     const angle = Math.atan2(this.y2 - this.y, this.x2 - this.x);
962     context.save();
963     context.fillStyle = this.color;
964     context.beginPath();
965     context.moveTo(this.x2, this.y2);
966     context.lineTo(
967       this.x2 - this.arrowSize * Math.cos(angle - Math.PI/6),
968       this.y2 - this.arrowSize * Math.sin(angle - Math.PI/6)
969     );
970     context.lineTo(
971       this.x2 - this.arrowSize * Math.cos(angle + Math.PI/6),
972       this.y2 - this.arrowSize * Math.sin(angle + Math.PI/6)
973     );
974     context.closePath();
975     context.fill();
976     context.restore();
977   }
978
979   toJSON() {
980     const json = super.toJSON();
981     return {
982       ...json,
983       arrowSize: this.arrowSize
984     };
985   }
986
987   static fromJSON(data) {
```

```
988         return new ArrowObject(data);
989     }
990 }
991
992 // Rectangle object
993 class RectangleObject extends DrawingObject {
994     constructor(props = {}) {
995         super(props);
996         this.width = props.width || 0;
997         this.height = props.height || 0;
998         this.fill = props.fill || false;
999         this.fillColor = props.fillColor || this.color;
1000        this.type = 'rectangle';
1001    }
1002
1003    draw(context) {
1004        context.save();
1005        context.strokeStyle = this.color;
1006        context.lineWidth = this.lineWidth;
1007
1008        // Draw rectangle
1009        if (this.fill) {
1010            context.fillStyle = this.fillColor;
1011            context.fillRect(this.x, this.y, this.width, this.height);
1012        }
1013        context.strokeRect(this.x, this.y, this.width, this.height);
1014        context.restore();
1015
1016        super.draw(context);
1017    }
1018
1019    getBounds() {
1020        return {
1021            x: this.x,
1022            y: this.y,
1023            width: this.width,
1024            height: this.height
1025        };
1026    }
1027
1028    containsPoint(x, y) {
1029        // Check if point is inside or near the edge of the rectangle
1030        if (this.fill) {
1031            // For filled rectangles, check if point is inside
1032            return (
1033                x >= this.x && x <= this.x + this.width &&
1034                y >= this.y && y <= this.y + this.height
1035            );
1036        } else {
1037            // For unfilled rectangles, check if point is near the edges

```

```
1038     const nearLeft = Math.abs(x - this.x) <= 5;
1039     const nearRight = Math.abs(x - (this.x + this.width)) <= 5;
1040     const nearTop = Math.abs(y - this.y) <= 5;
1041     const nearBottom = Math.abs(y - (this.y + this.height)) <= 5;
1042
1043     return (
1044         (nearLeft || nearRight) && (y >= this.y && y <= this.y +
1045             this.height) ||
1046             (nearTop || nearBottom) && (x >= this.x && x <= this.x + this.width)
1047         );
1048     }
1049
1050     toJSON() {
1051         const json = super.toJSON();
1052         return {
1053             ...json,
1054             width: this.width,
1055             height: this.height,
1056             fill: this.fill,
1057             fillColor: this.fillColor
1058         };
1059     }
1060
1061     static fromJSON(data) {
1062         return new RectangleObject(data);
1063     }
1064 }
1065
1066 // Circle/Ellipse object
1067 class EllipseObject extends DrawingObject {
1068     constructor(props = {}) {
1069         super(props);
1070         this.radiusX = props.radiusX || 0;
1071         this.radiusY = props.radiusY || 0;
1072         this.fill = props.fill || false;
1073         this.fillColor = props.fillColor || this.color;
1074         this.type = 'ellipse';
1075     }
1076
1077     draw(context) {
1078         context.save();
1079         context.beginPath();
1080         context.strokeStyle = this.color;
1081         context.lineWidth = this.lineWidth;
1082
1083         // Draw ellipse
1084         context.ellipse(
1085             this.x,
1086             this.y,
```

```
1087         this.radiusX,  
1088         this.radiusY,  
1089         0,  
1090         0,  
1091         2 * Math.PI  
1092     );  
1093  
1094     if (this.fill) {  
1095         context.fillStyle = this.fillColor;  
1096         context.fill();  
1097     }  
1098     context.stroke();  
1099     context.restore();  
1100  
1101     super.draw(context);  
1102 }  
1103  
1104 getBounds() {  
1105     return {  
1106         x: this.x - this.radiusX,  
1107         y: this.y - this.radiusY,  
1108         width: this.radiusX * 2,  
1109         height: this.radiusY * 2  
1110     };  
1111 }  
1112  
1113 containsPoint(x, y) {  
1114     // Check if point is inside or near the edge of the ellipse  
1115     const normalizedX = (x - this.x) / this.radiusX;  
1116     const normalizedY = (y - this.y) / this.radiusY;  
1117     const distance = Math.sqrt(normalizedX * normalizedX + normalizedY *  
normalizedY);  
1118  
1119     if (this.fill) {  
1120         // For filled ellipses, check if point is inside  
1121         return distance <= 1.0;  
1122     } else {  
1123         // For unfilled ellipses, check if point is near the edge  
1124         return Math.abs(distance - 1.0) <= 5 / this.radiusX;  
1125     }  
1126 }  
1127  
1128 toJSON() {  
1129     const json = super.toJSON();  
1130     return {  
1131         ...json,  
1132         radiusX: this.radiusX,  
1133         radiusY: this.radiusY,  
1134         fill: this.fill,  
1135         fillColor: this.fillColor
```

```
1136         };
1137     }
1138
1139     static fromJSON(data) {
1140         return new EllipseObject(data);
1141     }
1142 }
1143
1144 // Text object
1145 class TextObject extends DrawingObject {
1146     constructor(props = {}) {
1147         super(props);
1148         this.text = props.text || '';
1149         this.fontSize = props.fontSize || 14;
1150         this.fontFamily = props.fontFamily || 'Arial, sans-serif';
1151         this.align = props.align || 'left';
1152         this.type = 'text';
1153     }
1154
1155     draw(context) {
1156         context.save();
1157         context.fillStyle = this.color;
1158         context.font = `${this.fontSize}px ${this.fontFamily}`;
1159         context.textAlign = this.align;
1160
1161         // Draw text
1162         context.fillText(this.text, this.x, this.y);
1163         context.restore();
1164
1165         super.draw(context);
1166     }
1167
1168     getBounds() {
1169         // Estimate text dimensions
1170         const dummyCanvas = document.createElement('canvas');
1171         const ctx = dummyCanvas.getContext('2d');
1172         ctx.font = `${this.fontSize}px ${this.fontFamily}`;
1173         const metrics = ctx.measureText(this.text);
1174         const height = this.fontSize; // Approximation
1175
1176         return {
1177             x: this.align === 'center' ? this.x - metrics.width / 2 :
1178                 this.align === 'right' ? this.x - metrics.width : this.x,
1179             y: this.y - height,
1180             width: metrics.width,
1181             height: height
1182         };
1183     }
1184
1185     containsPoint(x, y) {
```

```
1186     const bounds = this.getBounds();
1187     return (
1188         x >= bounds.x && x <= bounds.x + bounds.width &&
1189         y >= bounds.y && y <= bounds.y + bounds.height
1190     );
1191 }
1192
1193 toJSON() {
1194     const json = super.toJSON();
1195     return {
1196         ...json,
1197         text: this.text,
1198         fontSize: this.fontSize,
1199         fontFamily: this.fontFamily,
1200         align: this.align
1201     };
1202 }
1203
1204 static fromJSON(data) {
1205     return new TextObject(data);
1206 }
1207 }
1208
1209 // Image object
1210 class ImageObject extends DrawingObject {
1211     constructor(props = {}) {
1212         super(props);
1213         this.width = props.width || 0;
1214         this.height = props.height || 0;
1215         this.imageUrl = props.imageUrl || '';
1216         this.image = null;
1217         this.loaded = false;
1218         this.type = 'image';
1219
1220         // Load the image if provided
1221         if (this.imageUrl) {
1222             this.loadImage(this.imageUrl);
1223         }
1224     }
1225
1226     loadImage(url) {
1227         this.image = new Image();
1228         this.image.onload = () => {
1229             this.loaded = true;
1230
1231             // If dimensions not specified, use image dimensions
1232             if (this.width === 0 || this.height === 0) {
1233                 this.width = this.image.width;
1234                 this.height = this.image.height;
1235             }
1236     }
1237 }
```

```
1236
1237         // Trigger redraw
1238         document.dispatchEvent(new Event('xsheet-redraw'));
1239     };
1240     this.image.src = url;
1241 }
1242
1243 draw(context) {
1244     if (this.loaded && this.image) {
1245         context.save();
1246         context.drawImage(this.image, this.x, this.y, this.width, this.height);
1247         context.restore();
1248     } else if (!this.loaded) {
1249         // Draw placeholder while loading
1250         context.save();
1251         context.strokeStyle = '#999999';
1252         context.lineWidth = 1;
1253         context.strokeRect(this.x, this.y, this.width, this.height);
1254         context.font = '10px Arial';
1255         context.fillStyle = '#999999';
1256         context.fillText('Loading Image...', this.x + 5, this.y + 15);
1257         context.restore();
1258     }
1259
1260     super.draw(context);
1261 }
1262
1263 getBounds() {
1264     return {
1265         x: this.x,
1266         y: this.y,
1267         width: this.width,
1268         height: this.height
1269     };
1270 }
1271
1272 containsPoint(x, y) {
1273     return (
1274         x >= this.x && x <= this.x + this.width &&
1275         y >= this.y && y <= this.y + this.height
1276     );
1277 }
1278
1279 toJSON() {
1280     const json = super.toJSON();
1281     return {
1282         ...json,
1283         width: this.width,
1284         height: this.height,
1285         imageUrl: this.imageUrl
1286     };
1287 }
```

```
1286         };
1287     }
1288
1289     static fromJSON(data) {
1290         return new ImageObject(data);
1291     }
1292 }
1293
1294 // Symbol object (predefined animation symbols)
1295 class SymbolObject extends DrawingObject {
1296     constructor(props = {}) {
1297         super(props);
1298         this.symbolType = props.symbolType || 'default';
1299         this.scale = props.scale || 1.0;
1300         this.type = 'symbol';
1301     }
1302
1303     draw(context) {
1304         context.save();
1305         context.strokeStyle = this.color;
1306         context.fillStyle = this.color;
1307         context.lineWidth = this.lineWidth;
1308
1309         // Draw based on symbol type
1310         switch (this.symbolType) {
1311             case 'anticipation':
1312                 this.drawAnticipation(context);
1313                 break;
1314             case 'impact':
1315                 this.drawImpact(context);
1316                 break;
1317             case 'keyframe':
1318                 this.drawKeyframe(context);
1319                 break;
1320             case 'inbetween':
1321                 this.drawInbetween(context);
1322                 break;
1323             case 'hold':
1324                 this.drawHold(context);
1325                 break;
1326             default:
1327                 this.drawDefault(context);
1328         }
1329
1330         context.restore();
1331         super.draw(context);
1332     }
1333
1334     drawAnticipation(context) {
1335         context.save();
```

```
1336     context.translate(this.x, this.y);
1337     context.scale(this.scale, this.scale);
1338
1339     // Draw curved arrow going back
1340     context.beginPath();
1341     context.moveTo(0, 0);
1342     context.bezierCurveTo(-20, -5, -25, 10, -10, 15);
1343     context.stroke();
1344
1345     // Draw arrowhead
1346     context.beginPath();
1347     context.moveTo(-10, 15);
1348     context.lineTo(-5, 10);
1349     context.lineTo(-15, 5);
1350     context.closePath();
1351     context.fill();
1352
1353     context.restore();
1354 }
1355
1356 drawImpact(context) {
1357     context.save();
1358     context.translate(this.x, this.y);
1359     context.scale(this.scale, this.scale);
1360
1361     // Draw impact star
1362     for (let i = 0; i < 8; i++) {
1363         const angle = (i / 8) * Math.PI * 2;
1364         const innerRadius = 5;
1365         const outerRadius = 15;
1366
1367         context.beginPath();
1368         context.moveTo(
1369             innerRadius * Math.cos(angle),
1370             innerRadius * Math.sin(angle)
1371         );
1372         context.lineTo(
1373             outerRadius * Math.cos(angle),
1374             outerRadius * Math.sin(angle)
1375         );
1376         context.stroke();
1377     }
1378
1379     context.restore();
1380 }
1381
1382 drawKeyframe(context) {
1383     context.save();
1384     context.translate(this.x, this.y);
1385     context.scale(this.scale, this.scale);
```

```
1386
1387      // Draw diamond
1388      context.beginPath();
1389      context.moveTo(0, -10);
1390      context.lineTo(10, 0);
1391      context.lineTo(0, 10);
1392      context.lineTo(-10, 0);
1393      context.closePath();
1394      context.stroke();
1395      context.fill();
1396
1397      context.restore();
1398 }
1399
1400 drawInbetween(context) {
1401     context.save();
1402     context.translate(this.x, this.y);
1403     context.scale(this.scale, this.scale);
1404
1405     // Draw circle
1406     context.beginPath();
1407     context.arc(0, 0, 7, 0, Math.PI * 2);
1408     context.stroke();
1409
1410     context.restore();
1411 }
1412
1413 drawHold(context) {
1414     context.save();
1415     context.translate(this.x, this.y);
1416     context.scale(this.scale, this.scale);
1417
1418     // Draw horizontal bar
1419     context.beginPath();
1420     context.moveTo(-15, 0);
1421     context.lineTo(15, 0);
1422     context.lineWidth = this.lineWidth * 2;
1423     context.stroke();
1424
1425     context.restore();
1426 }
1427
1428 drawDefault(context) {
1429     context.save();
1430     context.translate(this.x, this.y);
1431     context.scale(this.scale, this.scale);
1432
1433     // Draw square
1434     context.beginPath();
1435     context.rect(-7, -7, 14, 14);
```

```
1436         context.stroke();
1437
1438         context.restore();
1439     }
1440
1441     getBounds() {
1442         // Approximate bounds based on symbol type
1443         const size = 20 * this.scale;
1444         return {
1445             x: this.x - size / 2,
1446             y: this.y - size / 2,
1447             width: size,
1448             height: size
1449         };
1450     }
1451
1452     containsPoint(x, y) {
1453         const bounds = this.getBounds();
1454         const dx = x - this.x;
1455         const dy = y - this.y;
1456         const distance = Math.sqrt(dx * dx + dy * dy);
1457
1458         // Use a radius-based check as most symbols are roughly circular
1459         return distance <= bounds.width / 2;
1460     }
1461
1462     toJSON() {
1463         const json = super.toJSON();
1464         return {
1465             ...json,
1466             symbolType: this.symbolType,
1467             scale: this.scale
1468         };
1469     }
1470
1471     static fromJSON(data) {
1472         return new SymbolObject(data);
1473     }
1474 }
1475
1476 // Free-form path (for pen/brush tools)
1477 class PathObject extends DrawingObject {
1478     constructor(props = {}) {
1479         super(props);
1480         this.points = props.points || [];
1481         this.smoothing = props.smoothing !== undefined ? props.smoothing : true;
1482         this.closed = props.closed || false;
1483         this.fill = props.fill || false;
1484         this.fillColor = props.fillColor || this.color;
1485         this.type = 'path';
1486     }
1487 }
```

```
1486 }
1487
1488     addPoint(x, y) {
1489         this.points.push({ x, y });
1490     }
1491
1492     draw(context) {
1493         if (this.points.length < 2) return;
1494
1495         context.save();
1496         context.beginPath();
1497         context.strokeStyle = this.color;
1498         context.lineWidth = this.lineWidth;
1499         context.lineJoin = 'round';
1500         context.lineCap = 'round';
1501
1502         // Start from first point
1503         context.moveTo(this.points[0].x, this.points[0].y);
1504
1505         if (this.smoothing && this.points.length > 2) {
1506             // Draw using bezier curves for smoothing
1507             for (let i = 1; i < this.points.length - 1; i++) {
1508                 const p1 = this.points[i];
1509                 const p2 = this.points[i + 1];
1510
1511                 const xc = (p1.x + p2.x) / 2;
1512                 const yc = (p1.y + p2.y) / 2;
1513
1514                 context.quadraticCurveTo(p1.x, p1.y, xc, yc);
1515             }
1516
1517             // Connect to the last point
1518             const last = this.points[this.points.length - 1];
1519             context.lineTo(last.x, last.y);
1520         } else {
1521             // Simple line segments
1522             for (let i = 1; i < this.points.length; i++) {
1523                 context.lineTo(this.points[i].x, this.points[i].y);
1524             }
1525         }
1526
1527         if (this.closed) {
1528             context.closePath();
1529         }
1530
1531         if (this.fill) {
1532             context.fillStyle = this.fillColor;
1533             context.fill();
1534         }
1535 }
```

```
1536     context.stroke();
1537     context.restore();
1538
1539     super.draw(context);
1540 }
1541
1542 getBounds() {
1543     if (this.points.length === 0) {
1544         return { x: this.x, y: this.y, width: 0, height: 0 };
1545     }
1546
1547     let minX = this.points[0].x;
1548     let maxX = this.points[0].x;
1549     let minY = this.points[0].y;
1550     let maxY = this.points[0].y;
1551
1552     // Find min/max coordinates
1553     for (let i = 1; i < this.points.length; i++) {
1554         const point = this.points[i];
1555         minX = Math.min(minX, point.x);
1556         maxX = Math.max(maxX, point.x);
1557         minY = Math.min(minY, point.y);
1558         maxY = Math.max(maxY, point.y);
1559     }
1560
1561     return {
1562         x: minX,
1563         y: minY,
1564         width: maxX - minX,
1565         height: maxY - minY
1566     };
1567 }
1568
1569 containsPoint(x, y) {
1570     if (this.points.length < 2) return false;
1571
1572     // Check if point is near any line segment
1573     for (let i = 0; i < this.points.length - 1; i++) {
1574         const p1 = this.points[i];
1575         const p2 = this.points[i + 1];
1576
1577         const lineLength = Math.sqrt(
1578             Math.pow(p2.x - p1.x, 2) + Math.pow(p2.y - p1.y, 2)
1579         );
1580
1581         // If line is too short, check distance to point
1582         if (lineLength < 1) {
1583             const dx = x - p1.x;
1584             const dy = y - p1.y;
1585             if (Math.sqrt(dx * dx + dy * dy) <= 5) {
```

```

1586         return true;
1587     }
1588     continue;
1589 }
1590
1591     // Calculate distance from point to line segment
1592     const t = ((x - p1.x) * (p2.x - p1.x) + (y - p1.y) * (p2.y - p1.y)) /
1593     (lineLength * lineLength);
1594
1595     if (t < 0) {
1596         // Point is beyond start point
1597         const dx = x - p1.x;
1598         const dy = y - p1.y;
1599         if (Math.sqrt(dx * dx + dy * dy) <= 5) {
1600             return true;
1601         }
1602     } else if (t > 1) {
1603         // Point is beyond end point
1604         const dx = x - p2.x;
1605         const dy = y - p2.y;
1606         if (Math.sqrt(dx * dx + dy * dy) <= 5) {
1607             return true;
1608         }
1609     } else {
1610         // Calculate perpendicular distance
1611         const px = p1.x + t * (p2.x - p1.x);
1612         const py = p1.y + t * (p2.y - p1.y);
1613         const dx = x - px;
1614         const dy = y - py;
1615         if (Math.sqrt(dx * dx + dy * dy) <= 5) {
1616             return true;
1617         }
1618     }
1619
1620     // If closed and filled, also check if point is inside
1621     if (this.closed && this.fill) {
1622         // Use point-in-polygon algorithm
1623         let inside = false;
1624         for (let i = 0, j = this.points.length - 1; i < this.points.length; j =
1625             i++) {
1626             const xi = this.points[i].x;
1627             const yi = this.points[i].y;
1628             const xj = this.points[j].x;
1629             const yj = this.points[j].y;
1630
1631             const intersect = ((yi > y) !== (yj > y)) &&
1632                 (x < (xj - xi) * (y - yi) / (yj - yi) + xi);
1633
1634             if (intersect) inside = !inside;

```

```
1634         }
1635         return inside;
1636     }
1637
1638     return false;
1639 }
1640
1641     move(dx, dy) {
1642         super.move(dx, dy);
1643
1644         // Move all points
1645         this.points.forEach(point => {
1646             point.x += dx;
1647             point.y += dy;
1648         });
1649     }
1650
1651     toJSON() {
1652         const json = super.toJSON();
1653         return {
1654             ...json,
1655             points: this.points,
1656             smoothing: this.smoothing,
1657             closed: this.closed,
1658             fill: this.fill,
1659             fillColor: this.fillColor
1660         };
1661     }
1662
1663     static fromJSON(data) {
1664         return new PathObject(data);
1665     }
1666 }
1667
1668 // Frame-Spanning Line (connects specific cells in the grid)
1669 class FrameSpanningLineObject extends DrawingObject {
1670     constructor(props = {}) {
1671         super(props);
1672         this.startFrame = props.startFrame || 1;
1673         this.startColumn = props.startColumn || 1;
1674         this.endFrame = props.endFrame || 1;
1675         this.endColumn = props.endColumn || 1;
1676         this.type = 'frameSpanningLine';
1677         this.dashPattern = props.dashPattern || [];
1678         this.arrowStart = props.arrowStart || false;
1679         this.arrowEnd = props.arrowEnd || false;
1680         this.arrowSize = props.arrowSize || 10;
1681     }
1682
1683     draw(context) {
```

```
1684 // Calculate actual coordinates from frame and column
1685 const layerSystem = window.xsheetDrawing.layerSystem;
1686
1687     const startPos = layerSystem.gridToCanvas(this.startFrame,
1688 this.startColumn);
1689     const endPos = layerSystem.gridToCanvas(this.endFrame, this.endColumn);
1690
1691     if (!startPos || !endPos) return; // Skip if cells not found
1692
1693     context.save();
1694     context.beginPath();
1695     context.strokeStyle = this.color;
1696     context.lineWidth = this.lineWidth;
1697
1698     if (this.dashPattern.length > 0) {
1699         context.setLineDash(this.dashPattern);
1700     }
1701
1702     context.moveTo(startPos.x, startPos.y);
1703     context.lineTo(endPos.x, endPos.y);
1704     context.stroke();
1705
1706     // Draw arrows if needed
1707     if (this.arrowStart) {
1708         this.drawArrow(context, endPos.x, endPos.y, startPos.x, startPos.y);
1709     }
1710
1711     if (this.arrowEnd) {
1712         this.drawArrow(context, startPos.x, startPos.y, endPos.x, endPos.y);
1713     }
1714
1715     context.restore();
1716
1717     // Store computed coordinates for selection	hit testing
1718     this.computedStart = startPos;
1719     this.computedEnd = endPos;
1720
1721     super.draw(context);
1722 }
1723
1724 drawArrow(context, fromX, fromY, toX, toY) {
1725     const angle = Math.atan2(toY - fromY, toX - fromX);
1726
1727     context.save();
1728     context.fillStyle = this.color;
1729     context.beginPath();
1730     context.moveTo(toX, toY);
1731     context.lineTo(
1732         toX - this.arrowSize * Math.cos(angle - Math.PI/6),
1733         toY - this.arrowSize * Math.sin(angle - Math.PI/6)
```

