Developing Native Apps for BlackBerry 10 with Cascades

Lab # 1: *Getting Started with Cascades, Qt, and the Native SDK*

The objective of this lab is to review some of the concepts of Qt (pronounced "cute") and Cascades for creating Native applications for BlackBerry 10 devices. In this lab, we'll be dealing with the interface of the application. The instructions are based on a Windows environment.

Note: In Windows, be sure to run each of the following installers "as administrator". Before attempting this lab, please be sure to get a set of BB10 Device Code signing keys (NFC key optional) from https://www.blackberry.com/SignedKeys/nfc-form.html (be sure to select the box for BB10, BB7 and below optional), and download the BlackBerry 10 Native SDK with Cascades and the BlackBerry 10 Dev Alpha Simulator, both of which are available at https://developer.blackberry.com/cascades/download. To set up the environment, follow the steps from https://developer.blackberry.com/cascades/documentation/getting_started/setting_up.html. Note that you should have 2GB of free space on your computer, and that your computer meets the minimum system requirements listed (click "Requirements" under the download button on the Downloads page to view them). Next, download and install VMware Player (from http://www.vmware.com/products/player/ Note that you'll need to create an account with VMware if you do not already have one). To start the simulator, simply open VMware Player, click on "Open a Virtual Machine" and open the .vmx file from wherever you installed the BB10 simulator.

This lab assumes that you have some knowledge of Java (only for the use of comparing it to the Qt framework, thus this is not required), a basic knowledge of C++, an understanding of Object Oriented programming, and assumes you have little to no knowledge of Cascades, the BlackBerry 10 Native SDK, or Qt.

Exercise 1: Examine the Project, Deal with clicks

- Extract the contents of Lab1-Startup.rar to a folder of your choice.
- Run the QNX Momentics IDE (it will likely be named "BlackBerry Native SDK" in the Start menu, or on the desktop, if you chose to create a shortcut during installation).
- In the IDE, click "File" > "Open File…" and then open the .pro file.
- In the Project Explorer pane on the left, expand the project folder, and then expand the "src" folder.
- Open up the "app.hpp" C++ header file, and take a quick glance at it. You will notice that it has a lot of things declared already. This is so you won't have to go back and forth between the "app.hpp" and "app.cpp" files as you progress through the lab.
    - You'll notice a lot of #include statements. Those starting with Q (ex. QList, QDebug, etc.) are part of the Qt framework
        - <QtCore/QObject> - Like Java's Object class, QObject is the base class of all Qt objects.
        - <QtCore/QMetaType> - This is a special Qt object that holds meta-data for custom objects, this isn't too important to know right now, but it allows Qt to understand your custom objects in order to pass them around properly.
        - <QList> - Qt's equivalent of Java's ArrayList. A dynamically-sized list of objects or even primitive types like int or char.

        - <QDebug> - This is Qt's debug output object. This lets you print debug messages to the console.
        - <QVariant> - This is an abstract class of Qt's types. Qt uses QVariant when a function can return a variety of objects. It is up to the user to know what type of object the functions really return and need to call QVariant's conversion functions (ex. toInt, toString) to convert the QVariant to the expected type.
    - Below the Qt include statements you'll notice a whole bunch of #include statements starting with "bb/cascades". These are, of course, for the Cascades User Interface components. Most of them are straightforward and don't need explanations so I won't explain each of them here, but you can see the comments in the header file to get a better understanding of them.
    - You'll notice that below the include statements we state that we will be "**using namespace** bb::cascades;". This package contains all of the Cascades classes, and we state this so we don't have to type out "bb::cascades" in front of every component name. "Button" looks much friendlier than "bb::cascades::Button" after all.
    - If you are not familiar with Qt, there are a few important things to note in the class declaration.
        - "**class** App : **public** QObject" – This is how we start declaring a class. Its name is App, and it extends the base QObject class.
        - "Q_OBJECT" – This is a very important line. It tells Qt that your class is a subclass of QObject, which causes Qt to add meta-data to the class that will allow it to properly implement signals, slots, dynamic properties, and run-time type information.
        - "**App**()" and **"~App**()" – "App()" is the class constructor, while "~App()" is a destructor. A destructor is called at the end of an object's lifespan and should be used to release any resources the object is holding.
        - "**public:**" – This declares the access for the functions and variables that follow it, like in C++. There is also protected and private.
        - "**public** slots:" – This declares that the functions that follow are to be treated by Qt as SLOTs. SLOTs are functions that can be bound to SIGNALs, so that when a SIGNAL is emitted it will call all bound SLOTs. This allows you to attach "callback" functions to events like button clicks. SLOTs can also be called as regular functions. You can also declare "public signals:" but we won't have any in this lab, as all of our events come from Cascades components.
- Now that you've got a general understanding of the header format, open the "app.cpp" file
    - You'll notice that in the constructor it just contains a bunch of Containers, a single Label for the title, 9 Buttons, and a single Page to contain it all.
    - There is an *initialize()* function, were we load the three images we're going to use later on.
    - There are also function stubs for each of the SLOTs.
- Let's try running the App to see what it looks like.
    - Start up VMware Player and open the BB10 Dev Alpha Simulator and wait for it to finish loading.
    - Click the arrow beside the Build button (hammer image) in the IDE, and make sure it is set to "Simulator-Debug"
    - Click the Build button and wait until it finishes building.

