Preventing elements to be behind fixed elements

Overview

This code snippet demonstrates how to solve the problem with text or other elements left behind fixed or static elements by using "phantom" (aka "shadow" or "ghost") elements, which are invisible elements occupying space.

Problem

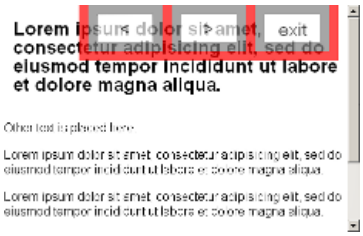
Sometimes in widget or in mobile web page, you want to place buttons or some other content on the top right corner of the widget, but at the same time, set also the title or other content on the top right of the page. For pages having many views (or sub-pages), which are handled with the JavaScript(tm), you might prefer to not use navigation buttons in each subpage separately, but instead use global navigation buttons.

One solution of using global navigation buttons is to position the global buttons as `position:fixed` or `position:absolute`. However, then it can happen that the content in the sub-pages are shown under the buttons. The question is, how to make sure that the elements are not placed under these global navigation buttons.

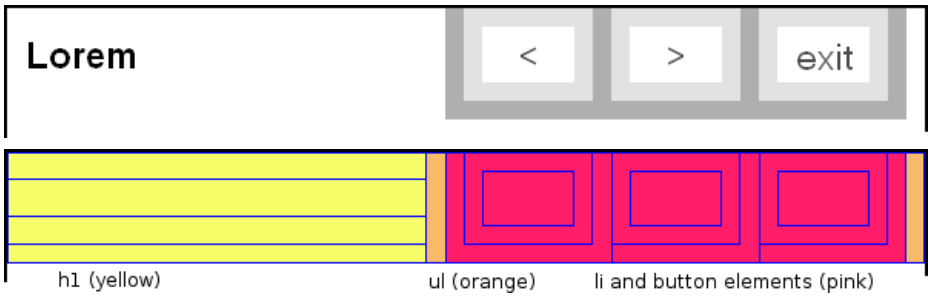
This code snippet demonstrates how to solve this problem by using "phantom" (aka "shadow" or "ghost") elements, which are invisible elements occupying space.

These phantom elements are placed into each subpage, are set to have a specific widths and heights, and are applied the same (or similar) styles as the real elements. They will then occupy the same space as the real objects, and will be positioned under the global navigation buttons, but will belong into the same "scope" as the other elements in the page. Thus, the result is that there is a space reserved for the global buttons on behalf of the phantom elements, and thus no other elements can go behind the global buttons.

The starting point of this code snippet can be obtained from here, which is a page as follows:



The global navigation buttons are formed by an ul element which contains a list of buttons. The header is a h1 element, as the following figures describe:



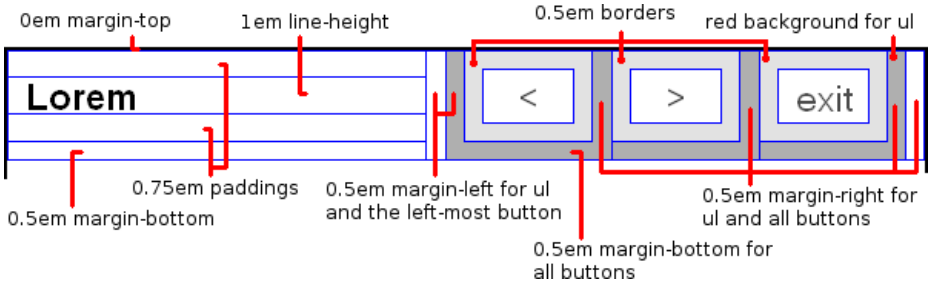
The HTML code for the header and the navigation buttons is shown below.

```
<body>
  <div id="content">
    <ul id="navButtons" class="navButtons">
      <li><input type="button" id="exitButton" class="navButton first" value="exit"
/></li>
      <li><input type="button" id="next" class="navButton" value="&gt;" /></li>
      <li><input type="button" id="previous" class="navButton last" value="&lt;" /></li>
    </ul>
    <div id="view1" class="view">
      <h1>
        Lorem ipsum dolor sit amet, consectetur adipisicing elit,
```


(C) Copyright Nokia 2014. All rights reserved.

```
        sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
      </h1>
      ...
    </div>
    ... <!-- other #viewX elements -->
  </div>
</body>
```

The following figure shows how the margins, borders and paddings are set:



Solution

To start, we add an ul list of .ghostNavButton divs as follows. Every #viewX elements must have the same list added.

```
<body>
  <div id="content">
    <ul id="navButtons" class="navButtons">
      <li><input type="button" id="exitButton" class="navButton first" value="exit"
/></li>
      <li><input type="button" id="next" class="navButton" value="&gt;" /></li>
      <li><input type="button" id="previous" class="navButton last" value="&lt;" /></li>
    </ul>
    <div id="view1" class="view">
      <!-- THE LIST ADDED HERE v -->
      <ul class="ghostNavButtons" >
        <li><div class="ghostNavButton first"></div></li>
        <li><div class="ghostNavButton"></div></li>
        <li><div class="ghostNavButton last"></div></li>
      </ul>
      <!-- LIST ADDITION ENDS -->
      <h1>
        Lorem ipsum dolor sit amet, consectetur adipisicing elit,
        sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
      </h1>
      ...
    </div>
    ... <!-- other #viewX elements -->
  </div>
</body>
```

Each of the view1, view2, etc., elements contain the contents for one subpage (view).

