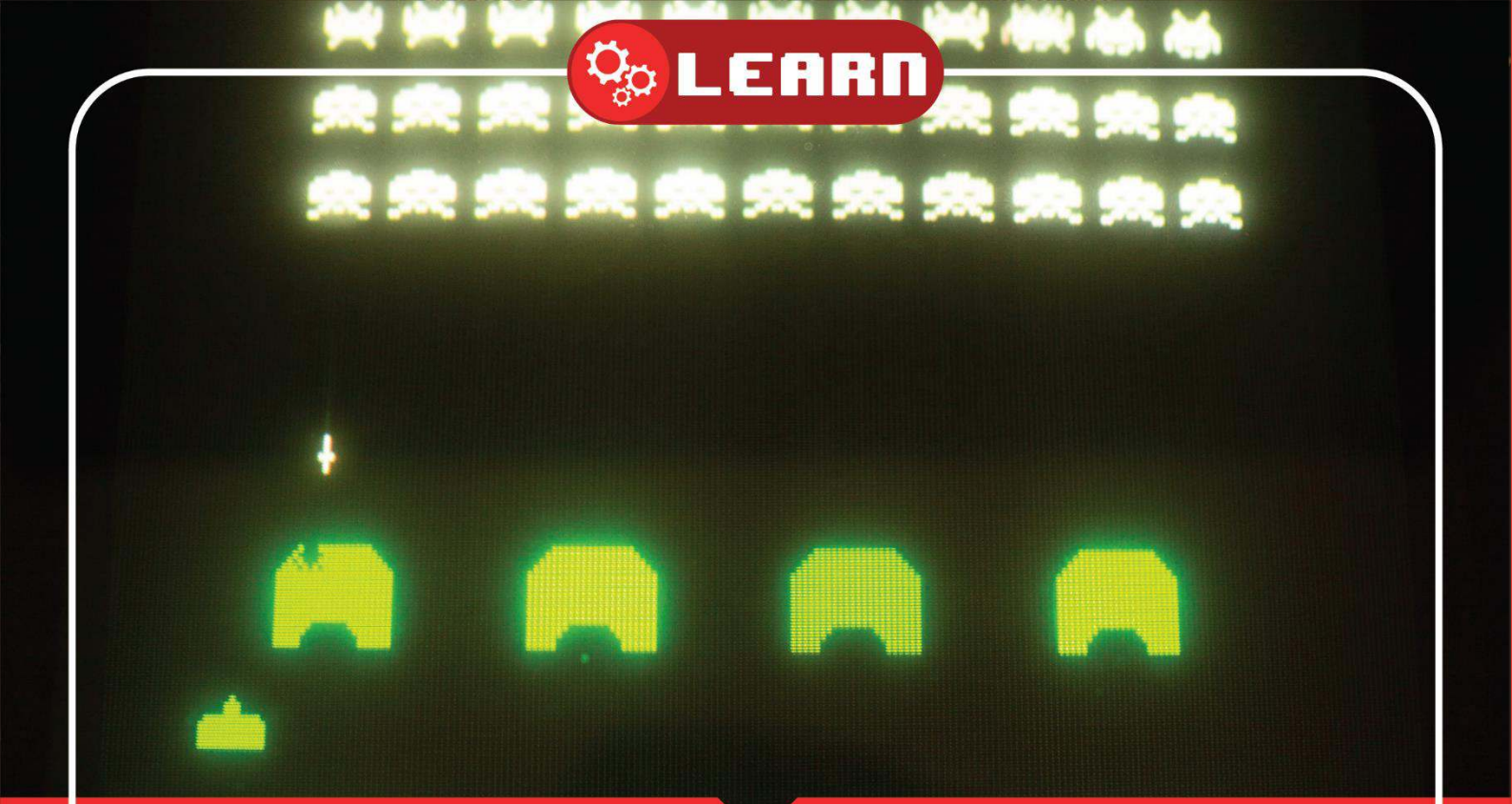
LEARN



CODING SPACE INVADERS

PART 8

With 8BitCADE

Compatible with:

8BitCADE XL

AND

8BitCADE

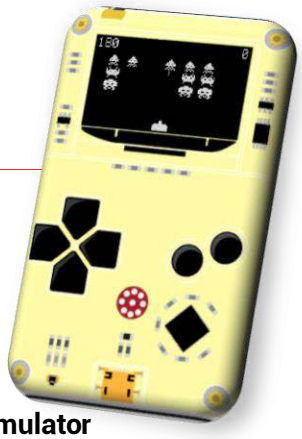
Contents

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: 8



Lesson Title: Adding Scoring

Code: Full Code for Lesson

System: Arduboy + Project ABE

Prerequisites to completing this tutorial

1. Tutorials 1a, 1b, 1c ,2, 3, 4, 5, 6, 7
2. Know how to write code in either **Arduino IDE** or **Project ABE online Emulator**

In this tutorial we'll add scoring and a hi-score.

