Microcontroller-Driven Touchscreen Game: Reversi

By: Andrew Walker

CWU Electronic Engineering Tech.

Abstract

The purpose of the project is to run a touchscreen applications on a 32-bit microcontroller. “Reversi” was chosen as the application to be developed for the project because it is a rules-light board game that requires a small amount of graphical resources and can be used to test the display of images and correlation to where the screen is touched and how the program responds by changing the display. The primary objectives include implementing the device libraries and drivers for the chosen microcontroller and display, getting the application to display the correct image in the correct place and detect a screen touch to correlate to the correct place on the touch screen.

Tools and Environment

MPLAB X IDE:
Modification of the Netbeans Development Interface, specifically for the creation of programs on embedded microcontrollers.

MPLAB XC32 Compiler:
A tool suite for compiling C/C++ code on a PIC32 Microcontroller.

MPLAB Harmony Integrated Software Framework:
A plugin for MPLAB X that replaces the old Application Libraries. Provides drivers for specific devices and functions which can be toggled on and compiled easily.

Graphics Display Designer X:
A plugin for MPLAB X for designing visual interfaces.

PIC32MZ EC Starter Kit:
Entry-level PIC32 Microcontroller board. USB powered, Ethernet-capable, and features a built-in code debugger.

Multimedia Expansion Board II (MEB-II):
An Expansion Board used to demonstrate the multimedia capabilities of the PIC32MZ Microcontroller. Has several media-based features, most relevant to this project being the 4.3” , 480x272 resolution touch-screen.

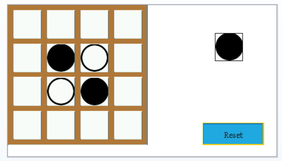
The Program

The game Reversi is traditionally played on an 8x8 grid by 2 players. The objective of the game is to have the most pieces on the board by the time the board is filled, or neither player can make another move. Players take turns placing pieces on the board that “sandwich” their opponent’s pieces to flip them. Players can only place pieces where it would capture one or more opponent’s pieces.

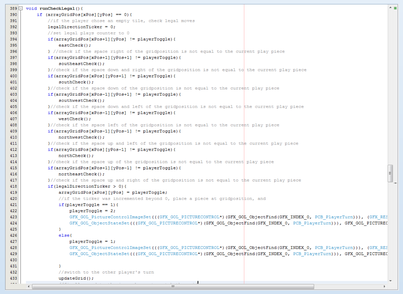
This version of the game is a 4x4 grid for a few reasons. First, a smaller grid is easier to test for errors. Second, with a resolution of 480x272 pixels, an 8x8 grid would be prohibitively hard to play due to the accuracy of the presses required. Finally, 4x4 is the smallest uniform size possible since the game must start with 2 pieces of each player’s in the center.

How it Works

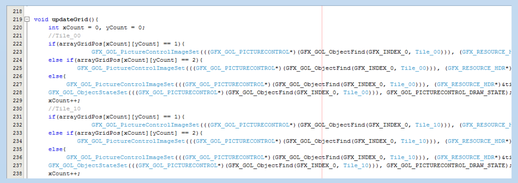
The interface consists of 18 image box objects and 1 button. The first 16 images make up the grid of the game board (named Tile_##, corresponding to their grid position for easy reference), the 17th is the board itself, and the final image represents the current player’s tile (who’s turn is currently “up”). The button, located on the lower-righthand corner, is a reset button.



Whenever a tile is “focused” (pressed), that tile will do two things. First, it will save it’s coordinates to a pair of integer variables named “xPos” and “yPos”. For example, if “Tile_23” is pressed, it will save 2 in xPos and 3 in yPos. Second, it will make a call to a function labelled, “runCheckLegal”. The job of runCheckLegal is to direct the code to determine if the pressed space represents a legal move, which is does by checking the corresponding location in an integer array. It begins the search by checking if the position on the grid is empty (represented by 0), and will then proceed to check if the adjoining positions (+1 or –1 in the x or y plane) have a value opposite of the current player (represented by either a 1 or 2).

