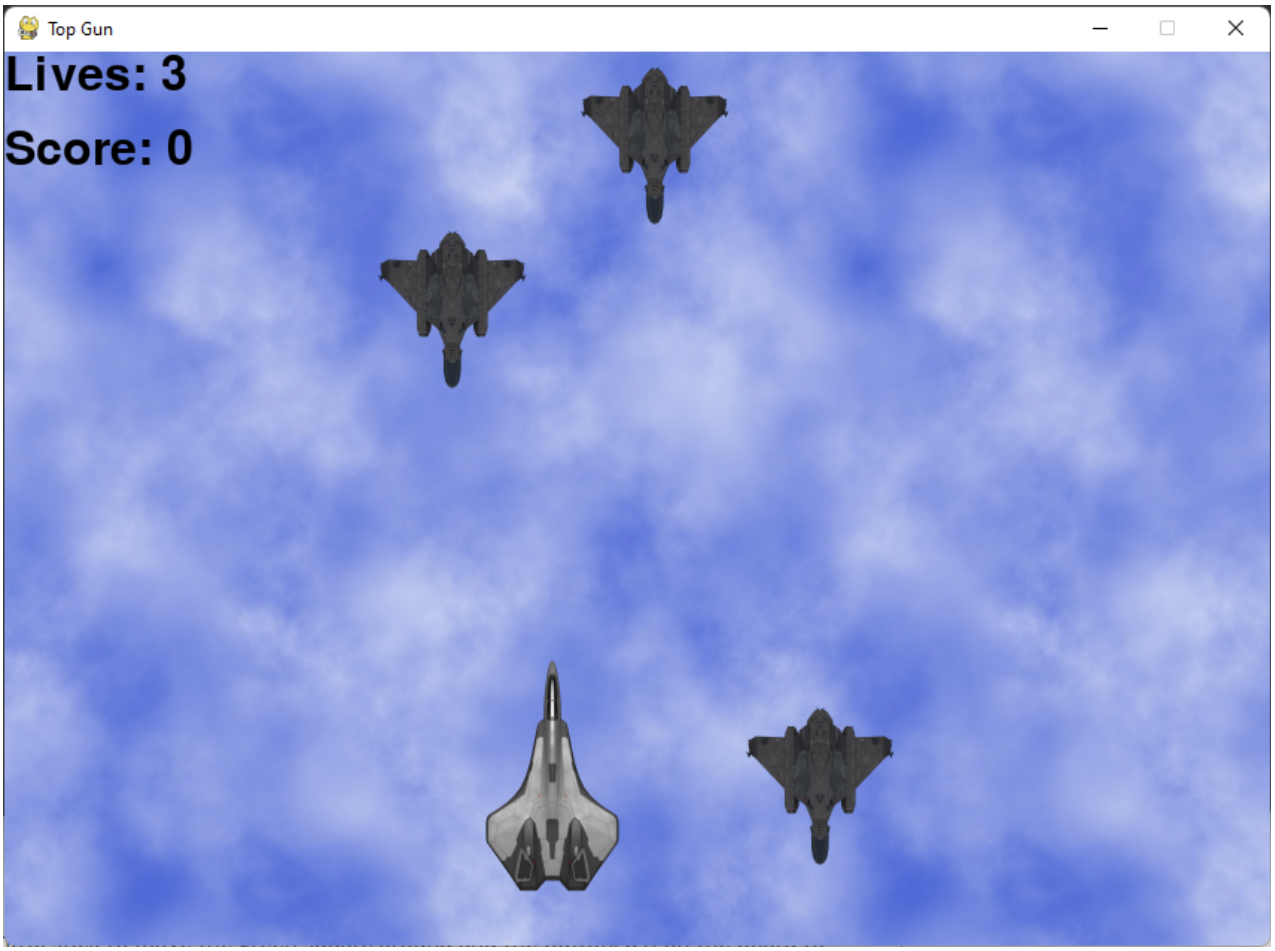
Lab 5 - Game Controller

Due Aug 8 by 11:59pm **Points** 100 **Submitting** a file upload
File Types txt and zip **Attempts** 1 **Allowed Attempts** 2
Available after Jul 31 at 12am

Introduction

In this lab, we will be creating a program to have our BOOSTXL-EDUMKII boards function as a controller for a python-based video game. A very basic Top Gun themed arcade game has been written in python and configured to listen to bytes coming across a serial port for inputs.



