Part 04

Pygame Zero Invaders

There must be very few people who have not played Space Invaders, and for some it may have been their very first experience of a computer game

MAKER

Mark Vanstone

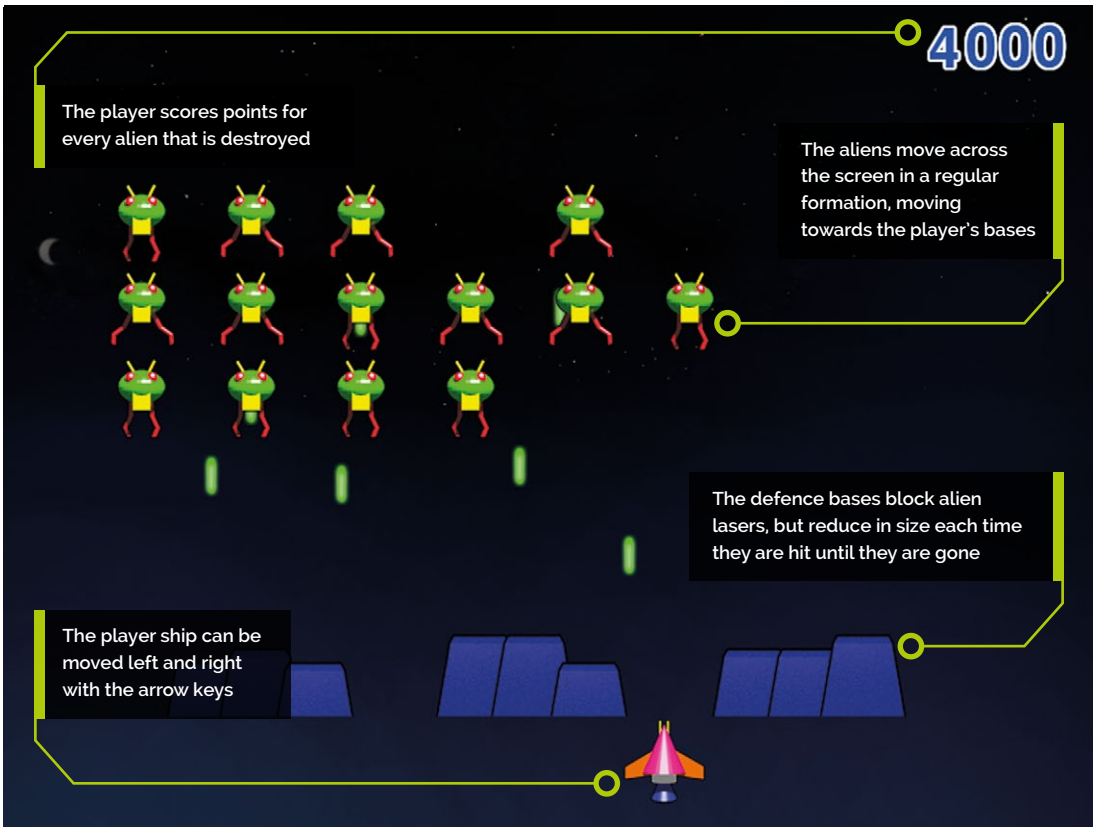
Educational software author from the nineties, author of the ArcVenture series, disappeared into the corporate software wasteland. Rescued by the Raspberry Pi!

magpi.cc/YiZnxl

@mindexplorers

The Space Invaders game format requires quite a few different coding techniques to make it work. For some time, if your author needed to learn a new coding language, he would task himself to write a Space Invaders game in it. This would give a good workout through the syntax and functions of the language.

This tutorial will be split into two parts. In the first we will build a basic invaders game with aliens, lasers, defence bases, and a score. The second part (next issue) will add all the extra bits that make it into the game that appeared in amusement arcades and sports halls in the 1970s.



You'll Need

- Raspbian Jessie or newer
- An image manipulation program such as GIMP, or images from **magpi.cc/MATfil**
- The latest version of Pygame Zero (1.2)
- A cool head as the lasers rain down on you

01 Let's get stuck in

If you have read the previous episodes of this series, you will know how we set up a basic Pygame Zero program, so we can jump right in to getting things on the screen. We will need some graphics for the various elements of the game – you can design them yourself or use ours from: **magpi.cc/MATfil**. The Pygame Zero default screen size is 800 width by 600 height, which is a good size for this game, so we don't need to define `WIDTH` or `HEIGHT`.

