CFGS Desenvolupament d'aplicacions multiplataforma
Curs 2021-2022

Pràctica/Projecte UD1: Joc amb PyGame

Desenvolupament d'interfícies

IES Jaume II El Just
Tavernes de la Valldigna

Professor:
Maria Sales Puig
Ferran Cunyat Pellicer

Continguts

1 Introducció

PyGame és una llibreria de Python3 pensada per a desenvolupar de forma senzilla jocs amb Python. Anem a utilitzar-la per a desenvolupar un xicotet joc. Serà la primera activitat avaluable del curs.

2 Instal·lació i configuració

El primer que farem és activar el nostre entorn virtual de desenvolupament i instal·lar la llibreria pygame. Per a això recordeu que utilitzarem venv tal com ve explicat a la teoria. Dins l'entorn virtual instal·lem pygame.

```
$ source .venv/bin/activate
(.venv) $ pip install pygame
```

3 *Hola món!* del pygame

Aquest programa crea una finestra, omple el fons de blanc i dibuixa un cercle blau al mig:

```
# Simple pygame program

# Import and initialize the pygame library
import pygame
pygame.init()

# Set up the drawing window
screen = pygame.display.set_mode([500, 500])

# Run until the user asks to quit
running = True
while running:

    # Did the user click the window close button?
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    # Fill the background with white
    screen.fill((255, 255, 255))

    # Draw a solid blue circle in the center
    pygame.draw.circle(screen, (0, 0, 255), (250, 250), 75)

    # Flip the display
    pygame.display.flip()
```

```
# Done! Time to quit.
pygame.quit()
```

Fixem-nos en alguns punts:

- La línia 8 configura la finestra de visualització del programa, amb unes dimensions de 500x500 pixels.
- Les línies 11 i 12 configuren un bucle de joc per controlar quan finalitza el programa.
- Les línies 15 a 17 recullen i gestionen esdeveniments dins del bucle del joc. L'únic esdeveniment gestionat de moment és pygame.QUIT, que es produeix quan l'usuari fa clic al botó de tancament de la finestra.
- La línia 20 omple la finestra d'un color sòlid. *screen.fill()* accepta una llista o una tupla que especifica els valors RGB del color. (255, 255, 255), és el color blanc.
- La línia 23 dibuixa un cercle a la finestra, utilitzant els paràmetres (finestra on dibuixar, color, posició central i radi)
- La línia 26 actualitza el contingut de la pantalla, que de moment no canvia al llarg del temps.
- La línia 29 ix del joc, sols s'executa quan ix del bucle per l'event QUIT.

Aquesta és la versió pygame de "Hola, món".

4 Disseny del joc

Una vegada vist els conceptes bàsics anem a desenvolupar un xicotet joc a mode d'exemple.

L'objectiu del joc és evitar obstacles que entren:

- El jugador comença a la part esquerra de la pantalla.
- Els obstacles entren a l'atzar per la dreta i es mouen a l'esquerra en línia recta.
- El jugador pot moure's cap a l'esquerra, cap a la dreta, cap amunt o cap avall per evitar els obstacles.
- El jugador no pot ixir-se'n de la pantalla.
- El joc finalitza quan el jugador és colpejat per un obstacle o quan l'usuari tanca la finestra. Mentre no es produisca açò, el joc continua, pot ser infinit.

4.1 Importem i inicialitzem el joc

```
# Import the pygame module
import pygame

# Import pygame.locals for easier access to key coordinates
# Updated to conform to flake8 and black standards
from pygame.locals import (
    K_UP,
    K_DOWN,
    K_LEFT,
    K_RIGHT,
    K_ESCAPE,
    KEYDOWN,
    QUIT
)

# Initialize pygame
pygame.init()
```

Utilitzem les constants definides en *locals* per veure quina tecla s'ha apretat, o si s'ha pulsat sobre l'aspa de tancar el programa.

4.2 Configurem la pantalla

```
# Define constants for the screen width and height
SCREEN_WIDTH = 800
SCREEN_HEIGHT = 600

# Create the screen object
```

```
# The size is determined by the constant SCREEN_WIDTH and SCREEN_HEIGHT
screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))
```

Definim una finestra és de 800x600, utilitzant les constants SCREEN_WIDTH i SCREEN_HEIGHT. El mètode ens torna una superfície que representa les dimensions interiors de la finestra. Aquesta és la part de la finestra que podem controlar, mentre que el sistema operatiu controla els límits de la finestra i la barra de títol.

Si executeu aquest programa ara, apareixerà una finestra que apareixerà breument i desapareixerà immediatament.

4.3 Configuració del bucle principal del joc

Tots els jocs des de Pong fins a Fortnite utilitzen un bucle de joc per controlar-lo. Fa quatre coses molt importants:

1. Processa l'entrada de l'usuari
2. Actualitza l'estat de tots els objectes del joc
3. Actualitza la pantalla i l'àudio
4. Manté la velocitat del joc

Tots els cicles del bucle del joc s'anomenen fotogrames i, com més ràpid es facen les coses a cada cicle, més ràpid es desenvoluparà el vostre joc. Els fotogrames continuen ocorrent fins que es compleix alguna condició per ixir del joc. Al vostre disseny, hi ha dues condicions que poden acabar amb el bucle del joc:

1. El jugador xoca amb un obstacle. (Més endavant, cobrirà la detecció de col·lisions.)
2. El jugador tanca la finestra (esdeveniment QUIT).

El primer que fa el bucle del joc és processar l'entrada de l'usuari per permetre al jugador moure's per la pantalla. Per tant, necessiteu alguna manera de capturar i processar una gran quantitat d'informacions. Ho farem mitjançant el sistema d'esdeveniments pygame.

4.4 Processament d'esdeveniments

Prémer una tecla, moure el ratolí o el joystick són algunes de les maneres en què un usuari pot proporcionar informació. Totes aquestes accions de l'usuari donen lloc a la generació d'un esdeveniment i poden passar en qualsevol moment.

Tots els esdeveniments de Pygame es col·loquen a la **cua d'esdeveniments**, la qual es pot accedir i manipular. El tractament que es fa dels esdeveniments s'anomena gestor d'esdeveniments.