In this game, the player is represented by the gray jet and enemy jets will randomly spawn across the screen. The user need to avoid being hit by the jets while trying to shoot down as many as possible.

In this lab we will utilize the MSPEXP-430FR2355 as our game controller which will read inputs from buttons and the analog potentiometer to send commands to the jet through UART and USB.

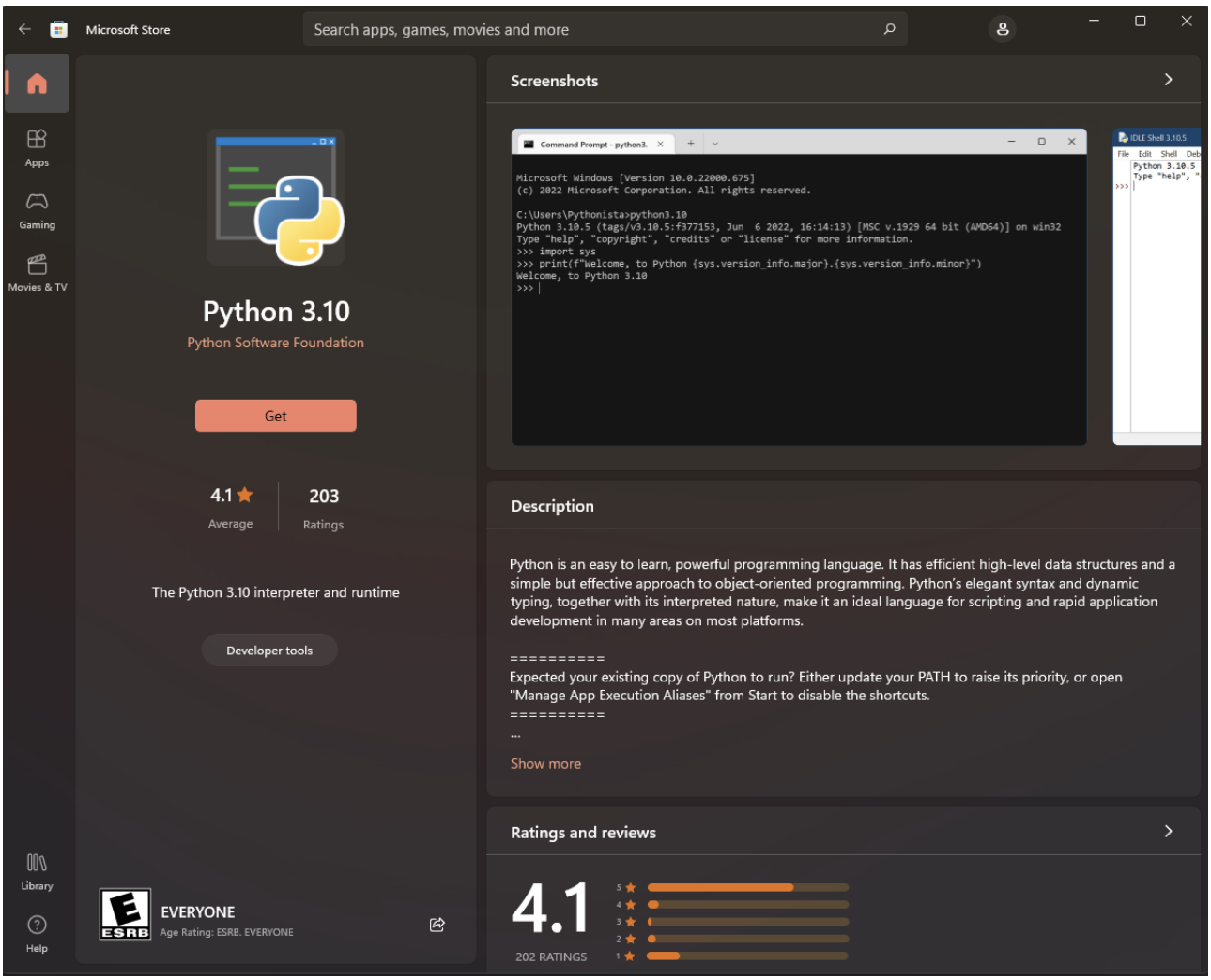
Required Materials

- MSP-EXP430FR2355
- USB Micro Cable
- Laptop (Windows preferable)
- Grove Potentiometer