02 A bit of a player

Let's start with getting the player ship on the screen. If we call our graphic **player.png**, then we can create the player Actor near the top of our code by writing `player = Actor("player", (400, 550))`.

We will probably want something a bit more interesting than just a plain black window, so we can add a background in our `draw()` function. If we draw this first, everything else that we draw will be on top of it. We can draw it using the `blit()` function by writing `screen.blit('background', (0, 0))` – assuming we have called our background image **background.png**. Then, to draw the player, just add `player.draw()` afterwards.

03 Let's get moving

We need the player ship to respond to key presses, so we'll check the Pygame Zero keyboard object to see if certain keys are currently pressed. Let's make a new function to deal with these inputs. We will call the function `checkKeys()` and we'll need to call it from our `update()` function.

In the `checkKeys()` function, we write `if keyboard.left:` and then `if player.x > 40: player.x -= 5`. We need to declare the player Actor object as global inside our `checkKeys()` function. We then write a similar piece of code to deal with the right arrow key; **figure1.py** shows how this all fits together.

04 An alien concept

We now want to create a load of aliens in formation. You can have them in whatever format you want, but we'll set up three rows of aliens with six on each row. We have an image called **alien.png** and can make an Actor for each alien that we will store in a list so that we can easily loop through the list to perform actions on them. When we create the alien Actors, we will use a bit of maths to set the initial x and y co-ordinates. It would be a good idea to define a function to set up the aliens – `initAliens()` – and because we will want to set up other elements too, we could define a function `init()`, from which we can call all the setup functions.

05 Doing the maths

To position our aliens and to create them as Actors, we can declare a list – `aliens = []` – and then create a loop using `for a in range(18):`. In this loop, we need to create each Actor and then work out where their x and y co-ordinates will be to start. We can do this in the loop by writing: `aliens.append(Actor("alien1", (210+(a % 6)*80,100+(int(a/6)*64))))`. This may look a little daunting, but we can break it down by saying 'x is 210 plus the remainder of dividing by 6 multiplied by 80'.

This will provide us with x co-ordinates starting at 210 and with a spacing of 80 between each. The y calculation is similar, but we use normal division, make it an integer, and multiply by 64.

figure1.py

```
001. import pgzrun
002.
003. player = Actor("player", (400, 550)) # Load in the player
     Actor image
004.
005. def draw(): # Pygame Zero draw function
006.     screen.blit('background', (0, 0))
007.     player.draw()
008.
009. def update(): # Pygame Zero update function
010.     checkKeys()
011.
012. def checkKeys():
013.     global player
014.     if keyboard.left:
015.         if player.x > 40: player.x -= 5
016.     if keyboard.right:
017.         if player.x < 760: player.x += 5
018.
019. pgzrun.go()
```

▲ Functions to create a player ship and background, display them, and handle moving the player ship

figure2.py

```
001.  def updateAliens():
002.      global moveSequence, moveDelay
003.      movex = movey = 0
004.      if moveSequence < 10 or moveSequence > 30: movex = -15
005.      if moveSequence == 10 or moveSequence == 30:
006.          movey = 50
007.      if moveSequence >10 and moveSequence < 30: movex = 15
008.      for a in range(len(aliens)):
009.          animate(aliens[a], pos=(aliens[a].x + movex,
      aliens[a].y + movey), duration=0.5, tween='linear')
010.          if randint(0, 1) == 0:
011.              aliens[a].image = "alien1"
012.          else:
013.              aliens[a].image = "alien1b"
014.      moveSequence +=1
015.      if moveSequence == 40: moveSequence = 0
```

▲ The `updateAliens()` function. Calculate the movement for the aliens based on the variable `moveSequence`

06 Believing the strangest things

After that slightly obscure title reference, we shall introduce the idea of the alien having a status. As we have seen in previous instalments, we can add extra data to our Actors, and in this case we will want to add a status variable to the alien after we have created it. We'll explain how we are going to use this a bit later. Now it's time to get the little guys on the screen and ready for action. We can write a simple function called `drawAlien()` and just loop through the alien list to draw them by writing: `for a in range(len(aliens)): aliens[a].draw()` . Call the `drawAlien()` function inside the `draw()` function.

