SUPER GAMEDUINO
ENTERTAINMENT SYSTEM
POWERED BY ESP32S

Sumitted To Dr. Emad Elsayed.

Sumitted by

-Mazen Elnahla.

-Yara Maher.

-Shrook Ibrahim.

-Massa Moustafa.

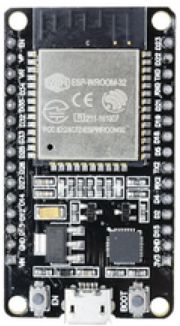
TABLE OF CONTENTS

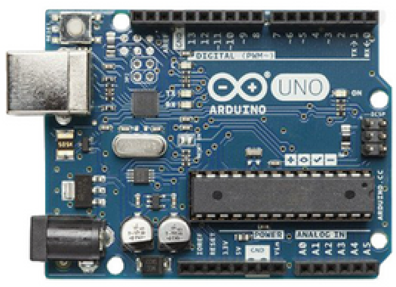
Introduction

In our project we collected old vantage acade games and bring it back with some modern technologies and less cost than old consoles in the market. Our project doesn't have fbga which is challenging part to display a signal to vga monitor but thanks to fabgl for their library we were able to generate signal and after so edits in the library we were able to get some acade games working very well.

ESP32

Arduino & ESP Systems





	ARDUINO		ESP32
Connectivity	I/O Pins	14	36
	PWM Pins	6	16
	Analog Pins	6	UP TO 18
	Analog Out Pins (DAC)	-	2
Computing	Processor	ATMega328P	Xtensa Dual Core 32-bit LX6 microprocessor
	Flash Memory	32 kB	161 MB
	SRAM	2 kB	520 kB
	EEPROM	1 kB	-
	Clock speed	16 MHz	Upto 240 MHz
	Voltage Level	5V	3.3V
	USB Connectivity	Standard A/B USB	Micro-USB
Communication	Hardware Serial Ports	1	3
	SPI Support	Yes (1x)	Yes (4x)
	CAN Support	no	yes
	I2C Support	Yes (1x)	**Yes (2x)**
Additional Features	**Wi-Fi**	-	802.11 b/g/n
	Bluetooth	-	v4.2 BR/EDR and BLE
	Touch Sensors		10
	CAM	-	-

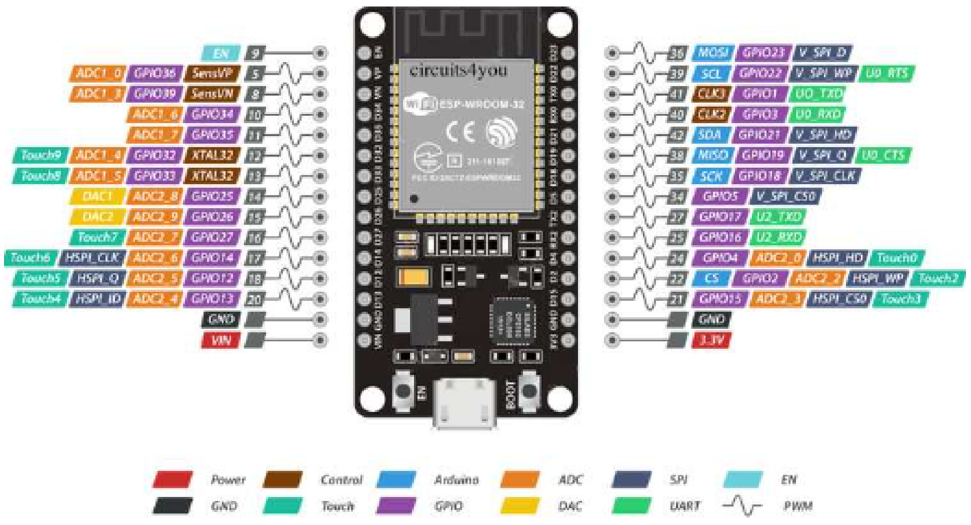
ESP32 Pinout Reference

ESP32 Peripherals and I/O

Although the ESP32 has total 48 GPIO pins, only 25 of them are broken out to the pin headers on both sides of the development board. These pins can be assigned to all sorts of peripheral duties, including:

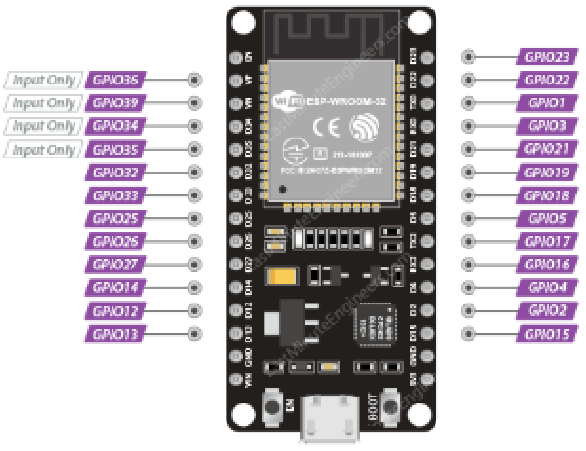
15 ADC channels	15 channels of 12-bit SAR ADC's. The ADC range can be set, in firmware, to either 0-1V, 0-1.4V, 0-2V, or 0-4V
2 UART interfaces	2 UART interfaces. One is used to load code serially. They feature flow control, and support IrDA too!
25 PWM outputs	25 channels of PWM pins for dimming LEDs or controlling motors.
2 DAC channels	8-bit DACs to produce true analog voltages.
3 SPI & 1 I2C interfa ces	There are 3 SPI and 1 I2C interfaces to hook up all sorts of sensors and peripherals.
9 Touch Pads	9 GPIOs feature capacitive touch sensing.

The ESP32 development board has a total of 30 pins that connect it to the outside world. For simplicity, pins with similar functionality are grouped together. The pinout is as follows:



GPIO Pins

- ESP32 development board has 25 GPIO pins which can be assigned to various functions programmatically. Each digital enabled GPIO can be configured to internal pull-up or pull-down or set to high impedance.



- Input Only GPIOs:

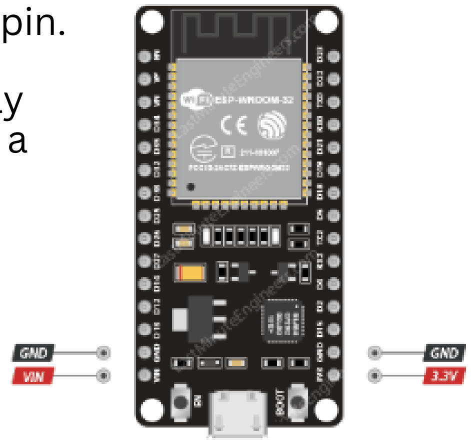
  Pins GPIO34, GPIO35, GPIO36(VP) and GPIO39(VN) cannot be configured as outputs, they can be used as either digital inputs, analog inputs, or for other unique purposes. Also note that they do not have internal pull-up or pull-down resistors, like the other GPIO pins.
  Also pins GPIO36(VP) and GPIO39(VN) are an integral part of the ultra-low-noise pre-amplifier for the ADC, which help to configure the sampling time and noise of the pre-amp.