Step 1 INCLUDE THE EEPROM LIBRARY

Type the following code into line 24-29

```
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.  #include <EEPROM.h>
```

Explanation (line 24-29) In every processor used in the Arduino range of boards there is some memory that is referred to as EEPROM (Electrically Erasable Programmable Read Only Memory). This is memory that we can access to store information during program runtime that will not be lost when the processor is powered down. It will remain there for all intents and purposes for ever. You may wonder why not have all dynamic memory (storage for program data, variables etc.) in EEPROM memory. Well, generally speaking it is slower to access than normal volatile memory so would reduce performance for general programs. So to access this facility on our Arduino's we use this library written for that purpose. Why do we need to use this special memory? To store the Hi-Score, we want this to be kept even when the unit is switched off.

Step 2 UPDATING PLAYER STRUCT TO INCLUDE PLAYER SCORE

Type the following code into line 90-91

```
88. struct PlayerStruct  {
89.   GameObjectStruct Ord;
90.   unsigned int Score;
91.   unsigned char Lives;
92. };
```

Explanation (line 90-91) We've added a extra property to the player structure, their score. Putting in the players structure would allow for an easier expansion to a two player option in the future if we decided to implement one. Also note we've added the Lives property to the player but for now we will do nothing more with it.

Step 3 ADD GAME VARIABLE HI SCORE

Type the following code into line 113

```
109.      // Player global variables
110.      PlayerStruct Player;
111.      GameObjectStruct Missile;
112.      // game variables
113.      unsigned int HiScore;
```

Explanation (line 113) The hi-score is defined at line 213: **unsigned int HiScore;**

Step 4 CHECKING HI SCORE ON STARTUP

Type the following code into line 126-132

```
126.        EEPROM.get(0, HiScore);
127.        if (HiScore == 65535) // new unit never written to
128.        {
129.          HiScore = 0;
130.          EEPROM.put(0, HiScore);
131.        }
132.      }
```

Explanation (line 126-132) The variable **HiScore** is a integer variable and takes up 2 bytes to store up to a maximum value of 65535, which for this implementation will be adequate. Note that with a brand new Arduino all the EEPROM bytes are set to "1"s. Two bytes that are all "1"s is 65535. So on initialisation if the value read using line **EEPROM.get(0,HiScore)** is this value then we clear it to 0. Later we will add in a button to reset the Hi-Score manually should you wish to. Let's look closer at that line **EEPROM.get(0,HiScore);.**

This calls a routine called **get** that will start retrieving bytes from the location passed (0, which is the start of EEPROM storage). It will read the number of bytes that the type **HiScore** needs, in this case as its an INT then it will read two bytes (0 and 1) and store them in HiScore. If you wanted to store another value in EEPROM then it must start at location 2 and not 1 as 1 is already being used to store part of the hi-score. It is very important to know how much storage space certain variable types require and to keep track of your EEPROM storage usage otherwise your code will not work correctly.

So if we are setting the Hi-Score to 0 then the line **EEPROM.put(0,HiScore);** will achieve this. It *puts* the values of the two bytes for the HiScore variable into the EEPROM starting at location 0 (so will use location 0 and 1 because we need two bytes).

Step 5 ADDING SCORE VALUES TO EACH ALIEN

Type the following code into line 24-29

```
151.        unsigned char GetScoreForAlien(int RowNumber)
152.        {
153.          // returns value for killing and alien at the row indicated
154.          switch (RowNumber)
155.          {
156.            case 0: return 30;
157.            case 1: return 20;
158.            case 2: return 10;
159.          }
160.        }
```

Explanation (line 24-29) Different Aliens score a differing amount, the higher up the more points are awarded. The following routine returns the score for a particular Invader on a particular row. All that is needed is to pass in the row number of the Alien that has just been destroyed and it will return the appropriate score. 0 is the top row and 2 is the bottom row.

