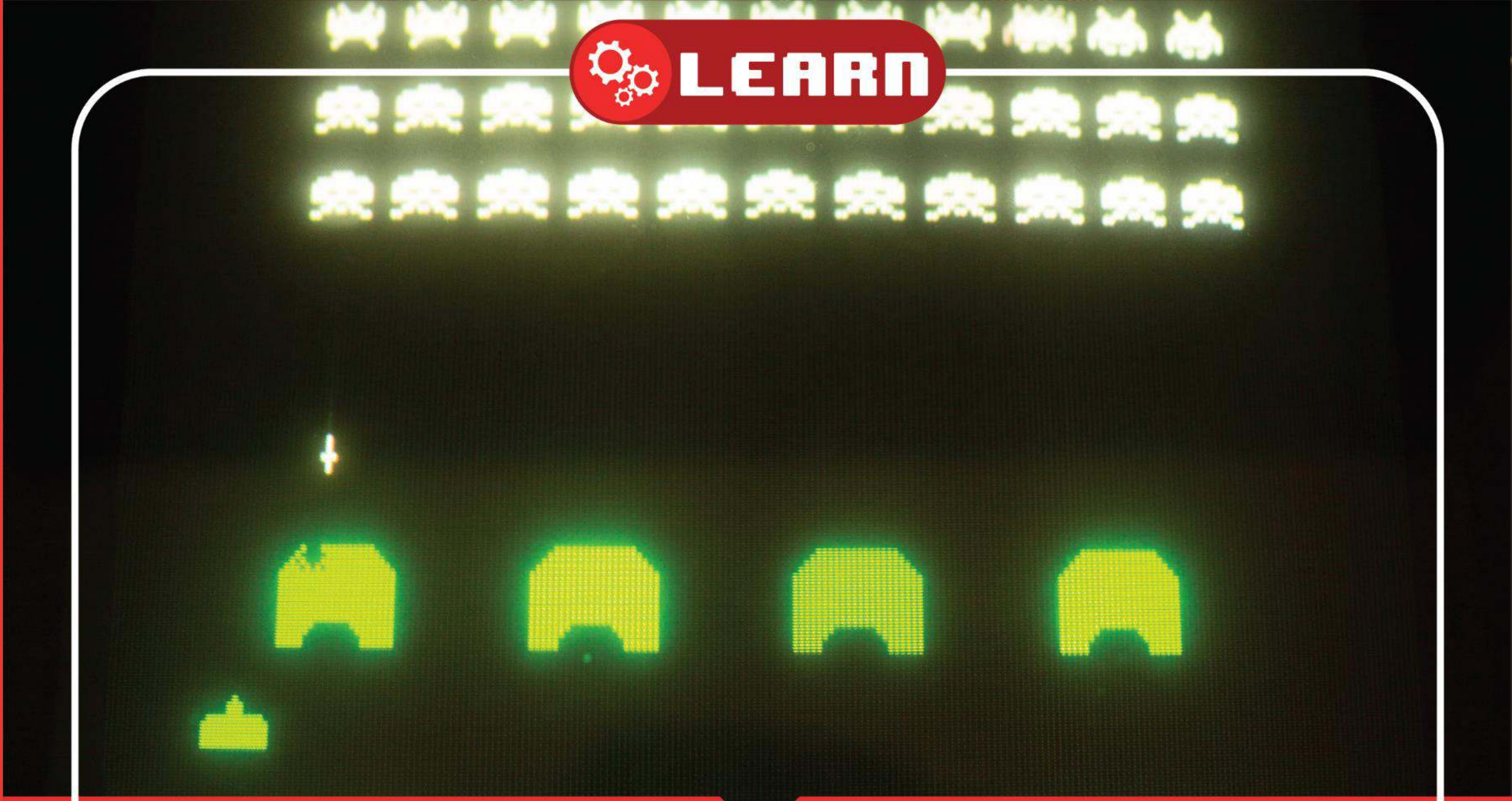
LEARN



CODING SPACE INVADERS

PART 4

With 8BitCADE

Compatible with:

8BitCADE XL

AND

8BitCADE

Contents

Tutorial by 8BitCADE adapted from the amazing work by Xtronical

The original guide can be followed online at

https://www.xtronical.com/projects/space-invaders/

Lesson Number:4

Lesson Title: **Player & Tank Movement**

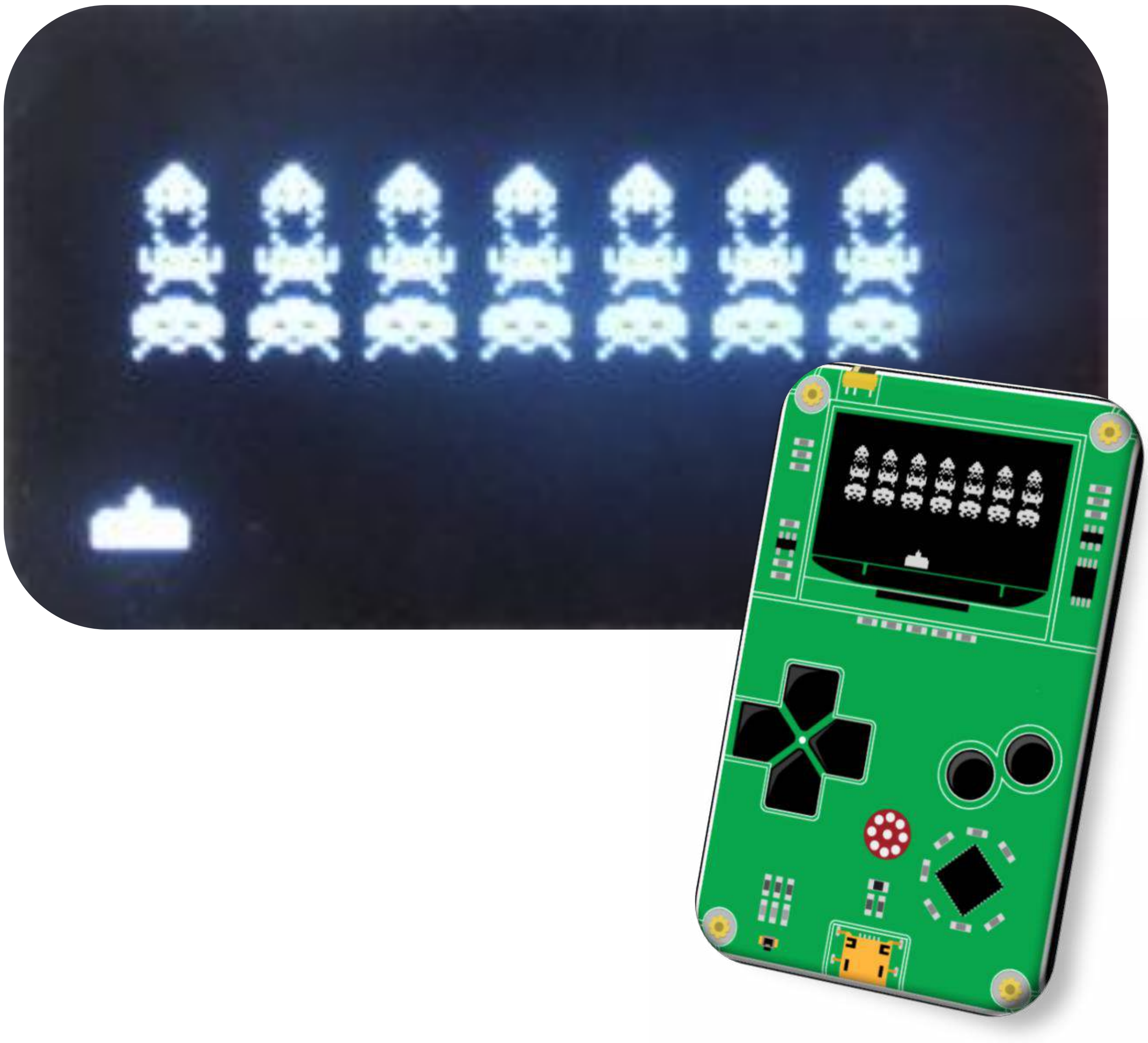
Code: Full Code for Lesson

System: Arduboy + Project ABE

Prerequisites to completing this tutorial

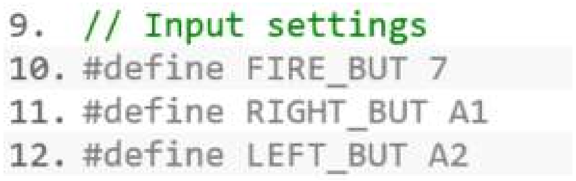
1. Tutorials 1,1a,1b,2,3
2. Know how to write code in either **Arduino IDE** or **Project ABE online Emulator**
3. Know how to draw pixel art by watching **Pixel Art Tutorial Space Invader**
4. Know how to convert pixel art into hexadecimal code by watching **Converting Pixel Art into Hexadecimal Code**
5. **Understand For loop, Switch Case, Arrays, Data Types**

In this tutorial we will add in the players tank and give them the ability to control it in the left or right direction. This will require some buttons putting onto our board. First the full source code (expand and copy as required). Once compiled and uploaded to your Arduino the display should look like this:



Step 1 ADDING INPUT BUTTON SETTINGS

Type the following code into line 9-11

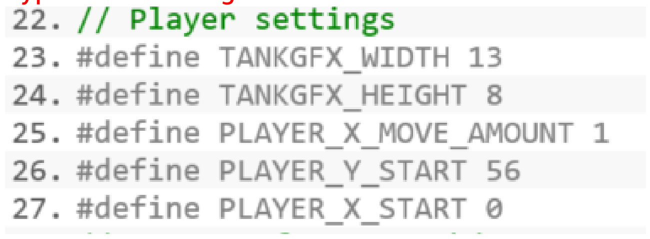