Procedure

Step 1: Installing Python

The game itself is written in python, so we'll need to install python 3 and a few dependencies. If you are on a Windows 10 or 11 PC, the easiest way to get python is through the Microsoft store. Open your start menu and search for "Microsoft store". Once you have launched it, search for python in the store. You should find the following page:



Your search may yield multiple versions of python, but you should select 3.10 (other versions that are 3 should work just fine, however). To download and install python, click the "get" button and it should

begin installing.

Once installed, we will need to open a shell terminal and install a few other dependencies. Open your start menu and search for "powershell" and launch it. This will open up a windows command line terminal, where we will launch our python game and install dependencies.

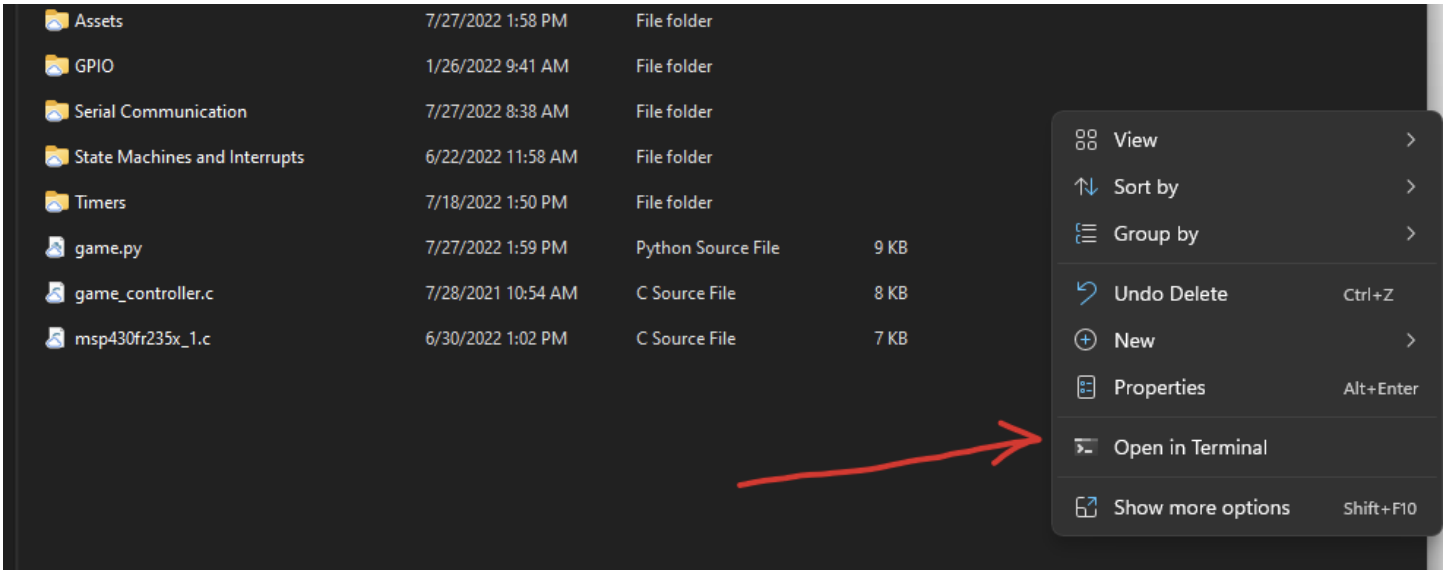
Once the shell is open, type in the following commands to install "pyGame" and "pySerial":

```
pip install pygame
```

```
pip install pySerial
```

Once installed, we are ready to download the game script and run it. Find the game script and associated assets **here (https://uncc.instructure.com/courses/197484/files/21699372?wrap=1)** **(https://uncc.instructure.com/courses/197484/files/21699372/download?download_frd=1)** . You will need to download this compressed folder and extract it. Once extracted, you will need to navigate to the directory where the script is stored in powershell (use the "cd" command to navigate to different folders, like you would in a Linux terminal.).