07 The aliens are coming!

We are going to create a function that we call inside our `update()` function that keeps track of what should happen to the aliens. We'll call it `updateAliens()`. We don't want to move the aliens every time the update cycle runs, so we'll keep a counter called moveCounter and increment it each `update()`; then, if it gets to a certain value (`moveDelay`), we will zero the counter. If the counter is zero, we call `updateAliens()`. The `updateAliens()` function will calculate how much they need to move in the x and y directions to get them to go backwards and forwards across the screen and move down when they reach the edges.

08 Updating the aliens

To work out where the aliens should move, we'll make a counter loop from 0 to 40. From 0 to 9 we'll move the aliens left, on 10 we'll move them down, then from 11 to 29 move them right. On 30 they move down and then from 31 to 40 move left. Have a look at **figure2.py** to see how we can do this in the `updateAliens()` function and how that function fits into our `update()` function. Notice how we can use the Pygame Zero function `animate()` to get them to move smoothly. We can also add a switch between images to make their legs move.

09 All your base are belong to us

Now we are going to build our defence bases. There are a few problems to overcome in that we want to construct our bases from Actors, but there are no methods for clipping an Actor when it is displayed. Clipping is a term to describe that we only display a part of the image. This is a method we need if we are going to make the bases shrink as they are hit by alien lasers. What we will have to do is add a function to the Actor, just like we have added extra variables to them before.

10 Build base

We will make three bases which will be made of three Actors each. If we wanted to display the whole image (**base1.png**), we would create a list of base Actors and display each Actor with some code like `bases[0].draw()`. What we want to do is add a variable to the base to show how high we want it to be. We will also need to write a new function to draw the base according to the height variable. Have a look at **figure3.py** to see how we write the new function and attach it to each Actor. This means we can now call this function from each base Actor using: `bases[b].drawClipped()`, as shown in the `drawBases()` function.

11 Can I shoot something now?

To make this into a shooting game, let's add some lasers. We need to fire lasers from the player ship and also from the aliens, but we are going to keep them all in the same list. When we create a new laser by making an Actor and adding it to the

Top Tip

Beware of deleting elements of a list

If you delete a list element while you are looping through it with `range (len(list))`, when you get to the end of the loop it will run out of elements and return an error because the range of the loop is the original length of the list.

list `lasers[]`, we can give the Actor a type. In this case we'll make alien lasers type 0 and player lasers type 1. We'll also need to add a status variable. The creation and updating of the lasers is similar to other elements we've looked at; **figure4.py** (overleaf) shows the functions that we can use.

12 Making the lasers work

You can see in **figure4.py** that we can create a laser from the player by adding a check for the **SPACE** key being pressed in our `checkKeys()` function. We will use the blue laser image called **laser2.png**. Once the new laser is in our list of lasers, it will be drawn to the screen if we call the `drawLasers()` function inside our `draw()` function. In our `updateLasers()` function we loop through the list of lasers and check which type it is. So if it is type 1 (player), we move the laser up the screen and then check to see if it hit anything. Notice the calls to a `listCleanup()` function at the bottom. We will come to this in a bit.

