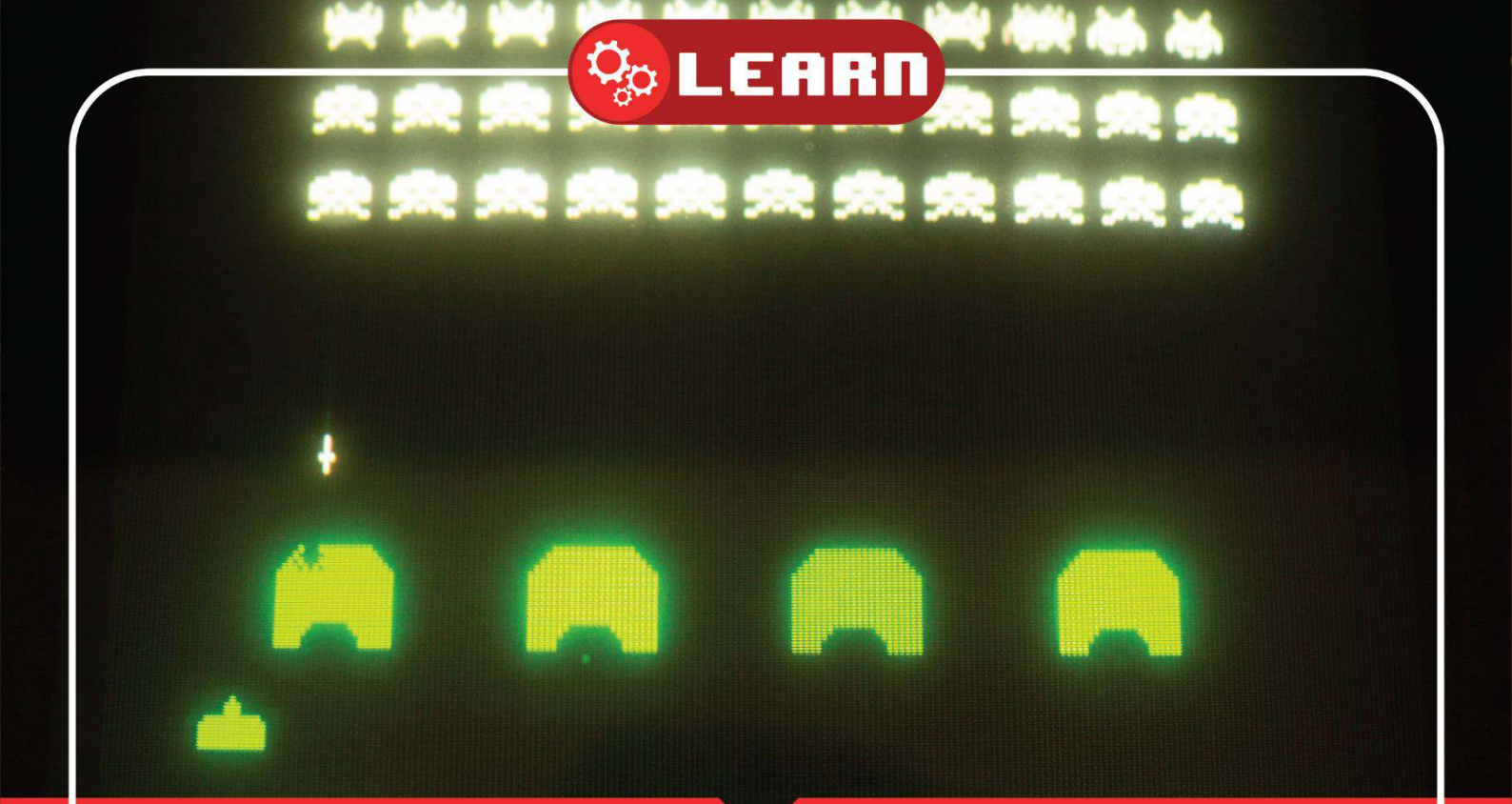
LEARN



# CODING SPACE INVADERS PART 6

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: 6

Lesson Title: Adding Exploding Sprite

Code: Full Code for Lesson

System: Arduboy + Project ABE

Prerequisites to completing this tutorial

1. Tutorials 1, 1a, 1b ,2, 3, 4, 5
2. Know how to write code in either **Arduino IDE** or **Project ABE online Emulator**

In this tutorial we will add the explosion animations for the Invaders.