Step 6 UPDATE MISSILE AND ALIEN COLLISIONS

Type the following code into line 306

```
291.      void MissileAndAlienCollisions()
292.      {
293.        for (int across = 0; across < NUM_ALIEN_COLUMNS; across++)
294.        {
295.          for (int down = 0; down < NUM_ALIEN_ROWS; down++)
296.          {
297.            if (Alien[across][down].Ord.Status == ACTIVE)
298.            {
299.              if (Missile.Status == ACTIVE)
300.              {
301.                if (Collision(Missile, MISSILE_WIDTH, MISSILE_HEIGHT, Alien[across][down].Ord, AlienWidth[down], INVADER_HEIGHT))
302.                {
303.                  // missile hit
304.                  Alien[across][down].Ord.Status = EXPLODING;
305.                  Missile.Status = DESTROYED;
306.                  Player.Score += GetScoreForAlien(down);
307.                }
308.              }
309.            }
310.          }
311.        }
312.      }
```

Explanation (line 306) Line 306 passes in the current row of the destroyed Invader and the players score is updated accordingly.

Step 7 UPDATE MOTHERSHIP COLLISIONS WITH MOTHERSHIP **SCORE**

Type the following code into line 280

```
261.    void MotherShipCollisions()
262.    {
263.      if ((Missile.Status == ACTIVE) & (MotherShip.Ord.Status == ACTIVE))
264.      {
265.        if (Collision(Missile, MISSILE_WIDTH, MISSILE_HEIGHT, MotherShip.Ord, MOTHERSHIP_WIDTH, MOTHERSHIP_HEIGHT))
266.        {
267.          MotherShip.Ord.Status = EXPLODING;
268.          MotherShip.ExplosionGfxCounter = EXPLOSION_GFX_TIME;
269.          Missile.Status = DESTROYED;
270.          // generate the score for the mothership hit, note in the real arcade space invaders the score was not random but
271.          // just appeared so, a player could infulence its value with clever play, but we'll keep it a little simpler
272.          MotherShipBonus = random(4); // a random number between 0 and 3
273.          switch (MotherShipBonus)
274.          {
275.            case 0: MotherShipBonus = 50; break;
276.            case 1: MotherShipBonus = 100; break;
277.            case 2: MotherShipBonus = 150; break;
278.            case 3: MotherShipBonus = 300; break;
279.          }
280.          Player.Score += MotherShipBonus;
281.          MotherShipBonusXPos = MotherShip.Ord.X;
282.          if (MotherShipBonusXPos > 100)                // to ensure isn't half off right hand side of screen
283.            MotherShipBonusXPos = 100;
284.          if (MotherShipBonusXPos < 0)                 // to ensure isn't half off right hand side of screen
285.            MotherShipBonusXPos = 0;
286.          MotherShipBonusCounter = DISPLAY_MOTHERSHIP_BONUS_TIME;
287.        }
288.      }
289.    }
```

Explanation (line 280) We also update the players score when the Mystery-ship is destroyed, here's the collision checking routine from the main code. Here you can see that line 280 updates the players score with whatever bonus has been awarded.

Step 8 ADDING SCORE RESET ON STARTUP

Type the following code into line 451

```
447.      void InitPlayer()  {
448.        Player.Ord.Y = PLAYER_Y_START;
449.        Player.Ord.X = PLAYER_X_START;
450.        Missile.Status = DESTROYED;
451.        Player.Score = 0;
452.      }
```

Explanation (line 451) We clear the players score back to 0 when the game starts.

Step 9 UPDATING VOID UPDATE DISPLAY TO DISPLAY THE SCORE

Type the following code into line 375-383

```
364.      void UpdateDisplay()
365.      {
366.        int i;
367.        arduboy.clear();
368.        // Mothership bonus display if required
369.        if (MotherShipBonusCounter > 0)
370.        {
371.          // mothership bonus
372.          arduboy.setCursor(MotherShipBonusXPos, 0);
373.          arduboy.print(MotherShipBonus);
374.          MotherShipBonusCounter--;
375.        }
376.        else
377.        {
378.          // draw score and lives, anything else can go above them
379.          arduboy.setCursor(0, 0);
380.          arduboy.print(Player.Score);
381.          arduboy.setCursor(SCREEN_WIDTH - 7, 0);
382.          arduboy.print(Player.Lives);
383.        }
```

Explanation (line 375-383) Looking in the display routine we find the previous lines for Mystery-Ship bonus and now we've added some more for score and lives. We can see that we only display them if there is no bonus being displayed. This is because if the bonus is printing over the score then having both together on screen at once makes the bonus illegible.

