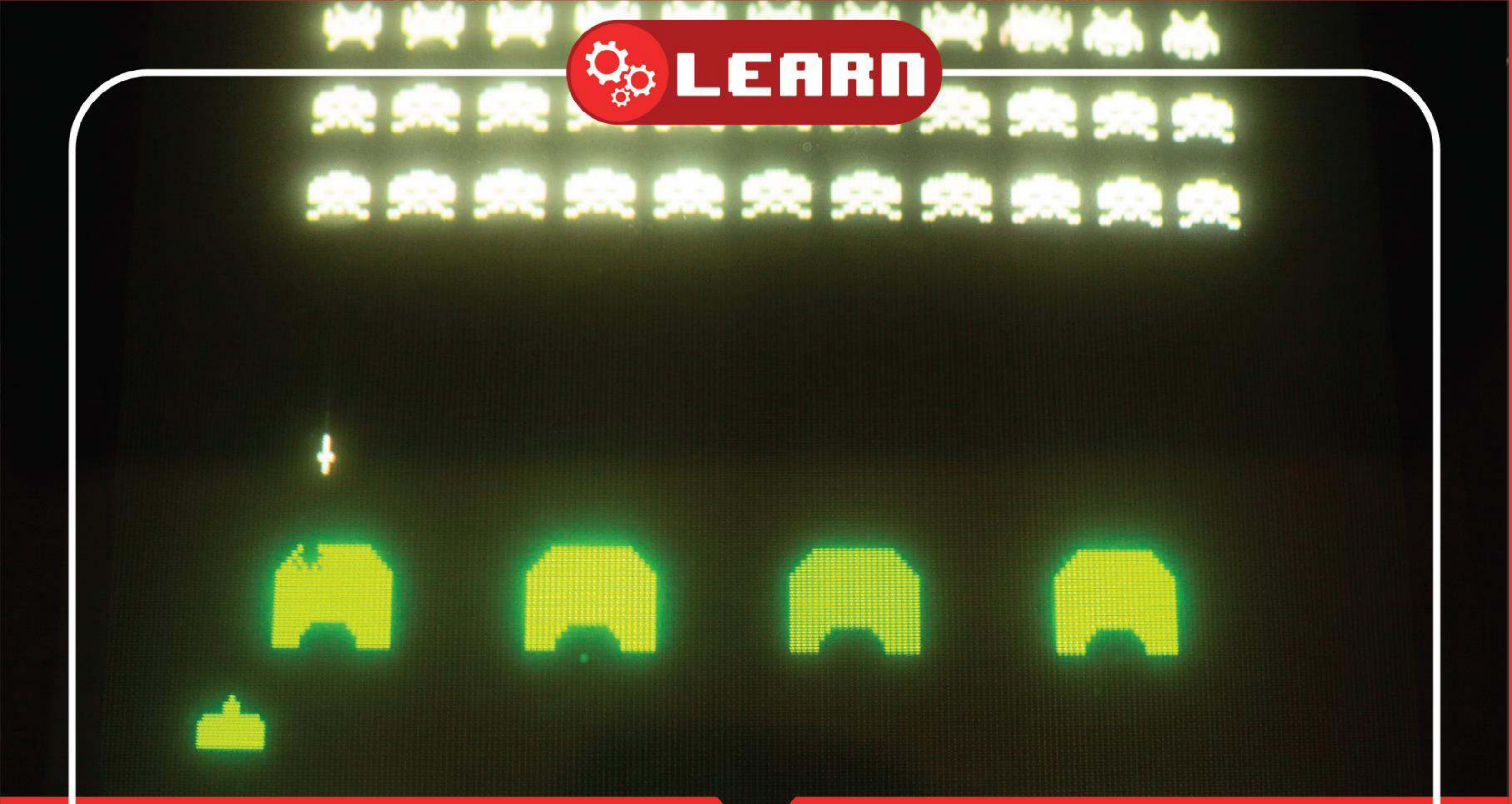
LEARN



CODING SPACE INVADERS

PART 7

With 8BitCADE

Compatible with:

8BitCADE XL

AND

8BitCADE

Contents

Lesson Number:7 3
Step 1 ADDING MOTHERSHIP CONSTANTS 3
Step 2 ADDING MOTHERSHIP GRAPHICS........ 3
Step 3 UPDATE ALIEN GLOBAL VARIABLES 4
Step 4 ADDING **MOTHERSHIP** GLOBAL VARIABLES LINKED TO **ALIENSTRUCT** 4
Step 5 UPDATING **VOID PHYSICS** TO INCLUDE MOTHERSHIP 4
Step 6 ADDING MOTHERSHIP PHYSICS 5
Step 7 UPDATING **CHECK COLLISIONS** TO INCLUDE MOTHERSHIPCOLLISIONS 5
Step 8 ADDING MOTHERSHIP COLLISIONS 6
Step 9 UPDATE THE **UPDATE DISPLAY** ROUTINE 6
Step 10 DISPLAY THE MOTHERSHIP 7
Step 11 INITIALISE THE MOTHERSHIP 7
Final code 7

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:7

Lesson Title: Adding Mothership

Code: Full Code for Lesson

System: Arduboy + Project ABE

Prerequisites to completing this tutorial

1. Tutorials 1, 1a, 1b ,2, 3, 4, 5, 6
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**

In this tutorial we will add the mothership. In this implementation it is random, in the real arcade game it appears approximately every 25 seconds or so. The scoring for it is also random, either 50,100,150 or 300 points. In the real arcade game it appears random but in fact it can be influenced by the player's actions.

Step 1 ADDING MOTHERSHIP CONSTANTS

Type the following code into line 24-29

```
24. // Mothership
25. #define MOTHERSHIP_HEIGHT 4
26. #define MOTHERSHIP_WIDTH 16
27. #define MOTHERSHIP_SPEED 2
28. #define MOTHERSHIP_SPAWN_CHANCE 1000          //HIGHER IS LESS CHANCE OF SPAWN
29. #define DISPLAY_MOTHERSHIP_BONUS_TIME 20       // how long bonus stays on screen for displaying mothership
```

Explanation (line 24-29)
Lines 24-29 define some values for the mothership ship. Lines 25-26 are size constants of the mothership in pixels. Line 27 sets the speed that the ship travels across the screen, the higher the faster. It's basically the number of pixels moved at a time. We then have a value that effects the frequency that the ship appears. The higher the less frequent. Line 29 controls how long the bonus score you got (i.e. 50,100,150,300) stays on screen in the position where the mothership was destroyed. The higher the number the longer it stays on screen.

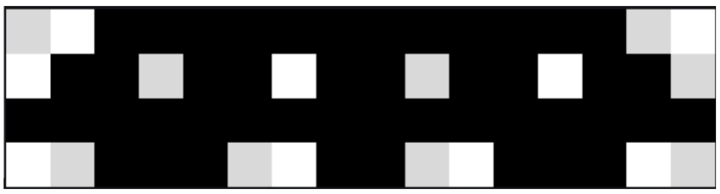
Step 2 ADDING MOTHERSHIP GRAPHICS

Type the following code into line 73-75

```
73. const unsigned char MotherShipGfx [] PROGMEM = {
74.   0x04, 0x06, 0x0f, 0x0d, 0x0f, 0x07, 0x05, 0x0f, 0x0f, 0x05, 0x07, 0x0f, 0x0d, 0x0f, 0x06, 0x04
75. };
```

Explanation (line 73-75)
This section is self-explanatory. Graphics were done using www.pixilart.com and are displayed below. 16 pixels wide and 4 pixels high. Image used to convert to hexadecimal to save memory. Hex is added as an array.

Binary Code
B00111111,B11111100,
B01101101,B10110110,
B11111111,B11111111,
B00111001,B10011100



Step 3 UPDATE ALIEN GLOBAL VARIABLES

Type the following code into line 93

```
90. //alien global vars
91. //The array of aliens across the screen
92. AlienStruct  Alien[NUM_ALIEN_COLUMNS][NUM_ALIEN_ROWS];
93. AlienStruct MotherShip;
```

Explanation (line 93)

Line 93 creates a global variable of type AlienStruct to keep track of the ship.

