# 13 ¡Alienígenas!



En este capítulo, añadiremos alienígenas a *Alien Invasion*. Añadiremos un alienígena cerca de la parte superior de la pantalla y luego generaremos toda una flota de alienígenas. Haremos que la flota avance hacia los lados y hacia abajo, y nos desharemos de cualquier alienígena alcanzado por una bala. Por último, limitaremos el número de naves del jugador y terminaremos la partida cuando el jugador se quede sin naves.

A medida que trabajes en este capítulo, aprenderás más sobre Pygame y sobre la gestión de un gran proyecto. También aprenderás a detectar colisiones entre objetos del juego, como balas y alienígenas. Detectar colisiones te ayuda a definir las interacciones entre los elementos de tus juegos. Por ejemplo, puedes confinar a un personaje dentro de las paredes de un laberinto o pasar una pelota entre dos personajes. Seguiremos trabajando a partir de un plan que revisaremos de vez en cuando para mantener el enfoque de nuestras sesiones de escritura de código.

Antes de empezar a escribir nuevo código para añadir una flota de alienígenas a la pantalla, echemos un vistazo al proyecto y actualicemos nuestro plan.

## Revisar el proyecto

Cuando empiezas una nueva fase de desarrollo en un proyecto grande, siempre es buena idea revisar tu plan y aclarar lo que quieres conseguir con el código que vas a escribir. En este capítulo, haremos lo siguiente:

* Añade un único alienígena en la esquina superior izquierda de la pantalla, con un espaciado adecuado a su alrededor.
* Llena la parte superior de la pantalla con tantos alienígenas como quepan horizontalmente. A continuación, crearemos filas adicionales de alienígenas hasta que tengamos una flota completa.
* Haz que la flota se mueva hacia los lados y hacia abajo hasta que toda la flota sea derribada, un alienígena golpee la nave o un alienígena llegue al suelo. Si toda la flota es derribada, crearemos una nueva flota. Si un alienígena alcanza la nave o el suelo, destruiremos la nave y crearemos una nueva flota.
* Limitaremos el número de naves que el jugador puede utilizar, y terminaremos la partida cuando el jugador haya agotado el número de naves asignado.

Perfeccionaremos este plan a medida que implementemos funciones, pero esto es lo suficientemente específico como para empezar a escribir código.

También debes revisar el código existente cuando empieces a trabajar en una nueva serie de características de un proyecto. Dado que cada nueva fase suele hacer más complejo un proyecto, es mejor limpiar cualquier código desordenado o ineficiente. Hemos ido refactorizando sobre la marcha, así que no hay código que tengamos que refactorizar en este momento.

## Crear el primer alienígena

Colocar un alienígena en la pantalla es como colocar una nave en la pantalla. El comportamiento de cada alienígena está controlado por una clase llamada Alien, que estructuraremos como la clase Ship. Seguiremos utilizando imágenes de mapa de bits por simplicidad. Puedes encontrar tu propia imagen para un alienígena o utilizar la que se muestra en la [Figura 13-1](#figure13-1), que está disponible en los recursos del libro en [https://ehmatthes.github.io/pcc\_3e.](https://ehmatthes.github.io/pcc_3e) Esta imagen tiene un fondo gris, que coincide con el color de fondo de la pantalla. Asegúrate de guardar el archivo de imagen que elijas en la carpeta *images*.

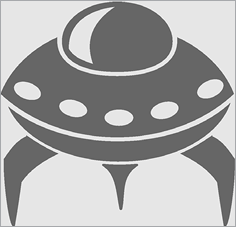


Figura13-1: El alienígena que utilizaremos para construir la flota

### Crear la clase Alien

Ahora escribiremos la clase Alien y la guardaremos como *alien.py*:

**alien.py**

import pygame  
from pygame.sprite import Sprite  
  
class Alien(Sprite):  
 """A class to represent a single alien in the fleet."""  
  
 def \_\_init\_\_(self, ai\_game):  
 """Initialize the alien and set its starting position."""  
 super().\_\_init\_\_()  
 self.screen = ai\_game.screen  
  
 # Load the alien image and set its rect attribute.  
 self.image = pygame.image.load('images/alien.bmp')  
 self.rect = self.image.get\_rect()  
  
 # Start each new alien near the top left of the screen.  
❶ self.rect.x = self.rect.width  
 self.rect.y = self.rect.height  
  
 # Store the alien's exact horizontal position.  
❷ self.x = float(self.rect.x)

La mayor parte de esta clase es como la clase Ship, excepto por la colocación del alienígena en la pantalla. Inicialmente colocamos cada alienígena cerca de la esquina superior izquierda de la pantalla; añadimos un espacio a su izquierda igual a la anchura del alienígena y un espacio por encima igual a su altura ❶, para que sea fácil de ver. Nos interesa sobre todo la velocidad horizontal de los alienígenas, así que seguiremos con precisión la posición horizontal de cada alienígena ❷.

Esta clase Alien no necesita un método para dibujarla en la pantalla; en su lugar, utilizaremos un método de grupo de Pygame que dibuja automáticamente todos los elementos de un grupo en la pantalla.

### Crear una instancia del Alien

Queremos crear una instancia de Alien para poder ver al primer alienígena en la pantalla. Como forma parte de nuestro trabajo de configuración, añadiremos el código de esta instancia al final del método \_\_init\_\_() en AlienInvasion. Con el tiempo, crearemos toda una flota de alienígenas, lo que supondrá bastante trabajo, así que crearemos un nuevo método de ayuda llamado \_create\_fleet().

El orden de los métodos en una clase no importa, siempre que haya cierta coherencia en su colocación. Yo colocaré \_create\_fleet() justo antes del método \_update\_screen(), pero cualquier lugar de AlienInvasion servirá. En primer lugar, importaremos la clase Alien.

Aquí están las declaraciones import actualizadas para *alien\_invasion.py*:

**alien\_invasion.py**

--snip--  
from bullet import Bullet  
from alien import Alien

Y aquí está el método \_\_init\_\_() actualizado:

**alien\_invasion.py**

def \_\_init\_\_(self):  
 --snip--  
 self.ship = Ship(self)  
 self.bullets = pygame.sprite.Group()  
 self.aliens = pygame.sprite.Group()  
  
 self.\_create\_fleet()

Creamos un grupo para contener la flota de extraterrestres, y llamamos a \_create\_fleet(), que estamos a punto de escribir.