- Click on the Run button. Then click on “BlackBerry C/C++ Application”.
- You should see something like **Figure 1**.

Figure 1

- You’ll notice that the background is a light black, which is what we set the background color to for the **appContainer**.
- That “grid” of buttons below the title is actually 3 horizontal StackLayouts inside of a vertical StackLayout. This is because Cascades does not currently have a Grid layout of any kind. We’ll refer this “grid” as the “game board” from now on.
- If you try to click on a Button, you’ll see that it reacts to your click, but nothing happens. In reality our SLOTs, that were bound to the Buttons with *connect()*, are being called, but since they are empty they don’t do anything. Try changing “void App::*clickedX0Y0()*{}” to “void App::*clickedX0Y0()*{ *qDebug()* << “First button clicked!”; }”. Try running it again. Now when you click the first button, the Console in the IDE should print out “First button clicked!”
- Note that each button and their respective click function represents the x and y coordinates for the corresponding spot on the game board, where X represents horizontal position, and Y represents vertical position (X0Y0 is top left, and X2Y2 is bottom right).
- Now we can start adding some functionality to the App. First let’s do some initialization. In the constructor for App you’ll see that we make a call to *initialize()*. We’re going to add some variable initialization in this function.
- Note that we are going to be using two class variables for storing some game data: the ***spots*** QList and the ***remainingSpots*** QList. These already exist in the header file.
  - ***spots*** is a list of 9 characters, one for each spot on the 3x3 game board. It is used to store who is occupying each spot: an ‘**X**’ means the player is, an ‘**O**’ means the computer is, and a ‘**-**’ means the spot is currently not taken.
  - ***remainingSpots*** stores the indexes of the remaining spots available on the game board. It starts off with 9 elements, and shrinks as spots on the game board are taken. This will be used later on for quickly picking a spot for the computer’s turn.

- Find the *initialize()* function. This function will be called once when the application starts. In this function you'll just need to add a call to *newGame()* after loading of the images. We will add more to this function later on.
- Find the *newGame()* function stub. It takes no parameters, and returns nothing.
    - Inside of it, empty the ***spots*** QList (i.e. call its *clear()* function). We're going to append the values into the list, so we want it to be empty.
    - Also empty the ***remainingSpots*** QList (i.e. call its *clear()* function). This way, if there were still indexes left in the list from the last game we can remove them.
    - After clearing the lists, create a for-loop to iterate through each of the 9 spots in the QLists (from 0 to 8).
        - Use the loop to append 9 dash ('-') characters to the ***spots*** QList to signify that each spot of the game board is available.
        - Also use the loop to append the numbers 0 to 8 into the ***remainingSpots*** QList.
    - After the loop we need to clear the game board. This involves setting the images of the buttons back to the **blankImg** image. Do this in a new function, call it *clearBoard()*. It takes no parameters and returns nothing.
        - Call *setImage(blankImg)* on each of the buttons from ***x0y0*** to ***x2y2***.
    - Your *newGame()* function should now look something like the following:

```
void App::newGame(){
	spots.clear();
	remainingSpots.clear();
	for(int i=0; i<=8; i=i+1){
		spots.append('-');
		remainingSpots.append(i);
	}
	clearBoard();
}
```

    - Your *clearBoard()* function should now look something like the following:

```
void App::clearBoard(){
	x0y0->setImage(blankImg);
	x1y0->setImage(blankImg);
	x2y0->setImage(blankImg);
	x0y1->setImage(blankImg);
	x1y1->setImage(blankImg);
	x2y1->setImage(blankImg);
	x0y2->setImage(blankImg);
	x1y2->setImage(blankImg);
	x2y2->setImage(blankImg);
}
```

- Next we'll handle the button clicks. When one of the Buttons in the game board is clicked, we want to display the '**X**' image in that Button, set the corresponding element in ***spots*** to an '**X**' and remove the index from the ***remainingSpots*** list. Since every Button will do the same thing except that it'll do it for a different spot, we will make a single function to call from each of the click SLOTs.
    - Create a new function, call it *spotSelect()*. It takes in one parameter, an integer **index**, and does not return anything.
    - First check that the element in ***spots*** at **index** is not already taken
        - If the element at **index** in the ***spots*** QList is not a '-', then the spot is already taken.