Tots els esdeveniments de Pygame tenen associat un tipus d'esdeveniment. Per al vostre joc, els tipus d'esdeveniments en què us centreu són les pulsacions de tecles i el tancament de la finestra.

- Els esdeveniments de premuda de tecles tenen el tipus d'esdeveniment KEYDOWN.
- L'esdeveniment de tancament de finestra té el tipus QUIT.

Els diferents tipus d'esdeveniments també poden tenir associades altres dades. Per exemple, el tipus d'esdeveniment KEYDOWN també té una variable anomenada *key* per indicar quina tecla s'ha premut.

Accediu a la llista de tots els esdeveniments actius a la cua trucant amb *pygame.event.get()*. A continuació, passeu per aquesta llista, inspeccioneu cada tipus d'esdeveniment i tracteu-los:

```
# Variable to keep the main loop running
running = True

# Main loop
while running:
    # Look at every event in the queue
    for event in pygame.event.get():
        # Did the user hit a key?
        if event.type == KEYDOWN:
            # Was it the Escape key? If so, stop the loop.
            if event.key == K_ESCAPE:
                running = False

        # Did the user click the window close button? If so, stop the loop
            .
        elif event.type == QUIT:
            running = False
```

Vegem més a fons aquest bucle principal del joc:

1. S'estableix una variable de control per al bucle del joc. Per a ixir del bucle i del joc, hem d'establir la variable *running = False*.
2. Recorrem el gestor d'esdeveniments, recollint tots els esdeveniments que hi ha actualment a la cua d'esdeveniments. Si no hi ha esdeveniments, la llista està buida i el gestor no farà res.
3. Per a cada esdeveniment de la cua, comprovem el tipus. Si *event.type == KEYDOWN*, s'ha apretat alguna tecla. Si és així, comprovem quina tecla s'ha apretat mirant l'atribut *event.key*. Si la clau és la tecla Esc, indicada per K_ESCAPE, surt del bucle del joc configurant running = False.
4. Es fa una comprovació similar per al tipus d'esdeveniment *QUIT*. Aquest esdeveniment només es produeix quan l'usuari fa clic al botó de tancament de la finestra. L'usuari també pot utilitzar qualsevol altra acció del sistema operatiu per tancar la finestra.

Quan afegiu aquestes línies al codi anterior i l'executeu, veureu una finestra amb una pantalla en blanc o negre. La finestra no desapareixerà fins que no premeu la tecla Esc tanqueu la finestra amb el ratolí

o combinació de tecles *Alt + F4*. (Comproveu-ho, però recordeu afegir *pygame.display.flip()* per a que pinte la pantalla al bucle).

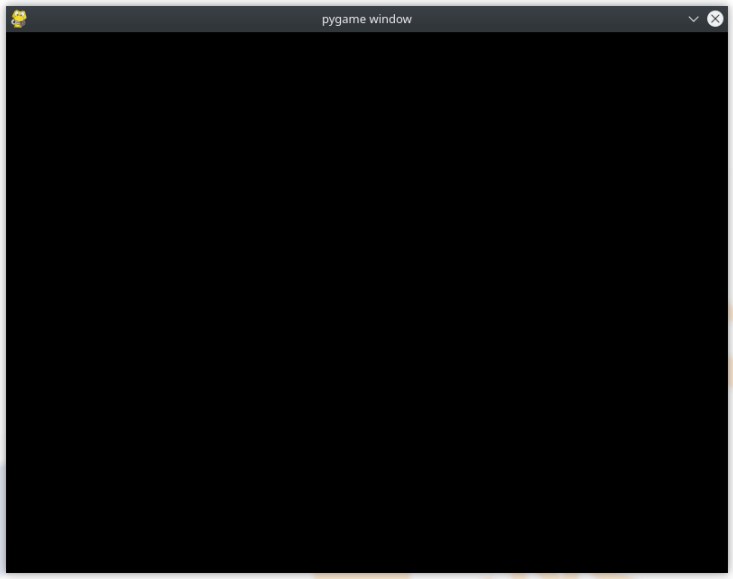

Figura 1: Finestra del joc

4.5 Pintant objectes a la pantalla

Al programa de mostra, hem dibuixat a la pantalla mitjançant dos mètodes:

- *screen.fill()* per omplir el fons
- *pygame.draw.circle()* per dibuixar un cercle
- Ara veurem una tercera manera de dibuixar a la pantalla: utilitzar una superfície *Surface*.

Una superfície és un objecte rectangular sobre el qual podem dibuixar, com un full de paper en blanc. La pantalla és una superfície i podeu crear els vostres propis objectes de superfície separats de la pantalla. Vegem com funciona:

```
# Fill the screen with white
screen.fill((255, 255, 255))

# Create a surface and pass in a tuple containing its length and width
surf = pygame.Surface((50, 50))

# Give the surface a color to separate it from the background
surf.fill((0, 0, 0))
rect = surf.get_rect()
```

A la primera línia la pantalla s'omple de blanc. Després es crea una nova superfície de 50 píxels d'ample, 50 píxels d'alçada i s'assigna a *surf*. Ara surf és una superfície igual que la pantalla principal i l'omplim de negre. Accedim al seu rectangle subjacentmitjançant *get_rect()* per poder-ho utilitzar posteriorment.

4.6 Ús de .blit () i .flip ()

El fet de crear una nova superfície no és suficient per veure-la a la pantalla. Per fer-ho, cal que col·loqueu la superfície sobre una altra superfície. El terme blit significa Block Transfer (Transferència de blocs) i *.blit()* és la forma de copiar el contingut d'una superfície a una altra. Només podeu copiar el contingut entre superfícies, però ja haviem dit que screen és una superfície. Vegem com dibuixem *surf* a la pantalla:

```
# This line says "Draw surf onto the screen at the center"
screen.blit(surf, (SCREEN_WIDTH/2, SCREEN_HEIGHT/2))
pygame.display.flip()
```

La funció *blit* pren dos arguments: 1. La superfície a pintar 2. Les coordenades del vèrtex *top-left*

Per tant si ho volem pintar exactament al centre de la pantalla hauriem de restar la seua amplada i altura respectivament:

```
# Put the center of surf at the center of the display
surf_center = (
    (SCREEN_WIDTH-surf.get_width())/2,
    (SCREEN_HEIGHT-surf.get_height())/2
)

# Draw surf at the new coordinates
screen.blit(surf, surf_center)
pygame.display.flip()
```

Recordeu fer la crida a la funció *flip* per a repintar el joc.