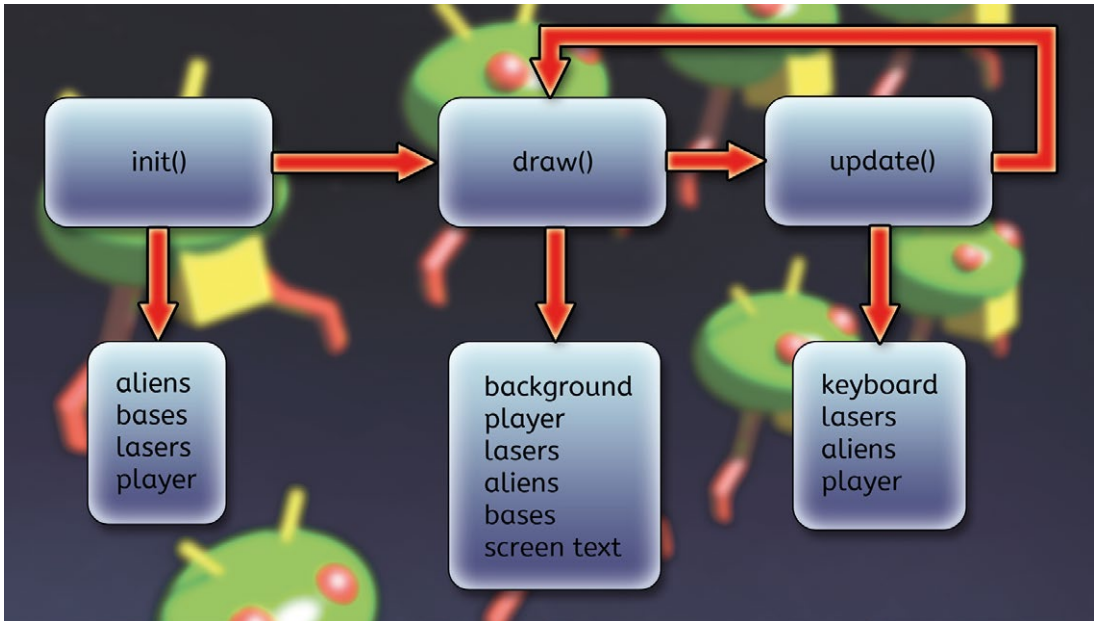
figure3.py

```
001. def drawClipped(self):
002.     screen.surface.blit(self._surf, (self.x-32, self.y-
     self.height+30),(0,0,64,self.height))
003.
004. def initBases():
005.     global bases
006.     bases = []
007.     bc = 0
008.     for b in range(3):
009.         for p in range(3):
010.             bases.append(Actor("base1",
     midbottom=(150+(b*200)+(p*40),520)))
011.             bases[bc].drawClipped = drawClipped.__get__
     (bases[bc])
012.             bases[bc].height = 60
013.             bc +=1
014.
015. def drawBases():
016.     for b in range(len(bases)): bases[b].drawClipped()
```

▲ Setting up an extension function to draw an Actor with clipping

13 Collision course

Let's look at `checkPlayerLaserHit()` first. We can detect if the laser has hit any aliens by looping round the alien list and checking with the Actor function – `collidepoint((lasers[l].x, lasers[l].y))` – to see if a collision has occurred. If an alien has been hit, this is where our status variables come into play. Rather than just removing the laser and the alien from their lists, we need to flag them as ready to remove. The reason for this is that if we remove anything from a list while we are



Top Tip

Write functions for each collective action

To make coding easier to read rather than having lots of code associated with one type of element in the `draw()` or `update()` functions, send it out to a function like `drawLasers()` or `checkKeys()`.

figure4.py

```
001.  def checkKeys():
002.      global player, lasers
003.      if keyboard.space:
004.          l = len(lasers)
005.          lasers.append(Actor("laser2",
      (player.x,player.y-32)))
006.          lasers[l].status = 0
007.          lasers[l].type = 1
008.
009.  def drawLasers():
010.      for l in range(len(lasers)): lasers[l].draw()
011.
012.  def updateLasers():
013.      global lasers, aliens
014.      for l in range(len(lasers)):
015.          if lasers[l].type == 0:
016.              lasers[l].y += (2*DIFFICULTY)
017.              checkLaserHit(l)
018.              if lasers[l].y > 600: lasers[l].status = 1
019.          if lasers[l].type == 1:
020.              lasers[l].y -= 5
021.              checkPlayerLaserHit(l)
022.              if lasers[l].y < 10: lasers[l].status = 1
023.      lasers = listCleanup(lasers)
024.      aliens = listCleanup(aliens)
```

▲ Checking the keys that are pressed, creating lasers, moving them, and checking if they have collided with anything

looping through any of the lists then by the time we get to the end of the list, we are an element short and an error will be created. So we set these Actors to be removed with `status` and then remove them afterwards with `listCleanup()`.

Top Tip

Collect all your setup code in one place

If possible, it is good to have as much of the code that sets everything back to the beginning in one place so that you can easily restart the game.

14 Cleaning up the mess

The `listCleanup()` function creates a new empty list, then runs through the list that is passed to it, only transferring items to the new list that have a status of 0. This new list is then returned back and used as the list going forward. Now that we have made a system for one type of laser we can easily adapt that for our alien laser type. We can create the alien lasers in the same way as the player lasers, but instead of waiting for a keyboard press we can just produce them at random intervals using `if randint(0, 5) == 0:` when we are updating our aliens. We set the type to 0 rather than 1 and move them down the screen in our `updateLasers()` function.