Note: On windows 11, you can right click in the folder and select "Open in Terminal" to quickly get a powershell window in the same directory as your game.py script. (this option might not be available on all versions of windows)



Once you are in the same directory, use the following command to launch the game:

```
python game.py
```

Note: if you are using windows and your game has no sound, it may be because of a missing dll file. It gets installed by the library but does not get placed in the System32 directory where it is expected

to be. The file is libmpg123-0.dll and can be found in:

```
C:\Users\<YOUR USER FOLDER HERE>\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\pygame
```

Take this file, copy it, and paste it in you System32 folder, which can be found at:

```
C:\Windows\System32
```

After you have done this, you should be able to load the music files. You may need to reboot your computer.

Note: You need to put your user folder name in place of <YOUR USER FOLDER HERE> in the file path, and your Python directory might be a slightly different name depending on the version. My windows is also installed on the C: drive, but that could also be different for you. AppData is also a hidden folder, so you will need to turn that on to see it as well.

Step 2: Create a New CCS Project

Before writing any code, go ahead and create a new project in CCS using the steps **here (https://uncc.instructure.com/courses/197484/pages/creating-a-new-ccs-project)** .

Step 3: Initializing the UART

The first step is to initialize the UART. The game is configured to look to open a serial at 9600 baud, 8 data bits, no parity, and 1 stop bit. We'll need to initialize our UART to do the same. Create a function to initialize the eUSCI_A1 module in UART mode with these parameters and call it at the beginning of your code.

Remember that the eUSCI's baud rate generator requires you to populate some register values according to the eUSCI's clock source. So you will need to configure one of the peripheral clock sources to run at an expected rate, then configure your eUSCI_A1 peripheral to use that clock signal. Once the clock signal is defined, use the formula found in the user's guide to populate the baud rate generator registers to produce the 9600 baud.

Don't forget as well to set the PxSEL pins to bypass the GPIO peripheral so that the UART can use the associated pins for communication!

Step 4: Writing Data and Verifying the UART

Once your UART is initialized, we need to create some functions to send data and verify that it's configured correctly. To write data to the UART, data needs to be written to the transmit buffer register. Create a function that takes an unsigned integer and writes to the tx buffer register. Inside your main while loop, put in a temporary __delay_cycles(10000) function and a call to your UART

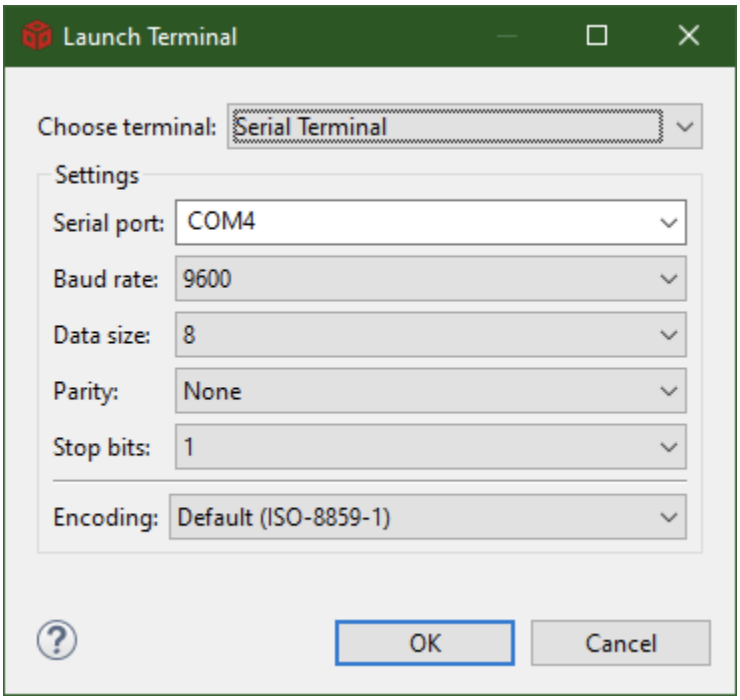
write function and pass in some data (doesn't matter what) to test it out.