- ESP32 Interrupt Pins:

  All GPIOs can be configured as interrupts.

Power Pins

- There are two power pins. VIN pin & 3.3V pin.
- The VIN pin can be used to directly supply the ESP32 and its peripherals if you have a regulated 5V voltage source.
- The 3.3V pin is the output of an on-board voltage regulator.
- This pin can be used to supply power to external components.



- GND is a ground pin of ESP32 development board.

FabGL Library

- FabGL is mainly a Graphics Library for ESP32. It implements several display drivers (for direct VGA output and for I2C and SPI LCD drivers).
- FabGL also implements: an Audio Engine, a Graphical User Interface (GUI), a Game Engine and an ANSI/VT Terminal.
- VGA output requires a digital to analog converter (DAC): it can be done by three 270 Ohm resistors to have 8 colors.
- There are several fixed and variable width fonts embedded.
- Unlimited number of sprites are supported. However big sprites and a large amount of them reduces the frame rate and could generate flickering.
- When there is enough memory (on low resolutions like 320x200), it is possible to allocate two screen buffers, so to implement double buffering

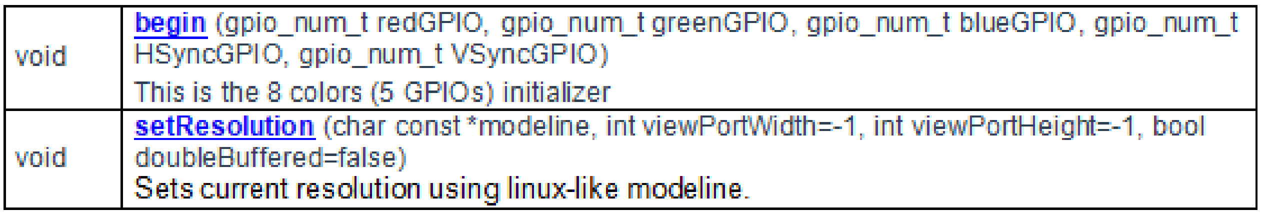
Main classes of FabGL library:

- **fabgl::VGA8Controller**, device driver for VGA 8 colors bitmapped output (low RAM requirements, CPU intensive)
- **fabgl::VGATextController**, device driver for VGA textual output (low RAM requirements, CPU intensive)
- **fabgl::Terminal**, that emulates an ANSI/VT100/VT102/etc and up terminal (look at vttest score)

VGAController Class Reference:

Represents the VGA bitmapped controller
#include <vgacontroller.h>

Public Member Functions



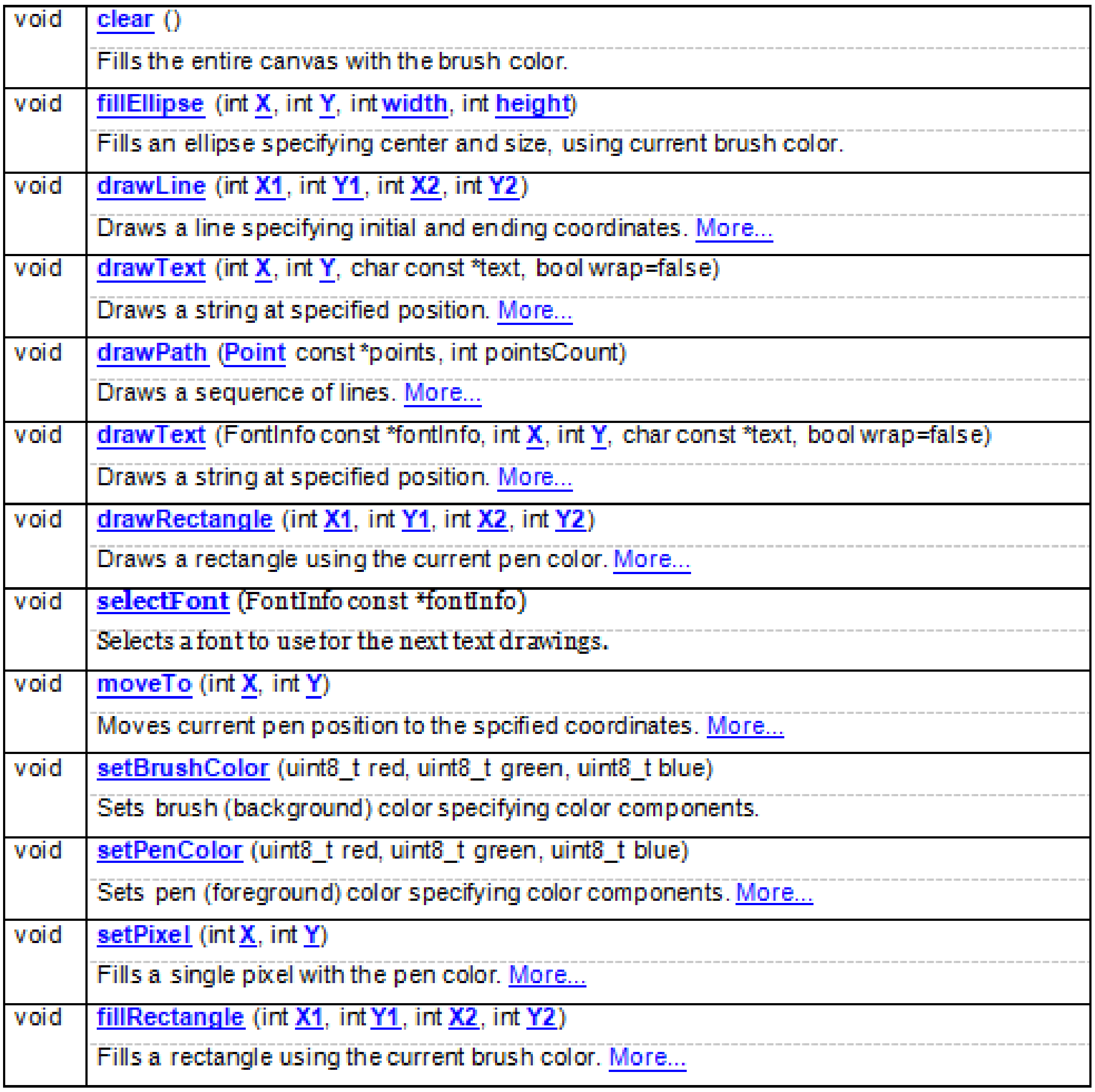
void	begin (gpio_num_t redGPIO, gpio_num_t greenGPIO, gpio_num_t blueGPIO, gpio_num_t HSyncGPIO, gpio_num_t VSyncGPIO) This is the 8 colors (5 GPIOs) initializer
void	setResolution (char const *modeline, int viewPortWidth=-1, int viewPortHeight=-1, bool doubleBuffered=false) Sets current resolution using linux-like modeline.

Canvas Class Reference

A class with a set of drawing methods. ..

