ECE 2534 Lab 4 – Fall 2015 – OLED Zombie Apocalypse

You must complete this assignment individually.

Honor Code Requirements

For laboratory assignments, students are not allowed to share or discuss any element or detail of a design or solution with any other student. Each student must treat all such details as proprietary. In particular, you may not share or discuss any specific algorithms or any source code with any other student; obtaining source code from any source other than your instructor, your course notes, Analog Devices reference code, or the PIC32 peripheral libraries is a violation of the Virginia Tech Honor Code, and will be prosecuted as such.

Overview

Your goal is to design and implement an OLED-displayed game where the objective is to avoid having a human player eaten by zombie player. The human is moved by tilting the PmodACL 3-axis accelerometer, however the zombie will also move in the direction of the human. You should build upon the code created for HW6. The game's interface and sophistication is up to you, although restarting the game should be accomplished with a single tap of the PmodACL in the z-axis, and single tap detection must use interrupts. In addition to code, you need to submit a PDF document that describes how to use your game and highlights any added features. For example, one could select more than one zombie, or the zombie(s) speed could be increased after each feeding.

Behavioral specification

1. At initialization you should create an appropriate splash screen with text that persists for about two seconds.
2. If there are any selections such as a difficulty level, additional screens should be displayed and options should be selected with debounced push buttons.
3. After the user selects the interface, the main playing field should appear on the OLED. The human and zombie(s) initially appear at random places on the screen. The size and design of the players and screen are up to you: 8x8 pixel glyphs can be used or you can define custom characters to create 4x4 pixel glyphs so that each cursor position can have 4 human or zombie positions. As you tilt the accelerometer, the human should move as if it is under the influence of gravity with a small amount of friction from wind resistance. If the human hits the border of the display, the human should undergo a perfectly elastic collision (i.e. its direction will change, but it speed will remain the same).
4. The goal of the game is to see how long you can prevent the human from being eaten by the zombie(s). The game is over when the human and zombie are in the same location. One way to score the game is to report how much time elapsed between the game starting and the game ending.
5. At any time tapping the accelerometer in the z-axis should send the program back to step 2.

More notes and requirements

1. You should enable "`additional warnings`" under `File` → `Project Properties` → `xc32-gcc` → `Errors and Warnings`. Develop your code so that no warnings are generated.
2. Do not use `float`s or `double`s. The PIC32 does not have floating-point hardware, and floating-point operations are much slower than integer operations. For the same reason, do not use transcendental functions from the math library such as sin() or `cos()`.
3. You must implement your main routine as a state machine that is non-blocking (at the human time scale). There should be no explicit delays (tight loops with no purpose other than to burn up cycles) in the code implementing the state machine.

4. You should implement your code in small steps that you debug as you go, so that you are always building on a strong foundation. As an example, the next section describes an incremental approach to detecting single taps.
5. The velocity of the human should be proportional to the degree of tilt, while the undead might have a constant velocity that is independent of tilt. If you wish to model the effects of gravity, the equations of motion for the *x* direction are:

$$\frac{dp}{dt}(t) = v(t)$$

$$\frac{dv}{dt}(t) = a(t) - \text{sgn}(v) \cdot c$$

where sgn(*v*) is the sign function returning -1, 0, or +1. In these equations, *p* is the *x*-direction position, *v* is the *x*-direction velocity, *a* is the acceleration due to gravity in the *x*-direction (from the accelerometer) and *c* is a small constant representing friction from moving through the air. For simplicity we will use a simple Euler integration scheme to solve these equations:

$$p(t + \Delta T) = p(t) + v(t) \cdot \Delta T$$

$$v(t + \Delta T) = v(t) + a(t) \cdot \Delta T - \text{sgn}(v) \cdot c \cdot \Delta T$$

where ΔT is the time step that you choose. Similar equations are needed for the *y* direction.
6. Since the `OledUpdate()` function can take a long time to execute, you must ensure that it does not interfere with updating player positions.
7. As in HW6, the game should work without source code changes regardless of whether the PmodACL is plugged into connector `JE` or `JF`.

Single tap detection guidance

Detailed guidance is given here since you won't have CEL support during Thanksgiving Break, and the lecture slides are a little misleading since they describe using `CNx` input pins rather than the actual `INTx` input pins connected through `JE` and `JF`.