```
9.  // Input settings
10. #define FIRE_BUT 7
11. #define RIGHT_BUT A1
12. #define LEFT_BUT A2
```

Explanation (line 9-11)
We are setting these “constants” to the digital pins 4 ,5 and 6 of the Arduino. These will be the pins that our buttons are connected to. The next lines are related to the player:

Step 2 ADDING PLAYER SETTINGS & CONSTANT FOR TANK GRAPHICS

Type the following code into line 22-27

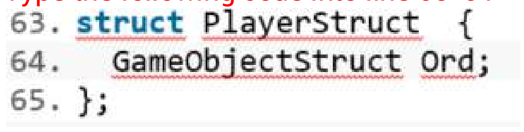


```
22. // Player settings
23. #define TANKGFX_WIDTH 13
24. #define TANKGFX_HEIGHT 8
25. #define PLAYER_X_MOVE_AMOUNT 1
26. #define PLAYER_Y_START 56
27. #define PLAYER_X_START 0
```

Explanation (line 22-27)
Lines 23 to 24 define the size of the players “Tank” graphics. The PLAYER_X_MOVE_AMOUNT is the number of pixels the players tank moves at a time. The X and Y for the player is the players start position.

Step 3 ADDING PLAYER STRUCT

Type the following code into line 63-64

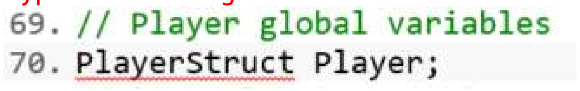


```
63. struct PlayerStruct  {
64.   GameObjectStruct Ord;
65. };
```

Explanation (line 63-65)
We have added a structure for the player, just like the Invader (AlienStruct) in the previous episode this at present just holds the position of the players tank.

