

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 |         this.container.style.overflow = 'hidden'; // Prevent drawings from
566 |         overflowing
567 |         document.body.appendChild(this.container);
568 |
569 |         // Create default background and foreground layers
570 |         this.addLayer('background');
571 |         this.addLayer('foreground');
572 |         this.setActiveLayer(1); // Set foreground as active by default
573 |
574 |         // Handle window resize and table changes
575 |         this.setupResizeHandling();
576 |     }
577 |
578 |     addLayer(name) {
579 |         const canvas = document.createElement('canvas');
580 |         canvas.className = `drawing-layer-${name}`;
581 |         canvas.width = this.container.clientWidth;
582 |         canvas.height = this.container.clientHeight;
583 |         canvas.style.position = 'absolute';
584 |         canvas.style.left = '0';
585 |         canvas.style.top = '0';
586 |         canvas.style.pointerEvents = 'none';
587 |
588 |         this.container.appendChild(canvas);
589 |
590 |         const layer = {
```

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

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

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

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

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

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

```
887         return { x: minX, y: minY, width, height };
888     }
889
890     containsPoint(x, y) {
891         // Check if point is near the line
892         const lineLength = Math.sqrt(
893             Math.pow(this.x2 - this.x, 2) + Math.pow(this.y2 - this.y, 2)
894         );
895
896         // If line is too short, use a minimum distance
897         if (lineLength < 1) {
898             const dx = x - this.x;
899             const dy = y - this.y;
900             return Math.sqrt(dx * dx + dy * dy) <= 5;
901         }
902
903         // Calculate distance from point to line segment
904         const t = ((x - this.x) * (this.x2 - this.x) + (y - this.y) * (this.y2 -
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
920         // Calculate perpendicular distance
921         const px = this.x + t * (this.x2 - this.x);
922         const py = this.y + t * (this.y2 - this.y);
923         const dx = x - px;
924         const dy = y - py;
925         return Math.sqrt(dx * dx + dy * dy) <= 5;
926     }
927
928     move(dx, dy) {
929         super.move(dx, dy);
930         this.x2 += dx;
931         this.y2 += dy;
932     }
933
934     toJSON() {
935         const json = super.toJSON();
```

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

```
986     }
987
988     static fromJSON(data) {
989         return new ArrowObject(data);
990     }
991 }
992
993 // Rectangle object
994 class RectangleObject extends DrawingObject {
995     constructor(props = {}) {
996         super(props);
997         this.width = props.width || 0;
998         this.height = props.height || 0;
999         this.fill = props.fill || false;
1000        this.fillColor = props.fillColor || this.color;
1001        this.type = 'rectangle';
1002    }
1003
1004    draw(context) {
1005        context.save();
1006        context.strokeStyle = this.color;
1007        context.lineWidth = this.lineWidth;
1008
1009        // Draw rectangle
1010        if (this.fill) {
1011            context.fillStyle = this.fillColor;
1012            context.fillRect(this.x, this.y, this.width, this.height);
1013        }
1014        context.strokeRect(this.x, this.y, this.width, this.height);
1015        context.restore();
1016
1017        super.draw(context);
1018    }
1019
1020    getBounds() {
1021        return {
1022            x: this.x,
1023            y: this.y,
1024            width: this.width,
1025            height: this.height
1026        };
1027    }
1028
1029    containsPoint(x, y) {
1030        // Check if point is inside or near the edge of the rectangle
1031        if (this.fill) {
1032            // For filled rectangles, check if point is inside
1033            return (
1034                x >= this.x && x <= this.x + this.width &&
1035                y >= this.y && y <= this.y + this.height
1036            );
1037        }
1038    }
1039}
```

```
1036     );
1037   } else {
1038     // For unfilled rectangles, check if point is near the edges
1039     const nearLeft = Math.abs(x - this.x) <= 5;
1040     const nearRight = Math.abs(x - (this.x + this.width)) <= 5;
1041     const nearTop = Math.abs(y - this.y) <= 5;
1042     const nearBottom = Math.abs(y - (this.y + this.height)) <= 5;
1043
1044     return (
1045       (nearLeft || nearRight) && (y >= this.y && y <= this.y +
1046         this.height) ||
1047         (nearTop || nearBottom) && (x >= this.x && x <= this.x + this.width)
1048     );
1049   }
1050
1051   toJSON() {
1052     const json = super.toJSON();
1053     return {
1054       ...json,
1055       width: this.width,
1056       height: this.height,
1057       fill: this.fill,
1058       fillColor: this.fillColor
1059     };
1060   }
1061
1062   static fromJSON(data) {
1063     return new RectangleObject(data);
1064   }
1065 }
1066
1067 // Circle/Ellipse object
1068 class EllipseObject extends DrawingObject {
1069   constructor(props = {}) {
1070     super(props);
1071     this.radiusX = props.radiusX || 0;
1072     this.radiusY = props.radiusY || 0;
1073     this.fill = props.fill || false;
1074     this.fillColor = props.fillColor || this.color;
1075     this.type = 'ellipse';
1076   }
1077
1078   draw(context) {
1079     context.save();
1080     context.beginPath();
1081     context.strokeStyle = this.color;
1082     context.lineWidth = this.lineWidth;
1083
1084     // Draw ellipse
```

```
1085     context.ellipse(  
1086         this.x,  
1087         this.y,  
1088         this.radiusX,  
1089         this.radiusY,  
1090         0,  
1091         0,  
1092         2 * Math.PI  
1093     );  
1094  
1095     if (this.fill) {  
1096         context.fillStyle = this.fillColor;  
1097         context.fill();  
1098     }  
1099     context.stroke();  
1100     context.restore();  
1101  
1102     super.draw(context);  
1103 }  
1104  
1105 getBounds() {  
1106     return {  
1107         x: this.x - this.radiusX,  
1108         y: this.y - this.radiusY,  
1109         width: this.radiusX * 2,  
1110         height: this.radiusY * 2  
1111     };  
1112 }  
1113  
1114 containsPoint(x, y) {  
1115     // Check if point is inside or near the edge of the ellipse  
1116     const normalizedX = (x - this.x) / this.radiusX;  
1117     const normalizedY = (y - this.y) / this.radiusY;  
1118     const distance = Math.sqrt(normalizedX * normalizedX + normalizedY *  
normalizedY);  
1119  
1120     if (this.fill) {  
1121         // For filled ellipses, check if point is inside  
1122         return distance <= 1.0;  
1123     } else {  
1124         // For unfilled ellipses, check if point is near the edge  
1125         return Math.abs(distance - 1.0) <= 5 / this.radiusX;  
1126     }  
1127 }  
1128  
1129 toJSON() {  
1130     const json = super.toJSON();  
1131     return {  
1132         ...json,  
1133         radiusX: this.radiusX,
```

```
1134         radiusY: this.radiusY,
1135         fill: this.fill,
1136         fillColor: this.fillColor
1137     );
1138 }
1139
1140     static fromJSON(data) {
1141         return new EllipseObject(data);
1142     }
1143 }
1144
1145 // Text object
1146 class TextObject extends DrawingObject {
1147     constructor(props = {}) {
1148         super(props);
1149         this.text = props.text || '';
1150         this.fontSize = props.fontSize || 14;
1151         this.fontFamily = props.fontFamily || 'Arial, sans-serif';
1152         this.align = props.align || 'left';
1153         this.type = 'text';
1154     }
1155
1156     draw(context) {
1157         context.save();
1158         context.fillStyle = this.color;
1159         context.font = `${this.fontSize}px ${this.fontFamily}`;
1160         context.textAlign = this.align;
1161
1162         // Draw text
1163         context.fillText(this.text, this.x, this.y);
1164         context.restore();
1165
1166         super.draw(context);
1167     }
1168
1169     getBounds() {
1170         // Estimate text dimensions
1171         const dummyCanvas = document.createElement('canvas');
1172         const ctx = dummyCanvas.getContext('2d');
1173         ctx.font = `${this.fontSize}px ${this.fontFamily}`;
1174         const metrics = ctx.measureText(this.text);
1175         const height = this.fontSize; // Approximation
1176
1177         return {
1178             x: this.align === 'center' ? this.x - metrics.width / 2 :
1179                 this.align === 'right' ? this.x - metrics.width : this.x,
1180             y: this.y - height,
1181             width: metrics.width,
1182             height: height
1183         };
1184     }
1185 }
```

```
1184 }
1185
1186     containsPoint(x, y) {
1187         const bounds = this.getBounds();
1188         return (
1189             x >= bounds.x && x <= bounds.x + bounds.width &&
1190             y >= bounds.y && y <= bounds.y + bounds.height
1191         );
1192     }
1193
1194     toJSON() {
1195         const json = super.toJSON();
1196         return {
1197             ...json,
1198             text: this.text,
1199             fontSize: this.fontSize,
1200             fontFamily: this.fontFamily,
1201             align: this.align
1202         };
1203     }
1204
1205     static fromJSON(data) {
1206         return new TextObject(data);
1207     }
1208 }
1209
1210 // Image object
1211 class ImageObject extends DrawingObject {
1212     constructor(props = {}) {
1213         super(props);
1214         this.width = props.width || 0;
1215         this.height = props.height || 0;
1216         this.imageUrl = props.imageUrl || '';
1217         this.image = null;
1218         this.loaded = false;
1219         this.type = 'image';
1220
1221         // Load the image if provided
1222         if (this.imageUrl) {
1223             this.loadImage(this.imageUrl);
1224         }
1225     }
1226
1227     loadImage(url) {
1228         this.image = new Image();
1229         this.image.onload = () => {
1230             this.loaded = true;
1231
1232             // If dimensions not specified, use image dimensions
1233             if (this.width === 0 || this.height === 0) {
```

```
1234         this.width = this.image.width;
1235         this.height = this.image.height;
1236     }
1237
1238     // Trigger redraw
1239     document.dispatchEvent(new Event('xsheet-redraw'));
1240 };
1241 this.image.src = url;
1242 }
1243
1244 draw(context) {
1245     if (this.loaded && this.image) {
1246         context.save();
1247         context.drawImage(this.image, this.x, this.y, this.width, this.height);
1248         context.restore();
1249     } else if (!this.loaded) {
1250         // Draw placeholder while loading
1251         context.save();
1252         context.strokeStyle = '#999999';
1253         context.lineWidth = 1;
1254         context.strokeRect(this.x, this.y, this.width, this.height);
1255         context.font = '10px Arial';
1256         context.fillStyle = '#999999';
1257         context.fillText('Loading Image...', this.x + 5, this.y + 15);
1258         context.restore();
1259     }
1260
1261     super.draw(context);
1262 }
1263
1264 getBounds() {
1265     return {
1266         x: this.x,
1267         y: this.y,
1268         width: this.width,
1269         height: this.height
1270     };
1271 }
1272
1273 containsPoint(x, y) {
1274     return (
1275         x >= this.x && x <= this.x + this.width &&
1276         y >= this.y && y <= this.y + this.height
1277     );
1278 }
1279
1280 toJSON() {
1281     const json = super.toJSON();
1282     return {
1283         ...json,
```

```
1284     width: this.width,
1285     height: this.height,
1286     imageUrl: this.imageUrl
1287   );
1288 }
1289
1290   static fromJSON(data) {
1291     return new ImageObject(data);
1292   }
1293 }
1294
1295 // Symbol object (predefined animation symbols)
1296 class SymbolObject extends DrawingObject {
1297   constructor(props = {}) {
1298     super(props);
1299     this.symbolType = props.symbolType || 'default';
1300     this.scale = props.scale || 1.0;
1301     this.type = 'symbol';
1302   }
1303
1304   draw(context) {
1305     context.save();
1306     context.strokeStyle = this.color;
1307     context.fillStyle = this.color;
1308     context.lineWidth = this.lineWidth;
1309
1310     // Draw based on symbol type
1311     switch (this.symbolType) {
1312       case 'anticipation':
1313         this.drawAnticipation(context);
1314         break;
1315       case 'impact':
1316         this.drawImpact(context);
1317         break;
1318       case 'keyframe':
1319         this.drawKeyframe(context);
1320         break;
1321       case 'inbetween':
1322         this.drawInbetween(context);
1323         break;
1324       case 'hold':
1325         this.drawHold(context);
1326         break;
1327       default:
1328         this.drawDefault(context);
1329     }
1330
1331     context.restore();
1332     super.draw(context);
1333 }
```

```
1334
1335     drawAnticipation(context) {
1336         context.save();
1337         context.translate(this.x, this.y);
1338         context.scale(this.scale, this.scale);
1339
1340         // Draw curved arrow going back
1341         context.beginPath();
1342         context.moveTo(0, 0);
1343         context.bezierCurveTo(-20, -5, -25, 10, -10, 15);
1344         context.stroke();
1345
1346         // Draw arrowhead
1347         context.beginPath();
1348         context.moveTo(-10, 15);
1349         context.lineTo(-5, 10);
1350         context.lineTo(-15, 5);
1351         context.closePath();
1352         context.fill();
1353
1354         context.restore();
1355     }
1356
1357     drawImpact(context) {
1358         context.save();
1359         context.translate(this.x, this.y);
1360         context.scale(this.scale, this.scale);
1361
1362         // Draw impact star
1363         for (let i = 0; i < 8; i++) {
1364             const angle = (i / 8) * Math.PI * 2;
1365             const innerRadius = 5;
1366             const outerRadius = 15;
1367
1368             context.beginPath();
1369             context.moveTo(
1370                 innerRadius * Math.cos(angle),
1371                 innerRadius * Math.sin(angle)
1372             );
1373             context.lineTo(
1374                 outerRadius * Math.cos(angle),
1375                 outerRadius * Math.sin(angle)
1376             );
1377             context.stroke();
1378         }
1379
1380         context.restore();
1381     }
1382
1383     drawKeyframe(context) {
```

```
1384     context.save();
1385     context.translate(this.x, this.y);
1386     context.scale(this.scale, this.scale);
1387
1388     // Draw diamond
1389     context.beginPath();
1390     context.moveTo(0, -10);
1391     context.lineTo(10, 0);
1392     context.lineTo(0, 10);
1393     context.lineTo(-10, 0);
1394     context.closePath();
1395     context.stroke();
1396     context.fill();
1397
1398     context.restore();
1399 }
1400
1401 drawInbetween(context) {
1402     context.save();
1403     context.translate(this.x, this.y);
1404     context.scale(this.scale, this.scale);
1405
1406     // Draw circle
1407     context.beginPath();
1408     context.arc(0, 0, 7, 0, Math.PI * 2);
1409     context.stroke();
1410
1411     context.restore();
1412 }
1413
1414 drawHold(context) {
1415     context.save();
1416     context.translate(this.x, this.y);
1417     context.scale(this.scale, this.scale);
1418
1419     // Draw horizontal bar
1420     context.beginPath();
1421     context.moveTo(-15, 0);
1422     context.lineTo(15, 0);
1423     context.lineWidth = this.lineWidth * 2;
1424     context.stroke();
1425
1426     context.restore();
1427 }
1428
1429 drawDefault(context) {
1430     context.save();
1431     context.translate(this.x, this.y);
1432     context.scale(this.scale, this.scale);
1433 }
```

```
1434         // Draw square
1435         context.beginPath();
1436         context.rect(-7, -7, 14, 14);
1437         context.stroke();
1438
1439         context.restore();
1440     }
1441
1442     getBounds() {
1443         // Approximate bounds based on symbol type
1444         const size = 20 * this.scale;
1445         return {
1446             x: this.x - size / 2,
1447             y: this.y - size / 2,
1448             width: size,
1449             height: size
1450         };
1451     }
1452
1453     containsPoint(x, y) {
1454         const bounds = this.getBounds();
1455         const dx = x - this.x;
1456         const dy = y - this.y;
1457         const distance = Math.sqrt(dx * dx + dy * dy);
1458
1459         // Use a radius-based check as most symbols are roughly circular
1460         return distance <= bounds.width / 2;
1461     }
1462
1463     toJSON() {
1464         const json = super.toJSON();
1465         return {
1466             ...json,
1467             symbolType: this.symbolType,
1468             scale: this.scale
1469         };
1470     }
1471
1472     static fromJSON(data) {
1473         return new SymbolObject(data);
1474     }
1475 }
1476
1477 // Free-form path (for pen/brush tools)
1478 class PathObject extends DrawingObject {
1479     constructor(props = {}) {
1480         super(props);
1481         this.points = props.points || [];
1482         this.smoothing = props.smoothing !== undefined ? props.smoothing : true;
1483         this.closed = props.closed || false;
```

```
1484     this.fill = props.fill || false;
1485     this.fillColor = props.fillColor || this.color;
1486     this.type = 'path';
1487 }
1488
1489     addPoint(x, y) {
1490         this.points.push({ x, y });
1491     }
1492
1493     draw(context) {
1494         if (this.points.length < 2) return;
1495
1496         context.save();
1497         context.beginPath();
1498         context.strokeStyle = this.color;
1499         context.lineWidth = this.lineWidth;
1500         context.lineJoin = 'round';
1501         context.lineCap = 'round';
1502
1503         // Start from first point
1504         context.moveTo(this.points[0].x, this.points[0].y);
1505
1506         if (this.smoothing && this.points.length > 2) {
1507             // Draw using bezier curves for smoothing
1508             for (let i = 1; i < this.points.length - 1; i++) {
1509                 const p1 = this.points[i];
1510                 const p2 = this.points[i + 1];
1511
1512                 const xc = (p1.x + p2.x) / 2;
1513                 const yc = (p1.y + p2.y) / 2;
1514
1515                 context.quadraticCurveTo(p1.x, p1.y, xc, yc);
1516             }
1517
1518             // Connect to the last point
1519             const last = this.points[this.points.length - 1];
1520             context.lineTo(last.x, last.y);
1521         } else {
1522             // Simple line segments
1523             for (let i = 1; i < this.points.length; i++) {
1524                 context.lineTo(this.points[i].x, this.points[i].y);
1525             }
1526         }
1527
1528         if (this.closed) {
1529             context.closePath();
1530         }
1531
1532         if (this.fill) {
1533             context.fillStyle = this.fillColor;
```

