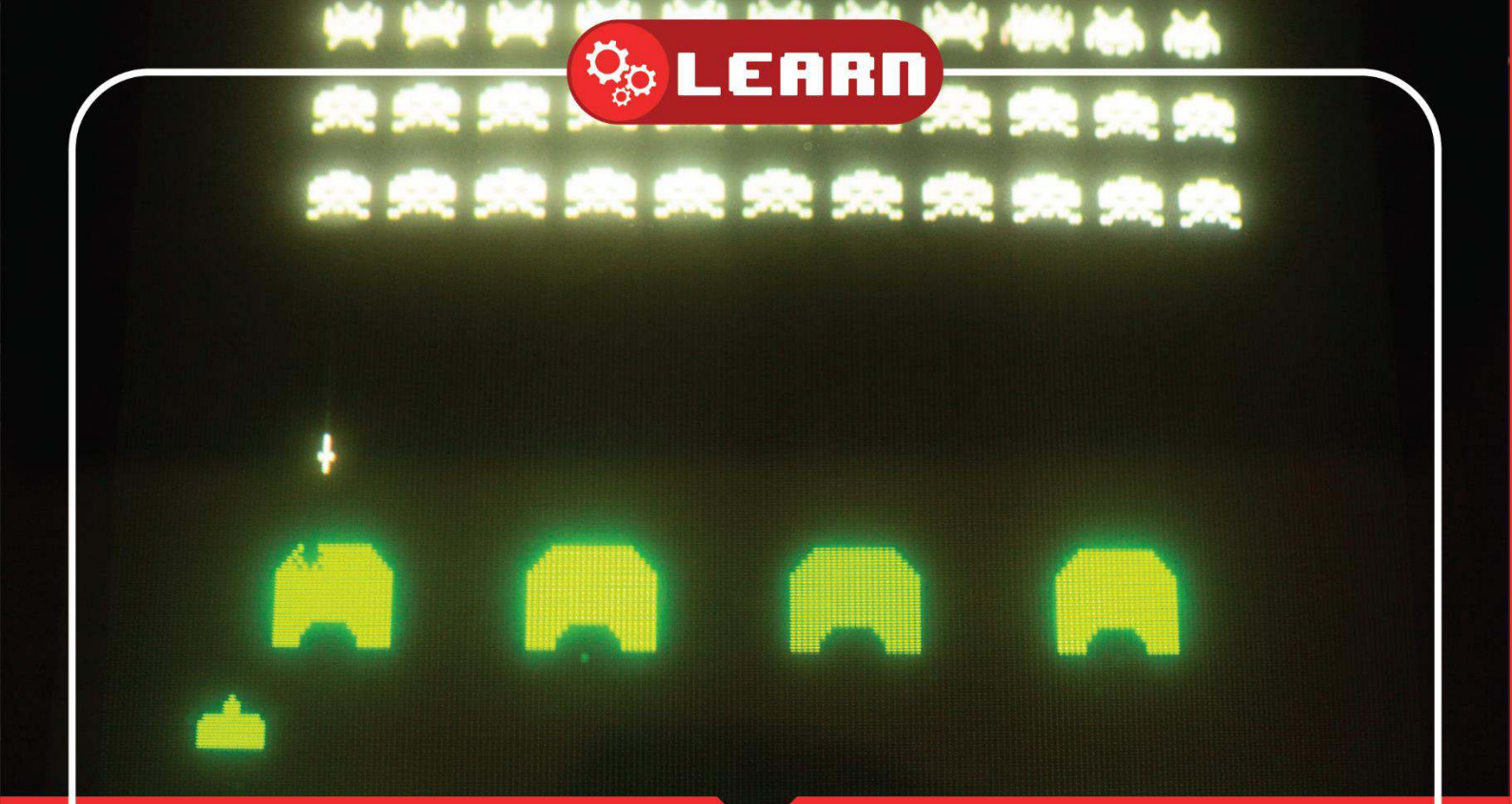
LEARN



CODING SPACE INVADERS PART 9

With 8BitCADE

Compatible with:

8BitCADE XL

AND

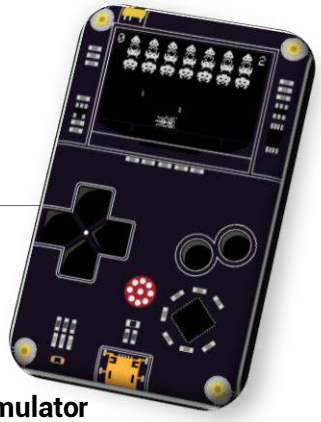
8BitCADE

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 Title: Lives, Attract, Difficulty

Code: Full Code for Lesson

System: Arduboy + Project ABE

Prerequisites to completing this tutorial

1. Tutorials 1, 1a, 1b ,2, 3, 4, 5, 6, 7, 8
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'll add Lives, Attract and Difficulty.

Additions in this tutorial:

- You can now die if an Invader's bomb hits you (but not by any other means).
- Lives will be reduced and if it gets to zero then it's game over
- An attract screen will be shown (splash or start screen)
- Congratulations screen for beating high score will be available
- The high score can be reset
- Level difficulty based on invader speed and start position, i.e. as you kills invaders, and they reduce in number, they move faster. The more levels you complete the lower the starting point of the invaders.

Did you know?
As in the arcade game the invaders speed increases as less of them are on screen. Famously this was not an intended game mechanic (i.e. not in any intentional design) but was a by-product of the speed of the available processor at the time. When there were a lot of Invaders to move on screen it took time to plot them to screen. But as their numbers declined the processor didn't have to plot as many so wasn't working quite so hard, meaning it could get round to moving and plotting those that were left much sooner than normal – hence they moved faster. This proved an excellent difficulty factor during any one level. We have very fast processors (compared to the late 70's early 80's) and as such we have to deliberately code speeding up.

Step 1 UPDATING CONSTANTS

Type the following code into line 36-43

```
36. #define INVADERS_Y_START MOTHERSHIP_HEIGHT-1
37. #define AMOUNT_TO_DROP_BY_PER_LEVEL 4   //NEW How much farther down aliens start per new level
38. #define LEVEL_TO_RESET_TO_START_HEIGHT 4  // EVERY MULTIPLE OF THIS LEVEL THE ALIEN y START POSITION WILL RESET TO TOP
39. #define ALIEN_X_MOVE_AMOUNT 1                       //number of pixels moved at start of wave
40. #define CHANCEOFBOMBDROPPING 20                     // Higher the number the rarer the bomb drop,
41. #define BOMB_HEIGHT 4
42. #define BOMB_WIDTH 2
43. #define MAXBOMBS 3   // Max number of bombs allowed to drop at a time
44.
```

Explanation (line 36-43)
These additional constant are used in the code later in the programme. An additional code has been added related to the Ystart position for the mothership in height. Line 37-38 are related to how the start position of the aliens will change as the player progresses through the waves or levels. Line 39 is the x movement at the beginning of a wave, this will be used later in the code to speed up movement as the number of aliens reduce, making the game more challenging. Line 40-43 is related to alien bomb dropping, its frequency and number. Line 41 and 42 are

constants used for the size of the graphic for the alien bomb. Note, constants are used as the value does not change, does not need to be stored in memory (Syntax #define constantName value).

Step 2 ADDING PLAYER CONSTANTS FOR LIVES

Type the following code into line 49-50

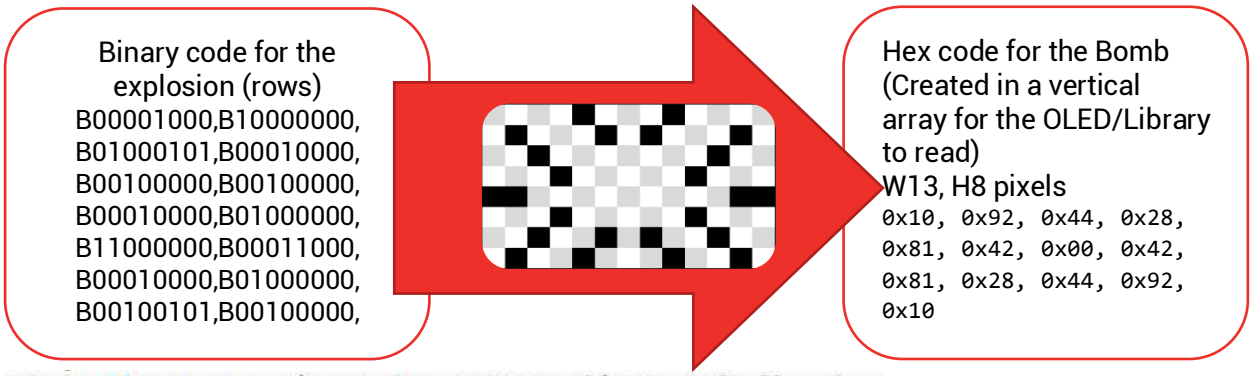
```
45. // Player settingsc
46. #define TANKGFX_WIDTH 13
47. #define TANKGFX_HEIGHT 8
48. #define PLAYER_X_MOVE_AMOUNT 1
49. #define LIVES 3                 //NEW
50. #define PLAYER_EXPLOSION_TIME 10  // How long an ExplosionGfx remains on screen before dissapearing
```

Explanation (line 49-50)
Constants related to the Player, for example, line 48 is tank movement amount in pixels per frame. Line 49 is the number of lives of the player.