Step 4 PLAYER GLOBAL VARIABLES

Type the following code into line 69-70



```
69. // Player global variables
70. PlayerStruct Player;
```

Explanation (line 69-70)
Line 70 initialises the player variable as a global variable (accessible to all parts of the code)

Step 5 UPDATING VOID SET_UP

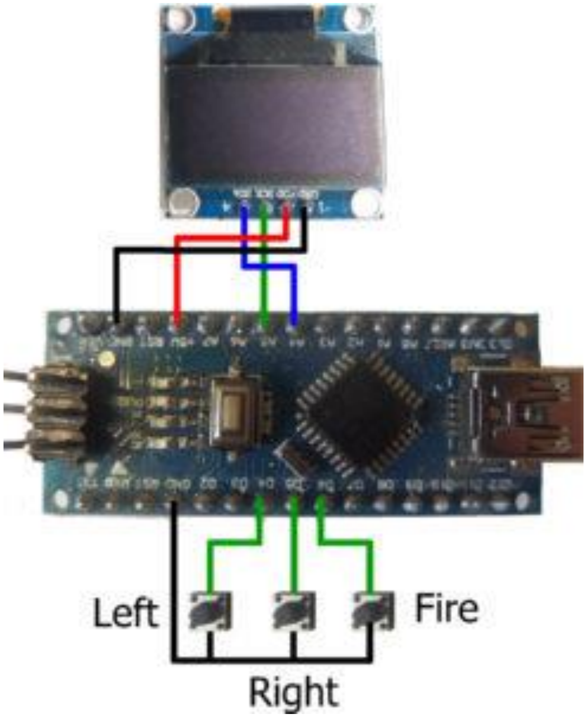
Type the following code into line 81-83

```
76. void setup() { // put your setup code here, to run once:
77.   arduboy.begin();
78.   arduboy.setFrameRate(60);
79.   InitAliens(0); // See voidInitAliens. initialises the aliens and sets up their X/Y Co-ordinates
80.   InitPlayer();
81.   pinMode(RIGHT_BUT, INPUT_PULLUP);
82.   pinMode(LEFT_BUT, INPUT_PULLUP);
83.   pinMode(FIRE_BUT, INPUT_PULLUP);
84. }
```

Explanation (line 81-83)
The **setup** routine has been expanded quite a lot. The **InitPlayer** initialise the player (more on that shortly) and then we set some pin modes for the Arduino pins that we will connect our buttons to. We are using the Arduino built in constant **INPUT_PULLUP**. This means these pins will be used for inputs and that we will used the Arduinos internal 10K (10,000) Ohm resistors to connect them (pull them up) to +5V. Why do we do this?...

Floating Inputs
On most electronics ***inputs,*** whether they are microprocessors (like the Arduino) or other types of chips, they should always be connected to something if you intend to use them in some way. If they are left "floating" i.e. with no connection to anything then they can give false values. That is to say they could report a 1 (+v) or 0 (gnd) input when actually the pin has not been set to any value by the user. Now traditionally in these situations you simply add a 10K resistor (although any high value resistor will usually be OK) to the input pin and connect it to either the positive or negative voltage. If connected to the positive then the pin will always report a "1" unless you deliberately connect it to the negative supply rail (gnd for our purposes). If connected to the 0V then it will always report a "0" unless we deliberately connect it to the +v rail. In either case the input value is most definitely a 1 or a 0. If you do not do this then you could get spurious **1**'s or **0**'s on the input pin when it is not deliberately connected to a voltage rail. The designers of the Atmel processor used by the Arduino spoilt us by putting some 10K resistors inside the chip, nice ☺ So we don't need to add them ourselves, to use them we just need to use **INPUT_PULLUP** setting to enable this and the pin will automatically be connected to the +ve rail via a 10K resistor.
So this pin will always report a "1" when we read it unless we connect it to the 0V rail. This is important to note when we come to wire up our pins to the push switches and when we come to read the values from the pins and make sense of them.

Wiring the buttons
We have three buttons to wire up, **Left, Right** and **Fire** to Arduino pins D7,A1, A2 respectively. An example diagram and real life wiring below. When a button is pressed it connects that pin to ground (0V). When we look at that pins value it will read a 0 of pressed and a 1 if not pressed.