## Step 1 ADDING LENGTH OF EXPLOSION CONSTANT

Type the following code into line 23

```
23. #define EXPLOSION_GFX_TIME 7  // How long an ExplosionGfx remains on screen before disappearing
```

**Explanation (line 23)**

The amount of time the explosion graphic is on screen is determined by this constant. The higher the number the longer the graphic remains on screen.

## Step 2 UPDATING **OBJECT STATUS**

Type the following code into line 35

```
33. // Status of a game object constants
34. #define ACTIVE 0
35. #define EXPLODING 1
36. #define DESTROYED 2
```

**Explanation (line 35)**

An object's status could be either ACTIVE or DESTROYED but now (if you look below) we have introduced a new status called EXPLODING. This code allows the **UpdateDisplay** routine to know when it should be displaying the explosion graphic for that object rather than its normal graphic.

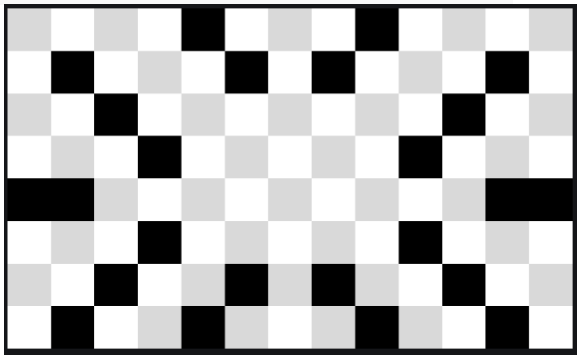
## Step 3 ADDING **EXPLOSION** GRAPHICS

Type the following code into line 75-77

```
75. static const unsigned char PROGMEM ExplosionGfx [] = {
76.   0x10, 0x92, 0x44, 0x28, 0x81, 0x42, 0x00, 0x42, 0x81, 0x28, 0x44, 0x92, 0x10
77. };
```

**Explanation (line 75-77)**

The definition for the explosion starts at line 75 and will not be discussed further as graphics were covered in an earlier tutorial. The graphic is below and was created using www.pixilart.com.



## Step 4 UPDATING **ALIEN STRUCTURE**

Type the following code into line 90

```
88. struct AlienStruct  {
89.   GameObjectStruct Ord;
90.   unsigned char ExplosionGfxCounter; // how long we want the ExplosionGfx to last
91. };
```

**Explanation (line 90)**

We have added the variable to time how long the explosion remains on this screen for this Alien. Initially when it is hit it will be filled with the EXPLOSION_GFX_TIME constant value and then decremented for every time the code loops round its main loop.

## Step 5 UPDATING MISSILE AND ALIEN COLLISIONS

UPDATE the following code in line 213

```
200.      void MissileAndAlienCollisions()
201.      {
202.        for (int across = 0; across < NUM_ALIEN_COLUMNS; across++)
203.        {
204.          for (int down = 0; down < NUM_ALIEN_ROWS; down++)
205.          {
206.            if (Alien[across][down].Ord.Status == ACTIVE)
207.            {
208.              if (Missile.Status == ACTIVE)
209.              {
210.                if (Collision(Missile, MISSILE_WIDTH, MISSILE_HEIGHT, Alien[across][down].Ord, AlienWidth[down], INVADER_HEIGHT))
211.                {
212.                  // missile hit
213.                  Alien[across][down].Ord.Status = EXPLODING;
214.                  Missile.Status = DESTROYED;
215.                }
216.              }
217.            }
218.          }
219.        }
220.      }
```

**Explanation (line 213)**

Previously within the routine **MissileAndAlienCollisions** we had a line that read **Alien[across][down].Ord.Status=DESTROYED;** Now (on line 213) we have changed this to **Alien[across][down].Ord.Status=EXPLODING;** To indicate that this Alien is exploding – not yet destroyed.