15 Covering the bases

So far, we haven't looked at what happens when a laser hits one of the defence bases. Because we are changing the height of the base Actors, the built-in collision detection won't give us the result we want, so we need to write another custom function to check laser collision on the base Actor. Our new function, `collideLaser()` will check the laser co-ordinates against the base's co-ordinates, taking into account the height of the base. We then attach the new function to our base Actor when it is created. We can use the new `collideLaser()` function for checking both the player and the alien lasers and remove the laser if it hits – and if it is an alien laser, reduce the height of the base that was hit.

16 Laser overkill

We may want to change the number of lasers being fired by the aliens, but at the moment our player ship gets to fire a laser every `update()` cycle. If the **SPACE** key is held down, a constant stream of lasers will be fired, which not only is a little bit unfair on the poor aliens but will also take its toll on the speed of the game. So we need to put some limits on the firing speed and we can do this with another built-in Pygame Zero object: the clock. If we add a variable `laserActive` to our player Actor and set it to zero when it fires, we can then call `clock.schedule(makeLaserActive, 1.0)` to call the function `makeLaserActive()` after 1 second.

17 I'm hit! I'm hit!

We need to look now at what happens when the player ship is hit by a laser. For this we will make a multi-frame animation. We have five explosion images to put into a list, with our normal ship image at the beginning, and attach it to our player Actor. We need to import the Math module, then in each `draw()` cycle we write: `player.image = player.images[math.floor(player.status/6)]`, which will display the normal ship image while `player.status` is 0. If we set it to 1 when the player ship is hit, we can start the animation in motion. In the `update()` function we write: `if player.status > 0: player.status += 1`. As the status value increases, it will start to draw the sequence of frames one after the other.

18 Initialisation

Now, it may seem a bit strange to be dealing with initialisation near the end of the tutorial, but we have been adding and changing the structure of our game elements as we have gone along and only now can we really see all the data that we need to set up before the game starts. In Step 04 we created a function called `init()` that we should call to get the game started. We could also use this function to reset everything back to start the game again. If we have included all the initialisation functions and variables we have talked about, we should have something like **figure5.py**.

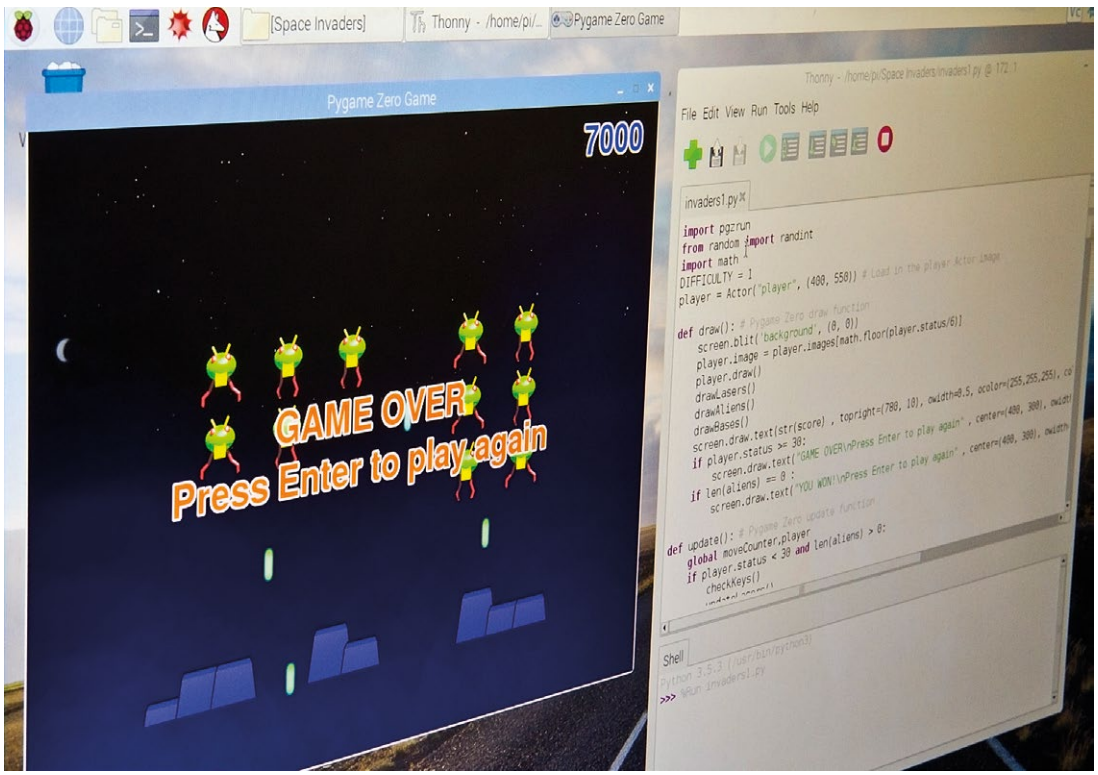
figure5.py

```
001. def init():
002.     global lasers, score, player, moveSequence,
     moveCounter, moveDelay
003.     initAliens()
004.     initBases()
005.     moveCounter = moveSequence = player.status = score =
     player.laserCountdown = 0
006.     lasers = []
007.     moveDelay = 30
008.     player.images = ["player","explosion1","explosion2",
     "explosion3","explosion4","explosion5"]
009.     player.laserActive = 1
```