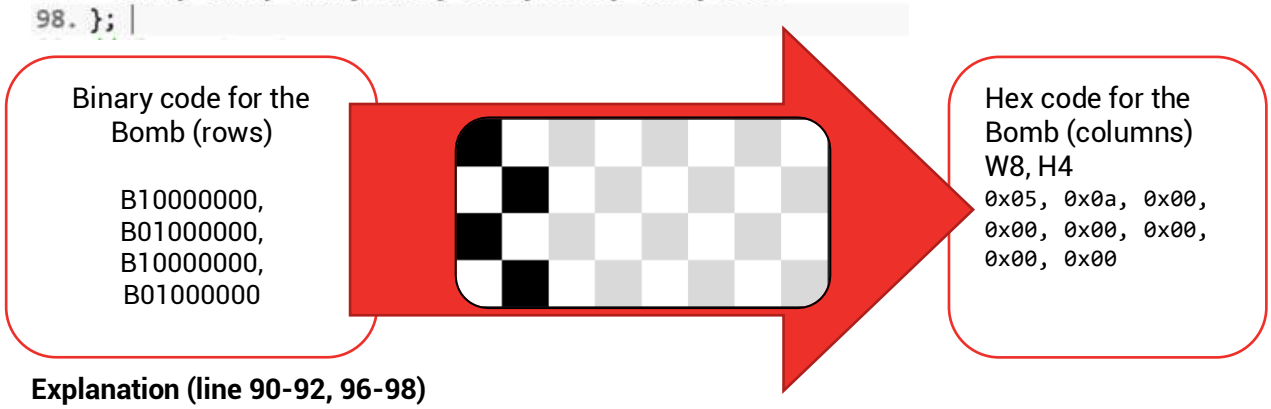
Step 3 ADDING ALIEN BOMB & EXPLOSION GRAPHICS (STORING THE GRAPHIC IMAGES IN PROGRAMME MEMORY)

Type the following code into line 90-92, 96-98

```
90. static const unsigned char PROGMEM ExplosionGfx [] = {
91.   0x10, 0x92, 0x44, 0x28, 0x81, 0x42, 0x00, 0x42, 0x81, 0x28, 0x44, 0x92, 0x10
92. };
```




```
96. static const unsigned char PROGMEM AlienBombGfx [] = {
97.   0x05, 0x0a, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
98. };
```



Explanation (line 90-92, 96-98)

The line: **static const unsigned char PROGMEM AlienBombGfx** sets up the start of the graphics definition for the Alien Bomb. **Unsigned char** is an unsigned data type that occupies

one byte of memory. The unsigned char data type encodes numbers from 0 to 255. The final graphics code is in hexadecimal. Below is the code in binary, to illustrate the bomb pattern, 1's being where the pixels are displayed in one bit array. As always, graphics are drawn using www.pixilart.com and converted using image to cpp converter. Links are at the very beginning of the full code. You can see from the illustration that binary is written in rows while the hex is written in columns. This is just because the image to cpp converter can create the array of the image vertically is need by the OLED, and its related library being used. This is all explained in Tutorial 1.

Step 4 UPDATE PLAYER STRUCT

Type the following code into line 116-121

```
112.      struct PlayerStruct  {
113.        GameObjectStruct Ord;
114.        unsigned int Score;
115.        unsigned char Lives;
116.        //NEW
117.        unsigned char Level;
118.        unsigned char AliensDestroyed;    // count of how many killed so far
119.        unsigned char AlienSpeed;         // higher the number slower they go, calculated when ever alien destroyed
120.        unsigned char ExplosionGfxCounter; // how long we want the ExplosionGfx to last
121.      };
```

Explanation (line 116-121)

As introduced in Tutorial 2, Struct enables us to group variables together. Struct has a variable name and members (other variables/data). Line 117 to 120 are additional variables related to new code which has been introduced, **Level** (we need to know what level we are on), **number of aliens destroyed**, **alien speed** (helps make the game more challenging) and **explosion time**. These variables are grouped under **struct PlayerStruct. Unsigned char** is an unsigned data type that occupies one byte of memory. The unsigned char data type encodes numbers from 0 to 255.

Step 5 UPDATING ALIEN GLOBAL VARIABLES

Type the following code into line 127-129

```
123.      //alien global vars
124.      //The array of aliens across the screen
125.      AlienStruct  Alien[NUM_ALIEN_COLUMNS][NUM_ALIEN_ROWS];
126.      AlienStruct MotherShip;
127.      GameObjectStruct AlienBomb[MAXBOMBS];
128.
129.      static const int TOTAL_ALIENS = NUM_ALIEN_COLUMNS * NUM_ALIEN_ROWS; //NEW
```

Explanation (line 127-129)

Struct enables us to group variables together. Variable can be either used within a function or block of code, (called local variables), inside the definition of a function (in the parenthesis), called formal parameters, or outside of all functions, these are called global variables. The above global variables can be used throughout the programme and can be accessed by any function. These variables have been grouped into Alien and **Gameobject Structs**. **Line 129** calculates the total aliens available so that this can be used later to monitor destroyed aliens, which will effect alien movement speed etc.

Step 6 UPDATING GAME VARIABLES

Type the following code into line 151-152

```
149.        // game variables
150.        unsigned int HiScore;
151.        bool GameInPlay = false;                    //NEW
152.
```

Explanation (line 151-152)

An additional bool **GameInPlay** variable has been introduced to know when the game is in play (true) or when it has ended (false). A **Boolean** (bool for short) holds one of two values, true or false. Each Boolean variable occupies one byte of memory. This variable is later used in void loop to direct game play.

Step 7 UPDATING VOID LOOP

Type the following code into line 176-213

```
176.      void loop()
177.      {
178.        if (!arduboy.nextFrame()) {
179.          return;
180.        }
181.          //NEW
182.          if (GameInPlay)
183.          {
184.            Physics();
185.            UpdateDisplay();
186.          }
187.          else
188.            AttractScreen();
189.        }
190.
191.        void AttractScreen()
192.        {
193.          arduboy.clear();
194.          CentreText("Play", 0);
195.          CentreText("Space Invaders", 12);
196.          CentreText("Press Fire to start", 24);
197.          CentreText("Hi Score     ", 36);
198.          arduboy.setCursor(80, 36);
199.          arduboy.print(HiScore);
200.          arduboy.display();
201.          if (digitalRead(FIRE_BUT) == 0) {
202.            GameInPlay = true;
203.            NewGame();
204.          }
205.          // CHECK FOR HIGH SCORE RESET, PLAYER MUST HOLD LEFT AND RIGHT TOGETHER
206.          if ((digitalRead(LEFT_BUT) == 0) & (digitalRead(RIGHT_BUT) == 0))
207.          {
208.            // Reset high score, don't need to worry about debounce as will ony write to EEPROM if changed, so writes a 0 then won't write again
209.            // if hiscore still 0 which it will be
210.            HiScore = 0;
211.            EEPROM.put(0, HiScore);
212.          }
213.        }
```

Explanation (line 176-213)

Void loop is the main repeated function in the programme. Initially if the **GameInPlay** global variable returns **true**, then the code is directed towards **Physics** and **UpdateDisplay** (code to execute game play), otherwise if **GameInPlay** returns **false**, then the code is directed to the **AttractScreen**. Once directed to the attract screen, the attract screen is displayed using code Line 191-200. Centre text is a function, later shown in Line 737-741 which is executed to centre any displayed text and provides its Y position on the screen. Line 201 is an If statement related to if the start button (Fire) is pressed to start the game, making **GameInPlay = True**, triggering a new game. The rest of the final code is related to resetting the high score.

Step 8 UPDATING VOID PHYSICS

Type the following code into line 216

```
215.        void Physics()  {
216.          if (Player.Ord.Status == ACTIVE) {
217.            AlienControl();
218.            MotherShipPhysics();
219.            PlayerControl();
220.            MissileControl();
221.            CheckCollisions();
222.          }
223.        }
```

Explanation (line 216)

This section of the code has been modified so that the code is only executed if the player status is **active** (not destroyed and run out of lives), otherwise the code should revert back to Attract Screen in the void loop.

