Canvas For Rendering UX

By Gökay Şatır

Software Engineer Collabora Online

gokaysatir@collabora.com

Canvas For Rendering UX

Aim

- Collabora Online uses images (tiles) sent from the core side for rendering the document. Users need to be able to view documents (consisting of tiles) comfortably. But the technology is rapidly improving and it created high resolution devices with tiny pixels. As we aimed to have consistent look across as many device as we can, we needed to have a new solution. For having crisp images in all targeted devices, we choose to use a canvas object.

Content

- Rendering the tiles.
- Rendering the UI.
- Implementation.
- Result.

Canvas For Rendering UX

Benefits In Detail

- Collabora Online is now using an HTML canvas element for rendering the tiles. Benefits are:
  - Easier to solve CSS related problems across different browsers: Like gaps between tiles & blurry images.
  - Same look in different devices.
  - Easier to maintain and improve.
  - Implementation is very easy.
  - It doesn't use hammer.js or other 3rd part libraries for event handling.
  - It removed the need for mCustomScrollbar and other libraries for scroll bar management & drawing.
  - Sometimes events are fired more than once because of 3rd party libraries. This implementation fires events selectively, so we can re-draw only when we need to.
  - It is touch ready, tested.
  - Using canvas doesn’t disable other solutions. For example we use an image for cursor.
  - We can draw markers, selections and objects easily. We can create different sections for them or we can draw them inside document section.
  - It is also synchronised with our testing framework.

Canvas For Rendering UX

Rendering The Tiles

- Tiles are now rendered on the canvas. Devices with higher resolution render more tiles and users benefit from the features of their devices.

Rendering The UI

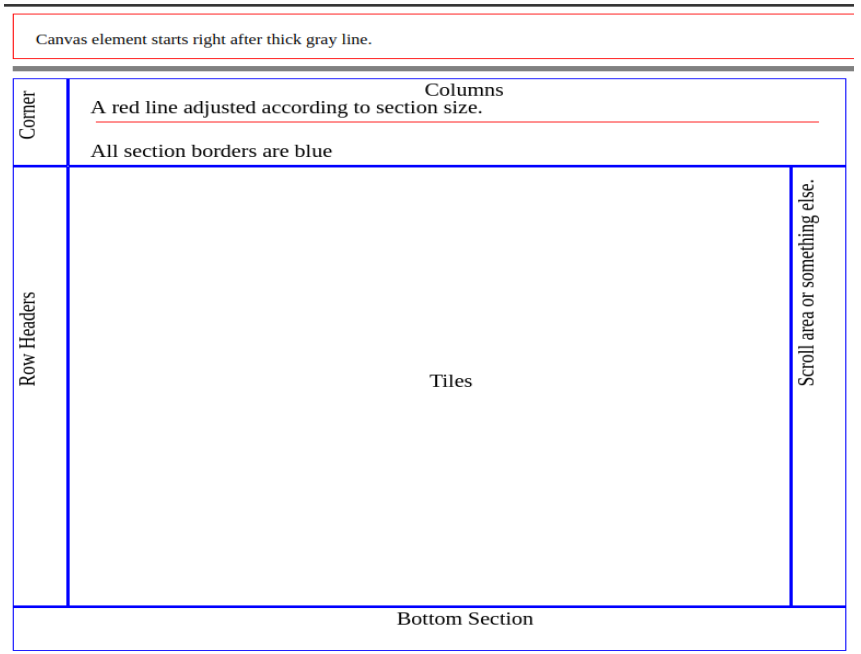
- For some parts of the UI, we are now using the same canvas element we use for rendering the tiles. Benefits:
  - UI may change too fast (like in Calc headers), HTML elements are not a good fit for fast redraws.

Canvas For Rendering UX

Implementation

- After deciding to use a canvas element for rendering some parts of the UI and the document itself, next step was implementing the idea.
- Tiles mean the document itself, user can click on it, write on it or scroll it. UI elements like row headers in Calc also can be scrolled, clicked etc. Handling these events are essential. UI elements and the document need their event handlers.
- At the end, we decided to centralize the event handling and drawings within a new class, which we call CanvasSectionContainer.
- This class would position the UI elements and the document inside the view (screen) automatically.

First Look Of The New Class



Layout

- Automatically managed by the section container.
- Every UI part and document has its own section.

Event Handling

- Events are caught by the section container.
- Events are propagated to the target sections.
- Sections implement the handlers according to their needs.