#include <canvas.h>

Public Member Functions



void	clear () Fills the entire canvas with the brush color.
void	fillEllipse (int X, int Y, int width, int height) Fills an ellipse specifying center and size, using current brush color.
void	drawLine (int X1, int Y1, int X2, int Y2) Draws a line specifying initial and ending coordinates. More...
void	drawText (int X, int Y, char const *text, bool wrap=false) Draws a string at specified position. More...
void	drawPath (Point const *points, int pointsCount) Draws a sequence of lines. More...
void	drawText (FontInfo const *fontInfo, int X, int Y, char const *text, bool wrap=false) Draws a string at specified position. More...
void	drawRectangle (int X1, int Y1, int X2, int Y2) Draws a rectangle using the current pen color. More...
void	selectFont (FontInfo const *fontInfo) Selects a font to use for the next text drawings.
void	moveTo (int X, int Y) Moves current pen position to the spcified coordinates. More...
void	setBrushColor (uint8_t red, uint8_t green, uint8_t blue) Sets brush (background) color specifying color components.
void	setPenColor (uint8_t red, uint8_t green, uint8_t blue) Sets pen (foreground) color specifying color components. More...
void	setPixel (int X, int Y) Fills a single pixel with the pen color. More...
void	fillRectangle (int X1, int Y1, int X2, int Y2) Fills a rectangle using the current brush color. More...

Pixels

What is a pixel

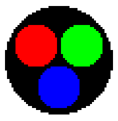
- The pixel -- a word invented from picture element -- is the basic unit of programmable color on a computer display or in a computer image.
- Pixels are the smallest unit in a digital display. Up to millions of pixels make up an image or video on a device's screen. Each pixel comprises a subpixel that emits a red, green and blue (RGB) color, which displays at different intensities. The RGB color components make up the gamut of different colors that appear on a display or computer monitor. the resolution of a display, numbers like 1920 x 1080 refer to the number of pixels.

How do pixels work

The number of pixels determines the resolution of a computer monitor or TV screen, and generally the more pixels, the clearer and sharper the image. The resolution of the newest 8K full ultra-high-definition TVs on the market is approximately 33 million pixels -- or 7680 x 4320. The number of pixels is calculated by multiplying the horizontal and vertical pixel measurements. For example, HD has 1,920 horizontal pixels and 1,080 vertical pixels, which totals 2,073,600. It's normally shown as 1920 x 1080 or just as 1080p. The p stands for progressive scan. A 4K video resolution, for example, has four times more pixels than full high definition (HD), and 8K has 16 times more pixels than 1080p.

Colors on A Computer Screen

Color from a computer monitor or a TV screen results from a different process than that due to reflection or transmission by a solid or solution. A monitor or TV screen generates three colors of light (red, green, and blue) and the different colors we see are due to different combinations and intensities of these three primary colors.

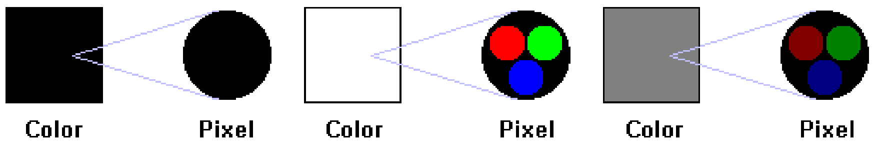


A sketch of a pixel showing the red, green and blue color produced by the three phosphors.

Nothing, All, or Some: Black, White, and Gray

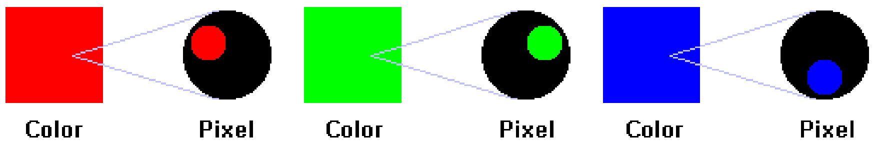
When no electrons strike the phosphors of a computer screen the phosphors emit no light and the screen appears black. On a white section of a screen all three phosphors are excited and produce light with about the same relative intensities as in sunlight so the light appears white. Gray parts of the screen have all three phosphors producing light, but at a much lower intensity. A sample color block and a sketch of a pixel from the block is shown below for each of these three colors.

The colors black, white, and gray with a sketch of a pixel from each.



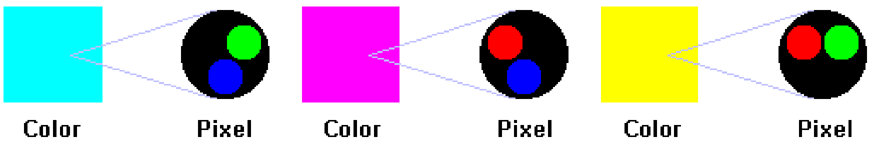
Pure Colors: Red, Green and Blue

Red, green, and blue colors are produced by exciting the respective phosphor.

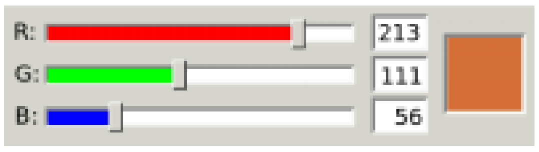


Combinations of Two Colors: Cyan, Purple, and Yellow

Cyan, purple, and yellow are mixtures of two of the primary colors with equal intensities of each.



RGB Values



- A color in the RGB color model is described by three component values (R,G,B).
- Computer monitors use the RGB color model. This model represents colors using three components of varying values (R,G,B) to indicate the amount of each of the primary colors that make up the color. The value of each component controls the intensity of each subpixel.
- The color information for each pixel is typically stored in a 24-bit format; eight bits of information is dedicated to each primary color, red, green and blue. Eight bits of information for each color channel allows 256 different possible intensities for each color (28 = 2 x 2 x 2 x 2 x 2 x 2 x 2 x 2 = 256). Combined, 24-bit color depth has a total of 16,777,216 color variations (2   24 or 2563), more colors than the human eye can actually discern.
- This color information is sent to the monitor, which adjusts its subpixels to the corresponding intensities, generating a precise array of colors and graphics on the screen.

Horizontal Timing

Horizontal Dots	640	640	640	
Vertical Scan Lines	350	400	480	
Horiz. Sync Polarity	POS	NEG	NEG	
A (us)	31.77	31.77	31.77	Scan-line time
B (us)	3.77	3.77	3.77	Sync pulse length
C (us)	1.89	1.89	1.89	Back porch
D (us)	25.17	25.17	25.17	Active video time
E (us)	0.94	0.94	0.94	Front porch

```
                ______________________                ________
________________|        VIDEO        |_______________| VIDEO (next line)
       |-C-|----------D-----------|-E-|
__   ______________________________   ___________
  |_|                              |_|
  |B|
  |---------------A----------------|
```

