# 14 Puntuación



En este capítulo, terminaremos de construir *Alien Invasion*. Añadiremos un botón Reproducir para iniciar el juego cuando se solicite y para reiniciarlo cuando termine. También cambiaremos el juego para que se acelere cuando el jugador suba de nivel, e implementaremos un sistema de puntuación. Al final del capítulo, sabrás lo suficiente para empezar a escribir juegos que aumenten su dificultad a medida que el jugador progresa y que incorporen sistemas completos de puntuación.

## Añadir el botón Jugar

En esta sección, añadiremos un botón Jugar que aparecerá antes de que empiece una partida y reaparecerá cuando ésta termine para que el jugador pueda volver a jugar.

Ahora mismo, el juego comienza en cuanto ejecutas *alien\_invasion.py*. Empecemos el juego en un estado inactivo y luego pidamos al jugador que pulse el botón Jugar para empezar. Para ello, modifica el método \_\_init\_\_() de AlienInvasion:

**alien\_invasion.py**

def \_\_init\_\_(self):  
 """Initialize the game, and create game resources."""  
 pygame.init()  
 --snip--  
  
 # Start Alien Invasion in an inactive state.  
 self.game\_active = False

Ahora el juego debe comenzar en un estado inactivo, sin que el jugador pueda iniciarlo hasta que hagamos un botón Jugar.

### Crear una clase de botón

Como Pygame no tiene un método incorporado para crear botones, escribiremos una clase Button para crear un rectángulo relleno con una etiqueta. Puedes utilizar este código para crear cualquier botón en un juego. Aquí tienes la primera parte de la clase Button; guárdala como *button.py*:

**button.py**

import pygame.font  
  
class Button:  
 """A class to build buttons for the game."""  
  
❶ def \_\_init\_\_(self, ai\_game, msg):  
 """Initialize button attributes."""  
 self.screen = ai\_game.screen  
 self.screen\_rect = self.screen.get\_rect()  
  
 # Set the dimensions and properties of the button.  
❷ self.width, self.height = 200, 50  
 self.button\_color = (0, 135, 0)  
 self.text\_color = (255, 255, 255)  
❸ self.font = pygame.font.SysFont(None, 48)  
  
 # Build the button's rect object and center it.  
❹ self.rect = pygame.Rect(0, 0, self.width, self.height)  
 self.rect.center = self.screen\_rect.center  
  
 # The button message needs to be prepped only once.  
❺ self.\_prep\_msg(msg)

En primer lugar, importamos el módulo pygame.font, que permite a Pygame renderizar texto en la pantalla. El método \_\_init\_\_() toma los parámetros self, el objeto ai\_game, y msg, que contiene el texto del botón ❶. Establecemos las dimensiones del botón ❷, configuramos button\_color para colorear el objeto rect del botón en verde oscuro, y configuramos text\_color para renderizar el texto en blanco.

A continuación, preparamos un atributo font para renderizar el texto ❸. El argumento None indica a Pygame que utilice la fuente por defecto, y 48 especifica el tamaño del texto. Para centrar el botón en la pantalla, creamos un rect para el botón ❹ y establecemos su atributo center para que coincida con el de la pantalla.

Pygame trabaja con texto renderizando la cadena que quieres mostrar como una imagen. Por último, llamamos a \_prep\_msg() para que se encargue de esta renderización ❺.

Aquí tienes el código de \_prep\_msg():

**button.py**

def \_prep\_msg(self, msg):  
 """Turn msg into a rendered image and center text on the button."""  
❶ self.msg\_image = self.font.render(msg, True, self.text\_color,  
 self.button\_color)  
❷ self.msg\_image\_rect = self.msg\_image.get\_rect()  
 self.msg\_image\_rect.center = self.rect.center

El método \_prep\_msg() necesita un parámetro self y el texto a representar como imagen (msg). La llamada a font.render() convierte el texto almacenado en msg en una imagen, que luego almacenamos en self.msg\_image ❶. El método font.render() también toma un valor booleano para activar o desactivar el antialiasing (*antialiasing* hace que los bordes del texto sean más suaves). Los argumentos restantes son el color de fuente y el color de fondo especificados. Establecemos el antialiasing en True y el fondo del texto en el mismo color que el botón. (Si no incluyes un color de fondo, Pygame intentará renderizar la fuente con un fondo transparente).

Centramos la imagen del texto en el botón creando un rect a partir de la imagen y fijando su atributo center para que coincida con el del botón ❷.

Por último, creamos un método draw\_button() al que podemos llamar para mostrar el botón en pantalla:

**button.py**

def draw\_button(self):  
 """Draw blank button and then draw message."""  
 self.screen.fill(self.button\_color, self.rect)  
 self.screen.blit(self.msg\_image, self.msg\_image\_rect)

Llamamos a screen.fill() para dibujar la parte rectangular del botón. Luego llamamos a screen.blit() para dibujar la imagen del texto en la pantalla, pasándole una imagen y el objeto rect asociado a la imagen. Esto completa la clase Button.

### Dibujar el botón en la pantalla

Utilizaremos la clase Button para crear un botón Reproducir en AlienInvasion. En primer lugar, actualizaremos las sentencias import:

**alien\_invasion.py**

--snip--  
from game\_stats import GameStats  
from button import Button

Como sólo necesitamos un botón Reproducir, crearemos el botón en el método \_\_init\_\_() de AlienInvasion. Podemos colocar este código al final de \_\_init\_\_():

**alien\_invasion.py**

def \_\_init\_\_(self):  
 --snip--  
 self.game\_active = False  
  
 # Make the Play button.  
 self.play\_button = Button(self, "Play")

Este código crea una instancia de Button con la etiqueta Play, pero no dibuja el botón en la pantalla. Para ello, llamaremos al método draw\_button() del botón en \_update\_screen():

**alien\_invasion.py**

def \_update\_screen(self):  
 --snip--  
 self.aliens.draw(self.screen)  
  
 # Draw the play button if the game is inactive.  
 if not self.game\_active:  
 self.play\_button.draw\_button()  
  
 pygame.display.flip()

Para que el botón Reproducir sea visible por encima de todos los demás elementos de la pantalla, lo dibujamos después de que se hayan dibujado todos los demás elementos, pero antes de pasar a una nueva pantalla. Lo incluimos en un bloque if, para que el botón sólo aparezca cuando el juego esté inactivo.

