2811 / User Interfaces
Tom Kelly

- v2 QtScrollArea → QScrollArea

Coursework 2: Responsive Layouts
Date set: 1.11.20
Date due: 15.11.20, 12 noon via Minerva
Weighting: 35%

Goals for this CW:

- Demonstrate a working understanding of writing custom layouts in C++ and Qt
- Create a prototype layout for an app
- Design a beautiful responsive layout

Getting started:

- Watch the video
- Download and extract this zip file
- Open the *.pro* file with Qt Creator
- Set the first command line argument to point to the folder which contains your unzipped source code. Use double quotes around the folder.
- Run the project. It creates a main window with several coloured labels which can be manually resized. Try changing the size of the window; observe the responsive design - the number, location and size of the widgets change - but the design is terrible.
- Run the project in automatic test mode (**warning your screen will flash rapidly**):
    1. Add a second command line argument: *test* (without quotes).
    2. Run the program again.
    3. Observe that a number of different window sizes are displayed, screenshots are taken, and written to the *report* directory in the root of the project. The program then exits. View the *report/index.html* in a browser and observe the terrible responsive design.
    4. Set the project back to manual mode for your development by removing the program argument *test*.
- There is a new system for generating a file to submit. Try this now.
    1. Add a second command line argument *pack* (without quotes)
    2. Run the program again. Check the "Application Output" tab in Qt Creator for any error messages.
    3. Your files and report are packed into a file *submit.patch* in the directory above your source output.
    4. Check that your *submit.patch* contains both your automatically generated report and your source code by replacing the command line argument *pack* with *unpack*. This will extract the submit.patch to the directory *tmp_xxxx*, located next to the patch file. (you should check this before submission!)
    5. Your project will be graded using code similar (but not identical) to this script.

6. You are encouraged to submit early and often so please submit this default *submit.patch* to Minerva now.

- The project contains three important C++ classes:
    1. `ResponsiveWindow`: This subclasses `QWidget` and creates the widgets to be displayed in the `ResponsiveWindow::createWidgets` function. It sets a `ResponsiveLayout` and adds various `ResponsiveLabel`s.
    2. `ResponsiveLayout`: This subclasses `QLayout` and arranges the `ResponsiveLabels` in the window using the `ResponsiveLayout::setGeometry` function (Lecture 6 discusses custom layouts in Qt).
    3. `ResponsiveLabel`: This subclasses `QLabel` and creates a label widget with a name and a coloured background.
- Read the classes to understand how they work together to create the responsive layout you have observed.
- The following widgets and associated colours have been predefined in the *responsive_label.h* file. (Lecture 5 introduces some of this terminology):
    1. home link (`kHomeLink`)
    2. shopping basket button (`kShoppingBasket`)
    3. sign-in button - assume user is logged out (`kSignIn`)
    4. menu button - (`kMenu`)
    5. navigation tabs - main site areas (`kNavTabs`)
    6. advertisements for related products (`kAdvert`)
    7. search box - a text field (`kSQuery`)
    8. search button - performs a new search (`kSButton`)
    9. search result, consisting of:
        - search result image (`kSResultImage`)
        - search result description (`kSResultText`)
    10. next page of search results button (`kSForward`)
    11. previous page of search results button (`kSBackward`)
    12. detailed search options - price, product rating, category etc.. (`kSOptions`)

Your task:

- You will create a *prototype layout* for a page of a mobile app. This prototype will demonstrate which widgets are shown on what sized devices and how their layout adapts in a responsive manner.
- In this coursework you will build a single responsive page layout for a *shopping app*. The page will show *search results*.

Marks will be awarded for:

- layouts which show appropriate placed widgets in response to varying window sizes and orientations.
    - show all provided labels at the largest size; fewer at smaller sizes.
    - marks are available for up to four *discrete* layouts (e.g., two horizontal and two vertical layouts).
- widgets which do not overlap and do not extend beyond the edge of the window.
- layouts which have beautiful, non-zero, and appropriate spacing between the widgets (Lecture 7) without any large gaps.
- adding up to two additional label types of your own.
- code which compiles, runs, and adheres to the below formatting rules.
- more challenging extension tasks:
    - research and use `QScrollArea` to allow the page to scroll and show more search results (a search result is a `kSResultImage` and a `kSResultText`).
    - create a layout for your scroll area which positions 17 search results in a responsively sized grid layout. Each whole search result (a `kSResultImage` and a `kSResultText`) should take up a square of space. The horizontal and vertical spacing between the results, and around the grid, should be consistent.

To submit:

- Generate your report with the *test* command line argument as described above.
- Compress your project and automatically generated report with the *pack* command line argument as described above.
    - Validate the contents of the *submit.patch* file by using the *unpack* command line argument. Ensure it includes the latest automatically generated report and your code.
    - Do not otherwise change the *.patch* file.
- Use Minerva to submit *submit.patch* by the deadline at the top of this page.
- You may submit multiple times. Only the most recent will be graded. Please submit early and often.

Notes:

- You should carefully plan which labels are shown at different screen sizes and orientations by researching which elements are prioritised at which sizes on real-world shopping apps and websites.
- Split your responsive design into several *discrete* layouts for different sizes. A discrete layout is a unique arrangement of labels optimised for a particular screen size.
- You may wish to sketch the widgets on paper before you write code.
- Only use `ResponsiveLabel` widgets with a simple text description (and a single `QScrollArea` if you attempt the extension task).

- The widgets don't need to do anything (i.e., signals and slots are not required; nothing is expected to happen when you click on a label).
- It is not necessary for all the label text to always be visible.
- None of Qt's built-in layouts may be used (`QVBoxLayout…`).
- Do not edit the files with the warning "`DO NOT EDIT THIS FILE`" at the top. They will be used during the marking process.
- The window should show a *usable, beautiful,* and *useful* layout whether sized at its minimum (320x320 pixels) or its maximum (1280 x 720 pixels), as well as all sizes in between. Landscape and portrait orientations should be considered.
- The prototype is for a full-screen app with a touch-based interface (for a variety of different sized mobile and tablets devices). The widgets should be positioned and sized accordingly.
- We will take the physical dimension of the minimum size window (320x320 pixels) to be 4.5x4.5 cm (about the size of BlackBerry Q5 mobile phone screen). Larger physical dimensions are taken to be scaled proportionally.
- Avoid large gaps in any layouts. Space after a final search result, e.g., to finish the row, is acceptable.
- It is suggested to add the maximum number of ResponsiveLabels required for all possible layouts in `ResponsiveWindow::createWidgets` and hide them as required in `ResponsiveLayout::setGeometry.` Widgets can be hidden by setting their size to zero.
- Follow the following code formatting rules:
    - comment your code with single-line comments (using `//`) such that someone familiar with C++ and this project description (i.e., the person marking) is able to follow it.
    - braces:
        - opening braces on the same line as the function definition or block (e.g., if/for/while statements) they open.
        - use *cuddled else*…
        - …empty braces {} on the same line…
        - …otherwise, closing braces on their own line.
        - no braces for single-line blocks.
    - file names are all lower-case.
    - class names begin with capital letters.
    - variable and function names begin with lower-case letters.
    - constant globals to begin with a lower-case 'k'.
    - function length should be limited to 50 lines (excluding empty lines and comments).
    - line length should be limited to 100 characters.
    - there should be no unused (commented or inaccessible) code.