Step 4 ADDING **MOTHERSHIP** GLOBAL VARIABLES LINKED TO **ALIENSTRUCT**

Type the following code into line 104-109

```
104.        // Mothership
105.        signed char MotherShipSpeed;
106.        unsigned int MotherShipBonus;
107.        signed int MotherShipBonusXPos;            // pos to display bonus at
108.        unsigned char MotherShipBonusCounter;         // how long bonus amount left on screen
109.
```

Explanation (line 104-109)

We also have some other global variables that keep track of various aspects of the mothership. **MotherShipSpeed** has the current speed and direction of the mothership and it takes its value directly from **MOTHERSHIP_SPEED** defined earlier. If its a positive value then it will be move from left to right, if negative then right to left. **MotherShipBonus** stores the bonus earned if you hit the mothership. This is displayed where the mothership was destroyed. **MotherShipBonusCounter** is the counter that starts with the value defined by **DISPLAY_MOTHERSHIP_BONUS_TIME** and it is decremented and on reaching 0 the bonus score is no longer displayed.

Step 5 UPDATING **VOID PHYSICS** TO INCLUDE MOTHERSHIP

Type the following code into line 135

```
130.        void Physics()  {
131.          AlienControl();
132.          PlayerControl();
133.          MissileControl();
134.          CheckCollisions();
135.          MotherShipPhysics();
136.        }
```

Explanation (line 135)

The physics routine has been expanded to include the mothership. It calls **MotherShipPhysics()**, which is below.

Step 6 ADDING MOTHERSHIP PHYSICS

Type the following code into line 252-289

```
252.      void MotherShipPhysics()  {
253.        if (MotherShip.Ord.Status == ACTIVE)  { // spawned, move it
254.          MotherShip.Ord.X += MotherShipSpeed;
255.          if (MotherShipSpeed > 0) // going left to right , check if off right hand side
256.          {
257.            if (MotherShip.Ord.X >= SCREEN_WIDTH)
258.            {
259.              MotherShip.Ord.Status = DESTROYED;
260.            }
261.          }
262.          else    // going right to left , check if off left hand side
263.          {
264.            if (MotherShip.Ord.X + MOTHERSHIP_WIDTH < 0)
265.            {
266.              MotherShip.Ord.Status = DESTROYED;
267.            }
268.          }
269.        }
270.        else  {
271.          // try to spawn mothership
272.          if (random(MOTHERSHIP_SPAWN_CHANCE) == 1)
273.          {
274.            // Spawn a mother ship, starts just off screen at top
275.            MotherShip.Ord.Status = ACTIVE;
276.            // need to set direction
277.            if (random(2) == 1) // values between 0 and 1
278.            {
279.              MotherShip.Ord.X = SCREEN_WIDTH;
280.              MotherShipSpeed = -MOTHERSHIP_SPEED;       // if we go in here swaps to right to left
281.            }
282.            else
283.            {
284.              MotherShip.Ord.X = -MOTHERSHIP_WIDTH;
285.              MotherShipSpeed = MOTHERSHIP_SPEED; // set to go left ot right
286.            }
287.          }
288.        }
289.      }
```

Explanation (line 252-289)

Looking at the code, if the ship is active we handle moving it across the screen and check if it goes off the screen, destroying it (without a score being earned) if it does. If not active then we randomly decide if a ship should appear, if one should we then set a 50/50 chance as to which direction it should travel (setting its position and speed accordingly).

Step 7 UPDATING **CHECK COLLISIONS** TO INCLUDE MOTHERSHIPCOLLISIONS

Type the following code into line 194

```
191.      void CheckCollisions()
192.      {
193.        MissileAndAlienCollisions();
194.        MotherShipCollisions();
195.      }
```

Explanation (line 194)

The check collisions routine at line 333 has also been expanded to call **MotherShipCollisions()**.