1. You ultimately want to add the following function to the ADXL345 (accelerometer) layer:
   ```
   bool ADXL345_SingleTapDetected()
   ```
   This function needs to be defined in `ADXL345.c`, with a prototype declared in `ADXL345.h`. The `bool` (Boolean) type was added to C and likely requires the following to be added to the start of `ADXL345.h`:
   ```
   #include <stdbool.h>
   ```
   Even though the `int` type can also be interpreted as true (any non-zero value) or false (a zero value), it is clearer to use the `bool` type in this situation.
2. The ADXL345 can be configured to assert its `INT1` or `INT2` output pin when any of 8 events occur including a single tap. I don't recommend using Analog Device's `ADXL345_SetTapDetection()` function which has a complex interface. As indicated in the ADXL345 data sheet's discussion of interrupts, you need to:
   a. Enable z-axis tap detection by writing to the `TAP_AXES` register.
   b. Possibly make adjustments to the `DUR` and `THRESH_TAP` registers. I found that `0x10` and `0x60` were reasonable values for these registers, but you should experiment with adjusting the sensitivities.
   c. In order to decide between the PmodACL's `INT1` and `INT2` outputs, you need to look up the Pmod pins assigned to these outputs and see what they connect to on the PIC32 by consulting Digilent's ChipKit Pro MX7 Reference Manual. You are looking for a PIC32 pin that can generate an external interrupt to the PIC32. Although the lecture slides imply that a `CN` (Change Notice) pin will be used, there is only the PIC32 `INT1` input on the `JE` connector, and the PIC32 `INT2` input on the `JF` connector. Even though these PIC32 inputs have the same names as the

ADXL345 outputs, don't assume they are tied together. In fact you will be connecting the ADXL345's INT2 output pin to the PIC32's INT1 input pin when the JE connector is used, and the ADXL345's INT2 output pin to the PIC32's INT2 input pin when the JF connector is used.

   d. The PIC32's INT1 and INT2 input pins need to be set as digital inputs using a PLIB function similar to that used in HW6.
   e. The ADXL345's INT_MAP register needs to select the ADXL345's INT2 output pin for single tap events.
   f. The ADXL345's INT_ENABLE register needs to enable interrupts for single tap events.

3. At this point the ADXL345's INT2 output pin and either the PIC32 INT1 or INT2 input pin (depending on whether connector JE or JF is used) should be asserted when a z-axis single tap is detected. Even though an interrupt signal is asserted by the ADXL345, the PIC32 has not yet been set up to generate interrupts. Before enabling PIC32 external interrupts, this would be a good time to confirm you have configured the ADXL345 correctly by seeing if the PIC32's INT1 and INT2 inputs are asserted when the ADXL345 is tapped in the z-axis. This is as simple as using a PLIB function in the ADXL345_SingleTapDetected() function to read the state of the port and pin corresponding to the PIC32 INT1 and INT2 input pins. You definitely want to make sure these pins are being asserted before setting up external interrupts on the PIC32.
4. If multi-vector interrupts are used, then the PIC32 INT1 and INT2 external interrupts will each need their own ISR. These ISR's should be put in the ADXL345.c file since they are part of the ADXL345 layer. The vector names for these interrupts are _EXTERNAL_1_VECTOR and _EXTERNAL_2_VECTOR, and the flags (which always need to be cleared at the end of an ISR) are INT_INT1 and INT_INT2.
5. As indicated in the ADXL345 data sheet's discussion of interrupts, the ISRs need to read the INT_SOURCE register in order to clear the interrupt. Note that the ISRs must clear the interrupt source in the ADXL345 *and* the interrupt flag in the PIC32's interrupt controller.
6. The ADXL345_Init() function needs to configure interrupts on a rising edge of either the INT1 or INT2 external inputs. This can be accomplished by using the ConfigINTx() PLIB function to enable external interrupts, select the active edge, and assign the interrupt priority.
7. With interrupts enabled, the ADXL345_SingleTapDetected() function can now be changed to look at the contents of the INT_SOURCE register read by the ISRs and saved in a static (file scoped) interruptSource variable, and check if the SINGLE_TAP bit has been set. If so then the interruptSource variable can be cleared. Use of the ADXL345_SingleTapDetected() function is much better than sharing the interruptSource variable across files since the implementation of ADXL345_SingleTapDetected() could change substantially from no interrupts to interrupts without requiring any changes to code using this function. In general function calls should be used instead of sharing global variables across functions.

Submitting your work

Submit a zipped archive containing your complete source code and a PDF document describing how to run your game, including any special features. A grader should be able to unzip the archive, open the project, and run it as is. Use this naming convention for the archive that you submit:

<Last name>_<First name>_Lab4.zip

Replace <Last name> and <First name> by your family name and given name respectively.