5 Sprites

Al disseny del joc, el jugador comença per l'esquerra i els obstacles entren per la dreta. Podeu representar tots els obstacles amb objectes de la superfície per fer tot el dibuix més fàcil, però com sabeu on dibuixar-los? Com se sap si un obstacle ha xocat amb el jugador? Què passa quan l'obstacle ix de la pantalla? Què passa si voleu dibuixar imatges de fons que també es moguin? Què passa si voleu que les vostres imatges siguen animades? Podeu gestionar totes aquestes situacions i molt més amb els *sprites*.

En termes de programació, un sprite és una representació 2D d'alguna cosa a la pantalla. Essencialment, és una imatge. pygame proporciona una classe Sprite, que està dissenyada per contenir una o diverses representacions gràfiques de qualsevol objecte de joc que vulgueu mostrar a la pantalla. Per utilitzar-lo, creeu una nova classe que herede d'Sprite. Això us permet utilitzar els seus mètodes heredats.

5.1 Jugadors

A continuació s'explica com s'utilitzen els objectes Sprite amb el joc actual per definir el jugador.

```
# Define a Player object by extending pygame.sprite.Sprite
# The surface drawn on the screen is now an attribute of 'player'
class Player(pygame.sprite.Sprite):
    def __init__(self):
        super(Player, self).__init__()
        self.surf = pygame.Surface((75, 25))
        self.surf.fill((255, 255, 255))
        self.rect = self.surf.get_rect()
```

Primer definiu Player ampliant pygame.sprite.Sprite. Després `.__init__()` utilitza `.super()` per cridar al constructor de la classe pare.

A continuació, definim i inicialitzem .surf per mantindre la imatge que voleu mostrar, que actualment és un quadre blanc. També definim i inicialitzem .rect, que s'utilitzarà més endavant. Per utilitzar aquesta nova classe, heu de crear un objecte nou i canviar també el codi de dibuix. Amplieu el bloc de codi següent per veure-ho tot junt:

```
# Import the pygame module
import pygame

# Import pygame.locals for easier access to key coordinates
# Updated to conform to flake8 and black standards
from pygame.locals import (
    K_UP,
    K_DOWN,
```

```
    K_LEFT,
    K_RIGHT,
    K_ESCAPE,
    KEYDOWN,
    QUIT,
)

# Define constants for the screen width and height
SCREEN_WIDTH = 800
SCREEN_HEIGHT = 600

# Define a player object by extending pygame.sprite.Sprite
# The surface drawn on the screen is now an attribute of 'player'
class Player(pygame.sprite.Sprite):
    def __init__(self):
        super(Player, self).__init__()
        self.surf = pygame.Surface((75, 25))
        self.surf.fill((255, 255, 255))
        self.rect = self.surf.get_rect()

# Initialize pygame
pygame.init()

# Create the screen object
# The size is determined by the constant SCREEN_WIDTH and SCREEN_HEIGHT
screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))

# Instantiate player. Right now, this is just a rectangle.
player = Player()

# Variable to keep the main loop running
running = True

# Main loop
while running:
    # for loop through the event queue
    for event in pygame.event.get():
        # Check for KEYDOWN event
        if event.type == KEYDOWN:
            # If the Esc key is pressed, then exit the main loop
            if event.key == K_ESCAPE:
                running = False
        # Check for QUIT event. If QUIT, then set running to false.
        elif event.type == QUIT:
            running = False

    # Fill the screen with black
    screen.fill((0, 0, 0))

    # Draw the player on the screen
    #screen.blit(player.surf, (SCREEN_WIDTH/2, SCREEN_HEIGHT/2))
```

```
    screen.blit(player.surf, player.rect)

    # Update the display
    pygame.display.flip()
```

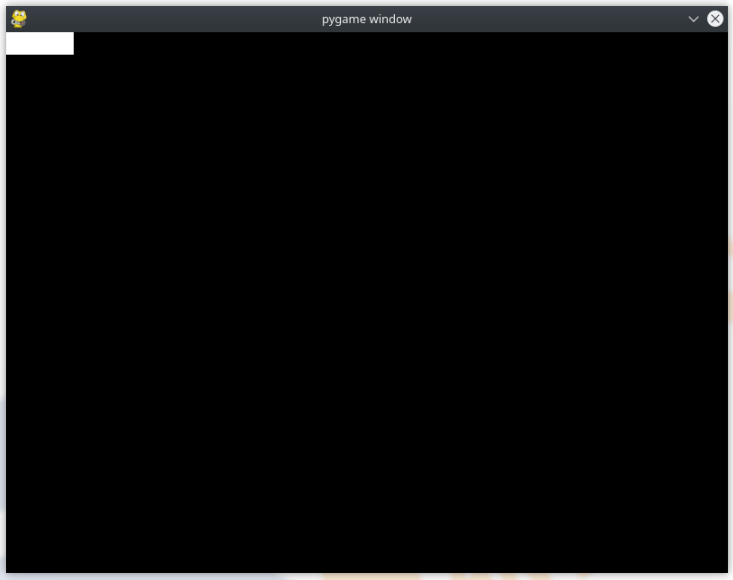

Figura 2: Player

5.2 Entrada d'usuari

Fins ara, hem après a configurar Pygame i dibuixar objectes a la pantalla. Ara comença la diversió. Fareu que el joc es puga controlar mitjançant el teclat.

Abans, haviem vist que *pygame.event.get()* retorna la llista dels esdeveniments de la cua, que analitzem per trobar els seus tipus. Bé, aquesta no és l'única manera de llegir les tecles. *pygame* també proporciona *pygame.event.get_pressed()*, que retorna un diccionari que conté tots els esdeveniments KEYDOWN actuals a la cua.