## Step 6 ADD CODE TO **UPDATE DISPLAY** TO DISPLAY EXPLOSION GRAPHICS

Type the following code into line 300-309

```
300.            else  {
301.              if (Alien[across][down].Ord.Status == EXPLODING) {
302.                Alien[across][down].ExplosionGfxCounter--;
303.                if (Alien[across][down].ExplosionGfxCounter > 0)  {
304.                  arduboy.drawBitmap(Alien[across][down].Ord.X, Alien[across][down].Ord.Y,ExplosionGfx, 13, 8, WHITE);
305.                }
306.                else
307.                  Alien[across][down].Ord.Status = DESTROYED;
308.              }
309.            }
```

**Explanation (line 300-309)**

In the **UpdateDisplay** routine we have added these lines at line 300. This code is executed if the Invader is not active (not shown here but in main code at the end of the tutorial under final code). Looking above we can see that if the Invader has the status of EXPLODING then we go on to decrement the explosion counter. Then we decide whether to display the explosion or mark the Invader as destroyed. If the explosion counter (ExplosionGfxCounter) is more than 0 we display the explosion else and we mark the Invader as DESTROYED and then nothing else will happen for this Invader for this wave of Invaders.

# Step 7 UPDATING INVADER INITIALISATION

Type the following code into line 335-336

```
333.            Alien[across][down].Ord.X = X_START_OFFSET + (across * (LARGEST_ALIEN_WIDTH + SPACE_BETWEEN_ALIEN_COLUMNS)) - (AlienWidth[down] / 2);
334.            Alien[across][down].Ord.Y = YStart + (down * SPACE_BETWEEN_ROWS);
335.            Alien[across][down].Ord.Status = ACTIVE;
336.            Alien[across][down].ExplosionGfxCounter = EXPLOSION_GFX_TIME;
337.          }
338.        }
339.      }
```

**Explanation (line 335-336)**
Lines 335-336 are the new lines showing that we are setting the Invader Status to ACTIVE (this could have been included prior to this tutorial to be fair) and we also set the explosion counter to its starting point ready for an explosion to happen.