```
1733     );
1734     context.lineTo(
1735         toX - this.arrowSize * Math.cos(angle + Math.PI/6),
1736         toY - this.arrowSize * Math.sin(angle + Math.PI/6)
1737     );
1738     context.closePath();
1739     context.fill();
1740     context.restore();
1741 }
1742
1743     getBounds() {
1744         if (!this.computedStart || !this.computedEnd) return { x: 0, y: 0, width: 0,
height: 0 };
1745
1746         const minX = Math.min(this.computedStart.x, this.computedEnd.x);
1747         const minY = Math.min(this.computedStart.y, this.computedEnd.y);
1748         const width = Math.abs(this.computedEnd.x - this.computedStart.x);
1749         const height = Math.abs(this.computedEnd.y - this.computedStart.y);
1750
1751         return { x: minX, y: minY, width, height };
1752     }
1753
1754     containsPoint(x, y) {
1755         if (!this.computedStart || !this.computedEnd) return false;
1756
1757         // Same algorithm as LineObject
1758         const lineLength = Math.sqrt(
1759             Math.pow(this.computedEnd.x - this.computedStart.x, 2) +
1760             Math.pow(this.computedEnd.y - this.computedStart.y, 2)
1761         );
1762
1763         if (lineLength < 1) {
1764             const dx = x - this.computedStart.x;
1765             const dy = y - this.computedStart.y;
1766             return Math.sqrt(dx * dx + dy * dy) <= 5;
1767         }
1768
1769         const t = ((x - this.computedStart.x) * (this.computedEnd.x -
this.computedStart.x) +
1770                 (y - this.computedStart.y) * (this.computedEnd.y -
this.computedStart.y)) /
1771                 (lineLength * lineLength);
1772
1773         if (t < 0) {
1774             const dx = x - this.computedStart.x;
1775             const dy = y - this.computedStart.y;
1776             return Math.sqrt(dx * dx + dy * dy) <= 5;
1777         }
1778
1779         if (t > 1) {
1780             const dx = x - this.computedEnd.x;
```

```
1781     const dy = y - this.computedEnd.y;
1782     return Math.sqrt(dx * dx + dy * dy) <= 5;
1783 }
1784
1785     const px = this.computedStart.x + t * (this.computedEnd.x -
1786 this.computedStart.x);
1787     const py = this.computedStart.y + t * (this.computedEnd.y -
1788 this.computedStart.y);
1789     const dx = x - px;
1790     const dy = y - py;
1791     return Math.sqrt(dx * dx + dy * dy) <= 5;
1792 }
1793
1794 // This object type doesn't use the regular move method
1795 // Instead, it updates the frame/column values
1796
1797 toJSON() {
1798     const json = super.toJSON();
1799     return {
1800         ...json,
1801         startFrame: this.startFrame,
1802         startColumn: this.startColumn,
1803         endFrame: this.endFrame,
1804         endColumn: this.endColumn,
1805         dashPattern: this.dashPattern,
1806         arrowStart: this.arrowStart,
1807         arrowEnd: this.arrowEnd,
1808         arrowSize: this.arrowSize
1809     };
1810 }
1811
1812     static fromJSON(data) {
1813         return new FrameSpanningLineObject(data);
1814     }
1815
1816 // Register all object types in a factory
1817 const DrawingObjectFactory = {
1818     types: {
1819         'drawingObject': DrawingObject,
1820         'line': LineObject,
1821         'arrow': ArrowObject,
1822         'rectangle': RectangleObject,
1823         'ellipse': EllipseObject,
1824         'text': TextObject,
1825         'image': ImageObject,
1826         'symbol': SymbolObject,
1827         'path': PathObject,
1828         'frameSpanningLine': FrameSpanningLineObject
1829     },
1830 }
```

```
1829
1830     createFromJSON(data) {
1831         const Type = this.types[data.type];
1832         if (Type) {
1833             return Type.fromJSON(data);
1834         }
1835         return null;
1836     }
1837 };
1838
1839 /**
1840 * DRAWING TOOL SYSTEM
1841 * Manages the drawing tools and interfaces with the layer system
1842 */
1843 class DrawingToolSystem {
1844     constructor(layerSystem) {
1845         this.layerSystem = layerSystem;
1846         this.activeTool = null;
1847         this.toolSettings = {
1848             color: '#ff0000',
1849             lineWidth: 2,
1850             fill: false,
1851             fillColor: '#ff8080',
1852             fontSize: 16,
1853             fontFamily: 'Arial, sans-serif',
1854             textAlign: 'left',
1855             symbolType: 'default',
1856             symbolScale: 1.0
1857         };
1858
1859         this.availableTools = {
1860             'select': new SelectTool(this),
1861             'pen': new PenTool(this),
1862             'line': new LineTool(this),
1863             'arrow': new ArrowTool(this),
1864             'rectangle': new RectangleTool(this),
1865             'ellipse': new EllipseTool(this),
1866             'text': new TextTool(this),
1867             'image': new ImageTool(this),
1868             'symbol': new SymbolTool(this),
1869             'frameLine': new FrameSpanningLineTool(this),
1870             'eraser': new EraserTool(this)
1871         };
1872
1873         // Default to select tool
1874         this.setActiveTool('select');
1875
1876         // Set up toolbar UI
1877         this.createToolbar();
1878     }
}
```

```
1879
1880     setActiveTool(toolName) {
1881         if (this.activeTool) {
1882             this.activeTool.deactivate();
1883         }
1884
1885         if (this.availableTools[toolName]) {
1886             this.activeTool = this.availableTools[toolName];
1887             this.activeTool.activate();
1888             return true;
1889         }
1890
1891         return false;
1892     }
1893
1894     createToolbar() {
1895         // Create toolbar container
1896         const toolbar = document.createElement('div');
1897         toolbar.className = 'drawing-toolbar';
1898         toolbar.style.display = 'flex';
1899         toolbar.style.flexWrap = 'wrap';
1900         toolbar.style.gap = '5px';
1901         toolbar.style.padding = '10px';
1902         toolbar.style.backgroundColor = '#f5f5f5';
1903         toolbar.style.marginBottom = '10px';
1904         toolbar.style.borderRadius = '5px';
1905
1906         // Add tool buttons
1907         this.addButton(toolbar, 'select', 'Select', '👉');
1908         this.addButton(toolbar, 'pen', 'Freehand Drawing', '✍');
1909         this.addButton(toolbar, 'line', 'Line', '‐');
1910         this.addButton(toolbar, 'arrow', 'Arrow', '→');
1911         this.addButton(toolbar, 'rectangle', 'Rectangle', '□');
1912         this.addButton(toolbar, 'ellipse', 'Circle/Ellipse', '○');
1913         this.addButton(toolbar, 'text', 'Text', 'T');
1914         this.addButton(toolbar, 'image', 'Insert Image', '🖼');
1915         this.addButton(toolbar, 'symbol', 'Animation Symbol', '⭐');
1916         this.addButton(toolbar, 'frameLine', 'Multi-Frame Line', '💠');
1917
1918         // Add separator
1919         toolbar.appendChild(document.createElement('div')).style.borderLeft = '1px
solid #ccc';
1920         toolbar.lastChild.style.height = '30px';
1921
1922         // Add color picker
1923         const colorContainer = document.createElement('div');
1924         colorContainer.style.display = 'flex';
1925         colorContainer.style.alignItems = 'center';
1926         colorContainer.style.gap = '5px';
1927 }
```

```
1928 const colorLabel = document.createElement('label');
1929 colorLabel.textContent = 'Color:';
1930 colorLabel.htmlFor = 'drawing-color';
1931
1932 const colorPicker = document.createElement('input');
1933 colorPicker.type = 'color';
1934 colorPicker.id = 'drawing-color';
1935 colorPicker.value = this.toolSettings.color;
1936 colorPicker.addEventListener('input', (e) => {
1937     this.toolSettings.color = e.target.value;
1938 });
1939
1940 colorContainer.appendChild(colorLabel);
1941 colorContainer.appendChild(colorPicker);
1942 toolbar.appendChild(colorContainer);
1943
1944 // Add line width selector
1945 const lineWidthContainer = document.createElement('div');
1946 lineWidthContainer.style.display = 'flex';
1947 lineWidthContainer.style.alignItems = 'center';
1948 lineWidthContainer.style.gap = '5px';
1949
1950 const lineWidthLabel = document.createElement('label');
1951 lineWidthLabel.textContent = 'Width:';
1952 lineWidthLabel.htmlFor = 'drawing-line-width';
1953
1954 const lineWidthSelect = document.createElement('select');
1955 lineWidthSelect.id = 'drawing-line-width';
1956
1957 const widths = [1, 2, 3, 5, 8, 12];
1958 widths.forEach(width => {
1959     const option = document.createElement('option');
1960     option.value = width;
1961     option.textContent = width + 'px';
1962     if (width === this.toolSettings.lineWidth) {
1963         option.selected = true;
1964     }
1965     lineWidthSelect.appendChild(option);
1966 });
1967
1968 lineWidthSelect.addEventListener('change', (e) => {
1969     this.toolSettings.lineWidth = parseInt(e.target.value);
1970 });
1971
1972 lineWidthContainer.appendChild(lineWidthLabel);
1973 lineWidthContainer.appendChild(lineWidthSelect);
1974 toolbar.appendChild(lineWidthContainer);
1975
1976 // Add fill option
1977 const fillContainer = document.createElement('div');
```



```
2027         layerSelector.appendChild(option);
2028     });
2029
2030     layerSelector.addEventListener('change', (e) => {
2031         this.layerSystem.setActiveLayer(parseInt(e.target.value));
2032     });
2033
2034     const layerLabel = document.createElement('label');
2035     layerLabel.textContent = 'Layer:';
2036     layerLabel.htmlFor = 'drawing-layer-selector';
2037     layerLabel.style.marginRight = '5px';
2038
2039     toolbar.appendChild(layerLabel);
2040     toolbar.appendChild(layerSelector);
2041
2042     // Add separator
2043     toolbar.appendChild(document.createElement('div')).style.borderLeft = '1px
solid #ccc';
2044     toolbar.lastChild.style.height = '30px';
2045
2046     // Add eraser tool
2047     this.addToolButton(toolbar, 'eraser', 'Eraser', 'eraser');
2048
2049     // Add clear button
2050     const clearButton = document.createElement('button');
2051     clearButton.textContent = 'Clear All Drawings';
2052     clearButton.style.backgroundColor = '#ff5555';
2053     clearButton.style.color = 'white';
2054     clearButton.style.border = 'none';
2055     clearButton.style.borderRadius = '4px';
2056     clearButton.style.padding = '5px 10px';
2057     clearButton.style.cursor = 'pointer';
2058
2059     clearButton.addEventListener('click', () => {
2060         if (confirm('Are you sure you want to clear all drawings?')) {
2061             this.layerSystem.clearAllLayers();
2062         }
2063     });
2064
2065     toolbar.appendChild(clearButton);
2066
2067     // Find controls div and add toolbar before it
2068     const controls = document.querySelector('.controls');
2069     if (controls) {
2070         controls.parentNode.insertBefore(toolbar, controls.nextSibling);
2071     } else {
2072         document.body.insertBefore(toolbar, document.body.firstChild);
2073     }
2074 }
2075 }
```

```
2076     addToolBar(toolbar, toolName, tooltip, icon) {
2077         const button = document.createElement('button');
2078         button.textContent = icon;
2079         button.title = tooltip;
2080         button.style.width = '36px';
2081         button.style.height = '36px';
2082         button.style.fontSize = '16px';
2083         button.style.margin = '0';
2084         button.style.padding = '5px';
2085         button.style.borderRadius = '4px';
2086         button.style.border = '1px solid #ccc';
2087         button.style.backgroundColor = 'white';
2088         button.style.cursor = 'pointer';
2089
2090         button.addEventListener('click', () => {
2091             this.setActiveTool(toolName);
2092
2093             // Update active button styling
2094             document.querySelectorAll('.drawing-toolbar button').forEach(btn => {
2095                 btn.style.backgroundColor = 'white';
2096                 btn.style.color = 'black';
2097             });
2098
2099             button.style.backgroundColor = '#4CAF50';
2100             button.style.color = 'white';
2101         });
2102
2103         // Set active state for default tool
2104         if (toolName === 'select') {
2105             button.style.backgroundColor = '#4CAF50';
2106             button.style.color = 'white';
2107         }
2108
2109         toolbar.appendChild(button);
2110     }
2111 }
2112
2113 /**
2114 * DRAWING TOOLS
2115 * Individual tool implementations
2116 */
2117
2118 // Base tool class
2119 class DrawingTool {
2120     constructor(toolSystem) {
2121         this.toolSystem = toolSystem;
2122         this.layerSystem = toolSystem.layerSystem;
2123         this.active = false;
2124         this.settings = toolSystem.toolSettings;
2125     }
}
```

```
2126
2127     activate() {
2128         this.active = true;
2129         this.layerSystem.enableDrawing();
2130         this.attachEvents();
2131     }
2132
2133     deactivate() {
2134         this.active = false;
2135         // turn pointer-events back off so clicks go to the table cells
2136         this.toolSystem.layerSystem.disableDrawing();
2137         this.detachEvents();
2138     }
2139
2140     attachEvents() {
2141         // Override in subclasses
2142     }
2143
2144     detachEvents() {
2145         // Override in subclasses
2146     }
2147 }
2148
2149 // Select tool for manipulating objects
2150 class SelectTool extends DrawingTool {
2151
2152     constructor(toolSystem) {
2153         super(toolSystem);
2154         this.selectedObject = null;
2155         this.selectedLayer = null;
2156         this.dragging = false;
2157         this.dragStart = { x: 0, y: 0 };
2158         this.objectStart = { x: 0, y: 0 };
2159     }
2160     activate() {
2161         this.active = true;
2162         // let pointer-events go to the table cells instead of the canvases
2163         this.layerSystem.disableDrawing();
2164         this.attachEvents();
2165     }
2166
2167     deactivate() {
2168         this.active = false;
2169         this.layerSystem.disableDrawing();
2170         this.detachEvents();
2171     }
2172     attachEvents() {
2173         const canvas = this.layerSystem.getActiveLayer().canvas;
2174
2175         canvas.addEventListener('mousedown', this.handleMouseDown);
```

```
2176     canvas.addEventListener('pointerdown', this.handleMouseDown);
2177     document.addEventListener('mousemove', this.handleMouseMove);
2178     document.addEventListener('pointermove', this.handleMouseMove);
2179     document.addEventListener('mouseup', this.handleMouseUp);
2180     document.addEventListener('pointerup', this.handleMouseUp);
2181     document.addEventListener('keydown', this.handleKeyDown);
2182 }
2183
2184 detachEvents() {
2185     const canvas = this.layerSystem.getActiveLayer().canvas;
2186
2187     canvas.removeEventListener('mousedown', this.handleMouseDown);
2188     canvas.removeEventListener('pointerdown', this.handleMouseDown);
2189     document.removeEventListener('mousemove', this.handleMouseMove);
2190     document.removeEventListener('pointermove', this.handleMouseMove);
2191     document.removeEventListener('mouseup', this.handleMouseUp);
2192     document.removeEventListener('pointerup', this.handleMouseUp);
2193     document.removeEventListener('keydown', this.handleKeyDown);
2194
2195     // Clear selection
2196     this.clearSelection();
2197 }
2198
2199 handleMouseDown = (e) => {
2200     // Convert to canvas coordinates
2201     const coords = this.layerSystem.screenToCanvas(e.clientX, e.clientY);
2202
2203     // Check if clicked on an object
2204     const hit = this.layerSystem.findObjectAt(coords.x, coords.y);
2205
2206     if (hit) {
2207         // Select the object
2208         this.selectObject(hit.object, hit.layerIndex);
2209
2210         // Start drag
2211         this.dragging = true;
2212         this.dragStart = { x: coords.x, y: coords.y };
2213         this.objectStart = { x: hit.object.x, y: hit.object.y };
2214     } else {
2215         // Clear selection if clicked empty space
2216         this.clearSelection();
2217     }
2218 }
2219
2220 handleMouseMove = (e) => {
2221     if (!this.dragging || !this.selectedObject) return;
2222
2223     const coords = this.layerSystem.screenToCanvas(e.clientX, e.clientY);
2224
2225     // Calculate move distance
```

```
2226     const dx = coords.x - this.dragStart.x;
2227     const dy = coords.y - this.dragStart.y;
2228
2229     // Move the selected object
2230     this.selectedObject.x = this.objectStart.x + dx;
2231     this.selectedObject.y = this.objectStart.y + dy;
2232
2233     // For objects with special move handling
2234     this.selectedObject.move(dx, dy);
2235     this.selectedObject.x = this.objectStart.x; // Reset x as move() already
2236     handled it
2237
2238     this.selectedObject.y = this.objectStart.y; // Reset y as move() already
2239     handled it
2240
2241
2242     handleMouseUp = () => {
2243         this.dragging = false;
2244     }
2245
2246
2247     handleKeyDown = (e) => {
2248         if (!this.selectedObject) return;
2249
2250         // Delete key
2251         if (e.key === 'Delete' || e.key === 'Backspace') {
2252             this.layerSystem.removeObject(this.selectedObject, this.selectedLayer);
2253             this.clearSelection();
2254         }
2255
2256         // Arrow keys for fine movement
2257         const moveDistance = e.shiftKey ? 10 : 1;
2258
2259         if (e.key === 'ArrowLeft') {
2260             this.selectedObject.move(-moveDistance, 0);
2261             this.layerSystem.redrawAll();
2262         } else if (e.key === 'ArrowRight') {
2263             this.selectedObject.move(moveDistance, 0);
2264             this.layerSystem.redrawAll();
2265         } else if (e.key === 'ArrowUp') {
2266             this.selectedObject.move(0, -moveDistance);
2267             this.layerSystem.redrawAll();
2268         } else if (e.key === 'ArrowDown') {
2269             this.selectedObject.move(0, moveDistance);
2270             this.layerSystem.redrawAll();
2271         }
2272     }
2273
2274     selectObject(object, layerIndex) {
```

```
2274         // Clear previous selection
2275         this.clearSelection();
2276
2277         // Set new selection
2278         this.selectedObject = object;
2279         this.selectedLayer = layerIndex;
2280         object.selected = true;
2281
2282         // Redraw with selection visual
2283         this.layerSystem.redrawAll();
2284     }
2285
2286     clearSelection() {
2287         if (this.selectedObject) {
2288             this.selectedObject.selected = false;
2289             this.selectedObject = null;
2290             this.selectedLayer = null;
2291             this.layerSystem.redrawAll();
2292         }
2293     }
2294 }
2295
2296 // Pen tool for free drawing
2297 class PenTool extends DrawingTool {
2298     constructor(toolSystem) {
2299         super(toolSystem);
2300         this.currentPath = null;
2301         this.drawing = false;
2302     }
2303
2304     attachEvents() {
2305         const canvas = this.layerSystem.getActiveLayer().canvas;
2306
2307         canvas.addEventListener('mousedown', this.handleMouseDown);
2308         canvas.addEventListener('pointerdown', this.handleMouseDown);
2309         document.addEventListener('mousemove', this.handleMouseMove);
2310         document.addEventListener('pointermove', this.handleMouseMove);
2311         document.addEventListener('mouseup', this.handleMouseUp);
2312         document.addEventListener('pointerup', this.handleMouseUp);
2313     }
2314
2315     detachEvents() {
2316         const canvas = this.layerSystem.getActiveLayer().canvas;
2317
2318         canvas.removeEventListener('mousedown', this.handleMouseDown);
2319         canvas.removeEventListener('pointerdown', this.handleMouseDown);
2320         document.removeEventListener('mousemove', this.handleMouseMove);
2321         document.removeEventListener('pointermove', this.handleMouseMove);
2322         document.removeEventListener('mouseup', this.handleMouseUp);
2323         document.removeEventListener('pointerup', this.handleMouseUp);
```

```
2324
2325          // Finish any in-progress drawing
2326          this.finishDrawing();
2327      }
2328
2329      handleMouseDown = (e) => {
2330          e.preventDefault();
2331          if (e.pointerId != null) {
2332              e.target.setPointerCapture(e.pointerId);
2333          }
2334
2335          const coords = this.layerSystem.screenToCanvas(e.clientX, e.clientY);
2336
2337          // Start a new path
2338          this.currentPath = new PathObject({
2339              color: this.settings.color,
2340              lineWidth: this.settings.lineWidth,
2341              x: coords.x,
2342              y: coords.y
2343          });
2344
2345          this.currentPath.addPoint(coords.x, coords.y);
2346          this.drawing = true;
2347
2348          // Add to layer
2349          this.layerSystem.addObject(this.currentPath);
2350      }
2351
2352      handleMouseMove = (e) => {
2353          if (!this.drawing || !this.currentPath) return;
2354
2355          const coords = this.layerSystem.screenToCanvas(e.clientX, e.clientY);
2356
2357          // Add point to path
2358          this.currentPath.addPoint(coords.x, coords.y);
2359
2360          // Redraw
2361          this.layerSystem.redrawLayer(this.layerSystem.activeLayerIndex);
2362      }
2363
2364      handleMouseUp = (e) => {
2365          if (e.pointerId != null) {
2366              e.target.releasePointerCapture(e.pointerId);
2367          }
2368          this.finishDrawing();
2369      }
2370
2371      finishDrawing() {
2372          if (this.drawing && this.currentPath) {
2373              // Finish the path
```

