ECE177: Programming I: From C...

Lab #8 — Keypad Movement

Week of 6 April 2026

1 Objectives

- Complete all of the steps outlined below while working on `paddle.c`
- Program the Pico breadboard with the modified `paddle.c` file and confirm working output

2 Materials

- Assembled and working Pico breadboard
- USB cable
- Laptop and Laptop charger
- Completed Lab#7

3 Procedure

1. Make sure to read all of the directions carefully before getting started.
2. **NOTE: Unlike previous labs the directions will be in this document and not in the comments of the code, as you will be using your code from lab7**
3. Download the `lab08_code.zip` file here:
   `https://web.eece.maine.edu/~vweaver/classes/ece177/labs/lab08_code.zip`
4. Extract the zip file in your `Documents/ece177/labs/` directory that you created in lab01.
5. In VS Code, use the Raspberry Pi Pico Extension to import the `lab08_code` folder as a project.

4 Copy your Lab#7 Code into this Project

1. Copy your `bounce.c` file from HW#7 into the `lab08_code` directory, but be sure it gets the name `paddle.c`
2. If you're doing this in a GUI this might take two steps, one to copy `bounce.c` file in to the `lab08_code` folder/directory and another to rename it to `paddle.c`
3. **Be sure the final file is called** `paddle.c` **and not** `paddle.c.c`. Windows hides file extensions and if you accidentally make the second, the build scripts won't be able to find it.

5 Modify the Code

1. Edit the copied over `paddle.c` file
2. Be sure to comment your code!
3. Test your code often! Don't try to write it all at once!
4. As with previous labs ideally you can just press the VS Code "Run" button and it will compile and upload your program to the Pico. If it doesn't you might have to find the `paddle.uf2` file in the `build` subdirectory and copy it over to the Pi Pico virtual drive.

Section A: Update the Code Comments

(a) Update the code comments at the top of the file to reflect that it's Lab#8 not Lab#7. Be sure it has your name, the date, and a brief description of the lab.

Section B : Add a Life Count

(a) Add a global variable called `lives` and initialize it to `3`.
Reminder, global variables are visible to all functions and you declare them outside of `main()`. There should be a place in the file that already has some previous globals from before, put it there.

(b) Find the existing code you wrote last time that checks if the ball hits the bottom of the screen. We are going to change this code so instead of bouncing, you lose a life.

(c) There are two cases to handle described below

(d) Case 1: If `lives` is equal to `0` we have run out of lives. Do the following:

i. Use `printf()` to print "Game Over".
ii. Play a sound effect for 100 ms using `play_tone()`.
iii. Then call `return` to exit game

(e) Case 2: If `lives` is greater than `0` do the following:

i. Decrement the `lives` variable
ii. Call an `update_lives()` function that we will write in the next step
iii. Reset the ball location.
Set `ball.x` to be a random number between `0` and `XMAX`. Use the `rand()` function for this.
   A. You might need to add an include of `stdlib.h` to your C program
   B. `rand()` returns a number from 0 to MAXINT (largest int)
   C. To reduce the range, you can use the modulus/remainder operator (`%`). For example, to get a random value from 0 to 9 you'd do `rand()%10`
iv. Set `ball.y` to be YMAX divided by 2
v. Set `ball.yadd` to be `-1` so it is moving upward
vi. Finally play a long tone on the speaker. Use `play_tone()` with a value of 100

(f) Setting up `update_lives()` function

i. Add a function prototype toward the top of the program for `update_lives()` that takes no parameters and returns nothing

ii. Add the actual `update_lives()` function down at the end of the source code file

iii. We don't want to use `printf()` to print the remaining lives as this clears the screen. Instead we will write the text using graphics routines.

iv. Create a string that contains "LIFE: X" where instead of X you have the number of lives

v. Use `sprintf()` for this. It's like printf but prints to a string. Something like:

```
char string[100];
sprintf(string,"LIFE: %d",lives);
```

vi. You can arbitrarily use 100 for the size of this string, though you can try to use a more appropriate size if you want.

vii. Note to use `sprintf()` you should include `string.h` at the top of the source code

viii. Use the function

```
size_t graphics_drawText(char *s, uint16_t x, uint16_t y);
```

to draw this string at location 0,0

ix. In addition to calling this function after lives has been decremented, also call it once just before the main game loop (near the top of `main()`. This will print the initial lives value at the start of the game.

(g) It turns out the `drawText()` code draws transparent text (so the background color shines through). To make sure we can actually read the number of lives you'll have to erase the old text. Add something like this to the beginning of the `update_lives()` function:

```
Adafruit_ST7735_fillRect(0,0,160,10,0x0000);
```

which will erase the text area at the top of the screen.