# Final Code

```
1.  /*   www.pixilart.com for sprite generation
2.    http://javl.github.io/image2cpp/ for image conversion
3.  */
4.  #include <Arduboy2.h>
5.  Arduboy2 arduboy;
6.  // DISPLAY SETTINGS
7.  #define SCREEN_WIDTH 128
8.  #define SCREEN_HEIGHT 64
9.  // Input settings
10. #define FIRE_BUT 7
11. #define RIGHT_BUT A1
12. #define LEFT_BUT A2
13. // Alien Settings
14. #define NUM_ALIEN_COLUMNS 7
15. #define NUM_ALIEN_ROWS 3
16. #define X_START_OFFSET 6
17. #define SPACE_BETWEEN_ALIEN_COLUMNS 5
18. #define LARGEST_ALIEN_WIDTH 11
19. #define SPACE_BETWEEN_ROWS 9
20. #define INVADERS_DROP_BY 4            // pixel amount that invaders move down by
21. #define INVADERS_SPEED 20             // speed of movement, lower=faster.
22. #define INVADER_HEIGHT 8
23. #define EXPLOSION_GFX_TIME 7  // How long an ExplosionGfx remains on screen before
    disappearing
24. // Player settingsc
25. #define TANKGFX_WIDTH 13
26. #define TANKGFX_HEIGHT 8
27. #define PLAYER_X_MOVE_AMOUNT 1
28. #define PLAYER_Y_START 56
29. #define PLAYER_X_START 0
30. #define MISSILE_HEIGHT 4
31. #define MISSILE_WIDTH 1
32. #define MISSILE_SPEED 1
33. // Status of a game object constants
34. #define ACTIVE 0
35. #define EXPLODING 1
36. #define DESTROYED 2
37. // graphics
38. // aliens
39.
40. const unsigned char InvaderTopGfx [] PROGMEM = {
41.   0x98, 0x5c, 0xb6, 0x5f, 0x5f, 0xb6, 0x5c, 0x98
42. };
43.
44. const unsigned char InvaderTopGfx2 [] PROGMEM = {
```

```
45.   0x58, 0xbc, 0x16, 0x1f, 0x1f, 0x16, 0xbc, 0x58
46. };
47.
48. const unsigned char PROGMEM InvaderMiddleGfx [] =
49. {
50.   0x1e, 0xb8, 0x7d, 0x36, 0x3c, 0x3c, 0x3c, 0x36, 0x7d, 0xb8, 0x1e
51. };
52.
53. const unsigned char PROGMEM InvaderMiddleGfx2 [] = {
54.   0x78, 0x18, 0x7d, 0xb6, 0xbc, 0x3c, 0xbc, 0xb6, 0x7d, 0x18, 0x78
55. };
56.
57. const unsigned char PROGMEM InvaderBottomGfx [] = {
58.   0x1c, 0x5e, 0xfe, 0xb6, 0x37, 0x5f, 0x5f, 0x37, 0xb6, 0xfe, 0x5e, 0x1c
59. };
60.
61. const unsigned char PROGMEM InvaderBottomGfx2 [] = {
62.   0x9c, 0xde, 0x7e, 0x36, 0x37, 0x5f, 0x5f, 0x37, 0x36, 0x7e, 0xde, 0x9c
63. };
64.
65. // Player grafix
66. const unsigned char PROGMEM TankGfx [] = {
67.   0xf0, 0xf8, 0xf8, 0xf8, 0xf8, 0xfe, 0xff, 0xfe, 0xf8, 0xf8, 0xf8, 0xf8, 0xf0
68. };
69.
70. static const unsigned char PROGMEM MissileGfx [] = {
71.   0x0f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
72.
73. };
74.
75. static const unsigned char PROGMEM ExplosionGfx [] = {
76.   0x10, 0x92, 0x44, 0x28, 0x81, 0x42, 0x00, 0x42, 0x81, 0x28, 0x44, 0x92, 0x10
77. };
78.
79. // Game structures
80.
81. struct GameObjectStruct  {
82.   // base object which most other objects will include
83.   signed int X;
84.   signed int Y;
85.   unsigned char Status;  //0 active, 1 exploding, 2 destroyed
86. };
87.
88. struct AlienStruct  {
89.   GameObjectStruct Ord;
90.   unsigned char ExplosionGfxCounter; // how long we want the ExplosionGfx to last
91. };
92.
93. struct PlayerStruct  {
94.   GameObjectStruct Ord;
95. };
96.
97. //alien global vars
98. //The array of aliens across the screen
99. AlienStruct  Alien[NUM_ALIEN_COLUMNS][NUM_ALIEN_ROWS];
100.
101.       // widths of aliens
102.       // as aliens are the same type per row we do not need to store their graphic
     width per alien in the structure above
103.       // that would take a byte per alien rather than just three entries here, 1 p
    er row, saving significnt memory
104.       byte AlienWidth[] = {8, 11, 12}; // top, middle ,bottom widths
105.
106.       char AlienXMoveAmount = 1; // norm is 2 , this is pixel movement in X
107.       signed char InvadersMoveCounter;            // counts down, when 0 move inva
    ders, set according to how many aliens on screen
```

```
108.       bool AnimationFrame = false; // two frames of animation, if true show one if
     false show the other
109.
110.       // Player global variables
111.       PlayerStruct Player;
112.       GameObjectStruct Missile;
113.
114.       void setup()
115.       {
116.         arduboy.begin();
117.         arduboy.setFrameRate(60);
118.         InitAliens(0);
119.         InitPlayer();
120.         pinMode(RIGHT_BUT, INPUT_PULLUP);
121.         pinMode(LEFT_BUT, INPUT_PULLUP);
122.         pinMode(FIRE_BUT, INPUT_PULLUP);
123.       }
124.
125.       void loop()
126.       {
127.         if (!arduboy.nextFrame()) {
128.           return;
129.         }
130.         Physics();
131.         UpdateDisplay();
132.       }
133.
134.       void Physics()  {
135.         AlienControl();