```
2374         this.drawing = false;
2375         this.currentPath = null;
2376     }
2377 }
2378 }
2379
2380 // Line tool
2381 class LineTool extends DrawingTool {
2382     constructor(toolSystem) {
2383         super(toolSystem);
2384         this.startPoint = null;
2385         this.currentLine = null;
2386         this.drawing = false;
2387     }
2388
2389     attachEvents() {
2390         const canvas = this.layerSystem.getActiveLayer().canvas;
2391
2392         canvas.addEventListener('mousedown', this.handleMouseDown);
2393         document.addEventListener('mousemove', this.handleMouseMove);
2394         document.addEventListener('mouseup', this.handleMouseUp);
2395     }
2396
2397     detachEvents() {
2398         const canvas = this.layerSystem.getActiveLayer().canvas;
2399
2400         canvas.removeEventListener('mousedown', this.handleMouseDown);
2401         document.removeEventListener('mousemove', this.handleMouseMove);
2402         document.removeEventListener('mouseup', this.handleMouseUp);
2403
2404         // Finish any in-progress drawing
2405         this.finishDrawing();
2406     }
2407
2408     handleMouseDown = (e) => {
2409         const coords = this.layerSystem.screenToCanvas(e.clientX, e.clientY);
2410
2411         this.startPoint = coords;
2412
2413         // Create temporary line
2414         this.currentLine = new LineObject({
2415             x: coords.x,
2416             y: coords.y,
2417             x2: coords.x,
2418             y2: coords.y,
2419             color: this.settings.color,
2420             lineWidth: this.settings.lineWidth
2421         });
2422
2423         this.drawing = true;
```

```
2424
2425          // Add to layer
2426          this.layerSystem.addObject(this.currentLine);
2427      }
2428
2429      handleMouseMove = (e) => {
2430          if (!this.drawing || !this.currentLine) return;
2431
2432          const coords = this.layerSystem.screenToCanvas(e.clientX, e.clientY);
2433
2434          // Update end point
2435          this.currentLine.x2 = coords.x;
2436          this.currentLine.y2 = coords.y;
2437
2438          // Redraw
2439          this.layerSystem.redrawLayer(this.layerSystem.activeLayerIndex);
2440      }
2441
2442      handleMouseUp = () => {
2443          this.finishDrawing();
2444      }
2445
2446      finishDrawing() {
2447          if (this.drawing && this.currentLine) {
2448              // Finish the line
2449              this.drawing = false;
2450              this.startPoint = null;
2451              this.currentLine = null;
2452          }
2453      }
2454  }
2455
2456  // Arrow tool (extends Line tool)
2457  class ArrowTool extends LineTool {
2458      handleMouseDown = (e) => {
2459          const coords = this.layerSystem.screenToCanvas(e.clientX, e.clientY);
2460
2461          this.startPoint = coords;
2462
2463          // Create temporary arrow
2464          this.currentLine = new ArrowObject({
2465              x: coords.x,
2466              y: coords.y,
2467              x2: coords.x,
2468              y2: coords.y,
2469              color: this.settings.color,
2470              lineWidth: this.settings.lineWidth
2471          });
2472
2473          this.drawing = true;
```

```
2474
2475          // Add to layer
2476          this.layerSystem.addObject(this.currentLine);
2477      }
2478  }
2479
2480 // Rectangle tool
2481 class RectangleTool extends DrawingTool {
2482     constructor(toolSystem) {
2483         super(toolSystem);
2484         this.startPoint = null;
2485         this.currentRect = null;
2486         this.drawing = false;
2487     }
2488
2489     attachEvents() {
2490         const canvas = this.layerSystem.getActiveLayer().canvas;
2491
2492         canvas.addEventListener('mousedown', this.handleMouseDown);
2493         document.addEventListener('mousemove', this.handleMouseMove);
2494         document.addEventListener('mouseup', this.handleMouseUp);
2495     }
2496
2497     detachEvents() {
2498         const canvas = this.layerSystem.getActiveLayer().canvas;
2499
2500         canvas.removeEventListener('mousedown', this.handleMouseDown);
2501         document.removeEventListener('mousemove', this.handleMouseMove);
2502         document.removeEventListener('mouseup', this.handleMouseUp);
2503
2504         // Finish any in-progress drawing
2505         this.finishDrawing();
2506     }
2507
2508     handleMouseDown = (e) => {
2509         const coords = this.layerSystem.screenToCanvas(e.clientX, e.clientY);
2510
2511         this.startPoint = coords;
2512
2513         // Create temporary rectangle
2514         this.currentRect = new RectangleObject({
2515             x: coords.x,
2516             y: coords.y,
2517             width: 0,
2518             height: 0,
2519             color: this.settings.color,
2520             lineWidth: this.settings.lineWidth,
2521             fill: this.settings.fill,
2522             fillColor: this.settings.fillColor
2523         });
2524 }
```

```
2524         this.drawing = true;
2525
2526
2527         // Add to layer
2528         this.layerSystem.addObject(this.currentRect);
2529     }
2530
2531
2532     handleMouseMove = (e) => {
2533         if (!this.drawing || !this.currentRect) return;
2534
2535         const coords = this.layerSystem.screenToCanvas(e.clientX, e.clientY);
2536
2537         // Update dimensions
2538         const width = coords.x - this.startPoint.x;
2539         const height = coords.y - this.startPoint.y;
2540
2541         if (width < 0) {
2542             this.currentRect.x = coords.x;
2543             this.currentRect.width = Math.abs(width);
2544         } else {
2545             this.currentRect.x = this.startPoint.x;
2546             this.currentRect.width = width;
2547         }
2548
2549         if (height < 0) {
2550             this.currentRect.y = coords.y;
2551             this.currentRect.height = Math.abs(height);
2552         } else {
2553             this.currentRect.y = this.startPoint.y;
2554             this.currentRect.height = height;
2555         }
2556
2557         // Redraw
2558         this.layerSystem.redrawLayer(this.layerSystem.activeLayerIndex);
2559     }
2560
2561     handleMouseUp = () => {
2562         this.finishDrawing();
2563     }
2564
2565     finishDrawing() {
2566         if (this.drawing && this.currentRect) {
2567             // Finish the rectangle
2568             this.drawing = false;
2569             this.startPoint = null;
2570             this.currentRect = null;
2571         }
2572     }
2573 }
```

```
2574 // Ellipse tool
2575 class EllipseTool extends DrawingTool {
2576     constructor(toolSystem) {
2577         super(toolSystem);
2578         this.center = null;
2579         this.currentEllipse = null;
2580         this.drawing = false;
2581     }
2582
2583     attachEvents() {
2584         const canvas = this.layerSystem.getActiveLayer().canvas;
2585
2586         canvas.addEventListener('mousedown', this.handleMouseDown);
2587         document.addEventListener('mousemove', this.handleMouseMove);
2588         document.addEventListener('mouseup', this.handleMouseUp);
2589     }
2590
2591     detachEvents() {
2592         const canvas = this.layerSystem.getActiveLayer().canvas;
2593
2594         canvas.removeEventListener('mousedown', this.handleMouseDown);
2595         document.removeEventListener('mousemove', this.handleMouseMove);
2596         document.removeEventListener('mouseup', this.handleMouseUp);
2597
2598         // Finish any in-progress drawing
2599         this.finishDrawing();
2600     }
2601
2602     handleMouseDown = (e) => {
2603         const coords = this.layerSystem.screenToCanvas(e.clientX, e.clientY);
2604
2605         this.center = coords;
2606
2607         // Create temporary ellipse
2608         this.currentEllipse = new EllipseObject({
2609             x: coords.x,
2610             y: coords.y,
2611             radiusX: 0,
2612             radiusY: 0,
2613             color: this.settings.color,
2614             lineWidth: this.settings.lineWidth,
2615             fill: this.settings.fill,
2616             fillColor: this.settings.fillColor
2617         });
2618
2619         this.drawing = true;
2620
2621         // Add to layer
2622         this.layerSystem.addObject(this.currentEllipse);
2623     }
```

```
2624
2625     handleMouseMove = (e) => {
2626         if (!this.drawing || !this.currentEllipse) return;
2627
2628         const coords = this.layerSystem.screenToCanvas(e.clientX, e.clientY);
2629
2630         // Update radii
2631         this.currentEllipse.radiusX = Math.abs(coords.x - this.center.x);
2632         this.currentEllipse.radiusY = Math.abs(coords.y - this.center.y);
2633
2634         // Redraw
2635         this.layerSystem.redrawLayer(this.layerSystem.activeLayerIndex);
2636     }
2637
2638     handleMouseUp = () => {
2639         this.finishDrawing();
2640     }
2641
2642     finishDrawing() {
2643         if (this.drawing && this.currentEllipse) {
2644             // Finish the ellipse
2645             this.drawing = false;
2646             this.center = null;
2647             this.currentEllipse = null;
2648         }
2649     }
2650 }
2651
2652 // Text tool
2653 class TextTool extends DrawingTool {
2654     constructor(toolSystem) {
2655         super(toolSystem);
2656         this.textInput = null;
2657     }
2658
2659     attachEvents() {
2660         const canvas = this.layerSystem.getActiveLayer().canvas;
2661
2662         canvas.addEventListener('click', this.handleClick);
2663     }
2664
2665     detachEvents() {
2666         const canvas = this.layerSystem.getActiveLayer().canvas;
2667
2668         canvas.removeEventListener('click', this.handleClick);
2669
2670         // Remove any active text input
2671         this.removeTextInput();
2672     }
2673 }
```

```
2674 handleClick = (e) => {
2675   const coords = this.layerSystem.screenToCanvas(e.clientX, e.clientY);
2676
2677   // Show text input at click position
2678   this.showTextInput(coords.x, coords.y);
2679 }
2680
2681 showTextInput(x, y) {
2682   // Remove any existing text input
2683   this.removeTextInput();
2684
2685   // Create text input element
2686   this.textInput = document.createElement('div');
2687   this.textInput.style.position = 'absolute';
2688   this.textInput.style.zIndex = '100';
2689
2690   // Position relative to canvas
2691   const canvasRect = this.layerSystem.container.getBoundingClientRect();
2692   this.textInput.style.left = (canvasRect.left + x) + 'px';
2693   this.textInput.style.top = (canvasRect.top + y) + 'px';
2694
2695   // Style the input
2696   this.textInput.style.backgroundColor = 'white';
2697   this.textInput.style.border = '1px solid #ccc';
2698   this.textInput.style.padding = '5px';
2699   this.textInput.style.borderRadius = '3px';
2700   this.textInput.style.boxShadow = '0 2px 5px rgba(0,0,0,0.1)';
2701
2702   // Create the actual input
2703   const input = document.createElement('input');
2704   input.type = 'text';
2705   input.placeholder = 'Enter text...';
2706   input.style.width = '200px';
2707   input.style.padding = '5px';
2708   input.style.border = '1px solid #ddd';
2709   input.style.borderRadius = '3px';
2710
2711   // Create font size selector
2712   const sizeSelect = document.createElement('select');
2713   [8, 10, 12, 14, 16, 18, 20, 24, 36].forEach(size => {
2714     const option = document.createElement('option');
2715     option.value = size;
2716     option.textContent = size + 'px';
2717     if (size === this.settings.fontSize) {
2718       option.selected = true;
2719     }
2720     sizeSelect.appendChild(option);
2721   });
2722
2723   sizeSelect.addEventListener('change', (e) => {
```

```
2724         this.settings.fontSize = parseInt(e.target.value);
2725     });
2726
2727     // Create alignment options
2728     const alignmentDiv = document.createElement('div');
2729     alignmentDiv.style.display = 'flex';
2730     alignmentDiv.style.marginTop = '5px';
2731
2732     ['left', 'center', 'right'].forEach((align) => {
2733         const button = document.createElement('button');
2734         button.textContent = align[0].toUpperCase();
2735         button.style.flex = '1';
2736         button.style.padding = '2px 5px';
2737         button.style.backgroundColor = this.settings.textAlign === align ?
2738             '#4CAF50' : '#f1f1f1';
2739         button.style.color = this.settings.textAlign === align ? 'white' :
2740             'black';
2741
2742         button.addEventListener('click', () => {
2743             this.settings.textAlign = align;
2744             alignmentDiv.querySelectorAll('button').forEach((btn) => {
2745                 btn.style.backgroundColor = '#f1f1f1';
2746                 btn.style.color = 'black';
2747             });
2748             button.style.backgroundColor = '#4CAF50';
2749             button.style.color = 'white';
2750         });
2751
2752         alignmentDiv.appendChild(button);
2753     });
2754
2755     // Add UI elements to the container
2756     this.textInput.appendChild(input);
2757     this.textInput.appendChild(document.createElement('br'));
2758     this.textInput.appendChild(document.createTextNode('Size: '));
2759     this.textInput.appendChild(sizeSelect);
2760     this.textInput.appendChild(document.createElement('br'));
2761     this.textInput.appendChild(alignmentDiv);
2762
2763     // Add buttons container
2764     const buttons = document.createElement('div');
2765     buttons.style.display = 'flex';
2766     buttons.style.marginTop = '5px';
2767     buttons.style.gap = '5px';
2768
2769     // Add button
2770     const addButton = document.createElement('button');
2771     addButton.textContent = 'Add Text';
```

```
2772     addButton.style.flex = '1';
2773     addButton.style.padding = '5px';
2774     addButton.style.backgroundColor = '#4CAF50';
2775     addButton.style.color = 'white';
2776     addButton.style.border = 'none';
2777     addButton.style.borderRadius = '3px';
2778     addButton.style.cursor = 'pointer';

2779
2780     addButton.addEventListener('click', () => {
2781         this.addText(x, y, input.value);
2782     });
2783
2784     // Cancel button
2785     const cancelButton = document.createElement('button');
2786     cancelButton.textContent = 'Cancel';
2787     cancelButton.style.flex = '1';
2788     cancelButton.style.padding = '5px';
2789     cancelButton.style.backgroundColor = '#f44336';
2790     cancelButton.style.color = 'white';
2791     cancelButton.style.border = 'none';
2792     cancelButton.style.borderRadius = '3px';
2793     cancelButton.style.cursor = 'pointer';

2794
2795     cancelButton.addEventListener('click', () => {
2796         this.removeTextInput();
2797     });
2798
2799     buttons.appendChild(addButton);
2800     buttons.appendChild(cancelButton);
2801     this.textInput.appendChild(buttons);

2802
2803     // Add to document
2804     document.body.appendChild(this.textInput);

2805
2806     // Focus the input
2807     input.focus();

2808
2809     // Handle enter key
2810     input.addEventListener('keydown', (e) => {
2811         if (e.key === 'Enter') {
2812             this.addText(x, y, input.value);
2813         } else if (e.key === 'Escape') {
2814             this.removeTextInput();
2815         }
2816     });
2817 }
2818
2819 removeTextInput() {
2820     if (this.textInput && this.textInput.parentNode) {
2821         this.textInput.parentNode.removeChild(this.textInput);
```

```
2822         this.textInput = null;
2823     }
2824 }
2825
2826 addText(x, y, text) {
2827     if (!text.trim()) {
2828         this.removeTextInput();
2829         return;
2830     }
2831
2832     // Create text object
2833     const textObj = new TextObject({
2834         x: x,
2835         y: y,
2836         text: text,
2837         color: this.settings.color,
2838         fontSize: this.settings.fontSize,
2839         fontFamily: this.settings.fontFamily,
2840         align: this.settings.textAlign
2841     });
2842
2843     // Add to layer
2844     this.layerSystem.addObject(textObj);
2845
2846     // Remove text input
2847     this.removeTextInput();
2848 }
2849 }
2850
2851 // Image tool
2852 class ImageTool extends DrawingTool {
2853     constructor(toolSystem) {
2854         super(toolSystem);
2855         this.fileInput = null;
2856     }
2857
2858     attachEvents() {
2859         const canvas = this.layerSystem.getActiveLayer().canvas;
2860
2861         canvas.addEventListener('click', this.handleClick);
2862     }
2863
2864     detachEvents() {
2865         const canvas = this.layerSystem.getActiveLayer().canvas;
2866
2867         canvas.removeEventListener('click', this.handleClick);
2868     }
2869
2870     handleClick = (e) => {
2871         // Show file upload dialog
```

```
2872         this.showFileDialog();
2873     }
2874
2875     showFileDialog() {
2876         // Create hidden file input if it doesn't exist
2877         if (!this.fileInput) {
2878             this.fileInput = document.createElement('input');
2879             this.fileInput.type = 'file';
2880             this.fileInput.accept = 'image/*';
2881             this.fileInput.style.display = 'none';
2882             document.body.appendChild(this.fileInput);
2883
2884             this.fileInput.addEventListener('change', (e) => {
2885                 if (e.target.files && e.target.files[0]) {
2886                     this.handleFileSelect(e.target.files[0]);
2887                 }
2888             });
2889         }
2890
2891         // Trigger file dialog
2892         this.fileInput.click();
2893     }
2894
2895     handleFileSelect(file) {
2896         // Read the file and create a data URL
2897         const reader = new FileReader();
2898
2899         reader.onload = (e) => {
2900             const imageUrl = e.target.result;
2901
2902             // Show image placement UI
2903             this.showImagePlacementUI(imageUrl);
2904         };
2905
2906         reader.readAsDataURL(file);
2907     }
2908
2909     showImagePlacementUI(imageUrl) {
2910         // Create a preview image to get dimensions
2911         const img = new Image();
2912
2913         img.onload = () => {
2914             // Calculate dimensions (max size 300px width/height while maintaining
aspect ratio)
2915             let width = img.width;
2916             let height = img.height;
2917
2918             const maxSize = 300;
2919             if (width > maxSize || height > maxSize) {
2920                 if (width > height) {
```

```
2921     height = (height / width) * maxSize;
2922     width = maxSize;
2923 } else {
2924     width = (width / height) * maxSize;
2925     height = maxSize;
2926 }
2927 }

2928 // Create placement UI
2929 const placementUI = document.createElement('div');
2930 placementUI.style.position = 'fixed';
2931 placementUI.style.top = '50%';
2932 placementUI.style.left = '50%';
2933 placementUI.style.transform = 'translate(-50%, -50%)';
2934 placementUI.style.backgroundColor = 'white';
2935 placementUI.style.padding = '20px';
2936 placementUI.style.borderRadius = '5px';
2937 placementUI.style.boxShadow = '0 0 10px rgba(0,0,0,0.3)';
2938 placementUI.style.zIndex = '1000';

2939
2940 // Add heading
2941 const heading = document.createElement('h3');
2942 heading.textContent = 'Place Image';
2943 heading.style.margin = '0 0 10px 0';

2944
2945 // Add image preview
2946 const preview = document.createElement('img');
2947 preview.src = imageUrl;
2948 preview.style maxWidth = '300px';
2949 preview.style maxHeight = '300px';
2950 preview.style display = 'block';
2951 preview.style marginBottom = '10px';

2952
2953 // Size controls
2954 const sizeControls = document.createElement('div');
2955 sizeControls.style.marginBottom = '10px';

2956
2957 const widthLabel = document.createElement('label');
2958 widthLabel.textContent = 'Width: ';
2959 const widthInput = document.createElement('input');
2960 widthInput.type = 'number';
2961 widthInput.value = Math.round(width);
2962 widthInput.style.width = '60px';

2963
2964 const heightLabel = document.createElement('label');
2965 heightLabel.textContent = 'Height: ';
2966 heightLabel.style.marginLeft = '10px';
2967 const heightInput = document.createElement('input');
2968 heightInput.type = 'number';
2969 heightInput.value = Math.round(height);
```

```
2971 heightInput.style.width = '60px';  
2972  
2973 // Maintain aspect ratio  
2974 const aspectRatio = img.width / img.height;  
2975  
2976 widthInput.addEventListener('input', () => {  
2977     const newWidth = parseInt(widthInput.value);  
2978     if (!isNaN(newWidth)) {  
2979         heightInput.value = Math.round(newWidth / aspectRatio);  
2980     }  
2981});  
2982  
2983 heightInput.addEventListener('input', () => {  
2984     const newHeight = parseInt(heightInput.value);  
2985     if (!isNaN(newHeight)) {  
2986         widthInput.value = Math.round(newHeight * aspectRatio);  
2987     }  
2988});  
2989  
2990 sizeControls.appendChild(widthLabel);  
2991 sizeControls.appendChild(widthInput);  
2992 sizeControls.appendChild(heightLabel);  
2993 sizeControls.appendChild(heightInput);  
2994  
2995 // Buttons container  
2996 const buttons = document.createElement('div');  
2997 buttons.style.display = 'flex';  
2998 buttons.style.justifyContent = 'space-between';  
2999 buttons.style.marginTop = '15px';  
3000  
3001 // Place button  
3002 const placeButton = document.createElement('button');  
3003 placeButton.textContent = 'Place Image';  
3004 placeButton.style.padding = '8px 15px';  
3005 placeButton.style.backgroundColor = '#4CAF50';  
3006 placeButton.style.color = 'white';  
3007 placeButton.style.border = 'none';  
3008 placeButton.style.borderRadius = '4px';  
3009 placeButton.style.cursor = 'pointer';  
3010  
3011 placeButton.addEventListener('click', () => {  
3012     // Get center of view as placement position  
3013     const containerRect =  
this.layerSystem.container.getBoundingClientRect();  
3014     const x = containerRect.width / 2;  
3015     const y = containerRect.height / 2;  
3016  
3017     // Get dimensions from inputs  
3018     const finalWidth = parseInt(widthInput.value);  
3019     const finalHeight = parseInt(heightInput.value);
```

```
3020
3021          // Create and add image object
3022          const imageObj = new ImageObject({
3023              x: x - finalWidth / 2,
3024              y: y - finalHeight / 2,
3025              width: finalWidth,
3026              height: finalHeight,
3027              imageUrl: imageUrl
3028      });
3029
3030      this.layerSystem.addObject(imageObj);
3031
3032      // Remove UI
3033      document.body.removeChild(placementUI);
3034  });
3035
3036      // Cancel button
3037      const cancelButton = document.createElement('button');
3038      cancelButton.textContent = 'Cancel';
3039      cancelButton.style.padding = '8px 15px';
3040      cancelButton.style.backgroundColor = '#f44336';
3041      cancelButton.style.color = 'white';
3042      cancelButton.style.border = 'none';
3043      cancelButton.style.borderRadius = '4px';
3044      cancelButton.style.cursor = 'pointer';
3045
3046      cancelButton.addEventListener('click', () => {
3047          document.body.removeChild(placementUI);
3048      });
3049
3050      buttons.appendChild(cancelButton);
3051      buttons.appendChild(placeButton);
3052
3053      // Assemble UI
3054      placementUI.appendChild(heading);
3055      placementUI.appendChild(preview);
3056      placementUI.appendChild(sizeControls);
3057      placementUI.appendChild(buttons);
3058
3059      // Add to document
3060      document.body.appendChild(placementUI);
3061  };
3062
3063      img.src = imageUrl;
3064  }
3065 }
3066
3067 // Symbol tool
3068 class SymbolTool extends DrawingTool {
3069     constructor(toolSystem) {
```

