

GamePad - Game Setup

Part I: Getting Started

1. Graphic Resources

Open Teensyduino and create a new sketch and save it as [FirstName]_GamePad. Create an appropriate header comment with information about your game. For example:

```
// HackBerry Attack!  
// Zane Cochran || 01 Jan 2021  
// CRT 360 Advanced Prototyping || GamePad
```

Export all pixel sprites from Piskel by going to Export -> Others -> Download C File. Make sure your scale is set to 1.0x to ensure proper sizing.

Then use the Piskel2LCD application to convert each Piskel file into an Arduino-friendly .h files. Each C file from Piskel will result in two Arduino files - an XXX_MASK.h and an XXX_PIX.h file. The MASK file contains all the transparency information for the images whereas the PIX file contains all the color information. The Arduino will use both of these files to draw the graphics you created.

Move these files into the folder created by Arduino when you saved your new sketch.

2. Frame Rate Sampling

Begin a Serial connection (9600 baud rate) in the setup() function.

Create a new tab called *frame.h* and include the Metro library. Create a new timer using the following format (replace *timerName* with a descriptive name for your timer and *timerDuration* for the number of milliseconds to delay):

```
#include <Metro.h> // This adds the Metro library to your sketch  
Metro timerName = Metro(timerDuration); // A prototype for a Metro timer
```

Create a function called *checkFrame* that will print how many times the Arduino has gone around the loop every 1 second. You can use the *.check()* function in Metro like this:

```
if(timerName.check() == true){  
    // do something  
}
```

Or even simpler - since the *.check()* function returns a true or false value, it is not necessary to include the *== true* part of the if statement:

```
if(timerName.check()){  
    // do something  
}
```

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

```
void checkFrame(){
  frame++;
  if(timer.check()){
    Serial.println(frame);
    frame = 0;
  }
}
```

Include the frame.h file in the header of your program's main tab and call the checkFrame function in the loop(). Record an example of the types of frame rates you observe below:

Observed Frame Rate

13952724

3. Screen Setup:

Connect your Teensy to your LCD screen using the following configuration:

Teensy / ILI9341 LCD Screen Pinouts			
ILI9341 Pin	Teensy 4.0 Pin	ILI9341 Pin	Teensy 4.0 Pin
VCC	VIN	GND	GND
CS	10	RESET	VCC
D/C	9	SDI (MOSI)	11
SCK	13	LED	VIN (w/ 100Ω Resistor)
SDO (MISO)	12	Source: https://www.pjrc.com/store/display_ili9341_touch.html	

Download the optimized ILI9341 library here: [ILI9341 Library](#) and add it to your sketch by navigating to Sketch->Include Library->Add .zip Library and selecting the downloaded .zip file. Create a screen.h tab and include the SPI and ILI9341 libraries, screen pin definitions, and create the screen object in the header of the tab:

```
#include "SPI.h"
#include "ILI9341_t3n.h"

#define TFT_DC 9
#define TFT_CS 10
#define TFT_RST 8

ILI9341_t3n tft = ILI9341_t3n(TFT_CS, TFT_DC, TFT_RST);
```

Also define the screen width and height and initialize the screen buffer in the header as well:

```
#define screenW 320
#define screenH 240
DMAMEM uint16_t screenBuffer[screenW * screenH];           // Screen Buffer
```

Create the function void initScreen() to begin communicating with the screen. Connect to the LCD and rotate the screen using the .setRotation() command. Initialize and activate the framebuffer and fill the screen with black:

```
tft.begin();           // Connect to LCD Screen
tft.setRotation(1);    // Rotate Screen 90 Degrees

tft.setFramebuffer(screenBuffer); // Initialize Frame Buffer
tft.useFramebuffer(1); // Use Frame Buffer

tft.fillScreen(ILI9341_BLACK); // Clear Screen
```

Create another function called screenTest(). This function should change the entire screen from black to white over and over again every 2 seconds. Use the Metro library, a boolean, and the resources from the [Adafruit GFX Documentation](#) to help you.

Include the screen.h file on the program's main tab and call the initScreen() function in the main program's setup(). Call the screenTest() function in the loop() and test. After testing, remove the screenTest() from the loop.

```
void screenTest() Copy/Paste your function and any helper variables.
```

```
void screenTest(){
  if(screenTimer.check()){
    tft.fillScreen(ILI9341_BLACK);
  }
  tft.fillScreen(ILI9341_WHITE);
}
```

4. Drawing Level Tiles

Create a new tab called *tile.h* and include all the pixel sprite .h files you created in Step 1 into its header by navigating to Sketch->Add File... and selecting the *_MASK.h* and *_PIX.h* files in the Arduino sketch folder. Then include these files in the *tile.h* tab using this example:

```
#include "tiles_MASK.h"
#include "tiles_PIX.h"
```

Define some parameters for the level sprite tiles. For example:

```
#define tileW 16    // 16 Tiles Across
#define tileH 12    // 12 Tiles Down
#define tileSize 20 // Tile Width (in pixels)
#define numLevels 6 // Number of Levels
```

