**PROJECTE**

**Entrega 19/02/2025**

**Objectiu: fer les ampliacions que creieu pertinents d’aquesta llista, la nota màxima és un 10.**

**Repositori de Github:** [**https://github.com/ilopez000/python\_game/**](https://github.com/ilopez000/python_game/)

**Entrega: codi i document amb les ampliacions implementades.**

**La presentació a classe incrementa en 1 la nota.**

|  |  |  |  |
| --- | --- | --- | --- |
| Nº | Ampliació | Descripció | Puntuació |
| 1 | Sistema de Puntuació | Incrementar la puntuació cada vegada que el jugador esquiva o supera un obstacle. | 1 |
| 2 | Dificultat Incremental | Augmentar progressivament la velocitat i la freqüència d'aparició dels obstacles amb el pas del temps. | 1,25 |
| 3 | Sistema de Vides | Permetre que el jugador disposi de diverses vides, de manera que una col·lisió no acabi la partida immediatament. | 1,25 |
| 4 | Menú d'Inici i Reinici | Implementar una pantalla d'inici per començar el joc i una pantalla de Game Over amb l'opció de reiniciar la partida. | 1,5 |
| 5 | Efectes de So i Música de Fons | Afegir efectes de so per a col·lisions i moviments, així com música de fons per millorar l'experiència del joc. | 1,5 |
| 6 | Power-ups | Introduir elements que, en ser recollits, atorguin millores temporals com invulnerabilitat, augment de velocitat o dispars extra. | 1,5 |
| 7 | Obstacles amb Comportaments Diferents | Crear obstacles que es moguin en patrons diferents, per exemple, obstacles que persegueixin al jugador o que es moguin de manera no lineal. | 1,5 |
| 8 | Pantalla de Pausa | Permetre pausar i reprendre el joc amb una tecla, mostrant un missatge o pantalla indicativa durant la pausa. | 1 |
| 9 | Marcador de Récords (High Scores) | Emmagatzemar la puntuació màxima aconseguida i mostrar-la en pantalla, permetent al jugador comparar els seus resultats. | 1,5 |
| 10 | Nivells i Progressió amb Escenaris Diferents | Dissenyar nivells diferenciats amb escenaris, objectius o temàtiques pròpies, amb una dificultat que augmenta progressivament. | 1,5 |
| 11 | Animacions i Efectes Visuals | Afegir animacions als moviments del jugador i dels obstacles, així com efectes visuals durant col·lisions o al recollir power-ups. | 1,5 |
| 12 | Sistema de Disparos | Permetre que el jugador dispari projectils per eliminar obstacles o enemics, afegint un element d'acció addicional al joc. | 1,0 |

**VideoJoc Amb Python**

Aquest projecte és un exemple senzill d’un joc desenvolupat en Python amb la llibreria Pygame. L'objectiu d'aquest projecte és proporcionar una base sòlida i fàcilment ampliable per a aquells que vulguin començar a crear els seus propis jocs, tot experimentant amb elements bàsics com el moviment del jugador, la generació d'obstacles i la detecció de col·lisions.

En aquest joc, el jugador controla un quadrat blau que es pot moure per la pantalla utilitzant les tecles de direcció. L'objectiu principal és esquivar obstacles vermells que apareixen aleatòriament per la part dreta de la pantalla i que es desplacen cap a l'esquerra. Quan es produeix una col·lisió entre el jugador i qualsevol d'aquests obstacles, el joc finalitza, mostrant la importància de la detecció de col·lisions en el desenvolupament de videojocs.

El codi del projecte està estructurat de manera modular, fent ús de la programació orientada a objectes per definir les diferents entitats del joc (com el jugador i els obstacles). Aquesta organització facilita la comprensió i posterior ampliació del projecte, permetent afegir fàcilment noves funcionalitats, com ara:

* Puntuació i nivells de dificultat: Incrementar la complexitat del joc amb el pas del temps o en funció dels punts aconseguits.
* Efectes sonors i música: Incorporar sons per millorar l'experiència immersiva.
* Menus i pantalles de benvinguda o de finalització: Crear una interfície d'usuari més completa.
* Nous elements i enemics: Afegir nous tipus d'obstacles o elements interactius per enriquir el joc.