```
3070         super(toolSystem);
3071         this.symbolSelector = null;
3072     }
3073
3074     attachEvents() {
3075         const canvas = this.layerSystem.getActiveLayer().canvas;
3076
3077         canvas.addEventListener('click', this.handleClick);
3078     }
3079
3080     detachEvents() {
3081         const canvas = this.layerSystem.getActiveLayer().canvas;
3082
3083         canvas.removeEventListener('click', this.handleClick);
3084
3085         // Remove any active symbol selector
3086         this.removeSymbolSelector();
3087     }
3088
3089     handleClick = (e) => {
3090         const coords = this.layerSystem.screenToCanvas(e.clientX, e.clientY);
3091
3092         // Show symbol selector
3093         this.showSymbolSelector(coords.x, coords.y);
3094     }
3095
3096     showSymbolSelector(x, y) {
3097         // Remove existing selector if any
3098         this.removeSymbolSelector();
3099
3100         // Create symbol selector
3101         this.symbolSelector = document.createElement('div');
3102         this.symbolSelector.style.position = 'absolute';
3103         this.symbolSelector.style.zIndex = '100';
3104
3105         // Position relative to canvas
3106         const canvasRect = this.layerSystem.container.getBoundingClientRect();
3107         this.symbolSelector.style.left = (canvasRect.left + x + 10) + 'px';
3108         this.symbolSelector.style.top = (canvasRect.top + y + 10) + 'px';
3109
3110         // Style
3111         this.symbolSelector.style.backgroundColor = 'white';
3112         this.symbolSelector.style.border = '1px solid #ccc';
3113         this.symbolSelector.style.padding = '10px';
3114         this.symbolSelector.style.borderRadius = '5px';
3115         this.symbolSelector.style.boxShadow = '0 2px 10px rgba(0,0,0,0.1)';
3116
3117         // Add title
3118         const title = document.createElement('h4');
3119         title.textContent = 'Select Animation Symbol';
```

```
3120     title.style.margin = '0 0 10px 0';
3121     this.symbolSelector.appendChild(title);
3122
3123     // Symbol grid
3124     const symbolGrid = document.createElement('div');
3125     symbolGrid.style.display = 'grid';
3126     symbolGrid.style.gridTemplateColumns = 'repeat(3, 1fr)';
3127     symbolGrid.style.gap = '10px';
3128     symbolGrid.style.marginBottom = '10px';
3129
3130     // Available symbols
3131     const symbols = [
3132         { type: 'anticipation', name: 'Anticipation' },
3133         { type: 'impact', name: 'Impact' },
3134         { type: 'keyframe', name: 'Keyframe' },
3135         { type: 'inbetween', name: 'Inbetween' },
3136         { type: 'hold', name: 'Hold' },
3137         { type: 'default', name: 'Default' }
3138     ];
3139
3140     // Create symbol preview for each type
3141     symbols.forEach(symbol => {
3142         const symbolItem = document.createElement('div');
3143         symbolItem.style.display = 'flex';
3144         symbolItem.style.flexDirection = 'column';
3145         symbolItem.style.alignItems = 'center';
3146         symbolItem.style.cursor = 'pointer';
3147         symbolItem.style.padding = '5px';
3148         symbolItem.style.border = '1px solid #ddd';
3149         symbolItem.style.borderRadius = '3px';
3150
3151         // Create canvas for symbol preview
3152         const canvas = document.createElement('canvas');
3153         canvas.width = 50;
3154         canvas.height = 50;
3155         const ctx = canvas.getContext('2d');
3156
3157         // Draw symbol preview
3158         ctx.save();
3159         ctx.strokeStyle = this.settings.color;
3160         ctx.fillStyle = this.settings.color;
3161         ctx.lineWidth = 2;
3162
3163         // Draw centered preview
3164         const symbolObj = new SymbolObject({
3165             x: 25,
3166             y: 25,
3167             symbolType: symbol.type,
3168             color: this.settings.color,
3169             lineWidth: 2,
```

```
3170         scale: 1.5
3171     });
3172
3173     symbolObj.draw(ctx);
3174     ctx.restore();
3175
3176     const name = document.createElement('div');
3177     name.textContent = symbol.name;
3178     name.style.marginTop = '5px';
3179     name.style.fontSize = '12px';
3180
3181     symbolItem.appendChild(canvas);
3182     symbolItem.appendChild(name);
3183
3184     // Add click handler
3185     symbolItem.addEventListener('click', () => {
3186         this.addSymbol(x, y, symbol.type);
3187         this.removeSymbolSelector();
3188     });
3189
3190     symbolGrid.appendChild(symbolItem);
3191 });
3192
3193 this.symbolSelector.appendChild(symbolGrid);
3194
3195 // Scale control
3196 const scaleContainer = document.createElement('div');
3197 scaleContainer.style.display = 'flex';
3198 scaleContainer.style.alignItems = 'center';
3199 scaleContainer.style.marginBottom = '10px';
3200
3201 const scaleLabel = document.createElement('label');
3202 scaleLabel.textContent = 'Scale: ';
3203
3204 const scaleInput = document.createElement('input');
3205 scaleInput.type = 'range';
3206 scaleInput.min = '0.5';
3207 scaleInput.max = '3';
3208 scaleInput.step = '0.1';
3209 scaleInput.value = this.settings.symbolScale;
3210 scaleInput.style.flex = '1';
3211 scaleInput.style.marginLeft = '5px';
3212
3213 const scaleValue = document.createElement('span');
3214 scaleValue.textContent = this.settings.symbolScale + 'x';
3215 scaleValue.style.marginLeft = '5px';
3216 scaleValue.style.width = '30px';
3217
3218 scaleInput.addEventListener('input', () => {
3219     this.settings.symbolScale = parseFloat(scaleInput.value);
```

```
3220     scaleValue.textContent = this.settings.symbolScale + 'x';
3221   });
3222
3223   scaleContainer.appendChild(scaleLabel);
3224   scaleContainer.appendChild(scaleInput);
3225   scaleContainer.appendChild(scaleValue);
3226
3227   this.symbolSelector.appendChild(scaleContainer);
3228
3229   // Add cancel button
3230   const cancelButton = document.createElement('button');
3231   cancelButton.textContent = 'Cancel';
3232   cancelButton.style.width = '100%';
3233   cancelButton.style.padding = '5px';
3234   cancelButton.style.backgroundColor = '#f44336';
3235   cancelButton.style.color = 'white';
3236   cancelButton.style.border = 'none';
3237   cancelButton.style.borderRadius = '3px';
3238   cancelButton.style.cursor = 'pointer';
3239
3240   cancelButton.addEventListener('click', () => {
3241     this.removeSymbolSelector();
3242   });
3243
3244   this.symbolSelector.appendChild(cancelButton);
3245
3246   // Add to document
3247   document.body.appendChild(this.symbolSelector);
3248 }
3249
3250 removeSymbolSelector() {
3251   if (this.symbolSelector && this.symbolSelector.parentNode) {
3252     this.symbolSelector.parentNode.removeChild(this.symbolSelector);
3253     this.symbolSelector = null;
3254   }
3255 }
3256
3257 addSymbol(x, y, symbolType) {
3258   // Create symbol object
3259   const symbolObj = new SymbolObject({
3260     x: x,
3261     y: y,
3262     symbolType: symbolType,
3263     color: this.settings.color,
3264     lineWidth: this.settings.lineWidth,
3265     scale: this.settings.symbolScale
3266   });
3267
3268   // Add to layer
3269   this.layerSystem.addObject(symbolObj);
```

```
3270     }
3271 }
3272
3273 // Frame-spanning line tool (for connecting cells across frames)
3274 class FrameSpanningLineTool extends DrawingTool {
3275     constructor(toolSystem) {
3276         super(toolSystem);
3277         this.startCell = null;
3278         this.currentLine = null;
3279         this.drawing = false;
3280     }
3281
3282     attachEvents() {
3283         // This tool works with the table cells, not the canvas
3284         this.setupCellEvents();
3285     }
3286
3287     detachEvents() {
3288         this.removeCellEvents();
3289
3290         // Finish any in-progress drawing
3291         this.finishDrawing();
3292     }
3293
3294     setupCellEvents() {
3295         // Find all table cells and add mousedown handler
3296         const cells = document.querySelectorAll('#xsheet-table td');
3297         cells.forEach(cell => {
3298             cell.addEventListener('mousedown', this.handleCellMouseDown);
3299             cell.addEventListener('mouseup', this.handleCellMouseUp);
3300
3301             // Add hover effect
3302             cell.addEventListener('mouseover', this.handleCellHover);
3303             cell.addEventListener('mouseout', this.handleCellOut);
3304
3305             // Store original background for hover effect
3306             if (!cell.dataset.originalBg) {
3307                 cell.dataset.originalBg = cell.style.backgroundColor || '';
3308             }
3309         });
3310     }
3311
3312     removeCellEvents() {
3313         const cells = document.querySelectorAll('#xsheet-table td');
3314         cells.forEach(cell => {
3315             cell.removeEventListener('mousedown', this.handleCellMouseDown);
3316             cell.removeEventListener('mouseup', this.handleCellMouseUp);
3317             cell.removeEventListener('mouseover', this.handleCellHover);
3318             cell.removeEventListener('mouseout', this.handleCellOut);
3319         });
3320     }
3321 }
```

```
3320         // Restore original background
3321         if (cell.dataset.originalBg) {
3322             cell.style.backgroundColor = cell.dataset.originalBg;
3323         }
3324     });
3325 }
3326
3327 handleCellHover = (e) => {
3328     const cell = e.currentTarget;
3329     cell.style.backgroundColor = 'rgba(0, 123, 255, 0.2)';
3330 }
3331
3332 handleCellOut = (e) => {
3333     const cell = e.currentTarget;
3334     if (!this.drawing || cell !== this.startCell) {
3335         cell.style.backgroundColor = cell.dataset.originalBg || '';
3336     }
3337 }
3338
3339 handleCellMouseDown = (e) => {
3340     const cell = e.currentTarget;
3341
3342         // Find the row and column index
3343         const row = cell.parentElement;
3344         const frameNumber = parseInt(row.getAttribute('data-frame') || row.className.replace('frame-', ''));
```

```
3345         const columnIndex = Array.from(row.children).indexOf(cell) + 1;
```

```
3346
3347         if (isNaN(frameNumber) || frameNumber <= 0) return;
```

```
3348
3349         // Store as start cell
3350         this.startCell = cell;
3351         this.drawing = true;
```

```
3352
3353         // Highlight the cell
3354         cell.style.backgroundColor = 'rgba(0, 123, 255, 0.4);
```

```
3355
3356         // Create temporary frame-spanning line object
3357         this.currentLine = new FrameSpanningLineObject({
```

```
3358             startFrame: frameNumber,
3359             startColumn: columnIndex,
3360             endFrame: frameNumber,
3361             endColumn: columnIndex,
3362             color: this.settings.color,
3363             lineWidth: this.settings.lineWidth,
3364             arrowEnd: true
3365         });
3366
3367         // Add to layer
3368         this.layerSystem.addObject(this.currentLine);
```

```
3369 }
3370
3371     handleCellMouseUp = (e) => {
3372         if (!this.drawing || !this.currentLine) return;
3373
3374         const cell = e.currentTarget;
3375
3376         // Skip if this is the same cell
3377         if (cell === this.startCell) {
3378             this.finishDrawing();
3379             return;
3380         }
3381
3382         // Find the row and column index
3383         const row = cell.parentElement;
3384         const frameNumber = parseInt(row.getAttribute('data-frame') || row.className.replace('frame-', ''));
```

3385 const columnIndex = Array.from(row.children).indexOf(cell) + 1;

```
3386
3387         if (isNaN(frameNumber) || frameNumber <= 0) {
3388             this.finishDrawing();
3389             return;
3390         }
3391
3392         // Update the end point of the line
3393         this.currentLine.endFrame = frameNumber;
3394         this.currentLine.endColumn = columnIndex;
3395
3396         // Redraw
3397         this.layerSystem.redrawLayer(this.layerSystem.activeLayerIndex);
3398
3399         // Finish drawing
3400         this.finishDrawing();
3401     }
3402
3403     finishDrawing() {
3404         if (this.drawing) {
3405             // Reset flags
3406             this.drawing = false;
3407
3408             // If the line is too short (same start and end), remove it
3409             if (this.currentLine &&
3410                 this.currentLine.startFrame === this.currentLine.endFrame &&
3411                 this.currentLine.startColumn === this.currentLine.endColumn) {
3412                 this.layerSystem.removeObject(this.currentLine);
3413             }
3414
3415             // Clear current line reference
3416             this.currentLine = null;
3417     }
```

```
3418         // Reset cell highlights
3419         if (this.startCell) {
3420             this.startCell.style.backgroundColor =
3421                 this.startCell.dataset.originalBg || '';
3422             this.startCell = null;
3423         }
3424     }
3425 }
3426
3427 // Eraser tool
3428 class EraserTool extends DrawingTool {
3429     constructor(toolSystem) {
3430         super(toolSystem);
3431     }
3432
3433     attachEvents() {
3434         const canvas = this.layerSystem.getActiveLayer().canvas;
3435
3436         canvas.addEventListener('mousedown', this.handleMouseDown);
3437     }
3438
3439     detachEvents() {
3440         const canvas = this.layerSystem.getActiveLayer().canvas;
3441
3442         canvas.removeEventListener('mousedown', this.handleMouseDown);
3443     }
3444
3445     handleMouseDown = (e) => {
3446         const coords = this.layerSystem.screenToCanvas(e.clientX, e.clientY);
3447
3448         // Find object at click position
3449         const hit = this.layerSystem.findObjectAt(coords.x, coords.y);
3450
3451         if (hit) {
3452             // Remove the object
3453             this.layerSystem.removeObject(hit.object, hit.layerIndex);
3454         }
3455     }
3456 }
3457
3458 /**
3459 * INTEGRATION WITH X-SHEET APPLICATION
3460 * Initialize the drawing system, save/load integration, and PDF/print handling
3461 */
3462
3463 // Initialize drawing system when document is loaded
3464 function initDrawingSystem() {
3465     // Setup after the table is rendered
3466     setTimeout(() => {
```

```
3467 // Get the X-Sheet table element
3468 const xsheetTable = document.getElementById('xsheet-table');
3469 if (!xsheetTable) {
3470     console.error('X-Sheet table not found');
3471     return;
3472 }
3473
3474 // Create drawing systems
3475 const drawingLayerSystem = new DrawingLayerSystem(xsheetTable);
3476 const drawingToolSystem = new DrawingToolSystem(drawingLayerSystem);
3477
3478 // Store global reference
3479 window.xsheetDrawing = {
3480     layerSystem: drawingLayerSystem,
3481     toolSystem: drawingToolSystem
3482 };
3483
3484 // Add to status message
3485 const statusElement = document.getElementById('status-message');
3486 if (statusElement) {
3487     statusElement.textContent = 'Drawing tools initialized';
3488 }
3489
3490 // Integrate with X-Sheet save/load system
3491 integrateWithXSheetSaveLoad();
3492
3493 // Integrate with X-Sheet PDF and printing
3494 integrateWithXSheetExport();
3495
3496 // Custom event for when drawing objects change
3497 document.addEventListener('xsheet-redraw', function() {
3498     drawingLayerSystem.redrawAll();
3499 });
3500
3501     console.log('Drawing tools initialized successfully');
3502 }, 500);
3503 }
3504
3505 // Integrate drawing data with X-Sheet save/load system
3506 function integrateWithXSheetSaveLoad() {
3507     // Store original functions
3508     const originalCollectData = window.collectData;
3509     const originalRestoreData = window.restoreData;
3510
3511     // Override collectData to include drawings
3512     window.collectData = function() {
3513         // Call original function to get base data
3514         const data = originalCollectData ? originalCollectData() : {};
3515
3516         // Add drawing data if drawing system is initialized
```

```
3517     if (window.xsheetDrawing && window.xsheetDrawing.layerSystem) {
3518         data.drawingLayers = [];
3519
3520         // Collect objects from each layer
3521         window.xsheetDrawing.layerSystem.layers.forEach(layer => {
3522             const layerData = {
3523                 name: layer.name,
3524                 visible: layer.visible,
3525                 objects: layer.objects.map(obj => obj.toJSON())
3526             };
3527
3528             data.drawingLayers.push(layerData);
3529         });
3530     }
3531
3532     return data;
3533 };
3534
3535     // Override restoreData to handle drawings
3536     window.restoreData = function(data) {
3537         // Call original function to restore base data
3538         if (originalRestoreData) {
3539             originalRestoreData(data);
3540         }
3541
3542         // Restore drawing data if available
3543         if (data.drawingLayers && window.xsheetDrawing &&
3544             window.xsheetDrawing.layerSystem) {
3545             const layerSystem = window.xsheetDrawing.layerSystem;
3546
3547             // Clear existing layers
3548             layerSystem.clearAllLayers();
3549
3550             // Restore each layer
3551             data.drawingLayers.forEach((layerData, index) => {
3552                 // Create layer if needed
3553                 if (index >= layerSystem.layers.length) {
3554                     layerSystem.addLayer(layerData.name);
3555                 } else {
3556                     layerSystem.layers[index].name = layerData.name;
3557                     layerSystem.layers[index].visible = layerData.visible;
3558                 }
3559
3560                 // Restore objects
3561                 layerData.objects.forEach(objData => {
3562                     const newObj = DrawingObjectFactory.createFromJSON(objData);
3563                     if (newObj) {
3564                         layerSystem.layers[index].objects.push(newObj);
3565                     }
3566                 });
3567             });
3568         }
3569     }
3570 }
```

```
3566         });
3567
3568         // Redraw
3569         layerSystem.redrawAll();
3570     }
3571   };
3572 }
3573
3574 // Integrate drawing layers with PDF export and printing
3575 function integrateWithXSheetExport() {
3576   // Store original functions
3577   const originalExportToPDF = window.exportToPDF;
3578   const originalPrintSheet = window.printSheet;
3579
3580   // Override PDF export to include drawings
3581   window.exportToPDF = function () {
3582     if (!window.xsheetDrawing || !window.xsheetDrawing.layerSystem) {
3583       return originalExportToPDF ? originalExportToPDF() : null;
3584     }
3585
3586     // Save current state
3587     const layerSystem = window.xsheetDrawing.layerSystem;
3588     const layerContainer = layerSystem.container;
3589
3590     // 1) Save original CSS & DOM properties
3591     const originalDisplay = layerContainer.style.display;
3592     const originalPos = layerContainer.style.position;
3593     const originalTop = layerContainer.style.top;
3594     const originalLeft = layerContainer.style.left;
3595     const originalZIndex = layerContainer.style.zIndex;
3596     const originalWidth = layerContainer.style.width;
3597     const originalHeight = layerContainer.style.height;
3598     const originalParent = layerContainer.parentNode;
3599
3600     // 2) Make it visible for capture
3601     layerContainer.style.display = 'block';
3602
3603     // 3) Compute offsets - FIXED POSITIONING LOGIC
3604     const printableArea = document.getElementById('printable-area');
3605     const printableRect = printableArea.getBoundingClientRect();
3606     const tableRect = document.getElementById('xsheet-
3607 table').getBoundingClientRect();
3608
3609     // 4) Reparent into the printable area
3610     printableArea.appendChild(layerContainer);
3611
3612     // 5) Force absolute positioning & explicit size - FIXED POSITIONING LOGIC
3613     layerContainer.style.position = 'absolute';
3614     layerContainer.style.top = `${tableRect.top - printableRect.top}px`;
3615     layerContainer.style.left = `${tableRect.left - printableRect.left}px`;
```

```
3615 layerContainer.style.width = `${tableRect.width}px`;
3616 layerContainer.style.height = `${tableRect.height}px`;
3617 layerContainer.style.zIndex = '1000';
3618
3619 // Force a redraw of all drawing objects
3620 layerSystem.redrawAll();
3621
3622 // 6) Wait a brief moment to ensure drawings are rendered
3623 setTimeout(() => {
3624     // 7) Generate the PDF
3625     const result = originalExportToPDF ? originalExportToPDF() : null;
3626
3627     // 8) Restore everything
3628     layerContainer.style.display = originalDisplay;
3629     layerContainer.style.position = originalPos;
3630     layerContainer.style.top = originalTop;
3631     layerContainer.style.left = originalLeft;
3632     layerContainer.style.zIndex = originalZIndex;
3633     layerContainer.style.width = originalWidth;
3634     layerContainer.style.height = originalHeight;
3635     originalParent.appendChild(layerContainer);
3636 }, 50);
3637
3638     return true; // Will be async due to setTimeout
3639 };
3640
3641 // Override print to include drawings
3642 window.printSheet = function () {
3643     // 1) Fallback if no drawing system
3644     if (!window.xsheetDrawing || !window.xsheetDrawing.layerSystem) {
3645         return originalPrintSheet ? originalPrintSheet() : null;
3646     }
3647
3648     // 2) Grab the container and save its original state
3649     const layerSystem = window.xsheetDrawing.layerSystem;
3650     const layerContainer = layerSystem.container;
3651     const originalDisplay = layerContainer.style.display;
3652     const originalPos = layerContainer.style.position;
3653     const originalTop = layerContainer.style.top;
3654     const originalLeft = layerContainer.style.left;
3655     const originalZIndex = layerContainer.style.zIndex;
3656     const originalWidth = layerContainer.style.width;
3657     const originalHeight = layerContainer.style.height;
3658     const originalParent = layerContainer.parentNode;
3659
3660     // 3) Make it visible for print capture
3661     layerContainer.style.display = 'block';
3662
3663     // 4) Compute offsets relative to the sheet - FIXED POSITIONING LOGIC
3664     const printableArea = document.getElementById('printable-area');
```