Aquí está el nuevo método \_create\_fleet():

**alien\_invasion.py**

def \_create\_fleet(self):  
 """Create the fleet of aliens."""  
 # Make an alien.  
 alien = Alien(self)  
 self.aliens.add(alien)

En este método, creamos una instancia de Alien y la añadimos al grupo que contendrá la flota. El alienígena se colocará por defecto en la zona superior izquierda de la pantalla.

Para que aparezca el alienígena, tenemos que llamar al método draw() del grupo en \_update\_screen():

**alien\_invasion.py**

def \_update\_screen(self):  
 --snip--  
 self.ship.blitme()  
 self.aliens.draw(self.screen)  
  
 pygame.display.flip()

Cuando llamas a draw() en un grupo, Pygame dibuja cada elemento del grupo en la posición definida por su atributo rect. El método draw() requiere un argumento: una superficie sobre la que dibujar los elementos del grupo. [La Figura 13-2](#figure13-2) muestra el primer alienígena en la pantalla.



Figura 13-2: Aparece el primer alienígena.

Ahora que el primer alienígena aparece correctamente, escribiremos el código para dibujar una flota entera.

## Construir la flota alienígena

Para dibujar una flota, tenemos que averiguar cómo llenar de alienígenas la parte superior de la pantalla, sin abarrotar la ventana del juego. Hay varias formas de lograr este objetivo. Lo abordaremos añadiendo alienígenas por la parte superior de la pantalla, hasta que no quede espacio para un nuevo alienígena. Entonces repetiremos este proceso, siempre que tengamos espacio vertical suficiente para añadir una nueva fila.

### Crear una fila de alienígenas

Ahora estamos listos para generar una fila completa de alienígenas. Para crear una fila completa, primero crearemos un único alienígena, de modo que tengamos acceso a la anchura del alienígena. Colocaremos un alienígena en el lado izquierdo de la pantalla y seguiremos añadiendo alienígenas hasta que nos quedemos sin espacio:

**alien\_invasion.py**

def \_create\_fleet(self):  
 """Create the fleet of aliens."""  
 # Create an alien and keep adding aliens until there's no room left.  
 # Spacing between aliens is one alien width.  
 alien = Alien(self)  
 alien\_width = alien.rect.width  
  
❶ current\_x = alien\_width  
❷ while current\_x < (self.settings.screen\_width - 2 \* alien\_width):  
❸ new\_alien = Alien(self)  
❹ new\_alien.x = current\_x  
 new\_alien.rect.x = current\_x  
 self.aliens.add(new\_alien)  
❺ current\_x += 2 \* alien\_width

Obtenemos la anchura del alienígena del primer alienígena que hemos creado, y luego definimos una variable llamada current\_x ❶. Se refiere a la posición horizontal del siguiente robot que queremos colocar en la pantalla. Inicialmente la fijamos a una anchura de alienígena, para desplazar al primer alienígena de la flota desde el borde izquierdo de la pantalla.

A continuación, iniciamos el bucle while ❷; vamos a seguir añadiendo aliens *while* hay espacio suficiente para colocar uno. Para determinar si hay espacio para colocar otro alienígena, compararemos current\_x con algún valor máximo. Un primer intento de definir este bucle podría tener este aspecto:

while current\_x < self.settings.screen\_width:

Esto parece que podría funcionar, pero colocaría al último alienígena de la fila en el extremo derecho de la pantalla. Así que añadimos un pequeño margen en el lado derecho de la pantalla. Mientras haya espacio de al menos dos anchos de alienígena en el borde derecho de la pantalla, entramos en el bucle y añadimos otro alienígena a la flota.

Siempre que haya espacio horizontal suficiente para continuar el bucle, queremos hacer dos cosas: crear un alienígena en la posición correcta y definir la posición horizontal del siguiente alienígena de la fila. Creamos un alienígena y lo asignamos a new\_alien ❸. Luego fijamos la posición horizontal precisa al valor actual de current\_x ❹. También posicionamos el rect del alienígena en este mismo *x*-valor, y añadimos el nuevo alienígena al grupo self.aliens.

Por último, incrementamos el valor de current\_x ❺. Añadimos dos anchos de alienígena a la posición horizontal, para que pase el alienígena que acabamos de añadir y para dejar también algo de espacio entre los alienígenas. Python volverá a evaluar la condición al inicio del bucle while y decidirá si hay espacio para otro alienígena. Cuando no quede espacio, el bucle terminará, y deberíamos tener una fila completa de alienígenas.

Cuando ejecutes ahora *Alien Invasion*, deberías ver aparecer la primera fila de alienígenas, como en la [Figura 13-3](#figure13-3).

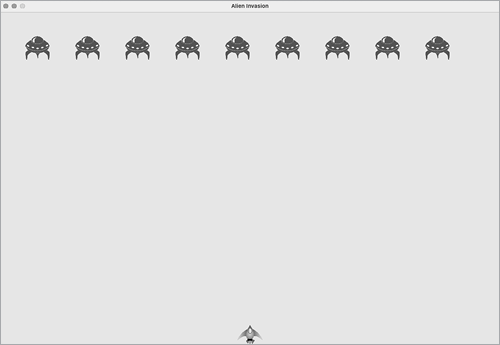


Figura13-3: La primera fila de alienígenas

## Nota

No siempre es obvio cómo construir exactamente un bucle como el que se muestra en esta sección. Una cosa buena de la programación es que tus ideas iniciales sobre cómo abordar un problema como éste no tienen por qué ser correctas. Es perfectamente razonable empezar un bucle como éste con los alienígenas situados demasiado a la derecha, y luego modificar el bucle hasta que tengas una cantidad adecuada de espacio en la pantalla.