**Llibreries utilitzades i per a què serveixen**

pygame

Aquesta llibreria és fonamental per desenvolupar jocs i aplicacions gràfiques amb Python. Permet gestionar la creació de finestres, el dibuix d'objectes, la reproducció de sons, la gestió d'entrades (teclat, ratolí, etc.) i molt més.

random

Serveix per generar nombres aleatoris. En aquest codi s'utilitza per donar variabilitat als obstacles, fent que tinguin mides, posicions i velocitats diferents en cada aparició.

sys

Aquesta llibreria s'utilitza per interactuar amb el sistema, per exemple, per acabar el programa de manera neta amb sys.exit().

**Configuració inicial i constants**

Es defineixen les dimensions de la finestra (WIDTH i HEIGHT), la velocitat del joc (FPS) i alguns colors bàsics (blanc, negre, vermell, blau). Aquesta configuració facilita l'ajust i la modificació de paràmetres clau del joc sense haver de canviar el codi en múltiples llocs.

WIDTH = 800

HEIGHT = 600

FPS = 60

WHITE = (255, 255, 255)

BLACK = (0, 0, 0)

RED = (255, 0, 0)

BLUE = (0, 0, 255)

**Inicialització de Pygame**

Es realitza la inicialització de Pygame, es crea la finestra de joc i es configura un rellotge (clock) per controlar la velocitat de refresc dels fotogrames.

pygame.init()

screen = pygame.display.set\_mode((WIDTH, HEIGHT))

pygame.display.set\_caption("Juego de Ejemplo Extensible")

clock = pygame.time.Clock()

**Definició de classes: Player i Obstacle**

El codi utilitza la programació orientada a objectes per estructurar els elements del joc. Així, cada element (jugador, obstacles) és un objecte amb propietats i mètodes que controlen el seu comportament.

Classe Player

Funció: Representa el jugador, que és un quadrat blau.

Constructor (\_\_init\_\_):

Crea una superfície de 50×50 píxels i la pinta de blau.

Estableix la posició inicial al centre esquerre de la pantalla.

Defineix una velocitat de moviment.

Mètode update:

Comprova les tecles premudes per moure el jugador en les quatre direccions.

Inclou condicions per evitar que el jugador surti dels límits de la finestra.

class Player(pygame.sprite.Sprite):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.image = pygame.Surface((50, 50))

self.image.fill(BLUE)

self.rect = self.image.get\_rect()

self.rect.center = (100, HEIGHT // 2)

self.speed = 5

def update(self):

keys = pygame.key.get\_pressed()

if keys[pygame.K\_UP]:

self.rect.y -= self.speed

if keys[pygame.K\_DOWN]:

self.rect.y += self.speed

if keys[pygame.K\_LEFT]:

self.rect.x -= self.speed

if keys[pygame.K\_RIGHT]:

self.rect.x += self.speed

if self.rect.left < 0:

self.rect.left = 0

if self.rect.right > WIDTH:

self.rect.right = WIDTH

if self.rect.top < 0:

self.rect.top = 0

if self.rect.bottom > HEIGHT:

self.rect.bottom = HEIGHT

**Classe Obstacle**

Funció: Representa els obstacles (quadrats o rectangles vermells) que apareixen a la dreta de la pantalla i es mouen cap a l'esquerra.

Constructor (\_\_init\_\_):

Es crea un obstacle amb dimensions aleatòries.

La seva posició inicial està fora de la pantalla per la dreta, amb una posició vertical aleatòria.

S'assigna una velocitat aleatòria per fer el moviment menys previsible.

Mètode update:

Mou l'obstacle cap a l'esquerra.

Quan l'obstacle surt completament de la pantalla, s'elimina (mitjançant self.kill()).

class Obstacle(pygame.sprite.Sprite):

def \_\_init\_\_(self):

super().\_\_init\_\_()

width = random.randint(20, 100)

height = random.randint(20, 100)

self.image = pygame.Surface((width, height))

self.image.fill(RED)

self.rect = self.image.get\_rect()

self.rect.x = WIDTH + random.randint(10, 100)

self.rect.y = random.randint(0, HEIGHT - height)

self.speed = random.randint(3, 7)

def update(self):

self.rect.x -= self.speed

if self.rect.right < 0:

self.kill()

**Grups de sprites**

S'utilitzen grups de sprites per gestionar de manera eficient tots els elements del joc.

all\_sprites: Conté tots els sprites (jugador i obstacles).

obstacles: Grup específic per als obstacles, el que facilita la detecció de col·lisions.

all\_sprites = pygame.sprite.Group()

obstacles = pygame.sprite.Group()

player = Player()

all\_sprites.add(player)

**Gestió d'esdeveniments i creació d'obstacles**

S'estableix un esdeveniment personalitzat (ADD\_OBSTACLE) que es dispara cada 1500 mil·lisegons (1,5 segons) per crear un nou obstacle i afegir-lo als grups corresponents.