```
3665     const printableRect = printableArea.getBoundingClientRect();
3666     const tableRect = document.getElementById('xsheet-
3667     table').getBoundingClientRect();
3668 
3669     // 5) Reparent into the printable-area element
3670     printableArea.appendChild(layerContainer);
3671 
3672     // 6) Force absolute positioning & lock size - FIXED POSITIONING LOGIC
3673     layerContainer.style.position = 'absolute';
3674     layerContainer.style.top = `${tableRect.top - printableRect.top}px`;
3675     layerContainer.style.left = `${tableRect.left - printableRect.left}px`;
3676     layerContainer.style.width = `${tableRect.width}px`;
3677     layerContainer.style.height = `${tableRect.height}px`;
3678     layerContainer.style.zIndex = '1000';
3679 
3680     // 7) Force a redraw of all drawing objects
3681     layerSystem.redrawAll();
3682 
3683     // 8) Add a print class to the container for print-specific CSS
3684     layerContainer.classList.add('printing');
3685 
3686     // 9) Call the original print with a slight delay to ensure drawing render
3687     setTimeout(() => {
3688         const result = originalPrintSheet ? originalPrintSheet() :
3689         window.print();
3690 
3691         // 10) Restore everything back
3692         layerContainer.style.display = originalDisplay;
3693         layerContainer.style.position = originalPos;
3694         layerContainer.style.top = originalTop;
3695         layerContainer.style.left = originalLeft;
3696         layerContainer.style.zIndex = originalZIndex;
3697         layerContainer.style.width = originalWidth;
3698         layerContainer.style.height = originalHeight;
3699         layerContainer.classList.remove('printing');
3700         originalParent.appendChild(layerContainer);
3701     }, 50);
3702 
3703     return true; // Will be async due to setTimeout
3704 }
3705 
3706 // Setup event listeners for table updates
3707 function setupXSheetUpdateHandling() {
3708     // Hook into existing update functions
3709     if (typeof window.generateTable === 'function') {
3710         const originalGenerateTable = window.generateTable;
3711         window.generateTable = function() {
3712             // Call original function
```

```
3713     const result = originalGenerateTable.apply(this, arguments);
3714
3715     // Fire update event
3716     document.dispatchEvent(new Event('xsheet-updated'));
3717
3718     return result;
3719   };
3720 }
3721
3722 if (typeof window.addEightRows === 'function') {
3723   const originalAddEightRows = window.addEightRows;
3724   window.addEightRows = function() {
3725     // Call original function
3726     const result = originalAddEightRows.apply(this, arguments);
3727
3728     // Fire update event
3729     document.dispatchEvent(new Event('xsheet-updated'));
3730
3731     return result;
3732   };
3733 }
3734
3735 if (typeof window.updateTemplate === 'function') {
3736   const originalUpdateTemplate = window.updateTemplate;
3737   window.updateTemplate = function() {
3738     // Call original function
3739     const result = originalUpdateTemplate.apply(this, arguments);
3740
3741     // Fire update event
3742     document.dispatchEvent(new Event('xsheet-updated'));
3743
3744     return result;
3745   };
3746 }
3747 }
3748
3749 // Original X-Sheet code
3750 // Wait for page load and libraries to initialize
3751 document.addEventListener('DOMContentLoaded', function () {
3752   // Global variables
3753   let frameCount = 96;
3754   let projectName = 'Animation_XSheet_' + new Date().toISOString().split('T')[0];
3755   let modified = false;
3756   let currentTemplate = 'large';
3757
3758   // Audio variables
3759   let audioContext = null;
3760   let audioBuffer = null;
3761   let audioSource = null;
3762   let audioData = null;
```

```
3763 let isPlaying = false;
3764 let startTime = 0;
3765 let startOffset = 0;
3766 let audioFileName = '';
3767 let waveformData = [];
3768 let waveformCanvases = [];
3769 let phonetics = [];
3770 let frameDuration = 1 / 24; // 24fps
3771 let currentFrame = 0;
3772 let phoneticEditPosition = null;
3773
3774 // Initialize UI elements
3775 const templateSelector = document.getElementById('template-selector');
3776 const frameCountInput = document.getElementById('frame-count');
3777 const saveButton = document.getElementById('save-button');
3778 const loadButton = document.getElementById('load-button');
3779 const pdfButton = document.getElementById('pdf-button');
3780 const printButton = document.getElementById('print-button');
3781 const addRowsButton = document.getElementById('add-rows-button');
3782 const clearButton = document.getElementById('clear-button');
3783 const tableBody = document.getElementById('xsheet-body');
3784 const statusMessage = document.getElementById('status-message');
3785 const audioButton = document.getElementById('audio-button');
3786 const audioUpload = document.getElementById('audio-upload');
3787 const playAudioButton = document.getElementById('play-audio');
3788 const stopAudioButton = document.getElementById('stop-audio');
3789 const audioScrubber = document.getElementById('audio-scrubber');
3790 const audioInfo = document.getElementById('audio-info');
3791 const addPhoneticButton = document.getElementById('add-phonetic');
3792 const phoneticInput = document.getElementById('phonetic-input');
3793 const phoneticText = document.getElementById('phonetic-text');
3794 const savePhoneticButton = document.getElementById('save-phonetic');
3795 const cancelPhoneticButton = document.getElementById('cancel-phonetic');
3796
3797 // Date field - set today by default
3798 const dateField = document.getElementById('project-date');
3799 dateField.valueAsDate = new Date();
3800
3801 // Initialize the table
3802 generateTable(frameCount);
3803
3804 // Initialize the drawing system
3805 initDrawingSystem();
3806 setupXSheetUpdateHandling();
3807
3808 // Event Listeners
3809 templateSelector.addEventListener('change', function () {
3810     currentTemplate = this.value;
3811     updateTemplate();
3812});
```

```
3813
3814     frameCountInput.addEventListener('change', function () {
3815         frameCount = parseInt(this.value);
3816         if (frameCount < 8) frameCount = 8;
3817         generateTable(frameCount);
3818         updateStatusMessage('Frame count updated to ' + frameCount);
3819
3820         // Re-render waveform if audio is loaded
3821         if (audioBuffer) {
3822             renderWaveform();
3823         }
3824     });
3825
3826     saveButton.addEventListener('click', saveProject);
3827     loadButton.addEventListener('click', loadProject);
3828     pdfButton.addEventListener('click', exportToPDF);
3829     printButton.addEventListener('click', printSheet);
3830     addRowsButton.addEventListener('click', addEightRows);
3831     clearButton.addEventListener('click', clearSheet);
3832
3833     // Audio related event listeners
3834     audioButton.addEventListener('click', function () {
3835         audioUpload.click();
3836     });
3837
3838     audioUpload.addEventListener('change', function (e) {
3839         if (e.target.files.length > 0) {
3840             const file = e.target.files[0];
3841             audioFileName = file.name;
3842             loadAudioFile(file);
3843         }
3844     });
3845
3846     playAudioButton.addEventListener('click', togglePlayAudio);
3847     stopAudioButton.addEventListener('click', stopAudio);
3848     audioScrubber.addEventListener('input', scrubAudio);
3849
3850     addPhoneticButton.addEventListener('click', function () {
3851         if (audioBuffer) {
3852             showPhoneticInput(null);
3853         } else {
3854             updateStatusMessage('Please load audio first');
3855         }
3856     });
3857
3858     savePhoneticButton.addEventListener('click', savePhoneticMarker);
3859     cancelPhoneticButton.addEventListener('click', function () {
3860         phoneticInput.style.display = 'none';
3861     });
3862
```

```
3863 // Monitor for changes to set modified flag
3864 document.addEventListener('input', function (e) {
3865     if (!e.target.matches('input, [contenteditable="true"]')) return;
3866     modified = true;
3867
3868     if (e.target.matches('[contenteditable="true"]')) {
3869         // Add the modified class when content is added
3870         e.target.classList.add('modified');
3871
3872         // Remove the modified class when the cell is empty
3873         if (e.target.textContent.trim() === '') {
3874             e.target.classList.remove('modified');
3875         }
3876     }
3877
3878     updateStatusMessage('Changes detected - not saved');
3879 });
3880
3881 // Variables for cell selection
3882 let selectedCells = [];
3883 let isSelecting = false;
3884 let selectionStart = null;
3885 let hasMovedDuringSelection = false;
3886
3887 // Set up multi-cell selection functionality
3888 function setupCellSelection() {
3889     const editableCells = document.querySelectorAll('[contenteditable="true"]');
3890
3891     // Clear selected cells when clicking outside
3892     document.addEventListener('click', function (e) {
3893         // If this was part of a drag operation, skip the click handler
3894         if (hasMovedDuringSelection) {
3895             // We removed this line to fix the selection issue
3896             // hasMovedDuringSelection = false;
3897             return;
3898         }
3899
3900         // Get the clicked element and check if it's inside an editable cell or
3901         // a selected cell
3902         const clickedCell = e.target.closest('[contenteditable="true"]');
3903         const clickedInSelection = e.target.closest('.selected-cell');
3904
3905         // If not clicking in a cell or selection and not using modifier keys
3906         if (!clickedCell && !clickedInSelection && !e.ctrlKey && !e.metaKey) {
3907             clearCellSelection();
3908         }
3909     });
3910
3911     // Handle clicks on the table for better cell focusing
```

```
3911 |     document.getElementById('xsheet-table').addEventListener('click', function
3912 |     (e) {
3913 |         // Find the closest editable cell from the click point
3914 |         const targetCell = e.target.closest('[contenteditable="true"]');
3915 |
3916 |         // If we found a cell and we're not in multi-select mode
3917 |         if (targetCell && !hasMovedDuringSelection && !e.ctrlKey && !e.metaKey)
3918 |             {
3919 |                 // Focus the cell and position cursor at the end
3920 |                 if (selectedCells.length === 1 && selectedCells[0] === targetCell) {
3921 |                     // If already selected, just ensure focus
3922 |                     targetCell.focus();
3923 |                     placeCaretAtEnd(targetCell);
3924 |                 }
3925 |             });
3926 |
3927 |             // Helper function to place caret at the end of content
3928 |             function placeCaretAtEnd(el) {
3929 |                 if (document.createRange) {
3930 |                     const range = document.createRange();
3931 |                     range.selectNodeContents(el);
3932 |                     range.collapse(false); // false means collapse to end
3933 |                     const selection = window.getSelection();
3934 |                     selection.removeAllRanges();
3935 |                     selection.addRange(range);
3936 |                 }
3937 |             }
3938 |
3939 |             // Handle mousedown for selection start
3940 |             editableCells.forEach(cell => {
3941 |                 // Mousedown - start potential selection
3942 |                 cell.addEventListener('mousedown', function (e) {
3943 |                     // Only handle left mouse button
3944 |                     if (e.button !== 0) return;
3945 |
3946 |                     // If using modifier keys for multi-select
3947 |                     if (e.ctrlKey || e.metaKey) {
3948 |                         toggleCellSelection(cell);
3949 |                         e.preventDefault(); // Prevent text cursor
3950 |                         return;
3951 |                     }
3952 |
3953 |                     // Clear previous selection unless clicking on already selected cell
3954 |                     if (!cell.classList.contains('selected-cell')) {
3955 |                         clearCellSelection();
3956 |                     }
3957 |
3958 |                     // Store starting cell and reset movement flag
3959 |                     selectionStart = cell;
```

```
3959         hasMovedDuringSelection = false;
3960         isSelecting = true;
3961
3962         // Initially add this cell to selection
3963         if (!cell.classList.contains('selected-cell')) {
3964             toggleCellSelection(cell, true);
3965         }
3966
3967         // If it's a single click (not start of drag), focus the cell
3968         cell.focus();
3969
3970         // Don't prevent default here to allow normal focus behavior
3971     });
3972
3973     // Additional keyboard events for selected cells
3974     cell.addEventListener('keydown', function (e) {
3975         // If Delete or Backspace key and we have selected cells
3976         if ((e.key === 'Delete' || e.key === 'Backspace') &&
3977             selectedCells.length > 1) {
3978             e.preventDefault();
3979             clearSelectedCellsContent();
3980         }
3981
3982         // Copy with Ctrl+C or Cmd+C
3983         if ((e.ctrlKey || e.metaKey) && e.key === 'c' &&
3984             selectedCells.length > 0) {
3985             e.preventDefault();
3986             copySelectedCells();
3987         }
3988
3989         // Select all cells in row with Ctrl+A or Cmd+A
3990         if ((e.ctrlKey || e.metaKey) && e.key === 'a') {
3991             e.preventDefault();
3992             selectRowCells(cell.parentElement);
3993         });
3994     });
3995
3996     // Global mousemove handler for extending selection
3997     document.addEventListener('mousemove', function (e) {
3998         if (!isSelecting) return;
3999
4000         // Set the flag that we've moved during this selection
4001         hasMovedDuringSelection = true;
4002
4003         // Find the element under the cursor
4004         const elementUnderCursor = document.elementFromPoint(e.clientX,
4005             e.clientY);
4006         if (!elementUnderCursor) return;
4007
4008         // Find the closest editable cell
4009     });
4010 }
```

```
4007     const targetCell =
4008       elementUnderCursor.closest('[contenteditable="true"]');
4009       if (targetCell) {
4010         // Add to selection
4011         toggleCellSelection(targetCell, true);
4012
4013         // Prevent text selection during drag
4014         e.preventDefault();
4015         if (window.getSelection) {
4016           window.getSelection().removeAllRanges();
4017         }
4018     });
4019
4020   // Global mouseup to end selection - THIS IS THE FIXED VERSION
4021   document.addEventListener('mouseup', function (e) {
4022     // End selection mode but keep selected cells
4023     if (isSelecting) {
4024       isSelecting = false;
4025
4026       // FIX: Use setTimeout to delay resetting the hasMovedDuringSelection flag
4027       // This gives the click handler a chance to see the flag first
4028       if (hasMovedDuringSelection) {
4029         setTimeout(function () {
4030           hasMovedDuringSelection = false;
4031         }, 10);
4032       }
4033
4034       // If this wasn't a drag and there's only one cell selected, focus
4035       // it
4036       if (!hasMovedDuringSelection && selectedCells.length === 1) {
4037         const cell = selectedCells[0];
4038         cell.focus();
4039         placeCaretAtEnd(cell);
4040       }
4041
4042       selectionStart = null;
4043     });
4044
4045   // Listen for paste events
4046   document.addEventListener('paste', function (e) {
4047     if (selectedCells.length > 0) {
4048       handlePaste(e);
4049     }
4050   });
4051
4052
4053   function toggleCellSelection(cell, addOnly = false) {
4054     const index = selectedCells.indexOf(cell);
```

```
4055
4056      if (index === -1) {
4057          // Add to selection
4058          selectedCells.push(cell);
4059          cell.classList.add('selected-cell');
4060      } else if (!addOnly) {
4061          // Remove from selection if not add-only mode
4062          selectedCells.splice(index, 1);
4063          cell.classList.remove('selected-cell');
4064      }
4065  }
4066
4067  function clearCellSelection() {
4068      selectedCells.forEach(cell => {
4069          cell.classList.remove('selected-cell');
4070      });
4071      selectedCells = [];
4072  }
4073
4074  function clearSelectedCellsContent() {
4075      selectedCells.forEach(cell => {
4076          cell.textContent = '';
4077          cell.classList.remove('modified');
4078      });
4079      modified = true;
4080      updateStatusMessage('Cleared selected cells');
4081  }
4082
4083  function copySelectedCells() {
4084      if (selectedCells.length === 0) return;
4085
4086      // Create a text representation of selected cells
4087      const cellData = selectedCells.map(cell => cell.textContent || '')
4088          .join('\t');
4089
4090      // Copy to clipboard using Clipboard API
4091      navigator.clipboard.writeText(cellData)
4092          .then(() => {
4093              updateStatusMessage('Copied selected cells to clipboard');
4094          })
4095          .catch(err => {
4096              updateStatusMessage('Failed to copy: ' + err);
4097          });
4098
4099  function handlePaste(e) {
4100      // Prevent default paste behavior
4101      e.preventDefault();
4102
4103      // Get clipboard data
```

```
4104     const clipboardData = e.clipboardData || window.clipboardData;
4105     const pastedText = clipboardData.getData('text');
4106
4107     // If we have a focused element within selection, paste there
4108     const activeElement = document.activeElement;
4109     if (activeElement && selectedCells.includes(activeElement)) {
4110         activeElement.textContent = pastedText;
4111         activeElement.classList.add('modified');
4112         modified = true;
4113     } else if (selectedCells.length > 0) {
4114         // Otherwise paste into first selected cell
4115         selectedCells[0].textContent = pastedText;
4116         selectedCells[0].classList.add('modified');
4117         modified = true;
4118     }
4119
4120     if (pastedText.trim() === '') {
4121         selectedCells.forEach(cell => {
4122             cell.classList.remove('modified');
4123         });
4124     }
4125
4126     updateStatusMessage('Pasted content into selected cell');
4127 }
4128
4129 function selectRowCells(row) {
4130     // Clear previous selection
4131     clearCellSelection();
4132
4133     // Select all editable cells in the row
4134     const cells = Array.from(row.cells).filter(cell => cell.contentEditable ===
4135     'true');
4136     cells.forEach(cell => {
4137         toggleCellSelection(cell, true);
4138     });
4139
4140     // Functions
4141     function generateTable(frames) {
4142         tableBody.innerHTML = '';
4143         waveformCanvases = [];
4144
4145         // Create a column container for the waveform
4146         const waveformColContainer = document.createElement('div');
4147         waveformColContainer.className = 'waveform-col-container';
4148
4149         for (let i = 1; i <= frames; i++) {
4150             const row = document.createElement('tr');
4151             row.className = `frame-${i}`;
4152             row.setAttribute('data-frame', i);
```

```
4153
4154
4155 // Create standard cell
4156 function createCell(className, isEditable = true, frameNum = null) {
4157     const cell = document.createElement('td');
4158     cell.className = className;
4159
4160     if (isEditable) {
4161         cell.contentEditable = true;
4162         cell.setAttribute('data-placeholder', '');
4163         cell.setAttribute('tabindex', '0');
4164     } else if (frameNum !== null) {
4165         cell.textContent = frameNum;
4166         cell.className += ' frame-number';
4167
4168         // Color the first frame green for reference (like in the
4169         // example image)
4170         if (frameNum === 1) {
4171             cell.style.backgroundColor = '#00cc00';
4172         } else {
4173             cell.style.backgroundColor = '#cccccc';
4174         }
4175
4176         return cell;
4177     }
4178
4179 // Create waveform cell (placeholder for the vertical waveform)
4180 function createWaveformCell(frameNum) {
4181     const cell = document.createElement('td');
4182     cell.className = 'waveform-col';
4183     cell.setAttribute('data-frame', frameNum);
4184     return cell;
4185
4186 // Add all cells to the row
4187 row.appendChild(createCell('action-col'));
4188 row.appendChild(createCell('frame-col', false, i));
4189 row.appendChild(createWaveformCell(i));
4190 row.appendChild(createCell('dialogue-col'));
4191 row.appendChild(createCell('sound-col'));
4192 row.appendChild(createCell('technical-col'));
4193 row.appendChild(createCell('extra1-col'));
4194 row.appendChild(createCell('extra2-col'));
4195 row.appendChild(createCell('frame-col', false, i));
4196 row.appendChild(createCell('camera-col'));
4197
4198 // Add special styling for 8-frame and 24-frame intervals
4199 if (i % 24 === 0) {
4200     row.style.borderBottom = '4px double #000';
4201     for (let cell of row.cells) {
```

```
4202         cell.style.borderBottom = '4px double #000';
4203         cell.style.fontWeight = 'bold';
4204     }
4205 } else if (i % 8 === 0) {
4206     row.style.borderBottom = '2px solid #000';
4207     for (let cell of row.cells) {
4208         cell.style.borderBottom = '2px solid #000';
4209         cell.style.fontWeight = 'bold';
4210     }
4211 }
4212
4213     tableBody.appendChild(row);
4214 }
4215
4216     setupCellNavigation();
4217     updateStatusMessage('Table generated with ' + frames + ' frames');
4218
4219     // Fire custom event for drawing system to update
4220     document.dispatchEvent(new Event('xsheet-updated'));
4221 }
4222
4223 function updateTemplate() {
4224     if (currentTemplate === 'large') {
4225         // 11"x17" template (96 frames)
4226         document.body.style.maxWidth = '11in';
4227         document.body.style.maxHeight = '17in';
4228         document.body.style.fontSize = '9pt';
4229         frameCountInput.value = 96;
4230         frameCount = 96;
4231     } else {
4232         // 8"x10" template (48 frames)
4233         document.body.style.maxWidth = '8in';
4234         document.body.style.maxHeight = '10in';
4235         document.body.style.fontSize = '8pt';
4236         frameCountInput.value = 48;
4237         frameCount = 48;
4238     }
4239
4240     generateTable(frameCount);
4241     updateStatusMessage('Template switched to ' + (currentTemplate === 'large' ?
4242         '11"x17"' : '8"x10"'));
4243
4244     // Re-render waveform if audio is loaded
4245     if (audioBuffer) {
4246         renderWaveform();
4247     }
4248
4249     function setupCellNavigation() {
4250         const editableCells = document.querySelectorAll('[contenteditable="true"]');
```

```
4251
4252     editableCells.forEach(cell => {
4253         cell.addEventListener('keydown', function (e) {
4254             // Tab navigation
4255             if (e.key === 'Tab') {
4256                 e.preventDefault();
4257                 const currentRow = this.parentElement;
4258                 const currentIndex = Array.from(currentRow.cells).indexOf(this);
4259
4260                 if (e.shiftKey) {
4261                     // Shift+Tab moves backward
4262                     let prevCell = null;
4263                     if (currentIndex > 0) {
4264                         // Find previous editable cell in same row
4265                         for (let i = currentIndex - 1; i >= 0; i--) {
4266                             if (currentRow.cells[i].contentEditable === 'true')
4267                             {
4268                                 prevCell = currentRow.cells[i];
4269                                 break;
4270                             }
4271                         }
4272
4273                         if (!prevCell) {
4274                             // Move to previous row, last cell
4275                             const prevRow = currentRow.previousElementSibling;
4276                             if (prevRow) {
4277                                 const cells = Array.from(prevRow.cells).filter(c =>
4278                                     c.contentEditable === 'true');
4279                                     prevCell = cells[cells.length - 1];
4280                                 }
4281
4282                             if (prevCell) {
4283                                 prevCell.focus();
4284                                 // Place cursor at end of content
4285                                 const range = document.createRange();
4286                                 const sel = window.getSelection();
4287                                 range.selectNodeContents(prevCell);
4288                                 range.collapse(false);
4289                                 sel.removeAllRanges();
4290                                 sel.addRange(range);
4291                             }
4292                         } else {
4293                             // Tab moves forward
4294                             let nextCell = null;
4295                             if (currentIndex < currentRow.cells.length - 1) {
4296                                 // Find next editable cell in same row
4297                                 for (let i = currentIndex + 1; i <
4298                                     currentRow.cells.length; i++) {
```

