Taylor Coogan & Jordan Volpe

ELE 409 SPRING 2017 Project

<u>Simon Says</u>

Objective:

- Create a project that utilizes and implements all parts of an embedded system.

Source_Code_Simon_Says.zip Folder Contents:

- Player1.py - Player 1 main Python script
- Player2.py - Player 2 main Python script
- sevenseg.exe - executable C program to control seven segment display
- ledset1.exe - executable C program to control LEDs
- sevenseg.c - C source code for controlling seven segment display
- ledset1.c - C source code for controlling LEDs

Section 1: Design & Implementation

Overview

Our project is based on the game Simon Says. Player 1 inputs a sequence of colors, and then Player 2 must accurately repeat that sequence. Then Player 1 must input that same sequence with one new color added, and Player 2 must repeat that. This continues, with the sequence length increasing by one each round, until either of the players inaccurately enters a sequence. When a player makes a mistake, they lose one of their three lives, and when a player loses all three lives the game ends.

The majority of our project was implemented in Python with some use of programming in C. There are two main python scripts, Player1.py and Player2.py. These are the files that will run the game. These python scripts make calls to C program executables as well, which are included in the project. More specific details about how to run our application can be found in Section 2 of this report.

Graphical User Interface (GUI)

We approached the GUI design by first considering what we wanted the game to look like. We knew that we wanted the game window to consist of 4 buttons, with one button in each quadrant. So we divided the size of the screen into the x,y coordinates necessary to draw 4 rectangles on the screen with Python. Each rectangle would be filled with a different color from the classic Simon Says handheld game (green, red, yellow, blue). We put all the code to draw these rectangles in a redraw() function in anticipation of redrawing the rectangles regularly during the game.

We also wanted some sort of visual feedback so the user would know they successfully pressed the button. We decided that having the pressed button flash would be a good solution. To implement the flashing button, we created a flash(color) function that would draw a lighter shade of the button's color, and then redraw the original color 0.5 seconds later.

We obviously also needed to be able to know which button the user tapped, so we needed to listen for touch screen taps (or clicks), and if a tap occurs, to get the x,y coordinates of that tap. We were able to implement this in Python, and were also able to only listen for button presses at appropriate times in the game, to avoid user confusion.

We also wanted to print text to the screen such as "You Win", "You Lose", "New Round", "Game Over", etc. We were able to create a Python function centerText(message) that would write a message to the screen, and we used that function throughout the game to implement this.

Game Logic

We needed to create the foundational logic for the operation of the game. All of this logic is done in Python in the main script for each player. The game starts with Player 1 inputting the initial sequence. The program knows how long the sequence length should be each round, and listens for that number of button presses from the user. Then, the inputted sequence is stored as a list called seq1. Then seq1 is sent over a TCP socket to Player 2. Once Player 2 receives the sequence, that sequence is flashed on their screen so they know which colors to attempt to repeat. At this point, Player 2 attempts to repeat the sequence and their input is saved as a list named seq2. Then, after seq2 is sent to Player 1, both players compare the two sequences. If there is a mistake, both players will find out and will display the proper text (ex. "You Win" or "You Lose"), otherwise the game continues. It is now Player 1's turn again. Player 1 must input the same sequence as before, but with one new color. To verify that Player 1 did this correctly, we perform the following for all rounds > 1: We take the new sequence and temporarily remove the last element of the list, then compare it to the old sequence. If they do not match, we know that Player 1 failed to remember their color sequence from the previous round. In this case, we send a special message to Player 2 through the TCP socket indicating Player 1's loss, as there is no other way for Player 2 to know this occurred. If they do match, the new sequence is sent to Player 2 and the game continues. Each Player starts with 3 “lives” and if at any point a player fails to correctly enter a sequence, they lose a “life” and the sequence resets. The game continues until a player loses all 3 lives, at which point “Game Over” displays on the screen, the game winner is displayed, and the application closes.

TCP Socket Communication

In order to pass the sequences and other information between boards, we needed a way to facilitate board-to-board communication. We did this in Python and decided to use TCP because we did not want to risk packet loss when communicating the sequence. We started with some Python sample code that our TA Riley gave us, however we needed to hard-code the boards’ IP addresses to get things working, instead of using gethostname() like most examples in the Python documentation.