ADD\_OBSTACLE = pygame.USEREVENT + 1

pygame.time.set\_timer(ADD\_OBSTACLE, 1500)

**Bucle principal del joc**

Aquest és el cor del joc, on es processen els esdeveniments, s'actualitzen els objectes i es dibuixa tot a la pantalla.

Control de la velocitat:

S'utilitza clock.tick(FPS) per assegurar que el joc s'executa a la mateixa velocitat en qualsevol màquina.

Gestió d'esdeveniments:

Si es tanca la finestra (pygame.QUIT), el bucle es trenca i el joc s'atura.

Si es rep l'esdeveniment ADD\_OBSTACLE, es crea un nou obstacle.

Actualització i detecció de col·lisions:

Es crida update() per a tots els sprites. A més, es comprova si el jugador ha col·lisionat amb algun obstacle amb pygame.sprite.spritecollideany(). En cas afirmatiu, es finalitza el joc.

Dibuix a la pantalla:

Es neteja la pantalla amb un fons blanc (screen.fill(WHITE)), es dibuixen tots els sprites i s'actualitza la pantalla amb pygame.display.flip().

running = True

while running:

clock.tick(FPS)

for event in pygame.event.get():

if event.type == pygame.QUIT:

running = False

elif event.type == ADD\_OBSTACLE:

obstacle = Obstacle()

all\_sprites.add(obstacle)

obstacles.add(obstacle)

all\_sprites.update()

if pygame.sprite.spritecollideany(player, obstacles):

print("¡Colisión! Fin del juego.")

running = False

screen.fill(WHITE)

all\_sprites.draw(screen)

pygame.display.flip()

pygame.quit()

sys.exit()

**Finalització del joc**

Quan el bucle principal s'acaba, es crida pygame.quit() per tancar la llibreria Pygame de manera ordenada i sys.exit() per finalitzar l'execució del programa.

**Ampliació1: Creació d'una variable global per la puntuació**

Al començament del codi, després d'inicialitzar la finestra i altres constants, es defineix una variable global score que emmagatzema la puntuació del joc:

# Variable global per la puntuació

score = 0

Aquesta variable s'incrementarà cada cop que un obstacle surti de la pantalla sense col·lidir amb el jugador.

**Inicialització de la font per mostrar la puntuació**

Per poder mostrar la puntuació en pantalla, s'ha creat una font amb pygame.font.SysFont. Això permet renderitzar el text que mostrarà el valor actual de la puntuació:

# Font per mostrar la puntuació

font = pygame.font.SysFont("Arial", 24)

La font s'utilitza més endavant dins del bucle principal per dibuixar el text a la pantalla.

**Modificació de la classe Obstacle**

En aquesta ampliació, s'ha modificat el mètode update de la classe Obstacle per incorporar el sistema de puntuació. Concretament:

* Es decrementa la posició x de l'obstacle per moure'l cap a l'esquerra.
* Quan l'obstacle surt completament de la pantalla (és a dir, quan self.rect.right < 0), s'incrementa la puntuació global i es crida a self.kill() per eliminar l'obstacle.

El codi modificat és el següent:

def update(self):

"""Actualitza la posició de l'obstacle movent-lo a l'esquerra.

Si surt de la pantalla, incrementa la puntuació i es elimina."""

global score

self.rect.x -= self.speed

if self.rect.right < 0:

score += 1 # Incrementar la puntuació

self.kill() # Eliminar l'obstacle

*Aspectes clau:*

* global score: S'indica que s'està fent servir la variable score definida fora de la funció.
* Increment de la puntuació: Cada vegada que un obstacle surt completament de la pantalla, la puntuació s'incrementa en 1.
* Eliminació de l'obstacle: Amb self.kill() s'eliminen els obstacles que ja no són visibles, per optimitzar el rendiment.

**Dibuix de la puntuació a la pantalla**

Dins del bucle principal del joc, després d'actualitzar i dibuixar tots els sprites, s'afegeix el codi per renderitzar i mostrar la puntuació en la pantalla:

# Renderitzar i mostrar la puntuació a la pantalla

score\_text = font.render("Puntuació: " + str(score), True, BLACK)

screen.blit(score\_text, (10, 10))

*Detalls:*

* font.render(...): Aquesta funció converteix el text (amb la puntuació actual) en una imatge que es pot dibuixar.
* screen.blit(score\_text, (10, 10)): Dibuixa el text a la posició (10, 10) de la pantalla, normalment a la cantonada superior esquerra.