```

4298 {
4299     if (currentRow.cells[i].contentEditable === 'true') {
4300         nextCell = currentRow.cells[i];
4301         break;
4302     }
4303 }
4304
4305     if (!nextCell) {
4306         // Move to next row, first cell
4307         const nextRow = currentRow.nextElementSibling;
4308         if (nextRow) {
4309             nextCell = Array.from(nextRow.cells).find(c =>
4310             c.contentEditable === 'true');
4311         }
4312
4313         if (nextCell) {
4314             nextCell.focus();
4315         }
4316     }
4317 }
4318
4319     // Enter key to move down
4320     if (e.key === 'Enter' && !e.shiftKey) {
4321         e.preventDefault();
4322         const currentRow = this.parentElement;
4323         const nextRow = currentRow.nextElementSibling;
4324         const currentIndex = Array.from(currentRow.cells).indexOf(this);
4325
4326         if (nextRow) {
4327             const cells = Array.from(nextRow.cells);
4328             const samePositionCell = cells[currentIndex];
4329             if (samePositionCell && samePositionCell.contentEditable ===
4330                 'true') {
4331                 samePositionCell.focus();
4332             }
4333         }
4334     });
4335
4336     // Add listener to remove the modified class when cell is emptied
4337     cell.addEventListener('keyup', function (e) {
4338         if ((e.key === 'Delete' || e.key === 'Backspace') &&
4339             this.textContent.trim() === '') {
4340             this.classList.remove('modified');
4341         }
4342     });
4343
4344     // Setup cell selection functionality

```

```
4345         setupCellSelection();
4346     }
4347
4348     // Audio functions
4349     function loadAudioFile(file) {
4350         if (!audioContext) {
4351             try {
4352                 window.AudioContext = window.AudioContext || window.webkitAudioContext;
4353                 audioContext = new AudioContext();
4354             } catch (e) {
4355                 updateStatusMessage('Web Audio API is not supported in this
4356 browser');
4357             return;
4358         }
4359
4360         const reader = new FileReader();
4361
4362         reader.onload = function (e) {
4363             const audioData = e.target.result;
4364
4365             // Update status
4366             updateStatusMessage('Decoding audio...');
4367
4368             // Decode the audio data
4369             audioContext.decodeAudioData(audioData, function (buffer) {
4370                 audioBuffer = buffer;
4371
4372                 // Update audio info
4373                 const duration = buffer.duration;
4374                 const minutes = Math.floor(duration / 60);
4375                 const seconds = Math.floor(duration % 60);
4376                 const frameCount = Math.ceil(duration * 24); // Assuming 24fps
4377
4378                 audioInfo.textContent = `${audioFileName}
4379 (${minutes}:${seconds.toString().padStart(2, '0')}, ${frameCount} frames @ 24fps}`;
4380
4381                 // Enable audio controls
4382                 playAudioButton.disabled = false;
4383                 stopAudioButton.disabled = false;
4384                 audioScrubber.disabled = false;
4385
4386                 // Generate waveform visualization
4387                 generateWaveformData(buffer);
4388
4389                 // Update status
4390                 updateStatusMessage('Audio loaded: ' + audioFileName);
4391
4392                 // If the x-sheet has fewer frames than the audio, suggest
4393                 increasing
```

```
4392         if (frameCount > frameCountInput.value) {
4393             if (confirm(`This audio is ${frameCount} frames long at 24fps,
but your X-sheet only has ${frameCountInput.value} frames. Do you want to increase the frame
count?`)) {
4394                 frameCountInput.value = frameCount;
4395                 frameCount = frameCount;
4396                 generateTable(frameCount);
4397                 renderWaveform();
4398             } else {
4399                 renderWaveform();
4400             }
4401         } else {
4402             renderWaveform();
4403         }
4404     }, function (e) {
4405         updateStatusMessage('Error decoding audio: ' + e.message);
4406     });
4407 };
4408
4409 reader.onerror = function () {
4410     updateStatusMessage('Error reading audio file');
4411 };
4412
4413 reader.readAsArrayBuffer(file);
4414 }
4415
4416 function generateWaveformData(buffer) {
4417     // Get the raw audio data from the buffer
4418     const rawData = buffer.getChannelData(0); // Use first channel
4419
4420     // Calculate how many samples we need for our visualization
4421     const totalSamples = rawData.length;
4422
4423     // Process for visualization - we need to reduce the resolution
4424     // to make it efficient to display
4425     waveformData = [];
4426
4427     // Calculate desired number of points for the visualization
4428     // For vertical waveform, we want more detail
4429     const pointsPerSecond = 100; // Increase detail for vertical display
4430     const totalPoints = Math.ceil(buffer.duration * pointsPerSecond);
4431
4432     // Calculate step size
4433     const step = Math.floor(totalSamples / totalPoints);
4434
4435     // Build the waveform data array
4436     for (let i = 0; i < totalPoints; i++) {
4437         const index = Math.floor(i * step);
4438         if (index < totalSamples) {
4439             // Get the absolute value for a nicer visual
```

```
4440         waveformData.push(Math.abs(rawData[index]));
4441     }
4442   }
4443 }
4444
4445 function renderWaveform() {
4446   if (!audioBuffer || waveformData.length === 0) return;
4447
4448   // Clear existing markers and components
4449   const existingMarkers = document.querySelectorAll('.waveform-marker');
4450   existingMarkers.forEach(marker => marker.remove());
4451
4452   const existingLabels = document.querySelectorAll('.phonetic-label');
4453   existingLabels.forEach(label => label.remove());
4454
4455   const existingContainer = document.querySelector('.waveform-container');
4456   if (existingContainer) {
4457     existingContainer.remove();
4458   }
4459
4460   // Get the table and first waveform column cell for positioning
4461   const table = document.getElementById('xsheet-table');
4462   const firstCell = document.querySelector('.waveform-col[data-frame="1"]');
4463   if (!firstCell) return;
4464
4465   // Calculate total height needed for the waveform
4466   const totalRows = document.querySelectorAll('tr[data-frame]').length;
4467   const rowHeight = firstCell.offsetHeight;
4468   const totalHeight = totalRows * rowHeight;
4469
4470   // Create a container for the vertical waveform that spans the entire table
4471   const waveformContainer = document.createElement('div');
4472   waveformContainer.className = 'waveform-container';
4473
4474   // Create a canvas for the waveform
4475   const canvas = document.createElement('canvas');
4476   canvas.className = 'waveform-canvas';
4477   canvas.width = firstCell.offsetWidth;
4478   canvas.height = totalHeight;
4479   waveformContainer.appendChild(canvas);
4480
4481   // Add overlay for event handling
4482   const overlay = document.createElement('div');
4483   overlay.className = 'waveform-overlay';
4484   overlay.style.height = totalHeight + 'px';
4485
4486   // Add event listener for clicking on the waveform
4487   overlay.addEventListener('click', function (e) {
4488     if (!audioBuffer) return;
4489   })
}
```

```
4490 const rect = overlay.getBoundingClientRect();
4491 const y = e.clientY - rect.top;
4492 const percentage = y / rect.height;
4493
4494 // Calculate time point in the audio
4495 const timePoint = percentage * audioBuffer.duration;
4496
4497 // Update audio position
4498 audioScrubber.value = (timePoint / audioBuffer.duration) * 100;
4499 if (isPlaying) {
4500     stopAudio();
4501     startOffset = timePoint;
4502     playAudio();
4503 } else {
4504     startOffset = timePoint;
4505 }
4506
4507 // Update the UI to show frame position
4508 updateFrameMarker();
4509 });
4510
4511 // Variables for scrubbing
4512 let isScrubbing = false;
4513
4514 // Add scrubbing functionality (dragging while holding mouse button)
4515 overlay.addEventListener('mousedown', function (e) {
4516     if (!audioBuffer) return;
4517     if (e.button !== 0) return; // Only respond to left mouse button
4518
4519     isScrubbing = true;
4520
4521     // Pause any playing audio
4522     if (isPlaying) {
4523         pauseAudio();
4524     }
4525     // - NEW: allow pen (pointerType="pen") to scrub as well -
4526     overlay.addEventListener('pointerdown', function (e) {
4527         if (!audioBuffer || e.pointerType !== 'pen') return;
4528         isScrubbing = true;
4529         if (isPlaying) pauseAudio(); // pause continuous playback
4530         updateScrubPosition(e); // scrub to this location
4531         e.preventDefault(); // prevent default pen
4532     behavior
4533     });
4534     overlay.addEventListener('pointermove', function (e) {
4535         // - FIX: only scrub while the pen is physically pressing
4536         if (isScrubbing && e.pointerType === 'pen' && e.pressure > 0) {
4537             updateScrubPosition(e);
4538         }
4539     });
4540 }
```

```
4538    });
4539    overlay.addEventListener('pointerup', function (e) {
4540        if (e.pointerType === 'pen') {
4541            isScrubbing = false;
4542        }
4543    });
4544
4545    // Create a scrub audio context if needed
4546    if (!audioContext) {
4547        try {
4548            window.AudioContext = window.AudioContext || window.webkitAudioContext;
4549            audioContext = new AudioContext();
4550        } catch (e) {
4551            console.error('Web Audio API not supported');
4552            return;
4553        }
4554    }
4555
4556    // Initial scrub to current position
4557    updateScrubPosition(e);
4558
4559    // Prevent text selection during drag
4560    e.preventDefault();
4561});
4562
4563    // Handle scrubbing movement
4564    overlay.addEventListener('mousemove', function (e) {
4565        if (!isScrubbing || !audioBuffer) return;
4566        updateScrubPosition(e);
4567    });
4568
4569    // Function to update position during scrubbing
4570    function updateScrubPosition(e) {
4571        const rect = overlay.getBoundingClientRect();
4572        const y = e.clientY - rect.top;
4573        const percentage = Math.max(0, Math.min(1, y / rect.height));
4574
4575        // Calculate time point in the audio
4576        const timePoint = percentage * audioBuffer.duration;
4577        startOffset = timePoint;
4578
4579        // Update audio scrubber
4580        audioScrubber.value = percentage * 100;
4581
4582        // Play a short snippet of audio at this position
4583        playScrubAudio(timePoint);
4584
4585        // Update the marker
4586        updateFrameMarker();
```

```
4587 }
4588
4589 // End scrubbing when mouse is released or leaves element
4590 overlay.addEventListener('mouseup', function () {
4591     isScrubbing = false;
4592 });
4593
4594 overlay.addEventListener('mouseleave', function () {
4595     isScrubbing = false;
4596 });
4597
4598 // Function to play a short snippet at the scrub position
4599 let scrubSource = null;
4600 function playScrubAudio(timePoint) {
4601     if (scrubSource) {
4602         try {
4603             scrubSource.stop();
4604         } catch (e) {
4605             // Ignore errors when stopping already stopped source
4606         }
4607     }
4608
4609     // Play a very short snippet at the current position
4610     scrubSource = audioContext.createBufferSource();
4611     scrubSource.buffer = audioBuffer;
4612     scrubSource.connect(audioContext.destination);
4613
4614     // Play just a short snippet (equivalent to 1-2 frames at 24fps)
4615     const snippetDuration = 1 / 12; // 1/12 of a second (2 frames at 24fps)
4616     scrubSource.start(0, timePoint, snippetDuration);
4617 }
4618
4619 // Right-click to add phonetic marker
4620 overlay.addEventListener('contextmenu', function (e) {
4621     e.preventDefault();
4622     if (!audioBuffer) return;
4623
4624     const rect = overlay.getBoundingClientRect();
4625     const y = e.clientY - rect.top;
4626     const percentage = y / rect.height;
4627
4628     // Calculate time point in the audio
4629     const timePoint = percentage * audioBuffer.duration;
4630
4631     showPhoneticInput(timePoint);
4632     return false;
4633 });
4634
4635 waveformContainer.appendChild(overlay);
4636
```

```
// Add the container to the table
document.body.appendChild(waveformContainer);

// Position the container over the waveform column
const tableRect = table.getBoundingClientRect();
const cellRect = firstCell.getBoundingClientRect();

waveformContainer.style.left = (cellRect.left - 1) + 'px';
waveformContainer.style.top = (cellRect.top) + 'px';
waveformContainer.style.width = (cellRect.width) + 'px';
waveformContainer.style.height = totalHeight + 'px';

// Draw the waveform on the canvas
const ctx = canvas.getContext('2d');
const width = canvas.width;
const height = canvas.height;

// Clear and set background
ctx.clearRect(0, 0, width, height);
ctx.fillStyle = '#ffffff';
ctx.fillRect(0, 0, width, height);

// Calculate scaling factors
const totalDuration = audioBuffer.duration;
const framesPerSecond = 24;
const totalFrames = Math.ceil(totalDuration * framesPerSecond);

// Draw center line
ctx.beginPath();
ctx.strokeStyle = '#cccccc';
ctx.moveTo(width / 2, 0);
ctx.lineTo(width / 2, height);
ctx.stroke();

// Draw the waveform
ctx.beginPath();
ctx.strokeStyle = '#000000';
ctx.lineWidth = 1;

// Map waveform data to vertical height
for (let i = 0; i < waveformData.length; i++) {
    const y = (i / waveformData.length) * height;
    const amplitude = waveformData[i] * (width * 0.4); // Scale amplitude to
40% of width
    const x = (width / 2) + amplitude;

    if (i === 0) {
        ctx.moveTo(x, y);
    } else {
        ctx.lineTo(x, y);
    }
}
```

```
4686         }
4687     }
4688
4689     // Draw mirrored waveform for visual effect
4690     for (let i = waveformData.length - 1; i >= 0; i--) {
4691         const y = (i / waveformData.length) * height;
4692         const amplitude = waveformData[i] * (width * 0.4);
4693         const x = (width / 2) - amplitude;
4694
4695         ctx.lineTo(x, y);
4696     }
4697
4698     ctx.stroke();
4699
4700     // Draw frame markers
4701     for (let i = 1; i <= totalFrames && i <= totalRows; i++) {
4702         const y = (i / totalRows) * height;
4703
4704         // Draw horizontal line at frame boundary
4705         if (i % 8 === 0) {
4706             ctx.beginPath();
4707             ctx.strokeStyle = '#999999';
4708             ctx.lineWidth = 1;
4709             ctx.moveTo(0, y);
4710             ctx.lineTo(width, y);
4711             ctx.stroke();
4712         }
4713     }
4714
4715     // Render phonetic markers
4716     renderPhoneticMarkers();
4717
4718     // Add the moving marker for current frame
4719     updateFrameMarker();
4720 }
4721
4722 function renderPhoneticMarkers() {
4723     if (!phonetics || !audioBuffer) return;
4724
4725     // Get the waveform container
4726     const waveformContainer = document.querySelector('.waveform-container');
4727     if (!waveformContainer) return;
4728
4729     // Get container dimensions
4730     const containerHeight = waveformContainer.offsetHeight;
4731
4732     // Add phonetic markers
4733     phonetics.forEach(phonic => {
4734         // Calculate vertical position based on time
4735         const percentage = phonic.time / audioBuffer.duration;
```

```
4736     const yPosition = percentage * containerHeight;  
4737  
4738         // Create and position the marker  
4739         const label = document.createElement('div');  
4740         label.className = 'phonetic-label';  
4741         label.textContent = phonetic.text;  
4742         label.style.position = 'absolute';  
4743         label.style.left = '2px';  
4744         label.style.top = yPosition + 'px';  
4745         label.style.zIndex = '25';  
4746  
4747             // Add event listener to edit the phonetic marker  
4748             label.addEventListener('dblclick', function () {  
4749                 showPhoneticInput(phonetic.time, phonetic.text,  
phonetics.indexOf(phonetic));  
4750             });  
4751  
4752                 waveformContainer.appendChild(label);  
4753             });  
4754         }  
4755  
4756     function updateFrameMarker() {  
4757         if (!audioBuffer) return;  
4758  
4759         // Remove existing markers  
4760         const existingMarkers = document.querySelectorAll('.waveform-marker');  
4761         existingMarkers.forEach(marker => marker.remove());  
4762  
4763         // Calculate which frame we're on  
4764         const time = isPlaying ?  
4765             (audioContext.currentTime - startTime + startOffset) :  
4766             startOffset;  
4767  
4768         const frame = Math.floor(time / frameDuration) + 1;  
4769         currentFrame = frame;  
4770  
4771         // Get the waveform container  
4772         const waveformContainer = document.querySelector('.waveform-container');  
4773         if (!waveformContainer) return;  
4774  
4775         // Calculate vertical position based on time  
4776         const containerHeight = waveformContainer.offsetHeight;  
4777         const percentage = time / audioBuffer.duration;  
4778         const yPosition = percentage * containerHeight;  
4779  
4780         // Create horizontal marker line  
4781         const marker = document.createElement('div');  
4782         marker.className = 'waveform-marker';  
4783         marker.style.position = 'absolute';  
4784         marker.style.top = yPosition + 'px';
```

```
4785     marker.style.left = '0';
4786     marker.style.width = '100%';
4787     marker.style.height = '2px';
4788     marker.style.backgroundColor = 'rgba(255, 0, 0, 0.7)';
4789     marker.style.zIndex = '30';
4790
4791     // Add frame number
4792     marker.textContent = `Frame ${frame}`;
4793     marker.style.lineHeight = '0';
4794     marker.style.textAlign = 'right';
4795     marker.style.color = 'red';
4796     marker.style.fontWeight = 'bold';
4797     marker.style.fontSize = '8pt';
4798
4799     waveformContainer.appendChild(marker);
4800
4801     // Scroll to the marker if it's not visible and if we're playing
4802     if (isPlaying) {
4803         const table = document.getElementById('xsheet-table');
4804         const tableRect = table.getBoundingClientRect();
4805         const markerY = tableRect.top + yPosition;
4806
4807         // Check if marker is outside visible area
4808         if (markerY < window.scrollY || markerY > window.scrollY +
4809             window.innerHeight) {
4810             window.scrollTo({
4811                 top: markerY - (window.innerHeight / 2),
4812                 behavior: 'smooth'
4813             });
4814         }
4815
4816         // Update scrubber position
4817         if (audioBuffer) {
4818             const scrubPercentage = (time / audioBuffer.duration) * 100;
4819             audioScrubber.value = scrubPercentage;
4820         }
4821
4822         // If playing, schedule the next update
4823         if (isPlaying) {
4824             requestAnimationFrame(updateFrameMarker);
4825         }
4826     }
4827
4828     function togglePlayAudio() {
4829         if (isPlaying) {
4830             pauseAudio();
4831         } else {
4832             playAudio();
4833         }
4834 }
```

```
4834 }
4835
4836     function playAudio() {
4837         if (!audioBuffer) return;
4838
4839         try {
4840             // Create a new source node
4841             audioSource = audioContext.createBufferSource();
4842             audioSource.buffer = audioBuffer;
4843             audioSource.connect(audioContext.destination);
4844
4845             // Calculate start position
4846             startTime = audioContext.currentTime;
4847
4848             // Start playback from the current offset
4849             audioSource.start(0, startOffset);
4850             isPlaying = true;
4851
4852             // Set up animation loop
4853             updateFrameMarker();
4854
4855             // Update UI
4856             playAudioButton.textContent = 'Pause';
4857
4858             // Set up ended event
4859             audioSource.onended = function () {
4860                 if (isPlaying) {
4861                     stopAudio();
4862                 }
4863             };
4864         } catch (e) {
4865             updateStatusMessage('Error playing audio: ' + e.message);
4866         }
4867     }
4868
4869     function pauseAudio() {
4870         if (!isPlaying || !audioSource) return;
4871
4872         // Stop the audio
4873         audioSource.stop();
4874
4875         // Calculate current position
4876         startOffset += audioContext.currentTime - startTime;
4877
4878         isPlaying = false;
4879
4880         // Update UI
4881         playAudioButton.textContent = 'Play';
4882     }
4883 }
```

```
4884 |     function stopAudio() {
4885 |         if (audioSource) {
4886 |             try {
4887 |                 audioSource.stop();
4888 |             } catch (e) {
4889 |                 // Ignore errors when stopping already stopped source
4890 |             }
4891 |         }
4892 |
4893 |         isPlaying = false;
4894 |         startOffset = 0;
4895 |
4896 |         // Update UI
4897 |         playAudioButton.textContent = 'Play';
4898 |         audioScrubber.value = 0;
4899 |
4900 |         // Clear frame marker
4901 |         updateFrameMarker();
4902 |
4903 |
4904 |     function scrubAudio() {
4905 |         if (!audioBuffer) return;
4906 |
4907 |         // Calculate time from scrubber position
4908 |         const percentage = audioScrubber.value / 100;
4909 |         const newTime = percentage * audioBuffer.duration;
4910 |
4911 |         // If playing, restart from new position
4912 |         if (isPlaying) {
4913 |             pauseAudio();
4914 |             startOffset = newTime;
4915 |             playAudio();
4916 |         } else {
4917 |             startOffset = newTime;
4918 |             updateFrameMarker();
4919 |         }
4920 |
4921 |
4922 |     function showPhoneticInput(time, initialText = ' ', editIndex = -1) {
4923 |         if (time === null) {
4924 |             // If no time provided, use current position
4925 |             time = isPlaying ?
4926 |                 (audioContext.currentTime - startTime + startOffset) :
4927 |                 startOffset;
4928 |         }
4929 |
4930 |         // Store position for later use
4931 |         phoneticEditPosition = {
4932 |             time: time,
4933 |             editIndex: editIndex
```

```
4934     };
4935
4936     // Set initial text if editing existing marker
4937     phoneticText.value = initialText;
4938
4939     // Get the waveform container and calculate position
4940     const waveformContainer = document.querySelector('.waveform-container');
4941     if (waveformContainer) {
4942         const containerRect = waveformContainer.getBoundingClientRect();
4943         const containerHeight = waveformContainer.offsetHeight;
4944         const percentage = time / audioBuffer.duration;
4945         const yPosition = percentage * containerHeight;
4946
4947         // Position the input near the click position
4948         phoneticInput.style.top = (containerRect.top + yPosition +
4949             window.scrollY) + 'px';
4950         phoneticInput.style.left = (containerRect.right + window.scrollX + 5) +
4951             'px';
4952     } else {
4953         // Default position if container not found
4954         phoneticInput.style.top = '200px';
4955         phoneticInput.style.left = '200px';
4956     }
4957
4958     // Show the input and focus it
4959     phoneticInput.style.display = 'block';
4960     phoneticText.focus();
4961 }
4962
4963 function savePhoneticMarker() {
4964     if (!phoneticEditPosition) return;
4965
4966     const text = phoneticText.value.trim();
4967     if (text === '') {
4968         // If empty text and editing an existing marker, remove it
4969         if (phoneticEditPosition.editIndex >= 0) {
4970             phonetics.splice(phoneticEditPosition.editIndex, 1);
4971         }
4972     } else {
4973         // Save or update the phonetic marker
4974         if (phoneticEditPosition.editIndex >= 0) {
4975             // Update existing
4976             phonetics[phoneticEditPosition.editIndex].text = text;
4977         } else {
4978             // Add new
4979             phonetics.push({
4980                 time: phoneticEditPosition.time,
4981                 text: text
4982             });
4983         }
4984     }
4985 }
```