### Refactorización de \_create\_fleet()

Si el código que hemos escrito hasta ahora fuera todo lo que necesitamos para crear una flota, probablemente dejaríamos \_create\_fleet() tal cual. Pero tenemos más trabajo que hacer, así que vamos a limpiar un poco el método. Añadiremos un nuevo método auxiliar, \_create\_alien(), y lo llamaremos desde \_create\_fleet():

**alien\_invasion.py**

def \_create\_fleet(self):  
 --snip--  
 while current\_x < (self.settings.screen\_width - 2 \* alien\_width):  
 self.\_create\_alien(current\_x)  
 current\_x += 2 \* alien\_width  
  
❶ def \_create\_alien(self, x\_position):  
 """Create an alien and place it in the row."""  
 new\_alien = Alien(self)  
 new\_alien.x = x\_position  
 new\_alien.rect.x = x\_position  
 self.aliens.add(new\_alien)

El método \_create\_alien() requiere un parámetro además de self: el -valor *x* que especifica dónde debe colocarse el ❶ alienígena . El código del cuerpo de \_create\_alien() es el mismo código que había en \_create\_fleet(), salvo que utilizamos el nombre de parámetro x\_position en lugar de current\_x. Esta refactorización facilitará la adición de nuevas filas y la creación de una flota completa.

### Añadir filas

Para terminar la flota, seguiremos añadiendo más filas hasta que nos quedemos sin espacio. Utilizaremos un bucle anidado: envolveremos otro bucle while alrededor del actual. El bucle interior colocará a los alienígenas horizontalmente en una fila centrándose en los valores *x* de los alienígenas. El bucle exterior colocará a los alienígenas verticalmente centrándose en los valores *y*. Dejaremos de añadir filas cuando nos acerquemos a la parte inferior de la pantalla, dejando espacio suficiente para la nave y algo de espacio para empezar a disparar a los alienígenas.

He aquí cómo anidar los dos bucles while en \_create\_fleet():

def \_create\_fleet(self):  
 """Create the fleet of aliens."""  
 # Create an alien and keep adding aliens until there's no room left.  
 # Spacing between aliens is one alien width and one alien height.  
 alien = Alien(self)  
❶ alien\_width, alien\_height = alien.rect.size  
  
❷ current\_x, current\_y = alien\_width, alien\_height  
❸ while current\_y < (self.settings.screen\_height - 3 \* alien\_height):  
 while current\_x < (self.settings.screen\_width - 2 \* alien\_width):  
❹ self.\_create\_alien(current\_x, current\_y)  
 current\_x += 2 \* alien\_width  
  
❺ # Finished a row; reset x value, and increment y value.  
 current\_x = alien\_width  
 current\_y += 2 \* alien\_height

Necesitaremos saber la altura del alienígena para colocar las filas, así que cogemos la anchura y la altura del alienígena utilizando el atributo size de un alienígena rect ❶. El atributo size de un rectes una tupla que contiene su anchura y altura.

A continuación, fijamos los valores iniciales *x*- y *y*- para la colocación del primer alienígena de la flota ❷. Lo colocamos a una anchura de alienígena hacia dentro desde la izquierda y a una altura de alienígena hacia abajo desde arriba. Después definimos el bucle while que controla cuántas filas se colocan en la pantalla ❸. Mientras el valor *y* de la siguiente fila sea menor que la altura de la pantalla, menos tres alturas de alienígena, seguiremos añadiendo filas. (Si esto no deja el espacio adecuado, podemos ajustarlo más tarde).

Llamamos a \_create\_alien(), y le pasamos el -valor *y* así como su -posición *x*❹. Modificaremos \_create\_alien() dentro de un momento.

Fíjate en la sangría de las dos últimas líneas de código ❺. Están dentro del bucle exterior while, pero fuera del bucle interior while. Este bloque se ejecuta una vez finalizado el bucle interno; se ejecuta una vez después de crear cada fila. Después de añadir cada fila, reiniciamos el valor de current\_x para que el primer alienígena de la siguiente fila se coloque en la misma posición que el primer alienígena de las filas anteriores. A continuación, añadimos dos alturas de alienígena al valor actual de current\_y, para que la siguiente fila se sitúe más abajo en la pantalla. La sangría es realmente importante aquí; si no ves la flota correcta cuando ejecutes *alien\_invasion.py* al final de esta sección, comprueba la sangría de todas las líneas en estos bucles anidados.

Tenemos que modificar \_create\_alien() para fijar correctamente la posición vertical del alienígena:

def \_create\_alien(self, x\_position, y\_position):  
 """Create an alien and place it in the fleet."""  
 new\_alien = Alien(self)  
 new\_alien.x = x\_position  
 new\_alien.rect.x = x\_position  
 new\_alien.rect.y = y\_position  
 self.aliens.add(new\_alien)

Modificamos la definición del método para que acepte el -valor *y* para el nuevo alienígena, y fijamos la posición vertical del rect en el cuerpo del método.

Cuando ejecutes ahora el juego, deberías ver una flota completa de alienígenas, como se muestra en la [Figura 13-4](#figure13-4).

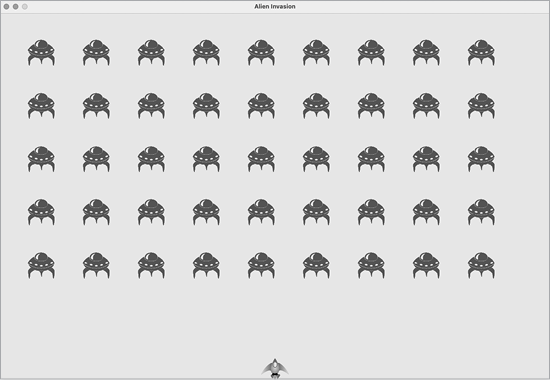


Figura13-4: Aparece la flota completa.

En la siguiente sección, ¡haremos que la flota se mueva!

## Inténtalo tú mismo

13-1. Estrellas: Busca la imagen de una estrella. Haz que aparezca una cuadrícula de estrellas en la pantalla.

13-2. Estrellas mejores: Puedes hacer un patrón de estrellas más realista introduciendo aleatoriedad al colocar cada estrella. Recuerda del Capítulo 9 que puedes obtener un número aleatorio de este modo:

from random import randint  
random\_number = randint(-10, 10)

Este código devuelve un número entero aleatorio entre -10 y 10. Utilizando tu código en el Ejercicio 13-1, ajusta la posición de cada estrella en una cantidad aleatoria.

## Hacer que la flota se mueva

Ahora vamos a hacer que la flota de alienígenas se mueva hacia la derecha a través de la pantalla hasta que llegue al borde, y luego hagamos que baje una cantidad determinada y se mueva en la otra dirección. Continuaremos este movimiento hasta que todos los alienígenas hayan sido derribados, uno colisione con la nave o uno llegue al fondo de la pantalla. Empecemos haciendo que la flota se mueva hacia la derecha.