Note: To prevent yourself from overwriting data that is in the process of being transmitted, we can check the interrupt flag register with the following line of code:

```
while(!(UCA1IFG & UCTXIFG));
```

This will mask the TX flag that is set high while a transmit is in progress and stall with a while loop. Include this line in your function before you write to the TX buffer so you don't accidentally overwrite the data you are in the process of sending.

Now that your write function and test code is written, let's open up a terminal to verify that all is correct. Enter debug mode in CCS and run your code. Once it's running, go to View->terminal to open up the terminal. Then click the blue "computer" icon on the terminal tab that just opened to configure it to open a serial port and set the serial settings to 9600 8N1:

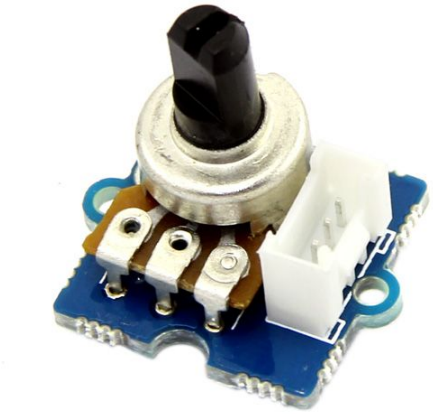


The default option might not be "Serial Terminal" so be sure to select that first from the list of options. Next, you need to determine which serial port is associated with the "Backchannel UART" of the development board. You will probably have several "COM" ports to choose from. To find out which is correct, open the "device manager" (search for it in your windows start menu) and look under "Ports" to find the names associated with each port. Select that port and hit "OK". If everything is working correctly you should see your test code printing some data to the terminal.

If you do not see data being printed at this point, your configuration is wrong and you should resolve it before moving on.

Step 5: Reading Inputs

The next step is to create functions to collect the user input. We'll be using two items for collecting user input: the on-board buttons, and the "rotary angle sensor" (which is just a potentiometer).