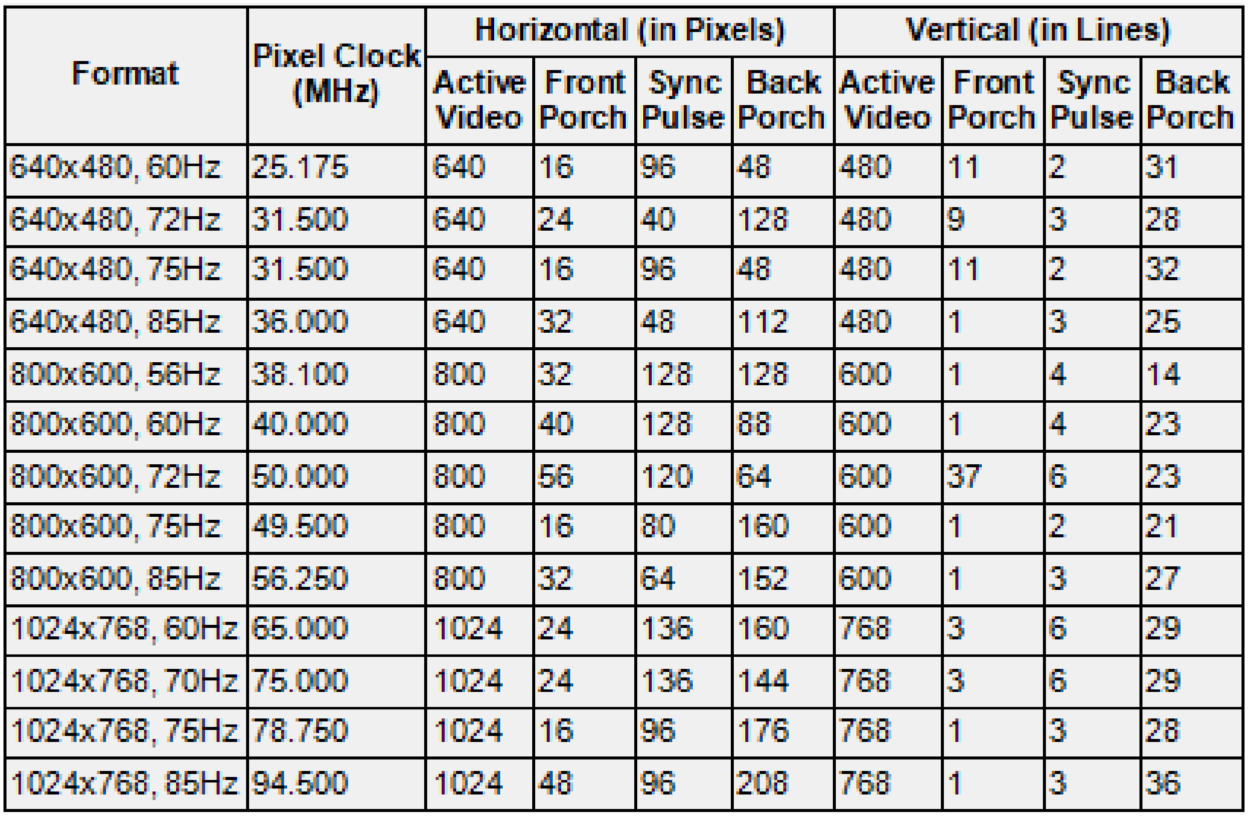
Vertical Timing

Horizontal Dots	640	640	640	
Vertical Scan Lines	350	400	480	
Vert. Sync Polarity	NEG	POS	NEG	
Vertical Frequency	70Hz	70Hz	60Hz	
O (ms)	14.27	14.27	16.68	Total frame time
P (ms)	0.06	0.06	0.06	Sync length
Q (ms)	1.88	1.08	1.02	Back porch
R (ms)	11.13	12.72	15.25	Active video time
S (ms)	1.2	0.41	0.35	Front porch

```
                ______________________                ________
________________|        VIDEO        |_______________|  VIDEO (next frame)
       |-Q-|----------R-----------|-S-|
__   ______________________________   ___________
  |_|                              |_|
  |P|
  |---------------O----------------|
```

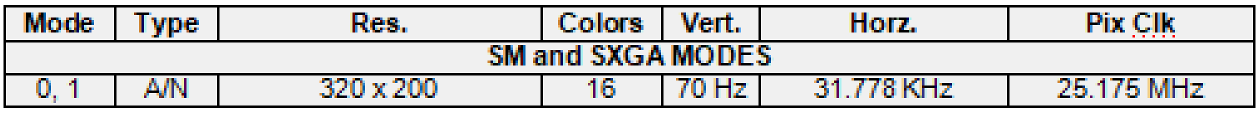
VGA timing html

The following table lists timing values for several popular resolutions.



Format	Pixel Clock (MHz)	Horizontal (in Pixels)				Vertical (in Lines)			
		Active Video	Front Porch	Sync Pulse	Back Porch	Active Video	Front Porch	Sync Pulse	Back Porch
640x480, 60Hz	25.175	640	16	96	48	480	11	2	31
640x480, 72Hz	31.500	640	24	40	128	480	9	3	28
640x480, 75Hz	31.500	640	16	96	48	480	11	2	32
640x480, 85Hz	36.000	640	32	48	112	480	1	3	25
800x600, 56Hz	38.100	800	32	128	128	600	1	4	14
800x600, 60Hz	40.000	800	40	128	88	600	1	4	23
800x600, 72Hz	50.000	800	56	120	64	600	37	6	23
800x600, 75Hz	49.500	800	16	80	160	600	1	2	21
800x600, 85Hz	56.250	800	32	64	152	600	1	3	27
1024x768, 60Hz	65.000	1024	24	136	160	768	3	6	29
1024x768, 70Hz	75.000	1024	24	136	144	768	3	6	29
1024x768, 75Hz	78.750	1024	16	96	176	768	1	3	28
1024x768, 85Hz	94.500	1024	48	96	208	768	1	3	36

Source: Rick Ballantyne, Xilinx Inc. TABLE 1 VGA CORE VIDEO MODE



Mode	Type	Res.	Colors	Vert.	Horz.	Pix Clk
SM and SXGA MODES						
0, 1	A/N	320 x 200	16	70 Hz	31.778 KHz	25.175 MHz

VGA Connector

The Video Graphics Array (VGA) connector is a three-row 15-pin DE-15 connector.

```
Video card side (female socket):

/-----------------------------------------------\
\          5         4         3         2         1          /
 \                                                           /
  \          10       9         8          7          6     /
   \                                                       /
    \   15        14        13        12        11        /
     \---------------------------------------------/


Monitor side (male plug):

/-----------------------------------------------\
\          1         2         3         4         5          /
 \                                                           /
  \    6        7         8         9         10              /
   \                                                       /
    \   11        12        13        14        15        /
     \---------------------------------------------/
```

Pins:-

1: Red Video (To monitor from video card)
2: Green Video (To monitor from video card)
3: Blue Video (To monitor from video card)
4: Monitor ID 2 (To video card from monitor)
5: TTL Ground (Monitor self-test, used for testing purposes only)
6: Red Analog Ground
7: Green Analog Ground
8: Blue Analog Ground
9: Key (Plugged hole, not used for electronic signals)
10: Sync Ground (For both sync pins)
11: Monitor ID 0 (To video card from monitor)
12: Monitor ID 1 (To video card from monitor)
13: Horizontal Sync (To monitor from video card)
14: Vertical Sync (To monitor from video card)
15: Monitor ID 3 (To video card from monitor)

- The most important pins on a monitor's connector are the synchronization, or sync pins.

- There are three: Horizontal, vertical, and ground. The sending of a horizontal sync pulse indicates the end of a horizontal line, and the sending of a vertical sync pulse indicates the end of a vertical screen frame.