Ahora, cuando ejecutes *Alien Invasion*, deberías ver un botón Jugar en el centro de la pantalla, como se muestra en la [Figura 14-1](#figure14-1).

|  |
| --- |
| Figura |



14-1: Aparece un

botón Jugar cuando el juego está inactivo.

### Iniciar el juego

Para iniciar una nueva partida cuando el jugador haga clic en Jugar, añade el siguiente bloque elif al final de \_check\_events() para controlar los eventos del ratón sobre el botón:

**alien\_invasion.py**

def \_check\_events(self):  
 """Respond to keypresses and mouse events."""  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 --snip--  
❶ elif event.type == pygame.MOUSEBUTTONDOWN:  
❷ mouse\_pos = pygame.mouse.get\_pos()  
❸ self.\_check\_play\_button(mouse\_pos)

Pygame detecta un evento MOUSEBUTTONDOWN cuando el jugador hace clic en cualquier lugar de la pantalla ❶, pero queremos restringir nuestro juego para que responda a los clics del ratón sólo sobre el botón Jugar. Para ello, utilizamos pygame.mouse.get\_pos(), que devuelve una tupla que contiene las coordenadas *x*- y *y*- del cursor del ratón cuando se pulsa el botón del ratón ❷. Enviamos estos valores al nuevo método \_check\_play\_button() ❸.

Aquí tienes \_check\_play\_button(), que decidí colocar después de \_check\_events():

**alien\_invasion.py**

def \_check\_play\_button(self, mouse\_pos):  
 """Start a new game when the player clicks Play."""  
❶ if self.play\_button.rect.collidepoint(mouse\_pos):  
 self.game\_active = True

Utilizamos el método rect collidepoint() para comprobar si el punto del clic del ratón se solapa con la región definida por el botón Jugar de rect ❶. Si es así, establecemos game\_active en True, ¡y comienza el juego!

Llegados a este punto, deberías poder iniciar y jugar una partida completa. Cuando el juego termine, el valor de game\_active debería convertirse en False y el botón Jugar debería reaparecer.

### Reiniciar el juego

El código del botón Jugar que acabamos de escribir funciona la primera vez que el jugador pulsa Jugar. Pero no funciona después de que termine la primera partida, porque las condiciones que hicieron que terminara la partida no se han reiniciado.

Para reiniciar la partida cada vez que el jugador haga clic en Jugar, tenemos que reiniciar las estadísticas de la partida, eliminar los antiguos alienígenas y balas, construir una nueva flota y centrar la nave, como se muestra aquí:

**alien\_invasion.py**

def \_check\_play\_button(self, mouse\_pos):  
 """Start a new game when the player clicks Play."""  
 if self.play\_button.rect.collidepoint(mouse\_pos):  
 # Reset the game statistics.  
❶ self.stats.reset\_stats()  
 self.game\_active = True  
  
 # 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()

Reiniciamos las estadísticas del juego ❶, lo que proporciona al jugador tres naves nuevas. Luego ponemos game\_active en True para que el juego comience en cuanto termine de ejecutarse el código de esta función. Vaciamos los grupos aliens y bullets ❷, y después creamos una nueva flota y centramos la nave ❸.

Ahora el juego se reiniciará correctamente cada vez que pulses Jugar, ¡lo que te permitirá jugarlo tantas veces como quieras!

### Desactivar el botón Jugar

Un problema de nuestro botón Jugar es que la región del botón en la pantalla seguirá respondiendo a los clics aunque el botón Jugar no esté visible. Si haces clic en la zona del botón Jugar por accidente después de que empiece una partida, ¡el juego se reiniciará!

Para solucionarlo, configura el juego para que sólo se inicie cuando game\_active sea False:

**alien\_invasion.py**

def \_check\_play\_button(self, mouse\_pos):  
 """Start a new game when the player clicks Play."""  
❶ button\_clicked = self.play\_button.rect.collidepoint(mouse\_pos)  
❷ if button\_clicked and not self.game\_active:  
 # Reset the game statistics.  
 self.stats.reset\_stats()  
 --snip--

La bandera button\_clicked almacena un valor True o False ❶, y el juego se reiniciará sólo si se pulsa Jugar *and* el juego no está activo en ese momento ❷. Para probar este comportamiento, inicia una nueva partida y pulsa repetidamente donde debería estar el botón Jugar. Si todo funciona como se espera, hacer clic en la zona del botón Jugar no debería tener ningún efecto sobre el juego.

### Ocultar el cursor del ratón

Queremos que el cursor del ratón esté visible cuando el juego está inactivo, pero una vez que comienza el juego, simplemente estorba. Para solucionar esto, lo haremos invisible cuando el juego se active. Podemos hacerlo al final del bloque if en \_check\_play\_button():

**alien\_invasion.py**

def \_check\_play\_button(self, mouse\_pos):  
 """Start a new game when the player clicks Play."""  
 button\_clicked = self.play\_button.rect.collidepoint(mouse\_pos)  
 if button\_clicked and not self.game\_active:  
 --snip--  
 # Hide the mouse cursor.  
 pygame.mouse.set\_visible(False)

Si pasas False a set\_visible(), le dirás a Pygame que oculte el cursor cuando el ratón esté sobre la ventana del juego.

Haremos que el cursor reaparezca cuando termine la partida para que el jugador pueda volver a pulsar Jugar para empezar una nueva partida. Aquí tienes el código para hacerlo:

**alien\_invasion.py**

def \_ship\_hit(self):  
 """Respond to ship being hit by alien."""  
 if self.stats.ships\_left > 0:  
 --snip--  
 else:  
 self.game\_active = False  
 pygame.mouse.set\_visible(True)

Hacemos que el cursor vuelva a ser visible en cuanto el juego se vuelve inactivo, lo que ocurre en \_ship\_hit(). Prestar atención a detalles como éste hace que tu juego tenga un aspecto más profesional y permite que el jugador se centre en jugar, en lugar de en descifrar la interfaz de usuario.