### Mover a los alienígenas hacia la derecha

Para mover a los alienígenas, utilizaremos un método update() en *alien.py*, que llamaremos para cada alienígena del grupo de alienígenas. En primer lugar, añade un ajuste para controlar la velocidad de cada alienígena:

**settings.py**

def \_\_init\_\_(self):  
 --snip--  
 # Alien settings  
 self.alien\_speed = 1.0

A continuación, utiliza este ajuste para implementar update() en *alien.py*:

**alien.py**

def \_\_init\_\_(self, ai\_game):  
 """Initialize the alien and set its starting position."""  
 super().\_\_init\_\_()  
 self.screen = ai\_game.screen  
 self.settings = ai\_game.settings  
 --snip--  
  
 def update(self):  
 """Move the alien to the right."""  
❶ self.x += self.settings.alien\_speed  
❷ self.rect.x = self.x

Creamos un parámetro settings en \_\_init\_\_() para poder acceder a la velocidad del alienígena en update(). Cada vez que actualizamos la posición de un robot, lo desplazamos a la derecha de la cantidad almacenada en alien\_speed. Seguimos la posición exacta del alienígena con el atributo self.x, que puede contener valores flotantes ❶. A continuación, utilizamos el valor de self.x para actualizar la posición de rect ❷ del alienígena .

En el bucle principal while, ya tenemos llamadas para actualizar las posiciones de la nave y de la bala. Ahora añadiremos una llamada para actualizar también la posición de cada alienígena:

**alien\_invasion.py**

while True:  
 self.\_check\_events()  
 self.ship.update()  
 self.\_update\_bullets()  
 self.\_update\_aliens()  
 self.\_update\_screen()  
 self.clock.tick(60)

Vamos a escribir código para gestionar el movimiento de la flota, así que creamos un nuevo método llamado \_update\_aliens(). Actualizamos las posiciones de los alienígenas después de que se hayan actualizado las balas, porque pronto comprobaremos si alguna bala ha alcanzado a algún alienígena.

El lugar donde coloques este método en el módulo no es crítico. Pero para mantener el código organizado, lo colocaré justo después de \_update\_bullets() para que coincida con el orden de las llamadas al método en el bucle while. Aquí tienes la primera versión de \_update\_aliens():

**alien\_invasion.py**

def \_update\_aliens(self):  
 """Update the positions of all aliens in the fleet."""  
 self.aliens.update()

Utilizamos el método update() en el grupo aliens, que llama al método update() de cada extraterrestre. Cuando ejecutes ahora *Alien Invasion*, deberías ver que la flota se mueve hacia la derecha y desaparece por el lateral de la pantalla.

### Crear los ajustes para la dirección de la flota

Ahora crearemos los ajustes que harán que la flota se mueva hacia abajo en la pantalla y hacia la izquierda cuando toque el borde derecho de la pantalla. He aquí cómo implementar este comportamiento:

**settings.py**

# Alien settings  
 self.alien\_speed = 1.0  
 self.fleet\_drop\_speed = 10  
 # fleet\_direction of 1 represents right; -1 represents left.  
 self.fleet\_direction = 1

El ajuste fleet\_drop\_speed controla la rapidez con la que la flota desciende por la pantalla cada vez que un alienígena alcanza cualquiera de los bordes. Es útil separar esta velocidad de la velocidad horizontal de los alienígenas para poder ajustar las dos velocidades de forma independiente.

Para implementar el ajuste fleet\_direction, podríamos utilizar un valor de texto como 'left' o 'right', pero acabaríamos con las sentencias if-elif comprobando la dirección de la flota. En lugar de eso, como sólo tenemos que tratar con dos direcciones, utilicemos los valores 1 y -1 y cambiemos entre ellos cada vez que la flota cambie de dirección. (Utilizar números también tiene sentido porque moverse a la derecha implica sumar al valor de la coordenada *x* de cada alienígena, y moverse a la izquierda implica restar al valor de la coordenada *x* de cada alienígena).

### Comprobar si un alienígena ha tocado el borde

Necesitamos un método para comprobar si un alienígena está en uno de los bordes, y tenemos que modificar update() para permitir que cada alienígena se mueva en la dirección adecuada. Este código forma parte de la clase Alien:

**alien.py**

def check\_edges(self):  
 """Return True if alien is at edge of screen."""  
 screen\_rect = self.screen.get\_rect()  
❶ return (self.rect.right >= screen\_rect.right) or (self.rect.left <= 0)  
  
 def update(self):  
 """Move the alien right or left."""  
❷ self.x += self.settings.alien\_speed \* self.settings.fleet\_direction  
 self.rect.x = self.x

Podemos llamar al nuevo método check\_edges() sobre cualquier alienígena para ver si está en el borde izquierdo o derecho. El alienígena está en el borde derecho si el atributo right de su rect es mayor o igual que el atributo right del rect de la pantalla. Está en el borde izquierdo si su valor left es menor o igual que 0 ❶. En lugar de poner esta prueba condicional en un bloque if, ponemos la prueba directamente en la sentencia return. Este método devolverá True si el alienígena está en el borde derecho o izquierdo, y False si no está en ninguno de los bordes.

Modificamos el método update() para permitir el movimiento hacia la izquierda o la derecha multiplicando la velocidad del alienígena por el valor de fleet\_direction ❷. Si fleet\_direction es 1, el valor de alien\_speed se sumará a la posición actual del alienígena, moviéndolo hacia la derecha; si fleet\_direction es -1, el valor se restará de la posición del alienígena, moviéndolo hacia la izquierda.

### Soltar la flota y cambiar de dirección

Cuando un alienígena alcanza el borde, toda la flota debe descender y cambiar de dirección. Por lo tanto, tenemos que añadir algo de código a AlienInvasion porque ahí es donde comprobaremos si hay algún alienígena en el borde izquierdo o derecho. Lo haremos escribiendo los métodos \_check\_fleet\_edges() y \_change\_fleet\_direction(), y modificando \_update\_aliens(). Pondré estos nuevos métodos después de \_create\_alien(), pero, de nuevo, la ubicación de estos métodos en la clase no es crítica.