- Each pulse on the horizontal pin creates one scanline across the screen.

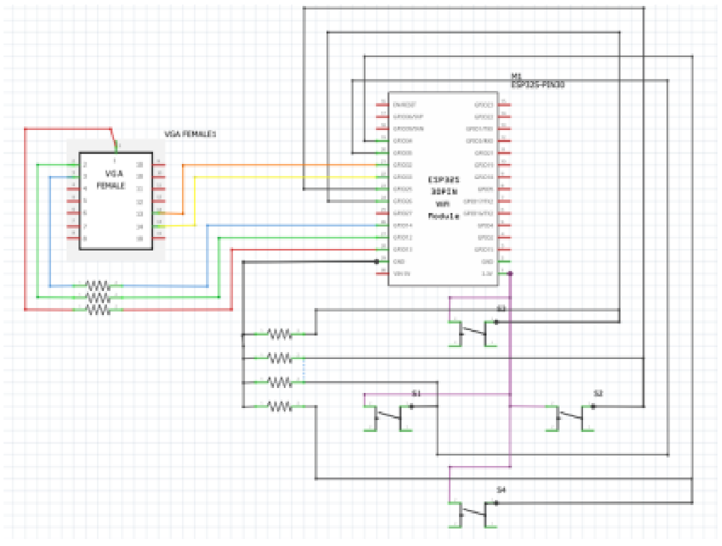
- The vertical pin pulses only when a whole screenful has been drawn; a pulse from the vertical pin makes the monitor return to the upper-left corner of the screen and begin drawing the next frame.

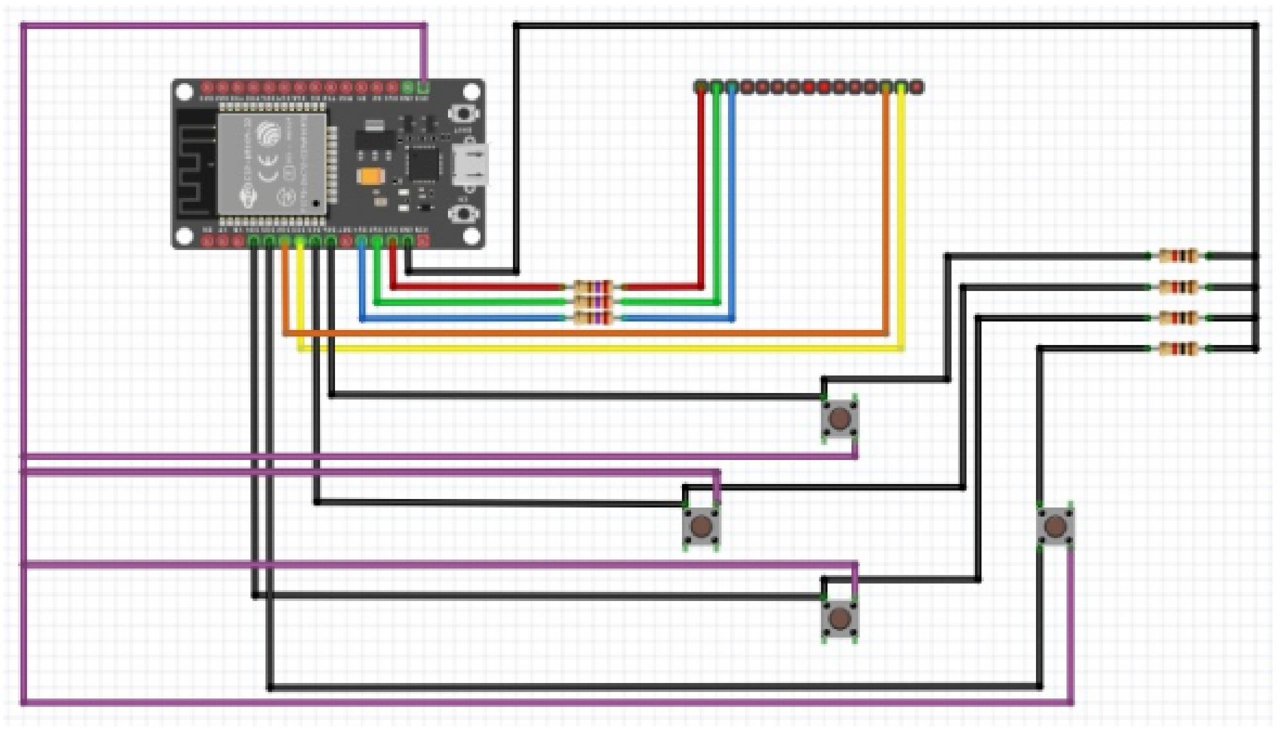
- The horizontal sync pin pulses several thousand times per second, while the vertical sync pin usually pulses less than 100 times per second (depending on the monitor's vertical refresh rate).

- For example, on a monitor operating at 640 x 480 resolution with a 70 Hz refresh rate, the horizontal sync pin would be pulsing approximately 33,600 times per second, or 33.6 KHz (480 scanlines, 70 times per second), while the vertical sync pin would be pulsing, of course, 70 times per second.

connection

- VGA red to ESP32 ping G13
- VGA green to ESP32 ping G12
- VGA blue to ESP32 ping G14
- VGA Hsync to ESP32 ping G32
- VGA Vsync to ESP32 ping G33
- VGA GND to ESP32 GND



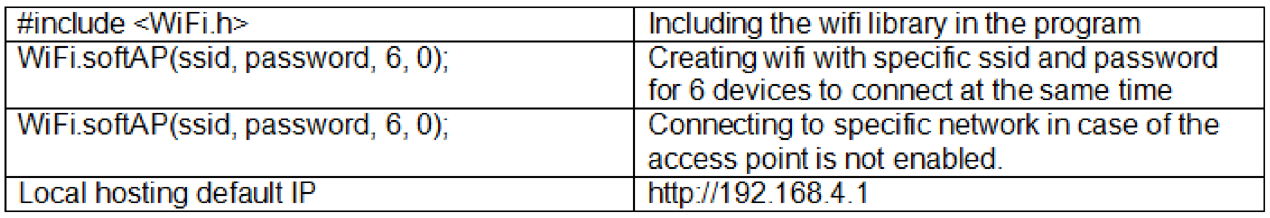


Terminal

WiFi

We use WiFi library which is a built in library on the esp32 which allows the esp to be an Access Point or a device with Ip and mac address to connect to regular network

Wifi used Functions:

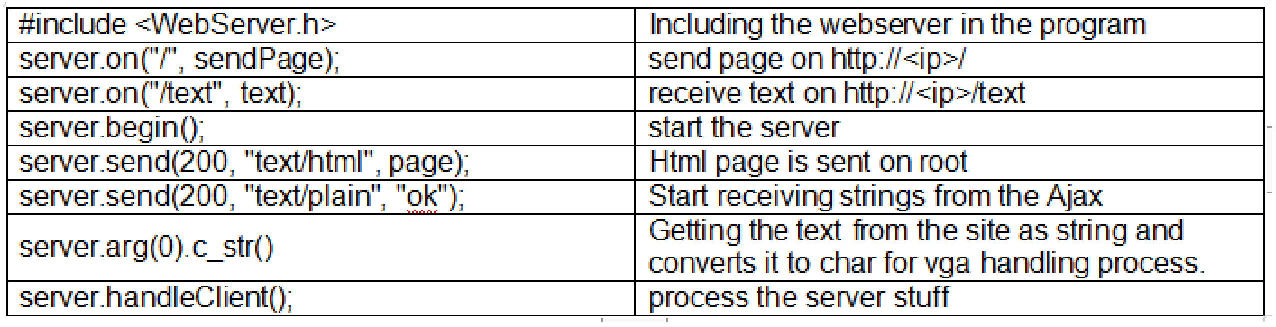


#include <WiFi.h>	Including the wifi library in the program
WiFi.softAP(ssid, password, 6, 0);	Creating wifi with specific ssid and password for 6 devices to connect at the same time
WiFi.softAP(ssid, password, 6, 0);	Connecting to specific network in case of the access point is not enabled.
Local hosting default IP	http://192.168.4.1

Webserver

A webserver is built in library on esp32 which allows the user to host a simple website using a single html file and converting the html to many strings so the esp32 could handle it while compiling.

Webserver used Functions:



#include <WebServer.h>	Including the webserver in the program
server.on("/", sendPage);	send page on http://<ip>/
server.on("/text", text);	receive text on http://<ip>/text
server.begin();	start the server
server.send(200, "text/html", page);	Html page is sent on root
server.send(200, "text/plain", "ok");	Start receiving strings from the Ajax
server.arg(0).c_str()	Getting the text from the site as string and converts it to char for vga handling process.
server.handleClient();	process the server stuff

bitluni Library

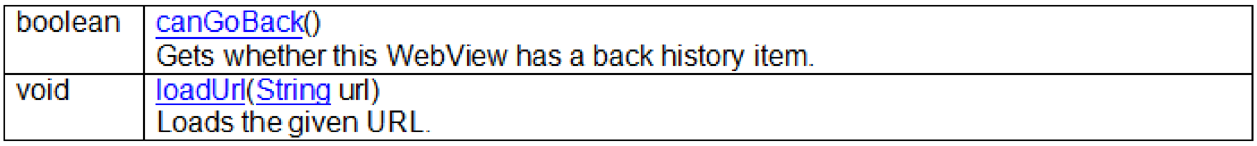
- ESP32Lib is a collection features for the ESP32 including highest performance VGA graphics (sprites, 3D), sound and game controllers packed in an Arduino library.
- ESP32Lib implements VGA output over I²S.
- The highest possible resolution with this library is 800x600. Many common resolutions like 320x240 are preconfigured und can be used without any effort. Two color depths are available. 14Bit R5G5B4 and 3Bit(8 color) R1G1B1 for convenience and memory savings.

Android App

Basic usage

WebView objects allow you to display web content as part of your activity layout but lack some of the features of fully developed browsers. A WebView is useful when you need increased control over the UI and advanced configuration options that will allow you to embed web pages in a specially designed environment for your app.

Public methods



boolean	canGoBack() Gets whether this WebView has a back history item.
void	loadUrl(String url) Loads the given URL.

webSettings.setJavaScriptEnabled(true); enable java Scrip code to run inside frame