Posar-ho al bucle del joc després del bucle de gestió d'esdeveniments torna un diccionari que conté les tecles apretades al començament de cada fotograma.

A continuació, escrivim un mètode a Player per analitzar aquest diccionari. Això definirà el comportament del sprite a partir de les tecles que es premen.

```
# Get the set of keys pressed and check for user input
pressed_keys = pygame.key.get_pressed()

# Move the sprite based on user keypresses
```

```
def update(self, pressed_keys):
    if pressed_keys[K_UP]:
        self.rect.move_ip(0, -5)
    if pressed_keys[K_DOWN]:
        self.rect.move_ip(0, 5)
    if pressed_keys[K_LEFT]:
        self.rect.move_ip(-5, 0)
    if pressed_keys[K_RIGHT]:
        self.rect.move_ip(5, 0)
```

K_UP, K_DOWN, K_LEFT i K_RIGHT corresponen a les tecles de les fletxes del teclat. Utilitzarem *.move_ip()*, que significa moure des del punt actual, i reb per paràmetre el número de pixels a moure's en horitzontal i en vertical com si es tractara d'un eix de coordenades.

A continuació, cridem a *.update()* cada fotograma per moure el sprite del jugador en resposta a les pulsacions de tecles.

```
# Main loop
while running:
    # for loop through the event queue
    for event in pygame.event.get():
        # Check for KEYDOWN event
        if event.type == KEYDOWN:
            # If the Esc key is pressed, then exit the main loop
            if event.key == K_ESCAPE:
                running = False
        # Check for QUIT event. If QUIT, then set running to false.
        elif event.type == QUIT:
            running = False

    # Get all the keys currently pressed
    pressed_keys = pygame.key.get_pressed()

    # Update the player sprite based on user keypresses
    player.update(pressed_keys)

    # Fill the screen with black
    screen.fill((0, 0, 0))
```

Amb això, ja veiem que el Player es mou tant en horitzontal com en vertical.

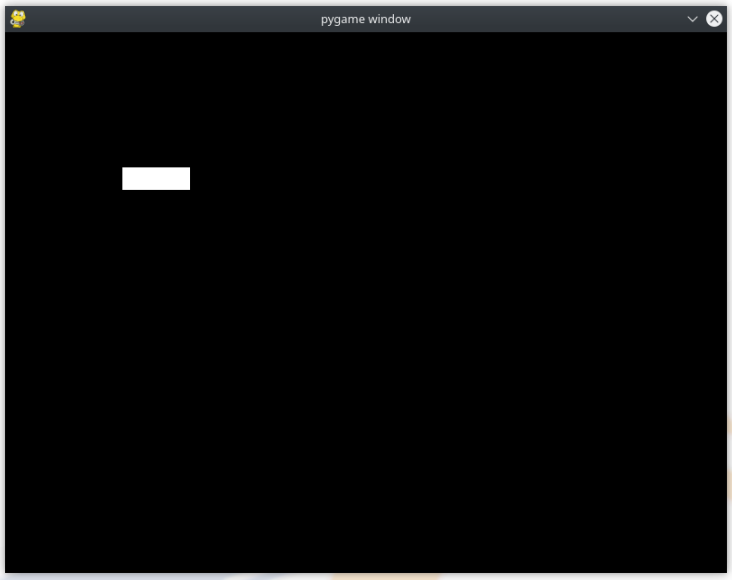

Figura 3: Moviment del jugador en resposta a pulsació de tecles

És possible que noteu dos problemes:

1. El rectangle del jugador es mou molt ràpid. Ho solucionarem més endavant.
2. El rectangle del jugador pot ixir-se'n de la pantalla. Solucionem-ho ara.

Per mantenir el jugador a la pantalla, cal afegir la lògica per detectar les colisions entre el jugador i els límits de la pantalla. Per fer-ho, comprovem si les coordenades del rectangle s'han desplaçat més enllà del límit de la pantalla. Si és així, indiquem al programa que el torne a la vora, quedant la funció update com a continuació s'indica:

```
# Move the sprite based on user keypresses
def update(self, pressed_keys):
    if pressed_keys[K_UP]:
        self.rect.move_ip(0, -5)
    if pressed_keys[K_DOWN]:
        self.rect.move_ip(0, 5)
    if pressed_keys[K_LEFT]:
        self.rect.move_ip(-5, 0)
    if pressed_keys[K_RIGHT]:
        self.rect.move_ip(5, 0)

    # Keep player on the screen
    if self.rect.left < 0:
        self.rect.left = 0
    if self.rect.right > SCREEN_WIDTH:
        self.rect.right = SCREEN_WIDTH
    if self.rect.top <= 0:
```

```
            self.rect.top = 0
        if self.rect.bottom >= SCREEN_HEIGHT:
            self.rect.bottom = SCREEN_HEIGHT
```

Ací, en lloc d'utilitzar *.move()*, només heu de canviar les coordenades corresponents de .top, .bottom, .left o .right directament. Proveu-ho i veureu que el rectangle del reproductor ja no pot ixir-se'n de la pantalla.

5.3 Enemics

Què és un joc sense enemics? Utilitzarem les mateixes tècniques que ja hem après per crear una classe enemiga bàsica i, a continuació, crearem moltes instàncies d'aquesta per a que el jugador intente evitar-les. Primer, importeu la llibreria *random*. A continuació, creeu una nova classe de *sprite* anomenada *Enemy*, seguint el mateix patró que utilitzarem per a *Player*:

```
# Import random for random numbers
import random

# Define the enemy object by extending pygame.sprite.Sprite
# The surface you draw on the screen is now an attribute of 'enemy'
class Enemy(pygame.sprite.Sprite):
    def __init__(self):
        super(Enemy, self).__init__()
        self.surf = pygame.Surface((20, 10))
        self.surf.fill((255, 255, 255))
        self.rect = self.surf.get_rect(
            center=(
                random.randint(SCREEN_WIDTH + 20, SCREEN_WIDTH + 100),
                random.randint(0, SCREEN_HEIGHT),
            )
        )
        self.speed = random.randint(5, 20)

    # Move the sprite based on speed
    # Remove the sprite when it passes the left edge of the screen
    def update(self):
        self.rect.move_ip(-self.speed, 0)
        if self.rect.right < 0:
            self.kill()
```

Hi ha quatre diferències notables entre Enemic i Jugador:

1. Quan creem un enemic ho fem a una ubicació aleatòria al llarg de la vora dreta de la pantalla. Es troba en una posició entre 20 i 100 píxels de distància de la vora dreta i en algun lloc entre la vora superior i la inferior. De forma que al principi no serà visible i anirà apareixent per la vora dreta de la pantalla.