**alien\_invasion.py**

def \_check\_fleet\_edges(self):  
 """Respond appropriately if any aliens have reached an edge."""  
❶ for alien in self.aliens.sprites():  
 if alien.check\_edges():  
❷ self.\_change\_fleet\_direction()  
 break  
  
 def \_change\_fleet\_direction(self):  
 """Drop the entire fleet and change the fleet's direction."""  
 for alien in self.aliens.sprites():  
❸ alien.rect.y += self.settings.fleet\_drop\_speed  
 self.settings.fleet\_direction \*= -1

En \_check\_fleet\_edges(), hacemos un bucle a través de la flota y llamamos a check\_edges() en cada ❶ alienígena . Si check\_edges() devuelve True, sabemos que un alienígena está en un borde y que toda la flota necesita cambiar de dirección; así que llamamos a \_change\_fleet\_direction() y salimos del bucle ❷. En \_change\_fleet\_direction(), hacemos un bucle a través de todos los alienígenas y soltamos a cada uno utilizando el parámetro fleet\_drop\_speed ❸; luego cambiamos el valor de fleet\_direction multiplicando su valor actual por -1. La línea que cambia la dirección de la flota no forma parte del bucle for. Queremos cambiar la posición vertical de cada alienígena, pero sólo queremos cambiar la dirección de la flota una vez.

Estos son los cambios en \_update\_aliens():

**alien\_invasion.py**

def \_update\_aliens(self):  
 """Check if the fleet is at an edge, then update positions."""  
 self.\_check\_fleet\_edges()  
 self.aliens.update()

Hemos modificado el método llamando a \_check\_fleet\_edges() antes de actualizar la posición de cada alienígena.

Ahora, al ejecutar el juego, la flota debería moverse de un lado a otro de los bordes de la pantalla y descender cada vez que toque un borde. Ahora podemos empezar a derribar alienígenas y vigilar si alguno choca contra la nave o llega al fondo de la pantalla.

## Inténtalo tú mismo

13-3. Gotas de lluvia: Busca una imagen de una gota de lluvia y crea una cuadrícula de gotas de lluvia. Haz que las gotas de lluvia caigan hacia la parte inferior de la pantalla hasta que desaparezcan.

13-4. Lluvia constante: Modifica tu código del Ejercicio 13-3 para que cuando desaparezca una fila de gotas de lluvia de la parte inferior de la pantalla, aparezca una nueva fila en la parte superior de la pantalla y empiece a caer.

## Disparando a los alienígenas

Hemos construido nuestra nave y una flota de alienígenas, pero cuando las balas alcanzan a los alienígenas, simplemente los atraviesan porque no estamos comprobando si hay colisiones. En programación de juegos, *collisions* se produce cuando los elementos del juego se solapan. Para que las balas derriben a los alienígenas, utilizaremos la función sprite.groupcollide() para buscar colisiones entre miembros de dos grupos.

### Detectar colisiones de balas

Queremos saber inmediatamente cuándo una bala alcanza a un alienígena, para poder hacer que desaparezca en cuanto sea alcanzado. Para ello, buscaremos colisiones inmediatamente después de actualizar la posición de todas las balas.

La función sprite.groupcollide() compara los rects de cada elemento de un grupo con los rects de cada elemento de otro grupo. En este caso, compara el rect de cada bala con el rect de cada alienígena y devuelve un diccionario que contiene las balas y los alienígenas que han colisionado. Cada clave del diccionario será una bala, y el valor correspondiente será el alienígena alcanzado. (También utilizaremos este diccionario cuando implementemos un sistema de puntuación en el Capítulo 14).

Añade el siguiente código al final de \_update\_bullets() para comprobar las colisiones entre balas y alienígenas:

**alien\_invasion.py**

def \_update\_bullets(self):  
 """Update position of bullets and get rid of old bullets."""  
 --snip--  
  
 # Check for any bullets that have hit aliens.  
 # If so, get rid of the bullet and the alien.  
 collisions = pygame.sprite.groupcollide(  
 self.bullets, self.aliens, True, True)

El nuevo código que hemos añadido compara las posiciones de todas las balas en self.bullets y de todos los alienígenas en self.aliens, e identifica los que se solapan. Siempre que las rects de una bala y un alienígena se solapen, groupcollide() añade un par clave-valor al diccionario que devuelve. Los dos argumentos True indican a Pygame que elimine las balas y los alienígenas que se hayan solapado. (Para hacer una bala muy potente que pueda viajar hasta la parte superior de la pantalla, destruyendo a todos los alienígenas que encuentre en su camino, podrías establecer el primer argumento booleano en False y mantener el segundo argumento booleano en True. Los alienígenas alcanzados desaparecerían, pero todas las balas seguirían activas hasta que desaparecieran de la parte superior de la pantalla).

Cuando ejecutes ahora *Alien Invasion*, los alienígenas alcanzados deberían desaparecer. [La Figura 13-5](#figure13-5) muestra una flota que ha sido parcialmente derribada.

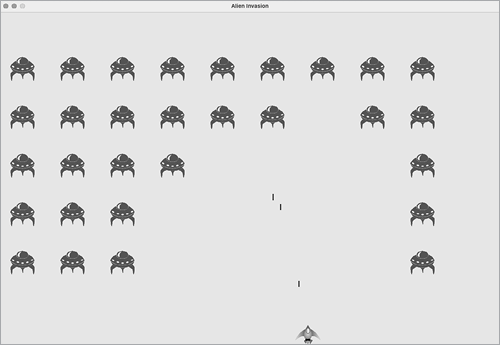


Figura13-5: ¡Ya podemos disparar a los alienígenas!

### Hacer balas más grandes para probarlas

Puedes probar muchas funciones de *Alien Invasion* simplemente ejecutando el juego, pero algunas funciones son tediosas de probar en la versión normal del juego. Por ejemplo, es mucho trabajo derribar a todos los alienígenas de la pantalla varias veces para probar si tu código responde correctamente a una flota vacía.

Para probar determinadas funciones, puedes cambiar ciertos ajustes del juego para centrarte en un área concreta. Por ejemplo, puedes encoger la pantalla para que haya menos alienígenas a los que derribar o aumentar la velocidad de las balas y darte muchas balas a la vez.

Mi cambio favorito para probar *Alien Invasion* es utilizar balas muy anchas que permanezcan activas incluso después de haber alcanzado a un alienígena (ver [Figura 13-6](#figure13-6)). Prueba a poner bullet\_width a 300, o incluso a 3.000, ¡para ver lo rápido que puedes derribar a la flota!

