Save the Cookie Cat

Desarrollado por: @Javieer57

Fecha de lanzamiento: Mayo 2022

Github: https://github.com/Javieer57/Save_the_cookie_cat

Idioma: Español

Creado con: Python y PyGame

Juego creado como proyecto final del Bootcamp de introducción a la programación impartido por Código fácilito

Introducción

Creo que el crear un video juego es uno de los proyectos que todos los programadores se plantean al menos una vez en su carrera pero, por alguna u otra razón, se deja de lado o no se termina porque uno aspira a crear grandes juegos como los de la saga de Zelda o de Mario.

Save the Cookie Cat es un juego que surge como proyecto final del Bootcamp de Introducción a la programación impartido por Código Fácilito. Este proyecto debía ajustarse a ciertos criterios como ser programado con Python y Pygame y que manejara una puntuación desde un archivo externo (con la finalidad de aprender a manipular archivos con este lenguaje).

El nombre es una referencia a otro juego de Steven Universe llamado Save the Light y en este documento hablaré un poco de los retos que enfrenté para programarlo, cómo los resolví y también sobre algunos problemas que quedaron pendientes.

Descripción del juego

Recoge todas las gato-galletas que puedas mientras esquivas los taladros que caen cada vez más rápido.

Por dónde empezar

El primer reto al que me enfrente para construir este juego fue que no sabía por dónde empezar, cuál era la lógica a seguir para este tipo de proyectos y cómo se trabaja con la librería de pygame.

Para cubrir esta parte me apoyé en el Curso para crear un juego con Pygame igualmente de Código Facilito. Tomé el juego que se creó durante el curso como una base y, conociendo mejor la lógica para trabajar con esta librería, empecé a crear Save the Cookie Cat.

Aumentar la dificultad

Para hacer de Save the Cookie Cat un juego más entretenido, decidí que los taladros cayeran cada vez más rápido conforme avanzaba el juego.

Esto lo hice agregando un método que subiera el nivel en 1 después de cada horda de taladros y, de igual forma, aumentara la velocidad de caída si el número de nivel era múltiplo de `LEVEL_TO_INCREASE_SPEED`.

```
def update_level(self):
      self.level += 1
      if(self.level % LEVEL_TO_INCREASE_SPEED == 0):
      self.fall_speed += SPEED_INCREASE
```

Cuando agregué esta característica pensaba en aumentar la velocidad de caída cada nivel pero esto hacía que los taladros cayeran muy rápido en poco tiempo, por eso condicioné el aumento de la velocidad al nivel en el que estuviera el jugador.

Otra ventaja que encontré de usar las constantes `LEVEL_TO_INCREASE_SPEED` y `SPEED_INCREASE` es que hacen más fácil configurar esta característica para el programador y también permiten al jugador manipular estos valores (desde el archivo `config.py`) para tener diferentes experiencias; ya sea configurando el juego para que vaya más lento o sea más rápido y desafiante.

Sistema de vidas

En el juego original del Curso para crear un juego con Pygame la partida terminaba cuando Cody colisiona con una pared. Sin embargo, para Save the Cookie Cat quise complicarme un poco añadiendo un sistema de vidas que le diera al jugador un cierto número de intentos antes de terminar la partida.

Lo primero que necesité fue establecer el número de vidas que el jugador iba a tener en cada partida:

```
# Dentro del método new()
self.lives = LIVES;
```

Después agregué el método `lost_live()` que se ejecuta después de cada colisión con un taladro. Esta función reduce las vidas del jugador y pregunta si las vidas no han llegado a 0. De ser el caso, se ejecutan todos los métodos para detener la partida.

```
def lost_live(self):
      self.lives -= 1

      if self.lives == 0:
      self.save_score()
      self.stop()
```

Para mostrar las vidas en pantalla utilicé el mismo método que usó Eduardo en el curso para mostrar el score y el nivel, pero agregué un pequeño ciclo que agregara una 'x' por cada vida que le quedara al jugador:

```
def lives_format(self):
      contador = ''
      for live in range(self.lives):
      contador += ' x '
      return 'Lives: {}'.format(contador)
```

En un inicio quise agregar imágenes que indicaran las vidas (como los típicos corazones en muchos juegos) y pensé que podrían agregarse dentro del texto como se agrega el código SVG en HTML. Sin embargo, no encontré la forma de hacerlo y opté por dejar las 'x'.

Animación del personaje al cambiar de dirección

Para hacer el efecto de animar al personaje y mostrar diferentes sprites de acuerdo a la dirección del jugador, tomé el consejo de Eduardo y guardé los diferentes sprites en un tuple.