Canvas For Rendering UX

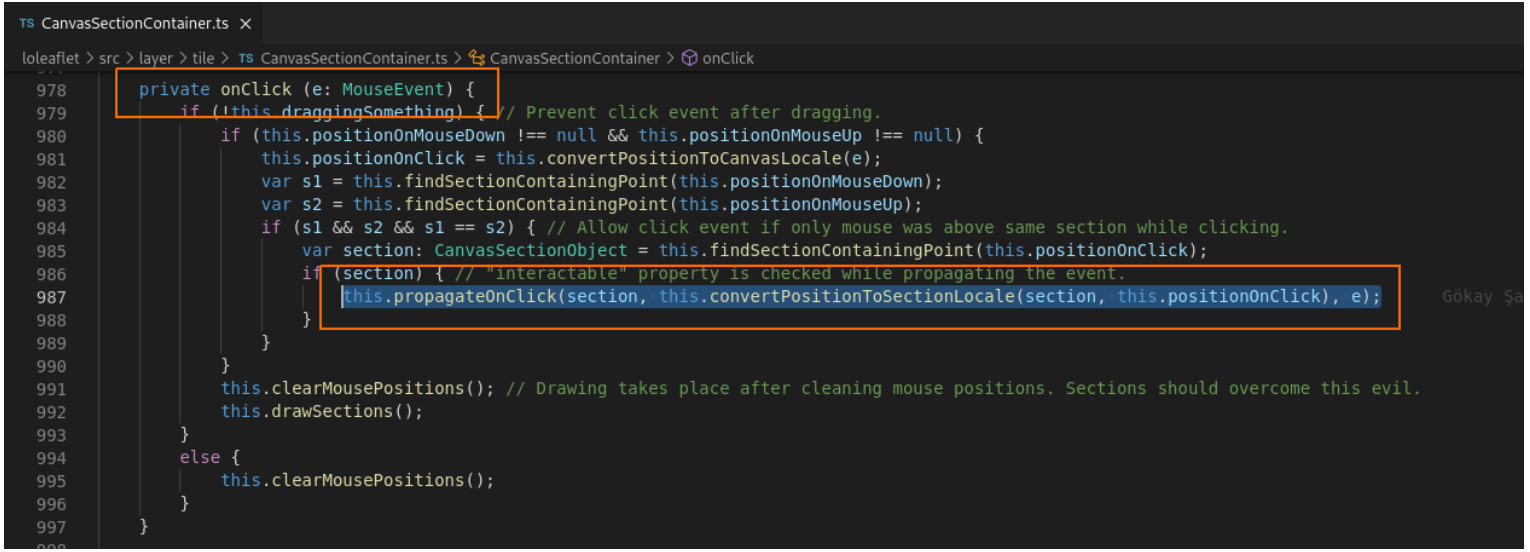
Features and Challenges

- Event Propagation:
  - It is easy to handle events of the canvas element. We can always use something like:
    - canvas.onclick = somefunction.
    - canvas.addEventListener('click', somefunction).
  - But we needed layer-like structures that overlap entirely. For example:
    - In Calc, tiles are drawn on a layer (in our terms, by a section).
    - Cell cursor also needs to be drawn on tiles (the document) by another section. Cell cursor may be anywhere on the document, so It needs to cover entire document.

Canvas For Rendering UX

Features & Challenges

- Event Propagation:
  - In the end, we needed to propagate events not only to sections but also between the sections.
  -



Canvas For Rendering UX

Features and Challenges

- Devices With Mouse & Touch Screen:
  - Collabora Online supports multiple device types, including mobile phones and tablets. Tablet devices can be used with mouse and keyboard support as well as via their touch screen. Before using canvas element and implementing CanvasSectionContainer class, we were using Hammer.js and other 3rd party libraries, in order to support touch & mouse usage. These libraries always come with their advantages and disadvantages. For example, it was a bit difficult to support touch screen & mouse at the same time.
  - While we change our direction towards canvas drawings and event handling with sections, we also solved these event handling problems.