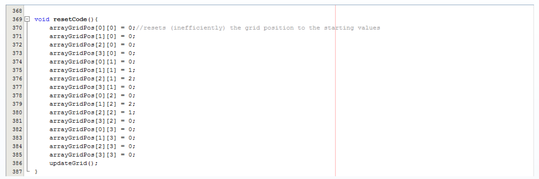


From there, the code will try and follow any potentially legal flipping moves by incrementing through a loop until it meets a terminating condition: either the direction hits a 0, a boundary of the array, or the current player’s piece. If the loop terminated via finding the player’s piece along the line, it will then de-increment back to the starting position and change the array value along the line. When all 8 directions have been checked, if any direction represented a legal play, it will finally place the player’s piece down, pass turns, and finally, update the images to correlate with their given grid positions.



How it Works (continued)

The reset button when released, manually sets the array values to their default states and then calls to the function that refreshes the images.

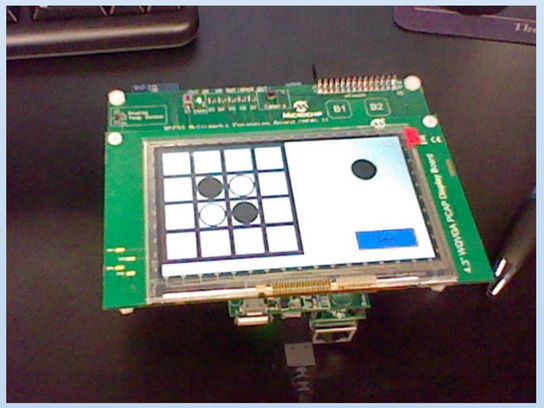


Multimedia Expansion Board II (MEB-II)

As mentioned in the tools section, the Multimedia Expansion Board II is a demonstration board for the PIC32MZ microcontroller made to show off it’s many media-savvy applications. Most relevant to this project is the “Projected Capacitive” touchscreen.

Projected Capacitive touch screens work by having a grid of conductive material in the glass, and when electricity is applied, it forms a uniform field. The screen uses disruptions in that field via other conductive materials (skin, for example) to locate the position of touches.

Other notable features of the device currently unused but potentially helpful include optional external memory, through the onboard MicroSD slot on the bottom of the board. This would allow for example, larger files to be included that might otherwise fill up the internal memory (such as flashier graphics).



The Harmony Configuration Tool

Rather than manually including every necessary file from a library of drivers that would be required for each individual project, Microchip has created “Harmony” to replace their old library packages. Driver packages can still be added manually from the various repositories, but now MPLAB X will prompt you on the creation of a new project on which features your program will need via checkboxes, and then retrieves and copies them into the project folder for use.

Combined with the installed demo programs in the library, this provides an easy way to jump right into coding the device with less headache. In this particular situation, the libraries “Object Layer Demo” under graphics demos served as the main reference point on what drivers where required, specifically the less obvious things like setting the clock and interrupt bits.

Acknowledgements

Special thanks to:

- Nathan Davis for the suggestion in board game
- Lad Holden for mentoring the project
- Greg Lyman for parts ordering

References

Amendments to header files enabling base code to compile:

“2.2.7 Building and Viewing Results of the Device." Graphics Display Designer X (GDD X) User's Guide. V2.20b ed. Microchip Technology, 2014. 54-56. Print.

Functional edit to the picturebox control definitions:

grafican (2015, Feb. 11). GFX_GOL_PictureControlImageSet causing build error in GDDX v2.20 Harmony v1.02 [Msg 2]. Message posted to http://www.microchip.com/forums/m845683.aspx

Pointing out header files for widgets where not included upon interface generation:

ysaitoysaito (2015, Mar. 24). Confusion on building the tutorials in Graphics Display Designer X plugin [Msg 5]. Message posted to http://www.microchip.com/forums/m852304.aspx