        - If the spot is already taken, simply return from the function. You can add a *qDebug()* statement before the return statement so you can see when this case occurs.
    - Next, we need to set the element at **index** in the ***spots*** QList. Since the user initiated the click on the spot, we want to set the spot to an '**X**'.
    - Now, use a switch statement to set the image of the Button corresponding to the **index** parameter to the **xImg** image. Note that the numbering system for spots on the game board works as follows:

	X0	**X1**	**X2**
Y0	X0Y0 = index 0	X1Y0 = index 1	X2Y0 = index 2
Y1	X0Y1 = index 3	X1Y1 = index 4	X2Y1 = index 5
Y2	X0Y2 = index 6	X1Y2 = index 7	X2Y2 = index 8

    - Finally, we need to remove the **index** from the ***remainingSpots*** list.
        - Make a new integer variable named **toRemove** and store the result of ***remainingSpots***.*indexOf(index)* in it.
        - Call ***remainingSpots***.*removeAt(**toRemove**)*.
    - Your *spotSelect()* function should look something like the following:

```
void App::spotSelect(int index){
	if(spots[index] != '-'){
		qDebug() << "Spot already taken.";
		return;
	}
	spots[index] = 'X';
	switch(index){
		case(0): x0y0->setImage(xImg); break;
		case(1): x1y0->setImage(xImg); break;
		case(2): x2y0->setImage(xImg); break;
		case(3): x0y1->setImage(xImg); break;
		case(4): x1y1->setImage(xImg); break;
		case(5): x2y1->setImage(xImg); break;
		case(6): x0y2->setImage(xImg); break;
		case(7): x1y2->setImage(xImg); break;
		case(8): x2y2->setImage(xImg); break;
		default: return;
	}
	int toRemove = remainingSpots.indexOf(index);
	remainingSpots.removeAt(toRemove);
}
```

- Now we can set up the click SLOTs to call *spotSelect()*
    - In each of the click SLOT stubs (i.e. *clickedX0Y0*, and so forth) call *spotSelected()* passing it the index corresponding to each respective button (refer to the above table if you need help remembering which button corresponds to which index).
- If you've tried to run your app so far, you may notice some errors appearing with regards to images not being found. This is because we haven't added the images to our project yet. Copy the images folder included with the startup files into the assets folder of your project.

- You can either find the project folder using the filesystem on your computer, or you can right-click on the assets folder of your project in the Momentics IDE and click Paste.

- Run your app again. The image on the buttons should change to 'X's when you click on them.
- You should see something like **Figure 2**.

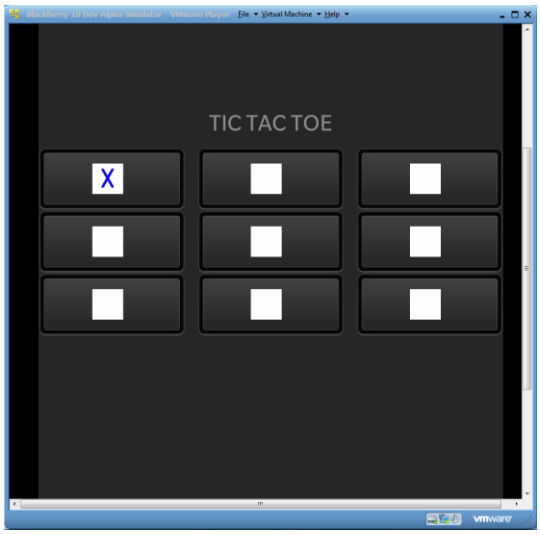

Figure 2

- As you can see in **Figure 3**, you can currently click on more squares than you should be able to in a regular game of TicTacToe. We're going to fix this in the next step by letting the App take spots on the game board.

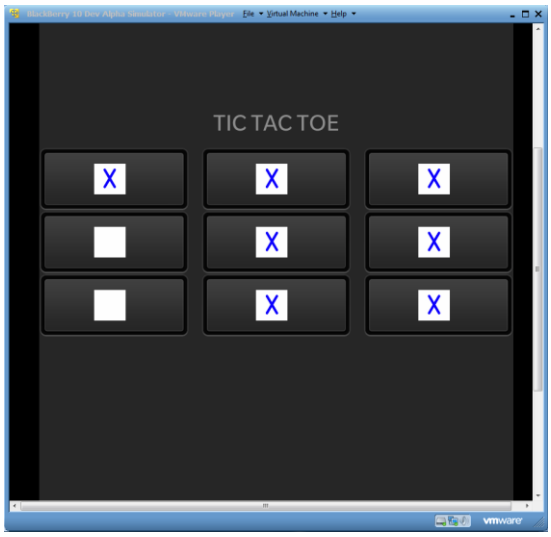

Figure 3

- Let's add a second player to the game. We're going to create a "computer" player. Any time the user chooses a spot, the computer can choose a spot right after, so we'll add this logic in the *spotSelect()* function after the code that handles the user's turn.

