Developing Native Apps for BlackBerry 10 with Cascades

Lab # 2: *Adding Game Logic and Improving the Game*

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 adding game logic as well as adding to the user interface. 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 a basic knowledge of C++, an understanding of Object Oriented programming, and assumes that you've completed the first lab so you should have a basic knowledge of Cascades, the BlackBerry 10 Native SDK, and Qt.

Exercise 2: Add game logic, improve UI

- 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).
- If your project from Lab 1 isn't listed in the Project Explorer pane of the IDE, click "File" > "Open File…" and then open the .pro file for the code you completed in Lab 1.
- In the Project Explorer pane on the left, expand the project folder, expand the "src" folder, and then open up the "app.cpp" source file.
- Ideally we want to end the game if there is a win, loss or tie. We also want to inform the player if they won, lost, or tied.
- First lets add an area to display text to the player
    - Go to the App constructor (*App()*).
    - Find the line where we add the **contentContainer** to the **appContainer** (i.e. the line **appContainer**.*add(***contentContainer***)*). We're going to want to add a label to **contentContainer** before we add it to the **appContainer**.
        - There is already a Label pointer variable in "app.hpp" called ***gameStatusLabel*** that we'll use.
    - Add the line: ***gameStatusLabel*** = new Label();
    - We want the label to be center-aligned horizontally inside of the **contentContainer** (which has a StackLayout). Look at the initialization of **titleLabel** in the App constructor to see how this is done.

    - Add the ***gameStatusLabel*** to the **contentContainer**.
    - In *newGame()* call ***gameStatusLabel****.setText("")* after calling *clearBoard()*.
    - In the App destructor, add: "delete ***gameStatusLabel***;"
- Now we can add checking for win, loss and tie conditions.
    - Create a new function called *checkWin()*. It doesn't take any parameters, but it returns a Boolean. If the player won it returns true, it returns false otherwise. Copy the code below into the function:

```
if ((spots[0]=='X')&&(spots[1]=='X')&&(spots[2]=='X')) return true;
if ((spots[0]=='X')&&(spots[3]=='X')&&(spots[6]=='X')) return true;
if ((spots[0]=='X')&&(spots[4]=='X')&&(spots[8]=='X')) return true;
if ((spots[1]=='X')&&(spots[4]=='X')&&(spots[7]=='X')) return true;
if ((spots[3]=='X')&&(spots[4]=='X')&&(spots[5]=='X')) return true;
if ((spots[6]=='X')&&(spots[7]=='X')&&(spots[8]=='X')) return true;
if ((spots[2]=='X')&&(spots[5]=='X')&&(spots[8]=='X')) return true;
if ((spots[6]=='X')&&(spots[4]=='X')&&(spots[2]=='X')) return true;

return false;
```

    - Create a new function called *checkTie()*. It doesn't take any parameters, but it returns a Boolean representing if the game ended in a tie (true) or not (false). Copy the code below into the function:

```
if ((spots[0] != '-')&&(spots[1] != '-')&&(spots[2] != '-')
  &&(spots[3] != '-')&&(spots[4] != '-')&&(spots[5] != '-')
  &&(spots[6] != '-')&&(spots[7] != '-')&&(spots[8] != '-')){
       return true;
}
return false;
```

    - Create a new function called *checkLose()*. It doesn't take any parameters, but it returns a Boolean representing if the computer won (true) or not (false). Copy the code below into the function:

```
if ((spots[0]=='O')&&(spots[1]=='O')&&(spots[2]=='O')) return true;
if ((spots[0]=='O')&&(spots[3]=='O')&&(spots[6]=='O')) return true;
if ((spots[0]=='O')&&(spots[4]=='O')&&(spots[8]=='O')) return true;
if ((spots[1]=='O')&&(spots[4]=='O')&&(spots[7]=='O')) return true;
if ((spots[3]=='O')&&(spots[4]=='O')&&(spots[5]=='O')) return true;
if ((spots[6]=='O')&&(spots[7]=='O')&&(spots[8]=='O')) return true;
if ((spots[2]=='O')&&(spots[5]=='O')&&(spots[8]=='O')) return true;
if ((spots[6]=='O')&&(spots[4]=='O')&&(spots[2]=='O')) return true;

return false;
```

    - Find the *spotSelect()* function. In the last lab we added a line before the computer's turn which checked if there were spots left in ***remainingSpots*** to prevent the computer from going when there were no spots left. We can now replace this line with checking if the player won, or if there was a tie.