```
1534         context.fill();
1535     }
1536
1537     context.stroke();
1538     context.restore();
1539
1540     super.draw(context);
1541 }
1542
1543 getBounds() {
1544     if (this.points.length === 0) {
1545         return { x: this.x, y: this.y, width: 0, height: 0 };
1546     }
1547
1548     let minX = this.points[0].x;
1549     let maxX = this.points[0].x;
1550     let minY = this.points[0].y;
1551     let maxY = this.points[0].y;
1552
1553     // Find min/max coordinates
1554     for (let i = 1; i < this.points.length; i++) {
1555         const point = this.points[i];
1556         minX = Math.min(minX, point.x);
1557         maxX = Math.max(maxX, point.x);
1558         minY = Math.min(minY, point.y);
1559         maxY = Math.max(maxY, point.y);
1560     }
1561
1562     return {
1563         x: minX,
1564         y: minY,
1565         width: maxX - minX,
1566         height: maxY - minY
1567     };
1568 }
1569
1570 containsPoint(x, y) {
1571     if (this.points.length < 2) return false;
1572
1573     // Check if point is near any line segment
1574     for (let i = 0; i < this.points.length - 1; i++) {
1575         const p1 = this.points[i];
1576         const p2 = this.points[i + 1];
1577
1578         const lineLength = Math.sqrt(
1579             Math.pow(p2.x - p1.x, 2) + Math.pow(p2.y - p1.y, 2)
1580         );
1581
1582         // If line is too short, check distance to point
1583         if (lineLength < 1) {
```

```
1584     const dx = x - p1.x;
1585     const dy = y - p1.y;
1586     if (Math.sqrt(dx * dx + dy * dy) <= 5) {
1587         return true;
1588     }
1589     continue;
1590 }
1591
1592 // Calculate distance from point to line segment
1593 const t = ((x - p1.x) * (p2.x - p1.x) + (y - p1.y) * (p2.y - p1.y)) /
1594 (lineLength * lineLength);
1595
1596     if (t < 0) {
1597         // Point is beyond start point
1598         const dx = x - p1.x;
1599         const dy = y - p1.y;
1600         if (Math.sqrt(dx * dx + dy * dy) <= 5) {
1601             return true;
1602         }
1603     } else if (t > 1) {
1604         // Point is beyond end point
1605         const dx = x - p2.x;
1606         const dy = y - p2.y;
1607         if (Math.sqrt(dx * dx + dy * dy) <= 5) {
1608             return true;
1609         }
1610     } else {
1611         // Calculate perpendicular distance
1612         const px = p1.x + t * (p2.x - p1.x);
1613         const py = p1.y + t * (p2.y - p1.y);
1614         const dx = x - px;
1615         const dy = y - py;
1616         if (Math.sqrt(dx * dx + dy * dy) <= 5) {
1617             return true;
1618         }
1619     }
1620
1621 // If closed and filled, also check if point is inside
1622 if (this.closed && this.fill) {
1623     // Use point-in-polygon algorithm
1624     let inside = false;
1625     for (let i = 0, j = this.points.length - 1; i < this.points.length; j =
1626 i++) {
1627         const xi = this.points[i].x;
1628         const yi = this.points[i].y;
1629         const xj = this.points[j].x;
1630         const yj = this.points[j].y;
1631
1632         const intersect = ((yi > y) !== (yj > y)) &&
```

```
1632             (x < (xj - xi) * (y - yi) / (yj - yi) + xi);
1633
1634         if (intersect) inside = !inside;
1635     }
1636     return inside;
1637 }
1638
1639     return false;
1640 }
1641
1642     move(dx, dy) {
1643         super.move(dx, dy);
1644
1645         // Move all points
1646         this.points.forEach(point => {
1647             point.x += dx;
1648             point.y += dy;
1649         });
1650     }
1651
1652     toJSON() {
1653         const json = super.toJSON();
1654         return {
1655             ...json,
1656             points: this.points,
1657             smoothing: this.smoothing,
1658             closed: this.closed,
1659             fill: this.fill,
1660             fillColor: this.fillColor
1661         };
1662     }
1663
1664     static fromJSON(data) {
1665         return new PathObject(data);
1666     }
1667 }
1668
1669 // Frame-Spanning Line (connects specific cells in the grid)
1670 class FrameSpanningLineObject extends DrawingObject {
1671     constructor(props = {}) {
1672         super(props);
1673         this.startFrame = props.startFrame || 1;
1674         this.startColumn = props.startColumn || 1;
1675         this.endFrame = props.endFrame || 1;
1676         this.endColumn = props.endColumn || 1;
1677         this.type = 'frameSpanningLine';
1678         this.dashPattern = props.dashPattern || [];
1679         this.arrowStart = props.arrowStart || false;
1680         this.arrowEnd = props.arrowEnd || false;
1681         this.arrowSize = props.arrowSize || 10;
```

```
1682 }
1683
1684     draw(context) {
1685         // Calculate actual coordinates from frame and column
1686         const layerSystem = window.xsheetDrawing.layerSystem;
1687
1688         const startPos = layerSystem.gridToCanvas(this.startFrame,
1689             this.startColumn);
1690         const endPos = layerSystem.gridToCanvas(this.endFrame, this.endColumn);
1691
1692         if (!startPos || !endPos) return; // Skip if cells not found
1693
1694         context.save();
1695         context.beginPath();
1696         context.strokeStyle = this.color;
1697         context.lineWidth = this.lineWidth;
1698
1699         if (this.dashPattern.length > 0) {
1700             context.setLineDash(this.dashPattern);
1701         }
1702
1703         context.moveTo(startPos.x, startPos.y);
1704         context.lineTo(endPos.x, endPos.y);
1705         context.stroke();
1706
1707         // Draw arrows if needed
1708         if (this.arrowStart) {
1709             this.drawArrow(context, endPos.x, endPos.y, startPos.x, startPos.y);
1710         }
1711
1712         if (this.arrowEnd) {
1713             this.drawArrow(context, startPos.x, startPos.y, endPos.x, endPos.y);
1714         }
1715
1716         context.restore();
1717
1718         // Store computed coordinates for selection	hit testing
1719         this.computedStart = startPos;
1720         this.computedEnd = endPos;
1721
1722         super.draw(context);
1723     }
1724
1725     drawArrow(context, fromX, fromY, toX, toY) {
1726         const angle = Math.atan2(toY - fromY, toX - fromX);
1727
1728         context.save();
1729         context.fillStyle = this.color;
1730         context.beginPath();
1731         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)
1734     );
1735     context.lineTo(
1736         toX - this.arrowSize * Math.cos(angle + Math.PI/6),
1737         toY - this.arrowSize * Math.sin(angle + Math.PI/6)
1738     );
1739     context.closePath();
1740     context.fill();
1741     context.restore();
1742 }
1743
1744     getBounds() {
1745         if (!this.computedStart || !this.computedEnd) return { x: 0, y: 0, width: 0,
height: 0 };
1746
1747         const minX = Math.min(this.computedStart.x, this.computedEnd.x);
1748         const minY = Math.min(this.computedStart.y, this.computedEnd.y);
1749         const width = Math.abs(this.computedEnd.x - this.computedStart.x);
1750         const height = Math.abs(this.computedEnd.y - this.computedStart.y);
1751
1752         return { x: minX, y: minY, width, height };
1753     }
1754
1755     containsPoint(x, y) {
1756         if (!this.computedStart || !this.computedEnd) return false;
1757
1758         // Same algorithm as LineObject
1759         const lineLength = Math.sqrt(
1760             Math.pow(this.computedEnd.x - this.computedStart.x, 2) +
1761             Math.pow(this.computedEnd.y - this.computedStart.y, 2)
1762         );
1763
1764         if (lineLength < 1) {
1765             const dx = x - this.computedStart.x;
1766             const dy = y - this.computedStart.y;
1767             return Math.sqrt(dx * dx + dy * dy) <= 5;
1768         }
1769
1770         const t = ((x - this.computedStart.x) * (this.computedEnd.x -
this.computedStart.x) +
1771                     (y - this.computedStart.y) * (this.computedEnd.y -
this.computedStart.y)) /
1772                     (lineLength * lineLength);
1773
1774         if (t < 0) {
1775             const dx = x - this.computedStart.x;
1776             const dy = y - this.computedStart.y;
1777             return Math.sqrt(dx * dx + dy * dy) <= 5;
1778         }

```

```
1779
1780      if (t > 1) {
1781          const dx = x - this.computedEnd.x;
1782          const dy = y - this.computedEnd.y;
1783          return Math.sqrt(dx * dx + dy * dy) <= 5;
1784      }
1785
1786      const px = this.computedStart.x + t * (this.computedEnd.x -
1787          this.computedStart.x);
1788      const py = this.computedStart.y + t * (this.computedEnd.y -
1789          this.computedStart.y);
1790      const dx = x - px;
1791      const dy = y - py;
1792      return Math.sqrt(dx * dx + dy * dy) <= 5;
1793  }
1794
1795 // This object type doesn't use the regular move method
1796 // Instead, it updates the frame/column values
1797
1798     toJSON() {
1799         const json = super.toJSON();
1800         return {
1801             ...json,
1802             startFrame: this.startFrame,
1803             startColumn: this.startColumn,
1804             endFrame: this.endFrame,
1805             endColumn: this.endColumn,
1806             dashPattern: this.dashPattern,
1807             arrowStart: this.arrowStart,
1808             arrowEnd: this.arrowEnd,
1809             arrowSize: this.arrowSize
1810         };
1811     }
1812
1813     static fromJSON(data) {
1814         return new FrameSpanningLineObject(data);
1815     }
1816
1817 // Register all object types in a factory
1818 const DrawingObjectFactory = {
1819     types: {
1820         'drawingObject': DrawingObject,
1821         'line': LineObject,
1822         'arrow': ArrowObject,
1823         'rectangle': RectangleObject,
1824         'ellipse': EllipseObject,
1825         'text': TextObject,
1826         'image': ImageObject,
1827         'symbol': SymbolObject,
```

```
1827         'path': PathObject,
1828         'frameSpanningLine': FrameSpanningLineObject
1829     },
1830
1831     createFromJSON(data) {
1832         const Type = this.types[data.type];
1833         if (Type) {
1834             return Type.fromJSON(data);
1835         }
1836         return null;
1837     }
1838 };
1839
1840 /**
1841 * DRAWING TOOL SYSTEM
1842 * Manages the drawing tools and interfaces with the layer system
1843 */
1844 class DrawingToolSystem {
1845     constructor(layerSystem) {
1846         this.layerSystem = layerSystem;
1847         this.activeTool = null;
1848         this.toolSettings = {
1849             color: '#ff0000',
1850             lineWidth: 2,
1851             fill: false,
1852             fillColor: '#ff8080',
1853             fontSize: 16,
1854             fontFamily: 'Arial, sans-serif',
1855             textAlign: 'left',
1856             symbolType: 'default',
1857             symbolScale: 1.0
1858         };
1859
1860         this.availableTools = {
1861             'select': new SelectTool(this),
1862             'pen': new PenTool(this),
1863             'line': new LineTool(this),
1864             'arrow': new ArrowTool(this),
1865             'rectangle': new RectangleTool(this),
1866             'ellipse': new EllipseTool(this),
1867             'text': new TextTool(this),
1868             'image': new ImageTool(this),
1869             'symbol': new SymbolTool(this),
1870             'frameLine': new FrameSpanningLineTool(this),
1871             'eraser': new EraserTool(this)
1872         };
1873
1874         // Default to select tool
1875         this.setActiveTool('select');
1876     }
```

```
1877         // Set up toolbar UI
1878         this.createToolbar();
1879     }
1880
1881     setActiveTool(toolName) {
1882         if (this.activeTool) {
1883             this.activeTool.deactivate();
1884         }
1885
1886         if (this.availableTools[toolName]) {
1887             this.activeTool = this.availableTools[toolName];
1888             this.activeTool.activate();
1889             return true;
1890         }
1891
1892         return false;
1893     }
1894
1895     createToolbar() {
1896         // Create toolbar container
1897         const toolbar = document.createElement('div');
1898         toolbar.className = 'drawing-toolbar';
1899         toolbar.style.display = 'flex';
1900         toolbar.style.flexWrap = 'wrap';
1901         toolbar.style.gap = '5px';
1902         toolbar.style.padding = '10px';
1903         toolbar.style.backgroundColor = '#f5f5f5';
1904         toolbar.style.marginBottom = '10px';
1905         toolbar.style.borderRadius = '5px';
1906
1907         // Add tool buttons
1908         this.addButton(toolbar, 'select', 'Select', '👉');
1909         this.addButton(toolbar, 'pen', 'Freehand Drawing', '📝');
1910         this.addButton(toolbar, 'line', 'Line', '‐');
1911         this.addButton(toolbar, 'arrow', 'Arrow', '→');
1912         this.addButton(toolbar, 'rectangle', 'Rectangle', '□');
1913         this.addButton(toolbar, 'ellipse', 'Circle/Ellipse', '○');
1914         this.addButton(toolbar, 'text', 'Text', 'T');
1915         this.addButton(toolbar, 'image', 'Insert Image', '🖼️');
1916         this.addButton(toolbar, 'symbol', 'Animation Symbol', '⭐');
1917         this.addButton(toolbar, 'frameLine', 'Multi-Frame Line', '⤧');
1918
1919         // Add separator
1920         toolbar.appendChild(document.createElement('div')).style.borderLeft = '1px
solid #ccc';
1921         toolbar.lastChild.style.height = '30px';
1922
1923         // Add color picker
1924         const colorContainer = document.createElement('div');
1925         colorContainer.style.display = 'flex';
```

```
1926 colorContainer.style.alignItems = 'center';
1927 colorContainer.style.gap = '5px';
1928
1929     const colorLabel = document.createElement('label');
1930     colorLabel.textContent = 'Color:';
1931     colorLabel.htmlFor = 'drawing-color';
1932
1933     const colorPicker = document.createElement('input');
1934     colorPicker.type = 'color';
1935     colorPicker.id = 'drawing-color';
1936     colorPicker.value = this.toolSettings.color;
1937     colorPicker.addEventListener('input', (e) => {
1938         this.toolSettings.color = e.target.value;
1939     });
1940
1941     colorContainer.appendChild(colorLabel);
1942     colorContainer.appendChild(colorPicker);
1943     toolbar.appendChild(colorContainer);
1944
1945     // Add line width selector
1946     const lineWidthContainer = document.createElement('div');
1947     lineWidthContainer.style.display = 'flex';
1948     lineWidthContainer.style.alignItems = 'center';
1949     lineWidthContainer.style.gap = '5px';
1950
1951     const lineWidthLabel = document.createElement('label');
1952     lineWidthLabel.textContent = 'Width:';
1953     lineWidthLabel.htmlFor = 'drawing-line-width';
1954
1955     const lineWidthSelect = document.createElement('select');
1956     lineWidthSelect.id = 'drawing-line-width';
1957
1958     const widths = [1, 2, 3, 5, 8, 12];
1959     widths.forEach(width => {
1960         const option = document.createElement('option');
1961         option.value = width;
1962         option.textContent = width + 'px';
1963         if (width === this.toolSettings.lineWidth) {
1964             option.selected = true;
1965         }
1966         lineWidthSelect.appendChild(option);
1967     });
1968
1969     lineWidthSelect.addEventListener('change', (e) => {
1970         this.toolSettings.lineWidth = parseInt(e.target.value);
1971     });
1972
1973     lineWidthContainer.appendChild(lineWidthLabel);
1974     lineWidthContainer.appendChild(lineWidthSelect);
1975     toolbar.appendChild(lineWidthContainer);
```

```
1976
1977    // Add fill option
1978    const fillContainer = document.createElement('div');
1979    fillContainer.style.display = 'flex';
1980    fillContainer.style.alignItems = 'center';
1981    fillContainer.style.gap = '5px';
1982
1983    const fillCheck = document.createElement('input');
1984    fillCheck.type = 'checkbox';
1985    fillCheck.id = 'drawing-fill';
1986    fillCheck.checked = this.toolSettings.fill;
1987
1988    const fillLabel = document.createElement('label');
1989    fillLabel.textContent = 'Fill';
1990    fillLabel.htmlFor = 'drawing-fill';
1991
1992    fillCheck.addEventListener('change', (e) => {
1993        this.toolSettings.fill = e.target.checked;
1994        fillColorPicker.disabled = !e.target.checked;
1995    });
1996
1997    const fillColorPicker = document.createElement('input');
1998    fillColorPicker.type = 'color';
1999    fillColorPicker.id = 'drawing-fill-color';
2000    fillColorPicker.value = this.toolSettings.fillColor;
2001    fillColorPicker.disabled = !this.toolSettings.fill;
2002
2003    fillColorPicker.addEventListener('input', (e) => {
2004        this.toolSettings.fillColor = e.target.value;
2005    });
2006
2007    fillContainer.appendChild(fillCheck);
2008    fillContainer.appendChild(fillLabel);
2009    fillContainer.appendChild(fillColorPicker);
2010    toolbar.appendChild(fillContainer);
2011
2012    // Add separator
2013    toolbar.appendChild(document.createElement('div')).style.borderLeft = '1px
solid #ccc';
2014    toolbar.lastChild.style.height = '30px';
2015
2016    // Add layer selector
2017    const layerSelector = document.createElement('select');
2018    layerSelector.id = 'drawing-layer-selector';
2019
2020    // Add options for each layer
2021    this.layerSystem.layers.forEach((layer, index) => {
2022        const option = document.createElement('option');
2023        option.value = index;
2024        option.textContent = layer.name;
```