Canvas For Rendering UX

Features & Challenges

- Devices With Mouse & Touch Screen:
  - We now support tablet devices with keyboard and touch screen, without a third party library.
  -

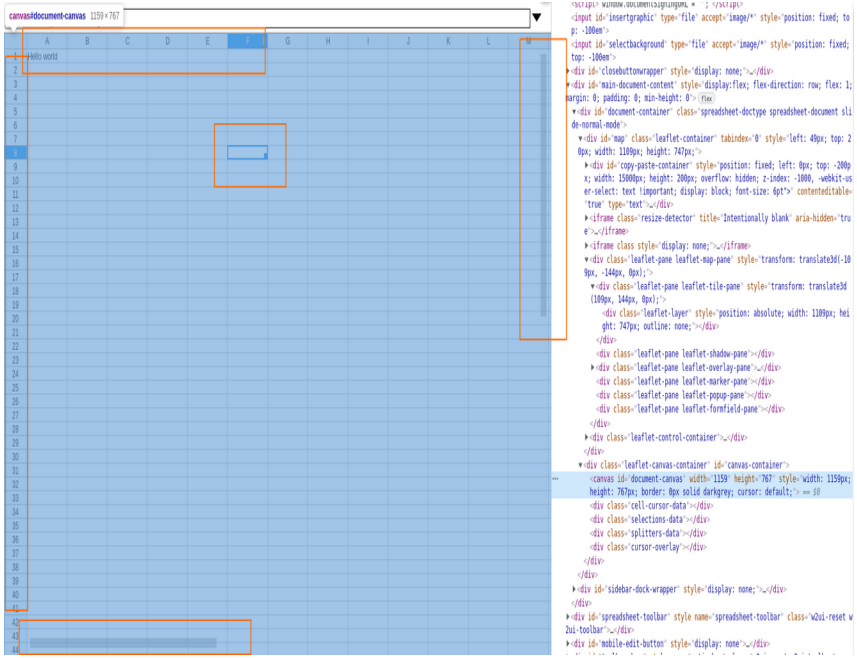
```
if (!this.potentialLongPress) {
    if (!this.touchEventInProgress) {
        this.mousePosition = this.convertPositionToCanvasLocale(e);
        if (this.positionOnMouseDown !== null && !this.draggingSomething) {
        }

        var section: CanvasSectionObject;

        if (this.draggingSomething) {
            this.dragDistance = [this.mousePosition[0] - this.positionOnMouseDown[0], this.mousePosition[1] - this.positionOnMouseDown[1]];
            section = this.getSectionWithName(this.sectionOnMouseDown);
        }
        else {
            section = this.findSectionContainingPoint(this.mousePosition);
        }

        if (section) {
            if (section.name !== this.sectionUnderMouse) {
                if (this.sectionUnderMouse !== null) {
                    var previousSection: CanvasSectionObject = this.getSectionWithName(this.sectionUnderMouse);
                    if (previousSection)
                        this.propagateOnMouseLeave(previousSection, this.convertPositionToSectionLocale(previousSection, this.mousePosition), e);
                }
                this.sectionUnderMouse = section.name;
                this.propagateOnMouseEnter(section, this.convertPositionToSectionLocale(section, this.mousePosition), e);
            }
            this.propagateOnMouseMove(section, this.convertPositionToSectionLocale(section, this.mousePosition), this.dragDistance, e);
        }
        else if (this.sectionUnderMouse !== null) {
            var previousSection: CanvasSectionObject = this.getSectionWithName(this.sectionUnderMouse);
            if (previousSection)
                this.propagateOnMouseLeave(previousSection, this.convertPositionToSectionLocale(previousSection, this.mousePosition), e);
            this.sectionUnderMouse = null;
        }
    }
}
else {
    this.mousePosition = this.convertPositionToCanvasLocale(e);
    var section: CanvasSectionObject = this.findSectionContainingPoint(this.mousePosition);
    if (section) {
        this.propagateOnLongPress(section, this.convertPositionToSectionLocale(section, this.mousePosition), e);
    }
}
}
```

Canvas For Rendering UX

Features & Challenges

- Crisp Images:
  - This was the most challenging part. I spent quite good time while implementing this. Thanks to Michael and Jan (Kendy) for their mentoring.
    - Making the images (tiles) look crisp on every device that Collabora Online works, was the main target.
    - Because of "***window.devicePixelRatio***" property of the browser (indeed, related to device), crisp images were difficult to render. This value is sometimes equal to something like 0.9~
    - We have tiles with dimensions 256X256 pixels.
    - When we draw these tiles onto a device (let's say) with device pixel ratio "2", they are drawn half-sized.
  - We kept the 256X256 px dimensions and we increased the number of tiles when the devicePixelRatio is above 1. We also solved problems with floating numbers. In the end, the resolution of the document/tiles/images became good once again.

First Look Of The New Class



Layout

- Automatically managed by the section container.
- Every UI part and document has its own section.

Event Handling

- Events are caught by the section container.
- Events are propagated to the target sections.
- Sections implement the handlers according to their needs.

Thanks !

Collabora Online

Gökay Şatır

@CollaboraOffice
hello@collaboraoffice.com
Collaboraoffice.com