De esta forma, se le pueden asignar diferentes valores a la propiedad `self.image` (la cual se encarga de mostrar el sprite del personaje) cuando se ejecuten los métodos que manipulan la dirección. Lo que no pude hacer funcionar fue el agregar un tercer sprite que mostrara al personaje de pie, sin movimiento.

Intenté llamar una función que hiciera esto cuando se dejara de apretar la tecla de dirección (usando `pygame.KEYUP`) pero no logré mostrar el tercer sprite por lo que solo hay dos estados del personaje, apuntando a la izquierda o apuntando a la derecha.

```
class Player(pygame.sprite.Sprite):
	def __init__(self, left, bottom):
	pygame.sprite.Sprite.__init__(self)

	self.images = (
		pygame.image.load(SPRITES_DIRECTORY / 'steven_right.png'),
		pygame.image.load(SPRITES_DIRECTORY / 'steven_left.png')
	)

	self.image = self.images[0]
	self.rect = self.image.get_rect()
	self.rect.left = left
	self.rect.bottom = bottom

	self.can_move = True

	def update_pos_right(self):
		if self.rect.right < WIDTH and self.can_move:
			self.image = self.images[0]
			self.rect.x += PLAYER_SPEED

	def update_pos_left(self):
		if self.rect.left > 0 and self.can_move:
			self.image = self.images[1]
		self.rect.x -= PLAYER_SPEED
```

Distribuir la aparición de los objetos en columnas

En un inicio, el juego iba a colocar los taladros en una posición aleatoria en el eje X pero este comportamiento no distribuía los taladros de una forma muy orgánica.

Para arreglar esto me pareció que sería mejor dividir el ancho de la pantalla a forma de columnas y que, al generar los taladros, su aparición estuviera más controlada.

El sistema de división entre columnas se compone de dos partes: la primera se encarga de dividir el ancho de la pantalla entre el número columnas que se quieren y, de esa forma, obtener el tamaño de una columna (este valor se guarda en `COLUMN_FOR_OBJECTS_FALL`); en la segunda parte se obtiene un múltiplo aleatorio de `COLUMN_FOR_OBJECTS_FALL` y se usa como coordenada en X para para generar un objeto (taladro o galleta).

```
COLUMN_FOR_OBJECTS_FALL = int((SURFACE_MARGIN_RIGHT) / 14)
```

```
def generate_drills(self):
      drill_initial_pos_y = -100

      if len(self.drills) == 0:
      for drill in range(0, DRILLS_PER_LEVEL):
            left_random = COLUMN_FOR_OBJECTS_FALL * random.randrange(1, 15)

            drill = Drill(left_random, drill_initial_pos_y,
self.fall_speed)

            random_gap_drills = random.randrange(100, DRILLS_GAP)
            drill_initial_pos_y = drill.rect.top - random_gap_drills

            self.sprites.add(drill)
            self.drills.add(drill)

      self.update_level()
      self.generate_cookies()
```

Escritura y lectura de score

La puntuación del juego se maneja en el archivo `score.txt` y para hacer referencia a este utilicé la constante `SCORE_DIRECTORY`.

Los métodos para manipular el score fueron: `read_score()` y `save_score()`. El primero incluye una validación en caso de que el archivo estuviera vacío (agregando 0 como la puntuación inicial) y el segundo solo guarda la puntuación de la partida en caso de ser mayor al score más alto:

```
def read_score(self):
    if SCORE_DIRECTORY.exists() and SCORE_DIRECTORY.name == 'score.txt':
       content = SCORE_DIRECTORY.read_text()
       if content == '':
             SCORE_DIRECTORY.write_text('0')

       return SCORE_DIRECTORY.read_text()
```

```
def save_score(self):
    if SCORE_DIRECTORY.exists() and SCORE_DIRECTORY.name == 'score.txt':
       if self.score > self.prev_score:
             new_score = str(self.score)
             SCORE_DIRECTORY.write_text(new_score)
```

Para mostrar la puntuación más alta antes y después de cada partida utilicé la variable `self.prev_score`, la cual también me ayuda a hacer la comparación de `save_score()`.