Step 9 UPDATE ALIEN CONTROL

Type the following code into line 326 - 338

```
323.            InvadersMoveCounter = Player.AlienSpeed;
324.            AnimationFrame = !AnimationFrame; ///swap to other frame
325.          }
326.          // should the alien drop a bomb
327.          if (random(CHANCEOFBOMBDROPPING) == 1)
328.            DropBomb();
329.          MoveBombs();
330.        }
331.
332.        void MoveBombs() {
333.          for (int i = 0; i < MAXBOMBS; i++) {
334.            if (AlienBomb[i].Status == ACTIVE)
335.              AlienBomb[i].Y += 2;
336.          }
337.        }
338.
```

Explanation (line 326-338)

Line 326-338 starts with an If statement linked to 2 functions, DropBomb & MoveBombs. Bombs must be random so if a bomb is dropped, the alien bomb status becomes active.

Step 10 ADD VOID DROP BOMB

Type the following code into line 339-392

```
339.      void DropBomb()  {
340.        // if there is a free bomb slot then drop a bomb else nothing happens
341.        bool Free = false;
342.        unsigned char ActiveCols[NUM_ALIEN_COLUMNS];
343.        unsigned char BombIdx = 0;
344.        // find a free bomb slot
345.        while ((Free == false) & (BombIdx < MAXBOMBS)) {
346.          if (AlienBomb[BombIdx].Status == DESTROYED)
347.            Free = true;
348.          else
349.            BombIdx++;
350.        }
351.        if (Free)  {
352.          unsigned char Columns = 0;
353.          // now pick and alien at random to drop the bomb
354.          // we first pick a column, obviously some columns may not exist, so we count number of remaining cols
355.          // first, this adds time but then also picking columns randomly that don't exist may take time also
356.          unsigned char ActiveColCount = 0;
357.          signed char Row;
358.          unsigned char ChosenColumn;
359.
360.          while (Columns < NUM_ALIEN_COLUMNS) {
361.            Row = 2;
362.            while (Row >= 0)  {
363.              if (Alien[Columns][Row].Ord.Status == ACTIVE)
364.              {
365.                ActiveCols[ActiveColCount] = Columns;
366.                ActiveColCount++;
367.                break;
368.              }
369.              Row--;
370.            }
371.            Columns++;
372.          }
373.          // we have ActiveCols array filled with the column numbers of the active cols and we have how many
374.          // in ActiveColCount, now choose a column at random
375.          ChosenColumn = random(ActiveColCount); // random number between 0 and the amount of columns
376.          // we now find the first available alien in this column to drop the bomb from
377.          Row = 2;
378.          while (Row >= 0)  {
379.            if (Alien[ActiveCols[ChosenColumn]][Row].Ord.Status == ACTIVE)  {
380.              // Set the bomb from this alien
381.              AlienBomb[BombIdx].Status = ACTIVE;
382.              AlienBomb[BombIdx].X = Alien[ActiveCols[ChosenColumn]][Row].Ord.X + int(AlienWidth[Row] / 2);
383.              // above sets bomb to drop around invaders centre, here we add a litrle randomness around this pos
384.              AlienBomb[BombIdx].X = (AlienBomb[BombIdx].X - 2) + random(0, 4);
385.              AlienBomb[BombIdx].Y = Alien[ActiveCols[ChosenColumn]][Row].Ord.Y + 4;
386.              break;
387.            }
388.            Row--;
389.          }
390.        }
391.      }
392.
```

Explanation (line 339-392)

This is a long section of code but essentially we need to know how many aliens are left, are there any available bomb slots left (aliens have limited number of bombs they can fire, MAXBOMBS) then we pick a random column and then a row with an alien in it and fire a bomb.

Step 11 ADDING VOID BOMB COLLISIONS

Type the following code into line 393-424

```
393.        void BombCollisions()
394.        {
395.          //check bombs collisions
396.          for (int i = 0; i < MAXBOMBS; i++)
397.          {
398.            if (AlienBomb[i].Status == ACTIVE)
399.            {
400.              if (AlienBomb[i].Y > 64)          // gone off bottom of screen
401.                AlienBomb[i].Status = DESTROYED;
402.              else
403.              {
404.                // HAS IT HIT PLAYERS missile
405.                if (Collision(AlienBomb[i], BOMB_WIDTH, BOMB_HEIGHT, Missile, MISSILE_WIDTH, MISSILE_HEIGHT))
406.                {
407.                  // destroy missile and bomb
408.                  AlienBomb[i].Status = EXPLODING;
409.                  Missile.Status = DESTROYED;
410.                }
411.                else
412.                {
413.                  // has it hit players ship
414.                  if (Collision(AlienBomb[i], BOMB_WIDTH, BOMB_HEIGHT, Player.Ord, TANKGFX_WIDTH, TANKGFX_HEIGHT))
415.                  {
416.                    PlayerHit();
417.                    AlienBomb[i].Status = DESTROYED;
418.                  }
419.                }
420.              }
421.            }
422.          }
423.        }
424.
```

Explanation (line 393-424)

Now the bomb has been set in motion we need to check for collisions with the player missiles. The if statement clearly shows that if the player missiles hit the alien missiles that they will then be destroyed.

Step 12 ADDING VOID PLAYER HIT

Type the following code into line 425-429

```
425.        void PlayerHit()  {
426.          Player.Ord.Status = EXPLODING;
427.          Player.ExplosionGfxCounter = PLAYER_EXPLOSION_TIME;
428.          Missile.Status = DESTROYED;
429.        }
```

Explanation (line 425-429)

This function and its variable status relate to the player Tank or Missile being hit. It updates the player status to exploding (needed for the explodingGFX) and changes the status of a missile that the player fires to destroyed.

Step 13 UPDATING CHECK COLLISIONS

Type the following code into line 435

```
431.        void CheckCollisions()
432.        {
433.          MissileAndAlienCollisions();
434.          MotherShipCollisions();
435.          BombCollisions();
436.        }
```

Explanation (line 435)

Updated the check collisions function to include Bomb Collisions.

Step 14 ADJUSTING **MISSILES AND ALIEN COLLISIONS** TO SPEED UP ALIEN MOVEMENT

Type the following code into line 468-520

```
468.  void MissileAndAlienCollisions()
469.  {
470.    for (int across = 0; across < NUM_ALIEN_COLUMNS; across++)
471.    {
472.      for (int down = 0; down < NUM_ALIEN_ROWS; down++)
473.      {
474.        if (Alien[across][down].Ord.Status == ACTIVE)
475.        {
476.          if (Missile.Status == ACTIVE)
477.          {
478.            if (Collision(Missile, MISSILE_WIDTH, MISSILE_HEIGHT, Alien[across][down].Ord, AlienWidth[down], INVADER_HEIGHT))
479.            {
480.              // missile hit
481.              Alien[across][down].Ord.Status = EXPLODING;
482.              Missile.Status = DESTROYED;
483.              Player.Score += GetScoreForAlien(down);
484.              Player.AliensDestroyed++;
485.              // calc new speed of aliens, note (float) must be before TOTAL_ALIENS to force float calc else
486.              // you will get an incorrect result
487.              Player.AlienSpeed = ((1 - (Player.AliensDestroyed / (float)TOTAL_ALIENS)) * INVADERS_SPEED);
488.              // for very last alien make to  fast!
489.              if (Player.AliensDestroyed == TOTAL_ALIENS - 2)
490.                if (AlienXMoveAmount > 0)
491.                  AlienXMoveAmount = ALIEN_X_MOVE_AMOUNT * 2;
492.                else
493.                  AlienXMoveAmount = -(ALIEN_X_MOVE_AMOUNT * 2);
494.              // for very last alien make to super fast!
495.              if (Player.AliensDestroyed == TOTAL_ALIENS - 1)
496.                if (AlienXMoveAmount > 0)
497.                  AlienXMoveAmount = ALIEN_X_MOVE_AMOUNT * 4;
498.                else
499.                  AlienXMoveAmount = -(ALIEN_X_MOVE_AMOUNT * 4);
500.
501.              if (Player.AliensDestroyed == TOTAL_ALIENS)
502.                NextLevel(&Player);
503.            }
504.          }
505.          if (Alien[across][down].Ord.Status == ACTIVE)   // double check still active afer above code
506.          {
507.            // check if this alien is in contact with TankGfx
508.            if (Collision(Player.Ord, TANKGFX_WIDTH, TANKGFX_HEIGHT, Alien[across][down].Ord, AlienWidth[down], ALIEN_HEIGHT))
509.              PlayerHit();
510.            else
511.            {
512.              // check if alien is below bottom of screen
513.              if (Alien[across][down].Ord.Y + 8 > SCREEN_HEIGHT)
514.                PlayerHit();
515.            }
516.          }
517.        }
518.      }
519.    }
520.  }
```

Explanation (line 468-520)

The code that handles this is in the routine **MissileAndAlienCollisions**. Its in this routine as this is where an Invader would be destroyed and thus they would be slightly less. We have a general speed calculation for most of the game (line 487) but then some specific code where we get to the last couple where we increase the speed dramatically.

Level Difficulty

In addition (an intended designed in game mechanic) was that for every level the Invaders would start a little lower down, up until a certain level (perhaps level 4 I cannot remember at time of writing). Then the start height position would revert to the level 1 screen increasing as the levels progressed again until another four levels had passed etc. etc.

