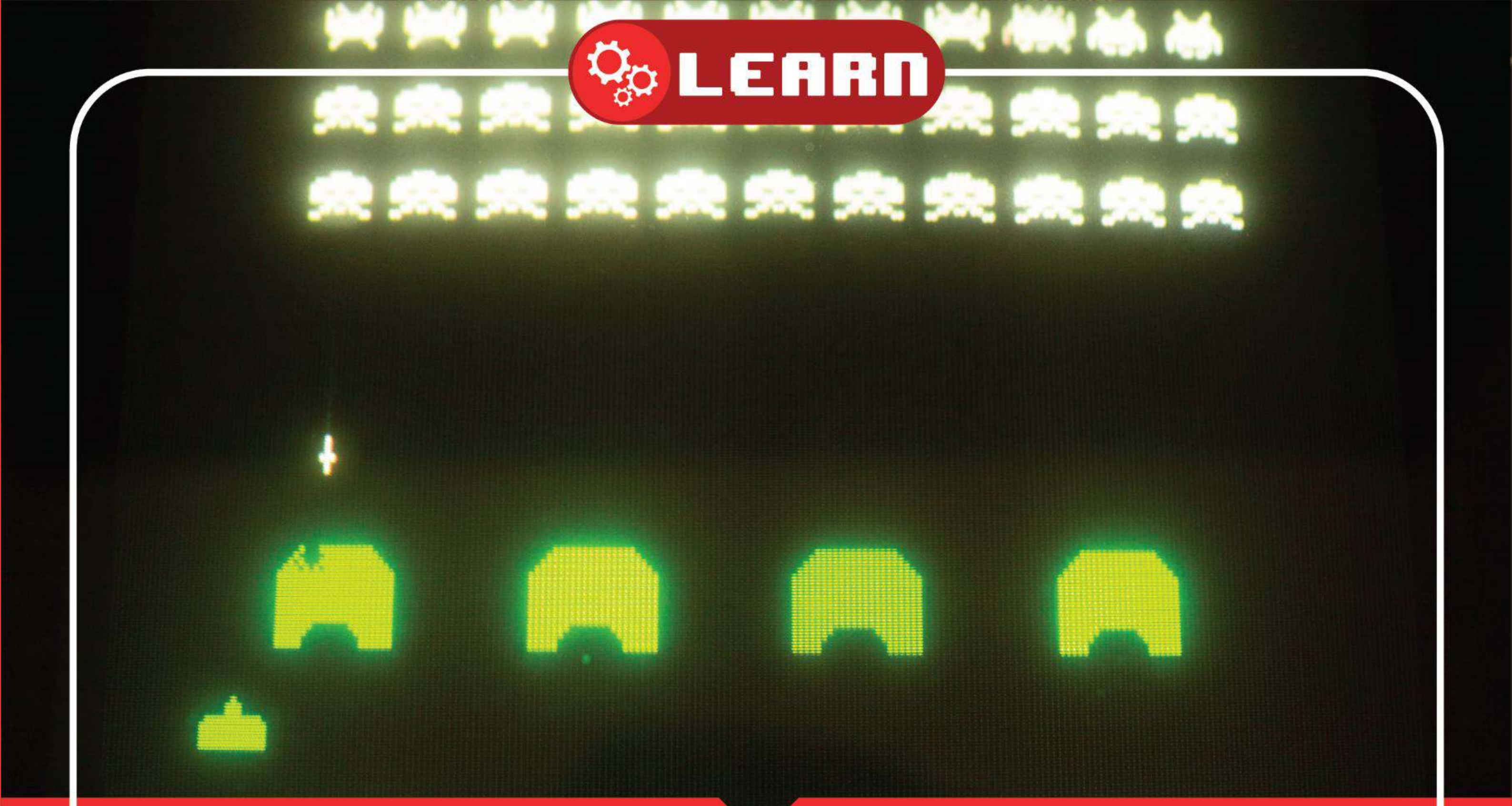


LEARN

CODING SPACE INVADERS PART 3

With 8BitCADE

Compatible with:

8BitCADE XL

AND

8BitCADE

8BitCADE.com

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: 3

Lesson Title: Animating Invaders

Code: Full Code for Lesson

System: Arduboy + Project ABE

Prerequisites to completing this tutorial

1. Tutorials 1a,1b,1c,2
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**

Step 1 ADDING CONSTANTS

Type the following code into line 16-17

```
16. #define INVADERS_DROP_BY 4
17. #define INVADERS_SPEED 12
```

Explanation (line 17-31)
We've introduced two more constants to support the alien movement on screen.
The **INVADERS_DROP_BY** definition is how many pixels the Invaders will move down when they reach the edge of the display. The **INVADERS_SPEED** is how fast they move across the screen at the start of a new level, the lower the number the faster they will go, which is kind of counter-intuitive I know, but the programming is a little easier that way. I've used an initial value of 12, which seems about right. Be warned that values should be between 0 and 127 as we use a counter variable later on in combination with this value which is a signed byte and the largest value for a signed byte is 127.

Step 2 UPDATING GAME STATUS FOR EACH INVADER

Type the following code into line 45 only

```
41. // Game Structures
42. struct GameObjectStruct {
43.   signed int X;
44.   signed int Y;
45.   unsigned char Status;  //0 active, 1 exploding, 2 destroyed
46. };
```

Explanation (line 17-31)

In the vast majority of games something can be destroyed and Invaders is obviously no different in this respect. As every object in this game can be active or destroyed we need a variable within the GameObjectStruct structure to identify if this object is “active” or not (destroyed).

Step 3 DEFINE ACTIVE CONSTANT FOR GAME STATUS

Type the following code into line 18-19

```
18. // Status of a game object constants
19. #define ACTIVE 0
```

Explanation (line 17-31)

We now have a new variable *Status* that represents an objects current status. To Support this we have a new #define as well.

Step 4 UPDATE GRAPHIC SPRITES FRAME 2

Type the following code into line 24,25, + 31,32, + 37,38

```
20. // Invader Sprites
21. const unsigned char InvaderTopGfx [] PROGMEM = {
22.   0x98, 0x5c, 0xb6, 0x5f, 0x5f, 0xb6, 0x5c, 0x98
23. };
24. const unsigned char InvaderTopGfx2 [] PROGMEM = {
25.   0x58, 0xbc, 0x16, 0x1f, 0x1f, 0x16, 0xbc, 0x58
26. };
27. const unsigned char PROGMEM InvaderMiddleGfx [] =
28. {
29.   0x1e, 0xb8, 0x7d, 0x36, 0x3c, 0x3c, 0x3c, 0x36, 0x7d, 0xb8, 0x1e
30. };
31. const unsigned char PROGMEM InvaderMiddleGfx2 [] = {
32.   0x78, 0x18, 0x7d, 0xb6, 0xbc, 0x3c, 0xbc, 0xb6, 0x7d, 0x18, 0x78
33. };
34. const unsigned char PROGMEM InvaderBottomGfx [] = {
35.   0x1c, 0x5e, 0xfe, 0xb6, 0x37, 0x5f, 0x5f, 0x37, 0xb6, 0xfe, 0x5e, 0x1c
36. };
37. const unsigned char PROGMEM InvaderBottomGfx2 [] = {
38.   0x9c, 0xde, 0x7e, 0x36, 0x37, 0x5f, 0x5f, 0x37, 0x36, 0x7e, 0xde, 0x9c
39. };
```

Explanation (line 24,25, + 31,32, + 37,38)

Space Invaders has very simple animations, for each alien character there are two separate graphics defined (i.e. two frames of animation) . We change between each one every time we move the Invaders. We just need to keep track of which one is currently being displayed. Each extra frame of animation for each Invader is simply named with a "2" at the end, as shown here for the middle Invader:

Step 5 VARIABLES FOR INVADER MOVEMENT

Type the following code into line 17-31

```
51. char AlienXMoveAmount = 1; // norm is 2 , this is pixel movement in x
52. signed char InvadersMoveCounter;            // counts down, when 0 move invaders, set according to how many aliens on screen
53. bool AnimationFrame = false; // two frames of animation, if true show one if false show the other
```