## Pruébalo tú mismo

14-1. Pulsa P para jugar: Como *Alien Invasion* utiliza la entrada del teclado para controlar la nave, sería útil iniciar el juego pulsando una tecla. Añade código que permita al jugador pulsar P para empezar. Podría ser útil mover parte del código de \_check\_play\_button() a un método de \_start\_game() que pueda ser llamado desde \_check\_play\_button() *and* \_check\_keydown\_events() .

14-2. Práctica de puntería: Crea un rectángulo en el borde derecho de la pantalla que se mueva hacia arriba y hacia abajo a un ritmo constante. Luego, en el lado izquierdo de la pantalla, crea una nave que el jugador pueda mover arriba y abajo mientras dispara balas al objetivo rectangular. Añade un botón Jugar que inicie el juego, y cuando el jugador falle tres veces en el blanco, termina el juego y haz que vuelva a aparecer el botón Jugar. Deja que el jugador reinicie el juego con este botón Jugar.

## Subir de nivel

En nuestro juego actual, una vez que el jugador derriba toda la flota alienígena, alcanza un nuevo nivel, pero la dificultad del juego no cambia. Vamos a animar un poco las cosas y hacer que el juego sea más desafiante aumentando la velocidad del juego cada vez que el jugador despeje la pantalla.

### Modificar los ajustes de velocidad

Primero reorganizaremos la clase Settings para agrupar los ajustes del juego en estáticos y dinámicos. También nos aseguraremos de que los ajustes que cambien durante el juego se restablezcan cuando empecemos una nueva partida. Este es el método \_\_init\_\_() para *settings.py*:

**settings.py**

def \_\_init\_\_(self):  
 """Initialize the game's static settings."""  
 # Screen settings  
 self.screen\_width = 1200  
 self.screen\_height = 800  
 self.bg\_color = (230, 230, 230)  
  
 # Ship settings  
 self.ship\_limit = 3  
  
 # Bullet settings  
 self.bullet\_width = 3  
 self.bullet\_height = 15  
 self.bullet\_color = 60, 60, 60  
 self.bullets\_allowed = 3  
  
 # Alien settings  
 self.fleet\_drop\_speed = 10  
  
 # How quickly the game speeds up  
❶ self.speedup\_scale = 1.1  
  
❷ self.initialize\_dynamic\_settings()

Seguimos inicializando los ajustes que permanecen constantes en el método \_\_init\_\_(). Añadimos un ajuste speedup\_scale ❶ para controlar la velocidad del juego: un valor de 2 duplicará la velocidad del juego cada vez que el jugador alcance un nuevo nivel; un valor de 1 mantendrá la velocidad constante. Un valor como 1.1 debería aumentar la velocidad lo suficiente como para que el juego sea desafiante pero no imposible. Por último, llamamos al método initialize\_dynamic\_settings() para inicializar los valores de los atributos que deben cambiar a lo largo del juego ❷.

Este es el código de initialize\_dynamic\_settings():

**settings.py**

def initialize\_dynamic\_settings(self):  
 """Initialize settings that change throughout the game."""  
 self.ship\_speed = 1.5  
 self.bullet\_speed = 2.5  
 self.alien\_speed = 1.0  
  
 # fleet\_direction of 1 represents right; -1 represents left.  
 self.fleet\_direction = 1

Este método establece los valores iniciales de las velocidades de la nave, la bala y el alienígena. Aumentaremos estas velocidades a medida que el jugador progrese en el juego y las restableceremos cada vez que el jugador comience una nueva partida. Incluimos fleet\_direction en este método para que los alienígenas se muevan siempre al principio de una nueva partida. No necesitamos aumentar el valor de fleet\_drop\_speed, porque cuando los alienígenas se muevan más rápido por la pantalla, también bajarán más rápido por ella.

Para aumentar las velocidades de la nave, las balas y los alienígenas cada vez que el jugador alcance un nuevo nivel, escribiremos un nuevo método llamado increase\_speed():

**settings.py**

def increase\_speed(self):  
 """Increase speed settings."""  
 self.ship\_speed \*= self.speedup\_scale  
 self.bullet\_speed \*= self.speedup\_scale  
 self.alien\_speed \*= self.speedup\_scale

Para aumentar la velocidad de estos elementos del juego, multiplicamos cada ajuste de velocidad por el valor de speedup\_scale.

Aumentamos el ritmo del juego llamando a increase\_speed() en \_check\_bullet\_alien\_collisions() cuando el último alienígena de una flota ha sido derribado:

**alien\_invasion.py**

def \_check\_bullet\_alien\_collisions(self):  
 --snip--  
 if not self.aliens:  
 # Destroy existing bullets and create new fleet.  
 self.bullets.empty()  
 self.\_create\_fleet()  
 self.settings.increase\_speed()

Cambiar los valores de los ajustes de velocidad ship\_speed, alien\_speed, y bullet\_speed ¡es suficiente para acelerar todo el juego!

### Restablecer la velocidad

Ahora tenemos que devolver los ajustes modificados a sus valores iniciales cada vez que el jugador comience una nueva partida; de lo contrario, cada nueva partida comenzaría con los ajustes de velocidad aumentados de la partida anterior:

**alien\_invasion.py**

def \_check\_play\_button(self, mouse\_pos):  
 """Start a new game when the player clicks Play."""  
 button\_clicked = self.play\_button.rect.collidepoint(mouse\_pos)  
 if button\_clicked and not self.game\_active:  
 # Reset the game settings.  
 self.settings.initialize\_dynamic\_settings()  
 --snip--

Jugar a *Alien Invasion* debería ser ahora más divertido y desafiante. Cada vez que despejes la pantalla, el juego debería acelerarse y volverse ligeramente más difícil. Si el juego se vuelve demasiado difícil demasiado rápido, disminuye el valor de settings.speedup\_scale. O si el juego no es lo suficientemente difícil, aumenta ligeramente el valor. Encuentra un punto óptimo aumentando la dificultad en un tiempo razonable. Las primeras pantallas deben ser fáciles, las siguientes deben ser difíciles pero factibles, y las siguientes deben ser casi imposiblemente difíciles.