Step 15 UPDATE VOID UPDATE DISPLAY

Type the following code into line 595-606

```
595.          //BOMBS
596.          // draw bombs next as aliens have priority of overlapping them
597.          for (i = 0; i < MAXBOMBS; i++)  {
598.            if (AlienBomb[i].Status == ACTIVE)
599.              arduboy.drawBitmap(AlienBomb[i].X, AlienBomb[i].Y,  AlienBombGfx, 2, 4, WHITE);
600.            else  {// must be destroyed
601.              if (AlienBomb[i].Status == EXPLODING)
602.                arduboy.drawBitmap(AlienBomb[i].X - 4, AlienBomb[i].Y,  ExplosionGfx, 4, 8, WHITE);
603.              // Ensure on next draw that ExplosionGfx disappears
604.              AlienBomb[i].Status = DESTROYED;
605.            }
606.          }
```

Explanation (line 595-606)

This code actually draws the bomb graphics to the OLED screen, only while they have the status ACTIVE. If destroyed by the player missile, the alien graphic explodes and disappears.

Step 16 UPDATE PLAYER GRAPHICS

Type the following code into line 650-661

```
647.          // player
648.          if (Player.Ord.Status == ACTIVE)
649.            arduboy.drawBitmap(Player.Ord.X, Player.Ord.Y,  TankGfx, TANKGFX_WIDTH, TANKGFX_HEIGHT, WHITE);
650.          else {
651.            if (Player.Ord.Status == EXPLODING)  {
652.              for (i = 0; i < TANKGFX_WIDTH; i += 2)  {
653.                arduboy.drawBitmap(Player.Ord.X + i, Player.Ord.Y,  ExplosionGfx, random(4) + 2, 8, WHITE);
654.              }
655.              Player.ExplosionGfxCounter--;
656.              if (Player.ExplosionGfxCounter == 0)  {
657.                Player.Ord.Status = DESTROYED;
658.                LoseLife();
659.              }
660.            }
661.          }
```

Explanation (line 650-661)

The routine updates the player graphics based on its current status. The if statement switches between ACTIVE to EXPLODING to DESTROYED and takes away a life each time this occurs (LoseLife).

Step 17 ADD VOID LOSE LIFE

Type the following code into line 24-29

```
685.        void LoseLife()  {
686.          Player.Lives--;
687.          if (Player.Lives > 0)  {
688.            DisplayPlayerAndLives(&Player);
689.            // clear alien missiles
690.            for (int i = 0; i < MAXBOMBS; i++)  {
691.              AlienBomb[i].Status = DESTROYED;
692.              AlienBomb[i].Y = 0;
693.            }
694.            // ENABLE PLAYER
695.            Player.Ord.Status = ACTIVE;
696.            Player.Ord.X = 0;
697.          }
698.          else  {
699.            GameOver();
700.          }
701.        }
702.
```

Explanation (line 24-29)

This is the **LoseLife** function which is linked to the players status and reduces the number of lives of the player every time the player status = destroyed. **Line 687**, once the number of lives = 0 game over!

Step 18 INCLUDE GAME OVER

Type the following code into line 24-29

```
703.        void GameOver()  {
704.          GameInPlay = false;
705.          arduboy.clear();
706.          CentreText("Player 1", 0);
707.          CentreText("Game Over", 12);
708.          CentreText("Score ", 24);
709.          arduboy.print(Player.Score);
710.          if (Player.Score > HiScore)
711.          {
712.            CentreText("NEW HIGH SCORE!!!", 36);
713.            CentreText("**CONGRATULATIONS**", 48);
714.          }
715.          arduboy.display();
716.          if (Player.Score > HiScore) {
717.            HiScore = Player.Score;
718.            EEPROM.put(0, HiScore);
719.          }
720.          delay(2500);
721.        }
```

Explanation (line 24-29)

The game over function, presents the players score and if higher than the recorded high score, presents congratulations (line 712,713). Line 716- 718 write the new hi score to memory using EEPROM.put (0, HiScore), syntax EEPROM.put(address, data). The parameters, address and data: the address to write to which can start at 0, data can be a variable of custom struct.

Step 19 DISPLY PLAYER DATA EVERY NEW LEVEL OF GAMEPLAY

Type the following code into line 723-736

```
723.        void DisplayPlayerAndLives(PlayerStruct * Player)  {
724.          arduboy.clear();
725.          CentreText("Player 1", 0);
726.          CentreText("Score ", 12);
727.          arduboy.print(Player->Score);
728.          CentreText("Lives ", 24);
729.          arduboy.print(Player->Lives);
730.          CentreText("Level ", 36);
731.          arduboy.print(Player->Level);
732.          arduboy.display();
733.          delay(2000);
734.          Player->Ord.X = PLAYER_X_START;
735.        }
736.
```

Explanation (line 723-736)

Once a level or wave has been completed we need to display the current score, how many lives are left and the level.

Step 20 INCLUDE CENTRE TEXT FUNCTION

Type the following code into line 737-741

```
737.        void CentreText(const char *Text, unsigned char Y)  {
738.          // centres text on screen
739.          arduboy.setCursor(int((float)(SCREEN_WIDTH) / 2 - ((strlen(Text) * 6) / 2)), Y);
740.          arduboy.print(Text);
741.        }
```

Explanation (line 737-741)

This function centres the text for display and includes a local variable Y for the Y axis position of the text on the OLED screen.

Step 21 UPDATE INITIAL PLAYER STATUS

Type the following code into line 743-751

```
743.        void InitPlayer()  {
744.          Player.Ord.Y = PLAYER_Y_START;
745.          Player.Ord.X = PLAYER_X_START;
746.          Player.Ord.Status = ACTIVE;
747.          Player.Lives = LIVES;
748.          Player.Level = 0;
749.          Missile.Status = DESTROYED;
750.          Player.Score = 0;
751.        }
```

Explanation (line 743-750)

This code is the initial player status, with its accompanying variables which include X&Y start positions, status as ACTIVE and DESTROYED and the number of lives.

Step 22 INCLUDE VOID NEXT LEVEL

Type the following code into line 753-770

```
753.        void NextLevel(PlayerStruct * Player)  {
754.          // reset any dropping bombs
755.          int YStart;
756.          for (int i = 0; i < MAXBOMBS; i++)
757.            AlienBomb[i].Status = DESTROYED;
758.          AnimationFrame = false;
759.          Player->Level++;
760.          YStart = ((Player->Level - 1) % LEVEL_TO_RESET_TO_START_HEIGHT) * AMOUNT_TO_DROP_BY_PER_LEVEL;
761.          InitAliens(YStart);
762.          AlienXMoveAmount = ALIEN_X_MOVE_AMOUNT;
763.          Player->AlienSpeed = INVADERS_SPEED;
764.          Player->AliensDestroyed = 0;
765.          MotherShip.Ord.X = -MOTHERSHIP_WIDTH;
766.          MotherShip.Ord.Status = DESTROYED;
767.          Missile.Status = DESTROYED;
768.          randomSeed(100);
769.          DisplayPlayerAndLives(Player);
770.        }
```

Explanation (line 753-770) Looks complex perhaps but all it's doing is setting the Invaders Y start position per level. The Amount to drop per level is defined in the constant **AMOUNT_TO_DROP_BY_PER_LEVEL**. 4 pixels seems about right for this screen. In essence all we need to do is multiply the level number by this amount to get the Y position. However that would make the game technically impossible for higher level numbers as it could result with the Invaders starting on top of the players tank! So (as it did in the original) we do this on a repeating cycle of levels. i.e. levels 1-4 they come down more and more then they reset back to the start for level 5 getting lower and lower again etc. etc. The part that accomplishes this is this:

((Player->Level-1) % LEVEL_TO_RESET_TO_START_HEIGHT+1)

The constant **LEVEL_TO_RESET_TO_START_HEIGHT** contains the level that we reset the Y start position back to the top. In this case I chose level 5. So the Invaders get lower and lower for levels 1 to 4 and start again at the top for level 5 etc. But how is this achieved? Well it's all in the **% (percentage)**. This is the **modulus** operator. It's purpose is to return the remainder of an integer division. i.e. of I divide 4 by 3, then 3 would go in once and 1 would be left over, so 4 % 3 would give me **1** as the answer. So in our code we divide the current level by 4 and then multiply this remainder by the amount to move down per level to get the Y Position for the Invaders. Look at this table of examples.

Player Level	Player level -1	LEVEL_TO_RESET_TO_START_HEIGHT	Remainder of : (Player level-1) % LEVEL_TO_RESET_TO_START_HEIGHT	Multiply remainder by AMOUNT_TO_DROP_BY_PER_LEVEL to get Y Position
1	0	4	0	0
2	1	4	1	4
3	2	4	2	8
4	3	4	3	12
5	4	4	0	0
6	5	4	1	4

Player Level	Player level -1	LEVEL_TO_RESET_TO_START_HEIGHT	Remainder of : (Player level-1) % LEVEL_TO_RESET_TO_START_HEIGHT	Multiply remainder by AMOUNT_TO_DROP_BY_PER_LEVEL to get Y Position
7	6	4	2	8

Resetting the high score
If you wish to reset the high score it is implemented by pressing both the left and right buttons at once when on the attract screen. If you look back at previous articles and code it was originally intended to have it's own button but at the last minute I've decided to reduce the button count and do it this way instead.