Explanation (line 17-31)

Line 51:AlienXMoveAmount : The number of pixels that the Invaders will move at a time. This changes as the game speeds up towards the end of the level. The higher the number the quicker they appear to move.
Line 52: InvadersMoveCounter : Used in conjunction with **INVADERS_SPEED** constant. This counter gets set to **INVADERS_SPEED** and then is decremented once per game cycle. When it reaches 0 we move the Invaders and reset back to the **INVADERS_SPEED** value.
Line 53: The variable that keeps track of which animation to display is this one. If it's false we will display one frame of the Invaders animation, if true the other frame. When and how we toggle this from value to another will be discussed later.

Step 6 THE LOOP

Type the following code into line 17-31

```
67. void loop()
68. {
69.   if (!arduboy.nextFrame()) {
70.     return;
71.   }
72.   Physics();
73.   UpdateDisplay();
74. }
```

Explanation (line 17-31)

Just two functions, the first **Physics()** is basically all the game logic and rules, from moving the Invaders to checking for collisions.

Step 7 CONTROLLING ALIENS

Type the following code into line 17-31

```
76. void Physics()  {
77.   AlienControl();
78. }
```

Explanation (line 17-31)

There will be more in this routine in later tutorials but for now the only thing we are implementing are the Invaders movements with the function **AlienControl()**.

Step 8 ALIEN CONTROL

Type the following code into line 80-107

```
80. void AlienControl()
81. {
82.   if ((InvadersMoveCounter--) < 0)
83.   {
84.     bool Dropped = false;
85.     if ((RightMostPos() + AlienXMoveAmount >= SCREEN_WIDTH) | (LeftMostPos() + AlienXMoveAmount < 0)) // at edge of screen
86.     {
87.       AlienXMoveAmount = -AlienXMoveAmount;             // reverse direction
88.       Dropped = true;                                   // and indicate we are dropping
89.     }
90.     // update the alien postions
91.     for (int Across = 0; Across < NUM_ALIEN_COLUMNS; Across++)
92.     {
93.       for (int Down = 0; Down < 3; Down++)
94.       {
95.         if (Alien[Across][Down].Ord.Status == ACTIVE)
96.         {
97.           if (Dropped == false)
98.             Alien[Across][Down].Ord.X += AlienXMoveAmount;
99.           else
100.                 Alien[Across][Down].Ord.Y += INVADERS_DROP_BY;
101.             }
102.           }
103.         }
104.         InvadersMoveCounter = INVADERS_SPEED;
105.         AnimationFrame = !AnimationFrame; ///swap to other frame
106.       }
107.     }
```

Explanation (line 80-107)

This entire routine exists just to move the invaders on screen, that is to say alter their X and Y positions, it does not plot them to the display, that is done separately by the **UpdateDisplay()** which is called just after the **Physics** routine. It is normal and good practice to separate out your game physics/ logic/rules from the final display to screen. It makes code easier to read, more manageable and disconnects any internal coordinate units we may use from the actual display. What do I mean by that? Well, what if we were suddenly blessed with a screen display of 256×128 (twice the resolution). It would be relatively trivial in the display routine to make our game work on the new display without changing any of the **physics** code at all. That could be all the same and we just scale to the new display in our display code. If you have plotting to screen all intertwined with your game logic it becomes an unmanageable mess even without considering different displays, so keep them separate. If you remember (hopefully, it was only a few lines ago!)

InvadersMoveCounter is basically controlling the speed of the Invaders across the screen. Looking at the line its value is first decremented by 1 and then a check is made to see if it is less than 0. If so then the Invaders are moved else we basically do nothing and exit the routine ready to be called again. At the beginning this variable is set to **INVADERS_SPEED** , which itself is set to 12. So every 12 cycles of the main loop we update the Invaders position. At least at the start of the level that is the case for their speed.
As an aside this line "**if((InvadersMoveCounter–)<0)**" may have some "**C**" purists shouting that it could have been written in a different slightly more compact way and in by-gone days it may have been a little faster executing. However modern compilers should ensure there is no speed penalty and for beginners the way the line is written is more approachable for beginners or those more used to strongly typed programming languages. So if it is time to update the Invaders positions (**InvadersMoveCounter** is less than 0) then we go on to line 145 and create a local variable:

```
84 bool Dropped=false;
```