▲ The initialisation of our data. Calling this function sets our variables back to their start values

19 They're coming in too fast!

There are a few finishing touches to do to complete this first part. We can set a `DIFFICULTY` value near the top of the code and use it on various elements to make the game harder. We should also add a score, which we do by adding 1000 to a global variable score if an alien is hit, and then display that in the top right of the screen in the `draw()` function. When the game finishes (the player has been hit or all the aliens are gone), we should display a suitable message. Have a look at the complete listing to see how these bits fit in. When that's all done, we should have the basis of a Space Invaders game. In the next part of this series we will add more into the game, such as levels, lives, sound, bonus aliens, and a leaderboard.



◀ It's game over for now, but we'll be back next issue to improve the game

Top Tip

Define several variables at once

If you are setting several variables to the same value, you can combine them into one line by writing `a = b = c = 0` to set a, b, and c to zero.

invaderspart1.py

```
import pgzrun
from random import randint
import math
DIFFICULTY = 1
player = Actor("player", (400, 550)) # Load in the player Actor image

def draw(): # Pygame Zero draw function
    screen.blit('background', (0, 0))
    player.image = player.images[math.floor(player.status/6)]
    player.draw()
    drawLasers()
    drawAliens()
    drawBases()
    screen.draw.text(str(score), topright=(780, 10), owidth=0.5, ocolor=(255,255,255), color=(0,64,255), fontsize=60)
    if player.status >= 30:
        screen.draw.text("GAME OVER\nPress Enter to play again" , center=(400, 300), owidth=0.5, ocolor=(255,255,255), color=(255,64,0), fontsize=60)
    if len(aliens) == 0 :
        screen.draw.text("YOU WON!\nPress Enter to play again" , center=(400, 300), owidth=0.5, ocolor=(255,255,255), color=(255,64,0) , fontsize=60)

def update(): # Pygame Zero update function
    global moveCounter,player
    if player.status < 30 and len(aliens) > 0:
        checkKeys()
        updateLasers()
        moveCounter += 1
        if moveCounter == moveDelay:
            moveCounter = 0
            updateAliens()
        if player.status > 0: player.status += 1
    else:
        if keyboard.RETURN: init()

def drawAliens():
    for a in range(len(aliens)): aliens[a].draw()

def drawBases():
    for b in range(len(bases)):
        bases[b].drawClipped()

def drawLasers():
    for l in range(len(lasers)): lasers[l].draw()

def checkKeys():
    global player, lasers
    if keyboard.left:
        if player.x > 40: player.x -= 5
    if keyboard.right:
        if player.x < 760: player.x += 5
    if keyboard.space:
        if player.laserActive == 1:
            player.laserActive = 0
            clock.schedule(makeLaserActive, 1.0)
            l = len(lasers)
            lasers.append(Actor("laser2", (player.x,player.y-32)))
            lasers[l].status = 0
            lasers[l].type = 1

def makeLaserActive():
    global player
    player.laserActive = 1

def checkBases():
    for b in range(len(bases)):
        if l < len(bases):
            if bases[b].height < 5:
                del bases[b]

def updateLasers():
    global lasers, aliens
    for l in range(len(lasers)):
        if lasers[l].type == 0:
            lasers[l].y += (2*DIFFICULTY)
            checkLaserHit(l)
            if lasers[l].y > 600:
                lasers[l].status = 1
        if lasers[l].type == 1:
            lasers[l].y -= 5
            checkPlayerLaserHit(l)
            if lasers[l].y < 10:
                lasers[l].status = 1
    lasers = listCleanup(lasers)
    aliens = listCleanup(aliens)

def listCleanup(l):
    newList = []
    for i in range(len(l)):
        if l[i].status == 0: newList.append(l[i])
    return newList

def checkLaserHit(l):
    global player
```