The style of h1 is defined as follows, and we don't need to change it:

```
h1 {
  padding-top: 0.75em;
  padding-bottom: 0.75em;
```


(C) Copyright Nokia 2014. All rights reserved.

```
  margin: 0em 0.5em 0.5em;
  font-size: 1.5em;
  line-height: 1em;
 font-weight: bolder;
}
```

The #navButtons list's position is set as fixed, and its background and color are defined as follows. No changes to them are needed:

```
#navButtons{
  position: fixed;
  top: 0;
  right: 0;
  z-index: 2;
}

.navButton {
  background: white;
  color: black;
}
```

Now we start to look what changes must be done. The .navButton has borders, width and height declared as follows:

```
.navButton {
  border: 0.5em solid gray;
  height: 2.5em;
  width: 3.5em;
}
```

Here we must be careful. For button elements, the paddings and borders are included into the height and width. For div elements, they are NOT by default, but since we set the padding and borders to none in our ghost div elements, we set the width and height of the .ghostNavButton elements to be the same in both.

```
.ghostNavButton {
  height: 2.5em;
  width: 3.5em;
}
```

Now both the .navButton and .ghostNavButton elements have the same size.

If the h1 title element would have any margin-top, it would have the same effect as if the whole viewX div element (of the title) would have that same margin-top. If buttons can be in two rows, the h1 margin-top must be zero for this layout style to work without modifications.

Next we go through each remaining declarations of .navButtons, .navButtons li and .navButton elements, and add a corresponding .ghostNavButtons, .ghostNavButtons li and .ghostNavButton element declarations, respectively. For example, if there would be a CSS declaration `.navButtons { ... }` , we modify it to be `.navButtons, .ghostNavButtons { ... }`

After all these changes, the result for the remaining declarations is as follows:

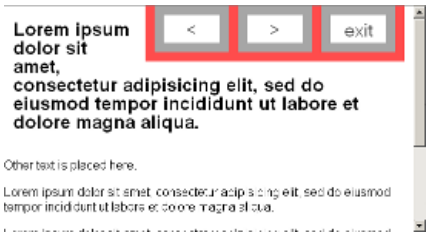
```
.navButtons, .ghostNavButtons {
  list-style-type: none;
  margin: 0 0.75em; /* corresponds to 0.5em of h1, since 0.75em = 1.5 x 0.5em */
  padding: 0;
```


(C) Copyright Nokia 2014. All rights reserved.

```
}

.navButtons li, .ghostNavButtons li {
  float: right;
}

.navButton, .ghostNavButton {
  font-size: 1.5em;
}

.navButton, .ghostNavButton {
  padding: 0em;
  margin-right: 0.5em;
  margin-bottom: 0.5em;
}

.navButton.last, .ghostNavButton.last {
  margin-left: 0.5em;
}

.navButton.first, .ghostNavButton.first {
}
```

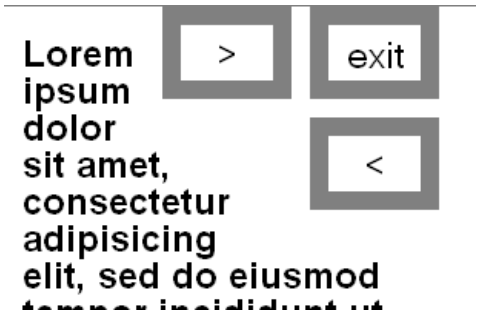
Postconditions

Now the .ghostNavButtons lists and their elements are layed out in a same way as the .navButtons, and we can see that the header text is not layed out behind the global navigation buttons anymore.



Supplementary material

In the attached solution zip file you can find the zipped solution web page, and in problem zip file you can find the starting point, before adding the "ghost" elements for the global navigation buttons. There is also a specific.css file in the exampleappnamelowercase folder, where you can find other variations for the solution provided, which you can uncomment and comment out to explore them more. One example for a variation is described by the following figure, which shows buttons in two rows and a text which is layed out so that it does not fall behind the buttons.



Discussion

The advantages of this approach are:

- There will be only the scrollbars of the whole web page, unlike in the case where the navigation buttons are placed on a div on the top of a page and the subpage content is placed in a div which is positioned so that it occupies the rest of the screen

(C) Copyright Nokia 2014. All rights reserved.

area. Furthermore, the scrollable elements are known to have some problems.

- You can animate the buttons, unlike in the case where each page have their own copies of the global navigation buttons.
- Furthermore, this page works as an plain HTML page even without the JavaScript disabled, if you first set .navButtons and .ghostNavButtons as "display: none" and then use the JavaScript to revert the effect of the previous definitions, which would have effect only when the JavaScript is enabled. The result would be a web page having the sub pages placed one after another.

The disadvantages of this approach are:

- it is more complex
- if the buttons are placed as position: fixed, the buttons will be over other elements when the page is scrolled. However, this should not be a problem in most cases.

(C) Copyright Nokia 2014. All rights reserved.