```
2025     if (index === this.layerSystem.activeLayerIndex) {
2026         option.selected = true;
2027     }
2028     layerSelector.appendChild(option);
2029 };
2030
2031     layerSelector.addEventListener('change', (e) => {
2032         this.layerSystem.setActiveLayer(parseInt(e.target.value));
2033     });
2034
2035     const layerLabel = document.createElement('label');
2036     layerLabel.textContent = 'Layer:';
2037     layerLabel.htmlFor = 'drawing-layer-selector';
2038     layerLabel.style.marginRight = '5px';
2039
2040     toolbar.appendChild(layerLabel);
2041     toolbar.appendChild(layerSelector);
2042
2043     // Add separator
2044     toolbar.appendChild(document.createElement('div')).style.borderLeft = '1px
solid #ccc';
2045     toolbar.lastChild.style.height = '30px';
2046
2047     // Add eraser tool
2048     this.addToolButton(toolbar, 'eraser', 'Eraser', '✗');
2049
2050     // Add clear button
2051     const clearButton = document.createElement('button');
2052     clearButton.textContent = 'Clear All Drawings';
2053     clearButton.style.backgroundColor = '#ff5555';
2054     clearButton.style.color = 'white';
2055     clearButton.style.border = 'none';
2056     clearButton.style.borderRadius = '4px';
2057     clearButton.style.padding = '5px 10px';
2058     clearButton.style.cursor = 'pointer';
2059
2060     clearButton.addEventListener('click', () => {
2061         if (confirm('Are you sure you want to clear all drawings?')) {
2062             this.layerSystem.clearAllLayers();
2063         }
2064     });
2065
2066     toolbar.appendChild(clearButton);
2067
2068     // Find controls div and add toolbar before it
2069     const controls = document.querySelector('.controls');
2070     if (controls) {
2071         controls.parentNode.insertBefore(toolbar, controls.nextSibling);
2072     } else {
2073         document.body.insertBefore(toolbar, document.body.firstChild);
```

```
2074         }
2075     }
2076
2077     addToolBarButton(toolbar, toolName, tooltip, icon) {
2078         const button = document.createElement('button');
2079         button.textContent = icon;
2080         button.title = tooltip;
2081         button.style.width = '36px';
2082         button.style.height = '36px';
2083         button.style.fontSize = '16px';
2084         button.style.margin = '0';
2085         button.style.padding = '5px';
2086         button.style.borderRadius = '4px';
2087         button.style.border = '1px solid #ccc';
2088         button.style.backgroundColor = 'white';
2089         button.style.cursor = 'pointer';
2090
2091         button.addEventListener('click', () => {
2092             this.setActiveTool(toolName);
2093
2094             // Update active button styling
2095             document.querySelectorAll('.drawing-toolbar button').forEach(btn => {
2096                 btn.style.backgroundColor = 'white';
2097                 btn.style.color = 'black';
2098             });
2099
2100             button.style.backgroundColor = '#4CAF50';
2101             button.style.color = 'white';
2102         });
2103
2104         // Set active state for default tool
2105         if (toolName === 'select') {
2106             button.style.backgroundColor = '#4CAF50';
2107             button.style.color = 'white';
2108         }
2109
2110         toolbar.appendChild(button);
2111     }
2112 }
2113
2114 /**
2115 * DRAWING TOOLS
2116 * Individual tool implementations
2117 */
2118
2119 // Base tool class
2120 class DrawingTool {
2121     constructor(toolSystem) {
2122         this.toolSystem = toolSystem;
2123         this.layerSystem = toolSystem.layerSystem;
```

```
2124         this.active = false;
2125         this.settings = toolSystem.toolSettings;
2126     }
2127
2128     activate() {
2129         this.active = true;
2130         this.layerSystem.enableDrawing();
2131         this.attachEvents();
2132     }
2133
2134     deactivate() {
2135         this.active = false;
2136         // turn pointer-events back off so clicks go to the table cells
2137         this.toolSystem.layerSystem.disableDrawing();
2138         this.detachEvents();
2139     }
2140
2141     attachEvents() {
2142         // Override in subclasses
2143     }
2144
2145     detachEvents() {
2146         // Override in subclasses
2147     }
2148 }
2149
2150 // Select tool for manipulating objects
2151 class SelectTool extends DrawingTool {
2152
2153     constructor(toolSystem) {
2154         super(toolSystem);
2155         this.selectedObject = null;
2156         this.selectedLayer = null;
2157         this.dragging = false;
2158         this.dragStart = { x: 0, y: 0 };
2159         this.objectStart = { x: 0, y: 0 };
2160     }
2161     activate() {
2162         this.active = true;
2163         // let pointer-events go to the table cells instead of the canvases
2164         this.layerSystem.disableDrawing();
2165         this.attachEvents();
2166     }
2167
2168     deactivate() {
2169         this.active = false;
2170         this.layerSystem.disableDrawing();
2171         this.detachEvents();
2172     }
2173     attachEvents() {
```

```
2174     const canvas = this.layerSystem.getActiveLayer().canvas;
2175
2176     canvas.addEventListener('mousedown', this.handleMouseDown);
2177     canvas.addEventListener('pointerdown', this.handleMouseDown);
2178     document.addEventListener('mousemove', this.handleMouseMove);
2179     document.addEventListener('pointermove', this.handleMouseMove);
2180     document.addEventListener('mouseup', this.handleMouseUp);
2181     document.addEventListener('pointerup', this.handleMouseUp);
2182     document.addEventListener('keydown', this.handleKeyDown);
2183 }
2184
2185 detachEvents() {
2186     const canvas = this.layerSystem.getActiveLayer().canvas;
2187
2188     canvas.removeEventListener('mousedown', this.handleMouseDown);
2189     canvas.removeEventListener('pointerdown', this.handleMouseDown);
2190     document.removeEventListener('mousemove', this.handleMouseMove);
2191     document.removeEventListener('pointermove', this.handleMouseMove);
2192     document.removeEventListener('mouseup', this.handleMouseUp);
2193     document.removeEventListener('pointerup', this.handleMouseUp);
2194     document.removeEventListener('keydown', this.handleKeyDown);
2195
2196     // Clear selection
2197     this.clearSelection();
2198 }
2199
2200 handleMouseDown = (e) => {
2201     // Convert to canvas coordinates
2202     const coords = this.layerSystem.screenToCanvas(e.clientX, e.clientY);
2203
2204     // Check if clicked on an object
2205     const hit = this.layerSystem.findObjectAt(coords.x, coords.y);
2206
2207     if (hit) {
2208         // Select the object
2209         this.selectObject(hit.object, hit.layerIndex);
2210
2211         // Start drag
2212         this.dragging = true;
2213         this.dragStart = { x: coords.x, y: coords.y };
2214         this.objectStart = { x: hit.object.x, y: hit.object.y };
2215     } else {
2216         // Clear selection if clicked empty space
2217         this.clearSelection();
2218     }
2219 }
2220
2221 handleMouseMove = (e) => {
2222     if (!this.dragging || !this.selectedObject) return;
2223 }
```

```
2224     const coords = this.layerSystem.screenToCanvas(e.clientX, e.clientY);  
2225  
2226     // Calculate move distance  
2227     const dx = coords.x - this.dragStart.x;  
2228     const dy = coords.y - this.dragStart.y;  
2229  
2230     // Move the selected object  
2231     this.selectedObject.x = this.objectStart.x + dx;  
2232     this.selectedObject.y = this.objectStart.y + dy;  
2233  
2234     // For objects with special move handling  
2235     this.selectedObject.move(dx, dy);  
2236     this.selectedObject.x = this.objectStart.x; // Reset x as move() already  
handled it  
2237     this.selectedObject.y = this.objectStart.y; // Reset y as move() already  
handled it  
2238  
2239     // Redraw  
2240     this.layerSystem.redrawAll();  
2241 }  
2242  
2243 handleMouseUp = () => {  
2244     this.dragging = false;  
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) {
2275         // Clear previous selection
2276         this.clearSelection();
2277
2278         // Set new selection
2279         this.selectedObject = object;
2280         this.selectedLayer = layerIndex;
2281         object.selected = true;
2282
2283         // Redraw with selection visual
2284         this.layerSystem.redrawAll();
2285     }
2286
2287     clearSelection() {
2288         if (this.selectedObject) {
2289             this.selectedObject.selected = false;
2290             this.selectedObject = null;
2291             this.selectedLayer = null;
2292             this.layerSystem.redrawAll();
2293         }
2294     }
2295 }
2296
2297 // Pen tool for free drawing
2298 class PenTool extends DrawingTool {
2299     constructor(toolSystem) {
2300         super(toolSystem);
2301         this.currentPath = null;
2302         this.drawing = false;
2303     }
2304
2305     attachEvents() {
2306         const canvas = this.layerSystem.getActiveLayer().canvas;
2307
2308         canvas.addEventListener('mousedown', this.handleMouseDown);
2309         canvas.addEventListener('pointerdown', this.handleMouseDown);
2310         document.addEventListener('mousemove', this.handleMouseMove);
2311         document.addEventListener('pointermove', this.handleMouseMove);
2312         document.addEventListener('mouseup', this.handleMouseUp);
2313         document.addEventListener('pointerup', this.handleMouseUp);
2314     }
2315
2316     detachEvents() {
2317         const canvas = this.layerSystem.getActiveLayer().canvas;
2318
2319         canvas.removeEventListener('mousedown', this.handleMouseDown);
2320         canvas.removeEventListener('pointerdown', this.handleMouseDown);
2321         document.removeEventListener('mousemove', this.handleMouseMove);
```

```
2322     document.removeEventListener('pointermove', this.handleMouseMove);
2323     document.removeEventListener('mouseup', this.handleMouseUp);
2324     document.removeEventListener('pointerup', this.handleMouseUp);
2325 
2326     // Finish any in-progress drawing
2327     this.finishDrawing();
2328 }
2329 
2330 handleMouseDown = (e) => {
2331     e.preventDefault();
2332     if (e.pointerId != null) {
2333         e.target.setPointerCapture(e.pointerId);
2334     }
2335 
2336     const coords = this.layerSystem.screenToCanvas(e.clientX, e.clientY);
2337 
2338     // Start a new path
2339     this.currentPath = new PathObject({
2340         color: this.settings.color,
2341         lineWidth: this.settings.lineWidth,
2342         x: coords.x,
2343         y: coords.y
2344     });
2345 
2346     this.currentPath.addPoint(coords.x, coords.y);
2347     this.drawing = true;
2348 
2349     // Add to layer
2350     this.layerSystem.addObject(this.currentPath);
2351 }
2352 
2353 handleMouseMove = (e) => {
2354     if (!this.drawing || !this.currentPath) return;
2355 
2356     const coords = this.layerSystem.screenToCanvas(e.clientX, e.clientY);
2357 
2358     // Add point to path
2359     this.currentPath.addPoint(coords.x, coords.y);
2360 
2361     // Redraw
2362     this.layerSystem.redrawLayer(this.layerSystem.activeLayerIndex);
2363 }
2364 
2365 handleMouseUp = (e) => {
2366     if (e.pointerId != null) {
2367         e.target.releasePointerCapture(e.pointerId);
2368     }
2369     this.finishDrawing();
2370 }
2371 }
```

```
2372     finishDrawing() {
2373         if (this.drawing && this.currentPath) {
2374             // Finish the path
2375             this.drawing = false;
2376             this.currentPath = null;
2377         }
2378     }
2379 }
2380
2381 // Line tool
2382 class LineTool extends DrawingTool {
2383     constructor(toolSystem) {
2384         super(toolSystem);
2385         this.startPoint = null;
2386         this.currentLine = null;
2387         this.drawing = false;
2388     }
2389
2390     attachEvents() {
2391         const canvas = this.layerSystem.getActiveLayer().canvas;
2392
2393         canvas.addEventListener('mousedown', this.handleMouseDown);
2394         document.addEventListener('mousemove', this.handleMouseMove);
2395         document.addEventListener('mouseup', this.handleMouseUp);
2396     }
2397
2398     detachEvents() {
2399         const canvas = this.layerSystem.getActiveLayer().canvas;
2400
2401         canvas.removeEventListener('mousedown', this.handleMouseDown);
2402         document.removeEventListener('mousemove', this.handleMouseMove);
2403         document.removeEventListener('mouseup', this.handleMouseUp);
2404
2405         // Finish any in-progress drawing
2406         this.finishDrawing();
2407     }
2408
2409     handleMouseDown = (e) => {
2410         const coords = this.layerSystem.screenToCanvas(e.clientX, e.clientY);
2411
2412         this.startPoint = coords;
2413
2414         // Create temporary line
2415         this.currentLine = new LineObject({
2416             x: coords.x,
2417             y: coords.y,
2418             x2: coords.x,
2419             y2: coords.y,
2420             color: this.settings.color,
2421             lineWidth: this.settings.lineWidth
2422         })
2423     }
2424 }
```

```
2422     });
2423
2424     this.drawing = true;
2425
2426     // Add to layer
2427     this.layerSystem.addObject(this.currentLine);
2428 }
2429
2430 handleMouseMove = (e) => {
2431   if (!this.drawing || !this.currentLine) return;
2432
2433   const coords = this.layerSystem.screenToCanvas(e.clientX, e.clientY);
2434
2435   // Update end point
2436   this.currentLine.x2 = coords.x;
2437   this.currentLine.y2 = coords.y;
2438
2439   // Redraw
2440   this.layerSystem.redrawLayer(this.layerSystem.activeLayerIndex);
2441 }
2442
2443 handleMouseUp = () => {
2444   this.finishDrawing();
2445 }
2446
2447 finishDrawing() {
2448   if (this.drawing && this.currentLine) {
2449     // Finish the line
2450     this.drawing = false;
2451     this.startPoint = null;
2452     this.currentLine = null;
2453   }
2454 }
2455
2456
2457 // Arrow tool (extends Line tool)
2458 class ArrowTool extends LineTool {
2459   handleMouseDown = (e) => {
2460     const coords = this.layerSystem.screenToCanvas(e.clientX, e.clientY);
2461
2462     this.startPoint = coords;
2463
2464     // Create temporary arrow
2465     this.currentLine = new ArrowObject({
2466       x: coords.x,
2467       y: coords.y,
2468       x2: coords.x,
2469       y2: coords.y,
2470       color: this.settings.color,
2471       lineWidth: this.settings.lineWidth
2472   }
```

```
2472     });
2473 
2474     this.drawing = true;
2475 
2476     // Add to layer
2477     this.layerSystem.addObject(this.currentLine);
2478 }
2479 }
2480 
2481 // Rectangle tool
2482 class RectangleTool extends DrawingTool {
2483     constructor(toolSystem) {
2484         super(toolSystem);
2485         this.startPoint = null;
2486         this.currentRect = null;
2487         this.drawing = false;
2488     }
2489 
2490     attachEvents() {
2491         const canvas = this.layerSystem.getActiveLayer().canvas;
2492 
2493         canvas.addEventListener('mousedown', this.handleMouseDown);
2494         document.addEventListener('mousemove', this.handleMouseMove);
2495         document.addEventListener('mouseup', this.handleMouseUp);
2496     }
2497 
2498     detachEvents() {
2499         const canvas = this.layerSystem.getActiveLayer().canvas;
2500 
2501         canvas.removeEventListener('mousedown', this.handleMouseDown);
2502         document.removeEventListener('mousemove', this.handleMouseMove);
2503         document.removeEventListener('mouseup', this.handleMouseUp);
2504 
2505         // Finish any in-progress drawing
2506         this.finishDrawing();
2507     }
2508 
2509     handleMouseDown = (e) => {
2510         const coords = this.layerSystem.screenToCanvas(e.clientX, e.clientY);
2511 
2512         this.startPoint = coords;
2513 
2514         // Create temporary rectangle
2515         this.currentRect = new RectangleObject({
2516             x: coords.x,
2517             y: coords.y,
2518             width: 0,
2519             height: 0,
2520             color: this.settings.color,
2521             lineWidth: this.settings.lineWidth,
```