Dropped is an indicator of whether the Invaders have reached the end of the screen and are dropping down. We need to know this as if it is true we only move down and do not update the X positions in this current Invader position update. The next three lines check if we are at the edge of screen and set and change some variables if we are

```
    if((RightMostPos()+AlienXMoveAmount>=SCREEN_WIDTH)
 |(LeftMostPos()+AlienXMoveAmount<0)) // at edge of screen
    {
      AlienXMoveAmount=-AlienXMoveAmount;          // reverse direction
      Dropped=true;                          // and indicate we are dropping
    }
```

The first line may look a little complex but it can be broken down into two parts, the check for the Invaders going beyond the right edge of the screen and the check for them going beyond the left edge. This is the check for the right:

(RightMostPos()+AlienXMoveAmount>=SCREEN_WIDTH)

The function (which we will look at shortly) **RightMostPos** returns the right most Invaders X position. Obviously the right most Invader changes as the game progresses as some are destroyed, so this routine simply scans the Invaders for the rightmost one that is still active and returns that X position. We then add to that, the amount we would like them to move by, **AlienXMoveAmount**. If this total is more than or equal to the screen width (**SCREEN_WIDTH**) we have therefore hit the right hand edge. The left hand edge check is very similar, **(LeftMostPos()+AlienXMoveAmount<0)**
If the left most Invader + **AlienXMoveAmount** is less than 0 we have hit the left hand edge of the screen. But… some of you may be wondering how that could possibly work, how could adding the left most position which must be 0 or above to a movement amount ever be a value less than 0? Well if we are moving to the left the **AlienXMoveAmount** will actually be a

negative number, i.e. if move to the right it would be, i.e. +2 (or just 2), if to the left it would be -2. So if the LeftMostPos was 0 we would have the following sum to be worked out of we were moving left:
0+**AlienXMoveAmount** and if **AlienXMoveAmount** is -2 then the line becomes 0+-2. Adding a minus number to a positive ends up subtracting it from that positive number. So the answer would be -2 and less than 0 so we know we've hit the left hand edge. OK, so if we have gone beyond the left or right edge we will execute the following code:

```
 AlienXMoveAmount=-AlienXMoveAmount;          // reverse direction
 Dropped=true;                                // and indicate we are dropping
```

Whichever edge we have reached the code is the same as we need to indicate we are dropping, **Dropped=true**. We then need to reverse the direction of movement, the line **AlienXMoveAmount=-AlienXMoveAmount.** Does this reverse of direction by changing the mathematical sign of the movement value. Let's look at a couple of examples, if **AlienXMoveAmount** was 2 then the line above (if we filled in the numbers) would be: **AlienXMoveAmount=-2** and so we have swapped a +2 to a -2. But what if it is already -2, well let's put the numbers in again: **AlienXMoveAmount=–2** We now have a minus minus 2 (–2) , one thing learnt from school is that if you minus a minus number you get a positive number so the above becomes **AlienXMoveAmount=2** and we've swapped the direction of movement.

Updating the Invaders positions

```
 // 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;
        }
      }
    }
```

Here we just loop through every single Invader and the key lines are these:

```
 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;
 }
```

If the invader is active (i.e. not destroyed) we update its position. If we are currently dropping (remember this is set above) we update its Y position only else we update its X position, which will move it either to the left or right depending on the mathematical sign of **AlienXMoveAmount**, discussed earlier. The last two lines of this routine reset the Invader Speed Counter back to its default and flip the animation frame to the next ready for the display routine.

InvadersMoveCounter= INVADERS_SPEED;
AnimationFrame=!AnimationFrame; ///swap to other frame

The **!** symbol in C (and other languages) means simply “invert” or make opposite and applies only to **True** and **false** expressions (**Boolean** values). So false would become true and true would become false etc.