## Pruébalo tú mismo

14-3. Práctica de objetivos desafiantes: Empieza con tu trabajo del Ejercicio 14-2(página 283). Haz que el objetivo se mueva más rápido a medida que avanza el juego, y reinicia el objetivo a la velocidad original cuando el jugador haga clic en Jugar.

14-4. Niveles de Dificultad: Haz un conjunto de botones para *Alien Invasion* que permita al jugador seleccionar un nivel de dificultad inicial adecuado para el juego. Cada botón debe asignar los valores apropiados para los atributos en Settings necesarios para crear diferentes niveles de dificultad.

## Puntuación

Implementemos un sistema de puntuación para seguir la puntuación del juego en tiempo real y mostrar la puntuación más alta, el nivel y el número de naves restantes.

La puntuación es una estadística del juego, así que añadiremos un atributo score a GameStats:

**game\_stats.py**

class GameStats:  
 --snip--  
 def reset\_stats(self):  
 """Initialize statistics that can change during the game."""  
 self.ships\_left = self.ai\_settings.ship\_limit  
 self.score = 0

Para restablecer la puntuación cada vez que comience una nueva partida, inicializaremos score en reset\_stats() en lugar de \_\_init\_\_().

### Mostrar el resultado

Para mostrar la puntuación en la pantalla, primero creamos una nueva clase, Scoreboard. Por ahora, esta clase sólo mostrará la puntuación actual. Con el tiempo, también la utilizaremos para informar de la puntuación máxima, el nivel y el número de naves restantes. Aquí tienes la primera parte de la clase; guárdala como *scoreboard.py*:

**scoreboard.py**

import pygame.font  
  
class Scoreboard:  
 """A class to report scoring information."""  
  
❶ def \_\_init\_\_(self, ai\_game):  
 """Initialize scorekeeping attributes."""  
 self.screen = ai\_game.screen  
 self.screen\_rect = self.screen.get\_rect()  
 self.settings = ai\_game.settings  
 self.stats = ai\_game.stats  
  
 # Font settings for scoring information.  
❷ self.text\_color = (30, 30, 30)  
❸ self.font = pygame.font.SysFont(None, 48)  
  
 # Prepare the initial score image.  
❹ self.prep\_score()

Como Scoreboard escribe texto en la pantalla, empezamos importando el módulo pygame.font. A continuación, damos a \_\_init\_\_() el parámetro ai\_game para que pueda acceder a los objetos settings, screen y stats, que necesitará para informar de los valores que estamos siguiendo ❶. A continuación, establecemos un color de texto ❷ e instanciamos un objeto fuente ❸.

Para convertir el texto que se va a mostrar en una imagen, llamamos a prep\_score() ❹, que definimos aquí:

**scoreboard.py**

def prep\_score(self):  
 """Turn the score into a rendered image."""  
❶ score\_str = str(self.stats.score)  
❷ self.score\_image = self.font.render(score\_str, True,  
 self.text\_color, self.settings.bg\_color)  
  
 # Display the score at the top right of the screen.  
❸ self.score\_rect = self.score\_image.get\_rect()  
❹ self.score\_rect.right = self.screen\_rect.right - 20  
❺ self.score\_rect.top = 20

En prep\_score(), convertimos el valor numérico stats.score en una cadena ❶ y luego pasamos esta cadena a render(), que crea la imagen ❷. Para mostrar claramente la puntuación en pantalla, pasamos el color de fondo de la pantalla y el color del texto a render().

Colocaremos la puntuación en la esquina superior derecha de la pantalla y haremos que se expanda hacia la izquierda a medida que aumente la puntuación y crezca la anchura del número. Para asegurarnos de que la puntuación siempre se alinea con el lado derecho de la pantalla, creamos un rect llamado score\_rect ❸ y fijamos su borde derecho a 20 píxeles del borde derecho de la pantalla ❹. A continuación, colocamos el borde superior a 20 píxeles de la parte superior de la pantalla ❺.

A continuación, creamos un método show\_score() para mostrar la imagen de puntuación renderizada:

**scoreboard.py**

def show\_score(self):  
 """Draw score to the screen."""  
 self.screen.blit(self.score\_image, self.score\_rect)

Este método dibuja la imagen de la partitura en la pantalla en el lugar que especifique score\_rect.

### Crear un marcador

Para mostrar el marcador, crearemos una instancia de Scoreboard en AlienInvasion. En primer lugar, vamos a actualizar las sentencias import:

**alien\_invasion.py**

--snip--  
from game\_stats import GameStats  
from scoreboard import Scoreboard  
--snip--

A continuación, creamos una instancia de Scoreboard en \_\_init\_\_():

**alien\_invasion.py**

def \_\_init\_\_(self):  
 --snip--  
 pygame.display.set\_caption("Alien Invasion")  
  
 # Create an instance to store game statistics,  
 # and create a scoreboard.  
 self.stats = GameStats(self)  
 self.sb = Scoreboard(self)  
 --snip--

Luego dibujamos el marcador en pantalla en \_update\_screen():

**alien\_invasion.py**

def \_update\_screen(self):  
 --snip--  
 self.aliens.draw(self.screen)  
  
 # Draw the score information.  
 self.sb.show\_score()  
  
 # Draw the play button if the game is inactive.  
 --snip--

Llamamos a show\_score() justo antes de dibujar el botón Reproducir.

Cuando ejecutes ahora *Alien Invasion*, debería aparecer un 0 en la parte superior derecha de la pantalla. (En este momento, sólo queremos asegurarnos de que la puntuación aparece en el lugar correcto antes de seguir desarrollando el sistema de puntuación). [La Figura 14-2](#figure14-2) muestra la puntuación tal y como aparece antes de que comience el juego.

A continuación, ¡asignaremos valores de puntos a cada alienígena!