```
2522         fill: this.settings.fill,
2523         fillColor: this.settings.fillColor
2524     });
2525
2526     this.drawing = true;
2527
2528     // Add to layer
2529     this.layerSystem.addObject(this.currentRect);
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
2575 // Ellipse tool
2576 class EllipseTool extends DrawingTool {
2577     constructor(toolSystem) {
2578         super(toolSystem);
2579         this.center = null;
2580         this.currentEllipse = null;
2581         this.drawing = false;
2582     }
2583
2584     attachEvents() {
2585         const canvas = this.layerSystem.getActiveLayer().canvas;
2586
2587         canvas.addEventListener('mousedown', this.handleMouseDown);
2588         document.addEventListener('mousemove', this.handleMouseMove);
2589         document.addEventListener('mouseup', this.handleMouseUp);
2590     }
2591
2592     detachEvents() {
2593         const canvas = this.layerSystem.getActiveLayer().canvas;
2594
2595         canvas.removeEventListener('mousedown', this.handleMouseDown);
2596         document.removeEventListener('mousemove', this.handleMouseMove);
2597         document.removeEventListener('mouseup', this.handleMouseUp);
2598
2599         // Finish any in-progress drawing
2600         this.finishDrawing();
2601     }
2602
2603     handleMouseDown = (e) => {
2604         const coords = this.layerSystem.screenToCanvas(e.clientX, e.clientY);
2605
2606         this.center = coords;
2607
2608         // Create temporary ellipse
2609         this.currentEllipse = new EllipseObject({
2610             x: coords.x,
2611             y: coords.y,
2612             radiusX: 0,
2613             radiusY: 0,
2614             color: this.settings.color,
2615             lineWidth: this.settings.lineWidth,
2616             fill: this.settings.fill,
2617             fillColor: this.settings.fillColor
2618         });
2619
2620         this.drawing = true;
2621     }
```

```
2622         // Add to layer
2623         this.layerSystem.addObject(this.currentEllipse);
2624     }
2625
2626     handleMouseMove = (e) => {
2627         if (!this.drawing || !this.currentEllipse) return;
2628
2629         const coords = this.layerSystem.screenToCanvas(e.clientX, e.clientY);
2630
2631         // Update radii
2632         this.currentEllipse.radiusX = Math.abs(coords.x - this.center.x);
2633         this.currentEllipse.radiusY = Math.abs(coords.y - this.center.y);
2634
2635         // Redraw
2636         this.layerSystem.redrawLayer(this.layerSystem.activeLayerIndex);
2637     }
2638
2639     handleMouseUp = () => {
2640         this.finishDrawing();
2641     }
2642
2643     finishDrawing() {
2644         if (this.drawing && this.currentEllipse) {
2645             // Finish the ellipse
2646             this.drawing = false;
2647             this.center = null;
2648             this.currentEllipse = null;
2649         }
2650     }
2651 }
2652
2653 // Text tool
2654 class TextTool extends DrawingTool {
2655     constructor(toolSystem) {
2656         super(toolSystem);
2657         this.textInput = null;
2658     }
2659
2660     attachEvents() {
2661         const canvas = this.layerSystem.getActiveLayer().canvas;
2662
2663         canvas.addEventListener('click', this.handleClick);
2664     }
2665
2666     detachEvents() {
2667         const canvas = this.layerSystem.getActiveLayer().canvas;
2668
2669         canvas.removeEventListener('click', this.handleClick);
2670
2671         // Remove any active text input
```

```
2672     this.removeTextInput();
2673 }
2674
2675 handleClick = (e) => {
2676     const coords = this.layerSystem.screenToCanvas(e.clientX, e.clientY);
2677
2678     // Show text input at click position
2679     this.showTextInput(coords.x, coords.y);
2680 }
2681
2682 showTextInput(x, y) {
2683     // Remove any existing text input
2684     this.removeTextInput();
2685
2686     // Create text input element
2687     this.textInput = document.createElement('div');
2688     this.textInput.style.position = 'absolute';
2689     this.textInput.style.zIndex = '100';
2690
2691     // Position relative to canvas
2692     const canvasRect = this.layerSystem.container.getBoundingClientRect();
2693     this.textInput.style.left = (canvasRect.left + x) + 'px';
2694     this.textInput.style.top = (canvasRect.top + y) + 'px';
2695
2696     // Style the input
2697     this.textInput.style.backgroundColor = 'white';
2698     this.textInput.style.border = '1px solid #ccc';
2699     this.textInput.style.padding = '5px';
2700     this.textInput.style.borderRadius = '3px';
2701     this.textInput.style.boxShadow = '0 2px 5px rgba(0,0,0,0.1)';
2702
2703     // Create the actual input
2704     const input = document.createElement('input');
2705     input.type = 'text';
2706     input.placeholder = 'Enter text...';
2707     input.style.width = '200px';
2708     input.style.padding = '5px';
2709     input.style.border = '1px solid #ddd';
2710     input.style.borderRadius = '3px';
2711
2712     // Create font size selector
2713     const sizeSelect = document.createElement('select');
2714     [8, 10, 12, 14, 16, 18, 20, 24, 36].forEach(size => {
2715         const option = document.createElement('option');
2716         option.value = size;
2717         option.textContent = size + 'px';
2718         if (size === this.settings.fontSize) {
2719             option.selected = true;
2720         }
2721         sizeSelect.appendChild(option);
```

```
2722     });
2723
2724     sizeSelect.addEventListener('change', (e) => {
2725         this.settings.fontSize = parseInt(e.target.value);
2726     });
2727
2728     // Create alignment options
2729     const alignmentDiv = document.createElement('div');
2730     alignmentDiv.style.display = 'flex';
2731     alignmentDiv.style.marginTop = '5px';
2732
2733     ['left', 'center', 'right'].forEach(align => {
2734         const button = document.createElement('button');
2735         button.textContent = align[0].toUpperCase();
2736         button.style.flex = '1';
2737         button.style.padding = '2px 5px';
2738         button.style.backgroundColor = this.settings.textAlign === align ?
2739 '#4CAF50' : '#f1f1f1';
2740         button.style.color = this.settings.textAlign === align ? 'white' :
2741 'black';
2742
2743         button.addEventListener('click', () => {
2744             this.settings.textAlign = align;
2745             alignmentDiv.querySelectorAll('button').forEach(btn => {
2746                 btn.style.backgroundColor = '#f1f1f1';
2747                 btn.style.color = 'black';
2748             });
2749             button.style.backgroundColor = '#4CAF50';
2750             button.style.color = 'white';
2751         });
2752
2753         alignmentDiv.appendChild(button);
2754     });
2755
2756     // Add UI elements to the container
2757     this.textInput.appendChild(input);
2758     this.textInput.appendChild(document.createElement('br'));
2759     this.textInput.appendChild(document.createTextNode('Size: '));
2760     this.textInput.appendChild(sizeSelect);
2761     this.textInput.appendChild(document.createElement('br'));
2762     this.textInput.appendChild(alignmentDiv);
2763
2764     // Add buttons container
2765     const buttons = document.createElement('div');
2766     buttons.style.display = 'flex';
2767     buttons.style.marginTop = '5px';
2768     buttons.style.gap = '5px';
2769 
```

```
2770 // Add button
2771 const addButton = document.createElement('button');
2772 addButton.textContent = 'Add Text';
2773 addButton.style.flex = '1';
2774 addButton.style.padding = '5px';
2775 addButton.style.backgroundColor = '#4CAF50';
2776 addButton.style.color = 'white';
2777 addButton.style.border = 'none';
2778 addButton.style.borderRadius = '3px';
2779 addButton.style.cursor = 'pointer';
2780
2781 addButton.addEventListener('click', () => {
2782     this.addText(x, y, input.value);
2783 });
2784
2785 // Cancel button
2786 const cancelButton = document.createElement('button');
2787 cancelButton.textContent = 'Cancel';
2788 cancelButton.style.flex = '1';
2789 cancelButton.style.padding = '5px';
2790 cancelButton.style.backgroundColor = '#f44336';
2791 cancelButton.style.color = 'white';
2792 cancelButton.style.border = 'none';
2793 cancelButton.style.borderRadius = '3px';
2794 cancelButton.style.cursor = 'pointer';
2795
2796 cancelButton.addEventListener('click', () => {
2797     this.removeTextInput();
2798 });
2799
2800 buttons.appendChild(addButton);
2801 buttons.appendChild(cancelButton);
2802 this.textInput.appendChild(buttons);
2803
2804 // Add to document
2805 document.body.appendChild(this.textInput);
2806
2807 // Focus the input
2808 input.focus();
2809
2810 // Handle enter key
2811 input.addEventListener('keydown', (e) => {
2812     if (e.key === 'Enter') {
2813         this.addText(x, y, input.value);
2814     } else if (e.key === 'Escape') {
2815         this.removeTextInput();
2816     }
2817 });
2818 }
2819 }
```

```
2820     removeTextInput() {
2821         if (this.textInput && this.textInput.parentNode) {
2822             this.textInput.parentNode.removeChild(this.textInput);
2823             this.textInput = null;
2824         }
2825     }
2826
2827     addText(x, y, text) {
2828         if (!text.trim()) {
2829             this.removeTextInput();
2830             return;
2831         }
2832
2833         // Create text object
2834         const textObj = new TextObject({
2835             x: x,
2836             y: y,
2837             text: text,
2838             color: this.settings.color,
2839             fontSize: this.settings.fontSize,
2840             fontFamily: this.settings.fontFamily,
2841             align: this.settings.textAlign
2842         });
2843
2844         // Add to layer
2845         this.layerSystem.addObject(textObj);
2846
2847         // Remove text input
2848         this.removeTextInput();
2849     }
2850 }
2851
2852 // Image tool
2853 class ImageTool extends DrawingTool {
2854     constructor(toolSystem) {
2855         super(toolSystem);
2856         this.fileInput = null;
2857     }
2858
2859     attachEvents() {
2860         const canvas = this.layerSystem.getActiveLayer().canvas;
2861
2862         canvas.addEventListener('click', this.handleClick);
2863     }
2864
2865     detachEvents() {
2866         const canvas = this.layerSystem.getActiveLayer().canvas;
2867
2868         canvas.removeEventListener('click', this.handleClick);
2869     }
}
```

```
2870
2871     handleClick = (e) => {
2872         // Show file upload dialog
2873         this.showFileDialog();
2874     }
2875
2876     showFileDialog() {
2877         // Create hidden file input if it doesn't exist
2878         if (!this.fileInput) {
2879             this.fileInput = document.createElement('input');
2880             this.fileInput.type = 'file';
2881             this.fileInput.accept = 'image/*';
2882             this.fileInput.style.display = 'none';
2883             document.body.appendChild(this.fileInput);
2884
2885             this.fileInput.addEventListener('change', (e) => {
2886                 if (e.target.files && e.target.files[0]) {
2887                     this.handleFileSelect(e.target.files[0]);
2888                 }
2889             });
2890         }
2891
2892         // Trigger file dialog
2893         this.fileInput.click();
2894     }
2895
2896     handleFileSelect(file) {
2897         // Read the file and create a data URL
2898         const reader = new FileReader();
2899
2900         reader.onload = (e) => {
2901             const imageUrl = e.target.result;
2902
2903             // Show image placement UI
2904             this.showImagePlacementUI(imageUrl);
2905         };
2906
2907         reader.readAsDataURL(file);
2908     }
2909
2910     showImagePlacementUI(imageUrl) {
2911         // Create a preview image to get dimensions
2912         const img = new Image();
2913
2914         img.onload = () => {
2915             // Calculate dimensions (max size 300px width/height while maintaining
aspect ratio)
2916             let width = img.width;
2917             let height = img.height;
2918 }
```

```
2919 |     const maxSize = 300;
2920 |     if (width > maxSize || height > maxSize) {
2921 |         if (width > height) {
2922 |             height = (height / width) * maxSize;
2923 |             width = maxSize;
2924 |         } else {
2925 |             width = (width / height) * maxSize;
2926 |             height = maxSize;
2927 |         }
2928 |     }
2929 |
2930 |     // Create placement UI
2931 |     const placementUI = document.createElement('div');
2932 |     placementUI.style.position = 'fixed';
2933 |     placementUI.style.top = '50%';
2934 |     placementUI.style.left = '50%';
2935 |     placementUI.style.transform = 'translate(-50%, -50%)';
2936 |     placementUI.style.backgroundColor = 'white';
2937 |     placementUI.style.padding = '20px';
2938 |     placementUI.style.borderRadius = '5px';
2939 |     placementUI.style.boxShadow = '0 0 10px rgba(0,0,0,0.3)';
2940 |     placementUI.style.zIndex = '1000';
2941 |
2942 |     // Add heading
2943 |     const heading = document.createElement('h3');
2944 |     heading.textContent = 'Place Image';
2945 |     heading.style.margin = '0 0 10px 0';
2946 |
2947 |     // Add image preview
2948 |     const preview = document.createElement('img');
2949 |     preview.src = imageUrl;
2950 |     preview.style.maxWidth = '300px';
2951 |     preview.style.maxHeight = '300px';
2952 |     preview.style.display = 'block';
2953 |     preview.style.marginBottom = '10px';
2954 |
2955 |     // Size controls
2956 |     const sizeControls = document.createElement('div');
2957 |     sizeControls.style.marginBottom = '10px';
2958 |
2959 |     const widthLabel = document.createElement('label');
2960 |     widthLabel.textContent = 'Width: ';
2961 |     const widthInput = document.createElement('input');
2962 |     widthInput.type = 'number';
2963 |     widthInput.value = Math.round(width);
2964 |     widthInput.style.width = '60px';
2965 |
2966 |     const heightLabel = document.createElement('label');
2967 |     heightLabel.textContent = 'Height: ';
2968 |     heightLabel.style.marginLeft = '10px';
```

```
2969 const heightInput = document.createElement('input');
2970 heightInput.type = 'number';
2971 heightInput.value = Math.round(height);
2972 heightInput.style.width = '60px';
2973
2974 // Maintain aspect ratio
2975 const aspectRatio = img.width / img.height;
2976
2977 widthInput.addEventListener('input', () => {
2978     const newWidth = parseInt(widthInput.value);
2979     if (!isNaN(newWidth)) {
2980         heightInput.value = Math.round(newWidth / aspectRatio);
2981     }
2982 });
2983
2984 heightInput.addEventListener('input', () => {
2985     const newHeight = parseInt(heightInput.value);
2986     if (!isNaN(newHeight)) {
2987         widthInput.value = Math.round(newHeight * aspectRatio);
2988     }
2989 });
2990
2991 sizeControls.appendChild(widthLabel);
2992 sizeControls.appendChild(widthInput);
2993 sizeControls.appendChild(heightLabel);
2994 sizeControls.appendChild(heightInput);
2995
2996 // Buttons container
2997 const buttons = document.createElement('div');
2998 buttons.style.display = 'flex';
2999 buttons.style.justifyContent = 'space-between';
3000 buttons.style.marginTop = '15px';
3001
3002 // Place button
3003 const placeButton = document.createElement('button');
3004 placeButton.textContent = 'Place Image';
3005 placeButton.style.padding = '8px 15px';
3006 placeButton.style.backgroundColor = '#4CAF50';
3007 placeButton.style.color = 'white';
3008 placeButton.style.border = 'none';
3009 placeButton.style.borderRadius = '4px';
3010 placeButton.style.cursor = 'pointer';
3011
3012 placeButton.addEventListener('click', () => {
3013     // Get center of view as placement position
3014     const containerRect =
3015         this.layerSystem.container.getBoundingClientRect();
3016     const x = containerRect.width / 2;
3017     const y = containerRect.height / 2;
```

```
3018     // Get dimensions from inputs
3019     const finalWidth = parseInt(widthInput.value);
3020     const finalHeight = parseInt(heightInput.value);
3021
3022     // Create and add image object
3023     const imageObj = new ImageObject({
3024         x: x - finalWidth / 2,
3025         y: y - finalHeight / 2,
3026         width: finalWidth,
3027         height: finalHeight,
3028         imageUrl: imageUrl
3029     });
3030
3031     this.layerSystem.addObject(imageObj);
3032
3033     // Remove UI
3034     document.body.removeChild(placementUI);
3035 );
3036
3037     // Cancel button
3038     const cancelButton = document.createElement('button');
3039     cancelButton.textContent = 'Cancel';
3040     cancelButton.style.padding = '8px 15px';
3041     cancelButton.style.backgroundColor = '#f44336';
3042     cancelButton.style.color = 'white';
3043     cancelButton.style.border = 'none';
3044     cancelButton.style.borderRadius = '4px';
3045     cancelButton.style.cursor = 'pointer';
3046
3047     cancelButton.addEventListener('click', () => {
3048         document.body.removeChild(placementUI);
3049     });
3050
3051     buttons.appendChild(cancelButton);
3052     buttons.appendChild(placeButton);
3053
3054     // Assemble UI
3055     placementUI.appendChild(heading);
3056     placementUI.appendChild(preview);
3057     placementUI.appendChild(sizeControls);
3058     placementUI.appendChild(buttons);
3059
3060     // Add to document
3061     document.body.appendChild(placementUI);
3062 );
3063
3064     img.src = imageUrl;
3065 }
3066
3067 }
```