Step 9 LEFT & RIGHTMOST POSITIONS

Type the following code into line 108-150

```
108.        int RightMostPos()  {
109.          //returns x pos of right most alien
110.          int Across = NUM_ALIEN_COLUMNS - 1;
111.          int Down;
112.          int Largest = 0;
113.          int RightPos;
114.          while (Across >= 0) {
115.            Down = 0;
116.            while (Down < NUM_ALIEN_ROWS) {
117.              if (Alien[Across][Down].Ord.Status == ACTIVE)
118.              {
119.                // different aliens have different widths, add to x pos to get rightpos
120.                RightPos = Alien[Across][Down].Ord.X + AlienWidth[Down];
121.                if (RightPos > Largest)
122.                  Largest = RightPos;
123.              }
124.              Down++;
125.            }
126.            if (Largest > 0) // we have found largest for this column
127.              return Largest;
128.            Across--;
129.          }
130.          return 0;  // should never get this far
131.        }
132.        int LeftMostPos()  {
133.          //returns x pos of left most alien
134.          int Across = 0;
135.          int Down;
136.          int Smallest = SCREEN_WIDTH * 2;
137.          while (Across < NUM_ALIEN_COLUMNS) {
138.            Down = 0;
139.            while (Down < 3) {
140.              if (Alien[Across][Down].Ord.Status == ACTIVE)
141.                if (Alien[Across][Down].Ord.X < Smallest)
142.                  Smallest = Alien[Across][Down].Ord.X;
143.              Down++;
144.            }
145.            if (Smallest < SCREEN_WIDTH * 2) // we have found smalest for this coloum
146.              return Smallest;
147.            Across++;
148.          }
149.          return 0;  // should never get this far
150.        }
```

Explanation (line 108-150)

The Left and Right most positions
In the previous routine we needed to know the left or right most positions of the Invaders and introduced two routines called **LeftMostPos** and **RightMostPos**, lets have a look at the code for **LeftMostPos**.

```
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<NUM_ALIEN_ROWS){
      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 smallest for this coloum
      return Smallest;
    Across++;
  }
  return 0;  // should never get this far
}
```

We set up two counters Across and Down to loop through all the Invaders and also we set a variable (**Smallest**) to store the currently smallest X position currently found which we initialise to a massive number of twice the screen width, which will be bigger than any current X position. Now we could have gone through every single Invader comparing their X position with the smallest X position and if that value was smaller storing it in the variable **Smallest**. Then at the end of the routine just return that variable, however there is a more efficient way. The smallest value for X will always be in the left most Invader column as all the other columns occur after this then they must be higher and thus cannot be smaller. So within the loop we go from the left column (Across=0) to the right column. We then scan down each invader in this column (Down=0 to last row). If the Invader is active we check if there X value is less than the current Smallest X value

if(Alien[Across][Down].Ord.X<Smallest)

If it is then we store this as the current smallest value

Smallest=Alien[Across][Down].Ord.X

When we've checked all the rows in a column we check if the **Smallest** variable has changed from what it was initially set at **if(Smallest<SCREEN_WIDTH*2)**. If so we know we have now found the smallest X position for the left most Invader and we exit the routine with **return Smallest**.

The **RightMostPos** function is very similar with just the small matter of having to also include an Invaders width into the calculation, so I won't go over that but leave it for the reader to examine.