Create a 2-dimensional array called *levels* to store each level's tile layout. Each value in the array will correspond to a specific bitmap tile you created in Piskel. You can reference the tiles using their hex values you recorded in the previous assignment. If you have some parts of the level that will not have a tile, indicate this by using the hex value *0xFF*. Use your level maps and game resources from the concept development assignment to help you. For example:

```
int levels[numLevels][tileW * tileH]{
  // Level 0 - Front
  {
    0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x01,
    0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01,
    0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x1C, 0x1D, 0x00, 0x00, 0x1E, 0x1F, 0x00, 0x00, 0x00,
    0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00,
    0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0xFF, 0xFF, 0xFF, 0xFF, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0xFF, 0xFF, 0xFF, 0xFF, 0x0F, 0x04, 0x04, 0x04, 0x04, 0x05, 0x05, 0x04, 0x04, 0x04, 0x04, 0x0C,
  },
  // Level 1 - Design Studio
  {
    0xFF, 0xFF, 0xFF, 0xFF, 0x0F, 0x04, 0x04, 0x04, 0x04, 0x05, 0x05, 0x04, 0x04, 0x04, 0x04, 0x0C,
    0xFF, 0xFF, 0xFF, 0xFF, 0x0A, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x06,
  }
}
```

```

0xFF, 0xFF, 0xFF, 0xFF, 0x0A, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x06,
0xFF, 0xFF, 0xFF, 0xFF, 0x0A, 0x03, 0x03, 0x17, 0x14, 0x03, 0x03, 0x17, 0x14, 0x03, 0x03, 0x06,
0xFF, 0xFF, 0xFF, 0xFF, 0x0A, 0x03, 0x03, 0x16, 0x15, 0x03, 0x03, 0x16, 0x15, 0x03, 0x03, 0x06,
0xFF, 0xFF, 0xFF, 0xFF, 0x0A, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x07,
0xFF, 0xFF, 0xFF, 0xFF, 0x0A, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x07,
0xFF, 0xFF, 0xFF, 0xFF, 0x0A, 0x03, 0x03, 0x17, 0x14, 0x03, 0x03, 0x17, 0x14, 0x03, 0x03, 0x06,
0xFF, 0xFF, 0xFF, 0xFF, 0x0A, 0x03, 0x03, 0x16, 0x15, 0x03, 0x03, 0x16, 0x15, 0x03, 0x03, 0x06,
0xFF, 0xFF, 0xFF, 0xFF, 0x0A, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x06,
0xFF, 0xFF, 0xFF, 0xFF, 0x0A, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x21, 0x06,
0xFF, 0xFF, 0xFF, 0xFF, 0x0E, 0x08, 0x08, 0x08, 0x08, 0x09, 0x09, 0x08, 0x08, 0x08, 0x08, 0x0D,
},
// Etc..
};

```

Write a function called `void drawLevel(int thisLevel)` that takes an input of which level should be drawn and store each tile at the appropriate location in the screen buffer using the `tft.drawRGBBitmap()` function.

Test the `drawLevel()` function by calling it in the main loop sampling each of the levels you designed. You will need to call the `tft.updateScreen()` function to send the screen buffer to the LCD screen. Also, open the Serial monitor and make a note of the frame rate. Remove these functions from the main loop when you are done testing.

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

```

void drawLevel(int thisLevel){
  for(int y = 0; y < tileH; y++){
    for (int x = 0; x < tileW; x++){
      int index = x + (y*tileW);
      int whichTile = levels[thisLevel][index];

      int finalX = x * tileSize;
      int finalY = y * tileSize;
      tft.drawRGBBitmap(finalX,finalY,Gamepad_PIX[whichTile],tileSize,tileSize);
    }
  }
}

```

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

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**

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 so that the hero can't walk off the edges of the screen.

2. Create a global variable called `isLevelDrawn` and set its initial value to false. This will help you determine if the entire level needs to be drawn or not.
3. Draw the current level using the `drawLevel` function you created in Part I. Check the value of `isLevelDrawn`. If false, call `tft.updateScreen()` and set the value to true. You will eventually need to reset `isLevelDrawn` back to false when the hero goes to a different level.
4. In order to only redraw the part of the screen that is changing, you will create a clipping rectangle with a function called `tft.setClipRect(x, y, w, h)` where `x` and `y` are the coordinates of the rectangle, and `w` and `h` are the width and height. You should set the position and size of the clipping rectangle to be slightly oversized for the hero (by 1 or 2 pixels). This will help prevent stray artifacts from appearing on the screen as the hero moves.
5. 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.
6. 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. 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 from the joystick. Every time the `heroFrameTimer` timer goes off, increase `heroFrame` and use this to draw the next frame of the hero animation (and go back to the beginning as necessary). You will need to adjust which frame you start at and which frame you end at in order to draw the hero running in the correct direction. Also check to see if an action/attack button has been used and show that frame of the animation. Rerun and test your code to make sure your hero is now changing directions when running or animates when using an action button.

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