136.         PlayerControl();
137.         MissileControl();
138.         CheckCollisions();
139.       }
140.
141.       void PlayerControl()  {
142.         // user input checks
143.         if ((digitalRead(RIGHT_BUT) == 0) & (Player.Ord.X + TANKGFX_WIDTH < SCREEN
    _WIDTH))
144.           Player.Ord.X += PLAYER_X_MOVE_AMOUNT;
145.         if ((digitalRead(LEFT_BUT) == 0) & (Player.Ord.X > 0))
146.           Player.Ord.X -= PLAYER_X_MOVE_AMOUNT;
147.         if ((digitalRead(FIRE_BUT) == 0) & (Missile.Status != ACTIVE))
148.         {
149.           Missile.X = Player.Ord.X + (6); // offset missile so its in the mideel o
    f the tank
150.           Missile.Y = PLAYER_Y_START;
151.           Missile.Status = ACTIVE;
152.
153.         }
154.       }
155.
156.       void MissileControl()
157.       {
158.         if (Missile.Status == ACTIVE)
159.         {
160.           Missile.Y -= MISSILE_SPEED;
161.           if (Missile.Y + MISSILE_HEIGHT < 0)  // If off top of screen destroy so
    can be used again
162.             Missile.Status = DESTROYED;
163.         }
164.       }
165.
166.       void AlienControl()
167.       {
168.         if ((InvadersMoveCounter--) < 0)
169.         {
```

```
170.           bool Dropped = false;
171.           if ((RightMostPos() + AlienXMoveAmount >= SCREEN_WIDTH) | (LeftMostPos()
     + AlienXMoveAmount < 0)) // at edge of screen
172.           {
173.             AlienXMoveAmount = -
    AlienXMoveAmount;             // reverse direction
174.             Dropped = true;                                    // and indicate we a
    re dropping
175.           }
176.           // update the alien postions
177.           for (int Across = 0; Across < NUM_ALIEN_COLUMNS; Across++)
178.           {
179.             for (int Down = 0; Down < 3; Down++)
180.             {
181.               if (Alien[Across][Down].Ord.Status == ACTIVE)
182.               {
183.                 if (Dropped == false)
184.                   Alien[Across][Down].Ord.X += AlienXMoveAmount;
185.                 else
186.                   Alien[Across][Down].Ord.Y += INVADERS_DROP_BY;
187.               }
188.             }
189.           }
190.           InvadersMoveCounter = INVADERS_SPEED;
191.           AnimationFrame = !AnimationFrame; ///swap to other frame
192.         }
193.       }
194.
195.       void CheckCollisions()
196.       {
197.         MissileAndAlienCollisions();
198.       }
199.
200.       void MissileAndAlienCollisions()
201.       {
202.         for (int across = 0; across < NUM_ALIEN_COLUMNS; across++)
203.         {
204.           for (int down = 0; down < NUM_ALIEN_ROWS; down++)
205.           {
206.             if (Alien[across][down].Ord.Status == ACTIVE)
207.             {
208.               if (Missile.Status == ACTIVE)
209.               {
210.                 if (Collision(Missile, MISSILE_WIDTH, MISSILE_HEIGHT, Alien[across
    ][down].Ord, AlienWidth[down], INVADER_HEIGHT))
211.                 {
212.                   // missile hit
213.                   Alien[across][down].Ord.Status = EXPLODING;
214.                   Missile.Status = DESTROYED;
215.                 }
216.               }
217.             }
218.           }
219.         }
220.       }
221.
222.       bool Collision(GameObjectStruct Obj1, unsigned char Width1, unsigned char He
    ight1, GameObjectStruct Obj2, unsigned char Width2, unsigned char Height2)
223.       {
224.         return ((Obj1.X + Width1 > Obj2.X) & (Obj1.X < Obj2.X + Width2) & (Obj1.Y
    + Height1 > Obj2.Y) & (Obj1.Y < Obj2.Y + Height2));
225.       }
226.
227.       int RightMostPos()  {
228.         //returns x pos of right most alien
229.         int Across = NUM_ALIEN_COLUMNS - 1;
```

```
230.          int Down;
231.          int Largest = 0;
232.          int RightPos;
233.          while (Across >= 0) {
234.            Down = 0;
235.            while (Down < NUM_ALIEN_ROWS) {
236.              if (Alien[Across][Down].Ord.Status == ACTIVE)
237.              {
238.                // different aliens have different widths, add to x pos to get right
    pos
239.                RightPos = Alien[Across][Down].Ord.X + AlienWidth[Down];
240.                if (RightPos > Largest)
241.                  Largest = RightPos;
242.              }
243.              Down++;
244.            }
245.            if (Largest > 0) // we have found largest for this coloum
246.              return Largest;
247.            Across--;
248.          }
249.          return 0;  // should never get this far