Step 8 ADDING MOTHERSHIP COLLISIONS

Type the following code into line 219-245

```
219.   void MotherShipCollisions()
220.   {
221.     if ((Missile.Status == ACTIVE) & (MotherShip.Ord.Status == ACTIVE))
222.     {
223.       if (Collision(Missile, MISSILE_WIDTH, MISSILE_HEIGHT, MotherShip.Ord, MOTHERSHIP_WIDTH, MOTHERSHIP_HEIGHT))
224.       {
225.         MotherShip.Ord.Status = EXPLODING;
226.         MotherShip.ExplosionGfxCounter = EXPLOSION_GFX_TIME;
227.         Missile.Status = DESTROYED;
228.         // generate the score for the mothership hit, note in the real arcade space invaders the score was not random but
229.         // just appeared so, a player could infulence its value with clever play, but we'll keep it a little simpler
230.         MotherShipBonus = random(4); // a random number between 0 and 3
231.         switch (MotherShipBonus)
232.         {
233.           case 0: MotherShipBonus = 50; break;
234.           case 1: MotherShipBonus = 100; break;
235.           case 2: MotherShipBonus = 150; break;
236.           case 3: MotherShipBonus = 300; break;
237.         }
238.         MotherShipBonusXPos = MotherShip.Ord.X;
239.         if (MotherShipBonusXPos > 100)             // to ensure isn't half off right hand side of screen
240.           MotherShipBonusXPos = 100;
241.         if (MotherShipBonusXPos < 0)             // to ensure isn't half off right hand side of screen
242.           MotherShipBonusXPos = 0;
243.         MotherShipBonusCounter = DISPLAY_MOTHERSHIP_BONUS_TIME;
244.       }
245.     }
```

Explanation (line 219-245)
In this routine if the players missile is active and the mothership is active then we check for a collision between the two. If we get a collision then we choose a random bonus and set the position that this will be shown on screen and set the counter for the duration that its shown on screen.

Step 9 UPDATE THE **UPDATE DISPLAY** ROUTINE

Type the following code into line 23

```
333.     void UpdateDisplay()
334.     {
335.       int i;
336.       arduboy.clear();
337.       // Mothership bonus display if required
338.       if (MotherShipBonusCounter > 0)
339.       {
340.         // mothership bonus
341.         arduboy.setCursor(MotherShipBonusXPos, 0);
342.         arduboy.print(MotherShipBonus);
343.         MotherShipBonusCounter--;
344.       }
```

Explanation (line 23)
The **UpdateDisplay()** routine now also displays any bonus earned as long as the bonus display counter is above 0. Lines 333 to 344 handle this.

Step 10 DISPLAY THE MOTHERSHIP

Type the following code into line 387-404

```
387.          // mothership (not bonus if hit)
388.          if (MotherShip.Ord.Status == ACTIVE)
389.            arduboy.drawBitmap(MotherShip.Ord.X, MotherShip.Ord.Y,  MotherShipGfx, MOTHERSHIP_WIDTH, MOTHERSHIP_HEIGHT, WHITE);
390.          else
391.          {
392.            if (MotherShip.Ord.Status == EXPLODING)
393.            {
394.              for (i = 0; i < MOTHERSHIP_WIDTH; i += 2)  {
395.                arduboy.drawBitmap(MotherShip.Ord.X + i, MotherShip.Ord.Y,  ExplosionGfx, random(4) + 2, MOTHERSHIP_HEIGHT, WHITE);
396.              }
397.              MotherShip.ExplosionGfxCounter--;
398.              if (MotherShip.ExplosionGfxCounter == 0)  {
399.                MotherShip.Ord.Status = DESTROYED;
400.              }
401.            }
402.          } |
403.          arduboy.display();
404.        }
```

Explanation (line 387-404)

If Active we just display it else we check if its currently exploding, if so we draw the explosion graphic at random widths where the ship was for the amount of time that explosions are supposed to appear on screen (discussed in an earlier article).