Step 23 INCLUDE VOID NEW GAME

Type the following code into line 772-775

```
772.          void NewGame() {
773.            InitPlayer();
774.            NextLevel(&Player);
775.          }
```

Explanation (line 772-775)

When a new game begins, this could be a brand new game or the next level. This function links these two variable groups.

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.
8.  // DISPLAY SETTINGS
9.  #define SCREEN_WIDTH 128
10. #define SCREEN_HEIGHT 64
11.
12. // Input settings
13. #define FIRE_BUT 7
14. #define RIGHT_BUT A1
15. #define LEFT_BUT A2
16.
17. // Mothership
18. #define MOTHERSHIP_HEIGHT 4
19. #define MOTHERSHIP_WIDTH 16
20. #define MOTHERSHIP_SPEED 1
21. #define MOTHERSHIP_SPAWN_CHANCE 1000          //HIGHER IS LESS CHANCE OF SPAWN
22. #define DISPLAY_MOTHERSHIP_BONUS_TIME 20       // how long bonus stays on screen for
     displaying mothership
23.
24. // Alien Settings
25. #define ALIEN_HEIGHT 8
26. #define NUM_ALIEN_COLUMNS 7
27. #define NUM_ALIEN_ROWS 3
28. #define X_START_OFFSET 6
```

```
29. #define SPACE_BETWEEN_ALIEN_COLUMNS 5
30. #define LARGEST_ALIEN_WIDTH 11
31. #define SPACE_BETWEEN_ROWS 9
32. #define INVADERS_DROP_BY 4            // pixel amount that invaders move down by
33. #define INVADERS_SPEED 20             // speed of movement, lower=faster.
34. #define INVADER_HEIGHT 8
35. #define EXPLOSION_GFX_TIME 7  // How long an ExplosionGfx remains on screen before
    dissapearing
36. #define INVADERS_Y_START MOTHERSHIP_HEIGHT-1
37. #define AMOUNT_TO_DROP_BY_PER_LEVEL 4   //NEW How much farther down aliens start pe
    r new level
38. #define LEVEL_TO_RESET_TO_START_HEIGHT 4  // EVERY MULTIPLE OF THIS LEVEL THE ALIEN
     y START POSITION WILL RESET TO TOP
39. #define ALIEN_X_MOVE_AMOUNT 1                       //number of pixels moved at sta
    rt of wave
40. #define CHANCEOFBOMBDROPPING 20                     // Higher the number the rarer
    the bomb drop,
41. #define BOMB_HEIGHT 4
42. #define BOMB_WIDTH 2
43. #define MAXBOMBS 3   // Max number of bombs allowed to drop at a time
44.
45. // Player settingsc
46. #define TANKGFX_WIDTH 13
47. #define TANKGFX_HEIGHT 8
48. #define PLAYER_X_MOVE_AMOUNT 1
49. #define LIVES 3                  //NEW
50. #define PLAYER_EXPLOSION_TIME 10  // How long an ExplosionGfx remains on screen bef
    ore dissapearing
51. #define PLAYER_Y_START 56
52. #define PLAYER_X_START 0
53.
54. #define MISSILE_HEIGHT 4
55. #define MISSILE_WIDTH 1
56. #define MISSILE_SPEED 4
57.
58. // Status of a game object constants
59. #define ACTIVE 0
60. #define EXPLODING 1
61. #define DESTROYED 2
62.
63. // graphics
64. // aliens
65. const unsigned char InvaderTopGfx [] PROGMEM = {
66.   0x98, 0x5c, 0xb6, 0x5f, 0x5f, 0xb6, 0x5c, 0x98
67. };
68. const unsigned char InvaderTopGfx2 [] PROGMEM = {
69.   0x58, 0xbc, 0x16, 0x1f, 0x1f, 0x16, 0xbc, 0x58
70. };
71. const unsigned char PROGMEM InvaderMiddleGfx [] = {
72.   0x1e, 0xb8, 0x7d, 0x36, 0x3c, 0x3c, 0x3c, 0x36, 0x7d, 0xb8, 0x1e
73. };
74. const unsigned char PROGMEM InvaderMiddleGfx2 [] = {
75.   0x78, 0x18, 0x7d, 0xb6, 0xbc, 0x3c, 0xbc, 0xb6, 0x7d, 0x18, 0x78
76. };
77. const unsigned char PROGMEM InvaderBottomGfx [] = {
78.   0x1c, 0x5e, 0xfe, 0xb6, 0x37, 0x5f, 0x5f, 0x37, 0xb6, 0xfe, 0x5e, 0x1c
79. };
80. const unsigned char PROGMEM InvaderBottomGfx2 [] = {
81.   0x9c, 0xde, 0x7e, 0x36, 0x37, 0x5f, 0x5f, 0x37, 0x36, 0x7e, 0xde, 0x9c
82. };
83. // Player grafix
84. const unsigned char PROGMEM TankGfx [] = {
85.   0xf0, 0xf8, 0xf8, 0xf8, 0xf8, 0xfe, 0xff, 0xfe, 0xf8, 0xf8, 0xf8, 0xf8, 0xf0
86. };
87. static const unsigned char PROGMEM MissileGfx [] = {
88.   0x0f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
```

```
89. };
90. static const unsigned char PROGMEM ExplosionGfx [] = {
91.   0x10, 0x92, 0x44, 0x28, 0x81, 0x42, 0x00, 0x42, 0x81, 0x28, 0x44, 0x92, 0x10
92. };
93. const unsigned char MotherShipGfx [] PROGMEM = {
94.   0x04, 0x06, 0x0f, 0x0d, 0x0f, 0x07, 0x05, 0x0f, 0x0f, 0x05, 0x07, 0x0f, 0x0d, 0x0
    f, 0x06, 0x04
95. };
96. static const unsigned char PROGMEM AlienBombGfx [] = {
97.   0x05, 0x0a, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
98. };
99. // Game structures
100.        struct GameObjectStruct  {
101.          // base object which most other objects will include
102.          signed int X;
103.          signed int Y;
104.          unsigned char Status;  //0 active, 1 exploding, 2 destroyed
105.        };
106.
107.        struct AlienStruct  {
108.          GameObjectStruct Ord;
109.          unsigned char ExplosionGfxCounter; // how long we want the ExplosionGfx to
     last
110.        };
111.
112.        struct PlayerStruct  {
113.          GameObjectStruct Ord;
114.          unsigned int Score;
115.          unsigned char Lives;
116.          //NEW
117.          unsigned char Level;
118.          unsigned char AliensDestroyed;    // count of how many killed so far
119.          unsigned char AlienSpeed;         // higher the number slower they go, cal
    culated when ever alien destroyed
120.          unsigned char ExplosionGfxCounter; // how long we want the ExplosionGfx to
     last
121.        };
122.
123.        //alien global vars
124.        //The array of aliens across the screen
125.        AlienStruct  Alien[NUM_ALIEN_COLUMNS][NUM_ALIEN_ROWS];
126.        AlienStruct MotherShip;
127.        GameObjectStruct AlienBomb[MAXBOMBS];
128.
129.        static const int TOTAL_ALIENS = NUM_ALIEN_COLUMNS * NUM_ALIEN_ROWS; //NEW
130.
131.        // widths of aliens
132.        // as aliens are the same type per row we do not need to store their graphic
     width per alien in the structure above
133.        // that would take a byte per alien rather than just three entries here, 1 p
    er row, saving significnt memory
134.        byte AlienWidth[] = {8, 11, 12}; // top, middle ,bottom widths
135.        char AlienXMoveAmount = 1;
136.        signed char InvadersMoveCounter;            // counts down, when 0 move inva
    ders, set according to how many aliens on screen
137.        bool AnimationFrame = false; // two frames of animation, if true show one if
     false show the other
138.
139.        // Mothership
140.        signed char MotherShipSpeed;
141.        unsigned int MotherShipBonus;
142.        signed int MotherShipBonusXPos;           // pos to display bonus at
143.        unsigned char MotherShipBonusCounter;         // how long bonus amount left
    on screen
144.
145.        // Player global variables
```