```
3068 // Symbol tool
3069 class SymbolTool extends DrawingTool {
3070     constructor(toolSystem) {
3071         super(toolSystem);
3072         this.symbolSelector = null;
3073     }
3074
3075     attachEvents() {
3076         const canvas = this.layerSystem.getActiveLayer().canvas;
3077
3078         canvas.addEventListener('click', this.handleClick);
3079     }
3080
3081     detachEvents() {
3082         const canvas = this.layerSystem.getActiveLayer().canvas;
3083
3084         canvas.removeEventListener('click', this.handleClick);
3085
3086         // Remove any active symbol selector
3087         this.removeSymbolSelector();
3088     }
3089
3090     handleClick = (e) => {
3091         const coords = this.layerSystem.screenToCanvas(e.clientX, e.clientY);
3092
3093         // Show symbol selector
3094         this.showSymbolSelector(coords.x, coords.y);
3095     }
3096
3097     showSymbolSelector(x, y) {
3098         // Remove existing selector if any
3099         this.removeSymbolSelector();
3100
3101         // Create symbol selector
3102         this.symbolSelector = document.createElement('div');
3103         this.symbolSelector.style.position = 'absolute';
3104         this.symbolSelector.style.zIndex = '100';
3105
3106         // Position relative to canvas
3107         const canvasRect = this.layerSystem.container.getBoundingClientRect();
3108         this.symbolSelector.style.left = (canvasRect.left + x + 10) + 'px';
3109         this.symbolSelector.style.top = (canvasRect.top + y + 10) + 'px';
3110
3111         // Style
3112         this.symbolSelector.style.backgroundColor = 'white';
3113         this.symbolSelector.style.border = '1px solid #ccc';
3114         this.symbolSelector.style.padding = '10px';
3115         this.symbolSelector.style.borderRadius = '5px';
3116         this.symbolSelector.style.boxShadow = '0 2px 10px rgba(0,0,0,0.1)';
3117     }
```

```
3118     // Add title
3119     const title = document.createElement('h4');
3120     title.textContent = 'Select Animation Symbol';
3121     title.style.margin = '0 0 10px 0';
3122     this.symbolSelector.appendChild(title);
3123
3124     // Symbol grid
3125     const symbolGrid = document.createElement('div');
3126     symbolGrid.style.display = 'grid';
3127     symbolGrid.style.gridTemplateColumns = 'repeat(3, 1fr)';
3128     symbolGrid.style.gap = '10px';
3129     symbolGrid.style.marginBottom = '10px';
3130
3131     // Available symbols
3132     const symbols = [
3133         { type: 'anticipation', name: 'Anticipation' },
3134         { type: 'impact', name: 'Impact' },
3135         { type: 'keyframe', name: 'Keyframe' },
3136         { type: 'inbetween', name: 'Inbetween' },
3137         { type: 'hold', name: 'Hold' },
3138         { type: 'default', name: 'Default' }
3139     ];
3140
3141     // Create symbol preview for each type
3142     symbols.forEach(symbol => {
3143         const symbolItem = document.createElement('div');
3144         symbolItem.style.display = 'flex';
3145         symbolItem.style.flexDirection = 'column';
3146         symbolItem.style.alignItems = 'center';
3147         symbolItem.style.cursor = 'pointer';
3148         symbolItem.style.padding = '5px';
3149         symbolItem.style.border = '1px solid #ddd';
3150         symbolItem.style.borderRadius = '3px';
3151
3152         // Create canvas for symbol preview
3153         const canvas = document.createElement('canvas');
3154         canvas.width = 50;
3155         canvas.height = 50;
3156         const ctx = canvas.getContext('2d');
3157
3158         // Draw symbol preview
3159         ctx.save();
3160         ctx.strokeStyle = this.settings.color;
3161         ctx.fillStyle = this.settings.color;
3162         ctx.lineWidth = 2;
3163
3164         // Draw centered preview
3165         const symbolObj = new SymbolObject({
3166             x: 25,
3167             y: 25,
```

```
3168     symbolType: symbol.type,
3169     color: this.settings.color,
3170     lineWidth: 2,
3171     scale: 1.5
3172   });
3173 
3174   symbolObj.draw(ctx);
3175   ctx.restore();
3176 
3177   const name = document.createElement('div');
3178   name.textContent = symbol.name;
3179   name.style.marginTop = '5px';
3180   name.style.fontSize = '12px';
3181 
3182   symbolItem.appendChild(canvas);
3183   symbolItem.appendChild(name);
3184 
3185   // Add click handler
3186   symbolItem.addEventListener('click', () => {
3187     this.addSymbol(x, y, symbol.type);
3188     this.removeSymbolSelector();
3189   });
3190 
3191   symbolGrid.appendChild(symbolItem);
3192 });
3193 
3194 this.symbolSelector.appendChild(symbolGrid);
3195 
3196 // Scale control
3197 const scaleContainer = document.createElement('div');
3198 scaleContainer.style.display = 'flex';
3199 scaleContainer.style.alignItems = 'center';
3200 scaleContainer.style.marginBottom = '10px';
3201 
3202 const scaleLabel = document.createElement('label');
3203 scaleLabel.textContent = 'Scale: ';
3204 
3205 const scaleInput = document.createElement('input');
3206 scaleInput.type = 'range';
3207 scaleInput.min = '0.5';
3208 scaleInput.max = '3';
3209 scaleInput.step = '0.1';
3210 scaleInput.value = this.settings.symbolScale;
3211 scaleInput.style.flex = '1';
3212 scaleInput.style.marginLeft = '5px';
3213 
3214 const scaleValue = document.createElement('span');
3215 scaleValue.textContent = this.settings.symbolScale + 'x';
3216 scaleValue.style.marginLeft = '5px';
3217 scaleValue.style.width = '30px';
```

```
3218
3219     scaleInput.addEventListener('input', () => {
3220         this.settings.symbolScale = parseFloat(scaleInput.value);
3221         scaleValue.textContent = this.settings.symbolScale + 'x';
3222     });
3223
3224     scaleContainer.appendChild(scaleLabel);
3225     scaleContainer.appendChild(scaleInput);
3226     scaleContainer.appendChild(scaleValue);
3227
3228     this.symbolSelector.appendChild(scaleContainer);
3229
3230     // Add cancel button
3231     const cancelButton = document.createElement('button');
3232     cancelButton.textContent = 'Cancel';
3233     cancelButton.style.width = '100%';
3234     cancelButton.style.padding = '5px';
3235     cancelButton.style.backgroundColor = '#f44336';
3236     cancelButton.style.color = 'white';
3237     cancelButton.style.border = 'none';
3238     cancelButton.style.borderRadius = '3px';
3239     cancelButton.style.cursor = 'pointer';
3240
3241     cancelButton.addEventListener('click', () => {
3242         this.removeSymbolSelector();
3243     });
3244
3245     this.symbolSelector.appendChild(cancelButton);
3246
3247     // Add to document
3248     document.body.appendChild(this.symbolSelector);
3249 }
3250
3251 removeSymbolSelector() {
3252     if (this.symbolSelector && this.symbolSelector.parentNode) {
3253         this.symbolSelector.parentNode.removeChild(this.symbolSelector);
3254         this.symbolSelector = null;
3255     }
3256 }
3257
3258 addSymbol(x, y, symbolType) {
3259     // Create symbol object
3260     const symbolObj = new SymbolObject({
3261         x: x,
3262         y: y,
3263         symbolType: symbolType,
3264         color: this.settings.color,
3265         lineWidth: this.settings.lineWidth,
3266         scale: this.settings.symbolScale
3267     });
3268 }
```

```
3268
3269      // Add to layer
3270      this.layerSystem.addObject(symbolObj);
3271  }
3272 }
3273
3274 // Frame-spanning line tool (for connecting cells across frames)
3275 class FrameSpanningLineTool extends DrawingTool {
3276     constructor(toolSystem) {
3277         super(toolSystem);
3278         this.startCell = null;
3279         this.currentLine = null;
3280         this.drawing = false;
3281     }
3282
3283     attachEvents() {
3284         // This tool works with the table cells, not the canvas
3285         this.setupCellEvents();
3286     }
3287
3288     detachEvents() {
3289         this.removeCellEvents();
3290
3291         // Finish any in-progress drawing
3292         this.finishDrawing();
3293     }
3294
3295     setupCellEvents() {
3296         // Find all table cells and add mousedown handler
3297         const cells = document.querySelectorAll('#xsheet-table td');
3298         cells.forEach(cell => {
3299             cell.addEventListener('mousedown', this.handleCellMouseDown);
3300             cell.addEventListener('mouseup', this.handleCellMouseUp);
3301
3302             // Add hover effect
3303             cell.addEventListener('mouseover', this.handleCellHover);
3304             cell.addEventListener('mouseout', this.handleCellOut);
3305
3306             // Store original background for hover effect
3307             if (!cell.dataset.originalBg) {
3308                 cell.dataset.originalBg = cell.style.backgroundColor || '';
3309             }
3310         });
3311     }
3312
3313     removeCellEvents() {
3314         const cells = document.querySelectorAll('#xsheet-table td');
3315         cells.forEach(cell => {
3316             cell.removeEventListener('mousedown', this.handleCellMouseDown);
3317             cell.removeEventListener('mouseup', this.handleCellMouseUp);
```

```
3318     cell.removeEventListener('mouseover', this.handleCellHover);
3319     cell.removeEventListener('mouseout', this.handleCellOut);
3320
3321         // Restore original background
3322         if (cell.dataset.originalBg) {
3323             cell.style.backgroundColor = cell.dataset.originalBg;
3324         }
3325     });
3326 }
3327
3328 handleCellHover = (e) => {
3329     const cell = e.currentTarget;
3330     cell.style.backgroundColor = 'rgba(0, 123, 255, 0.2)';
3331 }
3332
3333 handleCellOut = (e) => {
3334     const cell = e.currentTarget;
3335     if (!this.drawing || cell !== this.startCell) {
3336         cell.style.backgroundColor = cell.dataset.originalBg || '';
3337     }
3338 }
3339
3340 handleCellMouseDown = (e) => {
3341     const cell = e.currentTarget;
3342
3343         // Find the row and column index
3344         const row = cell.parentElement;
3345         const frameNumber = parseInt(row.getAttribute('data-frame') || row.className.replace('frame-', ''));
```

```
3346             const columnIndex = Array.from(row.children).indexOf(cell) + 1;
3347
3348             if (isNaN(frameNumber) || frameNumber <= 0) return;
3349
3350             // Store as start cell
3351             this.startCell = cell;
3352             this.drawing = true;
3353
3354             // Highlight the cell
3355             cell.style.backgroundColor = 'rgba(0, 123, 255, 0.4)';
3356
3357             // Create temporary frame-spanning line object
3358             this.currentLine = new FrameSpanningLineObject({
3359                 startFrame: frameNumber,
3360                 startColumn: columnIndex,
3361                 endFrame: frameNumber,
3362                 endColumn: columnIndex,
3363                 color: this.settings.color,
3364                 lineWidth: this.settings.lineWidth,
3365                 arrowEnd: true
3366             });
3367 }
```

```
3367
3368      // Add to layer
3369      this.layerSystem.addObject(this.currentLine);
3370  }
3371
3372  handleCellMouseUp = (e) => {
3373    if (!this.drawing || !this.currentLine) return;
3374
3375    const cell = e.currentTarget;
3376
3377    // Skip if this is the same cell
3378    if (cell === this.startCell) {
3379      this.finishDrawing();
3380      return;
3381    }
3382
3383    // Find the row and column index
3384    const row = cell.parentElement;
3385    const frameNumber = parseInt(row.getAttribute('data-frame') ||
row.className.replace('frame-', ''));  

3386    const columnIndex = Array.from(row.children).indexOf(cell) + 1;
3387
3388    if (isNaN(frameNumber) || frameNumber <= 0) {
3389      this.finishDrawing();
3390      return;
3391    }
3392
3393    // Update the end point of the line
3394    this.currentLine.endFrame = frameNumber;
3395    this.currentLine.endColumn = columnIndex;
3396
3397    // Redraw
3398    this.layerSystem.redrawLayer(this.layerSystem.activeLayerIndex);
3399
3400    // Finish drawing
3401    this.finishDrawing();
3402  }
3403
3404  finishDrawing() {
3405    if (this.drawing) {
3406      // Reset flags
3407      this.drawing = false;
3408
3409      // If the line is too short (same start and end), remove it
3410      if (this.currentLine &&
3411          this.currentLine.startFrame === this.currentLine.endFrame &&
3412          this.currentLine.startColumn === this.currentLine.endColumn) {
3413        this.layerSystem.removeObject(this.currentLine);
3414      }
3415    }
```

```
3416         // Clear current line reference
3417         this.currentLine = null;
3418
3419         // Reset cell highlights
3420         if (this.startCell) {
3421             this.startCell.style.backgroundColor =
3422             this.startCell.dataset.originalBg || '';
3423             this.startCell = null;
3424         }
3425     }
3426 }
3427
3428 // Eraser tool
3429 class EraserTool extends DrawingTool {
3430     constructor(toolSystem) {
3431         super(toolSystem);
3432     }
3433
3434     attachEvents() {
3435         const canvas = this.layerSystem.getActiveLayer().canvas;
3436
3437         canvas.addEventListener('mousedown', this.handleMouseDown);
3438     }
3439
3440     detachEvents() {
3441         const canvas = this.layerSystem.getActiveLayer().canvas;
3442
3443         canvas.removeEventListener('mousedown', this.handleMouseDown);
3444     }
3445
3446     handleMouseDown = (e) => {
3447         const coords = this.layerSystem.screenToCanvas(e.clientX, e.clientY);
3448
3449         // Find object at click position
3450         const hit = this.layerSystem.findObjectAt(coords.x, coords.y);
3451
3452         if (hit) {
3453             // Remove the object
3454             this.layerSystem.removeObject(hit.object, hit.layerIndex);
3455         }
3456     }
3457 }
3458
3459 /**
3460 * INTEGRATION WITH X-SHEET APPLICATION
3461 * Initialize the drawing system, save/load integration, and PDF/print handling
3462 */
3463
3464 // Initialize drawing system when document is loaded
```

```
3465 function initDrawingSystem() {
3466     // Setup after the table is rendered
3467     setTimeout(() => {
3468         // Get the X-Sheet table element
3469         const xsheetTable = document.getElementById('xsheet-table');
3470         if (!xsheetTable) {
3471             console.error('X-Sheet table not found');
3472             return;
3473         }
3474
3475         // Create drawing systems
3476         const drawingLayerSystem = new DrawingLayerSystem(xsheetTable);
3477         const drawingToolSystem = new DrawingToolSystem(drawingLayerSystem);
3478
3479         // Store global reference
3480         window.xsheetDrawing = {
3481             layerSystem: drawingLayerSystem,
3482             toolSystem: drawingToolSystem
3483         };
3484
3485         // Add to status message
3486         const statusElement = document.getElementById('status-message');
3487         if (statusElement) {
3488             statusElement.textContent = 'Drawing tools initialized';
3489         }
3490
3491         // Integrate with X-Sheet save/load system
3492         integrateWithXSheetSaveLoad();
3493
3494         // Integrate with X-Sheet PDF and printing
3495         integrateWithXSheetExport();
3496
3497         // Custom event for when drawing objects change
3498         document.addEventListener('xsheet-redraw', function() {
3499             drawingLayerSystem.redrawAll();
3500         });
3501
3502         console.log('Drawing tools initialized successfully');
3503     }, 500);
3504 }
3505
3506 // Integrate drawing data with X-Sheet save/load system
3507 function integrateWithXSheetSaveLoad() {
3508     // Store original functions
3509     const originalCollectData = window.collectData;
3510     const originalRestoreData = window.restoreData;
3511
3512     // Override collectData to include drawings
3513     window.collectData = function() {
3514         // Call original function to get base data
```

```
3515 |     const data = originalCollectData ? originalCollectData() : {};
3516 |
3517 |     // Add drawing data if drawing system is initialized
3518 |     if (window.xsheetDrawing && window.xsheetDrawing.layerSystem) {
3519 |         data.drawingLayers = [];
3520 |
3521 |         // Collect objects from each layer
3522 |         window.xsheetDrawing.layerSystem.layers.forEach(layer => {
3523 |             const layerData = {
3524 |                 name: layer.name,
3525 |                 visible: layer.visible,
3526 |                 objects: layer.objects.map(obj => obj.toJSON())
3527 |             };
3528 |
3529 |             data.drawingLayers.push(layerData);
3530 |         });
3531 |     }
3532 |
3533 |     return data;
3534 | };
3535 |
3536 | // Override restoreData to handle drawings
3537 | window.restoreData = function(data) {
3538 |     // Call original function to restore base data
3539 |     if (originalRestoreData) {
3540 |         originalRestoreData(data);
3541 |     }
3542 |
3543 |     // Restore drawing data if available
3544 |     if (data.drawingLayers && window.xsheetDrawing &&
3545 |         window.xsheetDrawing.layerSystem) {
3546 |         const layerSystem = window.xsheetDrawing.layerSystem;
3547 |
3548 |         // Clear existing layers
3549 |         layerSystem.clearAllLayers();
3550 |
3551 |         // Restore each layer
3552 |         data.drawingLayers.forEach((layerData, index) => {
3553 |             // Create layer if needed
3554 |             if (index >= layerSystem.layers.length) {
3555 |                 layerSystem.addLayer(layerData.name);
3556 |             } else {
3557 |                 layerSystem.layers[index].name = layerData.name;
3558 |                 layerSystem.layers[index].visible = layerData.visible;
3559 |             }
3560 |
3561 |             // Restore objects
3562 |             layerData.objects.forEach(objData => {
3563 |                 const newObj = DrawingObjectFactory.createFromJSON(objData);
3564 |                 if (newObj) {
```

```

3564                     layerSystem.layers[index].objects.push(newObj);
3565                 }
3566             });
3567         });
3568     }

3569         // Redraw
3570         layerSystem.redrawAll();
3571     }
3572   };
3573 }
3574

3575 // Integrate drawing layers with PDF export and printing
3576 function integrateWithXSheetExport() {
3577     // Store original functions
3578     const originalExportToPDF = window.exportToPDF;
3579     const originalPrintSheet = window.printSheet;