250.        }
251.
252.        int LeftMostPos()  {
253.          //returns x pos of left most alien
254.          int Across = 0;
255.          int Down;
256.          int Smallest = SCREEN_WIDTH * 2;
257.          while (Across < NUM_ALIEN_COLUMNS) {
258.            Down = 0;
259.            while (Down < 3) {
260.              if (Alien[Across][Down].Ord.Status == ACTIVE)
261.                if (Alien[Across][Down].Ord.X < Smallest)
262.                  Smallest = Alien[Across][Down].Ord.X;
263.              Down++;
264.            }
265.            if (Smallest < SCREEN_WIDTH * 2) // we have found smalest for this colou
    m
266.              return Smallest;
267.            Across++;
268.          }
269.          return 0;  // should nevr get this far
270.        }
271.
272.        void UpdateDisplay()
273.        {
274.          arduboy.clear();
275.          for (int across = 0; across < NUM_ALIEN_COLUMNS; across++)
276.          {
277.            for (int down = 0; down < NUM_ALIEN_ROWS; down++)
278.            {
279.              if (Alien[across][down].Ord.Status == ACTIVE) {
280.                switch (down)  {
281.                  case 0:
282.                    if (AnimationFrame)
283.                      arduboy.drawBitmap(Alien[across][down].Ord.X, Alien[across][do
    wn].Ord.Y,  InvaderTopGfx, AlienWidth[down], INVADER_HEIGHT, WHITE);
284.                    else
285.                      arduboy.drawBitmap(Alien[across][down].Ord.X, Alien[across][do
    wn].Ord.Y,  InvaderTopGfx2, AlienWidth[down], INVADER_HEIGHT, WHITE);
286.                    break;
287.                  case 1:
288.                    if (AnimationFrame)
289.                      arduboy.drawBitmap(Alien[across][down].Ord.X, Alien[across][do
    wn].Ord.Y,  InvaderMiddleGfx, AlienWidth[down], INVADER_HEIGHT, WHITE);
290.                    else
```

```
291.                     arduboy.drawBitmap(Alien[across][down].Ord.X, Alien[across][do
    wn].Ord.Y,  InvaderMiddleGfx2, AlienWidth[down], INVADER_HEIGHT, WHITE);
292.                   break;
293.                 default:
294.                   if (AnimationFrame)
295.                     arduboy.drawBitmap(Alien[across][down].Ord.X, Alien[across][do
    wn].Ord.Y,  InvaderBottomGfx, AlienWidth[down], INVADER_HEIGHT, WHITE);
296.                   else
297.                     arduboy.drawBitmap(Alien[across][down].Ord.X, Alien[across][do
    wn].Ord.Y,  InvaderBottomGfx2, AlienWidth[down], INVADER_HEIGHT, WHITE);
298.                }
299.              }
300.              else  {
301.                if (Alien[across][down].Ord.Status == EXPLODING) {
302.                  Alien[across][down].ExplosionGfxCounter--;
303.                  if (Alien[across][down].ExplosionGfxCounter > 0)  {
304.                    arduboy.drawBitmap(Alien[across][down].Ord.X, Alien[across][down
    ].Ord.Y,ExplosionGfx, 13, 8, WHITE);
305.                  }
306.                  else
307.                    Alien[across][down].Ord.Status = DESTROYED;
308.                }
309.              }
310.            }
311.          }
312.
313.          // player
314.          arduboy.drawBitmap(Player.Ord.X, Player.Ord.Y,  TankGfx, TANKGFX_WIDTH, TA
    NKGFX_HEIGHT, WHITE);
315.          //missile
316.          if (Missile.Status == ACTIVE)
317.            arduboy.drawBitmap(Missile.X, Missile.Y,  MissileGfx, MISSILE_WIDTH, MIS
    SILE_HEIGHT, WHITE);
318.
319.          arduboy.display();
320.        }
321.
322.        void InitPlayer()  {
323.          Player.Ord.Y = PLAYER_Y_START;
324.          Player.Ord.X = PLAYER_X_START;
325.          Missile.Status = DESTROYED;
326.        }
327.
328.        void InitAliens(int YStart)  {
329.          for (int across = 0; across < NUM_ALIEN_COLUMNS; across++)  {
330.            for (int down = 0; down < 3; down++)  {
331.              // we add down to centralise the aliens, just happens to be the right
    value we need per row!
332.              // we need to adjust a little as row zero should be 2, row 1 should be
     1 and bottom row 0
333.              Alien[across][down].Ord.X = X_START_OFFSET + (across * (LARGEST_ALIEN_
    WIDTH + SPACE_BETWEEN_ALIEN_COLUMNS)) - (AlienWidth[down] / 2);
334.              Alien[across][down].Ord.Y = YStart + (down * SPACE_BETWEEN_ROWS);
335.              Alien[across][down].Ord.Status = ACTIVE;
336.              Alien[across][down].ExplosionGfxCounter = EXPLOSION_GFX_TIME;
337.            }
338.          }
339.        }
```