2. Definim una velocitat *speed* com un número aleatori entre 5 i 20. Això especifica la velocitat amb què aquest enemic es mou cap al jugador.
3. *.update()* no necessita arguments, ja que els enemics es mouen automàticament cap a l'esquerra a la velocitat aleatòria definida quan es va crear i que ja no canvia.
4. Comprovem si l'enemic s'ha mogut fora de la pantalla al sobrepassar la vora esquerra. Per assegurar-nos que l'enemic estiga completament fora de la pantalla i que no desaparega mentre encara siga visible, comprovem que el costat dret de *.rect* haja sobrepassat el costat esquerre de la pantalla. Una vegada que l'enemic es troba fora de pantalla, cridem a *.kill()* per evitar anar consumint més recursos cada vegada.

Què fa *.kill()*? Per saber-ho, estudiem els *Sprite Groups*.

6 Sprite Groups

Una altra classe súper útil que proporciona Pygame són els *Sprite Groups*. Es tracta d'un objecte que conté un grup d'objectes Sprite. Aleshores, per què utilitzar-lo? No podem fer el seguiment dels nostres objectes Sprite en una llista? Bé, podem, però l'avantatge d'utilitzar un grup radica en els mètodes que exposa. Aquests mètodes ajuden a detectar si algun enemic ha xocat amb el jugador, cosa que facilita les actualitzacions.

Vegem com crear *Sprite Group*. Creem dos objectes de grup diferents:

1. El primer grup tindrà tots els Sprite del joc.
2. El segon grup tindrà només els objectes enemics.

A continuació, es mostra el codi:

```
# Create the 'player'
player = Player()

# Create groups to hold enemy sprites and all sprites
# - enemies is used for collision detection and position updates
# - all_sprites is used for rendering
enemies = pygame.sprite.Group()
all_sprites = pygame.sprite.Group()
all_sprites.add(player)

# Variable to keep the main loop running
running = True
```

Quan cridem el mètode *.kill()*, el Sprite s'elimina de tots els grups als quals pertany. Això també elimina les referències al Sprite, cosa que permet al *garbage collector* de Python recuperar la memòria quan siga necessari.

Ara que teniu un grup all_sprites, podeu canviar la manera com es dibuixen els objectes. En lloc de cridar a *.blit()* només amb Player, podem repintar tot sobre all_sprites:

```
# Fill the screen with black
screen.fill((0, 0, 0))

# Draw all sprites
for entity in all_sprites:
    screen.blit(entity.surf, entity.rect)

# Flip everything to the display
pygame.display.flip()
```

Ara, qualsevol objecte d'*all_sprites* es redibuixarà a tots els fotogrames, ja sigui un enemic o un jugador.

Només hi ha un problema … No tenim cap enemic. Podriem crear un munt d'enemics al principi del joc, però el joc es tornaria complicadíssim a l'apareixer tots junts. En el seu lloc, explorem com mantenir un subministrament constant d'enemics que arriben a mesura que avance el joc.

7 Esdeveniments personalitzats

El disseny demana que apareguen enemics a intervals regulars. Això significa que, a intervals establerts, hem de fer dues coses:

1. Crea un enemic nou.
2. Afegiu-lo a all_sprites i a enemics. (Sprite Groups)

Ja teniu codi que gestiona esdeveniments aleatoris. El bucle d'esdeveniments està dissenyat per buscar esdeveniments aleatoris que es produeixen a cada fotograma i tractar-los adequadament. Per sort, pygame no us limita a utilitzar només els tipus d'esdeveniments que té predefinits. Podeu definir els vostres propis esdeveniments per gestionar-los segons convinga.

Vegem com es crea un esdeveniment personalitzat que es genera cada pocs segons. Podeu crear un esdeveniment personalitzat com es mostra a continuació:

```
# Create the screen object
# The size is determined by the constant SCREEN_WIDTH and SCREEN_HEIGHT
screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))

# Create a custom event for adding a new enemy
ADDENEMY = pygame.USEREVENT + 1
pygame.time.set_timer(ADDENEMY, 250)

# Instantiate player. Right now, this is just a rectangle.
player = Player()
```

pygame defineix els esdeveniments internament com a enters, de manera que cal definir un nou esdeveniment amb un enter únic. L'últim esdeveniment reservat a Pygame es diu *USEREVENT*, de manera que definir *ADDENEMY = pygame.USEREVENT + 1* garanteix que siga únic.

A continuació, heu d'insertar aquest nou esdeveniment a la cua d'esdeveniments a intervals regulars durant tot el joc. Necessitem d'alguna forma gestionar el temps, per això utilitzarem el mòdul de temps.

Disparem el nou esdeveniment ADDENEMY cada 250 mil·lisegons, o quatre vegades per segon. Per això farem una crida a *.set_timer()* fora del bucle del joc, ja que només necessitem un temporitzador, però es dispararà durant tot el joc cada 250 milisegons.