```
4982     }
4983
4984     // Hide input
4985     phoneticInput.style.display = 'none';
4986
4987     // Re-render markers
4988     renderWaveform();
4989
4990     // Mark as modified
4991     modified = true;
4992 }
4993
4994 function collectData() {
4995     const data = {
4996         template: currentTemplate,
4997         frameCount: frameCount,
4998         metadata: {
4999             projectNumber: document.getElementById('project-number').value,
5000             date: document.getElementById('project-date').value,
5001             pageNumber: document.getElementById('page-number').value,
5002             animatorName: document.getElementById('animator-name').value,
5003             versionNumber: document.getElementById('version-number').value,
5004             shotNumber: document.getElementById('shot-number').value
5005         },
5006         audio: {
5007             fileName: audioFileName,
5008             phonetics: phonetics
5009         },
5010         rows: []
5011     };
5012
5013     // Collect data from all rows
5014     const rows = tableBody.querySelectorAll('tr');
5015     rows.forEach(row => {
5016         const rowData = {
5017             action: row.cells[0].innerText,
5018             frame: row.cells[1].innerText,
5019             // Skip waveform cell (2)
5020             dialogue: row.cells[3].innerText,
5021             sound: row.cells[4].innerText,
5022             technical: row.cells[5].innerText,
5023             extra1: row.cells[6].innerText,
5024             extra2: row.cells[7].innerText,
5025             frameRepeat: row.cells[8].innerText,
5026             camera: row.cells[9].innerText
5027         };
5028         data.rows.push(rowData);
5029     });
5030
5031     return data;
}
```

```
5032     }
5033
5034     function restoreData(data) {
5035         if (!data) return;
5036
5037         // Update template and frame count
5038         currentTemplate = data.template || 'large';
5039         frameCount = data.frameCount || 96;
5040
5041         // Update UI to match
5042         templateSelector.value = currentTemplate;
5043         frameCountInput.value = frameCount;
5044
5045         // Restore metadata
5046         if (data.metadata) {
5047             document.getElementById('project-number').value =
5048             data.metadata.projectNumber || '';
5049             document.getElementById('project-date').value = data.metadata.date ||
5050             '';
5051             document.getElementById('page-number').value = data.metadata.pageNumber
5052             || '';
5053             document.getElementById('animator-name').value =
5054             data.metadata.animatorName || '';
5055             document.getElementById('version-number').value =
5056             data.metadata.versionNumber || '';
5057             document.getElementById('shot-number').value = data.metadata.shotNumber
5058             || '';
5059         }
5060
5061         // Generate the table with the right frame count
5062         generateTable(frameCount);
5063         updateTemplate();
5064
5065         // Restore audio data
5066         if (data.audio) {
5067             audioFileName = data.audio.fileName || '';
5068             phonetics = data.audio.phonetics || [];
5069
5070             if (audioBuffer && phonetics.length > 0) {
5071                 renderWaveform();
5072             }
5073         }
5074
5075         // Restore row data
5076         if (data.rows && data.rows.length > 0) {
5077             const rows = tableBody.querySelectorAll('tr');
5078             data.rows.forEach((rowData, index) => {
5079                 if (index < rows.length) {
5080                     rows[index].cells[0].innerText = rowData.action || '';
5081                     // Don't restore frame number cells as they're auto-generated
5082                     rows[index].cells[3].innerText = rowData.dialogue || '';
5083                 }
5084             });
5085         }
5086     }
5087 }
```

```
5077     rows[index].cells[4].innerText = rowData.sound || '';
5078     rows[index].cells[5].innerText = rowData.technical || '';
5079     rows[index].cells[6].innerText = rowData.extra1 || '';
5080     rows[index].cells[7].innerText = rowData.extra2 || '';
5081     // Don't restore the second frame number cell
5082     rows[index].cells[9].innerText = rowData.camera || '';
5083   }
5084 }
5085
5086
5087 modified = false;
5088 updateStatusMessage('Project loaded successfully');
5089 }
5090
5091 function saveProject() {
5092   const data = collectData();
5093
5094   // Save to localStorage
5095   try {
5096     localStorage.setItem('animationXSheet', JSON.stringify(data));
5097     modified = false;
5098     updateStatusMessage('Project saved successfully');
5099
5100     // Also download as JSON file for backup
5101     const filename = projectName + '.json';
5102     const dataStr = "data:text/json;charset=utf-8," +
5103 encodeURIComponent(JSON.stringify(data));
5104     const downloadAnchorNode = document.createElement('a');
5105     downloadAnchorNode.setAttribute("href", dataStr);
5106     downloadAnchorNode.setAttribute("download", filename);
5107     document.body.appendChild(downloadAnchorNode);
5108     downloadAnchorNode.click();
5109     downloadAnchorNode.remove();
5110   } catch (e) {
5111     updateStatusMessage('Error saving project: ' + e.message);
5112   }
5113
5114 function loadProject() {
5115   // First try to load from localStorage
5116   try {
5117     const savedData = localStorage.getItem('animationXSheet');
5118     if (savedData) {
5119       restoreData(JSON.parse(savedData));
5120       return;
5121     }
5122   } catch (e) {
5123     updateStatusMessage('Error loading saved project: ' + e.message);
5124   }
5125 }
```

```
5126 // If no localStorage data, create file input for uploading JSON
5127 const fileInput = document.createElement('input');
5128 fileInput.type = 'file';
5129 fileInput.accept = '.json';
5130 fileInput.style.display = 'none';
5131 document.body.appendChild(fileInput);
5132
5133 fileInput.addEventListener('change', function (e) {
5134     const file = e.target.files[0];
5135     if (!file) return;
5136
5137     const reader = new FileReader();
5138     reader.onload = function (e) {
5139         try {
5140             const data = JSON.parse(e.target.result);
5141             restoreData(data);
5142         } catch (error) {
5143             updateStatusMessage('Error parsing file: ' + error.message);
5144         }
5145     };
5146     reader.readAsText(file);
5147 });
5148
5149 fileInput.click();
5150 fileInput.remove();
5151 }
5152
5153 function exportToPDF() {
5154     // Save the current state before PDF export
5155     const savedData = collectData();
5156
5157     updateStatusMessage('Preparing PDF. Please wait...');

5159     // Remove controls temporarily for PDF generation
5160     const controls = document.querySelector('.controls');
5161     const audioControls = document.querySelector('#audio-controls');
5162     const statusMsg = document.querySelector('.status');
5163     const phoneticInputEl = document.querySelector('#phonetic-input');

5165     controls.style.display = 'none';
5166     audioControls.style.display = 'none';
5167     statusMsg.style.display = 'none';
5168     phoneticInputEl.style.display = 'none';

5169
5170     // Clean up any previous waveform elements
5171     const previousContainers = document.querySelectorAll('.cell-waveform-
5172 window');
5173     previousContainers.forEach(container => container.remove());
5174
5175     // Draw the waveform directly into the cells
```

```
5175     if (audioBuffer && waveformData.length > 0) {
5176         drawWaveformInCells();
5177     }
5178
5179     // Wait a moment for DOM updates to complete
5180     setTimeout(() => {
5181         try {
5182             // Force a redraw of the drawing layer before capture
5183             if (window.xsheetDrawing && window.xsheetDrawing.layerSystem) {
5184                 window.xsheetDrawing.layerSystem.redrawAll();
5185             }
5186
5187             // Use html2canvas to capture the printable area
5188             html2canvas(document.getElementById('printable-area'), {
5189                 scale: 2, // Higher resolution
5190                 useCORS: true,
5191                 logging: false,
5192                 allowTaint: true,
5193                 onclone: function (clonedDoc) {
5194                     // Ensure drawing elements are visible in the cloned
5195                     document
5196                     const clonedDrawingLayer =
5197                     clonedDoc.querySelector('.drawing-layer-container');
5198                     if (clonedDrawingLayer) {
5199                         clonedDrawingLayer.style.display = 'block';
5200                         clonedDrawingLayer.style.visibility = 'visible';
5201                         clonedDrawingLayer.style.opacity = '1';
5202                     }
5203                 }).then(canvas => {
5204                     const imgData = canvas.toDataURL('image/png');
5205
5206                     // Determine PDF size based on template
5207                     let pdfWidth, pdfHeight;
5208                     if (currentTemplate === 'large') {
5209                         // 11"x17"
5210                         pdfWidth = 279.4; // mm
5211                         pdfHeight = 431.8; // mm
5212                     } else {
5213                         // 8.5"x11"
5214                         pdfWidth = 215.9; // mm
5215                         pdfHeight = 279.4; // mm
5216                     }
5217
5218                     // Create PDF with jsPDF
5219                     const { jsPDF } = window.jspdf;
5220                     const pdf = new jsPDF({
5221                         orientation: 'portrait',
5222                         unit: 'mm',
5223                         format: [pdfWidth, pdfHeight]
```

```
5223     });
5224
5225     // Calculate aspect ratio
5226     const imgWidth = pdfWidth - 20; // margins
5227     const imgHeight = canvas.height * imgWidth / canvas.width;
5228
5229     // Add image to PDF
5230     pdf.addImage(imgData, 'PNG', 10, 10, imgWidth, imgHeight);
5231
5232     // Save PDF
5233     pdf.save(`.${projectName}.pdf`);
5234
5235     // Clean up after PDF generation using saved data instead of
rebuiding
5236     cleanupAfterExport(savedData);
5237   });
5238 } catch (e) {
5239   // Clean up if there was an error
5240   cleanupAfterExport(savedData);
5241   updateStatusMessage('Error exporting PDF: ' + e.message);
5242 }
5243 }, 100);
5244
5245 function cleanupAfterExport(savedData) {
5246   // Restore controls
5247   controls.style.display = 'flex';
5248   audioControls.style.display = 'flex';
5249   statusMsg.style.display = 'block';
5250
5251   // Restore data instead of regenerating blank table
5252   restoreData(savedData);
5253
5254   updateStatusMessage('PDF exported successfully');
5255 }
5256
5257 function drawWaveformInCells() {
5258   const waveformCells = document.querySelectorAll('.waveform-col');
5259   if (waveformCells.length === 0) return;
5260
5261   // Calculate the number of data points per frame
5262   const totalDuration = audioBuffer.duration;
5263   const pointsPerFrame = waveformData.length / (totalDuration * 24);
5264
5265   // For each cell, draw its portion of the waveform
5266   waveformCells.forEach((cell, index) => {
5267     // Get the frame number (1-based)
5268     const frameNum = index + 1;
5269
5270     // Create a canvas for this cell
5271     const canvas = document.createElement('canvas');
```

```
5272     canvas.width = cell.offsetWidth;
5273     canvas.height = cell.offsetHeight;
5274     canvas.style.display = 'block';
5275     canvas.style.width = '100%';
5276     canvas.style.height = '100%';

5277

5278     const ctx = canvas.getContext('2d');

5279

5280     // Fill with white background
5281     ctx.fillStyle = 'white';
5282     ctx.fillRect(0, 0, canvas.width, canvas.height);

5283

5284     // Draw center line
5285     ctx.beginPath();
5286     ctx.strokeStyle = '#cccccc';
5287     ctx.moveTo(canvas.width / 2, 0);
5288     ctx.lineTo(canvas.width / 2, canvas.height);
5289     ctx.stroke();

5290

5291     // Calculate which section of the waveform to draw
5292     const startPoint = Math.floor((frameNum - 1) * pointsPerFrame);
5293     const endPoint = Math.floor(frameNum * pointsPerFrame);

5294

5295     if (startPoint < waveformData.length) {
5296         // Draw this portion of the waveform
5297         ctx.beginPath();
5298         ctx.strokeStyle = '#000000';

5299

5300         // Check if we have valid data to draw
5301         if (endPoint > startPoint) {
5302             // Map the waveform section to this cell
5303             for (let i = startPoint; i <= endPoint && i <
waveformData.length; i++) {
5304                 // Calculate position within this cell
5305                 const relativePos = (i - startPoint) / (endPoint -
startPoint);
5306                 const y = relativePos * canvas.height;

5307

5308                 // Draw waveform
5309                 const amplitude = waveformData[i] * (canvas.width *
0.4);
5310                 const x = (canvas.width / 2) + amplitude;

5311

5312                 if (i === startPoint) {
5313                     ctx.moveTo(x, y);
5314                 } else {
5315                     ctx.lineTo(x, y);
5316                 }
5317             }

5318             // Draw left side (mirror)
```

```

5320    waveformData.length; i--) {
5321        const relativePos = (i - startPoint) / (endPoint -
5322        startPoint);
5323        const y = relativePos * canvas.height;
5324
5325        const amplitude = waveformData[i] * (canvas.width *
5326        0.4);
5327        const x = (canvas.width / 2) - amplitude;
5328        ctx.lineTo(x, y);
5329    }
5330    ctx.stroke();
5331}
5332
5333 // Look for phonetic markers that might be in this frame
5334 if (phonetics && phonetics.length > 0) {
5335     phonetics.forEach(phonic => {
5336         // Calculate which frame this phonetic marker belongs to
5337         const frameOfMarker = Math.floor(phonic.time /
frameDuration) + 1;
5338
5339         // If it's in this frame, draw it
5340         if (frameOfMarker === frameNum) {
5341             // Calculate position within cell
5342             const posInFrameRatio = (phonic.time - ((frameNum
- 1) * frameDuration)) / frameDuration;
5343             const yPos = posInFrameRatio * canvas.height;
5344
5345             // Draw marker line
5346             ctx.beginPath();
5347             ctx.strokeStyle = '#ff0000';
5348             ctx.lineWidth = 1;
5349             ctx.moveTo(0, yPos);
5350             ctx.lineTo(canvas.width, yPos);
5351             ctx.stroke();
5352
5353             // Add label if it fits
5354             if (canvas.width > 30) {
5355                 ctx.fillStyle = 'rgba(255, 255, 255, 0.7)';
5356                 const textWidth =
5357                 ctx.measureText(phonic.text).width;
5358
5359                 if (textWidth < canvas.width - 4) {
5360                     ctx.fillRect(2, yPos - 8, textWidth + 4,
5361
5362                     14);
5363                     ctx.fillStyle = '#ff0000';
5364                     ctx.font = '8px Arial';
5365                     ctx.fillText(phonic.text, 4, yPos + 4);
5366                 }
5367             }
5368         }
5369     }
5370 }

```

```
5364 }  
5365     });  
5366     }  
5367 }  
5368  
5369     // Add frame number indicator  
5370     ctx.fillStyle = 'rgba(255, 255, 255, 0.7)';  
5371     ctx.fillRect(0, 0, 18, 12);  
5372     ctx.fillStyle = '#000000';  
5373     ctx.font = '8px Arial';  
5374     ctx.fillText(frameNum, 2, 8);  
5375  
5376     // Clear cell content and add the canvas  
5377     cell.innerHTML = '';  
5378     cell.appendChild(canvas);  
5379 };  
5380 }  
5381 }  
5382  
5383 function printSheet() {  
5384     updateStatusMessage('Preparing to print. Please wait...');  
5385  
5386     // Save current selection state and cell contents  
5387     const tableData = saveSelectionState();  
5388  
5389     // Hide the waveform container temporarily but don't remove it  
5390     const originalWf = document.querySelector('.waveform-container');  
5391     if (originalWf) {  
5392         originalWf.style.display = 'none';  
5393     }  
5394  
5395     // Draw the waveform directly into the cells before printing  
5396     if (audioBuffer && waveformData.length > 0) {  
5397         drawWaveformInCells();  
5398     }  
5399  
5400     // Wait a moment for DOM updates to complete  
5401     setTimeout(() => {  
5402         window.print();  
5403  
5404         // Clean up after printing  
5405         setTimeout(() => {  
5406             // Restore the original waveform container  
5407             if (originalWf) {  
5408                 originalWf.style.display = '';  
5409             }  
5410  
5411             // Restore original cell content for waveform cells only  
5412             const waveformCells = document.querySelectorAll('.waveform-col');  
5413             waveformCells.forEach(cell => {
```

```
5414     const originalContent = cell.getAttribute('data-original-  
5415         content');  
5416  
5417         if (originalContent !== null) {  
5418             cell.innerHTML = originalContent;  
5419             cell.removeAttribute('data-original-content');  
5420         }  
5421     });  
5422  
5423         // Restore all cell contents from saved data  
5424         if (tableData && tableData.cellContents) {  
5425             tableData.cellContents.forEach(cellData => {  
5426                 if (cellData.rowIndex >= 0 && cellData.rowIndex <  
5427                     tableBody.children.length) {  
5428                     const row = tableBody.children[cellData.rowIndex];  
5429                     if (cellData.colIndex >= 0 && cellData.colIndex <  
5430                         row.children.length) {  
5431                         const cell = row.children[cellData.colIndex];  
5432  
5433                         // Skip waveform cells (they were already restored  
5434                         above)  
5435                         if (!cell.classList.contains('waveform-col')) {  
5436                             // Only restore if cell is editable  
5437                             if (cell.contentEditable === 'true') {  
5438                                 cell.innerHTML = cellData.content;  
5439  
5440                                 // Restore modified status  
5441                                 if (cellData.isModified) {  
5442                                     cell.classList.add('modified');  
5443                                 } else {  
5444                                     cell.classList.remove('modified');  
5445                                 }  
5446                             }  
5447                         }  
5448                     }  
5449                 });  
5450             }  
5451  
5452             // Restore metadata fields  
5453             if (tableData && tableData.metadata) {  
5454                 document.getElementById('project-number').value =  
tableData.metadata.projectNumber || '';  
5455                 document.getElementById('project-date').value =  
tableData.metadata.date || '';  
5456                 document.getElementById('page-number').value =  
tableData.metadata.pageNumber || '';  
5457                 document.getElementById('animator-name').value =  
tableData.metadata.animatorName || '';  
5458                 document.getElementById('version-number').value =  
tableData.metadata.versionNumber || '';
```

```
5455     document.getElementById('shot-number').value =
5456     tableData.metadata.shotNumber || '';
5457
5458     // Finally restore cell selection state
5459     restoreSelectionState(tableData);
5460
5461     updateStatusMessage('Print complete');
5462 }, 500);
5463 }, 100);
5464
5465 function drawWaveformInCells() {
5466     const waveformCells = document.querySelectorAll('.waveform-col');
5467     if (waveformCells.length === 0) return;
5468
5469     // Calculate the number of data points per frame
5470     const totalDuration = audioBuffer.duration;
5471     const pointsPerFrame = waveformData.length / (totalDuration * 24);
5472
5473     // For each cell, draw its portion of the waveform
5474     waveformCells.forEach((cell, index) => {
5475         // Get the frame number (1-based)
5476         const frameNum = index + 1;
5477
5478         // Create a canvas for this cell
5479         const canvas = document.createElement('canvas');
5480         canvas.width = cell.offsetWidth;
5481         canvas.height = cell.offsetHeight;
5482         canvas.style.display = 'block';
5483         canvas.style.width = '100%';
5484         canvas.style.height = '100%';
5485
5486         const ctx = canvas.getContext('2d');
5487
5488         // Fill with white background
5489         ctx.fillStyle = 'white';
5490         ctx.fillRect(0, 0, canvas.width, canvas.height);
5491
5492         // Draw center line
5493         ctx.beginPath();
5494         ctx.strokeStyle = '#cccccc';
5495         ctx.moveTo(canvas.width / 2, 0);
5496         ctx.lineTo(canvas.width / 2, canvas.height);
5497         ctx.stroke();
5498
5499         // Calculate which section of the waveform to draw
5500         const startPoint = Math.floor((frameNum - 1) * pointsPerFrame);
5501         const endPoint = Math.floor(frameNum * pointsPerFrame);
5502
5503         if (startPoint < waveformData.length) {
```

```
5504 // Draw this portion of the waveform
5505 ctx.beginPath();
5506 ctx.strokeStyle = '#000000';
5507
5508 // Check if we have valid data to draw
5509 if (endPoint > startPoint) {
5510     // Map the waveform section to this cell
5511     for (let i = startPoint; i <= endPoint && i <
5512         waveformData.length; i++) {
5513         // Calculate position within this cell
5514         const relativePos = (i - startPoint) / (endPoint -
5515         startPoint);
5516         const y = relativePos * canvas.height;
5517
5518         // Draw waveform
5519         const amplitude = waveformData[i] * (canvas.width *
5520         0.4);
5521         const x = (canvas.width / 2) + amplitude;
5522
5523         if (i === startPoint) {
5524             ctx.moveTo(x, y);
5525         } else {
5526             ctx.lineTo(x, y);
5527         }
5528
5529         // Draw left side (mirror)
5530         for (let i = endPoint; i >= startPoint && i <
5531             waveformData.length; i--) {
5532             const relativePos = (i - startPoint) / (endPoint -
5533             startPoint);
5534             const y = relativePos * canvas.height;
5535
5536             const amplitude = waveformData[i] * (canvas.width *
5537             0.4);
5538             const x = (canvas.width / 2) - amplitude;
5539
5540             ctx.lineTo(x, y);
5541
5542             ctx.stroke();
5543         }
5544
5545         // Look for phonetic markers that might be in this frame
5546         if (phonetics && phonetics.length > 0) {
5547             phonetics.forEach(phnetic => {
5548                 // Calculate which frame this phonetic marker belongs to
5549                 const frameOfMarker = Math.floor(phnetic.time /
5550                 frameDuration) + 1;
5551
5552                 // If it's in this frame, draw it
5553             }
5554         }
5555     }
5556 }
5557
5558
5559
5560
5561
5562
5563
5564
5565
5566
5567
5568
5569
5570
5571
5572
5573
5574
5575
5576
5577
5578
5579
5580
5581
5582
5583
5584
5585
5586
5587
5588
5589
5590
5591
5592
5593
5594
5595
5596
5597
```