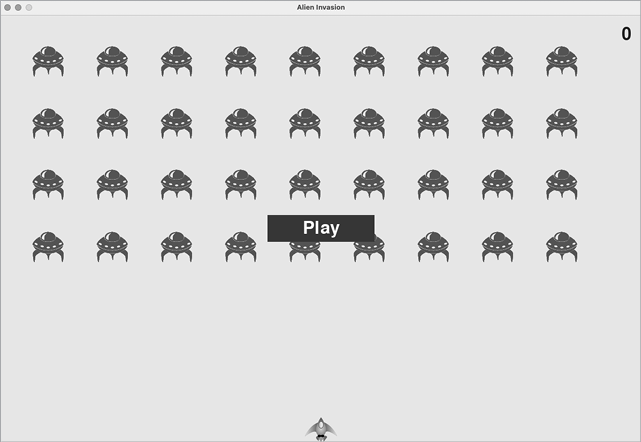


Figura 14-2: La puntuación aparece en la esquina superior derecha de la pantalla.

### Actualizar la puntuación a medida que se derriba a los alienígenas

Para escribir una puntuación en vivo en la pantalla, actualizamos el valor de stats.score cada vez que se derriba a un alienígena, y luego llamamos a prep\_score() para actualizar la imagen de la puntuación. Pero antes, determinemos cuántos puntos obtiene un jugador cada vez que derriba a un alienígena:

**settings.py**

def initialize\_dynamic\_settings(self):  
 --snip--  
  
 # Scoring settings  
 self.alien\_points = 50

Aumentaremos el valor en puntos de cada alienígena a medida que avance el juego. Para asegurarnos de que este valor en puntos se restablece cada vez que comienza una nueva partida, fijamos el valor en initialize\_dynamic\_settings().

Actualicemos la puntuación en \_check\_bullet\_alien\_collisions() cada vez que se derribe a un alienígena:

**alien\_invasion.py**

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 collisions:  
 self.stats.score += self.settings.alien\_points  
 self.sb.prep\_score()  
 --snip--

Cuando una bala alcanza a un alienígena, Pygame devuelve un diccionario collisions. Comprobamos si el diccionario existe, y si es así, el valor del alienígena se añade a la puntuación. A continuación, llamamos a prep\_score() para crear una nueva imagen para la puntuación actualizada.

Ahora, cuando juegues a *Alien Invasion*, ¡podrás acumular puntos!

### Restablecer la puntuación

En este momento, sólo estamos preparando una nueva puntuación *after* se ha golpeado a un alienígena, lo que funciona durante la mayor parte del juego. Pero cuando empecemos una nueva partida, seguiremos viendo nuestra puntuación de la partida anterior hasta que sea alcanzado el primer alienígena.

Podemos solucionarlo preparando la puntuación al iniciar una nueva partida:

**alien\_invasion.py**

def \_check\_play\_button(self, mouse\_pos):  
 --snip--  
 if button\_clicked and not self.game\_active:  
 --snip--  
 # Reset the game statistics.  
 self.stats.reset\_stats()  
 self.sb.prep\_score()  
 --snip--

Llamamos a prep\_score() después de reiniciar las estadísticas del juego al iniciar una nueva partida. Esto prepara el marcador con una puntuación de 0.

### Asegurarse de anotar todos los golpes

Tal y como está escrito actualmente, nuestro código podría no puntuar a algunos alienígenas. Por ejemplo, si dos balas chocan con los alienígenas durante la misma pasada por el bucle o si hacemos una bala extra ancha para golpear a varios alienígenas, el jugador sólo recibirá puntos por golpear a uno de los alienígenas. Para solucionar esto, vamos a refinar la forma en que se detectan las colisiones bala-extraterrestre.

En \_check\_bullet\_alien\_collisions(), cualquier bala que colisione con un alienígena se convierte en una clave del diccionario collisions. El valor asociado a cada bala es una lista de los alienígenas con los que ha colisionado. Recorremos en bucle los valores del diccionario collisions para asegurarnos de que otorgamos puntos por cada alienígena alcanzado:

**alien\_invasion.py**

def \_check\_bullet\_alien\_collisions(self):  
 --snip--  
 if collisions:  
 for aliens in collisions.values():  
 self.stats.score += self.settings.alien\_points \* len(aliens)  
 self.sb.prep\_score()  
 --snip--

Si se ha definido el diccionario collisions, recorremos todos los valores del diccionario. Recuerda que cada valor es una lista de alienígenas alcanzados por una sola bala. Multiplicamos el valor de cada alienígena por el número de alienígenas de cada lista y añadimos esta cantidad a la puntuación actual. Para probarlo, cambia la anchura de una bala a 300 píxeles y comprueba que recibes puntos por cada alienígena que golpeas con tus balas extra anchas; luego devuelve la anchura de la bala a su valor normal.

### Aumentar los valores de puntos

Dado que el juego se vuelve más difícil cada vez que el jugador alcanza un nuevo nivel, los alienígenas de los niveles posteriores deberían valer más puntos. Para implementar esta funcionalidad, añadiremos código para aumentar el valor en puntos cuando aumente la velocidad del juego:

**settings.py**

class Settings:  
 """A class to store all settings for Alien Invasion."""  
  
 def \_\_init\_\_(self):  
 --snip--  
 # How quickly the game speeds up  
 self.speedup\_scale = 1.1  
 # How quickly the alien point values increase  
❶ self.score\_scale = 1.5  
  
 self.initialize\_dynamic\_settings()  
  
 def initialize\_dynamic\_settings(self):  
 --snip--  
  
 def increase\_speed(self):  
 """Increase speed settings and alien point values."""  
 self.ship\_speed \*= self.speedup\_scale  
 self.bullet\_speed \*= self.speedup\_scale  
 self.alien\_speed \*= self.speedup\_scale  
  
❷ self.alien\_points = int(self.alien\_points \* self.score\_scale)

Definimos una velocidad a la que aumentan los puntos, que llamamos score\_scale ❶. Un pequeño aumento de la velocidad (1.1) hace que el juego sea más difícil rápidamente. Pero para ver una diferencia más notable en la puntuación, necesitamos cambiar el valor del punto ajeno en una cantidad mayor (1.5). Ahora, cuando aumentamos la velocidad del juego, también aumentamos el valor en puntos de cada golpe ❷. Utilizamos la función int() para aumentar el valor en puntos en enteros.

Para ver el valor de cada alienígena, añade una llamada print() al método increase\_speed() en Settings:

**settings.py**

def increase\_speed(self):  
 --snip--  
 self.alien\_points = int(self.alien\_points \* self.score\_scale)  
 print(self.alien\_points)

El nuevo valor en puntos debería aparecer en el terminal cada vez que alcances un nuevo nivel.