Afegim el codi per gestionar el nostre nou esdeveniment.

```
# Main loop
while running:
    # Look at every event in the queue
    for event in pygame.event.get():
        # Did the user hit a key?
        if event.type == KEYDOWN:
```

```
            # Was it the Escape key? If so, stop the loop.
            if event.key == K_ESCAPE:
                running = False

        # Did the user click the window close button? If so, stop the loop
            .
        elif event.type == QUIT:
            running = False

        # Add a new enemy?
        elif event.type == ADDENEMY:
            # Create the new enemy and add it to sprite groups
            new_enemy = Enemy()
            enemies.add(new_enemy)
            all_sprites.add(new_enemy)

    # Get the set of keys pressed and check for user input
    pressed_keys = pygame.key.get_pressed()
    player.update(pressed_keys)

    # Update enemy position
    enemies.update()
```

Sempre que el gestor d'esdeveniments veu el nou esdeveniment ADDENEMY, crea un enemic i l'afegeix a *enemies* i a *all_sprites*. Com que *Enemy* està en *all_sprites*, es dibuixarà a cada fotograma. També heu de cridar a *enemies.update()*, que actualitza totes les posicions dels enemics.

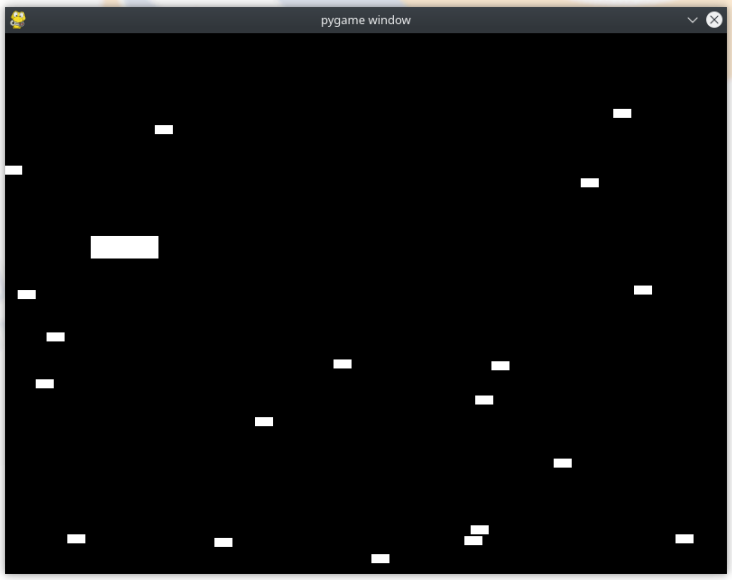

Figura 4: Moviment dels enemics

8 Detecció de col·lisions

El disseny del joc demana que finalitze el joc sempre que un enemic xoque amb el jugador. La comprovació de col·lisions és una tècnica bàsica de programació de jocs i, en general, requereix alguns càlculs matemàtics per determinar si dos sprites se superposaran.

Aquí és on resulta útil un *framework* com Pygame. Escriure un codi de detecció de col·lisions és tediós, però Pygame té MOLTS mètodes de detecció de col·lisions disponibles per utilitzar-los.

Per a aquest tutorial, utilitzeu un mètode anomenat *.spritecollideany()*, que detecta qualsevol col·lisió entre un *sprite* i els *sprites* d'un grup. Accepta un Sprite i un Grup com a paràmetres. Comprova les superposicions entre tots els *.rect* del grup i el *.rect* de l'sprite. Si és així, torna True, és a dir si detecta col·lisió. En cas contrari, torna False. Això s'ajusta perfectament a aquest joc, ja que hem de comprovar si un sol jugador xoca amb un grup d'enemics.

Vegem el codi:

```
# Draw all sprites
for entity in all_sprites:
    screen.blit(entity.surf, entity.rect)

# Check if any enemies have collided with the player
if pygame.sprite.spritecollideany(player, enemies):
    # If so, then remove the player and stop the loop
    player.kill()
    running = False
```

Es comprova si el jugador ha xocat amb algun enemic. Si és així, es crida a *player.kill()* per eliminar-lo de tots els grups als quals pertany. Com que els únics objectes que es representen es troben en *all_sprites*, el jugador ja no es renderitzarà. Una vegada que el jugador haja perdur, també haurem d'eixir del joc, de manera que configureu *running = False* per ixir del bucle del joc.

9 Velocitat del joc

En provar el joc, potser vos haureu adonat que els enemics es mouen massa ràpid. Si no, tranquils, ja que el joc s'executarà a diferents velocitats segons el hardware subjacent, el sistema operatiu, etc.

La raó d'això és que el bucle del joc processa els fotogrames tan ràpidament com el processador i l'entorn ho permeten. Com que tots els sprites es mouen una vegada per fotograma, es poden moure centenars de vegades cada segon. El nombre de fotogrames que es manegen cada segon s'anomena velocitat de fotogrames (*frame rate*), un terme molt utilitzat pels gamers. Aconseguir-ne un adequat és la diferència entre un joc jugable i un altre que no ho és.

Normalment, volem una freqüència de fotogrames el més alta possible, per poder apreciar el major nombre de detalls, però, per a aquest joc, cal reduir-lo un poc perquè el joc es puga jugar. Afortunadament, el mòdul de temps de Python conté un rellotge dissenyat exactament per a aquest propòsit.

L'ús del rellotge per establir una velocitat de fotogrames reproduïble requereix només dues línies de codi. El primer crea un rellotge nou abans que comence el bucle del joc. Després utilitzem la funció *.tick()* per informar a pygame que el programa ha arribat al final del fotograma.

```
# Setup the clock for a decent framerate
clock = pygame.time.Clock()

...

# Our main loop
while running:

    ...


    # Flip everything to the display
    pygame.display.flip()

    # Ensure program maintains a rate of 30 frames per second
    clock.tick(30)
```

L'argument passat a *.tick()* estableix la velocitat de fotogrames desitjada. Per fer-ho, es calcula el nombre de mil·lisegons que ha de tardar cada fotograma, en funció de la velocitat de fotogrames desitjada. A continuació, compara aquest nombre amb el nombre de mil·lisegons que han passat des de la darrera vegada que es va cridar *.tick()*. Si no ha passat prou temps, retarda el processament per assegurar-se que mai no supera la velocitat de fotogrames especificada.

Si es passa una freqüència de fotogrames més xicoteta, transcorrerà més temps entre fotogrames, mentre que una freqüència de fotogrames més gran proporcionarà un joc més suau (i més ràpid).

En aquest moment, tenim un joc totalment funcional i jugable.