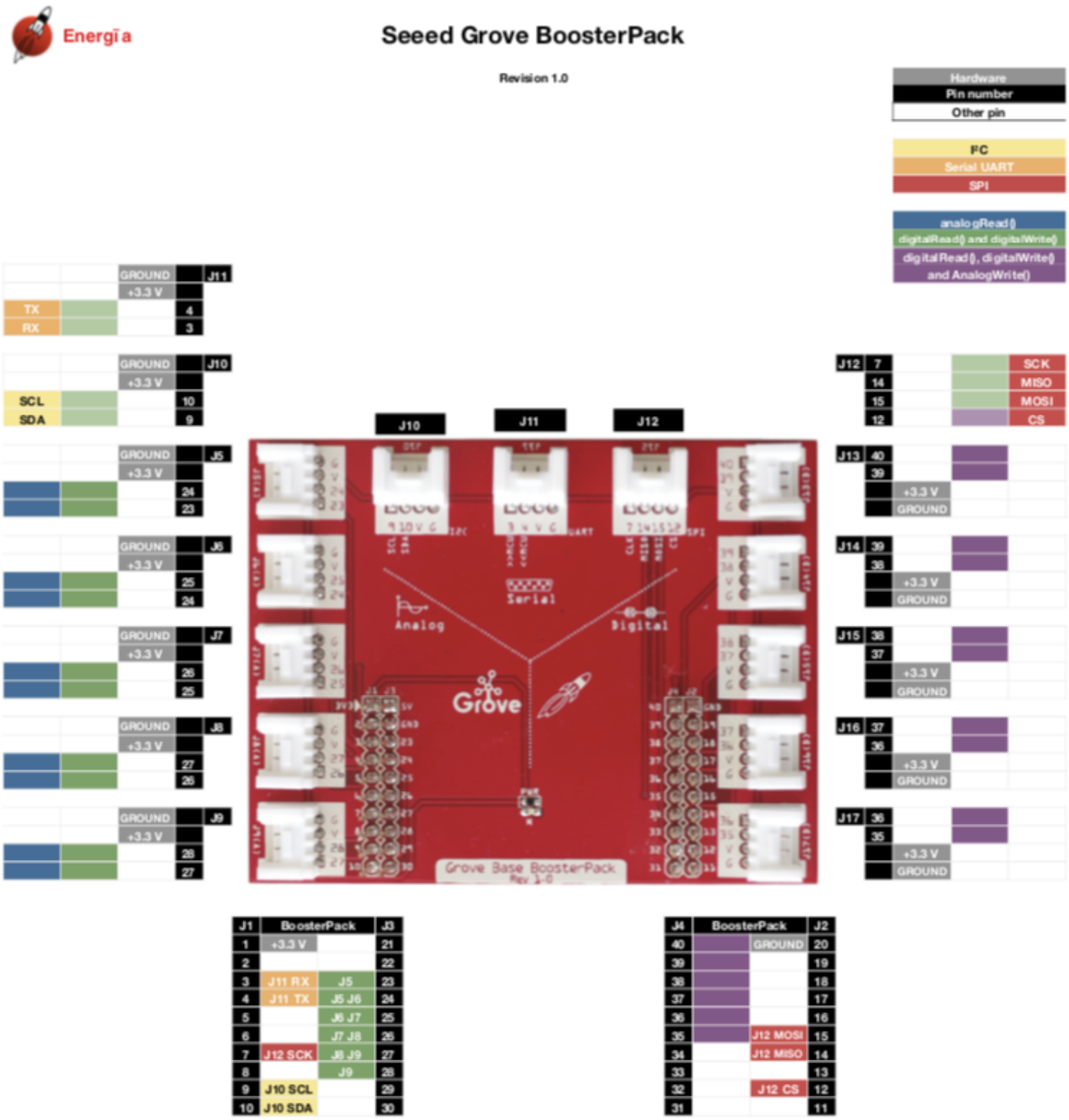
Step 5.1: Push button input

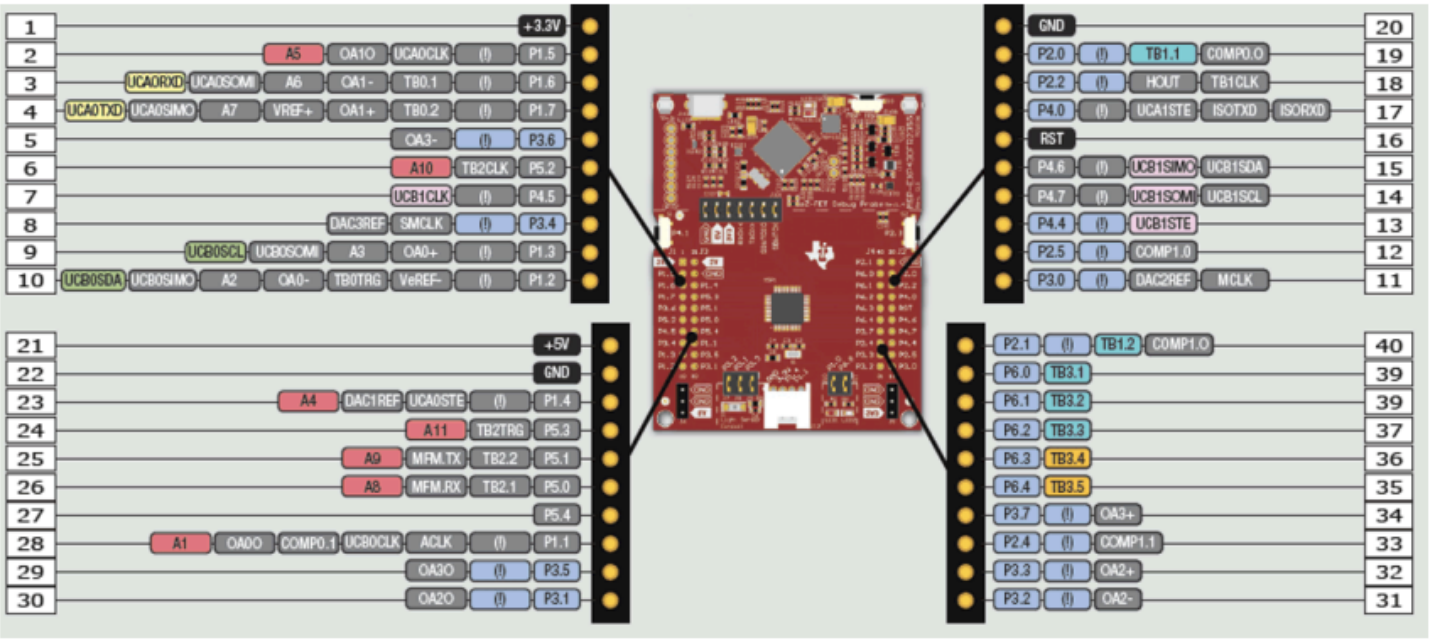
We'll use S1 and S2 on the development board as the trigger buttons to fire bullets at the enemies. Write a GPIO init function and another interface function to access if the button has been pressed or not. You may either poll or use interrupts to acquire your button inputs.