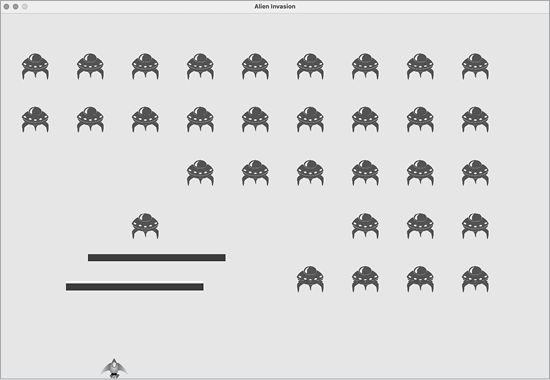


Figura13-6: Las balas extra potentes hacen que algunos aspectos del juego sean más fáciles de probar.

Cambios como éstos te ayudarán a probar el juego con más eficacia y posiblemente te darán ideas para dar a los jugadores poderes extra. Sólo recuerda restablecer los ajustes a la configuración normal cuando hayas terminado de probar una característica.

### Repoblar la flota

Una característica clave de *Alien Invasion* es que los alienígenas son implacables: cada vez que la flota es destruida, debe aparecer una nueva flota.

Para hacer que aparezca una nueva flota de alienígenas tras la destrucción de una flota, primero comprobamos si el grupo aliens está vacío. Si lo está, hacemos una llamada a \_create\_fleet(). Realizaremos esta comprobación al final de \_update\_bullets(), porque es ahí donde se destruyen los alienígenas individuales.

**alien\_invasion.py**

def \_update\_bullets(self):  
 --snip--  
❶ if not self.aliens:  
 # Destroy existing bullets and create new fleet.  
❷ self.bullets.empty()  
 self.\_create\_fleet()

Comprobamos si el grupo aliens está vacío ❶. Un grupo vacío se evalúa en False, por lo que es una forma sencilla de comprobar si el grupo está vacío. Si lo está, nos deshacemos de los sprites existentes utilizando el método empty(), que elimina todos los sprites restantes de un grupo ❷. También llamamos a \_create\_fleet(), que vuelve a llenar la pantalla de alienígenas.

Ahora aparece una nueva flota en cuanto destruyes la flota actual.

### Acelerar las balas

Si has intentado disparar a los alienígenas en el estado actual del juego, puede que te des cuenta de que las balas no viajan a la mejor velocidad para el juego. Pueden ser un poco demasiado lentas o un poco demasiado rápidas. En este punto, puedes modificar los ajustes para que la jugabilidad sea más interesante. Ten en cuenta que el juego se hará progresivamente más rápido, así que no lo hagas demasiado rápido al principio.

Modificamos la velocidad de las balas ajustando el valor de bullet\_speed en *settings.py*. En mi sistema, ajustaré el valor de bullet\_speed a 2,5, para que las balas viajen un poco más rápido:

**settings.py**

# Bullet settings  
 self.bullet\_speed = 2.5  
 self.bullet\_width = 3  
 --snip--

El mejor valor para este ajuste depende de tu experiencia en el juego, así que encuentra un valor que funcione para ti. También puedes ajustar otros parámetros.

### Refactorización de \_actualizar\_balas()

Vamos a refactorizar \_update\_bullets() para que no haga tantas tareas diferentes. Moveremos el código para tratar las colisiones bala-alien a un método aparte:

**alien\_invasion.py**

def \_update\_bullets(self):  
 --snip--  
 # Get rid of bullets that have disappeared.  
 for bullet in self.bullets.copy():  
 if bullet.rect.bottom <= 0:  
 self.bullets.remove(bullet)  
  
 self.\_check\_bullet\_alien\_collisions()  
  
 def \_check\_bullet\_alien\_collisions(self):  
 """Respond to bullet-alien collisions."""  
 # Remove any bullets and aliens that have collided.  
 collisions = pygame.sprite.groupcollide(  
 self.bullets, self.aliens, True, True)  
  
 if not self.aliens:  
 # Destroy existing bullets and create new fleet.  
 self.bullets.empty()  
 self.\_create\_fleet()

Hemos creado un nuevo método, \_check\_bullet\_alien\_collisions(), para buscar colisiones entre balas y alienígenas y responder adecuadamente si toda la flota ha sido destruida. Hacerlo así evita que \_update\_bullets() crezca demasiado y simplifica el desarrollo posterior.

## Pruébalo tú mismo

13-5. Tirador Lateral Parte 2: Hemos avanzado mucho desde el Ejercicio 12-6, *Sideways Shooter*. Para este ejercicio, intenta desarrollar *Sideways Shooter* hasta el mismo punto al que hemos llevado *Alien Invasion*. Añade una flota de alienígenas y haz que se muevan lateralmente hacia la nave. O bien, escribe un código que coloque a los alienígenas en posiciones aleatorias a lo largo del lado derecho de la pantalla y luego los envíe hacia la nave. Escribe también un código que haga desaparecer a los alienígenas cuando sean alcanzados.

## Finalizar el juego

¿Qué hay de divertido y desafiante en jugar a un juego que no puedes perder? Si el jugador no derriba la flota lo bastante rápido, haremos que los alienígenas destruyan la nave cuando hagan contacto. Al mismo tiempo, limitaremos el número de naves que puede utilizar el jugador, y destruiremos la nave cuando un alienígena llegue al fondo de la pantalla. El juego terminará cuando el jugador haya utilizado todas sus naves.

### Detección de colisiones entre naves alienígenas

Empezaremos comprobando si hay colisiones entre los alienígenas y la nave, para poder responder adecuadamente cuando un alienígena choque contra ella. Comprobaremos las colisiones alienígena-nave inmediatamente después de actualizar la posición de cada alienígena en AlienInvasion:

**alien\_invasion.py**

def \_update\_aliens(self):  
 --snip--  
 self.aliens.update()  
  
 # Look for alien-ship collisions.  
❶ if pygame.sprite.spritecollideany(self.ship, self.aliens):  
❷ print("Ship hit!!!")

La función spritecollideany() toma dos argumentos: un sprite y un grupo. La función busca cualquier miembro del grupo que haya colisionado con el sprite y deja de recorrer el grupo en cuanto encuentra un miembro que haya colisionado con el sprite. En este caso, recorre el grupo aliens y devuelve el primer extraterrestre que encuentre que haya colisionado con ship.