```
146.       PlayerStruct Player;
147.       GameObjectStruct Missile;
148.
149.       // game variables
150.       unsigned int HiScore;
151.       bool GameInPlay = false;                    //NEW
152.
153.       void setup()
154.       {
155.         arduboy.begin();
156.        arduboy.setFrameRate(25);
157.
158.         InitAliens(0);
159.         InitPlayer();
160.
161.         pinMode(RIGHT_BUT, INPUT_PULLUP);
162.         pinMode(LEFT_BUT, INPUT_PULLUP);
163.         pinMode(FIRE_BUT, INPUT_PULLUP);
164.
165.         arduboy.setTextSize(1);
166.         arduboy.setTextColor(WHITE);
167.
168.         EEPROM.get(0, HiScore);
169.         if (HiScore == 65535) // new unit never written to
170.         {
171.           HiScore = 0;
172.           EEPROM.put(0, HiScore);
173.         }
174.       }
175.
176.       void loop()
177.       {
178.         if (!arduboy.nextFrame()) {
179.           return;
180.         }
181.           //NEW
182.           if (GameInPlay)
183.           {
184.             Physics();
185.             UpdateDisplay();
186.           }
187.           else
188.             AttractScreen();
189.         }
190.
191.         void AttractScreen()
192.         {
193.           arduboy.clear();
194.           CentreText("Play", 0);
195.           CentreText("Space Invaders", 12);
196.           CentreText("Press Fire to start", 24);
197.           CentreText("Hi Score     ", 36);
198.           arduboy.setCursor(80, 36);
199.           arduboy.print(HiScore);
200.           arduboy.display();
201.           if (digitalRead(FIRE_BUT) == 0) {
202.             GameInPlay = true;
203.             NewGame();
204.           }
205.           // CHECK FOR HIGH SCORE RESET, PLAYER MUST HOLD LEFT AND RIGHT TOGETHER

206.           if ((digitalRead(LEFT_BUT) == 0) & (digitalRead(RIGHT_BUT) == 0))
207.           {
208.             // Reset high score, don't need to worry about debounce as will ony wr
    ite to EEPROM if changed, so writes a 0 then won't write again
209.             // if hiscore still 0 which it will be
```

```
            HiScore = 0;
            EEPROM.put(0, HiScore);
          }
        }

        void Physics()  {
          if (Player.Ord.Status == ACTIVE) {
            AlienControl();
            MotherShipPhysics();
            PlayerControl();
            MissileControl();
            CheckCollisions();
          }
        }

        unsigned char GetScoreForAlien(int RowNumber)
        {
          // returns value for killing and alien at the row indicated
          switch (RowNumber)
          {
            case 0: return 30;
            case 1: return 20;
            case 2: return 10;
          }
        }

        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 h
  and 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
              }
            }
          }
        }
```

```
274.
275.          void PlayerControl()  {
276.            // user input checks
277.            if ((digitalRead(RIGHT_BUT) == 0) & (Player.Ord.X + TANKGFX_WIDTH < SCRE
    EN_WIDTH))
278.              Player.Ord.X += PLAYER_X_MOVE_AMOUNT;
279.            if ((digitalRead(LEFT_BUT) == 0) & (Player.Ord.X > 0))
280.              Player.Ord.X -= PLAYER_X_MOVE_AMOUNT;
281.            if ((digitalRead(FIRE_BUT) == 0) & (Missile.Status != ACTIVE))
282.            {
283.              Missile.X = Player.Ord.X + (TANKGFX_WIDTH / 2);
284.              Missile.Y = PLAYER_Y_START;
285.              Missile.Status = ACTIVE;
286.            }
287.          }
288.
289.          void MissileControl()
290.          {
291.            if (Missile.Status == ACTIVE)
292.            {
293.              Missile.Y -= MISSILE_SPEED;
294.              if (Missile.Y + MISSILE_HEIGHT < 0)  // If off top of screen destroy s
    o can be used again
295.                Missile.Status = DESTROYED;
296.            }
297.          }
298.
299.          void AlienControl()
300.          {
301.            if ((InvadersMoveCounter--) < 0)
302.            {
303.              bool Dropped = false;
304.              if ((RightMostPos() + AlienXMoveAmount >= SCREEN_WIDTH) | (LeftMostPos
    () + AlienXMoveAmount < 0)) // at edge of screen
305.              {
306.                AlienXMoveAmount = -
    AlienXMoveAmount;             // reverse direction
307.                Dropped = true;                                    // and indicate we
     are dropping
308.              }
309.              // update the alien positions
310.              for (int Across = 0; Across < NUM_ALIEN_COLUMNS; Across++)
311.              {
312.                for (int Down = 0; Down < 3; Down++)
313.                {
314.                  if (Alien[Across][Down].Ord.Status == ACTIVE)
315.                  {
316.                    if (Dropped == false)
317.                      Alien[Across][Down].Ord.X += AlienXMoveAmount;
318.                    else
319.                      Alien[Across][Down].Ord.Y += INVADERS_DROP_BY;
320.                  }
321.                }
322.              }
323.              InvadersMoveCounter = Player.AlienSpeed;
324.              AnimationFrame = !AnimationFrame; ///swap to other frame
325.            }
326.            // should the alien drop a bomb
327.            if (random(CHANCEOFBOMBDROPPING) == 1)
328.              DropBomb();
329.            MoveBombs();
330.          }
331.
332.          void MoveBombs() {
333.            for (int i = 0; i < MAXBOMBS; i++) {
334.              if (AlienBomb[i].Status == ACTIVE)
```

```
335.               AlienBomb[i].Y += 2;
336.             }
337.           }
338.
339.           void DropBomb()  {
340.             // if there is a free bomb slot then drop a bomb else nothing happens
341.             bool Free = false;
342.             unsigned char ActiveCols[NUM_ALIEN_COLUMNS];
343.             unsigned char BombIdx = 0;
344.             // find a free bomb slot
345.             while ((Free == false) & (BombIdx < MAXBOMBS)) {
346.               if (AlienBomb[BombIdx].Status == DESTROYED)
347.                 Free = true;
348.               else
349.                 BombIdx++;
350.             }
351.             if (Free)  {
352.               unsigned char Columns = 0;
353.               // now pick and alien at random to drop the bomb
354.               // we first pick a column, obviously some columns may not exist, so we
     count number of remaining cols
355.               // first, this adds time but then also picking columns randomly that d
    on't exist may take time also
356.               unsigned char ActiveColCount = 0;
357.               signed char Row;
358.               unsigned char ChosenColumn;
359.
360.               while (Columns < NUM_ALIEN_COLUMNS) {
361.                 Row = 2;
362.                 while (Row >= 0)  {
363.                   if (Alien[Columns][Row].Ord.Status == ACTIVE)
364.                   {
365.                     ActiveCols[ActiveColCount] = Columns;
366.                     ActiveColCount++;
367.                     break;
368.                   }
369.                   Row--;
370.                 }
371.                 Columns++;
372.               }
373.               // we have ActiveCols array filled with the column numbers of the acti
    ve cols and we have how many
374.               // in ActiveColCount, now choose a column at random
375.               ChosenColumn = random(ActiveColCount); // random number between 0 and
    the amount of columns
376.               // we now find the first available alien in this column to drop the bo
    mb from
377.               Row = 2;
378.               while (Row >= 0)  {
379.                 if (Alien[ActiveCols[ChosenColumn]][Row].Ord.Status == ACTIVE)  {
380.                   // Set the bomb from this alien
381.                   AlienBomb[BombIdx].Status = ACTIVE;
382.                   AlienBomb[BombIdx].X = Alien[ActiveCols[ChosenColumn]][Row].Ord.X
    + int(AlienWidth[Row] / 2);
383.                   // above sets bomb to drop around invaders centre, here we add a l
    itrle randomness around this pos
384.                   AlienBomb[BombIdx].X = (AlienBomb[BombIdx].X - 2) + random(0, 4);

385.                   AlienBomb[BombIdx].Y = Alien[ActiveCols[ChosenColumn]][Row].Ord.Y
    + 4;
386.                   break;
387.                 }
388.                 Row--;
389.               }
390.             }
391.           }
```