Snake Code

```
#include "fabgl.h"
#include <canvas.h>

//-------------------- ESP32 pin definition for VGA port ----------------------------
//const int redPin = 13;
//const int greenPin = 12;
//const int bluePin = 14;
//const int hsyncPin = 32;
//const int vsyncPin = 33;
//----------------------------------------------------------------------------------

//-------------------- button pin definitions -----------------------
byte button_1 = 35; //right
byte button_2 = 34; //up (or rotate)
byte button_3 = 26; //left
byte button_4 = 25; //down fast
//-----------------------------------------------------------------

char str0[] PROGMEM="0";
char str1[] PROGMEM="1";
char str2[] PROGMEM="2";
char str3[] PROGMEM="3";
char str4[] PROGMEM="4";
char str5[] PROGMEM="5";
char str6[] PROGMEM="6";
char str7[] PROGMEM="7";
char str8[] PROGMEM="8";
char str9[] PROGMEM="9";
char str10[] PROGMEM="10";
char str20[] PROGMEM="ESP32 VGA Snake";
char str21[] PROGMEM="by SYMM";
char str22[] PROGMEM="Game Over";
char str23[] PROGMEM="Score";
char str24[] PROGMEM="Level";

boolean button1 = 0;
boolean button2 = 0;
boolean button3 = 0;
boolean button4 = 0;
boolean button;
byte counterMenu = 0;
byte counterMenu2 = 0;
byte state = 1;
byte score = 0;
byte level = 1;
byte scoreMax = 12;
int foodX;
int foodY;
int snakeMaxLength = 199;
int sx[200];    // > slength + scoreMax*delta + 1 = 40
int sy[200];
int slength = 9; // snake starting length
int slengthIni = 9; // snake starting length
int delta = 9;  // snake length increment
int i;
int x;
int y;
byte direct = 3;
int speedDelay = 32;
byte colA, colB, colC;
int x0Area = 100;
int y0Area = 20;
int x1Area = 300;
int y1Area = 180;
float cornerStep = 50.;

void setup() {
 // 8 colors
 VGAController.begin(GPIO_NUM_13, GPIO_NUM_12, GPIO_NUM_14, GPIO_NUM_32, GPIO_NUM_33);
 VGAController.setResolution(VGA_320x200_75Hz);
 randomSeed(analogRead(34));
 pinMode(button_1,INPUT);
 pinMode(button_2,INPUT);
 pinMode(button_3,INPUT);
 pinMode(button_4,INPUT);
 foodIni();
}

void foodIni() {
 do{
   foodX = random(x1Area - x0Area - 4) + x0Area + 2;
   foodY = random(y1Area - y0Area - 4) + y0Area + 2;
 } while ( myGetPixel(foodX, foodY) > 1 );
}

void processInputs() {
 button1 = digitalRead(button_1);
 button2 = digitalRead(button_2);
 button3 = digitalRead(button_3);
 button4 = digitalRead(button_4);
//  button1 = 0;
//  button2 = 0;
//  button3 = 0;
//  button4 = 0;
 button = button1 | button2 | button3 | button4;
}

void drawMenu() {
 counterMenu2++;
 delay(10);
 if (counterMenu2 > 50){
  counterMenu++;
  smoothRect(60, 60, 210, 60, 20, (counterMenu%5) + 1);
  vgaPrint(str20, 100, 70, (counterMenu%5) + 2);
  vgaPrint(str21, 100, 92, (counterMenu%5) + 3);
  counterMenu2 = 0;
 }
}
```

```
void drawBorder() {
    myColor(4);
    Canvas.drawRectangle(x0Area - 1, y0Area - 1, x1Area + 1, y1Area + 1);
}

void drawScore() {
  myColor(2);
  vgaPrint(str23, 35, 20, 2);
  myColor(5);
  vgaPrint(str24, 35, 60, 5);
  myColor(0);
  Canvas.setBrushColor(0, 0, 0);
  Canvas.fillRectangle(20, 40, 70, 52);
  Canvas.setBrushColor(0, 0, 0);
  Canvas.fillRectangle(20, 80, 70, 92);
  vgaPrintNumber(score%10, 55, 40, 4);
  vgaPrintNumber(level%10, 55, 80, 4);
  if (score > 9) {
    vgaPrintNumber(1, 45, 40, 4);
  }
  if (level > 9) {
    vgaPrintNumber(1, 45, 80, 4);
  }
}

// this is for the beginning game window ----------------------------------------------------------------------------------
void drawStartScreen() {
   Canvas.clear();
   drawBorder();
   drawSnakeIni();
   drawScore();
   button = 0;
   delay(200);
}

void drawSnakeIni() {
   for (byte i = 0; i < slength ; i++) {
     sx[i] = x0Area + 100 + i;
     sy[i] = y0Area + 70;
     putBigPixel(sx[i], sy[i], 2);
   }
   //direct = 1;
   for (byte i = slength; i < snakeMaxLength ; i++) {
    sx[i] = 1;
    sy[i] = 1;
  }
   putBigPixel(foodX, foodY, 1);
}

// re-inizialize new match ------------------------------------------------------------------------
void newMatch(){
  score = 0;
  slength = slengthIni;
  i = slength - 1;
  for (int i = slength; i < snakeMaxLength; i++){
    sx[i] = 0;
    sy[i] = 0;
  }
  Canvas.clear();
  drawBorder();
  drawScore();
  putBigPixel(foodX, foodY, 1);
}
```

```
//------------------------------------------------------------------------------------------------------------------------------------------------------
//---------------------------- This is the main loop of the game --------------------------------------------------------------------------
//------------------------------------------------------------------------------------------------------------------------------------------------------
void loop() {
 processInputs();
 if(state == 1) { //-------------------- start screen --------------------------------------------
   drawMenu();
   delay(10);
   processInputs();
   if (button == 1){
     button = 0;
     Canvas.clear();
     drawStartScreen();
     state = 2;
   }
 }

 if(state == 2){ //--------------------- snake waiting for start -----------------------------------------------
   if(score == scoreMax || score == 0){
     processInputs();
   }
   if (button == 1){
     score = 0;
     drawScore();
     button = 0;
     button1 = 0;
     button2 = 0;
     button3 = 0;
     button4 = 0;
     direct = 3;
     x = -1;
     y = 0;
     i = slength - 1;
     state = 3;
   }
 }

 if(state == 3) {
   processInputs();
   //-------------------- change direction --------------------------------------------
   if (direct == 1){
     if (button2 == 1){ x = 0; y = -1; direct = 2; button4 = 0;}
     if (button4 == 1){ x = 0; y = +1; direct = 4;}
   }
   else {
     if (direct == 2){
       if (button1 == 1){ x = +1; y = 0; direct = 1; button3 = 0;}
       if (button3 == 1){ x = -1; y = 0; direct = 3;}
     }
     else {
       if (direct == 3){
         if (button2 == 1){ x = 0; y = -1; direct = 2; button4 = 0;}
         if (button4 == 1){ x = 0; y = +1; direct = 4;}
       }
       else {
         if (direct == 4){
           if (button1 == 1){ x = +1; y = 0; direct = 1; button3 = 0;}
           if (button3 == 1){ x = -1; y = 0; direct = 3;}
         }
       }
     }
   }
```

```
//---------------------- delete tail -------------------------------------
    putBigPixel(sx[i], sy[i], 0);
    if (i>0) {
      putBigPixel(sx[i - 1], sy[i - 1], 2);
    }
    else {
      putBigPixel(sx[slength - 1], sy[slength - 1], 2);
    }
    if ( i == slength - 1){
      sx[i] = sx[0] + x;
      sy[i] = sy[0] + y;
    }
    else {
      sx[i] = sx[i + 1] + x;
      sy[i] = sy[i + 1] + y;
    }


//-------------------- out from border -----------------------------------
    if(sx[i] < x0Area + 1) {sx[i] = x1Area - 1;}
    if(sx[i] > x1Area - 1) {sx[i] = x0Area + 1;}
    if(sy[i] < y0Area + 1) {sy[i] = y1Area - 1;}
    if(sy[i] > y1Area - 1) {sy[i] = y0Area + 1;}

/*
//-------------------- out from border -----------------------------------
    if(sx[i] < x0Area + 1) {gameOver();}
    if(sx[i] > x1Area - 1) {gameOver();}
    if(sy[i] < y0Area + 1) {gameOver();}
    if(sy[i] > y1Area - 1) {gameOver();}
*/

//-------------------- check eating food -----------------------------------------------------------------------------------
----------------------------------
    if ( sx[i] > foodX - 3 && sx[i] < foodX + 3 && sy[i] > foodY - 3 && sy[i] < foodY + 3 ){
      putBigPixel(foodX, foodY, 0);
      //putBigPixel(sx[i], sy[i], 2);
      toneSafe(660,30);
      foodIni();
      drawBorder();
      putBigPixel(foodX, foodY, 1);
      if ( sx[i] == foodX || sy[i] == foodY ){
        slength = slength + 2*delta;
        score += 2;
      }
      else {
        slength = slength + delta;
        score++;
      }
      if (score > scoreMax) {
        speedDelay = int(speedDelay*0.8);
        level += 1;
        toneSafe(880,30);
        newMatch();
        drawSnakeIni();
        state = 2;
      }
      drawScore();
    }
    putBigPixel(foodX, foodY, 1);
```

```
//----------------------- increase head and Game Over -------------------------------------
    //if (myGetPixel(sx[i], sy[i]) == 2) {
    if (checkHit(sx[i], sy[i]) == 0) {
      putBigPixel(sx[i], sy[i], 2);
    }
    else //-------- Sneke hit himself ---------------------------------------------------
    {
      gameOver();
      //putBigPixel(40, 40 + cancellami, 6);
      //cancellami += 4;
    }
    putBigPixel(1, 1, 0);
    i--;
    if ( i < 0){i = slength - 1;}
    delay(speedDelay);
  }
}
//------------------------------------------------------------------------------------------------------------------------------------------
//----------------------------- end of the main loop of the game ---------------------------------------------------------------------------
//------------------------------------------------------------------------------------------------------------------------------------------

void toneSafe(int freq, int duration) {
  //vga.tone(freq);
  delay(duration);
  //vga.noTone();
}

void vgaPrint(char* str, int x, int y, byte color){
  myPrint(str, x, y, color);
}

void vgaPrintNumber(int number, int x, int y, byte color){
  char scoreChar[2];
  sprintf(scoreChar,"%d",number);
  myPrint(scoreChar, x, y, color);
}

void draw_line(int x0, int y0, int x1, int y1, byte color){
  myColor(color);
  Canvas.drawLine(x0, y0, x1, y1);
}

void putpixel(int x0, int y0,byte color){
  myColor(color);
  Canvas.setPixel(x0, y0);
}

void putBigPixel(int x0, int y0,byte color){
  myColor(color);
  Canvas.setBrushColor(colA, colB, colC);
  Canvas.fillRectangle(x0 - 1, y0 - 1, x0 + 1, y0 + 1);
  Canvas.setBrushColor(0, 0, 0);
}
```

```
void myColor(int color){
  if (color == 0){colA = 0; colB = 0; colC = 0;}//black
  if (color == 1){colA = 1; colB = 0; colC = 0;}//red
  if (color == 2){colA = 0; colB = 1; colC = 0;}//green
  if (color == 3){colA = 0; colB = 0; colC = 1;}//blue
  if (color == 4){colA = 1; colB = 1; colC = 0;}//yellow
  if (color == 5){colA = 1; colB = 0; colC = 1;}//purple
  if (color == 6){colA = 0; colB = 1; colC = 1;}//baby blue
  if (color == 7){colA = 1; colB = 1; colC = 1;}//white
  Canvas.setPenColor(colA, colB, colC);
}

void myPrint(char* str, byte x, byte y, byte color){
  Canvas.selectFont(Canvas.getPresetFontInfo(40, 14));
  myColor(color);
  Canvas.drawText(x, y, str);
}

void gameOver(){
  smoothRect(16, 118, 78, 20, 6, 6);
  vgaPrint(str22, 20, 121, 6);
  delay(300);
//  toneSafe(660, 200);
//  toneSafe(330, 200);
//  toneSafe(165, 200);
//  toneSafe(82, 200);
  button == 0;
  while(button == 0){processInputs();}
  speedDelay = 32;
  level = 1;
  newMatch();
  drawSnakeIni();
  state = 2;
}

void smoothRect(int x0, int y0, int w, int h, int r, int color){  //----- 1.6 comes from the rsolution ratio - 320/200 -------------
  myColor(color);
  draw_line(x0 + 1.6*r, y0 - 1, x0 + w - 1.6*r, y0 - 1, color);
  draw_line(x0 + 1.6*r, y0 + h, x0 + w - 1.6*r, y0 + h, color);
  draw_line(x0 - 1, y0 + r, x0 - 1, y0 + h - r, color);
  draw_line(x0 + w, y0 + r, x0 + w, y0 + h - r, color);
  for (int i = 0; i <= cornerStep; i++) {
    Canvas.setPixel(x0 + w - r*1.6*(1 - cos(i/cornerStep*3.1415/2.)), y0 + r*(1 - sin(i/cornerStep*3.1415/2.)));
    Canvas.setPixel(x0 + r*1.6*(1 - cos(i/cornerStep*3.1415/2.)), y0 + r*(1 - sin(i/cornerStep*3.1415/2.)));
    Canvas.setPixel(x0 + w - r*1.6*(1 - cos(i/cornerStep*3.1415/2.)), y0 + h - r*(1 - sin(i/cornerStep*3.1415/2.)));
    Canvas.setPixel(x0 + r*1.6*(1 - cos(i/cornerStep*3.1415/2.)), y0 + h - r*(1 - sin(i/cornerStep*3.1415/2.)));
  }
}
```

```
int checkHit(int x, int y){ //--------------------- check if snake hit himself ----------------------------------------------
   if (direct == 1){
      if (myGetPixel(x + 1, y) == 2 || myGetPixel(x + 1, y - 1) == 2 || myGetPixel(x + 1, y + 1) == 2) { return 1; };
   }
   if (direct == 2){
      if (myGetPixel(x + 1, y - 1) == 2 || myGetPixel(x , y - 1) == 2 || myGetPixel(x - 1, y - 1) == 2) { return 1; };
   }
   if (direct == 3){
      if (myGetPixel(x - 1, y) == 2 || myGetPixel(x - 1, y - 1) == 2 || myGetPixel(x - 1, y + 1) == 2) { return 1; };
   }
   if (direct == 4){
      if (myGetPixel(x + 1, y + 1) == 2 || myGetPixel(x , y + 1) == 2 || myGetPixel(x - 1, y + 1) == 2) { return 1; };
   }
   return 0;
}

int myGetPixel(int x, int y){
  int red = Canvas.getPixel(x, y).R;
  int green = Canvas.getPixel(x, y).G;
  int blue = Canvas.getPixel(x, y).B;
  if(red == 0 && green == 0 && blue == 0) {return 0;}
  if(red == 1 && green == 0 && blue == 0) {return 1;}
  if(red == 0 && green == 1 && blue == 0) {return 2;}
  if(red == 0 && green == 0 && blue == 1) {return 3;}
  if(red == 1 && green == 1 && blue == 0) {return 4;}
  if(red == 1 && green == 0 && blue == 1) {return 5;}
  if(red == 0 && green == 1 && blue == 1) {return 6;}
  if(red == 1 && green == 1 && blue == 1) {return 7;}
}
```