Si no se produce ninguna colisión, spritecollideany() devuelve None y el bloque if no se ejecuta ❶. Si encuentra un alienígena que ha colisionado con la nave, devuelve ese alienígena y el bloque if se ejecuta: imprime Ship hit!!! ❷. Cuando un alienígena choque con la nave, tendremos que realizar una serie de tareas: eliminar todos los alienígenas y balas restantes, recentrar la nave y crear una nueva flota. Antes de escribir código para hacer todo esto, queremos saber si nuestro enfoque para detectar colisiones alienígena-nave funciona correctamente. Escribir una llamada a print() es una forma sencilla de asegurarnos de que detectamos estas colisiones correctamente.

Ahora, cuando ejecutes *Alien Invasion,*, debería aparecer el mensaje *Ship hit!!!* en el terminal cada vez que un alienígena choque contra la nave. Cuando estés probando esta función, ajusta fleet\_drop\_speed a un valor más alto, como 50 o 100, para que los alienígenas alcancen tu nave más rápidamente.

### Respuesta a las colisiones entre naves alienígenas

Ahora tenemos que averiguar qué ocurrirá exactamente cuando un alienígena colisione con la nave. En lugar de destruir la instancia ship y crear una nueva, contaremos cuántas veces ha sido golpeada la nave mediante las estadísticas de seguimiento del juego. Las estadísticas de seguimiento también serán útiles para la puntuación.

Escribamos una nueva clase, GameStats, para realizar el seguimiento de las estadísticas del juego, y guardémosla como *game\_stats.py*:

**game\_stats.py**

class GameStats:  
 """Track statistics for Alien Invasion."""  
  
 def \_\_init\_\_(self, ai\_game):  
 """Initialize statistics."""  
 self.settings = ai\_game.settings  
❶ self.reset\_stats()  
  
 def reset\_stats(self):  
 """Initialize statistics that can change during the game."""  
 self.ships\_left = self.settings.ship\_limit

Crearemos una instancia de GameStats para todo el tiempo que se ejecute *Alien Invasion*, pero necesitaremos restablecer algunas estadísticas cada vez que el jugador comience una nueva partida. Para ello, inicializaremos la mayoría de las estadísticas en el método reset\_stats(), en lugar de hacerlo directamente en \_\_init\_\_(). Llamaremos a este método desde \_\_init\_\_() para que las estadísticas se establezcan correctamente cuando se cree por primera vez la instancia GameStats ❶. Pero también podremos llamar a reset\_stats() cada vez que el jugador empiece una nueva partida. Ahora mismo sólo tenemos una estadística, ships\_left, cuyo valor cambiará a lo largo de la partida.

El número de naves con las que comienza el jugador debe almacenarse en *settings.py* como ship\_limit:

**settings.py**

# Ship settings  
 self.ship\_speed = 1.5  
 self.ship\_limit = 3

También tenemos que hacer algunos cambios en *alien\_invasion.py* para crear una instancia de GameStats. En primer lugar, actualizaremos las sentencias import de la parte superior del archivo:

**alien\_invasion.py**

import sys  
from time import sleep  
  
import pygame  
  
from settings import Settings  
from game\_stats import GameStats  
from ship import Ship  
--snip--

Importamos la función sleep() del módulo time de la biblioteca estándar de Python, para que podamos pausar el juego un momento cuando la nave sea alcanzada. También importamos GameStats.

Crearemos una instancia de GameStats en \_\_init\_\_():

**alien\_invasion.py**

def \_\_init\_\_(self):  
 --snip--  
 self.screen = pygame.display.set\_mode(  
 (self.settings.screen\_width, self.settings.screen\_height))  
 pygame.display.set\_caption("Alien Invasion")  
  
 # Create an instance to store game statistics.  
 self.stats = GameStats(self)  
  
 self.ship = Ship(self)  
 --snip--

Hacemos la instancia después de crear la ventana del juego, pero antes de definir otros elementos del juego, como la nave.

Cuando un alien impacte contra la nave, restaremos 1 al número de naves restantes, destruiremos todos los alienígenas y balas existentes, crearemos una nueva flota y recolocaremos la nave en el centro de la pantalla. También pausaremos el juego un momento para que el jugador pueda darse cuenta de la colisión y reagruparse antes de que aparezca una nueva flota.

Pongamos la mayor parte de este código en un nuevo método llamado \_ship\_hit(). Llamaremos a este método desde \_update\_aliens() cuando un alien impacte contra la nave:

**alien\_invasion.py**

def \_ship\_hit(self):  
 """Respond to the ship being hit by an alien."""  
 # Decrement ships\_left.  
❶ self.stats.ships\_left -= 1  
  
 # Get rid of any remaining bullets and aliens.  
❷ self.bullets.empty()  
 self.aliens.empty()  
  
 # Create a new fleet and center the ship.  
❸ self.\_create\_fleet()  
 self.ship.center\_ship()  
  
 # Pause.  
❹ sleep(0.5)

El nuevo método \_ship\_hit() coordina la respuesta cuando un alienígena choca contra una nave. Dentro de \_ship\_hit(), el número de naves restantes se reduce en 1 ❶, tras lo cual vaciamos los grupos bullets y aliens ❷.

A continuación, creamos una nueva flota y centramos la nave ❸.(Dentro de un momento añadiremos el método center\_ship() a Ship.) A continuación, añadimos una pausa después de que se hayan realizado las actualizaciones de todos los elementos del juego, pero antes de que se haya dibujado ningún cambio en la pantalla, para que el jugador pueda ver que su nave ha sido golpeada ❹. La llamada a sleep() detiene la ejecución del programa durante medio segundo, el tiempo suficiente para que el jugador vea que el alienígena ha golpeado la nave. Cuando finaliza la función sleep(), la ejecución del código pasa al método \_update\_screen(), que dibuja la nueva flota en la pantalla.

En \_update\_aliens(), sustituimos la llamada a print() por una llamada a \_ship\_hit() cuando un alienígena golpea la nave:

**alien\_invasion.py**

def \_update\_aliens(self):  
 --snip--  
 if pygame.sprite.spritecollideany(self.ship, self.aliens):  
 self.\_ship\_hit()

Este es el nuevo método center\_ship(), que pertenece a *ship.py*:

**ship.py**

def center\_ship(self):  
 """Center the ship on the screen."""  
 self.rect.midbottom = self.screen\_rect.midbottom  
 self.x = float(self.rect.x)

Centramos la nave del mismo modo que en \_\_init\_\_(). Después de centrarla, reseteamos el atributo self.x, que nos permite seguir la posición exacta de la nave.