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.  #include <EEPROM.h>
7.  // DISPLAY SETTINGS
8.  #define SCREEN_WIDTH 128
9.  #define SCREEN_HEIGHT 64
10. // Input settings
11. #define FIRE_BUT 7
12. #define RIGHT_BUT A1
13. #define LEFT_BUT A2
14. // Alien Settings
15. #define NUM_ALIEN_COLUMNS 7
16. #define NUM_ALIEN_ROWS 3
17. #define X_START_OFFSET 6
18. #define SPACE_BETWEEN_ALIEN_COLUMNS 5
19. #define LARGEST_ALIEN_WIDTH 11
20. #define SPACE_BETWEEN_ROWS 9
21. #define INVADERS_DROP_BY 4            // pixel amount that invaders move down by
22. #define INVADERS_SPEED 20             // speed of movement, lower=faster.
23. #define INVADER_HEIGHT 8
24. #define EXPLOSION_GFX_TIME 7  // How long an ExplosionGfx remains on screen before
    disappearing
25. // Mothership
26. #define MOTHERSHIP_HEIGHT 4
```

```
27. #define MOTHERSHIP_WIDTH 16
28. #define MOTHERSHIP_SPEED 2
29. #define MOTHERSHIP_SPAWN_CHANCE 250         //HIGHER IS LESS CHANCE OF SPAWN
30. #define DISPLAY_MOTHERSHIP_BONUS_TIME 20      // how long bonus stays on screen for
     displaying mothership
31. // Player settingsc
32. #define TANKGFX_WIDTH 13
33. #define TANKGFX_HEIGHT 8
34. #define PLAYER_X_MOVE_AMOUNT 1
35. #define PLAYER_Y_START 56
36. #define PLAYER_X_START 0
37. #define MISSILE_HEIGHT 4
38. #define MISSILE_WIDTH 1
39. #define MISSILE_SPEED 1
40. // Status of a game object constants
41. #define ACTIVE 0
42. #define EXPLODING 1
43. #define DESTROYED 2
44. // graphics
45. // aliens
46. const unsigned char InvaderTopGfx [] PROGMEM = {
47.   0x98, 0x5c, 0xb6, 0x5f, 0x5f, 0xb6, 0x5c, 0x98
48. };
49. const unsigned char InvaderTopGfx2 [] PROGMEM = {
50.   0x58, 0xbc, 0x16, 0x1f, 0x1f, 0x16, 0xbc, 0x58
51. };
52. const unsigned char PROGMEM InvaderMiddleGfx [] = {
53.   0x1e, 0xb8, 0x7d, 0x36, 0x3c, 0x3c, 0x3c, 0x36, 0x7d, 0xb8, 0x1e
54. };
55. const unsigned char PROGMEM InvaderMiddleGfx2 [] = {
56.   0x78, 0x18, 0x7d, 0xb6, 0xbc, 0x3c, 0xbc, 0xb6, 0x7d, 0x18, 0x78
57. };
58. const unsigned char PROGMEM InvaderBottomGfx [] = {
59.   0x1c, 0x5e, 0xfe, 0xb6, 0x37, 0x5f, 0x5f, 0x37, 0xb6, 0xfe, 0x5e, 0x1c
60. };
61. const unsigned char PROGMEM InvaderBottomGfx2 [] = {
62.   0x9c, 0xde, 0x7e, 0x36, 0x37, 0x5f, 0x5f, 0x37, 0x36, 0x7e, 0xde, 0x9c
63. };
64. // Player grafix
65. const unsigned char PROGMEM TankGfx [] = {
66.   0xf0, 0xf8, 0xf8, 0xf8, 0xf8, 0xfe, 0xff, 0xfe, 0xf8, 0xf8, 0xf8, 0xf8, 0xf0
67. };
68. static const unsigned char PROGMEM MissileGfx [] = {
69.   0x0f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
70. };
71. static const unsigned char PROGMEM ExplosionGfx [] = {
72.   0x10, 0x92, 0x44, 0x28, 0x81, 0x42, 0x00, 0x42, 0x81, 0x28, 0x44, 0x92, 0x10
73. };
74. const unsigned char MotherShipGfx [] PROGMEM = {
75.   0x04, 0x06, 0x0f, 0x0d, 0x0f, 0x07, 0x05, 0x0f, 0x0f, 0x05, 0x07, 0x0f, 0x0d, 0x0
    f, 0x06, 0x04
76. };
77. // Game structures
78. struct GameObjectStruct  {
79.   // base object which most other objects will include
80.   signed int X;
81.   signed int Y;
82.   unsigned char Status;  //0 active, 1 exploding, 2 destroyed
83. };
84. struct AlienStruct  {
85.   GameObjectStruct Ord;
86.   unsigned char ExplosionGfxCounter; // how long we want the ExplosionGfx to last
87. };
88. struct PlayerStruct  {
89.   GameObjectStruct Ord;
90.   unsigned int Score;
```