Section C: Draw the Paddle

(a) Setup a struct to hold the paddle's state. The struct type should be `paddle_state` and then declare a `struct paddle_state` named `paddle`. This struct should contain elements `x`, `y`, `xsize`, and `ysize` that are signed integers, and `color` that's a short integer

(b) Initialize the `paddle` structure before the main game loop (put it after the code used to initialize the ball). Initialize it with the following settings:

i. `x` should be the middle of the screen (make it half the value of the provided `XMAX`)

ii. `y` should be near the bottom of the screen. Make it `PADDLE_Y` and define `PADDLE_Y` to be `300` with a `#define` at the top of the program code.

iii. `xsize` is how wide the paddle is. Make it `64`

iv. `ysize` is how tall the paddle is. Make it `10`

v. Color should start out as white (this is a 16-bit value that's all 1 bits)

(c) Erase and Draw the paddle

i. Use `Adafruit_ST7735_fillRect()` to draw the paddle.

ii. Use the parameters `paddle.x`, `paddle.y`, `paddle.xsize`, `paddle.ysize` and `paddle.col`

iii. Add the paddle drawing routines at the end of the main game loop.

iv. You will add two calls:
First add one that erases it by calling `Adafruit_ST7735_fillRect()` with color 0 (black).
Immediately after add a call to `Adafruit_ST7735_fillRect()` that draws the paddle using `paddle.color`

Section D: Paddle Collision Detection

1. We want the ball to bounce off of the paddle
2. Add some collision check code as described here after the wall collision detect code
3. Check if `ball.y+YSIZE` is equal to `paddle.y`
4. If that's true, then also check if `ball.x` is greater than `paddle.x`
5. If that's true, then also check if `ball.x` is less than `paddle.x` plus `paddle.xsize`
6. If these all true, we have hit the paddle, so do the following, the code is similar to what used to happen when the ball hit the bottom of the screen:
   (a) Flip the y direction by negating `ball.yadd`
   (b) Play a short collision tone

Section E: Keypad Code

1. First, remove the `jump_to_MSD()` function from the end of the while loop
2. Next create an `init_keypad()` function at the end of the file. It can take no parameters and also can return no results. Add a corresponding function prototype at the top of the file with the other function prototypes.
   (a) For the contents of this function we will cut and paste in the keypad GPIO init code from Lab#6
   (b) Make sure this `init_keypad()` function is called at the start of your `main()` function
3. Next create a `read_keypad()` function. This should return an `int` but take no arguments. Again put a corresponding function prototype at the top of the file.
   (a) Copy into this function the keypad scan code from Lab#6 Section D part B.
   (b) In that Lab this was inside a while loop, but in our case we don't want the while, we just want the code that does the scan. Each time we call `read_keypad()` it should do one scan
   (c) At the start of this function declare an `int` called `result` which you initialize to `-1`. (We will use `-1` to mean no key was pressed)
   (d) At the end of the function `return` this `result` value.
   (e) In the code, instead or using `printf()` to show the row/column, instead take the row/col value and set `result` to be `((row<<4)|col)`

(f) This lets us return the row and column values in a single value, with the top 4 bits the row and the bottom 4 bits the column.

(g) To make parsing this custom row/column value, add the following `#define`s to the top of the file:

```
/* row/column */
#define KEYPAD_1 ((3<<4)|3)
#define KEYPAD_7 ((1<<4)|3)
#define KEYPAD_A ((3<<4)|0)
#define KEYPAD_C ((1<<4)|0)
#define KEYPAD_D ((0<<4)|0)
```

Section F: Add code to allow disabling sound

1. Create a global variable called `sound_enabled` and initialize it to `1`
2. In the `play_tone()` function surround the entire contents of the function with

```
if (sound_enabled) {
   ....
}
```

Section G: Add Keyboard Handler

1. Add this code to `main()`, near the end of the game loop in between the paddle erase and paddle draw calls
2. Read the result of `read_keypad()` into an integer
3. Use a switch/case construct on the integer returned from `read_keypad()` and handle the following cases:
   (a) `KEYPAD_1`, move paddle left. In this case check if paddle.x is greater than 0. If so, decrement `paddle.x`
   (b) `KEYPAD_A`, move paddle right. In this case check if `paddle.x` plus `paddle.xsize` is less than `XMAX`. If so, increment `paddle.x`
   (c) `KEYPAD_C` set `sound_enabled` to `0` to disable the sound
   (d) `KEYPAD_7` set `sound_enabled` to `1` to enable the sound
   (e) `KEYPAD_D`, return to boot mode. Call

```
printf("Reboot to MSD\n");
sleep_ms(2000);
reset_usb_boot(0,0);
```

4. Remember to call `break` properly at the end of each set of case statements

6 Troubleshooting / Common Issues

1. If your keypad code isn't working, be sure you remember to call `init_keypad()` in main()
2. If the keypad code still isn't working, be sure you are passing back row/column properly. If your printf was printing `row-9` or `col-5` because your loops were doing fancy stuff, be sure you subtract the right values off when creating the result you pass back.

7 Grading/Checkoff

1. Upload your `paddle.c` file to BrightSpace.
2. Upload a picture of your board running and showing the code running.
3. TAs will check the output of your final program running on your hardware. They will ask you to demonstrate the lab requirements and will enter grades after the requested information has been uploaded on the assignment rubric.
4. TAs: Enter grades only after verifying full functionality of the code running, the requested information has been uploaded, and the assignment has been submitted.