        - If the player won, set the ***gameStatusLabel***'s text to a win message, and return.
        - If there was a tie, set the ***gameStatusLabel***'s text to a tie message, and return.
    - After the code for the computer's turn, check if the computer won.
        - If the computer won, set the ***gameStatusLabel***'s text to a lose message, and return.
- Test your app to make sure the game stops when it's supposed to and displays the corresponding message.
- You may notice that if you win before all the spots are taken, you are still able to put 'X's in the remaining spots even though the game is over. To prevent this you can use the ***playerTurn*** Boolean. This will also somewhat prevent the player from pressing buttons before the screen is ready and also from pressing buttons before the computer's turn ended.
    - At the very beginning of *spotSelect()* return if ***playerTurn*** is false
    - After checking that the spot the player clicked isn't already taken, set ***playerTurn*** to false
    - Set it back to true at the end of the computer's turn
    - Set it to false in *initialize()*
    - Set it to true in *newGame()*
- Next we'll add a button to start a new game, so that the player doesn't have to re-open the App every time the game ends.
    - Back in the App constructor, add a Button under where you added the ***gameStatusLabel***.
        - There is already a Button pointer variable in "app.hpp" called ***newGameBtn*** that we'll use.
    - Add the line: ***newGameBtn*** = new Button();
    - Set the text of the button (***newGameBtn****.setText("New Game")*).
    - Horizontally center the button, like you did for the ***gameStatusLabel***.
    - If you want the button to only be clickable when the game ends, set its enabled property to false here (*setEnabled(false)*). Re-enable it on a win, loss, or tie.
    - If you want the button to only be visible when the game ends, set its visible property to false here (*setVisible(false)*). Make it visible again on a win, loss, or tie.
    - Connect the ***newGameBtn***'s *clicked()* SIGNAL to App's *newGame()* SLOT. Refer to the game board's buttons if you're unsure of how to do this.
    - Add the ***newGameBtn*** button to the **contentContainer**.
    - In the App destructor, add: "delete ***newGameBtn***;"
- Run your app again. It should look something like **Figure 1**:

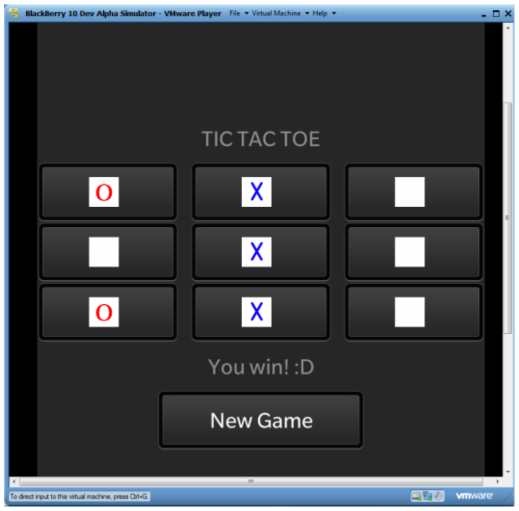

Figure 1

- Now we'll make the App more our own by replacing the default icon that shows up on the user's Apps screen.
  - Copy the "icon.png" file from the "lab2_images" folder from the start-up files directly into the project folder, overwriting BlackBerry's default "icon.png".
  - Open the "bar-descriptor.xml" file, and click on the Application tab. In the Icon field, you should see a tic tac toe image.
- Again we'll make the App more our own by replacing the default BlackBerry splash screen that shows up when the app starts.
  - Copy the "splashscreen-Portrait.png" file from the "lab2_images" folder from the start-up files directly into the project folder.
  - Open the "bar-descriptor.xml" file, and click the Application tab. There is a "Splash Screens" section. Beside the Portrait field, click the Browse button. Click the "Workspace…" button, then find and select the "splashscreen-Portrait.png" file and click OK.
  - Save the "bar-descriptor.xml" file
- We're not currently supporting landscape orientation, so we can skip setting the landscape splash screen.
- Run the app again. You should see it has a tic tac toe icon now, and when the App loads you should see something like **Figure 2**:

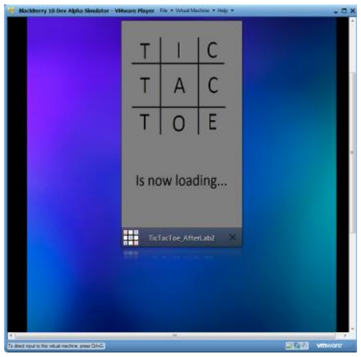

Figure 2

- Right now our “computer” player isn’t very good at playing. Let’s improve on this by adding some logic to make it try to win if it has two ‘O’s lined up, and try to block you from winning if you have two ‘X’s lined up.
    - Create a *computerGoForTheWin()* function that takes no arguments and returns an integer. Copy the following code into it:

```
if ((spots[0]=='O')&&(spots[1]=='O')&&(spots[2]=='-')) return 2;
if ((spots[0]=='O')&&(spots[1]=='-')&&(spots[2]=='O')) return 1;
if ((spots[0]=='-')&&(spots[1]=='O')&&(spots[2]=='O')) return 0;
if ((spots[0]=='O')&&(spots[3]=='O')&&(spots[6]=='-')) return 6;
if ((spots[0]=='O')&&(spots[3]=='-')&&(spots[6]=='O')) return 3;
if ((spots[0]=='-')&&(spots[3]=='O')&&(spots[6]=='O')) return 0;
if ((spots[0]=='O')&&(spots[4]=='O')&&(spots[8]=='-')) return 8;
if ((spots[0]=='O')&&(spots[4]=='-')&&(spots[8]=='O')) return 4;
if ((spots[0]=='-')&&(spots[4]=='O')&&(spots[8]=='O')) return 0;
if ((spots[1]=='O')&&(spots[4]=='O')&&(spots[7]=='-')) return 7;
if ((spots[1]=='O')&&(spots[4]=='-')&&(spots[7]=='O')) return 3;
if ((spots[1]=='-')&&(spots[4]=='O')&&(spots[7]=='O')) return 1;
if ((spots[3]=='O')&&(spots[4]=='O')&&(spots[5]=='-')) return 5;
if ((spots[3]=='O')&&(spots[4]=='-')&&(spots[5]=='O')) return 4;
if ((spots[3]=='-')&&(spots[4]=='O')&&(spots[5]=='O')) return 3;
if ((spots[6]=='O')&&(spots[7]=='O')&&(spots[8]=='-')) return 8;
if ((spots[6]=='O')&&(spots[7]=='-')&&(spots[8]=='O')) return 7;
if ((spots[6]=='-')&&(spots[7]=='O')&&(spots[8]=='O')) return 6;
if ((spots[2]=='O')&&(spots[5]=='O')&&(spots[8]=='-')) return 8;
if ((spots[2]=='O')&&(spots[5]=='-')&&(spots[8]=='O')) return 5;
if ((spots[2]=='-')&&(spots[5]=='O')&&(spots[8]=='O')) return 2;
if ((spots[6]=='O')&&(spots[4]=='O')&&(spots[2]=='-')) return 2;
if ((spots[6]=='O')&&(spots[4]=='-')&&(spots[2]=='O')) return 4;
if ((spots[6]=='-')&&(spots[4]=='O')&&(spots[2]=='O')) return 6;

return -1;
```

    - This code checks for any win condition. If it finds one, it returns the spot the computer needs to choose in order to win. If none of the “win” conditions match then it returns negative one.
    - Create a *computerGoForTheBlock()* function that takes in no parameters and returns an integer. Copy the following code into it:

```
if ((spots[0]=='X')&&(spots[1]=='X')&&(spots[2]=='-')) return 2;
if ((spots[0]=='X')&&(spots[1]=='-')&&(spots[2]=='X')) return 1;
if ((spots[0]=='-')&&(spots[1]=='X')&&(spots[2]=='X')) return 0;
if ((spots[0]=='X')&&(spots[3]=='X')&&(spots[6]=='-')) return 6;
if ((spots[0]=='X')&&(spots[3]=='-')&&(spots[6]=='X')) return 3;
if ((spots[0]=='-')&&(spots[3]=='X')&&(spots[6]=='X')) return 0;
if ((spots[0]=='X')&&(spots[4]=='X')&&(spots[8]=='-')) return 8;
if ((spots[0]=='X')&&(spots[4]=='-')&&(spots[8]=='X')) return 4;
if ((spots[0]=='-')&&(spots[4]=='X')&&(spots[8]=='X')) return 0;
if ((spots[1]=='X')&&(spots[4]=='X')&&(spots[7]=='-')) return 7;
if ((spots[1]=='X')&&(spots[4]=='-')&&(spots[7]=='X')) return 4;
if ((spots[1]=='-')&&(spots[4]=='X')&&(spots[7]=='X')) return 1;
if ((spots[3]=='X')&&(spots[4]=='X')&&(spots[5]=='-')) return 5;
if ((spots[3]=='X')&&(spots[4]=='-')&&(spots[5]=='X')) return 4;
if ((spots[3]=='-')&&(spots[4]=='X')&&(spots[5]=='X')) return 3;
if ((spots[6]=='X')&&(spots[7]=='X')&&(spots[8]=='-')) return 8;
if ((spots[6]=='X')&&(spots[7]=='-')&&(spots[8]=='X')) return 7;
if ((spots[6]=='-')&&(spots[7]=='X')&&(spots[8]=='X')) return 6;
if ((spots[2]=='X')&&(spots[5]=='X')&&(spots[8]=='-')) return 8;
if ((spots[2]=='X')&&(spots[5]=='-')&&(spots[8]=='X')) return 5;
if ((spots[2]=='-')&&(spots[5]=='X')&&(spots[8]=='X')) return 2;
if ((spots[6]=='X')&&(spots[4]=='X')&&(spots[2]=='-')) return 2;
if ((spots[6]=='X')&&(spots[4]=='-')&&(spots[2]=='X')) return 4;
if ((spots[6]=='-')&&(spots[4]=='X')&&(spots[2]=='X')) return 6;

return -1;
```