## Nota

Asegúrate de eliminar la llamada a print() después de comprobar que el valor en puntos aumenta, o podría afectar al rendimiento de tu juego y distraer al jugador.

### Redondear la puntuación

La mayoría de los juegos de disparos de estilo arcade informan de las puntuaciones como múltiplos de 10, así que vamos a seguir esa pauta con nuestras puntuaciones. Además, formateemos la puntuación para incluir separadores de comas en los números grandes. Haremos este cambio en Scoreboard:

**scoreboard.py**

def prep\_score(self):  
 """Turn the score into a rendered image."""  
 rounded\_score = round(self.stats.score, -1)  
 score\_str = f"{rounded\_score:,}"  
 self.score\_image = self.font.render(score\_str, True,  
 self.text\_color, self.settings.bg\_color)  
 --snip--

La función round() normalmente redondea un flotante a un número determinado de decimales dado como segundo argumento. Sin embargo, cuando pasas un número negativo como segundo argumento, round() redondeará el valor al 10, 100, 1.000, etc., más próximos. Este código le dice a Python que redondee el valor de stats.score al 10 más próximo y lo asigne a rounded\_score.

A continuación, utilizamos un especificador de formato en la cadena f para la puntuación. Un *format specifier* es una secuencia especial de caracteres que modifica la forma en que se presenta el valor de una variable. En este caso, la secuencia :, indica a Python que inserte comas en los lugares adecuados del valor numérico que se proporcione. Esto da como resultado cadenas como 1,000,000 en lugar de 1000000.

Ahora, cuando ejecutes el juego, deberías ver una puntuación redondeada y con un formato limpio, incluso cuando acumules muchos puntos, como se muestra en la [Figura 14-3](#figure14-3).

|  |
| --- |
| Figura |



14-3: Puntuación redondeada con separadores de comas

### Puntuaciones altas

Todos los jugadores quieren superar la puntuación máxima de un juego, así que vamos a registrar e informar de las puntuaciones máximas para dar a los jugadores algo por lo que esforzarse. Almacenaremos las puntuaciones altas en GameStats:

**game\_stats.py**

def \_\_init\_\_(self, ai\_game):  
 --snip--  
 # High score should never be reset.  
 self.high\_score = 0

Como la puntuación máxima no debe restablecerse nunca, inicializamos high\_score en \_\_init\_\_() en lugar de en reset\_stats().

A continuación, modificaremos Scoreboard para que muestre la puntuación máxima. Empecemos con el método \_\_init\_\_():

**scoreboard.py**

def \_\_init\_\_(self, ai\_game):  
 --snip--  
 # Prepare the initial score images.  
 self.prep\_score()  
❶ self.prep\_high\_score()

La puntuación alta se mostrará separada de la puntuación, por lo que necesitamos un nuevo método, prep\_high\_score(), para preparar la imagen de la puntuación alta ❶.

Aquí tienes el método prep\_high\_score():

**scoreboard.py**

def prep\_high\_score(self):  
 """Turn the high score into a rendered image."""  
❶ high\_score = round(self.stats.high\_score, -1)  
 high\_score\_str = f"{high\_score:,}"  
❷ self.high\_score\_image = self.font.render(high\_score\_str, True,  
 self.text\_color, self.settings.bg\_color)  
  
 # Center the high score at the top of the screen.  
 self.high\_score\_rect = self.high\_score\_image.get\_rect()  
❸ self.high\_score\_rect.centerx = self.screen\_rect.centerx  
❹ self.high\_score\_rect.top = self.score\_rect.top

Redondeamos la puntuación máxima a la decena más próxima y le damos formato con comas ❶. A continuación, generamos una imagen a partir de la puntuación máxima ❷, centramos horizontalmente la puntuación máxima rect ❸, y establecemos su atributo top para que coincida con la parte superior de la imagen de la puntuación ❹.

El método show\_score() dibuja ahora la puntuación actual en la parte superior derecha y la puntuación alta en la parte superior central de la pantalla:

**scoreboard.py**

def show\_score(self):  
 """Draw score to the screen."""  
 self.screen.blit(self.score\_image, self.score\_rect)  
 self.screen.blit(self.high\_score\_image, self.high\_score\_rect)

Para comprobar las puntuaciones altas, escribiremos un nuevo método, check\_high\_score(), en Scoreboard:

**scoreboard.py**

def check\_high\_score(self):  
 """Check to see if there's a new high score."""  
 if self.stats.score > self.stats.high\_score:  
 self.stats.high\_score = self.stats.score  
 self.prep\_high\_score()

El método check\_high\_score() comprueba la puntuación actual comparándola con la puntuación alta. Si la puntuación actual es mayor, actualizamos el valor de high\_score y llamamos a prep\_high\_score() para actualizar la imagen de la puntuación alta.

Tenemos que llamar a check\_high\_score() cada vez que se golpea a un alienígena después de actualizar la puntuación en \_check\_bullet\_alien\_collisions():

**alien\_invasion.py**

def \_check\_bullet\_alien\_collisions(self):  
 --snip--  
 if collisions:  
 for aliens in collisions.values():  
 self.stats.score += self.settings.alien\_points \* len(aliens)  
 self.sb.prep\_score()  
 self.sb.check\_high\_score()  
 --snip--

Llamamos a check\_high\_score() cuando el diccionario collisions está presente, y lo hacemos después de actualizar la puntuación de todos los alienígenas que han sido golpeados.

La primera vez que juegues a *Alien Invasion*, tu puntuación será la puntuación máxima, por lo que se mostrará como la puntuación actual y la puntuación máxima. Pero cuando empieces una segunda partida, tu puntuación alta debería aparecer en el centro y tu puntuación actual debería aparecer a la derecha, como se muestra en la [Figura 14-4](#figure14-4).

|  |
| --- |
| Figura |



14-4: La puntuación máxima aparece en la parte superior central de la pantalla.