Step 11 INITIALISE THE MOTHERSHIP

Type the following code into line 423-426

```
423.          MotherShip.Ord.Y = 0;
424.          MotherShip.Ord.X = -MOTHERSHIP_WIDTH;
425.          MotherShip.Ord.Status = DESTROYED;
426.        }
```

Explanation (line 423-426)

Lines 423-426 initialise the mothership to ensure sensible values

Final code

```
/*   www.pixilart.com for sprite generation
  http://javl.github.io/image2cpp/ for image conversion
*/
#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
// Alien Settings
#define NUM_ALIEN_COLUMNS 7
#define NUM_ALIEN_ROWS 3
#define X_START_OFFSET 6
#define SPACE_BETWEEN_ALIEN_COLUMNS 5
#define LARGEST_ALIEN_WIDTH 11
#define SPACE_BETWEEN_ROWS 9
#define INVADERS_DROP_BY 4            // pixel amount that invaders move down by
#define INVADERS_SPEED 20             // speed of movement, lower=faster.
#define INVADER_HEIGHT 8
#define EXPLOSION_GFX_TIME 7  // How long an ExplosionGfx remains on screen before disappearing
// Mothership
#define MOTHERSHIP_HEIGHT 4
```

```
#define MOTHERSHIP_WIDTH 16
#define MOTHERSHIP_SPEED 2
#define MOTHERSHIP_SPAWN_CHANCE 1000          //HIGHER IS LESS CHANCE OF SPAWN
#define DISPLAY_MOTHERSHIP_BONUS_TIME 20      // how long bonus stays on screen for
 displaying mothership
// Player settingsc
#define TANKGFX_WIDTH 13
#define TANKGFX_HEIGHT 8
#define PLAYER_X_MOVE_AMOUNT 1
#define PLAYER_Y_START 56
#define PLAYER_X_START 0
#define MISSILE_HEIGHT 4
#define MISSILE_WIDTH 1
#define MISSILE_SPEED 1
// Status of a game object constants
#define ACTIVE 0
#define EXPLODING 1
#define DESTROYED 2
// graphics
// aliens
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
};
static const unsigned char PROGMEM MissileGfx [] = {
  0x0f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
};
static const unsigned char PROGMEM ExplosionGfx [] = {
  0x10, 0x92, 0x44, 0x28, 0x81, 0x42, 0x00, 0x42, 0x81, 0x28, 0x44, 0x92, 0x10
};
const unsigned char MotherShipGfx [] PROGMEM = {
  0x04, 0x06, 0x0f, 0x0d, 0x0f, 0x07, 0x05, 0x0f, 0x0f, 0x05, 0x07, 0x0f, 0x0d, 0x0
f, 0x06, 0x04
};
// Game structures
struct GameObjectStruct  {
  // base object which most other objects will include
  signed int X;
  signed int Y;
  unsigned char Status;  //0 active, 1 exploding, 2 destroyed
};
struct AlienStruct  {
  GameObjectStruct Ord;
  unsigned char ExplosionGfxCounter; // how long we want the ExplosionGfx to last
};
struct PlayerStruct  {
  GameObjectStruct Ord;
};
```

```
//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 = 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;
GameObjectStruct Missile;
// 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

void setup()
{
  arduboy.begin();
  arduboy.setFrameRate(25);
  InitAliens(0);
  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();
  MissileControl();
  CheckCollisions();
  MotherShipPhysics();
}

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;
  if ((digitalRead(FIRE_BUT) == 0) & (Missile.Status != ACTIVE))
  {
    Missile.X = Player.Ord.X + (6); // offset missile so its in the mideel of the tank
    Missile.Y = PLAYER_Y_START;
    Missile.Status = ACTIVE;
```

```
      }
    }

    void MissileControl()
    {
      if (Missile.Status == ACTIVE)
      {
        Missile.Y -= MISSILE_SPEED;
        if (Missile.Y + MISSILE_HEIGHT < 0)  // If off top of screen destroy so can be used again
          Missile.Status = DESTROYED;
      }
    }

    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
      }
    }

    void CheckCollisions()
    {
      MissileAndAlienCollisions();
      MotherShipCollisions();
    }

    void MissileAndAlienCollisions()
    {
      for (int across = 0; across < NUM_ALIEN_COLUMNS; across++)
      {
        for (int down = 0; down < NUM_ALIEN_ROWS; down++)
        {