```
  unsigned char Lives;
};
//alien global vars
//The array of aliens across the screen
AlienStruct  Alien[NUM_ALIEN_COLUMNS][NUM_ALIEN_ROWS];
AlienStruct MotherShip;
// widths of aliens
// as aliens are the same type per row we do not need to store their graphic width per alien in the structure above
// that would take a byte per alien rather than just three entries here, 1 per row, saving significnt memory
      byte AlienWidth[] = {8, 11, 12}; // top, middle ,bottom widths
      char AlienXMoveAmount = 2;
      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
      // Mothership
      signed char MotherShipSpeed;
      unsigned int MotherShipBonus;
      signed int MotherShipBonusXPos;            // pos to display bonus at
      unsigned char MotherShipBonusCounter;         // how long bonus amount left on screen
      // Player global variables
      PlayerStruct Player;
      GameObjectStruct Missile;
      // game variables
      unsigned int HiScore;

      void setup()
      {
        arduboy.begin();
        arduboy.setFrameRate(60);
        InitAliens(0);
        InitPlayer();
        pinMode(RIGHT_BUT, INPUT_PULLUP);
        pinMode(LEFT_BUT, INPUT_PULLUP);
        pinMode(FIRE_BUT, INPUT_PULLUP);
        arduboy.setTextSize(1);
        arduboy.setTextColor(WHITE);
        EEPROM.get(0, HiScore);
        if (HiScore == 65535) // new unit never written to
        {
          HiScore = 0;
          EEPROM.put(0, HiScore);
        }
      }

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

      void Physics()  {
        AlienControl();
        MotherShipPhysics();
        PlayerControl();
        MissileControl();
        CheckCollisions();
      }

      unsigned char GetScoreForAlien(int RowNumber)
```

```
152.       {
153.         // returns value for killing and alien at the row indicated
154.         switch (RowNumber)
155.         {
156.           case 0: return 30;
157.           case 1: return 20;
158.           case 2: return 10;
159.         }
160.       }
161.
162.       void MotherShipPhysics()  {
163.         if (MotherShip.Ord.Status == ACTIVE)  { // spawned, move it
164.           MotherShip.Ord.X += MotherShipSpeed;
165.           if (MotherShipSpeed > 0) // going left to right , check if off right han
    d side
166.           {
167.             if (MotherShip.Ord.X >= SCREEN_WIDTH)
168.             {
169.               MotherShip.Ord.Status = DESTROYED;
170.             }
171.           }
172.           else    // going right to left , check if off left hand side
173.           {
174.             if (MotherShip.Ord.X + MOTHERSHIP_WIDTH < 0)
175.             {
176.               MotherShip.Ord.Status = DESTROYED;
177.             }
178.           }
179.
180.         }
181.         else  {
182.           // try to spawn mothership
183.           if (random(MOTHERSHIP_SPAWN_CHANCE) == 1)
184.           {
185.             // Spawn a mother ship, starts just off screen at top
186.             MotherShip.Ord.Status = ACTIVE;
187.             // need to set direction
188.             if (random(2) == 1) // values between 0 and 1
189.             {
190.               MotherShip.Ord.X = SCREEN_WIDTH;
191.               MotherShipSpeed = -
    MOTHERSHIP_SPEED;       // if we go in here swaps to right to left
192.             }
193.             else
194.             {
195.               MotherShip.Ord.X = -MOTHERSHIP_WIDTH;
196.               MotherShipSpeed = MOTHERSHIP_SPEED; // set to go left ot right
197.             }
198.           }
199.         }
200.       }
201.
202.       void PlayerControl()  {
203.         // user input checks
204.         if ((digitalRead(RIGHT_BUT) == 0) & (Player.Ord.X + TANKGFX_WIDTH < SCREEN
    _WIDTH))
205.           Player.Ord.X += PLAYER_X_MOVE_AMOUNT;
206.         if ((digitalRead(LEFT_BUT) == 0) & (Player.Ord.X > 0))
207.           Player.Ord.X -= PLAYER_X_MOVE_AMOUNT;
208.         if ((digitalRead(FIRE_BUT) == 0) & (Missile.Status != ACTIVE))
209.         {
210.           Missile.X = Player.Ord.X + (TANKGFX_WIDTH / 2);
211.           Missile.Y = PLAYER_Y_START;
212.           Missile.Status = ACTIVE;
213.         }
214.       }
```