  - Go back to the *spotSelect()* function. We're going to now replace the line were we called *computerGoForRandomSpot()* with the new logic.
    - Replace *computerGoForRandomSpot()* with *computerGoForTheWin()*
    - If **computerChoice** is negative one, then set it to the returned value from *computerGoForTheBlock()*.
    - If **computerChoice** is still negative one, then set it to the returned value from *computerGoForRandomSpot()*.
- Loading the App onto a device
  - Make sure the device is on, and has development mode on (from the settings [swipe down the top bezel, or click the gear icon] and then selecting **Security**, and **Development Mode**), and that "Connect to Windows" or "Automatically Detect" is on if connecting to a Windows machine (settings > **Storage & Sharing**).
  - Connect the device and wait for it to be recognized on the computer.
  - The computer and device will prompt you for the password. Enter it into both.
  - If you have not set up the device when you were setting up your environment, then you'll have to follow additional steps to set the IDE to recognize the device:
    - Click **Window** > **Preferences**
    - In the left pane, click on **BlackBerry**
    - In the right pane, click the link to the **BlackBerry Deployment Setup Window**
    - Click **Next**

        - Check the box labeled “Device connected using USB”
        - Fill in the device’s IP address (this can be found on the Device by clicking the Developer icon in the top bar, or by going to the settings [swipe down the top bezel, or click the gear icon] and then selecting **Security**, and **Development Mode**), fill in the device password and then click **Next**.
        - The IDE will take a moment to connect to the device
        - Add the location of your registered BB10 signing keys in the field that asks for them and click **Next**
        - Generate a debug key, or (if you already have one created) add the location of an existing debug key to the field that asks.
        - You may need to generate a debug token on the device if it does not already have one.
        - Click **Next** then click **Finish**
    - Click the down arrow beside the Build button, and change the build setting from “Simulator-Debug” to “Device-Debug”. Click build.
    - Click on the down arrow beside the Run button, and click on “Run Configurations…” and add a new Run Configuration if you haven’t already made one for the Device. Enter your project name, or “Browse…” for it. Set the Build Configuration to match what you built the project with (Device-Debug). Select the device you added for the Target (or, if it does not show up, you might have to “Add New Target…”), and click “Apply”. You can choose to run it here by clicking “Run”, or close the Run Configurations window and run it from the main screen. To do so, hover over the Run button. If it says “Run <APP CONFIGURATION>” (where <APP CONFIGURATION> is the App Configuration you just made in this step) then you can click the button. If not, click the down arrow and select your new App Configuration from the list.
    - You should see your App working on the device as it did in the simulator.
        - If you get an error about the Debug Key being invalid or in the future, make sure that the date and time on the device is set passed the date and time that you created the debug token.
    - One final note: Since you cannot currently sign BB10 apps in Release mode the App will stop working on the Device when your debug token on the device expires, or if it is removed. To get it working again you just need to right-click on the device in the Project Explorer and click **Properties** then click **Blackberry Target** in the left pane, and click the **Debug Token Details…** button in the right pane, click the debug token in the list, and click the **Renew** button.