10 Imatges dels *Sprites*

Molt bé, tens un joc, però siguem sincers … L'aspecte no és massa amigable. El jugador i els enemics són només blocs blancs sobre fons negre. Va ser una revolució quan va aparèixer el Pong el 1972, però ha quedat antiquat. Anem a substituir aquests rectangles blancs per unes imatges que faran el joc més amigable.

Anem a carregar una imatge d'un avió per al jugador i alguns míssils per als enemics, encara que podeu utilitzar altres per personalitzar el vostre joc.

10.1 Alteració dels constructors d'objectes

Abans d'utilitzar imatges per representar el jugador i els sprites enemics, hem de fer alguns canvis als seus constructors. El codi següent substitueix el codi utilitzat anteriorment.

```
class Player(pygame.sprite.Sprite):
    def __init__(self):
        super(Player, self).__init__()
        self.surf = pygame.image.load("resources/jet.png").convert()
        self.surf.set_colorkey((255, 255, 255), RLEACCEL)
        self.rect = self.surf.get_rect()
```

Amb *pygame.image.load()* carreguem una imatge del disc. Li passem com a argument el path a l'arxiu. Retorna una superfície i la crida *.convert()* optimitza la superfície, fent que les futures crides al mètode *.blit()* siguen més ràpides.

Amb *.set_colorkey()* indiquem quin color es farà transparent. En aquest cas, triem el blanc, perquè és el color de fons de la imatge. La constant RLEACCEL és un paràmetre opcional que ajuda a renderitzar pygame més ràpidament en pantalles no accelerades. Per poder utilitzar esta constant, necessitem importar-la a la declaració d'importació de pygame.locals del principi del codi.

No hem de canviar res més, la imatge continua sent una superfície, tret que ara hi ha una imatge pintada.

Farem el mateix al constructor d'enemic, esta vegada carregant la imatge *resources/missile.png*.

Figura 5: Imatges per al jugador i enemics

10.2 Afegir imatges de fons

Afegirem ara uns núvols com a imatges de fons, utilitzant els mateixos passos que per a Player i Enemy:

1. Creem la classe Cloud.
2. Afegim la imatge d'un núvol.
3. Creem un mètode *.update()* que mou el núvol cap a la part esquerra de la pantalla 5 píxels en cada fotograma.
4. Creem un gestor i un esdeveniment personalitzats per crear objectes de núvol nous a un interval de temps definit de 1000ms.
5. Afegim els objectes de núvols creats a un grup nou anomenat núvols.
6. Actualitzem i dibuixem els núvols al bucle principal del joc.

Tingueu en compte que cada núvol i enemic nous s'afegeixen a *all_sprites*, mentre que els núvols ho fan al grup *clouds* i els enemics a *enemies*. Perquè?

- El renderitzat (repintat) es fa amb all_sprites.
- Les actualitzacions de posició es fan a núvols i enemics.
- La detecció de col·lisions es fa entre el jugador i enemics, però no amb núvols, ja que un avió pot atrevessar núvols sense problemes.

Per últim canvieu el color de fons del negre a (135, 206, 250).

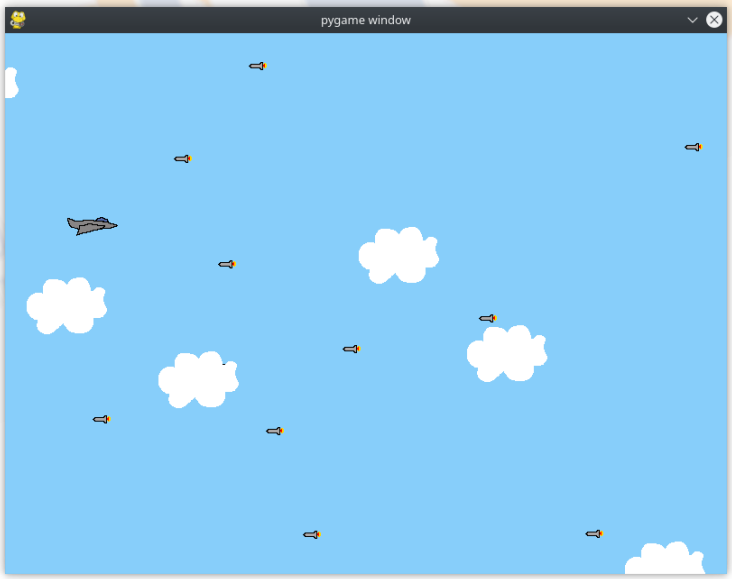

Figura 6: Aspecte final del joc

11 Efectes de so

Fins ara ens hem centrat en el joc i els aspectes visuals del mateix. Ara anem a veure com incorporar sons. pygame proporciona un mesclador per gestionar totes les activitats relacionades amb el so. Utilitzarà les classes i mètodes d'aquest mòdul per proporcionar música de fons i efectes de so per a diverses accions.

El nom de mesclador fa referència al fet que el mòdul barreja diversos sons en un tot cohesionat. Mitjançant el submòdul de música, podeu reproduir fitxers de so individuals en diversos formats. Tota la reproducció es produeix en segon pla, de manera que mentre es reprodueix un so, el mètode s'executa en paral·lel i torna el control immediatament al punt on s'ha fet la crida.

11.1 Inicialització

L'ús del mesclador comença amb la seua inicialització *pygame.mixer.init()*. Si no volem canviar els valors per defecte, no cal passar-li arguments. S'ha d'inicialitzar el mesclador abans que el pygame.

```
# Setup for sounds. Defaults are good.
pygame.mixer.init()

# Initialize pygame
pygame.init()
```

11.2 Música de fons

Una vegada inicialitzat el sistema, podeu configurar els vostres sons i música de fons com es mostra a continuació.

```
# Load and play background music
pygame.mixer.music.load("Apoxode_-_Electric_1.ogg")
pygame.mixer.music.play(loops=-1)

# Load all sound files
# Sound sources: Jon Fincher
move_up_sound = pygame.mixer.Sound("Rising_putter.ogg")
move_down_sound = pygame.mixer.Sound("Falling_putter.ogg")
collision_sound = pygame.mixer.Sound("Collision.ogg")
```

Carreguen un clip de so de fons i comencen a reproduir-lo. Podeu dir al clip de so que es reproduisca en bucle i que no acabe mai establint el paràmetre *loop = -1*.