```
392.
393.          void BombCollisions()
394.          {
395.            //check bombs collisions
396.            for (int i = 0; i < MAXBOMBS; i++)
397.            {
398.              if (AlienBomb[i].Status == ACTIVE)
399.              {
400.                if (AlienBomb[i].Y > 64)          // gone off bottom of screen
401.                  AlienBomb[i].Status = DESTROYED;
402.                else
403.                {
404.                  // HAS IT HIT PLAYERS missile
405.                  if (Collision(AlienBomb[i], BOMB_WIDTH, BOMB_HEIGHT, Missile, MISS
    ILE_WIDTH, MISSILE_HEIGHT))
406.                  {
407.                    // destroy missile and bomb
408.                    AlienBomb[i].Status = EXPLODING;
409.                    Missile.Status = DESTROYED;
410.                  }
411.                  else
412.                  {
413.                    // has it hit players ship
414.                    if (Collision(AlienBomb[i], BOMB_WIDTH, BOMB_HEIGHT, Player.Ord,
     TANKGFX_WIDTH, TANKGFX_HEIGHT))
415.                    {
416.                      PlayerHit();
417.                      AlienBomb[i].Status = DESTROYED;
418.                    }
419.                  }
420.                }
421.              }
422.            }
423.          }
424.
425.          void PlayerHit()  {
426.            Player.Ord.Status = EXPLODING;
427.            Player.ExplosionGfxCounter = PLAYER_EXPLOSION_TIME;
428.            Missile.Status = DESTROYED;
429.          }
430.
431.          void CheckCollisions()
432.          {
433.            MissileAndAlienCollisions();
434.            MotherShipCollisions();
435.            BombCollisions();
436.          }
437.
438.          void MotherShipCollisions()
439.          {
440.            if ((Missile.Status == ACTIVE) & (MotherShip.Ord.Status == ACTIVE))
441.            {
442.              if (Collision(Missile, MISSILE_WIDTH, MISSILE_HEIGHT, MotherShip.Ord,
    MOTHERSHIP_WIDTH, MOTHERSHIP_HEIGHT))
443.              {
444.                MotherShip.Ord.Status = EXPLODING;
445.                MotherShip.ExplosionGfxCounter = EXPLOSION_GFX_TIME;
446.                Missile.Status = DESTROYED;
447.                // generate the score for the mothership hit, note in the real arcad
    e space invaders the score was not random but
448.                // just appeared so, a player could infulence its value with clever
    play, but we'll keep it a little simpler
449.                MotherShipBonus = random(4); // a random number between 0 and 3
450.                switch (MotherShipBonus)
451.                {
452.                  case 0: MotherShipBonus = 50; break;
```

```
453.                 case 1: MotherShipBonus = 100; break;
454.                 case 2: MotherShipBonus = 150; break;
455.                 case 3: MotherShipBonus = 300; break;
456.               }
457.               Player.Score += MotherShipBonus;
458.               MotherShipBonusXPos = MotherShip.Ord.X;
459.               if (MotherShipBonusXPos > 100)                // to ensure isn't hal
    f off right hand side of screen
460.                 MotherShipBonusXPos = 100;
461.               if (MotherShipBonusXPos < 0)               // to ensure isn't half
    off right hand side of screen
462.                 MotherShipBonusXPos = 0;
463.               MotherShipBonusCounter = DISPLAY_MOTHERSHIP_BONUS_TIME;
464.             }
465.           }
466.         }
467.
468.         void MissileAndAlienCollisions()
469.         {
470.           for (int across = 0; across < NUM_ALIEN_COLUMNS; across++)
471.           {
472.             for (int down = 0; down < NUM_ALIEN_ROWS; down++)
473.             {
474.               if (Alien[across][down].Ord.Status == ACTIVE)
475.               {
476.                 if (Missile.Status == ACTIVE)
477.                 {
478.                   if (Collision(Missile, MISSILE_WIDTH, MISSILE_HEIGHT, Alien[acro
    ss][down].Ord, AlienWidth[down], INVADER_HEIGHT))
479.                   {
480.                     // missile hit
481.                     Alien[across][down].Ord.Status = EXPLODING;
482.                     Missile.Status = DESTROYED;
483.                     Player.Score += GetScoreForAlien(down);
484.                     Player.AliensDestroyed++;
485.                     // calc new speed of aliens, note (float) must be before TOTAL
    _ALIENS to force float calc else
486.                     // you will get an incorrect result
487.                     Player.AlienSpeed = ((1 - (Player.AliensDestroyed / (float)TOT
    AL_ALIENS)) * INVADERS_SPEED);
488.                     // for very last alien make to  fast!
489.                     if (Player.AliensDestroyed == TOTAL_ALIENS - 2)
490.                       if (AlienXMoveAmount > 0)
491.                         AlienXMoveAmount = ALIEN_X_MOVE_AMOUNT * 2;
492.                       else
493.                         AlienXMoveAmount = -(ALIEN_X_MOVE_AMOUNT * 2);
494.                     // for very last alien make to super fast!
495.                     if (Player.AliensDestroyed == TOTAL_ALIENS - 1)
496.                       if (AlienXMoveAmount > 0)
497.                         AlienXMoveAmount = ALIEN_X_MOVE_AMOUNT * 4;
498.                       else
499.                         AlienXMoveAmount = -(ALIEN_X_MOVE_AMOUNT * 4);
500.
501.                     if (Player.AliensDestroyed == TOTAL_ALIENS)
502.                       NextLevel(&Player);
503.                   }
504.                 }
505.                 if (Alien[across][down].Ord.Status == ACTIVE)   // double check st
    ill active afer above code
506.                 {
507.                   // check if this alien is in contact with TankGfx
508.                   if (Collision(Player.Ord, TANKGFX_WIDTH, TANKGFX_HEIGHT, Alien[a
    cross][down].Ord, AlienWidth[down], ALIEN_HEIGHT))
509.                     PlayerHit();
510.                   else
511.                   {
```

```
512.                     // check if alien is below bottom of screen
513.                     if (Alien[across][down].Ord.Y + 8 > SCREEN_HEIGHT)
514.                       PlayerHit();
515.                   }
516.                 }
517.               }
518.             }
519.           }
520.         }
521.
522.         bool Collision(GameObjectStruct Obj1, unsigned char Width1, unsigned char
    Height1, GameObjectStruct Obj2, unsigned char Width2, unsigned char Height2)
523.         {
524.           return ((Obj1.X + Width1 > Obj2.X) & (Obj1.X < Obj2.X + Width2) & (Obj1.
    Y + Height1 > Obj2.Y) & (Obj1.Y < Obj2.Y + Height2));
525.         }
526.
527.         int RightMostPos()  {
528.           //returns x pos of right most alien
529.           int Across = NUM_ALIEN_COLUMNS - 1;
530.           int Down;
531.           int Largest = 0;
532.           int RightPos;
533.           while (Across >= 0) {
534.             Down = 0;
535.             while (Down < NUM_ALIEN_ROWS) {
536.               if (Alien[Across][Down].Ord.Status == ACTIVE)
537.               {
538.                 // different aliens have different widths, add to x pos to get rig
    htpos
539.                 RightPos = Alien[Across][Down].Ord.X + AlienWidth[Down];
540.                 if (RightPos > Largest)
541.                   Largest = RightPos;
542.               }
543.               Down++;
544.             }
545.             if (Largest > 0) // we have found largest for this coloum
546.               return Largest;
547.             Across--;
548.           }
549.           return 0;  // should never get this far
550.         }
551.
552.         int LeftMostPos()  {
553.           //returns x pos of left most alien
554.           int Across = 0;
555.           int Down;
556.           int Smallest = SCREEN_WIDTH * 2;
557.           while (Across < NUM_ALIEN_COLUMNS) {
558.             Down = 0;
559.             while (Down < 3) {
560.               if (Alien[Across][Down].Ord.Status == ACTIVE)
561.                 if (Alien[Across][Down].Ord.X < Smallest)
562.                   Smallest = Alien[Across][Down].Ord.X;
563.               Down++;
564.             }
565.             if (Smallest < SCREEN_WIDTH * 2) // we have found smalest for this col
    oum
566.               return Smallest;
567.             Across++;
568.           }
569.           return 0;  // should nevr get this far
570.         }
571.
572.         void UpdateDisplay()
573.         {
```

```
574.            int i;
575.
576.            arduboy.clear();
577.
578.            // Mothership bonus display if required
579.            if (MotherShipBonusCounter > 0)
580.            {
581.              // mothership bonus
582.              arduboy.setCursor(MotherShipBonusXPos, 0);
583.              arduboy.print(MotherShipBonus);
584.              MotherShipBonusCounter--;
585.            }
586.            else
587.            {
588.              // draw score and lives, anything else can go above them
589.              arduboy.setCursor(0, 0);
590.              arduboy.print(Player.Score);
591.              arduboy.setCursor(SCREEN_WIDTH - 7, 0);
592.              arduboy.print(Player.Lives);
593.            }
594.
595.            //BOMBS
596.            // draw bombs next as aliens have priority of overlapping them
597.            for (i = 0; i < MAXBOMBS; i++)  {
598.              if (AlienBomb[i].Status == ACTIVE)
599.                arduboy.drawBitmap(AlienBomb[i].X, AlienBomb[i].Y,  AlienBombGfx, 2,
     4, WHITE);
600.              else  {// must be destroyed
601.                if (AlienBomb[i].Status == EXPLODING)
602.                  arduboy.drawBitmap(AlienBomb[i].X - 4, AlienBomb[i].Y,  ExplosionG
    fx, 4, 8, WHITE);
603.                // Ensure on next draw that ExplosionGfx disappears
604.                AlienBomb[i].Status = DESTROYED;
605.              }
606.            }
607.
608.            //Invaders
609.            for (int across = 0; across < NUM_ALIEN_COLUMNS; across++)
610.            {
611.              for (int down = 0; down < NUM_ALIEN_ROWS; down++)
612.              {
613.                if (Alien[across][down].Ord.Status == ACTIVE) {
614.                  switch (down)  {
615.                    case 0:
616.                      if (AnimationFrame)
617.                        arduboy.drawBitmap(Alien[across][down].Ord.X, Alien[across][
    down].Ord.Y,  InvaderTopGfx, AlienWidth[down], INVADER_HEIGHT, WHITE);
618.                      else
619.                        arduboy.drawBitmap(Alien[across][down].Ord.X, Alien[across][
    down].Ord.Y,  InvaderTopGfx2, AlienWidth[down], INVADER_HEIGHT, WHITE);
620.                      break;
621.                    case 1:
622.                      if (AnimationFrame)
623.                        arduboy.drawBitmap(Alien[across][down].Ord.X, Alien[across][
    down].Ord.Y,  InvaderMiddleGfx, AlienWidth[down], INVADER_HEIGHT, WHITE);
624.                      else
625.                        arduboy.drawBitmap(Alien[across][down].Ord.X, Alien[across][
    down].Ord.Y,  InvaderMiddleGfx2, AlienWidth[down], INVADER_HEIGHT, WHITE);
626.                      break;
627.                    default:
628.                      if (AnimationFrame)
629.                        arduboy.drawBitmap(Alien[across][down].Ord.X, Alien[across][
    down].Ord.Y,  InvaderBottomGfx, AlienWidth[down], INVADER_HEIGHT, WHITE);
630.                      else
631.                        arduboy.drawBitmap(Alien[across][down].Ord.X, Alien[across][
    down].Ord.Y,  InvaderBottomGfx2, AlienWidth[down], INVADER_HEIGHT, WHITE);
```