```
215.
216.       void MissileControl()
217.       {
218.         if (Missile.Status == ACTIVE)
219.         {
220.           Missile.Y -= MISSILE_SPEED;
221.           if (Missile.Y + MISSILE_HEIGHT < 0)  // If off top of screen destroy so
    can be used again
222.             Missile.Status = DESTROYED;
223.         }
224.       }
225.
226.       void AlienControl()
227.       {
228.         if ((InvadersMoveCounter--) < 0)
229.         {
230.           bool Dropped = false;
231.           if ((RightMostPos() + AlienXMoveAmount >= SCREEN_WIDTH) | (LeftMostPos()
     + AlienXMoveAmount < 0)) // at edge of screen
232.           {
233.             AlienXMoveAmount = -
    AlienXMoveAmount;             // reverse direction
234.             Dropped = true;                                    // and indicate we a
    re dropping
235.           }
236.           // update the alien postions
237.           for (int Across = 0; Across < NUM_ALIEN_COLUMNS; Across++)
238.           {
239.             for (int Down = 0; Down < 3; Down++)
240.             {
241.               if (Alien[Across][Down].Ord.Status == ACTIVE)
242.               {
243.                 if (Dropped == false)
244.                   Alien[Across][Down].Ord.X += AlienXMoveAmount;
245.                 else
246.                   Alien[Across][Down].Ord.Y += INVADERS_DROP_BY;
247.               }
248.             }
249.           }
250.           InvadersMoveCounter = INVADERS_SPEED;
251.           AnimationFrame = !AnimationFrame; ///swap to other frame
252.         }
253.       }
254.
255.       void CheckCollisions()
256.       {
257.         MissileAndAlienCollisions();
258.         MotherShipCollisions();
259.       }
260.
261.       void MotherShipCollisions()
262.       {
263.         if ((Missile.Status == ACTIVE) & (MotherShip.Ord.Status == ACTIVE))
264.         {
265.           if (Collision(Missile, MISSILE_WIDTH, MISSILE_HEIGHT, MotherShip.Ord, MO
    THERSHIP_WIDTH, MOTHERSHIP_HEIGHT))
266.           {
267.             MotherShip.Ord.Status = EXPLODING;
268.             MotherShip.ExplosionGfxCounter = EXPLOSION_GFX_TIME;
269.             Missile.Status = DESTROYED;
270.             // generate the score for the mothership hit, note in the real arcade
    space invaders the score was not random but
271.             // just appeared so, a player could infulence its value with clever pl
    ay, but we'll keep it a little simpler
272.             MotherShipBonus = random(4); // a random number between 0 and 3
273.             switch (MotherShipBonus)
```