Step 10 UPDATE DISPLAY

Update the following code to include additional graphics/sprite images

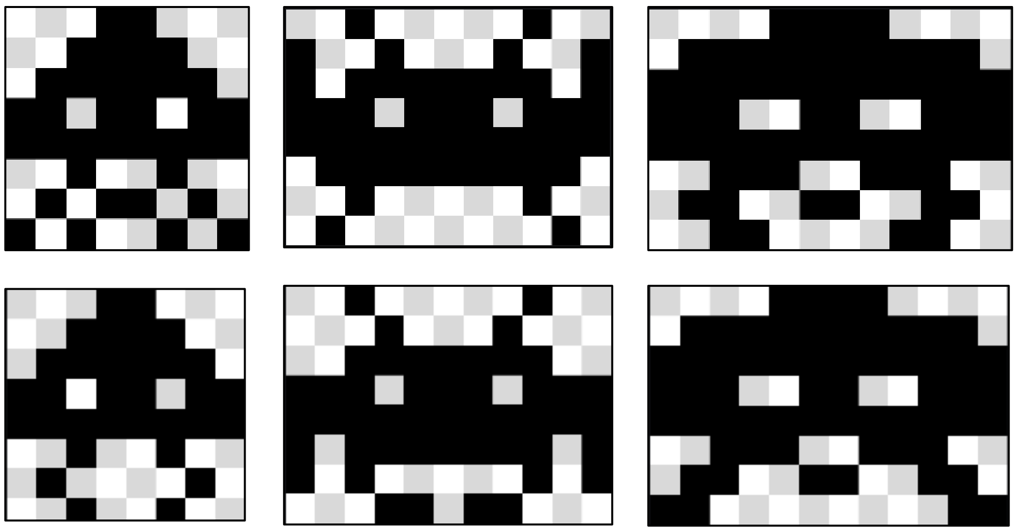
```
152.     void UpdateDisplay()
153.     {
154.       arduboy.clear();
155.       for (int across = 0; across < NUM_ALIEN_COLUMNS; across++)
156.       {
157.         for (int down = 0; down < NUM_ALIEN_ROWS; down++)
158.         {
159.           switch (down)  {
160.             case 0:
161.               if (AnimationFrame)
162.                 arduboy.drawBitmap(Alien[across][down].Ord.X, Alien[across][down].Ord.Y,  InvaderTopGfx, AlienWidth[down], 8, WHITE);
163.               else
164.                 arduboy.drawBitmap(Alien[across][down].Ord.X, Alien[across][down].Ord.Y,  InvaderTopGfx2, AlienWidth[down], 8, WHITE);
165.               break;
166.             case 1:
167.               if (AnimationFrame)
168.                 arduboy.drawBitmap(Alien[across][down].Ord.X, Alien[across][down].Ord.Y,  InvaderMiddleGfx, AlienWidth[down], 8, WHITE);
169.               else
170.                 arduboy.drawBitmap(Alien[across][down].Ord.X, Alien[across][down].Ord.Y,  InvaderMiddleGfx2, AlienWidth[down], 8, WHITE);
171.               break;
172.             default:
173.               if (AnimationFrame)
174.                 arduboy.drawBitmap(Alien[across][down].Ord.X, Alien[across][down].Ord.Y,  InvaderBottomGfx, AlienWidth[down], 8, WHITE);
175.               else
176.                 arduboy.drawBitmap(Alien[across][down].Ord.X, Alien[across][down].Ord.Y,  InvaderBottomGfx2, AlienWidth[down], 8, WHITE);
177.           }
178.         }
179.       }
180.       arduboy.display();
181.     }
```

Explanation (line 152-181)

This was discussed in part 2 and has been altered only slightly with the addition of a decision as to which animation frame to display, for example for the top Invader (row 0) these are the lines that accomplish this:

```
      if(AnimationFrame)
161   display.drawBitmap(Alien[across][down].Ord.X,
162 Alien[across][down].Ord.Y,  InvaderTopGfx, AlienWidth[down], 8,WHITE);
163 else
164   display.drawBitmap(Alien[across][down].Ord.X,
      Alien[across][down].Ord.Y,  InvaderTopGfx2, AlienWidth[down], 8,WHITE);
```

So if **AnimationFrame** is true draw the normal **InvaderTopGfx** graphic else draw **InvaderTopGfx2** graphic (which of course it the other frame of the animation.



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
//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
// 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
};

// Game Structures
struct GameObjectStruct {
  signed int X;
  signed int Y;
  unsigned char Status;  //0 active, 1 exploding, 2 destroyed
};
struct AlienStruct {
  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

//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
}

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

void Physics()  {
  AlienControl();
}

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 column
          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 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);
          }
        }
      }
      arduboy.display();
    }

    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; // -down helps centrilse them
          Alien[across][down].Ord.Y = YStart + (down * SPACE_BETWEEN_ROWS); // calculation to space aliens on Y axis
        }
      }
    }
```