3580

3581     // Override PDF export to include drawings
3582     window.exportToPDF = function () {
3583         if (!window.xsheetDrawing || !window.xsheetDrawing.layerSystem) {
3584             return originalExportToPDF ? originalExportToPDF() : null;
3585         }
3586

3587         // Save current state
3588         const layerSystem = window.xsheetDrawing.layerSystem;
3589         const layerContainer = layerSystem.container;

3590

3591         // 1) Save original CSS & DOM properties
3592         const originalDisplay = layerContainer.style.display;
3593         const originalPos = layerContainer.style.position;
3594         const originalTop = layerContainer.style.top;
3595         const originalLeft = layerContainer.style.left;
3596         const originalZIndex = layerContainer.style.zIndex;
3597         const originalWidth = layerContainer.style.width;
3598         const originalHeight = layerContainer.style.height;
3599         const originalParent = layerContainer.parentNode;

3600

3601         // 2) Make it visible for capture
3602         layerContainer.style.display = 'block';

3603

3604         // 3) Compute offsets - FIXED POSITIONING LOGIC
3605         const printableArea = document.getElementById('printable-area');
3606         const printableRect = printableArea.getBoundingClientRect();
3607         const tableRect = document.getElementById('xsheet-
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`;
3616     layerContainer.style.width = `${tableRect.width}px`;
3617     layerContainer.style.height = `${tableRect.height}px`;
3618     layerContainer.style.zIndex = '1000';
3619
3620     // Force a redraw of all drawing objects
3621     layerSystem.redrawAll();
3622
3623     // 6) Wait a brief moment to ensure drawings are rendered
3624     setTimeout(() => {
3625         // 7) Generate the PDF
3626         const result = originalExportToPDF ? originalExportToPDF() : null;
3627
3628         // 8) Restore everything
3629         layerContainer.style.display = originalDisplay;
3630         layerContainer.style.position = originalPos;
3631         layerContainer.style.top = originalTop;
3632         layerContainer.style.left = originalLeft;
3633         layerContainer.style.zIndex = originalZIndex;
3634         layerContainer.style.width = originalWidth;
3635         layerContainer.style.height = originalHeight;
3636         originalParent.appendChild(layerContainer);
3637     }, 50);
3638
3639     return true; // Will be async due to setTimeout
3640 };
3641
3642 // Override print to include drawings
3643 window.printSheet = function () {
3644     // 1) Fallback if no drawing system
3645     if (!window.xsheetDrawing || !window.xsheetDrawing.layerSystem) {
3646         return originalPrintSheet ? originalPrintSheet() : null;
3647     }
3648
3649     // 2) Grab the container and save its original state
3650     const layerSystem = window.xsheetDrawing.layerSystem;
3651     const layerContainer = layerSystem.container;
3652     const originalDisplay = layerContainer.style.display;
3653     const originalPos = layerContainer.style.position;
3654     const originalTop = layerContainer.style.top;
3655     const originalLeft = layerContainer.style.left;
3656     const originalZIndex = layerContainer.style.zIndex;
3657     const originalWidth = layerContainer.style.width;
3658     const originalHeight = layerContainer.style.height;
3659     const originalParent = layerContainer.parentNode;
3660
3661     // 3) Make it visible for print capture
3662     layerContainer.style.display = 'block';
```

```
3663
3664      // 4) Compute offsets relative to the sheet - FIXED POSITIONING LOGIC
3665      const printableArea = document.getElementById('printable-area');
3666      const printableRect = printableArea.getBoundingClientRect();
3667      const tableRect = document.getElementById('xsheet-
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
3707      // Setup event listeners for table updates
3708      function setupXSheetUpdateHandling() {
3709          // Hook into existing update functions
3710          if (typeof window.generateTable === 'function') {
```

```
3711     const originalGenerateTable = window.generateTable;
3712     window.generateTable = function() {
3713         // Call original function
3714         const result = originalGenerateTable.apply(this, arguments);
3715
3716         // Fire update event
3717         document.dispatchEvent(new Event('xsheet-updated'));
3718
3719         return result;
3720     };
3721 }
3722
3723 if (typeof window.addEightRows === 'function') {
3724     const originalAddEightRows = window.addEightRows;
3725     window.addEightRows = function() {
3726         // Call original function
3727         const result = originalAddEightRows.apply(this, arguments);
3728
3729         // Fire update event
3730         document.dispatchEvent(new Event('xsheet-updated'));
3731
3732         return result;
3733     };
3734 }
3735
3736 if (typeof window.updateTemplate === 'function') {
3737     const originalUpdateTemplate = window.updateTemplate;
3738     window.updateTemplate = function() {
3739         // Call original function
3740         const result = originalUpdateTemplate.apply(this, arguments);
3741
3742         // Fire update event
3743         document.dispatchEvent(new Event('xsheet-updated'));
3744
3745         return result;
3746     };
3747 }
3748 }
3749
3750 // Original X-Sheet code
3751 // Wait for page load and libraries to initialize
3752 document.addEventListener('DOMContentLoaded', function () {
3753     // Global variables
3754     let frameCount = 96;
3755     let projectName = 'Animation_XSheet_' + new Date().toISOString().split('T')[0];
3756     let modified = false;
3757     let currentTemplate = 'large';
3758
3759     // Audio variables
3760     let audioContext = null;
```

```
3761 let audioBuffer = null;
3762 let audioSource = null;
3763 let audioData = null;
3764 let isPlaying = false;
3765 let startTime = 0;
3766 let startOffset = 0;
3767 let audioFileName = '';
3768 let waveformData = [];
3769 let waveformCanvases = [];
3770 let phonetics = [];
3771 let frameDuration = 1 / 24; // 24fps
3772 let currentFrame = 0;
3773 let phoneticEditPosition = null;
3774
3775 // Initialize UI elements
3776 const templateSelector = document.getElementById('template-selector');
3777 const frameCountInput = document.getElementById('frame-count');
3778 const saveButton = document.getElementById('save-button');
3779 const loadButton = document.getElementById('load-button');
3780 const pdfButton = document.getElementById('pdf-button');
3781 const printButton = document.getElementById('print-button');
3782 const addRowsButton = document.getElementById('add-rows-button');
3783 const clearButton = document.getElementById('clear-button');
3784 const tableBody = document.getElementById('xsheet-body');
3785 const statusMessage = document.getElementById('status-message');
3786 const audioButton = document.getElementById('audio-button');
3787 const audioUpload = document.getElementById('audio-upload');
3788 const playAudioButton = document.getElementById('play-audio');
3789 const stopAudioButton = document.getElementById('stop-audio');
3790 const audioScrubber = document.getElementById('audio-scrubber');
3791 const audioInfo = document.getElementById('audio-info');
3792 const addPhoneticButton = document.getElementById('add-phonetic');
3793 const phoneticInput = document.getElementById('phonetic-input');
3794 const phoneticText = document.getElementById('phonetic-text');
3795 const savePhoneticButton = document.getElementById('save-phonetic');
3796 const cancelPhoneticButton = document.getElementById('cancel-phonetic');
3797
3798 // Date field - set today by default
3799 const dateField = document.getElementById('project-date');
3800 dateField.valueAsDate = new Date();
3801
3802 // Initialize the table
3803 generateTable(frameCount);
3804
3805 // Initialize the drawing system
3806 initDrawingSystem();
3807 setupXSheetUpdateHandling();
3808
3809 // Event Listeners
3810 templateSelector.addEventListener('change', function () {
```

```
3811     currentTemplate = this.value;
3812     updateTemplate();
3813 });
3814
3815 frameCountInput.addEventListener('change', function () {
3816     frameCount = parseInt(this.value);
3817     if (frameCount < 8) frameCount = 8;
3818     generateTable(frameCount);
3819     updateStatusMessage('Frame count updated to ' + frameCount);
3820
3821     // Re-render waveform if audio is loaded
3822     if (audioBuffer) {
3823         renderWaveform();
3824     }
3825 });
3826
3827 saveButton.addEventListener('click', saveProject);
3828 loadButton.addEventListener('click', loadProject);
3829 pdfButton.addEventListener('click', exportToPDF);
3830 printButton.addEventListener('click', printSheet);
3831 addRowsButton.addEventListener('click', addEightRows);
3832 clearButton.addEventListener('click', clearSheet);
3833
3834 // Audio related event listeners
3835 audioButton.addEventListener('click', function () {
3836     audioUpload.click();
3837 });
3838
3839 audioUpload.addEventListener('change', function (e) {
3840     if (e.target.files.length > 0) {
3841         const file = e.target.files[0];
3842         audioFileName = file.name;
3843         loadAudioFile(file);
3844     }
3845 });
3846
3847 playAudioButton.addEventListener('click', togglePlayAudio);
3848 stopAudioButton.addEventListener('click', stopAudio);
3849 audioScrubber.addEventListener('input', scrubAudio);
3850
3851 addPhoneticButton.addEventListener('click', function () {
3852     if (audioBuffer) {
3853         showPhoneticInput(null);
3854     } else {
3855         updateStatusMessage('Please load audio first');
3856     }
3857 });
3858
3859 savePhoneticButton.addEventListener('click', savePhoneticMarker);
3860 cancelPhoneticButton.addEventListener('click', function () {
```

```
3861     phoneticInput.style.display = 'none';
3862 });
3863
3864 // Monitor for changes to set modified flag
3865 document.addEventListener('input', function (e) {
3866     if (!e.target.matches('input, [contenteditable="true"]')) return;
3867     modified = true;
3868
3869     if (e.target.matches('[contenteditable="true"]')) {
3870         // Add the modified class when content is added
3871         e.target.classList.add('modified');
3872
3873         // Remove the modified class when the cell is empty
3874         if (e.target.textContent.trim() === '') {
3875             e.target.classList.remove('modified');
3876         }
3877     }
3878
3879     updateStatusMessage('Changes detected - not saved');
3880 });
3881
3882 // Variables for cell selection
3883 let selectedCells = [];
3884 let isSelecting = false;
3885 let selectionStart = null;
3886 let hasMovedDuringSelection = false;
3887
3888 // Set up multi-cell selection functionality
3889 function setupCellSelection() {
3890     const editableCells = document.querySelectorAll('[contenteditable="true"]');
3891
3892     // Clear selected cells when clicking outside
3893     document.addEventListener('click', function (e) {
3894         // If this was part of a drag operation, skip the click handler
3895         if (hasMovedDuringSelection) {
3896             // We removed this line to fix the selection issue
3897             // hasMovedDuringSelection = false;
3898             return;
3899         }
3900
3901         // Get the clicked element and check if it's inside an editable cell or
3902         // a selected cell
3903         const clickedCell = e.target.closest('[contenteditable="true"]');
3904         const clickedInSelection = e.target.closest('.selected-cell');
3905
3906         // If not clicking in a cell or selection and not using modifier keys
3907         if (!clickedCell && !clickedInSelection && !e.ctrlKey && !e.metaKey) {
3908             clearCellSelection();
3909         }
3910     });
3911 }
```

```
3910
3911    // Handle clicks on the table for better cell focusing
3912    document.getElementById('xsheet-table').addEventListener('click', function
3913      (e) {
3914        // Find the closest editable cell from the click point
3915        const targetCell = e.target.closest('[contenteditable="true"]');
3916
3917        // If we found a cell and we're not in multi-select mode
3918        if (targetCell && !hasMovedDuringSelection && !e.ctrlKey && !e.metaKey)
3919        {
3920          // Focus the cell and position cursor at the end
3921          if (selectedCells.length === 1 && selectedCells[0] === targetCell) {
3922            // If already selected, just ensure focus
3923            targetCell.focus();
3924            placeCaretAtEnd(targetCell);
3925          }
3926        });
3927
3928        // Helper function to place caret at the end of content
3929        function placeCaretAtEnd(el) {
3930          if (document.createRange) {
3931            const range = document.createRange();
3932            range.selectNodeContents(el);
3933            range.collapse(false); // false means collapse to end
3934            const selection = window.getSelection();
3935            selection.removeAllRanges();
3936            selection.addRange(range);
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        );
3959      }
3960    }
3961  
```

```
3958         // Store starting cell and reset movement flag
3959         selectionStart = cell;
3960         hasMovedDuringSelection = false;
3961         isSelecting = true;
3962
3963         // Initially add this cell to selection
3964         if (!cell.classList.contains('selected-cell')) {
3965             toggleCellSelection(cell, true);
3966         }
3967
3968         // If it's a single click (not start of drag), focus the cell
3969         cell.focus();
3970
3971         // Don't prevent default here to allow normal focus behavior
3972     });
3973
3974         // Additional keyboard events for selected cells
3975         cell.addEventListener('keydown', function (e) {
3976             // If Delete or Backspace key and we have selected cells
3977             if ((e.key === 'Delete' || e.key === 'Backspace') &&
3978                 selectedCells.length > 1) {
3979                 e.preventDefault();
3980                 clearSelectedCellsContent();
3981             }
3982
3983             // Copy with Ctrl+C or Cmd+C
3984             if ((e.ctrlKey || e.metaKey) && e.key === 'c' &&
3985                 selectedCells.length > 0) {
3986                 e.preventDefault();
3987                 copySelectedCells();
3988             }
3989
3990             // Select all cells in row with Ctrl+A or Cmd+A
3991             if ((e.ctrlKey || e.metaKey) && e.key === 'a') {
3992                 e.preventDefault();
3993                 selectRowCells(cell.parentElement);
3994             }
3995         });
3996
3997         // Global mousemove handler for extending selection
3998         document.addEventListener('mousemove', function (e) {
3999             if (!isSelecting) return;
4000
4001             // Set the flag that we've moved during this selection
4002             hasMovedDuringSelection = true;
4003
4004             // Find the element under the cursor
4005             const elementUnderCursor = document.elementFromPoint(e.clientX,
e.clientY);
4006             if (!elementUnderCursor) return;
```

```
4006
4007         // Find the closest editable cell
4008         const targetCell =
4009         elementUnderCursor.closest('[contenteditable="true"]');
4010         if (targetCell) {
4011             // Add to selection
4012             toggleCellSelection(targetCell, true);
4013
4014             // Prevent text selection during drag
4015             e.preventDefault();
4016             if (window.getSelection) {
4017                 window.getSelection().removeAllRanges();
4018             }
4019         });
4020
4021         // Global mouseup to end selection - THIS IS THE FIXED VERSION
4022         document.addEventListener('mouseup', function (e) {
4023             // End selection mode but keep selected cells
4024             if (isSelecting) {
4025                 isSelecting = false;
4026
4027                 // FIX: Use setTimeout to delay resetting the hasMovedDuringSelection flag
4028                 // This gives the click handler a chance to see the flag first
4029                 if (hasMovedDuringSelection) {
4030                     setTimeout(function () {
4031                         hasMovedDuringSelection = false;
4032                     }, 10);
4033                 }
4034
4035                 // If this wasn't a drag and there's only one cell selected, focus
4036                 // it
4037                 if (!hasMovedDuringSelection && selectedCells.length === 1) {
4038                     const cell = selectedCells[0];
4039                     cell.focus();
4040                     placeCaretAtEnd(cell);
4041                 }
4042
4043                 selectionStart = null;
4044             });
4045
4046             // Listen for paste events
4047             document.addEventListener('paste', function (e) {
4048                 if (selectedCells.length > 0) {
4049                     handlePaste(e);
4050                 }
4051             });
4052         });
4053     }
```

```
4054 |     function toggleCellSelection(cell, addOnly = false) {
4055 |         const index = selectedCells.indexOf(cell);
4056 |
4057 |         if (index === -1) {
4058 |             // Add to selection
4059 |             selectedCells.push(cell);
4060 |             cell.classList.add('selected-cell');
4061 |         } else if (!addOnly) {
4062 |             // Remove from selection if not add-only mode
4063 |             selectedCells.splice(index, 1);
4064 |             cell.classList.remove('selected-cell');
4065 |         }
4066 |     }
4067 |
4068 |     function clearCellSelection() {
4069 |         selectedCells.forEach(cell => {
4070 |             cell.classList.remove('selected-cell');
4071 |         });
4072 |         selectedCells = [];
4073 |     }
4074 |
4075 |     function clearSelectedCellsContent() {
4076 |         selectedCells.forEach(cell => {
4077 |             cell.textContent = '';
4078 |             cell.classList.remove('modified');
4079 |         });
4080 |         modified = true;
4081 |         updateStatusMessage('Cleared selected cells');
4082 |     }
4083 |
4084 |     function copySelectedCells() {
4085 |         if (selectedCells.length === 0) return;
4086 |
4087 |         // Create a text representation of selected cells
4088 |         const cellData = selectedCells.map(cell => cell.textContent ||
4089 | '').join('\t');
4090 |
4091 |         // Copy to clipboard using Clipboard API
4092 |         navigator.clipboard.writeText(cellData)
4093 |             .then(() =>
4094 |                 updateStatusMessage('Copied selected cells to clipboard'));
4095 |             .catch(err => {
4096 |                 updateStatusMessage('Failed to copy: ' + err);
4097 |             });
4098 |
4099 |
4100 |     function handlePaste(e) {
4101 |         // Prevent default paste behavior
4102 |         e.preventDefault();
```