11.3 Sons d'esdeveniments

Després carreguen tres sons que farem servir per a diversos efectes de so. Els dos primers són sons ascendents i descendents, que es reprodueixen quan el jugador es mou cap amunt o cap avall. L'últim és el so que s'utilitza sempre que hi ha una col·lisió. També podeu afegir altres sons, com ara un so per a la creació d'un enemic o un so final per a la finalització del joc.

Llavors, com s'utilitzen els efectes de so? Volem reproduir cada so quan es produeixi un esdeveniment determinat. Per exemple, quan l'avió es mou cap amunt, volem reproduir move_up_sound. Per tant, afegiu una crida a *.play()* sempre que gestioneu este l'esdeveniment.

Per a una col·lisió entre el jugador i un enemic, reproduïu el so *Collisions.ogg*.

Utilitzeu el mètode *stop()* quan vulgau parar un so que encara s'està reproduint i no voleu que es mescle amb un nou.

11.4 Fi del joc

Finalment, quan acabe el joc, tots els sons haurien de parar-se. Per fer-ho, afegiu les línies següents al final del programa després del bucle:

```
# All done! Stop and quit the mixer.
pygame.mixer.music.stop()
pygame.mixer.quit()
```

Tècnicament, aquestes últimes línies no són necessàries, ja que el programa finalitza i els recursos s'alliberen. No obstant això, si més endavant decidiu afegir una pantalla d'introducció o una pantalla d'eixida al vostre joc, és possible que hi haja més codi executat-se després que finalitze el joc.

12 Conclusió

Al llarg d'aquesta pràctica, hem après els fonaments de la programació de jocs amb pygame. Concretament:

- Implementar bucles d'esdeveniments
- Dibuixar elements a la pantalla
- Reproduir efectes de so i música
- Gestioneu l'entrada de l'usuari

Per fer-ho, hem utilitzat un subconjunt dels mòduls Pygame, inclosos els mòduls de visualització, mesclador i música, hora, imatge i esdeveniment. També hem utilitzat diverses classes, inclosos Rect,

Surface, Sound i Sprite. Però això només és una xicoteta part del que pot fer Pygame. Consulteu la documentació oficial de Pygame per obtenir una llista completa de mòduls i classes disponibles.

13 Tasques

13.1 Part obligatòria. Pràctica. (30%)

1. Seguix la pràctica pas a pas i construeix el codi del joc. Per fer-ho seguix el mateix ordre que es va explicant al llarg de la pràctica i ves provant els canvis que vas introduint.

13.2 Part d'ampliació. Projecte. (70%)

En esta segona part pots escollir entre dues opcions:

1. **Primera opció**. Amplia la funcionalitat del joc dels avions:
    1. La primera ampliació proposada és un **sistema de puntuacions** que serà visible a la pantalla del joc. Cada vegada que s'esquive un míssil i aquest sobrepasse la part esquerra de la pantalla, es sumaran 10 punts al marcador.
    2. La segona ampliació serà que el joc anirà canviant cada 20 segons entre **el dia i la nit**, és a dir, el fons canviarà del blau del dia al negre de la nit automàticament.
    3. La tercera ampliació serà un **sistema de nivells**. Cada vegada que el jugador supere un múltiple de 500 punts, el joc canviarà a un nivell superior, començant la partida en el nivell 1. A més, el nivell serà visible en la pantalla, al costat de la puntuació.
    4. La quarta ampliació serà que la **dificultat** del joc anirà creixent amb els nivells. Al principi del joc, es crearan enemics cada 500ms i la seua velocitat serà un número aleatori entre 1 i 10. En el segon nivell, la velocitat de creació seran 450 ms i les velocitats aniran entre 3 i 12. La idea és parametritzar els valors segons el nivell, per exemple, per a la velocitat de creació podria ser:

$$v_c = 100 + (450 - 50 * nivell)$$

$$v_e = random(2 * nivell, 10 + 3 * nivell)$$

> On v_c és la velocitat de creació d'enemics
> On v_e és la velocitat de desplaçament dels enemics

5. Fes que la puntuació, en cas de ser un nou rècord, es guarde en un document de text anomenat **punt_max.txt** en la mateixa carpeta on està el codi del joc. Per a saber si és un nou rècord, quan es carrega el joc, haurà de llegir el document i extraure la puntuació màxima.
6. Afegeix una **pantalla de benvinguda** on es done la benvinguda al joc. A més, ha de mostrar el rècord fins al moment i esperar fins que es polse la tecla p, moment en que es passarà a la pantalla de joc.
7. Fes una **pantalla final** on s'indique que la partida ha finalitzat, mostre la puntuació i el nivell al que s'ha arribat, i en cas de ser rècord, ho indique i felicite el jugador.
8. Proposeu una funcionalitat extra al joc per fer-lo més interessant. Qualsevol funcionalitat que resulte interessant serà valorada positivament.

2. **Segona opció**. Proposa la creació d'un joc del teu gust del tipus que hem desenvolupat en la pràctica. Pots pensar en jocs tipus el Pong, el Tetris, … Si agafes aquesta opció conta-li-la al professor abans de començar a desenvolupar perquè et done l'aprovació de la idea, ja que pot implicar una dificultat massa elevada o massa baixa. El joc hauria d'incloure tots els tractaments que hem anat desenvolupant durant la pràctica.

13.3 Qualificació

Per a qualificar aquesta primera unitat, es valorarà un 30% la part de pràctica i un 70% la part del projecte. Per a la part del projecte s'utilitzarà una rúbrica que es posarà a disposició dels alumnes abans de començar el desenvolupament. D'esta forma, l'alumne podrà saber anticipadament, els aspectes que es tindran en compte a l'hora de corregir. A més, es realitzarà una coavaluació. Per a fer açò, cada alumne exposarà en un període breu, al voltant de 5 minuts el seu projecte, i la resta d'alumnes l'avaluaran.