- Before starting to generate a spot for the computer, first check that there are spots left in ***remainingSpots*** by returning if ***remainingSpots***.*size()* returns less than 1.
- Create an integer variable for the computer's spot choice. Set the variable to zero for now.
- Next do the exact same thing that you did for handling the user's choice, except change the '**X**' to an '**O**', change **xImg** to **oImg**, replace **index** with the new variable you created for the computer's choice, and finally remove the index from ***remainingSpots***.

- Right now the computer will always choose the first spot. Let's change this so that it picks an available spot randomly.
  - Make a function called *computerGoForRandomSpot()* that takes no parameters, but returns an integer.
  - Change the computer's spot to equal *computerGoForRandomSpot()* instead of zero.
  - In *computerGoForRandomSpot()* return ***remainingSpots***[0]
  - Your *spotSelect()* function should now look something like the following:

```
void App::spotSelect(int index){
	if(spots[index] != '-'){
		qDebug() << "Spot already taken.";
		return;
	}

	//Handle user's choice
	spots[index] = 'X';
	switch(index){
		case(0): x0y0->setImage(xImg); break;
		…
		case(8): x2y2->setImage(xImg); break;
		default: return;
	}
	int toRemove = remainingSpots.indexOf(index);
	remainingSpots.removeAt(toRemove);

	if(remainingSpots.size() < 1){ return; }

	//Generate computer's choice
	int computerChoice = computerGoForRandomSpot();
	spots[computerChoice] = 'O';
	switch(computerChoice){
		case(0): x0y0->setImage(oImg); break;
		…
		case(8): x2y2->setImage(oImg); break;
		default: return;
	}
	toRemove = remainingSpots.indexOf(computerChoice);
	remainingSpots.removeAt(toRemove);
}
```

- Run the app again. You'll notice that the computer will just try to pick the next available spot. This isn't very fun, so next let's add randomization to its choice.
- To add randomization (note that it will be pseudo-random) to the computer's choices we will first need to create and initialize the random number generator. We're going to use Qt's *qrand()* function, which we can seed using *qsrand()*.

- Go back to your *initialize()* function.
- Before the call to *newGame()* add the following lines:

```
QTime midnight(0, 0, 0);
int seed = midnight.msecsTo(QTime::currentTime());
qsrand(seed);
```

  - The first line creates a QTime object named **midnight** and initializes it to hour zero, minute zero and second zero.
  - The second line calculates the number of milliseconds from **midnight** to the current time (essentially giving you an integer representation of the current time of the day in milliseconds).
  - The third line seeds *qrand()* with the new **seed**.
  - Note: Starting the App at a certain time on one day and starting it at the same time on another day (or starting the App on two devices at the same time) may cause the same seed to be generated. This will only happen if the QTime::*currentTime()* line for both days (or on both devices) is executed at the exact same millisecond. In these cases the same sequence of random numbers will be generated from *qrand()*. You should add better randomization to your App if you were using the random numbers for selecting a contest winner, for security purposes, or for other such things.
- Go back to your *computerGoForRandomSpot()* function.
  - Replace the function contents with the following:

```
int rand = int( qrand() % remainingSpots.size() );
return remainingSpots[rand];
```

  - The first line generates a random number, and then makes sure it is less than the size of the ***remainingSpots*** list.
  - The second line returns the index stored at the random spot in the list.

- Run your app again, and you should notice that the computer picks spots (pseudo)randomly.

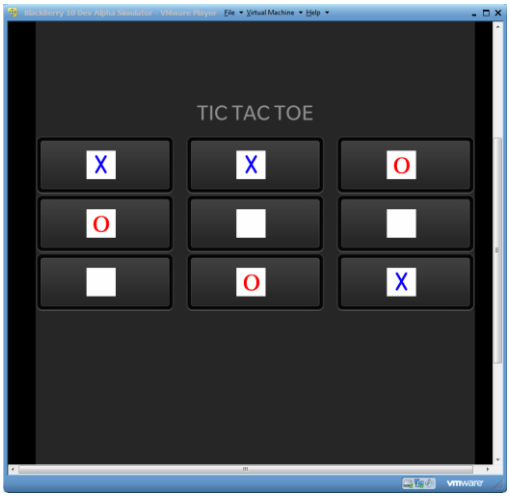

Figure 4

- In the next lab we'll add checking for win conditions, improve the computer logic, and add a custom icon and splash screen to our app.