**Ampliació 2: Variables Globals per la Dificultat i Control del Temps**

Dificultat inicial:

Es defineix la variable difficulty\_level, que comença amb el valor 1. Aquesta variable s'utilitza per augmentar la velocitat dels obstacles i per modificar la freqüència d'aparició.

difficulty\_level = 1

Control del temps:

Es registra el temps inicial amb pygame.time.get\_ticks() en la variable last\_difficulty\_update\_time per saber quan s'ha de pujar la dificultat. En aquest exemple, la dificultat s'incrementa cada 15.000 mil·lisegons (15 segons).

last\_difficulty\_update\_time = pygame.time.get\_ticks()

**Ajust de l'Interval d'Aparició d'Obstacles**

Interval d'aparició inicial:

Es defineix una variable spawn\_interval amb un valor inicial (1500 mil·lisegons). Aquesta variable determina cada quant temps es genera un nou obstacle.

spawn\_interval = 1500

Temporitzador d'obstacles:

Mitjançant pygame.time.set\_timer(), es programa un esdeveniment personalitzat (ADD\_OBSTACLE) que generarà obstacles a intervals definits pel spawn\_interval.

ADD\_OBSTACLE = pygame.USEREVENT + 1

pygame.time.set\_timer(ADD\_OBSTACLE, spawn\_interval)

Reducció progressiva de l'interval:

Cada vegada que han passat 15 segons, s'incrementa la dificultat i es redueix l'interval d'aparició dels obstacles. Es fa servir max(500, 1500 - difficulty\_level \* 100) per assegurar que l'interval no sigui inferior a 500 ms.

if current\_time - last\_difficulty\_update\_time >= 15000:

difficulty\_level += 1

last\_difficulty\_update\_time = current\_time

spawn\_interval = max(500, 1500 - difficulty\_level \* 100)

pygame.time.set\_timer(ADD\_OBSTACLE, spawn\_interval)

print("Dificultat incrementada a:", difficulty\_level)

**Augment de la Velocitat dels Obstacles**

Velocitat en funció de la dificultat:

En la classe Obstacle, la velocitat de cada obstacle es defineix afegint el valor de difficulty\_level al rang base de velocitat (entre 3 i 7). D'aquesta manera, a mesura que la dificultat augmenta, els obstacles es mouen més ràpid.

self.speed = random.randint(3 + difficulty\_level, 7 + difficulty\_level)

Moviment dels obstacles:

El mètode update de la classe Obstacle mou l'obstacle cap a l'esquerra utilitzant la seva velocitat. Quan un obstacle surt completament de la pantalla, s'incrementa la puntuació i es elimina.

def update(self):

global score

self.rect.x -= self.speed

if self.rect.right < 0:

score += 1

self.kill()

**Visualització en Pantalla de la Puntuació i la Dificultat**

Creació de la font:

Es crea una font amb pygame.font.SysFont per poder renderitzar textos a la pantalla (tant per la puntuació com per la dificultat).

font = pygame.font.SysFont("Arial", 24)

Renderització del text:

A cada iteració del bucle principal, es renderitza el text per mostrar la puntuació i el nivell de dificultat. Aquest text es dibuixa a la pantalla en posicions determinades (per exemple, la puntuació a (10, 10) i la dificultat a (10, 40)).

score\_text = font.render("Puntuació: " + str(score), True, BLACK)

difficulty\_text = font.render("Dificultat: " + str(difficulty\_level), True, BLACK)

screen.blit(score\_text, (10, 10))

screen.blit(difficulty\_text, (10, 40))

**Ampliació3: Afegiment del Sistema de Vides**

Nova Variable Global:

S'ha afegit la variable lives, inicialitzada a 3, que determina el nombre d'oportunitats que té el jugador abans que el joc acabi.

lives = 3

Funcionalitat de Vides:

En lloc d'acabar el joc immediatament en detectar una col·lisió entre el jugador i un obstacle, ara es decrementa el nombre de vides. Si encara en queden, es reinicialitza la posició del jugador i s'esborren tots els obstacles de la pantalla, donant al jugador una nova oportunitat per continuar jugant.