          if (Alien[across][down].Ord.Status == ACTIVE)
          {
            if (Missile.Status == ACTIVE)
            {
              if (Collision(Missile, MISSILE_WIDTH, MISSILE_HEIGHT, Alien[across][down].Ord, AlienWidth[down], INVADER_HEIGHT))
              {
                // missile hit
                Alien[across][down].Ord.Status = EXPLODING;
                Missile.Status = DESTROYED;
              }
            }
          }
        }
      }
    }

    void MotherShipCollisions()
    {
      if ((Missile.Status == ACTIVE) & (MotherShip.Ord.Status == ACTIVE))
      {
        if (Collision(Missile, MISSILE_WIDTH, MISSILE_HEIGHT, MotherShip.Ord, MOTHERSHIP_WIDTH, MOTHERSHIP_HEIGHT))
        {
          MotherShip.Ord.Status = EXPLODING;
          MotherShip.ExplosionGfxCounter = EXPLOSION_GFX_TIME;
          Missile.Status = DESTROYED;
          // generate the score for the mothership hit, note in the real arcade space invaders the score was not random but
          // just appeared so, a player could infulence its value with clever play, but we'll keep it a little simpler
          MotherShipBonus = random(4); // a random number between 0 and 3
          switch (MotherShipBonus)
          {
            case 0: MotherShipBonus = 50; break;
            case 1: MotherShipBonus = 100; break;
            case 2: MotherShipBonus = 150; break;
            case 3: MotherShipBonus = 300; break;
          }
          MotherShipBonusXPos = MotherShip.Ord.X;
          if (MotherShipBonusXPos > 100)                // to ensure isn't half off right hand side of screen
            MotherShipBonusXPos = 100;
          if (MotherShipBonusXPos < 0)                // to ensure isn't half off right hand side of screen
            MotherShipBonusXPos = 0;
          MotherShipBonusCounter = DISPLAY_MOTHERSHIP_BONUS_TIME;
        }
      }
    }
    bool Collision(GameObjectStruct Obj1, unsigned char Width1, unsigned char Height1, GameObjectStruct Obj2, unsigned char Width2, unsigned char Height2)
    {
      return ((Obj1.X + Width1 > Obj2.X) & (Obj1.X < Obj2.X + Width2) & (Obj1.Y + Height1 > Obj2.Y) & (Obj1.Y < Obj2.Y + Height2));
    }

    void MotherShipPhysics()  {
      if (MotherShip.Ord.Status == ACTIVE)  { // spawned, move it
        MotherShip.Ord.X += MotherShipSpeed;
        if (MotherShipSpeed > 0) // going left to right , check if off right hand side
        {
          if (MotherShip.Ord.X >= SCREEN_WIDTH)
          {
            MotherShip.Ord.Status = DESTROYED;
          }
        }
        else    // going right to left , check if off left hand side
        {
          if (MotherShip.Ord.X + MOTHERSHIP_WIDTH < 0)
          {
            MotherShip.Ord.Status = DESTROYED;
          }
```

```
        }
      }
      else  {
        // try to spawn mothership
        if (random(MOTHERSHIP_SPAWN_CHANCE) == 1)
        {
          // Spawn a mother ship, starts just off screen at top
          MotherShip.Ord.Status = ACTIVE;
          // need to set direction
          if (random(2) == 1) // values between 0 and 1
          {
            MotherShip.Ord.X = SCREEN_WIDTH;
            MotherShipSpeed = -MOTHERSHIP_SPEED;        // if we go in here swaps to right to left
          }
          else
          {
            MotherShip.Ord.X = -MOTHERSHIP_WIDTH;
            MotherShipSpeed = MOTHERSHIP_SPEED; // set to go left ot right
          }
        }
      }
    }
    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 nevr get this far
    }
    void UpdateDisplay()
    {
      int i;
      arduboy.clear();
      // Mothership bonus display if required
      if (MotherShipBonusCounter > 0)
      {
        // mothership bonus
        arduboy.setCursor(MotherShipBonusXPos, 0);
        arduboy.print(MotherShipBonus);