Step 6 UPDATING VOID PHYSICS WITH PLAYER CONTROL

Type the following code into line 97

```
95. void Physics()  {
96.   AlienControl();
97.   PlayerControl();
98. }
```

Explanation (line 97)
So where do we scan for these button presses, within the Physics routines mentioned in the last episode, here is the function again with an important addition:

Step 7 PLAYER CONTROL

Type the following code into line 172-178

```
172.        void PlayerControl()  {
173.          // user input checks
174.          if ((digitalRead(RIGHT_BUT) == 0) & (Player.Ord.X + TANKGFX_WIDTH < SCREEN_WIDTH))
175.            Player.Ord.X += PLAYER_X_MOVE_AMOUNT;
176.          if ((digitalRead(LEFT_BUT) == 0) & (Player.Ord.X > 0))
177.            Player.Ord.X -= PLAYER_X_MOVE_AMOUNT;
178.        }
```

Explanation (line 172-178)
We scan for two button presses at present (as fire is not yet implemented). If the **RIGHT_BUT** pin is 0 then we've pressed the right move button, BUT we only update the players X position if the end of the tank is still on screen
– **(Player.Ord.X+TANKGFX_WIDTH<SCREEN_WIDTH)**, If it is we simply increase the players X position by the amount of pixels the player moves per move (**PLAYER_X_MOVE_AMOUNT**). Similarly, the LEFT_BUT is scanned and if the pressed and the players X position is still on screen then the X position will be decremented by the player movement amount. This all ensures that the tank cannot go off the ends of the screen.

Step 8 DISPLYING THE PLAYER

Type the following code into line 208-211

```
208.          // player
209.         arduboy.drawBitmap(Player.Ord.X, Player.Ord.Y,  TankGfx, TANKGFX_WIDTH, TANKGFX_HEIGHT, WHITE);
210.         arduboy.display();
211.       }
```

Explanation (line 208-211)
Added to the **UpdateDisplay** function is just one line that plots the players tank.

Step 9 INITIALISING THE PLAYER

Type the following code into line 213-216

```
213.      void InitPlayer()  {
214.        Player.Ord.Y = PLAYER_Y_START;
215.        Player.Ord.X = PLAYER_X_START;
216.      }
```

Explanation (line 213-216)
At present it doesn't do much apart from set the players initial start position.