```
4103
4104    // Get clipboard data
4105    const clipboardData = e.clipboardData || window.clipboardData;
4106    const pastedText = clipboardData.getData('text');
4107
4108    // If we have a focused element within selection, paste there
4109    const activeElement = document.activeElement;
4110    if (activeElement && selectedCells.includes(activeElement)) {
4111        activeElement.textContent = pastedText;
4112        activeElement.classList.add('modified');
4113        modified = true;
4114    } else if (selectedCells.length > 0) {
4115        // Otherwise paste into first selected cell
4116        selectedCells[0].textContent = pastedText;
4117        selectedCells[0].classList.add('modified');
4118        modified = true;
4119    }
4120
4121    if (pastedText.trim() === '') {
4122        selectedCells.forEach(cell => {
4123            cell.classList.remove('modified');
4124        });
4125    }
4126
4127    updateStatusMessage('Pasted content into selected cell');
4128}
4129
4130 function selectRowCells(row) {
4131     // Clear previous selection
4132     clearCellSelection();
4133
4134     // Select all editable cells in the row
4135     const cells = Array.from(row.cells).filter(cell => cell.contentEditable ===
4136     'true');
4137     cells.forEach(cell => {
4138         toggleCellSelection(cell, true);
4139     });
4140
4141     // Functions
4142     function generateTable(frames) {
4143         tableBody.innerHTML = '';
4144         waveformCanvases = [];
4145
4146         // Create a column container for the waveform
4147         const waveformColContainer = document.createElement('div');
4148         waveformColContainer.className = 'waveform-col-container';
4149
4150         for (let i = 1; i <= frames; i++) {
4151             const row = document.createElement('tr');
```

```
4152     row.className = `frame-${i}`;
4153     row.setAttribute('data-frame', i);

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
example image)
4169             if (frameNum === 1) {
4170                 cell.style.backgroundColor = '#00cc00';
4171             } else {
4172                 cell.style.backgroundColor = '#cccccc';
4173             }
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

4187     // Add all cells to the row
4188     row.appendChild(createCell('action-col'));
4189     row.appendChild(createCell('frame-col', false, i));
4190     row.appendChild(createWaveformCell(i));
4191     row.appendChild(createCell('dialogue-col'));
4192     row.appendChild(createCell('sound-col'));
4193     row.appendChild(createCell('technical-col'));
4194     row.appendChild(createCell('extra1-col'));
4195     row.appendChild(createCell('extra2-col'));
4196     row.appendChild(createCell('frame-col', false, i));
4197     row.appendChild(createCell('camera-col'));

4198

4199     // Add special styling for 8-frame and 24-frame intervals
4200     if (i % 24 === 0) {
```

```
4201         row.style.borderBottom = '4px double #000';
4202         for (let cell of row.cells) {
4203             cell.style.borderBottom = '4px double #000';
4204             cell.style.fontWeight = 'bold';
4205         }
4206     } else if (i % 8 === 0) {
4207         row.style.borderBottom = '2px solid #000';
4208         for (let cell of row.cells) {
4209             cell.style.borderBottom = '2px solid #000';
4210             cell.style.fontWeight = 'bold';
4211         }
4212     }
4213
4214     tableBody.appendChild(row);
4215 }
4216
4217     setupCellNavigation();
4218     updateStatusMessage('Table generated with ' + frames + ' frames');
4219
4220     // Fire custom event for drawing system to update
4221     document.dispatchEvent(new Event('xsheet-updated'));
4222 }
4223
4224 function updateTemplate() {
4225     if (currentTemplate === 'large') {
4226         // 11"x17" template (96 frames)
4227         document.body.style.maxWidth = '11in';
4228         document.body.style.maxHeight = '17in';
4229         document.body.style.fontSize = '9pt';
4230         frameCountInput.value = 96;
4231         frameCount = 96;
4232     } else {
4233         // 8"x10" template (48 frames)
4234         document.body.style.maxWidth = '8in';
4235         document.body.style.maxHeight = '10in';
4236         document.body.style.fontSize = '8pt';
4237         frameCountInput.value = 48;
4238         frameCount = 48;
4239     }
4240
4241     generateTable(frameCount);
4242     updateStatusMessage('Template switched to ' + (currentTemplate === 'large' ?
4243         '11"x17"' : '8"x10"' ));
4244
4245         // Re-render waveform if audio is loaded
4246         if (audioBuffer) {
4247             renderWaveform();
4248         }
4249 }
```

```
4250 function setupCellNavigation() {
4251     const editableCells = document.querySelectorAll('[contenteditable="true"]');
4252
4253     editableCells.forEach(cell => {
4254         cell.addEventListener('keydown', function (e) {
4255             // Tab navigation
4256             if (e.key === 'Tab') {
4257                 e.preventDefault();
4258                 const currentRow = this.parentElement;
4259                 const currentIndex = Array.from(currentRow.cells).indexOf(this);
4260
4261                 if (e.shiftKey) {
4262                     // Shift+Tab moves backward
4263                     let prevCell = null;
4264                     if (currentIndex > 0) {
4265                         // Find previous editable cell in same row
4266                         for (let i = currentIndex - 1; i >= 0; i--) {
4267                             if (currentRow.cells[i].contentEditable === 'true')
4268                             {
4269                                 prevCell = currentRow.cells[i];
4270                                 break;
4271                             }
4272                         }
4273
4274                         if (!prevCell) {
4275                             // Move to previous row, last cell
4276                             const prevRow = currentRow.previousElementSibling;
4277                             if (prevRow) {
4278                                 const cells = Array.from(prevRow.cells).filter(c =>
4279 c.contentEditable === 'true');
4280                                 prevCell = cells[cells.length - 1];
4281                             }
4282
4283                             if (prevCell)
4284                                 prevCell.focus();
4285                             // Place cursor at end of content
4286                             const range = document.createRange();
4287                             const sel = window.getSelection();
4288                             range.selectNodeContents(prevCell);
4289                             range.collapse(false);
4290                             sel.removeAllRanges();
4291                             sel.addRange(range);
4292                         }
4293                     } else {
4294                         // Tab moves forward
4295                         let nextCell = null;
4296                         if (currentIndex < currentRow.cells.length - 1) {
4297                             // Find next editable cell in same row
4298                         }
4299                     }
4300                 }
4301             }
4302         }
4303     })
4304 }
```

```

4298     currentRow.cells.length; i++) {
4299         if (let i = currentIndex + 1; i <
4300             currentRow.cells[i].contentEditable === 'true')
4301         {
4302             nextCell = currentRow.cells[i];
4303             break;
4304         }
4305     }
4306     if (!nextCell) {
4307         // Move to next row, first cell
4308         const nextRow = currentRow.nextElementSibling;
4309         if (nextRow) {
4310             nextCell = Array.from(nextRow.cells).find(c =>
4311             c.contentEditable === 'true');
4312         }
4313     }
4314     if (nextCell) {
4315         nextCell.focus();
4316     }
4317 }
4318 }
4319
4320 // Enter key to move down
4321 if (e.key === 'Enter' && !e.shiftKey) {
4322     e.preventDefault();
4323     const currentRow = this.parentElement;
4324     const nextRow = currentRow.nextElementSibling;
4325     const currentIndex = Array.from(currentRow.cells).indexOf(this);
4326
4327     if (nextRow) {
4328         const cells = Array.from(nextRow.cells);
4329         const samePositionCell = cells[currentIndex];
4330         if (samePositionCell && samePositionCell.contentEditable ===
4331             'true') {
4332             samePositionCell.focus();
4333         }
4334     }
4335 });
4336
4337 // Add listener to remove the modified class when cell is emptied
4338 cell.addEventListener('keyup', function (e) {
4339     if ((e.key === 'Delete' || e.key === 'Backspace') &&
4340         this.textContent.trim() === '') {
4341         this.classList.remove('modified');
4342     }
4343 });

```

```
4344
4345          // Setup cell selection functionality
4346          setupCellSelection();
4347      }
4348
4349      // Audio functions
4350      function loadAudioFile(file) {
4351          if (!audioContext) {
4352              try {
4353                  window.AudioContext = window.AudioContext || window.webkitAudioContext;
4354                  audioContext = new AudioContext();
4355              } catch (e) {
4356                  updateStatusMessage('Web Audio API is not supported in this
4357 browser');
4358                  return;
4359              }
4360          }
4361
4362          const reader = new FileReader();
4363
4364          reader.onload = function (e) {
4365              const audioData = e.target.result;
4366
4367              // Update status
4368              updateStatusMessage('Decoding audio...');

4369              // Decode the audio data
4370              audioContext.decodeAudioData(audioData, function (buffer) {
4371                  audioBuffer = buffer;

4372
4373                  // Update audio info
4374                  const duration = buffer.duration;
4375                  const minutes = Math.floor(duration / 60);
4376                  const seconds = Math.floor(duration % 60);
4377                  const frameCount = Math.ceil(duration * 24); // Assuming 24fps

4378
4379                  audioInfo.textContent = `${audioFileName}
4380 (${minutes}:${seconds.toString().padStart(2, '0')}, ${frameCount} frames @ 24fps`;

4381
4382                  // Enable audio controls
4383                  playAudioButton.disabled = false;
4384                  stopAudioButton.disabled = false;
4385                  audioScrubber.disabled = false;

4386
4387                  // Generate waveform visualization
4388                  generateWaveformData(buffer);

4389
4390                  // Update status
4391                  updateStatusMessage('Audio loaded: ' + audioFileName);
4392          }
```

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

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

```
4490
4491     const rect = overlay.getBoundingClientRect();
4492     const y = e.clientY - rect.top;
4493     const percentage = y / rect.height;
4494
4495     // Calculate time point in the audio
4496     const timePoint = percentage * audioBuffer.duration;
4497
4498     // Update audio position
4499     audioScrubber.value = (timePoint / audioBuffer.duration) * 100;
4500     if (isPlaying) {
4501         stopAudio();
4502         startOffset = timePoint;
4503         playAudio();
4504     } else {
4505         startOffset = timePoint;
4506     }
4507
4508     // Update the UI to show frame position
4509     updateFrameMarker();
4510 });
4511
4512     // Variables for scrubbing
4513     let isScrubbing = false;
4514
4515     // Add scrubbing functionality (dragging while holding mouse button)
4516     overlay.addEventListener('mousedown', function (e) {
4517         if (!audioBuffer) return;
4518         if (e.button !== 0) return; // Only respond to left mouse button
4519
4520         isScrubbing = true;
4521
4522         // Pause any playing audio
4523         if (isPlaying) {
4524             pauseAudio();
4525         }
4526         // - NEW: allow pen (pointerType="pen") to scrub as well -
4527         overlay.addEventListener('pointerdown', function (e) {
4528             if (!audioBuffer || e.pointerType !== 'pen') return;
4529             isScrubbing = true;
4530             if (isPlaying) pauseAudio();           // pause continuous playback
4531             updateScrubPosition(e);            // scrub to this location
4532             e.preventDefault();              // prevent default pen
4533     behavior
4534     });
4535     overlay.addEventListener('pointermove', function (e) {
4536         // - FIX: only scrub while the pen is physically pressing
4537         (pressure>0)
4538             if (isScrubbing && e.pointerType === 'pen' && e.pressure > 0) {
4539                 updateScrubPosition(e);
4540             }
4541     });
4542 }
```

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

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

```
4637
4638     // Add the container to the table
4639     document.body.appendChild(waveformContainer);
4640
4641     // Position the container over the waveform column
4642     const tableRect = table.getBoundingClientRect();
4643     const cellRect = firstCell.getBoundingClientRect();
4644
4645     waveformContainer.style.left = (cellRect.left - 1) + 'px';
4646     waveformContainer.style.top = (cellRect.top) + 'px';
4647     waveformContainer.style.width = (cellRect.width) + 'px';
4648     waveformContainer.style.height = totalHeight + 'px';
4649
4650     // Draw the waveform on the canvas
4651     const ctx = canvas.getContext('2d');
4652     const width = canvas.width;
4653     const height = canvas.height;
4654
4655     // Clear and set background
4656     ctx.clearRect(0, 0, width, height);
4657     ctx.fillStyle = '#ffffff';
4658     ctx.fillRect(0, 0, width, height);
4659
4660     // Calculate scaling factors
4661     const totalDuration = audioBuffer.duration;
4662     const framesPerSecond = 24;
4663     const totalFrames = Math.ceil(totalDuration * framesPerSecond);
4664
4665     // Draw center line
4666     ctx.beginPath();
4667     ctx.strokeStyle = '#cccccc';
4668     ctx.moveTo(width / 2, 0);
4669     ctx.lineTo(width / 2, height);
4670     ctx.stroke();
4671
4672     // Draw the waveform
4673     ctx.beginPath();
4674     ctx.strokeStyle = '#000000';
4675     ctx.lineWidth = 1;
4676
4677     // Map waveform data to vertical height
4678     for (let i = 0; i < waveformData.length; i++) {
4679         const y = (i / waveformData.length) * height;
4680         const amplitude = waveformData[i] * (width * 0.4); // Scale amplitude to
4681         const x = (width / 2) + amplitude;
4682
4683         if (i === 0) {
4684             ctx.moveTo(x, y);
4685         } else {
```

40% of width

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

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

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

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

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

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

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

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

```

5077     rows[index].cells[3].innerText = rowData.dialogue || '';
5078     rows[index].cells[4].innerText = rowData.sound || '';
5079     rows[index].cells[5].innerText = rowData.technical || '';
5080     rows[index].cells[6].innerText = rowData.extra1 || '';
5081     rows[index].cells[7].innerText = rowData.extra2 || '';
5082     // Don't restore the second frame number cell
5083     rows[index].cells[9].innerText = rowData.camera || '';
5084   }
5085 });
5086 }
5087
5088 modified = false;
5089 updateStatusMessage('Project loaded successfully');
5090 }
5091
5092 function saveProject() {
5093   const data = collectData();
5094
5095   // Save to localStorage
5096   try {
5097     localStorage.setItem('animationXSheet', JSON.stringify(data));
5098     modified = false;
5099     updateStatusMessage('Project saved successfully');
5100
5101     // Also download as JSON file for backup
5102     const filename = projectName + '.json';
5103     const dataStr = "data:text/json;charset=utf-8," +
5104 encodeURIComponent(JSON.stringify(data));
5105     const downloadAnchorNode = document.createElement('a');
5106     downloadAnchorNode.setAttribute("href", dataStr);
5107     downloadAnchorNode.setAttribute("download", filename);
5108     document.body.appendChild(downloadAnchorNode);
5109     downloadAnchorNode.click();
5110     downloadAnchorNode.remove();
5111   } catch (e) {
5112     updateStatusMessage('Error saving project: ' + e.message);
5113   }
5114 }
5115
5116 function loadProject() {
5117   // First try to load from localStorage
5118   try {
5119     const savedData = localStorage.getItem('animationXSheet');
5120     if (savedData) {
5121       restoreData(JSON.parse(savedData));
5122       return;
5123     }
5124   } catch (e) {
5125     updateStatusMessage('Error loading saved project: ' + e.message);
5126   }

```

```
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
5134 fileInput.addEventListener('change', function (e) {
5135     const file = e.target.files[0];
5136     if (!file) return;
5137
5138     const reader = new FileReader();
5139     reader.onload = function (e) {
5140         try {
5141             const data = JSON.parse(e.target.result);
5142             restoreData(data);
5143         } catch (error) {
5144             updateStatusMessage('Error parsing file: ' + error.message);
5145         }
5146     };
5147     reader.readAsText(file);
5148 });
5149
5150     fileInput.click();
5151     fileInput.remove();
5152 }
5153
5154 function exportToPDF() {
5155     // Save the current state before PDF export
5156     const savedData = collectData();
5157
5158     updateStatusMessage('Preparing PDF. Please wait...');
5159
5160     // Remove controls temporarily for PDF generation
5161     const controls = document.querySelector('.controls');
5162     const audioControls = document.querySelector('#audio-controls');
5163     const statusMsg = document.querySelector('.status');
5164     const phoneticInputEl = document.querySelector('#phonetic-input');
5165
5166     controls.style.display = 'none';
5167     audioControls.style.display = 'none';
5168     statusMsg.style.display = 'none';
5169     phoneticInputEl.style.display = 'none';
5170
5171     // Clean up any previous waveform elements
5172     const previousContainers = document.querySelectorAll('.cell-waveform-
5173 window');
5174     previousContainers.forEach(container => container.remove());
```

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

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

```
5272 const canvas = document.createElement('canvas');
5273 canvas.width = cell.offsetWidth;
5274 canvas.height = cell.offsetHeight;
5275 canvas.style.display = 'block';
5276 canvas.style.width = '100%';
5277 canvas.style.height = '100%';
5278
5279 const ctx = canvas.getContext('2d');
5280
5281 // Fill with white background
5282 ctx.fillStyle = 'white';
5283 ctx.fillRect(0, 0, canvas.width, canvas.height);
5284
5285 // Draw center line
5286 ctx.beginPath();
5287 ctx.strokeStyle = '#cccccc';
5288 ctx.moveTo(canvas.width / 2, 0);
5289 ctx.lineTo(canvas.width / 2, canvas.height);
5290 ctx.stroke();
5291
5292 // Calculate which section of the waveform to draw
5293 const startPoint = Math.floor((frameNum - 1) * pointsPerFrame);
5294 const endPoint = Math.floor(frameNum * pointsPerFrame);
5295
5296 if (startPoint < waveformData.length) {
5297     // Draw this portion of the waveform
5298     ctx.beginPath();
5299     ctx.strokeStyle = '#000000';
5300
5301     // Check if we have valid data to draw
5302     if (endPoint > startPoint) {
5303         // Map the waveform section to this cell
5304         for (let i = startPoint; i <= endPoint && i <
waveformData.length; i++) {
5305             // Calculate position within this cell
5306             const relativePos = (i - startPoint) / (endPoint -
startPoint);
5307             const y = relativePos * canvas.height;
5308
5309             // Draw waveform
5310             const amplitude = waveformData[i] * (canvas.width *
0.4);
5311             const x = (canvas.width / 2) + amplitude;
5312
5313             if (i === startPoint) {
5314                 ctx.moveTo(x, y);
5315             } else {
5316                 ctx.lineTo(x, y);
5317             }
5318         }
5319     }
5320 }
```