DOWNLOAD THE FULL CODE:
magpi.cc/lwqLZj

```
092.     if player.collidepoint((lasers[l].x,
    lasers[l].y)):
093.         player.status = 1
094.         lasers[l].status = 1
095.     for b in range(len(bases)):
096.         if bases[b].collideLaser(lasers[l]):
097.             bases[b].height -= 10
098.             lasers[l].status = 1
099.
100. def checkPlayerLaserHit(l):
101.     global score
102.     for b in range(len(bases)):
103.         if bases[b].collideLaser(lasers[l]):
104.             lasers[l].status = 1
105.     for a in range(len(aliens)):
106.         if aliens[a].collidepoint((lasers[l].x,
    lasers[l].y)):
107.             lasers[l].status = 1
108.             aliens[a].status = 1
109.             score += 1000
110.
111. def updateAliens():
112.     global moveSequence, lasers, moveDelay
113.     movex = movey = 0
114.     if moveSequence < 10 or moveSequence > 30:
115.         movex = -15
116.     if moveSequence == 10 or moveSequence == 30:
117.         movey = 50 + (10 * DIFFICULTY)
118.         moveDelay -= 1
119.     if moveSequence >10 and moveSequence < 30:
120.         movex = 15
121.     for a in range(len(aliens)):
122.         animate(aliens[a], pos=(aliens[a].x + movex,
    aliens[a].y + movey), duration=0.5, tween='linear')
123.         if randint(0, 1) == 0:
124.             aliens[a].image = "alien1"
125.         else:
126.             aliens[a].image = "alien1b"
127.             if randint(0, 5) == 0:
128.                 lasers.append(Actor("laser1",
    (aliens[a].x,aliens[a].y)))
129.                 lasers[len(lasers)-1].status = 0
130.                 lasers[len(lasers)-1].type = 0
131.         if aliens[a].y > 500 and player.status ==
    0:
132.             player.status = 1
133.     moveSequence +=1
134.     if moveSequence == 40: moveSequence = 0
135.
136. def init():
137.     global lasers, score, player, moveSequence,
    moveCounter, moveDelay
138.     initAliens()
139.     initBases()
140.     moveCounter = moveSequence = player.status =
    score = player.laserCountdown = 0
141.     lasers = []
142.     moveDelay = 30
143.     player.images =
    ["player","explosion1","explosion2",
    "explosion3","explosion4","explosion5"]
144.     player.laserActive = 1
145.
146. def initAliens():
147.     global aliens
148.     aliens = []
149.     for a in range(18):
150.         aliens.append(Actor("alien1", (210+
151. (a % 6)*80,100+(int(a/6)*64))))
152.         aliens[a].status = 0
153.
154. def drawClipped(self):
155.     screen.surface.blit(self._surf, (self.x-32,
    self.y-self.height+30),(0,0,64,self.height))
156.
157. def collideLaser(self, other):
158.     return (
159.         self.x-20 < other.x+5 and
160.         self.y-self.height+30 < other.y and
161.         self.x+32 > other.x+5 and
162.         self.y-self.height+30 + self.height >
    other.y
163.     )
164.
165. def initBases():
166.     global bases
167.     bases = []
168.     bc = 0
169.     for b in range(3):
170.         for p in range(3):
171.             bases.append(Actor("base1",
    midbottom=(150+(b*200)+(p*40),520)))
172.             bases[bc].drawClipped =
    drawClipped.__get__(bases[bc])
173.             bases[bc].collideLaser =
    collideLaser.__get__(bases[bc])
174.             bases[bc].height = 60
175.             bc +=1
176.
177. init()
178. pgzrun.go()
```