### Visualización del nivel

Para mostrar el nivel del jugador en el juego, primero necesitamos un atributo en GameStats que represente el nivel actual. Para restablecer el nivel al comienzo de cada nueva partida, inicialízalo en reset\_stats():

**game\_stats.py**

def reset\_stats(self):  
 """Initialize statistics that can change during the game."""  
 self.ships\_left = self.settings.ship\_limit  
 self.score = 0  
 self.level = 1

Para que Scoreboard muestre el nivel actual, llamamos a un nuevo método, prep\_level(), desde \_\_init\_\_():

**scoreboard.py**

def \_\_init\_\_(self, ai\_game):  
 --snip--  
 self.prep\_high\_score()  
 self.prep\_level()

Aquí tienes prep\_level():

**scoreboard.py**

def prep\_level(self):  
 """Turn the level into a rendered image."""  
 level\_str = str(self.stats.level)  
❶ self.level\_image = self.font.render(level\_str, True,  
 self.text\_color, self.settings.bg\_color)  
  
 # Position the level below the score.  
 self.level\_rect = self.level\_image.get\_rect()  
❷ self.level\_rect.right = self.score\_rect.right  
❸ self.level\_rect.top = self.score\_rect.bottom + 10

El método prep\_level() crea una imagen a partir del valor almacenado en stats.level ❶ y establece el atributo right de la imagen para que coincida con el atributo right de la puntuación ❷. A continuación, establece el atributo top 10 píxeles por debajo de la parte inferior de la imagen de la puntuación para dejar espacio entre la puntuación y el nivel ❸.

También tenemos que actualizar show\_score():

**scoreboard.py**

def show\_score(self):  
 """Draw scores and level to the screen."""  
 self.screen.blit(self.score\_image, self.score\_rect)  
 self.screen.blit(self.high\_score\_image, self.high\_score\_rect)  
 self.screen.blit(self.level\_image, self.level\_rect)

Esta nueva línea dibuja la imagen del nivel en la pantalla.

Incrementaremos stats.level y actualizaremos la imagen de nivel en \_check\_bullet\_alien\_collisions():

**alien\_invasion.py**

def \_check\_bullet\_alien\_collisions(self):  
 --snip--  
 if not self.aliens:  
 # Destroy existing bullets and create new fleet.  
 self.bullets.empty()  
 self.\_create\_fleet()  
 self.settings.increase\_speed()  
  
 # Increase level.  
 self.stats.level += 1  
 self.sb.prep\_level()

Si se destruye una flota, incrementamos el valor de stats.level y llamamos a prep\_level() para asegurarnos de que el nuevo nivel se muestra correctamente.

Para asegurarnos de que la imagen del nivel se actualiza correctamente al inicio de una nueva partida, también llamamos a prep\_level() cuando el jugador pulsa el botón Jugar:

**alien\_invasion.py**

def \_check\_play\_button(self, mouse\_pos):  
 --snip--  
 if button\_clicked and not self.game\_active:  
 --snip--  
 self.sb.prep\_score()  
 self.sb.prep\_level()  
 --snip--

Llamamos a prep\_level() justo después de llamar a prep\_score().

Ahora verás cuántos niveles has completado, como se muestra en la [Figura 14-5](#figure14-5).

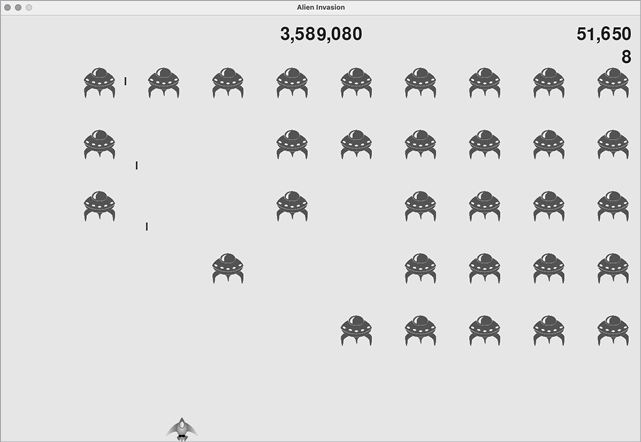


Figura14-5: El nivel actual aparece justo debajo de la puntuación actual.

## Nota

En algunos juegos clásicos, las puntuaciones tienen etiquetas, como Puntuación, Puntuación máxima y Nivel. Hemos omitido estas etiquetas porque el significado de cada número queda claro una vez que has jugado al juego. Para incluir estas etiquetas, añádelas a las cadenas de puntuación justo antes de las llamadas a font.render() en Scoreboard.

### Mostrar el número de naves

Por último, vamos a mostrar el número de naves que le quedan al jugador, pero esta vez, utilizaremos un gráfico. Para ello, dibujaremos naves en la esquina superior izquierda de la pantalla para representar cuántas naves quedan, tal y como hacen muchos juegos arcade clásicos.

Primero, tenemos que hacer que Ship herede de Sprite para poder crear un grupo de naves:

**ship.py**

import pygame  
from pygame.sprite import Sprite  
  
❶ class Ship(Sprite):  
 """A class to manage the ship."""  
  
 def \_\_init\_\_(self, ai\_game):  
 """Initialize the ship and set its starting position."""  
❷ super().\_\_init\_\_()  
 --snip--

Aquí importamos Sprite, nos aseguramos de que Ship hereda de Sprite ❶, y llamamos a super() al principio de \_\_init\_\_() ❷.

A continuación, tenemos que modificar Scoreboard para crear un grupo de naves que podamos mostrar. Estas son las sentencias import para Scoreboard:

**scoreboard.py**

import pygame.font  
from pygame.sprite import Group  
  
from ship import Ship

Como estamos creando un grupo de barcos, importamos las clases Group y Ship.

Aquí tienes \_\_init\_\_():

**scoreboard.py**

def \_\_init\_\_(self, ai\_game):  
 """Initialize scorekeeping attributes."""  
 self.ai\_game = ai\_game  
 self.screen = ai\_game.screen  
 --snip--  
 self.prep\_level()  
 self.prep\_ships()

Asignamos la instancia del juego a un atributo, porque la necesitaremos para crear algunas naves. Llamamos a prep\_ships() después de la llamada a prep\_level().