Final Code

```
/* Animating Invaders, www.pixilart.com for sprite generation, http://javl.github.io/image2cpp/ for image conversion,
*/
//libraries for graphics/display
#include <Arduboy2.h>
Arduboy2 arduboy;
// DISPLAY SETTINGS
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
// Input settings
#define FIRE_BUT 7
#define RIGHT_BUT A1
#define LEFT_BUT A2
//Sprite settings see graphic for explanation. We have 3 aliens
#define NUM_ALIEN_COLUMNS 7               //Columns of aliens going across
#define NUM_ALIEN_ROWS 3                  //Rows of aliens going down
#define SPACE_BETWEEN_ALIEN_COLUMNS 5     //Space in pixels between each alien in the columns
#define SPACE_BETWEEN_ROWS 9              //Space in pixels between each alien in the rows
#define LARGEST_ALIEN_WIDTH 11            //Size in pixels of the largest allien width
#define X_START_OFFSET 6                  //Position of the first alien from in X,from the 0 position
#define INVADERS_DROP_BY 4
#define INVADERS_SPEED 12
// Player settings
#define TANKGFX_WIDTH 13
#define TANKGFX_HEIGHT 8
#define PLAYER_X_MOVE_AMOUNT 1
#define PLAYER_Y_START 56
#define PLAYER_X_START 0
// Status of a game object constants
#define ACTIVE 0
// Invader Sprites
const unsigned char InvaderTopGfx [] PROGMEM = {
  0x98, 0x5c, 0xb6, 0x5f, 0x5f, 0xb6, 0x5c, 0x98
};
const unsigned char InvaderTopGfx2 [] PROGMEM = {
  0x58, 0xbc, 0x16, 0x1f, 0x1f, 0x16, 0xbc, 0x58
};
const unsigned char PROGMEM InvaderMiddleGfx [] =
{
  0x1e, 0xb8, 0x7d, 0x36, 0x3c, 0x3c, 0x3c, 0x36, 0x7d, 0xb8, 0x1e
};
const unsigned char PROGMEM InvaderMiddleGfx2 [] = {
  0x78, 0x18, 0x7d, 0xb6, 0xbc, 0x3c, 0xbc, 0xb6, 0x7d, 0x18, 0x78
};
const unsigned char PROGMEM InvaderBottomGfx [] = {
  0x1c, 0x5e, 0xfe, 0xb6, 0x37, 0x5f, 0x5f, 0x37, 0xb6, 0xfe, 0x5e, 0x1c
};
const unsigned char PROGMEM InvaderBottomGfx2 [] = {
  0x9c, 0xde, 0x7e, 0x36, 0x37, 0x5f, 0x5f, 0x37, 0x36, 0x7e, 0xde, 0x9c
};
// Player grafix
const unsigned char PROGMEM TankGfx [] = {
  0xf0, 0xf8, 0xf8, 0xf8, 0xf8, 0xfe, 0xff, 0xfe, 0xf8, 0xf8, 0xf8, 0xf8, 0xf0
};
// Game Structures
struct GameObjectStruct {
  signed int X;
  signed int Y;
  unsigned char Status;  //0 active, 1 exploding, 2 destroyed
};
struct AlienStruct {
  GameObjectStruct Ord;
};
struct PlayerStruct  {
  GameObjectStruct Ord;
};
char AlienXMoveAmount = 1; // norm is 2 , this is pixel movement in X
signed char InvadersMoveCounter;            // counts down, when 0 move invaders, set according to how many aliens on screen
bool AnimationFrame = false; // two frames of animation, if true show one if false show the other
// Player global variables
PlayerStruct Player;
//Alien Global Veriables
// create an 2D array of aliens across the screen
AlienStruct Alien[NUM_ALIEN_COLUMNS][NUM_ALIEN_ROWS]; // columns and rows relate to the define code above so 7 and 3.
byte AlienWidth[] = {8, 11, 12}; // top middle and bottom widths of the graphics

void setup() { // put your setup code here, to run once:
  arduboy.begin();
  arduboy.setFrameRate(60);
  InitAliens(0); // See voidInitAliens. initialises the aliens and sets up their X/Y Co-ordinates
  InitPlayer();
  pinMode(RIGHT_BUT, INPUT_PULLUP);
  pinMode(LEFT_BUT, INPUT_PULLUP);
  pinMode(FIRE_BUT, INPUT_PULLUP);
}

void loop()
{
  if (!arduboy.nextFrame()) {
    return;
  }
  Physics();
  UpdateDisplay();
}

void Physics()  {
  AlienControl();
  PlayerControl();
}

void AlienControl()
{
  if ((InvadersMoveCounter--) < 0)
  {
    bool Dropped = false;
    if ((RightMostPos() + AlienXMoveAmount >= SCREEN_WIDTH) | (LeftMostPos() + AlienXMoveAmount < 0)) // at edge of screen
    {
      AlienXMoveAmount = -AlienXMoveAmount;             // reverse direction
      Dropped = true;                                   // and indicate we are dropping
    }
    // update the alien postions
    for (int Across = 0; Across < NUM_ALIEN_COLUMNS; Across++)
    {
      for (int Down = 0; Down < 3; Down++)
      {
        if (Alien[Across][Down].Ord.Status == ACTIVE)
        {
```

```
                  if (Dropped == false)
                    Alien[Across][Down].Ord.X += AlienXMoveAmount;
                  else
                    Alien[Across][Down].Ord.Y += INVADERS_DROP_BY;
                }
              }
            }
            InvadersMoveCounter = INVADERS_SPEED;
            AnimationFrame = !AnimationFrame; ///swap to other frame
          }
        }
        int RightMostPos()  {
          //returns x pos of right most alien
          int Across = NUM_ALIEN_COLUMNS - 1;
          int Down;
          int Largest = 0;
          int RightPos;
          while (Across >= 0) {
            Down = 0;
            while (Down < NUM_ALIEN_ROWS) {
              if (Alien[Across][Down].Ord.Status == ACTIVE)
              {
                // different aliens have different widths, add to x pos to get right pos
                RightPos = Alien[Across][Down].Ord.X + AlienWidth[Down];
                if (RightPos > Largest)
                  Largest = RightPos;
              }
              Down++;
            }
            if (Largest > 0) // we have found largest for this coloum
              return Largest;
            Across--;
          }
          return 0;  // should never get this far
        }
        int LeftMostPos()  {
          //returns x pos of left most alien
          int Across = 0;
          int Down;
          int Smallest = SCREEN_WIDTH * 2;
          while (Across < NUM_ALIEN_COLUMNS) {
            Down = 0;
            while (Down < 3) {
              if (Alien[Across][Down].Ord.Status == ACTIVE)
                if (Alien[Across][Down].Ord.X < Smallest)
                  Smallest = Alien[Across][Down].Ord.X;
              Down++;
            }
            if (Smallest < SCREEN_WIDTH * 2) // we have found smalest for this coloum
              return Smallest;
            Across++;
          }
          return 0;  // should never get this far
        }

        void PlayerControl()  {
          // user input checks
          if ((digitalRead(RIGHT_BUT) == 0) & (Player.Ord.X + TANKGFX_WIDTH < SCREEN_WIDTH))
            Player.Ord.X += PLAYER_X_MOVE_AMOUNT;
          if ((digitalRead(LEFT_BUT) == 0) & (Player.Ord.X > 0))
            Player.Ord.X -= PLAYER_X_MOVE_AMOUNT;
        }

