WiiPong

Andrew Barba abarba@ccs.neu.edu

Hardik Kapoor kapoor.h@husky.neu.edu

Description

Our project implemented an interactive pong game where a user can control his paddle using the Bluetooth enabled Wii remote. A ball is rendered and bounces around on screen, as the ball approaches the bottom of the screen the user should attempt to move the paddle under the ball and reflect it back in the other direction. The user earns one point every time he reflects the ball and the game ends as soon as the user misses the ball and it falls below his paddle.

We display the users score in realtime using the LED's on the ZedBoard. It is a binary representation of the score so it is highly suggested you master your power of 2's before playing.

To summarize the main features of the game:

1. Console based GUI
2. Bluetooth remote control for moving on screen paddle
3. Full collision detection
4. Realtime scoring using the LED's on the ZedBoard

In the code we heavily used Classes, Structs, Threads, file descriptors and writing to standard out. All of the code is organized into folders that separate the functionality of each component. This makes the code easy to read and even easier to navigate. All header files are fully documented so a new reader can easily learn how the components fit together without digging into implementation. Finally, a single Makefile is responsible for compiling the code into a `main` executable.

In order to expose our game to the end user we ended up building a single `render` function in the `Game` class that handled redrawing the state of the game on each *tick*. A *tick* in this case is just one iteration of the main run loop. In the *main* run loop we did 3 main things:

1. Update the state of the game
2. Render the game
3. Sleep for a fraction of a second

In another background thread is where we instantiated the Wii class and listened for button events. This allowed us to render the game independtly of the blocking call to listen for events from the remote.

The run loop would continue infinitely as long as the game is considered *not over*. We had a simple definition for *game over* which was just based on whether the ball had gone below the paddle in which case the user essentially missed the ball and lost. At this point the users final score would be printed to the console and to the LED's on the ZedBoard.

Relevance

This project heavily involved material presented in the first half of the course, specifically object-oriented design in C++, and Bluetooth IO with the ZedBoard and Wii remote. Here is an organized list of topics used in this project:

1. ZedBoard
2. Object Oriented Design in C++
3. Bluetooth enabled Wii remote
4. Makefiles
5. Shell Scripts

Something not discussed in the course is building any sort of Graphical User Interface (GUI). There are many libraries available online that help ease the process of creating a GUI in standard out but we ultimately chose to explore a custom solution to keep the dependencies for this assignment to a minimum.

Roles

Andrew Barba

Andrew was responsible for architecting the game. Proper object-oriented design, clean code and documentation were an important design goal of the project along with, of course, a compiling and working version of the code. Thorough testing of the code was done before ever running on a ZedBoard. Andrew simulated the moving paddle using the right and left arrow keys and a variety of board and paddle configurations. Since time with the ZedBoard and Wii remote could be limited due to a large number of groups working on final projects, we needed the confidence that we had a working game and could just focus on the hardware components when we were in the lab.

Hardik Kapoor

Hardik was responsible for the Wii remote connection setup and scripts so we could quickly and reliably connect the remote to the ZedBoard and begin testing our code. Hardik also filmed and edited the final version of our video that is linked below.

YouTube Link

https://www.youtube.com/watch?v=O8p-385OEZ4