Creating Widgets - a Pie Menu (ii)

If you completed the last exercise successfully, you should have a web-page that includes a pop-up pie menu: clicking anywhere on the page causes the menu to appear at the cursor position.

According to Fitts' Law, pie menus are very efficient because all options are equidistant from the starting point. Thus any option can be reached as quickly as any other option.

Another implication of Fitts' Law is that selection times can be reduced by making menus infinitely large in the direction of travel.

In the Macintosh OS, menus are placed at the edge of the screen so that it is impossible to move the cursor beyond the menu.

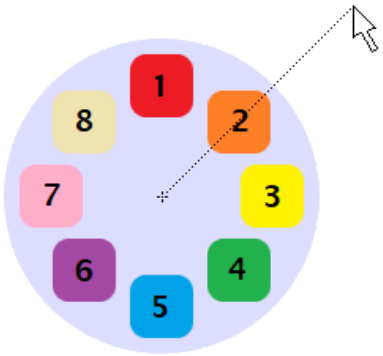
Thus the user does not have to juggle the speed so as to land on the menu item - s/he can simply swipe in the required direction and then click. This makes selection much faster.

It's possible to make a pie menu even more efficient by making the icons behave as if they are infinitely large.

Instead of having to click on an icon to select it, the user merely has to swipe over it and then click anywhere else on the page.

This allows a form of gestural interaction.

For example, to select the icon labelled '2' in the diagram, the user only has to click anywhere on the document to bring up the menu, then swipe diagonally over the icon and click again.



Similarly, any of the other icons can be selected by swiping in the appropriate direction and clicking.

This is much faster and easier than aiming to land on an icon without overshooting it.

You can modify the menu created in the last exercise to support similar functionality. This will involve:

- Modifying the event-listener attached to each icon so that it responds to `mouseover` events rather than `click` events.
- Adding an event-listener to the `document` which responds to `click` events.
    - The user will no longer be clicking on an icon, but on the `document`, so the `click` event must now be captured by the `document`.
- Adding a function which is called when the user clicks on the `document`.
    - This function should determine which icon was most recently the target of a `mouseover` event.
    - This will be the selected icon.

1. Add a new global variable at the start of the script called (e.g.) `selectedIcon`.
   - This will hold the `id` of the last icon over which the cursor was swiped.

     For example:

```
window.addEventListener('DOMContentLoaded', setup, false);
var popupMenu = null;
var selectedIcon = null;

function setup() {

  document.addEventListener('click', createMenu, false);
}
```

2. The `addIcon()` function currently attaches an event listener to each new icon which calls a `closeMenu()` function when the icon is clicked, e.g.:

```
newIcon.addEventListener('click', closeMenu, false);
```

   Modify this line of code so that it responds to `mouseover` events and calls a function named `selectIcon()`:

```
newIcon.addEventListener('mouseover', selectIcon, false);
```

   For example:

```
function addIcon(xCoord, yCoord, iconNumber) {

  var iconDiv = document.createElement('div');
  iconDiv.id = 'icon' + iconNumber;
  iconDiv.style.position = 'absolute';
  iconDiv.style.left = xCoord + 'px';
  iconDiv.style.top = yCoord + 'px';
  iconDiv.style.width = '40px';
  iconDiv.style.height = '40px';
  iconDiv.style.backgroundColor = '#ff0000';
  iconDiv.style.borderRadius = '10px';
  newIcon = popupMenu.appendChild(iconDiv);
  newIcon.addEventListener('mouseover', selectIcon, false);
}
```

3. Create the function `selectIcon()` that is called by the event listener you have just modified.

   This function should obtain the `id` of the icon which called it (`evt.target.id`) and store it in `selectedIcon`.

   For example:

```
function selectIcon(evt) {

  selectedIcon = evt.target.id
}
```

4. If the menu is to be used efficiently, it is essential that, when it appears, it is centred around the cursor position.

Therefore, if you've not already done so, add code to the `createMenu()` function to offset the menu by 100 pixels (half its diameter) vertically and horizontally from the cursor position.

For example:

```
function createMenu(evt) {

 if(!popupMenu) {

    var cursorX = parseInt(evt.clientX);
    var cursorY = parseInt(evt.clientY);
    var leftPosn = topPosn = 100;
    if(cursorX > 100) leftPosn = cursorX - 100;
    if(cursorY > 100) topPosn = cursorY - 100;

    ...

    menuDiv.style.left = leftPosn + 'px';
    menuDiv.style.top = topPosn + 'px';

    ...
 }
}
```

5. As described in the previous exercise, the `createMenu()` function should include a line of code that attaches an event-listener to the popup menu, e.g.;

```
popupMenu.addEventListener('click', closeMenu, false);
```

Add another line to this function which attaches a similar event listener to the document, e.g.:

```
document.addEventListener('click', closeMenu, false);
```

This allows the menu to be closed by clicking over the document at the end of a swipe.

For example:

```
function createMenu(evt) {

  if(!popupMenu) {

     ...

     popupMenu = document.documentElement.appendChild(menuDiv);
     popupMenu.addEventListener('click', closeMenu, false);
     document.addEventListener('click', closeMenu, false);
  }
}
```

6. Finally, modify the `closeMenu()` function so that it:
    a. uses the value stored in `selectedIcon` to determine which icon was selected, instead of using `evt.target.id`
    b. removes the event-listener attached to the document by the code described in step 4.
    c. sets `selectedIcon` to `null`.

For example:

```
function closeMenu(evt) {

   evt.preventDefault();
   evt.stopPropagation();
   document.documentElement.removeChild(popupMenu);
   if (selectedIcon) alert('Selected icon = ' + selectedIcon);
   document.removeEventListener('click', closeMenu, false);
   popupMenu = null;
   selectedIcon = null;
}
```

Test your code in a browser.

You should find that everything works as before, except that now you can swipe over an icon and then click anywhere to select it.

Compare this version with the previous version.

You should find that selecting icons is easier and faster in the new version because you don't have to navigate onto an icon before clicking but can simply swipe over it.

This illustrates the advantages of using so-called *infinite* menus or icons.

As described, the menu is launched by clicking the left mouse button.

Such menus are normally launched by clicking the *right* mouse button.

The `click` event is only triggered by the left mouse button, but most browsers also support a `contextmenu` event which fires when the *right* mouse button is clicked.

In the `createMenu()` function, modify the `addEventListener()` method so that it responds to a `contextmenu` event rather than a `click` event.

Test your code again. It should now be possible to use a right-click to open the menu and a left-click to select an icon and/or close the menu.

There are many varieties of pie menu, some incorporating a form of infinite menu and some not.

New types of interaction widget appear quite frequently as usability engineers seek to improve the ease and speed of selection/navigation.

This exercise and the previous one illustrate just a few of the possibilities for developing your own interaction widgets.