```
274.             {
275.               case 0: MotherShipBonus = 50; break;
276.               case 1: MotherShipBonus = 100; break;
277.               case 2: MotherShipBonus = 150; break;
278.               case 3: MotherShipBonus = 300; break;
279.             }
280.             Player.Score += MotherShipBonus;
281.             MotherShipBonusXPos = MotherShip.Ord.X;
282.             if (MotherShipBonusXPos > 100)                // to ensure isn't half
    off right hand side of screen
283.               MotherShipBonusXPos = 100;
284.             if (MotherShipBonusXPos < 0)                // to ensure isn't half of
    f right hand side of screen
285.               MotherShipBonusXPos = 0;
286.             MotherShipBonusCounter = DISPLAY_MOTHERSHIP_BONUS_TIME;
287.           }
288.         }
289.       }
290.
291.       void MissileAndAlienCollisions()
292.       {
293.         for (int across = 0; across < NUM_ALIEN_COLUMNS; across++)
294.         {
295.           for (int down = 0; down < NUM_ALIEN_ROWS; down++)
296.           {
297.             if (Alien[across][down].Ord.Status == ACTIVE)
298.             {
299.               if (Missile.Status == ACTIVE)
300.               {
301.                 if (Collision(Missile, MISSILE_WIDTH, MISSILE_HEIGHT, Alien[across
    ][down].Ord, AlienWidth[down], INVADER_HEIGHT))
302.                 {
303.                   // missile hit
304.                   Alien[across][down].Ord.Status = EXPLODING;
305.                   Missile.Status = DESTROYED;
306.                   Player.Score += GetScoreForAlien(down);
307.                 }
308.               }
309.             }
310.           }
311.         }
312.       }
313.
314.       bool Collision(GameObjectStruct Obj1, unsigned char Width1, unsigned char He
    ight1, GameObjectStruct Obj2, unsigned char Width2, unsigned char Height2)
315.       {
316.         return ((Obj1.X + Width1 > Obj2.X) & (Obj1.X < Obj2.X + Width2) & (Obj1.Y
    + Height1 > Obj2.Y) & (Obj1.Y < Obj2.Y + Height2));
317.       }
318.
319.       int RightMostPos()  {
320.         //returns x pos of right most alien
321.         int Across = NUM_ALIEN_COLUMNS - 1;
322.         int Down;
323.         int Largest = 0;
324.         int RightPos;
325.         while (Across >= 0) {
326.           Down = 0;
327.           while (Down < NUM_ALIEN_ROWS) {
328.             if (Alien[Across][Down].Ord.Status == ACTIVE)
329.             {
330.               // different aliens have different widths, add to x pos to get right
    pos
331.               RightPos = Alien[Across][Down].Ord.X + AlienWidth[Down];
332.               if (RightPos > Largest)
333.                 Largest = RightPos;
```

```
334.             }
335.             Down++;
336.           }
337.           if (Largest > 0) // we have found largest for this coloum
338.             return Largest;
339.           Across--;
340.         }
341.         return 0;  // should never get this far
342.       }
343.
344.       int LeftMostPos()  {
345.         //returns x pos of left most alien
346.         int Across = 0;
347.         int Down;
348.         int Smallest = SCREEN_WIDTH * 2;
349.         while (Across < NUM_ALIEN_COLUMNS) {
350.           Down = 0;
351.           while (Down < 3) {
352.             if (Alien[Across][Down].Ord.Status == ACTIVE)
353.               if (Alien[Across][Down].Ord.X < Smallest)
354.                 Smallest = Alien[Across][Down].Ord.X;
355.             Down++;
356.           }
357.           if (Smallest < SCREEN_WIDTH * 2) // we have found smalest for this colou
    m
358.             return Smallest;
359.           Across++;
360.         }
361.         return 0;  // should nevr get this far
362.       }
363.
364.       void UpdateDisplay()
365.       {
366.         int i;
367.         arduboy.clear();
368.         // Mothership bonus display if required
369.         if (MotherShipBonusCounter > 0)
370.         {
371.           // mothership bonus
372.           arduboy.setCursor(MotherShipBonusXPos, 0);
373.           arduboy.print(MotherShipBonus);
374.           MotherShipBonusCounter--;
375.         }
376.         else
377.         {
378.           // draw score and lives, anything else can go above them
379.           arduboy.setCursor(0, 0);
380.           arduboy.print(Player.Score);
381.           arduboy.setCursor(SCREEN_WIDTH - 7, 0);
382.           arduboy.print(Player.Lives);
383.         }
384.         //Invaders
385.         for (int across = 0; across < NUM_ALIEN_COLUMNS; across++)
386.         {
387.           for (int down = 0; down < NUM_ALIEN_ROWS; down++)
388.           {
389.             if (Alien[across][down].Ord.Status == ACTIVE) {
390.               switch (down)  {
391.                 case 0:
392.                   if (AnimationFrame)
393.                     arduboy.drawBitmap(Alien[across][down].Ord.X, Alien[across][do
    wn].Ord.Y,  InvaderTopGfx, AlienWidth[down], INVADER_HEIGHT, WHITE);
394.                   else
395.                     arduboy.drawBitmap(Alien[across][down].Ord.X, Alien[across][do
    wn].Ord.Y,  InvaderTopGfx2, AlienWidth[down], INVADER_HEIGHT, WHITE);
396.                   break;
```