```

5548         if (frameOfMarker === frameNum) {
5549             // Calculate position within cell
5550             const posInFrameRatio = (phonetic.time - ((frameNum
5551 - 1) * frameDuration)) / frameDuration;
5552             const yPos = posInFrameRatio * canvas.height;
5553
5554             // Draw marker line
5555             ctx.beginPath();
5556             ctx.strokeStyle = '#ff0000';
5557             ctx.lineWidth = 1;
5558             ctx.moveTo(0, yPos);
5559             ctx.lineTo(canvas.width, yPos);
5560             ctx.stroke();
5561
5562             // Add label if it fits
5563             if (canvas.width > 30) {
5564                 ctx.fillStyle = 'rgba(255, 255, 255, 0.7)';
5565                 const textWidth =
5566                 ctx.measureText(phnetic.text).width;
5567
5568                 if (textWidth < canvas.width - 4) {
5569                     ctx.fillRect(2, yPos - 8, textWidth + 4,
5570
5571                     14);
5572                     ctx.fillStyle = '#ff0000';
5573                     ctx.font = '8px Arial';
5574                     ctx.fillText(phnetic.text, 4, yPos + 4);
5575                 }
5576             }
5577
5578             // Add frame number indicator
5579             ctx.fillStyle = 'rgba(255, 255, 255, 0.7)';
5580             ctx.fillRect(0, 0, 18, 12);
5581             ctx.fillStyle = '#000000';
5582             ctx.font = '8px Arial';
5583             ctx.fillText(frameNum, 2, 8);
5584
5585             // Store original content and add the canvas
5586             cell.setAttribute('data-original-content', cell.innerHTML);
5587             cell.innerHTML = '';
5588             cell.appendChild(canvas);
5589         });
5590
5591     function saveSelectionState() {
5592         // Save current selection and cell contents
5593         const tableData = {
5594             // Save which cells are selected
5595             selectedIndices: selectedCells.map(cell => {

```

```
5596     const row = cell.closest('tr');
5597     const rowIndex = Array.from(tableBody.children).indexOf(row);
5598     const colIndex = Array.from(row.children).indexOf(cell);
5599     return { rowIndex, colIndex };
5600   },
5601   // Save metadata fields
5602   metadata: {
5603     projectNumber: document.getElementById('project-number').value,
5604     date: document.getElementById('project-date').value,
5605     pageNumber: document.getElementById('page-number').value,
5606     animatorName: document.getElementById('animator-name').value,
5607     versionNumber: document.getElementById('version-number').value,
5608     shotNumber: document.getElementById('shot-number').value
5609   },
5610   // Store editable cells' content
5611   cellContents: []
5612 };
5613
5614   // Store all editable cells' content
5615   const allEditableCells =
5616     document.querySelectorAll('[contenteditable="true"]');
5617     allEditableCells.forEach(cell => {
5618       const row = cell.closest('tr');
5619       if (row) {
5620         const rowIndex = Array.from(tableBody.children).indexOf(row);
5621         const colIndex = Array.from(row.children).indexOf(cell);
5622         tableData.cellContents.push({
5623           rowIndex,
5624           colIndex,
5625           content: cell.innerHTML,
5626           isModified: cell.classList.contains('modified')
5627         });
5628       }
5629     });
5630
5631   return tableData;
5632 }
5633
5634   function restoreSelectionState(tableData) {
5635     if (!tableData || !tableData.selectedIndices) return;
5636
5637     // Clear current selection
5638     clearCellSelection();
5639
5640     // Restore selection
5641     tableData.selectedIndices.forEach(index => {
5642       if (index.rowIndex >= 0 && index.rowIndex <
5643         tableBody.children.length) {
5644         const row = tableBody.children[index.rowIndex];
5645         if (index.colIndex >= 0 && index.colIndex < row.children.length)
5646       {

```

```
5644     const cell = row.children[index.colIndex];
5645     if (cell.contentEditable === 'true') {
5646         toggleCellSelection(cell, true);
5647     }
5648 }
5649 }
5650 });
5651 }
5652 }
5653
5654 function addEightRows() {
5655     frameCount += 8;
5656     frameCountInput.value = frameCount;
5657     generateTable(frameCount);
5658     updateStatusMessage('Added 8 rows. Total frames: ' + frameCount);
5659
5660     // Re-render waveform if audio is loaded
5661     if (audioBuffer) {
5662         renderWaveform();
5663     }
5664 }
5665
5666 function clearSheet() {
5667     if (confirm('Are you sure you want to clear all data? This cannot be undone.')) {
5668         // Clear metadata
5669         document.getElementById('project-number').value = '';
5670         document.getElementById('page-number').value = '';
5671         document.getElementById('animator-name').value = '';
5672         document.getElementById('version-number').value = '';
5673         document.getElementById('shot-number').value = '';
5674
5675         // Reset date to today
5676         document.getElementById('project-date').valueAsDate = new Date();
5677
5678         // Clear all editable cells
5679         const editableCells =
5680             document.querySelectorAll('[contenteditable="true"]');
5681         editableCells.forEach(cell => {
5682             cell.innerText = '';
5683             cell.classList.remove('modified');
5684         });
5685
5686         // Clear audio
5687         audioBuffer = null;
5688         audioSource = null;
5689         waveformData = [];
5690         phonetics = [];
5691         audioFileName = '';
5692         audioInfo.textContent = 'No audio loaded';
5693     }
5694 }
```

```
5692
5693         // Stop any playing audio
5694         stopAudio();
5695
5696         // Clear waveform visualization
5697         waveformCanvases.forEach(canvas => {
5698             const ctx = canvas.getContext('2d');
5699             ctx.clearRect(0, 0, canvas.width, canvas.height);
5700         });
5701
5702         // Clear phonetic markers
5703         const labels = document.querySelectorAll('.phonetic-label');
5704         labels.forEach(label => label.remove());
5705
5706         // Clear drawings
5707         if (window.xsheetDrawing && window.xsheetDrawing.layerSystem) {
5708             window.xsheetDrawing.layerSystem.clearAllLayers();
5709         }
5710
5711         modified = false;
5712         updateStatusMessage('Sheet cleared');
5713     }
5714 }
5715
5716 function updateStatusMessage(message) {
5717     statusMessage.textContent = message;
5718     console.log(message);
5719
5720     // Clear status message after 3 seconds
5721     setTimeout(() => {
5722         if (statusMessage.textContent === message) {
5723             if (modified) {
5724                 statusMessage.textContent = 'Unsaved changes';
5725             } else {
5726                 statusMessage.textContent = '';
5727             }
5728         }
5729     }, 3000);
5730 }
5731
5732 // Auto-save timer every 2 minutes
5733 setInterval(() => {
5734     if (modified) {
5735         try {
5736             const data = collectData();
5737             localStorage.setItem('animationXSheet_autosave',
5738                 JSON.stringify(data));
5739             updateStatusMessage('Auto-saved');
5740         } catch (e) {
5741             console.error('Auto-save failed:', e);
5742         }
5743     }
5744 });
5745
5746
5747
5748
5749
5750
5751
5752
5753
5754
5755
5756
5757
5758
5759
5760
5761
5762
5763
5764
5765
5766
5767
5768
5769
5770
5771
5772
5773
5774
5775
5776
5777
5778
5779
5780
5781
5782
5783
5784
5785
5786
5787
5788
5789
5790
5791
5792
5793
5794
5795
5796
5797
5798
5799
5800
5801
5802
5803
5804
5805
5806
5807
5808
5809
5810
5811
5812
5813
5814
5815
5816
5817
5818
5819
5820
5821
5822
5823
5824
5825
5826
5827
5828
5829
5830
5831
5832
5833
5834
5835
5836
5837
5838
5839
5840
5841
5842
5843
5844
5845
5846
5847
5848
5849
5850
5851
5852
5853
5854
5855
5856
5857
5858
5859
5860
5861
5862
5863
5864
5865
5866
5867
5868
5869
5870
5871
5872
5873
5874
5875
5876
5877
5878
5879
5880
5881
5882
5883
5884
5885
5886
5887
5888
5889
5890
5891
5892
5893
5894
5895
5896
5897
5898
5899
5900
5901
5902
5903
5904
5905
5906
5907
5908
5909
5910
5911
5912
5913
5914
5915
5916
5917
5918
5919
5920
5921
5922
5923
5924
5925
5926
5927
5928
5929
5930
5931
5932
5933
5934
5935
5936
5937
5938
5939
5940
5941
5942
5943
5944
5945
5946
5947
5948
5949
5950
5951
5952
5953
5954
5955
5956
5957
5958
5959
5960
5961
5962
5963
5964
5965
5966
5967
5968
5969
5970
5971
5972
5973
5974
5975
5976
5977
5978
5979
5980
5981
5982
5983
5984
5985
5986
5987
5988
5989
5990
5991
5992
5993
5994
5995
5996
5997
5998
5999
5999
6000
6001
6002
6003
6004
6005
6006
6007
6008
6009
6009
6010
6011
6012
6013
6014
6015
6016
6017
6018
6019
6019
6020
6021
6022
6023
6024
6025
6026
6027
6028
6029
6029
6030
6031
6032
6033
6034
6035
6036
6037
6038
6039
6039
6040
6041
6042
6043
6044
6045
6046
6047
6048
6049
6049
6050
6051
6052
6053
6054
6055
6056
6057
6058
6059
6059
6060
6061
6062
6063
6064
6065
6066
6067
6068
6069
6069
6070
6071
6072
6073
6074
6075
6076
6077
6078
6079
6079
6080
6081
6082
6083
6084
6085
6086
6087
6088
6089
6089
6090
6091
6092
6093
6094
6095
6096
6097
6098
6099
6099
6100
6101
6102
6103
6104
6105
6106
6107
6108
6109
6109
6110
6111
6112
6113
6114
6115
6116
6117
6118
6119
6119
6120
6121
6122
6123
6124
6125
6126
6127
6128
6129
6129
6130
6131
6132
6133
6134
6135
6136
6137
6138
6139
6139
6140
6141
6142
6143
6144
6145
6146
6147
6148
6149
6149
6150
6151
6152
6153
6154
6155
6156
6157
6158
6159
6159
6160
6161
6162
6163
6164
6165
6166
6167
6168
6169
6169
6170
6171
6172
6173
6174
6175
6176
6177
6178
6179
6179
6180
6181
6182
6183
6184
6185
6186
6187
6188
6189
6189
6190
6191
6192
6193
6194
6195
6196
6197
6198
6199
6199
6200
6201
6202
6203
6204
6205
6206
6207
6208
6209
6209
6210
6211
6212
6213
6214
6215
6216
6217
6218
6219
6219
6220
6221
6222
6223
6224
6225
6226
6227
6228
6229
6229
6230
6231
6232
6233
6234
6235
6236
6237
6238
6239
6239
6240
6241
6242
6243
6244
6245
6246
6247
6248
6249
6249
6250
6251
6252
6253
6254
6255
6256
6257
6258
6259
6259
6260
6261
6262
6263
6264
6265
6266
6267
6268
6269
6269
6270
6271
6272
6273
6274
6275
6276
6277
6278
6279
6279
6280
6281
6282
6283
6284
6285
6286
6287
6288
6289
6289
6290
6291
6292
6293
6294
6295
6296
6297
6298
6299
6299
6300
6301
6302
6303
6304
6305
6306
6307
6308
6309
6309
6310
6311
6312
6313
6314
6315
6316
6317
6318
6319
6319
6320
6321
6322
6323
6324
6325
6326
6327
6328
6329
6329
6330
6331
6332
6333
6334
6335
6336
6337
6338
6339
6339
6340
6341
6342
6343
6344
6345
6346
6347
6348
6349
6349
6350
6351
6352
6353
6354
6355
6356
6357
6358
6359
6359
6360
6361
6362
6363
6364
6365
6366
6367
6368
6369
6369
6370
6371
6372
6373
6374
6375
6376
6377
6378
6379
6379
6380
6381
6382
6383
6384
6385
6386
6387
6388
6389
6389
6390
6391
6392
6393
6394
6395
6396
6397
6398
6399
6399
6400
6401
6402
6403
6404
6405
6406
6407
6408
6409
6409
6410
6411
6412
6413
6414
6415
6416
6417
6418
6419
6419
6420
6421
6422
6423
6424
6425
6426
6427
6428
6429
6429
6430
6431
6432
6433
6434
6435
6436
6437
6438
6439
6439
6440
6441
6442
6443
6444
6445
6446
6447
6448
6449
6449
6450
6451
6452
6453
6454
6455
6456
6457
6458
6459
6459
6460
6461
6462
6463
6464
6465
6466
6467
6468
6469
6469
6470
6471
6472
6473
6474
6475
6476
6477
6478
6479
6479
6480
6481
6482
6483
6484
6485
6486
6487
6488
6489
6489
6490
6491
6492
6493
6494
6495
6496
6497
6498
6499
6499
6500
6501
6502
6503
6504
6505
6506
6507
6508
6509
6509
6510
6511
6512
6513
6514
6515
6516
6517
6518
6519
6519
6520
6521
6522
6523
6524
6525
6526
6527
6528
6529
6529
6530
6531
6532
6533
6534
6535
6536
6537
6538
6539
6539
6540
6541
6542
6543
6544
6545
6546
6547
6548
6549
6549
6550
6551
6552
6553
6554
6555
6556
6557
6558
6559
6559
6560
6561
6562
6563
6564
6565
6566
6567
6568
6569
6569
6570
6571
6572
6573
6574
6575
6576
6577
6578
6579
6579
6580
6581
6582
6583
6584
6585
6586
6587
6588
6589
6589
6590
6591
6592
6593
6594
6595
6596
6597
6598
6599
6599
6600
6601
6602
6603
6604
6605
6606
6607
6608
6609
6609
6610
6611
6612
6613
6614
6615
6616
6617
6618
6619
6619
6620
6621
6622
6623
6624
6625
6626
6627
6628
6629
6629
6630
6631
6632
6633
6634
6635
6636
6637
6638
6639
6639
6640
6641
6642
6643
6644
6645
6646
6647
6648
6649
6649
6650
6651
6652
6653
6654
6655
6656
6657
6658
6659
6659
6660
6661
6662
6663
6664
6665
6666
6667
6668
6669
6669
6670
6671
6672
6673
6674
6675
6676
6677
6678
6679
6679
6680
6681
6682
6683
6684
6685
6686
6687
6688
6689
6689
6690
6691
6692
6693
6694
6695
6696
6697
6698
6699
6699
6700
6701
6702
6703
6704
6705
6706
6707
6708
6709
6709
6710
6711
6712
6713
6714
6715
6716
6717
6718
6719
6719
6720
6721
6722
6723
6724
6725
6726
6727
6728
6729
6729
6730
6731
6732
6733
6734
6735
6736
6737
6738
6739
6739
6740
6741
6742
6743
6744
6745
6746
6747
6748
6749
6749
6750
6751
6752
6753
6754
6755
6756
6757
6758
6759
6759
6760
6761
6762
6763
6764
6765
6766
6767
6768
6769
6769
6770
6771
6772
6773
6774
6775
6776
6777
6778
6779
6779
6780
6781
6782
6783
6784
6785
6786
6787
6788
6789
6789
6790
6791
6792
6793
6794
6795
6796
6797
6798
6799
6799
6800
6801
6802
6803
6804
6805
6806
6807
6808
6809
6809
6810
6811
6812
6813
6814
6815
6816
6817
6818
6819
6819
6820
6821
6822
6823
6824
6825
6826
6827
6828
6829
6829
6830
6831
6832
6833
6834
6835
6836
6837
6838
6839
6839
6840
6841
6842
6843
6844
6845
6846
6847
6848
6849
6849
6850
6851
6852
6853
6854
6855
6856
6857
6858
6859
6859
6860
6861
6862
6863
6864
6865
6866
6867
6868
6869
6869
6870
6871
6872
6873
6874
6875
6876
6877
6878
6879
6879
6880
6881
6882
6883
6884
6885
6886
6887
6888
6889
6889
6890
6891
6892
6893
6894
6895
6896
6897
6898
6899
6899
6900
6901
6902
6903
6904
6905
6906
6907
6908
6909
6909
6910
6911
6912
6913
6914
6915
6916
6917
6918
6919
6919
6920
6921
6922
6923
6924
6925
6926
6927
6928
6929
6929
6930
6931
6932
6933
6934
6935
6936
6937
6938
6939
6939
6940
6941
6942
6943
6944
6945
6946
6947
6948
6949
6949
6950
6951
6952
6953
6954
6955
6956
6957
6958
6959
6959
6960
6961
6962
6963
6964
6965
6966
6967
6968
6969
6969
6970
6971
6972
6973
6974
6975
6976
6977
6978
6979
6979
6980
6981
6982
6983
6984
6985
6986
6987
6988
6989
6989
6990
6991
6992
6993
6994
6995
6996
6997
6998
6999
6999
7000
7001
7002
7003
7004
7005
7006
7007
7008
7009
7009
7010
7011
7012
7013
7014
7015
7016
7017
7018
7019
7019
7020
7021
7022
7023
7024
7025
7026
7027
7028
7029
7029
7030
7031
7032
7033
7034
7035
7036
7037
7038
7039
7039
7040
7041
7042
7043
7044
7045
7046
7047
7048
7049
7049
7050
7051
7052
7053
7054
7055
7056
7057
7058
7059
7059
7060
7061
7062
7063
7064
7065
7066
7067
7068
7069
7069
7070
7071
7072
7073
7074
7075
7076
7077
7078
7079
7079
7080
7081
7082
7083
7084
7085
7086
7087
7088
7089
7089
7090
7091
7092
7093
7094
7095
7096
7097
7098
7099
7099
7100
7101
7102
7103
7104
7105
7106
7107
7108
7109
7109
7110
7111
7112
7113
7114
7115
7116
7117
7118
7119
7119
7120
7121
7122
7123
7124
7125
7126
7127
7128
7129
7129
7130
7131
7132
7133
7134
7135
7136
7137
7138
7139
7139
7140
7141
7142
7143
7144
7145
7146
7147
7148
7149
7149
7150
7151
7152
7153
7154
7155
7156
7157
7158
7159
7159
7160
7161
7162
7163
7164
7165
7166
7167
7168
7169
7169
7170
7171
7172
7173
7174
7175
7176
7177
7178
7179
7179
7180
7181
7182
7183
7184
7185
7186
7187
7188
7189
7189
7190
7191
7192
7193
7194
7195
7196
7197
7198
7199
7199
7200
7201
7202
7203
7204
7205
7206
7207
7208
7209
7209
7210
7211
7212
7213
7214
7215
7216
7217
7218
7219
7219
7220
7221
7222
7223
7224
7225
7226
7227
7228
7229
7229
7230
7231
7232
7233
7234
7235
7236
7237
7238
7239
7239
7240
7241
7242
7243
7244
7245
7246
7247
7248
7249
7249
7250
7251
7252
7253
7254
7255
7256
7257
7258
7259
7259
7260
7261
7262
7263
7264
7265
7266
7267
7268
7269
7269
7270
7271
7272
7273
7274
7275
7276
7277
7278
7279
7279
7280
7281
7282
7283
7284
7285
7286
7287
7288
7289
7289
7290
7291
7292
7293
7294
7295
7296
7297
7298
7299
7299
7300
7301
7302
7303
7304
7305
7306
7307
7308
7309
7309
7310
7311
7312
7313
7314
7315
7316
7317
7318
7319
7319
7320
7321
7322
7323
7324
7325
7326
7327
7328
7329
7329
7330
7331
7332
7333
7334
7335
7336
7337
7338
7339
7339
7340
7341
7342
7343
7344
7345
7346
7347
7348
7349
7349
7350
7351
7352
7353
7354
7355
7356
7357
7358
7359
7359
7360
7361
7362
7363
7364
7365
7366
7367
7368
7369
7369
7370
7371
7372
7373
7374
7375
7376
7377
7378
7379
7379
7380
7381
7382
7383
7384
7385
7386
7387
7388
7389
7389
7390
7391
7392
7393
7394
7395
7396
7397
7398
7399
7399
7400
7401
7402
7403
7404
7405
7406
7407
7408
7409
7409
7410
7411
7412
7413
7414
7415
7416
7417
7418
7419
7419
7420
7421
7422
7423
7424
7425
7426
7427
7428
7429
7429
7430
7431
7432
7433
7434
7435
7436
7437
7438
7439
7439
7440
7441
7442
7443
7444
7445
7446
7447
7448
7449
7449
7450
7451
7452
7453
7454
7455
7456
7457
7458
7459
7459
7460
7461
7462
7463
7464
7465
7466
7467
7468
7469
7469
7470
7471
7472
7473
7474
7475
7476
7477
7478
7479
7479
7480
7481
7482
7483
7484
7485
7486
7487
7488
7489
7489
7490
7491
7492
7493
7494
7495
7496
7497
7498
7499
7499
7500
7501
7502
7503
7504
7505
7506
7507
7508
7509
7509
7510
7511
7512
7513
7514
7515
7516
7517
7518
7519
7519
7520
7521
7522
7523
7524
7525
7526
7527
7528
7529
7529
7530
7531
7532
7533
7534
7535
7536
7537
7538
7539
7539
7540
7541
7542
7543
7544
7545
7546
7547
7548
7549
7549
7550
7551
7552
7553
7554
7555
7556
7557
7558
7559
7559
7560
7561
7562
7563
7564
7565
7566
7567
7568
7569
7569
7570
7571
7572
7573
7574
7575
7576
7577
7578
7579
7579
7580
7581
7582
7583
7584
7585
7586
7587
7588
7589
7589
7590
7591
7592
7593
7594
7595
7596
7597
7598
7599
7599
7600
7601
7602
7603
7604
7605
7606
7607
7608
7609
7609
7610
7611
7612
7613
7614
7615
7616
7617
7618
7619
7619
7620
7621
7622
7623
7624
7625
7626
7627
7628
7629
7629
7630
7631
7632
7633
7634
7635
7636
7637
7638
7639
7639
7640
7641
7642
7643
7644
7645
7646
7647
7648
7649
7649
7650
7651
7652
7653
7654
7655
7656
7657
7658
7659
7659
7660
7661
7662
7663
7664
7665
7666
7667
7668
7669
7669
7670
7671
7672
7673
7674
7675
7676
7677
7678
7679
7679
7680
7681
7682
7683
7684
7685
7686
7687
7688
7689
7689
7690
7691
7692
7693
7694
7695
7696
7697
7698
7699
7699
7700
7701
7702
7703
7704
7705
7706
7707
7708
7709
7709
7710
7711
7712
7713
7714
7715
7716
7717
7718
7719
7719
7720
7721
7722
7723
772
```

```
5741         }
5742     }
5743 }, 120000);
5744
5745 // Check for auto-saved data on load
5746 try {
5747     const autoSavedData = localStorage.getItem('animationXSheet_autosave');
5748     if (autoSavedData && !localStorage.getItem('animationXSheet')) {
5749         if (confirm('Found auto-saved data. Would you like to restore it?')) {
5750             restoreData(JSON.parse(autoSavedData));
5751         }
5752     }
5753 } catch (e) {
5754     console.error('Error checking for auto-saved data:', e);
5755 }
5756
5757 // Initial status
5758 updateStatusMessage('X-Sheet ready');
5759 });
5760 </script>
5761 </body>
5762
5763 </html>
```