Step 5.2: Reading control stick inputs

The potentiometer is an analog device. It changes resistance as the knob is twisted, causing the voltage to be divided. When twisted all the way to the left, it will create an open (meaning we'll read 0 volts) and twisting all the way to the right will create a short (which means we'll read 3.3 volts). Positions in between will vary the voltage (somewhere between 0 and 3.3). By reading the voltages as the resistors are varied, we can determine how far we've twisted the knob.

Before doing anything, you need to plug the potentiometer into one of the grove connections on your board. Ensure that the connection contains an ADC peripheral. Use the diagrams below to determine which pins have an ADC, and which channel it belongs to:


Energia
Seeed Grove BoosterPack
Revision 1.0
Hardware
Pin number
Other pin
I²C
Serial UART
SPI
analogRead()
digitalRead() and digitalWrite()
digitalRead(), digitalWrite() and AnalogWrite()
J11
GROUND
+3.3 V
TX
RX
4
3
J10
SCL
SDA
10
9
J5
24
23
J6
25
24
J7
26
25
J8
27
26
J9
28
27
J10
J11
J12
Analog
Digital
Serial
Grove
Grove Base BoosterPack
J12
7
14
15
12
SCK
MISO
MOSI
CS
J13
40
39
J14
38
J15
37
J16
36
J17
35
J1
BoosterPack
J3
+3.3 V
J11 RX
J11 TX
J12 SCK
J10 SCL
J10 SDA
J5
J5 J6
J6 J7
J7 J8
J8 J9
J9
J4
BoosterPack
J2
GROUND
J12 MOSI
J12 MISO
J12 CS
Rei Vilo, 2012-2017
embeddedcomputing.weebly.com
version 2.0 2015-09-10



Once you've selected the connections, create two functions: one to initialize the ADC and another to retrieve a reading. The ADC should be initialized with the following specifications:

- 3.3 V positive reference voltage
- 0 V negative reference voltage
- 8-bit resolution

All other specifications not listed here are up to you. You do not need to use interrupts for this function (although you can if you want to!)

Also, don't forget to configure the PxSEL0 and PxSEL1 registers to bypass the GPIO peripheral so the ADC can read the voltages on the associated channel pins!

Step 6: Reading inputs and writing commands

Now that our interfaces are configured, we are ready to read input and transmit commands to the game! The game is programmed to respond to 8-bit values, which we will emulate on the UART.

The potentiometer will control the position of the jet along the x axis of the window. The game will position the jet relative to the digital representation of the voltage read by the potentiometer. If we send the value read by the ADC across UART using our UART transmit function, we will see the position of the jet change! However, there is one exception: the number 255.

255 is the value encoded to the "shoot" command. We need to make sure if we read 255 from the ADC, we do not transmit this value across the UART. This will be reserved for when the button is pressed. When the button is pressed, call your UART transmit function and pass it 255 to fire a bullet.

Once you are finished, you are almost ready to use the controller with the game.

Step 7: Setting the game COM port

In order for the game to recognize your controller, you will need to modify one line of code. Open the game.py script and look for the following line on line 6:

```
SERIAL_PORT = 'COM3'
```

In Step 4 you had to determine which COM port was associated with the dev board's backchannel UART. Replace COM4 with the name of the COM port of the backchannel UART on your computer. (Remember to keep the quotes)