In our application each player uses client socket code to send data to the other board, and server socket code when listening for the opponents sequence. We have a global variable called seqData which is a list of lists. When the server function receives data from the other player, it first checks if the data is the designated code that we set in order to signify that the other player lost. If it is not, it must be a sequence, and it appends this list on to seqData. Then we reference the appropriate index of seqData by using a counter and resetting it when necessary.

One challenge we faced was trying to send a list through the socket. We found that we could only send normal text data, and we could not find an easy way to convert list to string and then back to list. Our solution was to use the Python Pickle package which allowed us to convert the list into a byte stream before sending it through the socket, and then back to a list after it’s received on the other end.

Seven Segment Display and LEDs

Our application displays the sequence length to the seven segment display. We did this by using the fpga.c functions that we used in the class labs to create a C program called sevenseg. Sevenseg accepts one argument which is the number to display on the seven segment display.

Our application also displays each player’s number of lives on the LEDs. LEDs 0-2 display the player’s lives, and LEDs 7-9 display the opponent’s lives. We used the LEDset function in a C program to light up the appropriate LEDs, and we used a shift left operator in order to swiftly display both players lives count.

We were able to call these C executables from our main Python script by using the *call* function imported from Python *subprocesses.* This allowed us to easily change these displays throughout the game through calls to these C executables.

Section 2: Setting Up The Project

Setup:

1. Extract the contents of Source_Code_Simon_Says.zip to a folder somewhere on your computer. These are the files necessary to run the game.
2. Install the linux operating system from ELE 409 Lab 0 on 2 SD cards, put each of them into a board, and start up both of the boards. Have each board connected to a computer through the USB cables provided and to an ethernet port via an ethernet cable.
3. For each board, use the `screen` command (in Lab 1) to use the board's terminal. Type `ifconfig` to get the ip address of the device. Record the ip address of each board for later use. You will need them to secure copy the files in the next step, as well as later steps.
4. From a separate terminal on the device containing the source code (not connected to the board), secure copy (scp) the files from step 1 onto each board with the following command:

➢ `scp -r /FolderWith/ExtractedContents root@board.ip.address:~/Desktop/simonSays/`

Part 1: Get Necessary Python Libraries (Python 2.7 and Pygame)

Perform these steps for each board:

1. First we need to install python 2.7 on the board. From the board's terminal, type:

   ➢ `sudo apt-get install python`

2. Next we need to install the Pygame library on the board:

   ➢ `sudo apt-get install python-pygame`

Part 2: Define the Correct Socket Parameters (IP Addresses)

One of the boards will be running the Player1.py Python script, and the other will run the Player2.py Python script. You will need to edit the socket information in Player1.py and Player2.py so that the boards can properly communicate. Otherwise they will not know who to send and receive information to and from.

1. You should have both board's ip addresses recorded from earlier steps.

2. Go back to the terminal that is connected to the board (through the screen command) and navigate to the simonSays directory:
   - ➢ `cd ~/Desktop/simonSays`
3. Use vi or your preferred text editor to open `Player1.py` and add the following:
   - ➢ Line 100: Change the ip address to the ip address of the *other* board (the board that will be running `Player2.py`). Leave the port number the same, do not change it.
   - ➢ Line 169: Change the ip address to the ip address of *your* board (the board that will be running `Player1.py`). Leave the port number the same, do not change it.
4. Use vi or your preferred text editor to open `Player2.py` and add the following:
   - ➢ Line 99: Change the ip address to the ip address of the *other* board (the board that will be running `Player1.py`). Leave the port number the same, do not change it.
   - ➢ Line 143: Change the ip address to the ip address of *your* board (the board that will be running `Player2.py`). Leave the port number the same, do not change it.
5. Now the two scripts should be configured to communicate with the proper devices.

Part 3: Playing the Game

1. First, on the board that you decided will run `Player2.py,` navigate to the folder with the project files and run `Player2.py`:
   - ➢ `python Player2.py`
2. Next, on the board that you decided will run `Player1.py,` navigate to the folder with the project files and run `Player1.py`:
   - ➢ `python Player1.py`
3. Now the game should be ready to play. The game always starts with Player 1 inputting their sequence, so make sure that you start the game on the right board. If you try to start with Player 2, it won't respond.
4. You now should have everything properly configured. Enjoy the game!
   - ➢ **NOTE:** do NOT touch the screen if it is not your turn to enter a sequence.