        MotherShipBonusCounter--;
      }
      for (int across = 0; across < NUM_ALIEN_COLUMNS; across++)
      {
        for (int down = 0; down < NUM_ALIEN_ROWS; down++)
        {
          if (Alien[across][down].Ord.Status == ACTIVE) {
            switch (down)  {
              case 0:
                if (AnimationFrame)
                  arduboy.drawBitmap(Alien[across][down].Ord.X, Alien[across][down].Ord.Y,  InvaderTopGfx, AlienWidth[down], INVADER_HEIGHT, WHITE);
                else
                  arduboy.drawBitmap(Alien[across][down].Ord.X, Alien[across][down].Ord.Y,  InvaderTopGfx2, AlienWidth[down], INVADER_HEIGHT, WHITE);
                break;
              case 1:
                if (AnimationFrame)
                  arduboy.drawBitmap(Alien[across][down].Ord.X, Alien[across][down].Ord.Y,  InvaderMiddleGfx, AlienWidth[down], INVADER_HEIGHT, WHITE);
                else
                  arduboy.drawBitmap(Alien[across][down].Ord.X, Alien[across][down].Ord.Y,  InvaderMiddleGfx2, AlienWidth[down], INVADER_HEIGHT, WHITE);
                break;
              default:
                if (AnimationFrame)
                  arduboy.drawBitmap(Alien[across][down].Ord.X, Alien[across][down].Ord.Y,  InvaderBottomGfx, AlienWidth[down], INVADER_HEIGHT, WHITE);
                else
                  arduboy.drawBitmap(Alien[across][down].Ord.X, Alien[across][down].Ord.Y,  InvaderBottomGfx2, AlienWidth[down], INVADER_HEIGHT, WHITE);
            }
          }
          else  {
            if (Alien[across][down].Ord.Status == EXPLODING) {
              Alien[across][down].ExplosionGfxCounter--;
              if (Alien[across][down].ExplosionGfxCounter > 0)  {
                arduboy.drawBitmap(Alien[across][down].Ord.X, Alien[across][down].Ord.Y, ExplosionGfx, 13, 8, WHITE);
              }
              else
                Alien[across][down].Ord.Status = DESTROYED;
            }
          }
        }
      }
      // player
      arduboy.drawBitmap(Player.Ord.X, Player.Ord.Y,  TankGfx, TANKGFX_WIDTH, TANKGFX_HEIGHT, WHITE);
      //missile
      if (Missile.Status == ACTIVE)
        arduboy.drawBitmap(Missile.X, Missile.Y,  MissileGfx, MISSILE_WIDTH, MISSILE_HEIGHT, WHITE);
      // mothership (not bonus if hit)
      if (MotherShip.Ord.Status == ACTIVE)
        arduboy.drawBitmap(MotherShip.Ord.X, MotherShip.Ord.Y,  MotherShipGfx, MOTHERSHIP_WIDTH, MOTHERSHIP_HEIGHT, WHITE);
      else
      {
        if (MotherShip.Ord.Status == EXPLODING)
        {
          for (i = 0; i < MOTHERSHIP_WIDTH; i += 2)  {
            arduboy.drawBitmap(MotherShip.Ord.X + i, MotherShip.Ord.Y,  ExplosionGfx, random(4) + 2, MOTHERSHIP_HEIGHT, WHITE);
          }
          MotherShip.ExplosionGfxCounter--;
          if (MotherShip.ExplosionGfxCounter == 0)  {
            MotherShip.Ord.Status = DESTROYED;
          }
        }
      }
      arduboy.display();
    }

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

    void InitAliens(int YStart)  {
      for (int across = 0; across < NUM_ALIEN_COLUMNS; across++)  {
        for (int down = 0; down < 3; down++)  {
          // we add down to centralise the aliens, just happens to be the right value we need per row!
          // we need to adjust a little as row zero should be 2, row 1 should be 1 and bottom row 0
          Alien[across][down].Ord.X = X_START_OFFSET + (across * (LARGEST_ALIEN_WIDTH + SPACE_BETWEEN_ALIEN_COLUMNS)) - (AlienWidth[down] / 2);
          Alien[across][down].Ord.Y = YStart + (down * SPACE_BETWEEN_ROWS);
          Alien[across][down].Ord.Status = ACTIVE;
          Alien[across][down].ExplosionGfxCounter = EXPLOSION_GFX_TIME;
        }
      }
      MotherShip.Ord.Y = 0;
      MotherShip.Ord.X = -MOTHERSHIP_WIDTH;
      MotherShip.Ord.Status = DESTROYED;
    }
```