## Nota

Observa que nunca hacemos más de una nave; sólo hacemos una instancia de nave para toda la partida y la volvemos a centrar siempre que la nave haya sido golpeada. La estadística ships\_left nos indicará cuándo el jugador se ha quedado sin naves.

Ejecuta el juego, dispara a unos cuantos alienígenas y deja que un alienígena golpee la nave. El juego debería hacer una pausa, y debería aparecer una nueva flota con la nave centrada de nuevo en la parte inferior de la pantalla.

### Alienígenas que llegan al fondo de la pantalla

Si un alienígena llega a la parte inferior de la pantalla, haremos que el juego responda del mismo modo que cuando un alienígena golpea la nave. Para comprobar cuándo ocurre esto, añade un nuevo método en *alien\_invasion.py*:

**alien\_invasion.py**

def \_check\_aliens\_bottom(self):  
 """Check if any aliens have reached the bottom of the screen."""  
 for alien in self.aliens.sprites():  
❶ if alien.rect.bottom >= self.settings.screen\_height:  
 # Treat this the same as if the ship got hit.  
 self.\_ship\_hit()  
 break

El método \_check\_aliens\_bottom() comprueba si algún alienígena ha llegado al fondo de la pantalla. Un alienígena llega al fondo cuando su valor rect.bottom es mayor o igual que la altura de la pantalla ❶. Si un alienígena llega al fondo, llamamos a \_ship\_hit(). Si un alienígena llega al fondo, no es necesario comprobar el resto, así que salimos del bucle tras llamar a \_ship\_hit().

Llamaremos a este método desde \_update\_aliens():

**alien\_invasion.py**

def \_update\_aliens(self):  
 --snip--  
 # Look for alien-ship collisions.  
 if pygame.sprite.spritecollideany(self.ship, self.aliens):  
 self.\_ship\_hit()  
  
 # Look for aliens hitting the bottom of the screen.  
 self.\_check\_aliens\_bottom()

Llamamos a \_check\_aliens\_bottom() después de actualizar las posiciones de todos los alienígenas y después de buscar colisiones alienígena-nave. Ahora aparecerá una nueva flota cada vez que la nave sea alcanzada por un alienígena o que un alienígena llegue al fondo de la pantalla.

### ¡Game Over!

*Alien Invasion* parece más completo ahora, pero el juego nunca termina. El valor de ships\_left sólo se vuelve cada vez más negativo. Añadamos una bandera game\_active, para que podamos terminar el juego cuando el jugador se quede sin naves. Estableceremos esta bandera al final del método \_\_init\_\_() en AlienInvasion:

**alien\_invasion.py**

def \_\_init\_\_(self):  
 --snip--  
 # Start Alien Invasion in an active state.  
 self.game\_active = True

Ahora añadimos código a \_ship\_hit() que establece game\_active en False cuando el jugador haya agotado todas sus naves:

**alien\_invasion.py**

def \_ship\_hit(self):  
 """Respond to ship being hit by alien."""  
 if self.stats.ships\_left > 0:  
 # Decrement ships\_left.  
 self.stats.ships\_left -= 1  
 --snip--  
 # Pause.  
 sleep(0.5)  
 else:  
 self.game\_active = False

La mayor parte de \_ship\_hit() no se modifica. Hemos trasladado todo el código existente a un bloque if, que comprueba que al jugador le queda al menos una nave. Si es así, creamos una nueva flota, hacemos una pausa y seguimos adelante. Si al jugador no le queda ninguna nave, ponemos game\_active en False.

### Identificar cuándo deben ejecutarse las partes del juego

Tenemos que identificar las partes del juego que deben ejecutarse siempre y las partes que deben ejecutarse sólo cuando el juego está activo:

**alien\_invasion.py**

def run\_game(self):  
 """Start the main loop for the game."""  
 while True:  
 self.\_check\_events()  
  
 if self.game\_active:  
 self.ship.update()  
 self.\_update\_bullets()  
 self.\_update\_aliens()  
  
 self.\_update\_screen()  
 self.clock.tick(60)

En el bucle principal, siempre tenemos que llamar a \_check\_events(), aunque el juego esté inactivo. Por ejemplo, seguimos necesitando saber si el usuario pulsa Q para salir del juego o si hace clic en el botón para cerrar la ventana. También seguimos actualizando la pantalla para poder hacer cambios en ella mientras esperamos a ver si el jugador decide empezar una nueva partida. El resto de las llamadas a funciones sólo tienen que ocurrir cuando el juego está activo, porque cuando el juego está inactivo, no necesitamos actualizar las posiciones de los elementos del juego.

Ahora, cuando juegues a *Alien Invasion*, el juego debería congelarse cuando hayas utilizado todas tus naves.

## Pruébalo tú mismo

13-6. Fin del juego: En *Sideways Shooter*, lleva la cuenta del número de veces que la nave es alcanzada y del número de veces que un alien es alcanzado por la nave. Decide una condición apropiada para terminar la partida, y detén el juego cuando se produzca esta situación.

## Resumen

En este capítulo has aprendido a añadir un gran número de elementos idénticos a un juego creando una flota de alienígenas. Utilizaste bucles anidados para crear una cuadrícula de elementos, e hiciste que un gran conjunto de elementos del juego se movieran llamando al método update() de cada elemento. Aprendiste a controlar la dirección de los objetos en la pantalla y a responder a situaciones concretas, como cuando la flota alcanza el borde de la pantalla. Detectaste y respondiste a las colisiones cuando las balas golpeaban a los alienígenas y los alienígenas golpeaban a la nave. También aprendiste a seguir las estadísticas de una partida y a utilizar una bandera game\_active para determinar cuándo termina la partida.

En el próximo y último capítulo de este proyecto, añadiremos un botón Jugar para que el jugador pueda elegir cuándo empezar su primera partida y si quiere volver a jugar cuando termine la partida. Aceleraremos el juego cada vez que el jugador derribe toda la flota, y añadiremos un sistema de puntuación. ¡El resultado final será un juego totalmente jugable!