```
397.                 case 1:
398.                   if (AnimationFrame)
399.                     arduboy.drawBitmap(Alien[across][down].Ord.X, Alien[across][do
    wn].Ord.Y,  InvaderMiddleGfx, AlienWidth[down], INVADER_HEIGHT, WHITE);
400.                   else
401.                     arduboy.drawBitmap(Alien[across][down].Ord.X, Alien[across][do
    wn].Ord.Y,  InvaderMiddleGfx2, AlienWidth[down], INVADER_HEIGHT, WHITE);
402.                   break;
403.                 default:
404.                   if (AnimationFrame)
405.                     arduboy.drawBitmap(Alien[across][down].Ord.X, Alien[across][do
    wn].Ord.Y,  InvaderBottomGfx, AlienWidth[down], INVADER_HEIGHT, WHITE);
406.                   else
407.                     arduboy.drawBitmap(Alien[across][down].Ord.X, Alien[across][do
    wn].Ord.Y,  InvaderBottomGfx2, AlienWidth[down], INVADER_HEIGHT, WHITE);
408.                 }
409.               }
410.               else  {
411.                 if (Alien[across][down].Ord.Status == EXPLODING) {
412.                   Alien[across][down].ExplosionGfxCounter--;
413.                   if (Alien[across][down].ExplosionGfxCounter > 0)  {
414.                     arduboy.drawBitmap(Alien[across][down].Ord.X, Alien[across][down
    ].Ord.Y,  ExplosionGfx, 13, 8, WHITE);
415.                   }
416.                   else
417.                     Alien[across][down].Ord.Status = DESTROYED;
418.                 }
419.               }
420.             }
421.           }
422.           // player
423.           arduboy.drawBitmap(Player.Ord.X, Player.Ord.Y,  TankGfx, TANKGFX_WIDTH, TA
    NKGFX_HEIGHT, WHITE);
424.           //missile
425.           if (Missile.Status == ACTIVE)
426.             arduboy.drawBitmap(Missile.X, Missile.Y,  MissileGfx, MISSILE_WIDTH, MIS
    SILE_HEIGHT, WHITE);
427.
428.           // mothership (not bonus if hit)
429.           if (MotherShip.Ord.Status == ACTIVE)
430.             arduboy.drawBitmap(MotherShip.Ord.X, MotherShip.Ord.Y,  MotherShipGfx, M
    OTHERSHIP_WIDTH, MOTHERSHIP_HEIGHT, WHITE);
431.           else
432.           {
433.             if (MotherShip.Ord.Status == EXPLODING)
434.             {
435.               for (i = 0; i < MOTHERSHIP_WIDTH; i += 2)  {
436.                 arduboy.drawBitmap(MotherShip.Ord.X + i, MotherShip.Ord.Y,  Explosio
    nGfx, random(4) + 2, MOTHERSHIP_HEIGHT, WHITE);
437.               }
438.               MotherShip.ExplosionGfxCounter--;
439.               if (MotherShip.ExplosionGfxCounter == 0)  {
440.                 MotherShip.Ord.Status = DESTROYED;
441.               }
442.             }
443.           }
444.           arduboy.display();
445.         }
446.
447.         void InitPlayer()  {
448.           Player.Ord.Y = PLAYER_Y_START;
449.           Player.Ord.X = PLAYER_X_START;
450.           Missile.Status = DESTROYED;
451.           Player.Score = 0;
452.         }
453.
```

```
454.       void InitAliens(int YStart)  {
455.         for (int across = 0; across < NUM_ALIEN_COLUMNS; across++)  {
456.           for (int down = 0; down < 3; down++)  {
457.             // we add down to centralise the aliens, just happens to be the right
    value we need per row!
458.             // we need to adjust a little as row zero should be 2, row 1 should be
     1 and bottom row 0
459.             Alien[across][down].Ord.X = X_START_OFFSET + (across * (LARGEST_ALIEN_
    WIDTH + SPACE_BETWEEN_ALIEN_COLUMNS)) - (AlienWidth[down] / 2);
460.             Alien[across][down].Ord.Y = YStart + (down * SPACE_BETWEEN_ROWS);
461.             Alien[across][down].Ord.Status = ACTIVE;
462.             Alien[across][down].ExplosionGfxCounter = EXPLOSION_GFX_TIME;
463.           }
464.         }
465.         MotherShip.Ord.Y = 0;
466.         MotherShip.Ord.X = -MOTHERSHIP_WIDTH;
467.         MotherShip.Ord.Status = DESTROYED;
468.       }
```