Aquí tienes prep\_ships():

**scoreboard.py**

def prep\_ships(self):  
 """Show how many ships are left."""  
❶ self.ships = Group()  
❷ for ship\_number in range(self.stats.ships\_left):  
 ship = Ship(self.ai\_game)  
❸ ship.rect.x = 10 + ship\_number \* ship.rect.width  
❹ ship.rect.y = 10  
❺ self.ships.add(ship)

El método prep\_ships() crea un grupo vacío, self.ships, para contener las instancias de naves ❶. Para llenar este grupo, se ejecuta un bucle una vez por cada nave que el jugador haya dejado ❷. Dentro del bucle, creamos una nueva nave y establecemos el valor de la coordenada *x* de cada nave para que las naves aparezcan una junto a otra con un margen de 10 píxeles a la izquierda del grupo de naves ❸. Fijamos el valor de *y*-coordenada 10 píxeles hacia abajo desde la parte superior de la pantalla para que las naves aparezcan en la esquina superior izquierda de la pantalla ❹. A continuación, añadimos cada nave nueva al grupo ships ❺.

Ahora tenemos que dibujar las naves en la pantalla:

**scoreboard.py**

def show\_score(self):  
 """Draw scores, level, and ships to the screen."""  
 self.screen.blit(self.score\_image, self.score\_rect)  
 self.screen.blit(self.high\_score\_image, self.high\_score\_rect)  
 self.screen.blit(self.level\_image, self.level\_rect)  
 self.ships.draw(self.screen)

Para mostrar las naves en la pantalla, llamamos a draw() en el grupo, y Pygame dibuja cada nave.

Para mostrar al jugador cuántas naves tiene para empezar, llamamos a prep\_ships() cuando comienza una nueva partida. Esto lo hacemos en \_check\_play\_button() en AlienInvasion:

**alien\_invasion.py**

def \_check\_play\_button(self, mouse\_pos):  
 --snip--  
 if button\_clicked and not self.game\_active:  
 --snip--  
 self.sb.prep\_level()  
 self.sb.prep\_ships()  
 --snip--

También llamamos a prep\_ships() cuando una nave es alcanzada, para actualizar la visualización de las imágenes de las naves cuando el jugador pierde una nave:

**alien\_invasion.py**

def \_ship\_hit(self):  
 """Respond to ship being hit by alien."""  
 if self.stats.ships\_left > 0:  
 # Decrement ships\_left, and update scoreboard.  
 self.stats.ships\_left -= 1  
 self.sb.prep\_ships()  
 --snip--

Llamamos a prep\_ships() después de disminuir el valor de ships\_left, para que se muestre el número correcto de naves restantes cada vez que se destruye una nave.

[La Figura 14-6](#figure14-6) muestra el sistema de puntuación completo, con las naves restantes mostradas en la parte superior izquierda de la pantalla.

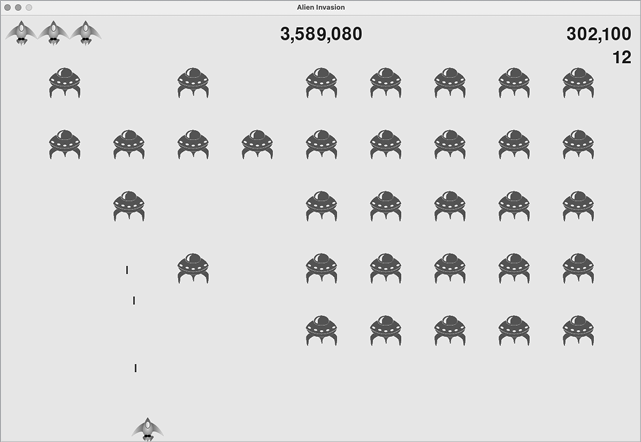


Figura14-6: El sistema de puntuación completo de *Alien Invasion*

## Pruébalo tú mismo

14-5. Puntuación máxima de todos los tiempos: La puntuación máxima se reinicia cada vez que un jugador cierra y reinicia *Alien Invasion*. Arréglalo escribiendo la puntuación máxima en un archivo antes de llamar a sys.exit() y leyendo la puntuación máxima al inicializar su valor en GameStats.

14-6. Refactorización: Busca métodos que estén haciendo más de una tarea, y refactorízalos para organizar tu código y hacerlo eficiente. Por ejemplo, mueve parte del código en \_check\_bullet\_alien\_collisions(), que inicia un nuevo nivel cuando la flota de alienígenas ha sido destruida, a una función llamada start\_new\_level(). Además, traslada las cuatro llamadas a métodos independientes del método \_\_init\_\_() de Scoreboard a un método llamado prep\_images() para acortar \_\_init\_\_(). El método prep\_images() también podría ayudar a simplificar \_check\_play\_button() o start\_game() si ya has refactorizado \_check\_play\_button().

## nota

Antes de intentar refactorizar el proyecto, consulta el Apéndice D para aprender a restaurar el proyecto a un estado de funcionamiento si introduces errores mientras refactorizas.

14-7. Expandir el juego: Piensa en una forma de ampliar *Alien Invasion*. Por ejemplo, puedes programar a los alienígenas para que disparen balas a tu nave. También puedes añadir escudos para que tu nave se esconda detrás, que puedan ser destruidos por las balas de cualquier lado. O puedes utilizar algo como el módulo pygame.mixer para añadir efectos de sonido, como explosiones y sonidos de disparos.

14-8. Tirador lateral, versión final: Continúa desarrollando *Sideways Shooter*, utilizando todo lo que hemos hecho en este proyecto. Añade un botón Jugar, haz que el juego se acelere en los puntos adecuados y desarrolla un sistema de puntuación. Asegúrate de refactorizar tu código a medida que trabajas, y busca oportunidades para personalizar el juego más allá de lo que se ha mostrado en este capítulo.

## Resumen

En este capítulo has aprendido a implementar un botón Jugar para iniciar una nueva partida. También has aprendido a detectar eventos del ratón y a ocultar el cursor en los juegos activos. Puedes utilizar lo que has aprendido para crear otros botones, como un botón Ayuda para mostrar instrucciones sobre cómo jugar a tus juegos. También aprendiste a modificar la velocidad de un juego a medida que avanza, a implementar un sistema de puntuación progresiva y a mostrar información de forma textual y no textual.