Terminal Code

```
#include <stdio.h>
#include <WiFi.h>
#include <WebServer.h>
//ESP32Lib headers
#include <ESP32Lib.h>
#include <Ressources/Font6x8.h>

//true: creates an access point, false: connects to an existing wifi
const bool AccessPointMode = true;
//wifi credentials (enter yours if you arne not using the AccessPointMode)
const char *ssid = "SuperGameduino";
const char *password = "";

//pin configuration, change if you need to
const int redPin = 13;
const int greenPin = 12;
const int bluePin = 14;
const int hsyncPin = 32;
const int vsyncPin = 33;

//the webserver at port 80
WebServer server(80);

//The VGA Device
VGA3Bit vga;

//include html page
const char *page =
#include "page.h"
;

///Html page is sent on root
void sendPage()
{
 server.send(200, "text/html", page);
}

///Received text will be displayed on the screen
void text()
{
 server.send(200, "text/plain", "ok");
 vga.println(server.arg(0).c_str());
}

///initialization
void setup()
{
 Serial.begin(115200);
 //Handle the WiFi AP or STA mode and display results on the screen
 if (AccessPointMode)
 {
  Serial.println("Creating access point...");
  WiFi.softAP(ssid, password, 6, 0);
 }
 else
 {
  Serial.print("Connecting to SSID ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED)
  {
   delay(500);
   vga.print(".");
  }
 }
 //start vga on the specified pins
 vga.init(vga.MODE400x300, redPin, greenPin, bluePin, hsyncPin, vsyncPin);
 //make the background black
 vga.clear(vga.RGBA(0, 0, 0));
 vga.backColor = vga.RGB(0, 0, 0);
 //select the font
 vga.setFont(Font6x8);

 //send page on http://<ip>/
 server.on("/", sendPage);
 //receive text on http://<ip>/text
 server.on("/text", text);
 //start the server
 server.begin();

 //display some text header on the screen including the ip
 vga.clear(vga.RGBA(0, 0, 0));
 vga.setCursor(0, 0);
 vga.println("----------------------");
 vga.println("SUPER GAMEDUINO Terminal");
 if (AccessPointMode)
 {
  vga.print("SSID: ");
  vga.println(ssid);
  if (strlen(password))
  {
   vga.print("password: ");
   vga.println(password);
  }
  vga.println("http://192.168.4.1");
 }
 else
 {
  vga.print("http://");
  vga.println(WiFi.localIP().toString().c_str());
 }
 vga.println("----------------------");
}

void loop()
{
 //process the server stuff
 server.handleClient();
 delay(10);
}
```