Gestió de les Col·lisions amb Vides

Detecció de Col·lisions:

Quan es detecta una col·lisió amb algun obstacle, en comptes d'acabar el joc, es decrementa la variable lives.

if pygame.sprite.spritecollideany(player, obstacles):

lives -= 1

Accions en Funció del Nombre de Vides Restants:

Si encara queden vides:

Es reinicialitza la posició del jugador i s'esborren tots els obstacles per permetre una recuperació, evitant així una penalització massa severa per una única col·lisió.

if lives > 0:

player.rect.center = (100, HEIGHT // 2)

for obs in obstacles:

obs.kill()

print("Has perdut una vida! Vides restants:", lives)

Si les vides s'esgoten:

Quan lives arriba a 0, es mostra un missatge de "Game Over" amb la puntuació final, i es finalitza el joc.

else:

print("Game Over! Puntuació final:", score)

running = False

**Visualització de les Vides en Pantalla**

Mostrant les Vides Restants:

S'ha afegit el renderitzat d'un nou element de text a la interfície del joc per mostrar en temps real el nombre de vides que li queden al jugador. Aquesta informació es dibuixa a la pantalla, normalment a la cantonada superior esquerra, just a sota de la puntuació i el nivell de dificultat.

lives\_text = font.render("Vides: " + str(lives), True, BLACK)

screen.blit(lives\_text, (10, 70))

**Integració amb la Lògica del Joc Existente**

Manteniment dels Altres Elements:

El sistema de puntuació i el mecanisme de dificultat incremental (on la velocitat dels obstacles i la freqüència d'aparició es modifiquen amb el temps) es manté intacte.

Adaptació de la Lògica:

La lògica per a la generació d'obstacles, el moviment dels mateixos i la detecció de col·lisions s'ha integrat amb el nou sistema de vides, assegurant que la partida només es finalitzi quan el jugador es quedi sense vides.

**Ampliació 4**

Funció show\_menu():

Aquesta funció crea una pantalla inicial que mostra el títol del joc i una instrucció ("Prem qualsevol tecla per començar").

Es crea un bucle que s'executa mentre la variable waiting sigui certa.

S'escolten els esdeveniments i, si es prem qualsevol tecla, es surt del bucle.

Es pinta la pantalla de color blanc i es dibuixen els textos d'informació mitjançant la funció auxiliar draw\_text().

Quan l'usuari prem una tecla, es trenca el bucle i es continua amb la partida.

Reinicialització del Joc

Funció new\_game():

Aquesta funció és cridada cada cop que s'inicia una nova partida.

Reinicialitza les variables globals: score, difficulty\_level, lives, etc.

Restableix el temporitzador dels obstacles amb pygame.time.set\_timer.

Crea nous grups de sprites (all\_sprites i obstacles) i el jugador.

Així, cada partida comença amb un estat net.

Bucle de la Partida

Funció game\_loop():

Aquesta funció conté el bucle principal del joc, on es duen a terme les següents operacions:

Gestió d'esdeveniments:

Es comprova si l'usuari tanca la finestra o si arriba l'esdeveniment per afegir un obstacle (ADD\_OBSTACLE).

Increment de la dificultat:

Cada 15 segons s'incrementa el difficulty\_level, es redueix l'interval de generació d'obstacles (fins a un mínim de 500 ms) i es reinicia el temporitzador.

Actualització dels sprites:

Es crida al mètode update() de cada sprite per moure'ls.

Detecció de col·lisions:

Si el jugador col·lisiona amb algun obstacle:

Es decrementa el nombre de lives.

Si encara hi ha vides, es reinicialitza la posició del jugador i s'eliminen els obstacles.

Si no hi ha vides, es canvia l'estat del joc a "game\_over".

Dibuix de la interfície:

Es pinta la pantalla amb color blanc i s'hi dibuixen els sprites i els textos de la puntuació, la dificultat i les vides.

Al final del bucle, quan el jugador es queda sense vides, la funció retorna la puntuació final.

Pantalla de Game Over i Reinici

Funció show\_game\_over(final\_score):

Aquesta funció s'encarrega de mostrar una pantalla de final de joc:

Es mostra un missatge "Game Over!" amb la puntuació final.

Es demana a l'usuari que premi qualsevol tecla per reiniciar.

Un cop premuda, la funció finalitza, permetent al bucle principal reiniciar el joc tornant al menú d'inici.

Bucle Principal del Programa

El bucle while True: al final del codi coordina tot el flux:

Primer, es mostra el menú d'inici amb show\_menu().

Després, s'executa el joc amb game\_loop(), que retorna la puntuació final.

Finalment, es mostra la pantalla de Game Over amb show\_game\_over(final\_score), i un cop l'usuari prem una tecla, es torna al menú d'inici per començar una nova partida.

**Data actualització i autor/res:** Ignacio López-Aylagas 09/12/2024