-CRT 360: Advanced Prototyping

GamePad - Game Setup

Part II: Get Moving

1. Setting up Controls

On your breadboard, hook up 4 push buttons that use the Teeny's internal pullup resistors (no external 10k resistors needed) on pins 0 - 3. Also, attach the X-axis of an analog stick to pin 14 and the Y-axis to pin 15.

Create a controls.h tab and include the Bouce2 library. Create a variable called buttonBounce and set it to 10 to represent a 10ms debounce time for reading the button. Create an array of 4 buttons to keep track of the debounce state of each of them like so:

```
#include <Bounce2.h>
int buttonBounce = 10;
Bounce * buttons = new Bounce[4];
```

Also create an array called buttonPins[4] and initialize it with the pin assignments of each of your buttons. Create another array called buttonBuffer[4] and initialize it with zeros. The buttonBuffer will serve as a place to save the current state of each button. Similarly, create an array called joystickPins[2] and initialize it with the joystick pin assignments and create a joystickBuffer[2] as well.

Create a function called initControls( ) and use a for loop to attach and set the interval for each button. Those functions will look like this:

```
buttons[i].attach(buttonPins[i], INPUT_PULLUP);
buttons[i].interval(buttonBounce);
```

Add this initControls( ) function to the main setup of your program.

void initControls( ) Copy/Paste your function in the space below

```
void initControls() {
 for (int i = 0; i < buttons.length; i++) {
   buttons[i].attach(buttonPins[i], INPUT_PULLUP);
   buttons[i].interval(buttonBounce);
 }
}
```

Create a function called getControls( ) that will get that current status of each button and update the buttonBuffer with its state (0 - Not Pressed, 1 - Pressed). Use a loop to avoid writing code for each individual button. You can check the status of each button using the .fell ( ) and .rose( ) functions. For example:

```
if(myButton.fell( ) ){ } // button is not pressed anymore
if(myButton.rose( ) ){ } // button was just pressed
```

AnalogRead the status of the analog sticks, but this time, find the centerpoint of each axis on the analog sticks and set the buffer to -1 if the axis dips below a certain threshold, 1 if the axis is above that threshold, and 0 if it is within that threshold of the center point. You will need to experiment to determine this threshold. Your stick should conform to the following logic:

Analog Stick - Left/Up Stick Array [-1, -1]	Analog Stick - Neutral/Up Stick Array [0, -1]	Analog Stick - Right/Up Stick Array [1, -1]
Analog Stick - Left/Neutral Stick Array [-1, 0]	Analog Stick - Neutral Stick Array [0, 0]	Analog Stick - Right/Neutral Stick Array [1, 0]
Analog Stick - Left/Down Stick Array [-1, 1]	Analog Stick - Neutral/Down Stick Array [0, 1]	Analog Stick -Right/Down Stick Array [1, 1]

Serial print the status of the analog sticks and push buttons to a single line and test all your code and hardware. Put these Serial print statements in a Metro timer that updates every 25ms to avoid overwhelming the Serial monitor. Call the getControls( ) function in the main loop and debug and adjust as necessary. Once you have dialed in the controls, comment out the Serial.print statements.

void getControls( ) Copy/Paste your function in the space below

```
void getControls() {
 for (int i = 0; i < 4; i++) {
   buttons[i].update();
   if (buttons[i].fell()) { buttonBuffer[i] =  1;} // button is not pressed anymore
   if (buttons[i].rose()) { buttonBuffer[i] = 0;} // button was just pressed

   Serial.print(buttonBuffer[i]);
   Serial.print("\t");
 }

 joystickX = analogRead(joystickPins[1]);
 joystickY = analogRead(joystickPins[0]);

 if(joystickX > 750){ joystickBuffer[0] = -1; }
 else if(joystickX < 450) { joystickBuffer[0] = 1; }
 else { joystickBuffer[0] = 0; }

 if(joystickY > 750){ joystickBuffer[1] = 1; }
 else if(joystickY < 450) { joystickBuffer[1] = -1; }
 else { joystickBuffer[1] = 0; }

 Serial.print(joystickBuffer[0]);
 Serial.print("\t");
 Serial.print(joystickBuffer[1]);
 Serial.println();
}
```

2. Drawing a Hero

Create a new tab named hero.h and include it in the main program header. On the hero.h tab include the graphics for your hero bitmaps. Create two floating point variables (heroX and heroY) and initialize them to the middle of the screen. Also create a floating point variable heroSpeed and initialize it to 1.0 to keep track of how fast the hero should move. This can be adjusted as necessary.

Create a function called drawHero( ). This function should update the location of the hero based on input received from the buttonBuffer and joystickBuffer created on the controls.h tab. It should then use the following logic to draw the hero:

1. Update the location of the hero (heroX, heroY) from the button and joystick buffers. Remember to multiply the values in the buffers by the heroSpeed (adjust speed as necessary) and constrain the movement to the screen.
2. Draw the current level using the drawLevel function
3. Set a clipping rectangle using the .setClipRect( ) function. Be sure to include a margin around the edges to prevent artifacts
4. Draw the hero using .drawRGBBitmap( ) function. You should use the version of this function that takes both a _PIX and _MASK array for transparency. Do not worry about animating the hero yet -- just use the first sprite in the hero_MASK and hero_PIX arrays.
5. Update the screen using the tft.updateScreen( ) function.

Run and test the getControls( ) and drawHero( ) functions to make sure that you are able to move your hero sprite around the screen.

Next, create a global variable called heroDir to keep track of the direction the hero should be facing as well as a global variable called heroFrame to keep track of which frame in the hero running animation should be shown. Also create a new Metro timer called heroFrameTimer and set its value to 250 ms.

In the drawHero( ) function, update heroDir based on the direction of the input. Every time the timer goes off, advance to the next frame of the hero animation (and go back to the beginning as necessary). Use this to draw the hero running in the correct direction. Also check to see if the action/attack button has been used and show that frame of the animation

void drawHero( ) Copy/Paste your function in the space below

```
void movement() {
  if (heroY < 160 && heroY > 0) {
    if (heroX < 300 && heroX > 0) {
      heroX = heroX + (heroSpeed * joystickBuffer[0]);
      heroY = heroY + (heroSpeed * joystickBuffer[1]);
    }
  }

  if(joystickBuffer[0] == -1){ heroDir = -1; }
  else if(joystickBuffer[0] == 1){ heroDir = 1; }
  else { heroDir = 0; }
}

void drawHero() {
  movement();

  if(heroFrameTimer.check()){
    if(buttonBuffer[0] == 0 && buttonBuffer[1] == 0){
      if(heroDir == -1){ heroFrame = (heroFrame + 1) % 3; }
      else if(heroDir == 1){ heroFrame = 5 + ((heroFrame + 1) % 3); } //Issue here with not looping
animation through when '5' is first frame
      else { heroFrame = 0; }
    } else {
      if(buttonBuffer[0] == 1){ heroFrame = 3 + ((heroFrame + 1) % 2); } //Issue here with not looping
animation through when '3' is first frame
      else if(buttonBuffer[1] == 1){ heroFrame = 8 + ((heroFrame + 1) % 2); }
    }
  }

  tft.setClipRect(heroX - 2, heroY - 2, heroW + 4, heroH + 4);
  tft.drawRGBBitmap(heroX, heroY, mojiSprites_PIX[heroFrame], mojiSprites_MASK[heroFrame],
heroW, heroH);

  tft.updateScreen();

}
```