```
5320 // Draw left side (mirror)
5321 for (let i = endPoint; i >= startPoint && i <
5322 waveformData.length; i--) {
5323     const relativePos = (i - startPoint) / (endPoint -
5324     startPoint);
5325     const y = relativePos * canvas.height;
5326     const amplitude = waveformData[i] * (canvas.width *
5327     0.4);
5328     const x = (canvas.width / 2) - amplitude;
5329     ctx.lineTo(x, y);
5330 }
5331 ctx.stroke();
5332 }
5333
5334 // Look for phonetic markers that might be in this frame
5335 if (phonetics && phonetics.length > 0) {
5336     phonetics.forEach(phonic => {
5337         // Calculate which frame this phonetic marker belongs to
5338         const frameOfMarker = Math.floor(phonic.time /
5339         frameDuration) + 1;
5340
5341         // If it's in this frame, draw it
5342         if (frameOfMarker === frameNum) {
5343             // Calculate position within cell
5344             const posInFrameRatio = (phonic.time - ((frameNum
5345 - 1) * frameDuration)) / frameDuration;
5346             const yPos = posInFrameRatio * canvas.height;
5347
5348             // Draw marker line
5349             ctx.beginPath();
5350             ctx.strokeStyle = '#ff0000';
5351             ctx.lineWidth = 1;
5352             ctx.moveTo(0, yPos);
5353             ctx.lineTo(canvas.width, yPos);
5354             ctx.stroke();
5355
5356             // Add label if it fits
5357             if (canvas.width > 30) {
5358                 ctx.fillStyle = 'rgba(255, 255, 255, 0.7)';
5359                 const textWidth =
5360                 ctx.measureText(phonic.text).width;
5361
5362                 if (textWidth < canvas.width - 4) {
5363                     ctx.fillRect(2, yPos - 8, textWidth + 4,
5364                     14);
5365                     ctx.fillStyle = '#ff0000';
5366                     ctx.font = '8px Arial';
5367                     ctx.fillText(phonic.text, 4, yPos + 4);
5368                 }
5369             }
5370         }
5371     })
5372 }
```

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

```
5414     waveformCells.forEach(cell => {
5415         const originalContent = cell.getAttribute('data-original-
5416             content');
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
5436                     if (!cell.classList.contains('waveform-col')) {
5437                         // Only restore if cell is editable
5438                         if (cell.contentEditable === 'true') {
5439                             cell.innerHTML = cellData.content;
5440
5441                             // Restore modified status
5442                             if (cellData.isModified) {
5443                                 cell.classList.add('modified');
5444                             } else {
5445                                 cell.classList.remove('modified');
5446                             }
5447                         }
5448                     }
5449
5450                     // Restore metadata fields
5451                     if (tableData && tableData.metadata) {
5452                         document.getElementById('project-number').value =
5453                             tableData.metadata.projectNumber || '';
5454                         document.getElementById('project-date').value =
5455                             tableData.metadata.date || '';
5456                         document.getElementById('page-number').value =
5457                             tableData.metadata.pageNumber || '';
5458                         document.getElementById('animator-name').value =
5459                             tableData.metadata.animatorName || '';
5460                         document.getElementById('version-number').value =
5461                             tableData.metadata.versionNumber || '';
5462
5463
5464
5465
5466
5467
5468
5469
5470
5471
5472
5473
5474
5475
5476
5477
5478
5479
5480
5481
5482
5483
5484
5485
5486
5487
5488
5489
5490
5491
5492
5493
5494
5495
5496
5497
5498
5499
5500
5501
5502
5503
5504
5505
5506
5507
5508
5509
5510
5511
5512
5513
5514
5515
5516
5517
5518
5519
5520
5521
5522
5523
5524
5525
5526
5527
5528
5529
5530
5531
5532
5533
5534
5535
5536
5537
5538
5539
5540
5541
5542
5543
5544
5545
5546
5547
5548
5549
5550
5551
5552
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
5598
5599
5599
5600
5601
5602
5603
5604
5605
5606
5607
5608
5609
5610
5611
5612
5613
5614
5615
5616
5617
5618
5619
5620
5621
5622
5623
5624
5625
5626
5627
5628
5629
5629
5630
5631
5632
5633
5634
5635
5636
5637
5638
5639
5639
5640
5641
5642
5643
5644
5645
5646
5647
5648
5649
5650
5651
5652
5653
5654
5655
5656
5657
5658
5659
5659
5660
5661
5662
5663
5664
5665
5666
5667
5668
5669
5669
5670
5671
5672
5673
5674
5675
5676
5677
5678
5679
5679
5680
5681
5682
5683
5684
5685
5686
5687
5688
5689
5689
5690
5691
5692
5693
5694
5695
5696
5697
5698
5699
5699
5700
5701
5702
5703
5704
5705
5706
5707
5708
5709
5709
5710
5711
5712
5713
5714
5715
5716
5717
5718
5719
5719
5720
5721
5722
5723
5724
5725
5726
5727
5728
5729
5729
5730
5731
5732
5733
5734
5735
5736
5737
5738
5739
5739
5740
5741
5742
5743
5744
5745
5746
5747
5748
5749
5749
5750
5751
5752
5753
5754
5755
5756
5757
5758
5759
5759
5760
5761
5762
5763
5764
5765
5766
5767
5768
5769
5769
5770
5771
5772
5773
5774
5775
5776
5777
5778
5778
5779
5780
5781
5782
5783
5784
5785
5786
5787
5788
5789
5789
5790
5791
5792
5793
5794
5795
5796
5797
5798
5799
5799
5800
5801
5802
5803
5804
5805
5806
5807
5808
5809
5809
5810
5811
5812
5813
5814
5815
5816
5817
5818
5819
5819
5820
5821
5822
5823
5824
5825
5826
5827
5828
5829
5829
5830
5831
5832
5833
5834
5835
5836
5837
5838
5839
5839
5840
5841
5842
5843
5844
5845
5846
5847
5848
5849
5849
5850
5851
5852
5853
5854
5855
5856
5857
5858
5859
5859
5860
5861
5862
5863
5864
5865
5866
5867
5868
5869
5869
5870
5871
5872
5873
5874
5875
5876
5877
5878
5879
5879
5880
5881
5882
5883
5884
5885
5886
5887
5888
5889
5889
5890
5891
5892
5893
5894
5895
5896
5897
5898
5899
5899
5900
5901
5902
5903
5904
5905
5906
5907
5908
5909
5909
5910
5911
5912
5913
5914
5915
5916
5917
5918
5919
5919
5920
5921
5922
5923
5924
5925
5926
5927
5928
5929
5929
5930
5931
5932
5933
5934
5935
5936
5937
5938
5939
5939
5940
5941
5942
5943
5944
5945
5946
5947
5948
5949
5949
5950
5951
5952
5953
5954
5955
5956
5957
5958
5959
5959
5960
5961
5962
5963
5964
5965
5966
5967
5968
5969
5969
5970
5971
5972
5973
5974
5975
5976
5977
5978
5979
5979
5980
5981
5982
5983
5984
5985
5986
5987
5988
5989
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
7
```

```
5456     document.getElementById('shot-number').value =
5457     tableData.metadata.shotNumber || '';
5458
5459         }
5460
5461         // Finally restore cell selection state
5462         restoreSelectionState(tableData);
5463
5464         updateStatusMessage('Print complete');
5465     }, 500);
5466 }, 100);
5467
5468 function drawWaveformInCells() {
5469     const waveformCells = document.querySelectorAll('.waveform-col');
5470     if (waveformCells.length === 0) return;
5471
5472     // Calculate the number of data points per frame
5473     const totalDuration = audioBuffer.duration;
5474     const pointsPerFrame = waveformData.length / (totalDuration * 24);
5475
5476     // For each cell, draw its portion of the waveform
5477     waveformCells.forEach((cell, index) => {
5478         // Get the frame number (1-based)
5479         const frameNum = index + 1;
5480
5481         // Create a canvas for this cell
5482         const canvas = document.createElement('canvas');
5483         canvas.width = cell.offsetWidth;
5484         canvas.height = cell.offsetHeight;
5485         canvas.style.display = 'block';
5486         canvas.style.width = '100%';
5487         canvas.style.height = '100%';
5488
5489         const ctx = canvas.getContext('2d');
5490
5491         // Fill with white background
5492         ctx.fillStyle = 'white';
5493         ctx.fillRect(0, 0, canvas.width, canvas.height);
5494
5495         // Draw center line
5496         ctx.beginPath();
5497         ctx.strokeStyle = '#cccccc';
5498         ctx.moveTo(canvas.width / 2, 0);
5499         ctx.lineTo(canvas.width / 2, canvas.height);
5500         ctx.stroke();
5501
5502         // Calculate which section of the waveform to draw
5503         const startPoint = Math.floor((frameNum - 1) * pointsPerFrame);
5504         const endPoint = Math.floor(frameNum * pointsPerFrame);
5505
5506         if (startPoint < waveformData.length) {
```

```
5505 // Draw this portion of the waveform
5506 ctx.beginPath();
5507 ctx.strokeStyle = '#000000';
5508
5509 // Check if we have valid data to draw
5510 if (endPoint > startPoint) {
5511     // Map the waveform section to this cell
5512     for (let i = startPoint; i <= endPoint && i <
5513         waveformData.length; i++) {
5514         // Calculate position within this cell
5515         const relativePos = (i - startPoint) / (endPoint -
5516         startPoint);
5517         const y = relativePos * canvas.height;
5518
5519         // Draw waveform
5520         const amplitude = waveformData[i] * (canvas.width *
5521             0.4);
5522         const x = (canvas.width / 2) + amplitude;
5523
5524         if (i === startPoint) {
5525             ctx.moveTo(x, y);
5526         } else {
5527             ctx.lineTo(x, y);
5528         }
5529
5530         // Draw left side (mirror)
5531         for (let i = endPoint; i >= startPoint && i <
5532             waveformData.length; i--) {
5533             const relativePos = (i - startPoint) / (endPoint -
5534             startPoint);
5535             const y = relativePos * canvas.height;
5536
5537             const amplitude = waveformData[i] * (canvas.width *
5538                 0.4);
5539             const x = (canvas.width / 2) - amplitude;
5540
5541             ctx.lineTo(x, y);
5542
5543             ctx.stroke();
5544         }
5545
5546         // Look for phonetic markers that might be in this frame
5547         if (phonetics && phonetics.length > 0) {
5548             phonetics.forEach(phnetic => {
5549                 // Calculate which frame this phonetic marker belongs to
5550                 const frameOfMarker = Math.floor(phnetic.time /
5551                     frameDuration) + 1;
5552
5553                 // If it's in this frame, draw it
5554             });
5555         }
5556     }
5557 }
```

```

5549         if (frameOfMarker === frameNum) {
5550             // Calculate position within cell
5551             const posInFrameRatio = (phonetic.time - ((frameNum
5552 - 1) * frameDuration)) / frameDuration;
5553             const yPos = posInFrameRatio * canvas.height;
5554
5555             // Draw marker line
5556             ctx.beginPath();
5557             ctx.strokeStyle = '#ff0000';
5558             ctx.lineWidth = 1;
5559             ctx.moveTo(0, yPos);
5560             ctx.lineTo(canvas.width, yPos);
5561             ctx.stroke();
5562
5563             // Add label if it fits
5564             if (canvas.width > 30) {
5565                 ctx.fillStyle = 'rgba(255, 255, 255, 0.7)';
5566                 const textWidth =
5567                     ctx.measureText(phnetic.text).width;
5568
5569                 if (textWidth < canvas.width - 4) {
5570                     ctx.fillRect(2, yPos - 8, textWidth + 4,
5571
5572                         14);
5573
5574                     ctx.fillStyle = '#ff0000';
5575                     ctx.font = '8px Arial';
5576                     ctx.fillText(phnetic.text, 4, yPos + 4);
5577
5578                 }
5579             }
5580
5581             // Add frame number indicator
5582             ctx.fillStyle = 'rgba(255, 255, 255, 0.7)';
5583             ctx.fillRect(0, 0, 18, 12);
5584             ctx.fillStyle = '#000000';
5585             ctx.font = '8px Arial';
5586             ctx.fillText(frameNum, 2, 8);
5587
5588             // Store original content and add the canvas
5589             cell.setAttribute('data-original-content', cell.innerHTML);
5590             cell.innerHTML = '';
5591             cell.appendChild(canvas);
5592
5593         function saveSelectionState() {
5594             // Save current selection and cell contents
5595             const tableData = {
5596                 // Save which cells are selected
5597                 selectedIndices: selectedCells.map(cell => {

```

```
5597     const row = cell.closest('tr');
5598     const rowIndex = Array.from(tableBody.children).indexOf(row);
5599     const colIndex = Array.from(row.children).indexOf(cell);
5600     return { rowIndex, colIndex };
5601   },
5602   // Save metadata fields
5603   metadata: {
5604     projectNumber: document.getElementById('project-number').value,
5605     date: document.getElementById('project-date').value,
5606     pageNumber: document.getElementById('page-number').value,
5607     animatorName: document.getElementById('animator-name').value,
5608     versionNumber: document.getElementById('version-number').value,
5609     shotNumber: document.getElementById('shot-number').value
5610   },
5611   // Store editable cells' content
5612   cellContents: []
5613 };
5614
5615   // Store all editable cells' content
5616   const allEditableCells =
5617     document.querySelectorAll('[contenteditable="true"]');
5618     allEditableCells.forEach(cell => {
5619       const row = cell.closest('tr');
5620       if (row) {
5621         const rowIndex = Array.from(tableBody.children).indexOf(row);
5622         const colIndex = Array.from(row.children).indexOf(cell);
5623         tableData.cellContents.push({
5624           rowIndex,
5625           colIndex,
5626           content: cell.innerHTML,
5627           isModified: cell.classList.contains('modified')
5628         });
5629       }
5630     });
5631
5632   return tableData;
5633 }
5634
5635   function restoreSelectionState(tableData) {
5636     if (!tableData || !tableData.selectedIndices) return;
5637
5638     // Clear current selection
5639     clearCellSelection();
5640
5641     // Restore selection
5642     tableData.selectedIndices.forEach(index => {
5643       if (index.rowIndex >= 0 && index.rowIndex <
5644         tableBody.children.length) {
5645         const row = tableBody.children[index.rowIndex];
5646         if (index.colIndex >= 0 && index.colIndex < row.children.length)
5647       {

```

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

```
5693
5694         // Stop any playing audio
5695         stopAudio();
5696
5697         // Clear waveform visualization
5698         waveformCanvases.forEach(canvas => {
5699             const ctx = canvas.getContext('2d');
5700             ctx.clearRect(0, 0, canvas.width, canvas.height);
5701         });
5702
5703         // Clear phonetic markers
5704         const labels = document.querySelectorAll('.phonetic-label');
5705         labels.forEach(label => label.remove());
5706
5707         // Clear drawings
5708         if (window.xsheetDrawing && window.xsheetDrawing.layerSystem) {
5709             window.xsheetDrawing.layerSystem.clearAllLayers();
5710         }
5711
5712         modified = false;
5713         updateStatusMessage('Sheet cleared');
5714     }
5715 }
5716
5717 function updateStatusMessage(message) {
5718     statusMessage.textContent = message;
5719     console.log(message);
5720
5721     // Clear status message after 3 seconds
5722     setTimeout(() => {
5723         if (statusMessage.textContent === message) {
5724             if (modified) {
5725                 statusMessage.textContent = 'Unsaved changes';
5726             } else {
5727                 statusMessage.textContent = '';
5728             }
5729         }
5730     }, 3000);
5731 }
5732
5733         // Auto-save timer every 2 minutes
5734         setInterval(() => {
5735             if (modified) {
5736                 try {
5737                     const data = collectData();
5738                     localStorage.setItem('animationXSheet_autosave',
5739 JSON.stringify(data));
5740                     updateStatusMessage('Auto-saved');
5741                 } catch (e) {
5742                     console.error('Auto-save failed:', e);
5743                 }
5744             }
5745         }, 120000);
5746 }
```

```
5742         }
5743     }
5744 }, 120000);
5745
5746 // Check for auto-saved data on load
5747 try {
5748     const autoSavedData = localStorage.getItem('animationXSheet_autosave');
5749     if (autoSavedData && !localStorage.getItem('animationXSheet')) {
5750         if (confirm('Found auto-saved data. Would you like to restore it?')) {
5751             restoreData(JSON.parse(autoSavedData));
5752         }
5753     }
5754 } catch (e) {
5755     console.error('Error checking for auto-saved data:', e);
5756 }
5757
5758 // Initial status
5759 updateStatusMessage('X-Sheet ready');
5760 });
5761 </script>
5762 </body>
5763
5764 </html>
```