```
632.                 }
633.               }
634.               else  {
635.                 if (Alien[across][down].Ord.Status == EXPLODING) {
636.                   Alien[across][down].ExplosionGfxCounter--;
637.                   if (Alien[across][down].ExplosionGfxCounter > 0)  {
638.                     arduboy.drawBitmap(Alien[across][down].Ord.X, Alien[across][do
    wn].Ord.Y,  ExplosionGfx, 13, 8, WHITE);
639.                   }
640.                   else
641.                     Alien[across][down].Ord.Status = DESTROYED;
642.                 }
643.               }
644.             }
645.           }
646.
647.           // player
648.           if (Player.Ord.Status == ACTIVE)
649.             arduboy.drawBitmap(Player.Ord.X, Player.Ord.Y,  TankGfx, TANKGFX_WIDTH
    , TANKGFX_HEIGHT, WHITE);
650.           else {
651.             if (Player.Ord.Status == EXPLODING)  {
652.               for (i = 0; i < TANKGFX_WIDTH; i += 2)  {
653.                 arduboy.drawBitmap(Player.Ord.X + i, Player.Ord.Y,  ExplosionGfx,
    random(4) + 2, 8, WHITE);
654.               }
655.               Player.ExplosionGfxCounter--;
656.               if (Player.ExplosionGfxCounter == 0)  {
657.                 Player.Ord.Status = DESTROYED;
658.                 LoseLife();
659.               }
660.             }
661.           }
662.           //missile
663.           if (Missile.Status == ACTIVE)
664.             arduboy.drawBitmap(Missile.X, Missile.Y,  MissileGfx, MISSILE_WIDTH, M
    ISSILE_HEIGHT, WHITE);
665.
666.           // mothership (not bonus if hit)
667.           if (MotherShip.Ord.Status == ACTIVE)
668.             arduboy.drawBitmap(MotherShip.Ord.X, MotherShip.Ord.Y,  MotherShipGfx,
     MOTHERSHIP_WIDTH, MOTHERSHIP_HEIGHT, WHITE);
669.           else
670.           {
671.             if (MotherShip.Ord.Status == EXPLODING)
672.             {
673.               for (i = 0; i < MOTHERSHIP_WIDTH; i += 2)  {
674.                 arduboy.drawBitmap(MotherShip.Ord.X + i, MotherShip.Ord.Y,  Explos
    ionGfx, random(4) + 2, MOTHERSHIP_HEIGHT, WHITE);
675.               }
676.               MotherShip.ExplosionGfxCounter--;
677.               if (MotherShip.ExplosionGfxCounter == 0)  {
678.                 MotherShip.Ord.Status = DESTROYED;
679.               }
680.             }
681.           }
682.           arduboy.display();
683.         }
684.
685.         void LoseLife()  {
686.           Player.Lives--;
687.           if (Player.Lives > 0)  {
688.             DisplayPlayerAndLives(&Player);
689.             // clear alien missiles
690.             for (int i = 0; i < MAXBOMBS; i++)  {
691.               AlienBomb[i].Status = DESTROYED;
```

```
692.               AlienBomb[i].Y = 0;
693.             }
694.             // ENABLE PLAYER
695.             Player.Ord.Status = ACTIVE;
696.             Player.Ord.X = 0;
697.           }
698.           else  {
699.             GameOver();
700.           }
701.         }
702.
703.         void GameOver()  {
704.           GameInPlay = false;
705.           arduboy.clear();
706.           CentreText("Player 1", 0);
707.           CentreText("Game Over", 12);
708.           CentreText("Score ", 24);
709.           arduboy.print(Player.Score);
710.           if (Player.Score > HiScore)
711.           {
712.             CentreText("NEW HIGH SCORE!!!", 36);
713.             CentreText("**CONGRATULATIONS**", 48);
714.           }
715.           arduboy.display();
716.           if (Player.Score > HiScore) {
717.             HiScore = Player.Score;
718.             EEPROM.put(0, HiScore);
719.           }
720.           delay(2500);
721.         }
722.
723.         void DisplayPlayerAndLives(PlayerStruct * Player)  {
724.           arduboy.clear();
725.           CentreText("Player 1", 0);
726.           CentreText("Score ", 12);
727.           arduboy.print(Player->Score);
728.           CentreText("Lives ", 24);
729.           arduboy.print(Player->Lives);
730.           CentreText("Level ", 36);
731.           arduboy.print(Player->Level);
732.           arduboy.display();
733.           delay(2000);
734.           Player->Ord.X = PLAYER_X_START;
735.         }
736.
737.         void CentreText(const char *Text, unsigned char Y)  {
738.           // centres text on screen
739.           arduboy.setCursor(int((float)(SCREEN_WIDTH) / 2 - ((strlen(Text) * 6) /
    2)), Y);
740.           arduboy.print(Text);
741.         }
742.
743.         void InitPlayer()  {
744.           Player.Ord.Y = PLAYER_Y_START;
745.           Player.Ord.X = PLAYER_X_START;
746.           Player.Ord.Status = ACTIVE;
747.           Player.Lives = LIVES;
748.           Player.Level = 0;
749.           Missile.Status = DESTROYED;
750.           Player.Score = 0;
751.         }
752.
753.         void NextLevel(PlayerStruct * Player)  {
754.           // reset any dropping bombs
755.           int YStart;
756.           for (int i = 0; i < MAXBOMBS; i++)
```

```
757.             AlienBomb[i].Status = DESTROYED;
758.           AnimationFrame = false;
759.           Player->Level++;
760.           YStart = ((Player-
    >Level - 1) % LEVEL_TO_RESET_TO_START_HEIGHT) * AMOUNT_TO_
761.           InitAliens(YStart);
762.           AlienXMoveAmount = ALIEN_X_MOVE_AMOUNT;
763.           Player->AlienSpeed = INVADERS_SPEED;
764.           Player->AliensDestroyed = 0;
765.           MotherShip.Ord.X = -MOTHERSHIP_WIDTH;
766.           MotherShip.Ord.Status = DESTROYED;
767.           Missile.Status = DESTROYED;
768.           randomSeed(100);
769.           DisplayPlayerAndLives(Player);
770.         }
771.
772.         void NewGame() {
773.           InitPlayer();
774.           NextLevel(&Player);
775.         }
776.
777.         void InitAliens(int YStart)  {
778.           for (int across = 0; across < NUM_ALIEN_COLUMNS; across++)  {
779.             for (int down = 0; down < 3; down++)  {
780.               // we add down to centralise the aliens, just happens to be the righ
    t value we need per row!
781.               // we need to adjust a little as row zero should be 2, row 1 should
    be 1 and bottom row 0
782.               Alien[across][down].Ord.X = X_START_OFFSET + (across * (LARGEST_ALIE
    N_WIDTH + SPACE_BETWEEN_ALIEN_COLUMNS)) - (AlienWidth[down] / 2);
783.               Alien[across][down].Ord.Y = YStart + (down * SPACE_BETWEEN_ROWS);
784.               Alien[across][down].Ord.Status = ACTIVE;
785.               Alien[across][down].ExplosionGfxCounter = EXPLOSION_GFX_TIME;
786.             }
787.           }
788.           MotherShip.Ord.Y = 0;
789.           MotherShip.Ord.X = -MOTHERSHIP_WIDTH;
790.           MotherShip.Ord.Status = DESTROYED;
791.         }
```

Game Over Screen with

Attract Screen

Level Player Data

Game Physics

Game Over Screen with Player Data