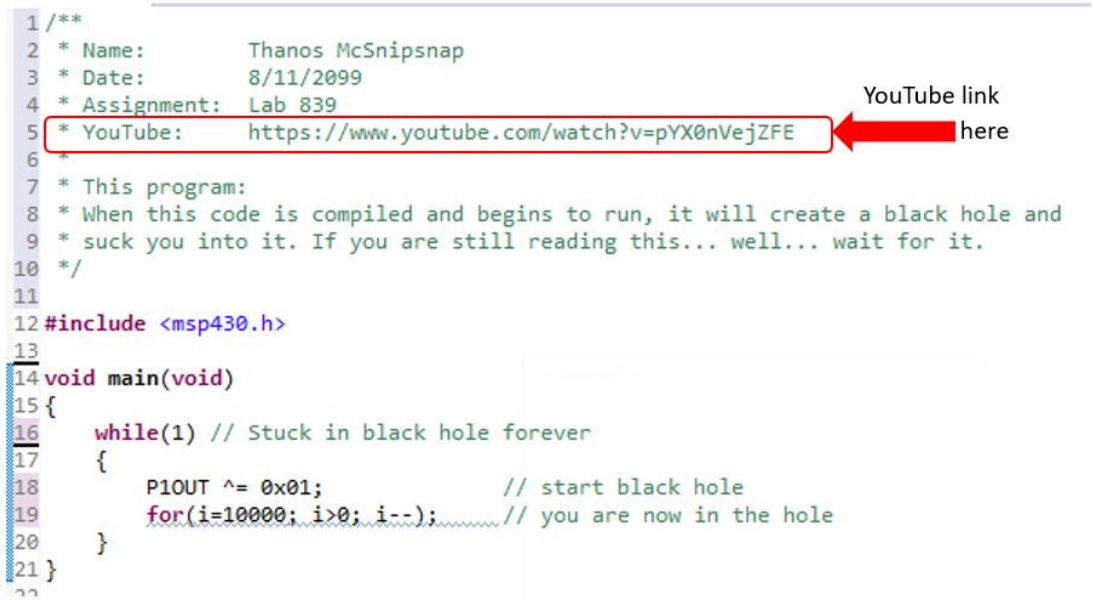
You should be able to press the button and turn the control stick to play the game now! For debugging purposes, the values being transmitted by your controller are printed to the python terminal in the powershell. You may notice a slight lag, but that is due to the driver/debugger and cannot be helped.

How to Submit:

Upload your main C file as a text file using the usual instructions.

Take a video of your code working on your board. In your video, be sure to clearly demonstrate and explain all requirements `(which are listed in the grading rubric below)` of your working code.

Upload your video to **YouTube only** (do not use google drive or other platforms) and then provide a link within your code submission as a comment like the picture shows below. Remember to include your name, program details, etc., within the program header at the top. It is common coding practice to provide a program header, so make sure you do this. If you used an example template, remember to delete the previous program header and replace it with your own.




```
/**
 * Name:         Thanos McSnipsnap
 * Date:         8/11/2099
 * Assignment:   Lab 839
 * YouTube:      https://www.youtube.com/watch?v=pYX0nVejZFE
 *
 * This program:
 * When this code is compiled and begins to run, it will create a black hole and
 * suck you into it. If you are still reading this... well... wait for it.
 */

#include <msp430.h>

void main(void)
{
    while(1) // Stuck in black hole forever
    {
        P1OUT ^= 0x01;              // start black hole
        for(i=10000; i>0; i--);     // you are now in the hole
    }
}
```

Copy your lab code from CCS into a `text file (.txt)` using notepad, Word, etc. Save your text file as `lastname_firstname_lab#.txt` with `#` being the current lab you are submitting.

Submit this text file (.txt) to Canvas.

If you have edited other files within your project workspace, then you will need to compress (.zip) your project workspace and upload it to canvas as well.

No other submission files will be accepted. You will only have two attempts to submit during the submission deadline. If a second attempt is needed, make sure all the necessary changes are made before submitting, as this would be your final attempt.

Game Controller

Criteria	Ratings		Pts
Code is neatly organized	**10 pts** **Full Marks**	**0 pts** **No Marks**	10 pts
UART Communication Functions	**30 pts** **Full Marks**	**0 pts** **No Marks**	30 pts
ADC Reading control the X axis of the plane	**30 pts** **Full Marks**	**0 pts** **No Marks**	30 pts
User inputs properly evaluated transmit the correct characters	**30 pts** **Full Marks**	**0 pts** **No Marks**	30 pts
			Total Points: 100