```

```
void UpdateDisplay()
{
  arduboy.clear();
  for (int across = 0; across < NUM_ALIEN_COLUMNS; across++)
  {
    for (int down = 0; down < NUM_ALIEN_ROWS; down++)
    {
      switch (down)  {
        case 0:
          if (AnimationFrame)
            arduboy.drawBitmap(Alien[across][down].Ord.X, Alien[across][down].Ord.Y,  InvaderTopGfx, AlienWidth[down], 8, WHITE);
          else
            arduboy.drawBitmap(Alien[across][down].Ord.X, Alien[across][down].Ord.Y,  InvaderTopGfx2, AlienWidth[down], 8, WHITE);
          break;
        case 1:
          if (AnimationFrame)
            arduboy.drawBitmap(Alien[across][down].Ord.X, Alien[across][down].Ord.Y,  InvaderMiddleGfx, AlienWidth[down], 8, WHITE);
          else
            arduboy.drawBitmap(Alien[across][down].Ord.X, Alien[across][down].Ord.Y,  InvaderMiddleGfx2, AlienWidth[down], 8, WHITE);
          break;
        default:
          if (AnimationFrame)
            arduboy.drawBitmap(Alien[across][down].Ord.X, Alien[across][down].Ord.Y,  InvaderBottomGfx, AlienWidth[down], 8, WHITE);
          else
            arduboy.drawBitmap(Alien[across][down].Ord.X, Alien[across][down].Ord.Y,  InvaderBottomGfx2, AlienWidth[down], 8, WHITE);
      }
    }
  }
   // player
  arduboy.drawBitmap(Player.Ord.X, Player.Ord.Y,  TankGfx, TANKGFX_WIDTH, TANKGFX_HEIGHT, WHITE);
  arduboy.display();
}

void InitPlayer()  {
  Player.Ord.Y = PLAYER_Y_START;
  Player.Ord.X = PLAYER_X_START;
}

void InitAliens(int YStart) {
  for (int across = 0; across < NUM_ALIEN_COLUMNS; across++) { // plots all 21 aliens using the array by ploting each alien across repeatedly
    for (int down = 0; down < 3; down++) {                      //plots each alien down repeatedly.....we add down to centralise the aliens
      Alien[across][down].Ord.X = X_START_OFFSET + (across * (LARGEST_ALIEN_WIDTH + SPACE_BETWEEN_ALIEN_COLUMNS)) - down; // this spaces each alien on x axis define OFFSETs-down helps centralise them
      Alien[across][down].Ord.Y = YStart + (down * SPACE_BETWEEN_ROWS); // calculation to space aliens on Y axis
    }
  }
}
```