La puntuación de la partida se va actualizando con `update_score()` y guardando en `self.score` cada vez que el jugador toca una galleta. Para mostrarla en pantalla utilicé el método `display_text()` de Eduardo y decidí que al final de cada partida se dibujara junto con `self.prev_score` para que el jugador viera su puntuación y recordara cuál era el score a superar.

```
def update_score(self):
      self.score += 1
```

```
def show_lose_message(self):
    self.display_text(str(self.score), FONT_SIZE + 40, BLACK, 215 , 240)
    self.display_text(str(self.prev_score), FONT_SIZE + 40, BLACK,
```

Mostrar diferentes menús al inicio

En el Curso para crear un videojuego con python, Eduardo agregó un código que daba pie a incluir un menú antes de iniciar cada partida y lo aproveché para cumplir con el requerimiento de mostrar el mayor puntaje. Las diferentes pantallas que se muestran en el juego son

imágenes que dan la ilusión de incluir botones. Estas se muestran dependiendo del valor que tenga la variable `actual_menu`.

El método `start_menu()` se encarga de mostrar el menú de inicio y dependiendo del menú que se esté mostrando, agrega diferentes acciones a las teclas para moverse entre las diferentes pantallas. La variable `actual_menu` es la más importante para poder cambiar entre menús pero terminé con muchas condicionales que no pude reducir a otros métodos que dejaran más limpia esta parte del código.

Solo el menú de inicio y el de instrucciones (`actual_menu='start'` y `actual_menu='controls'`) tienen habilitada la tecla de espacio para terminar con el ciclo while que dibuja el menú en pantalla y poder iniciar la partida.

```
def start_menu(self):
      wait = True

      menu_img = pygame.image.load(SPRITES_DIRECTORY / 'start_menu.jpg')
      rect = menu_img.get_rect()
      rect.center = (WIDTH // 2, HEIGHT // 2)

      actual_menu = 'start'
      show_score = True

      while wait:
            for event in pygame.event.get():
                  if event.type == pygame.QUIT:
                  wait = False
                  self.running = False
                  pygame.quit()
                  sys.exit()

                  key = pygame.key.get_pressed()

                  if actual_menu == 'start':
                  if key[pygame.K_RIGHT]:
                        menu_img = pygame.image.load(SPRITES_DIRECTORY /
'controls_menu.jpg')
                        actual_menu = 'controls'
                        show_score = False
```

```
                if key[pygame.K_LEFT]:
                      menu_img = pygame.image.load(SPRITES_DIRECTORY /
'credits.jpg')
                      actual_menu = 'credits'
                      show_score = False

                if key[pygame.K_SPACE]:
                      wait = False

                if actual_menu == 'controls':
                if key[pygame.K_LEFT]:
                      menu_img = pygame.image.load(SPRITES_DIRECTORY /
'start_menu.jpg')
                      actual_menu = 'start'
                      show_score = True

                if key[pygame.K_SPACE]:
                      wait = False

                if actual_menu == 'credits':
                if key[pygame.K_RIGHT]:
                      menu_img = pygame.image.load(SPRITES_DIRECTORY /
'start_menu.jpg')
                      actual_menu = 'start'
                      show_score = True

            self.surface.blit(menu_img, rect)
            if show_score:
                  self.show_higher_score()
            pygame.display.update()
```

Tareas que quedaron sin resolver

Clase Game ocupando muchas tareas

Investigando un poco sobre la POO me di cuenta que mi clase `Game` era demasiado grande y que tal vez estaba abarcando tareas que deberían delegarse a otras clases (como una clase que se encargue de manejar los textos en pantalla o el manejo del archivo `score.txt`).

Por falta de tiempo y de experiencia para el manejo de código entre clases, dejaré el código de esta forma, con la clase `Game` manipulando todos los métodos.

Dar estilos al texto de score

Para mostrar el score al inicio y al final de la partida, mi primera intención era agregar un sombreado al texto que le diera mayor contraste y se pudiera leer mejor en pantalla. Sin embargo, no encontré la forma de agregar este sombreado y tuve que apoyarme en editar las imágenes de las pantallas para crear cuadros blancos que dieran el contraste que buscaba.

Sobreposición de taladros y galletas

Durante la generación de los sprites para los taladros y para las gato-galletas, está la posibilidad de que se generen en el mismo espacio y se sobrepongan durante el juego.

La forma que se me ocurrió para arreglar esto era obtener las posiciones en la que se generaban los sprites de ambos objetos y evitar que se muestren en la misma coordenada de X pero se tendrían que mezclar ambos métodos de generación de los objetos.

Esta acción complicaba mucho el código y me pareció que el hecho de ver una gato galleta y un taladro muy pegados podría representar un reto para algunos jugadores. Por ahora veré este bug como un feature.