



# ZOMBIE KILLER

Silvestri Ignazio, Tornatola Mattia, Enis Gjini, Azzolari Emilio  
3Ai - A.S. 2024-2025

## Indice

1. Introduzione
2. Funzioni utilizzate
3. Librerie implementate
4. Organizzazione del codice
5. Test effettuati



# ZOMBIE KILLER

Zombie Killer è un gioco in stile Top-Down, nato dalla proposta di creare un gioco in Pygame con l'implementazione esterna di file che potessero creare classifiche e mappe personalizzate



# LIBRERIE UTILIZZATE

```
import pygame  
import os  
import math  
import time  
import random  
import datetime
```

## OS

abbiamo utilizzato la libreria OS all'interno delle funzioni riguardanti la visualizzazione e il caricamento delle mappe esterne.

è stata si fondamentale importanza per raccogliere tutte le mappe aggiunte dal giocatore in una lista, che viene successivamente visualizzata nella schermata delle proprie mappe

## MATH

abbiamo utilizzato la libreria math per calcolare gli angoli, che nel gioco sono stati molto utilizzati per far ruotare gli zombie e il personaggio nella direzione corretta

## TIME

abbiamo utilizzato la libreria time per andare a calcolare il tempo che passava nelle varie situazioni, ciò ci è servito quindi per regolare lo scorrere del tempo tra i diversi momenti di gioco

## TIME

abbiamo utilizzato la libreria random per far in modo tale che il gioco potesse essere più casuale e meno ripetitivo.

abbiamo randomizzato le posizioni di spawn dei nemici e dei powerUp.

## DATETIME

abbiamo utilizzato la libreria Datetime per calcolare con esattezza la data e l'ora che vengono inserite nella classifica del file esterno

# STRUTTURE DATI

## LISTE

Le liste sono state utilizzate per andare a memorizzare gli zombie attivi all'interno della mappa, quelli da rimuovere, i proiettili attivi e da rimuovere, e file presenti all'interno della cartella delle mappe del giocatore

```
ListaZombie = []
```

```
listaProiettili = []
```

## DIZIONARI

I dizionari sono stati utilizzati per avere un accesso facilitato alle immagini delle mappe che sono già caricate nel gioco, alle minimappe e ai tile che servono per trasformare i caratteri contenuti in un file in una mappa.

```
DizionarioMappe = {
    1: pygame.image.load("mappe/mappa1.png"),
    2: pygame.image.load("mappe/mappa2.png"),
    3: pygame.image.load("mappe/mappa3.png"),
    "m": pygame.image.load("mappe/aggiuntaMappa.png")
}
Minimappe = {
    1: pygame.transform.scale(DizionarioMappe[1], (MINIMAPPA_LARGHEZZA, MINIMAPPA_ALTEZZA)),
    2: pygame.transform.scale(DizionarioMappe[2], (MINIMAPPA_LARGHEZZA, MINIMAPPA_ALTEZZA)),
    3: pygame.transform.scale(DizionarioMappe[3], (MINIMAPPA_LARGHEZZA, MINIMAPPA_ALTEZZA)),
    "m": pygame.transform.scale(DizionarioMappe["m"], (MINIMAPPA_LARGHEZZA, MINIMAPPA_ALTEZZA))
}
```

# FUNZIONI UTILIZZATE

ZOMBIE KILLER, è programmato interamente in pygame, una libreria di python, per supportarci nella realizzazione del videogioco abbiamo avuto bisogno di consultare la documentazione ufficiale per capire come utilizzare nuove funzioni, come per esempio:

## set\_volume() e pygame.mixer.Channel(0)

La set\_volume() è una funzione che abbiamo utilizzato in modo tale che i suoni non si sovrapponessero a causa della stessa altezza di volume.

Questa funzione è stata supportata anche dalla Channel, che ci ha permesso di dividere il suono di sottofondo e i suoni del gioco

```
set_volume(0.09)
```

```
pygame.mixer.Channel(0)
```

## math.atan2(dy, dx) e math.degrees()

La prima funzione math.atan2(dy,dx) è servita per calcolare l'angolo in radianti tra i due punti e l'asseX che tramite la funzione math.degrees() verrà trasformato in gradi e ciò ci permetterà di ruotare il personaggio/gli zombie.

Il - viene utilizzato per invertire la direzione, che ci serve per ruotarlo nel verso giusto, poichè in pygame il sistema delle coordinate è diverso.

```
-math.degrees(math.atan2(dy, dx))
```

Nel programma viene poi utilizzata anche la rotate, che serve per ruotare l'immagine per l'angolo calcolato in precedenza.

```
pygame.transform.rotate(immagine, angolo)
```

## Collidrect() e Rect()

La collidrect nel programma ci è servita per gestire le collisioni tra PowerUp, il personaggio, gli zombie e il boss.

La collidrect ci chiede di passargli la rect, ovvero il rettangolo dell'immagine.

Questa viene calcolata tramite la Rect che tramite la lista che contiene gli elementi di ciascuno Zombie e Proiettile, che scorrendo la lista salva le coordinate X e Y e poi le dimensioni dell'immagine

```
rectP = pygame.Rect(p[0], p[1], 10, 10)
if rectZ.collidrect(rectP):
```

## Get\_ticks()

La funzione get\_ticks serve a misurare il tempo che è passato da un evento, nel programma è utilizzata diverse volte, in quanto diversi elementi dovevano restare solo per uno specifico lasso di tempo.

```
pygame.time.get_ticks()
```

## Get\_pos()

```
pygame.mouse.get_pos()
```

La funzione pose serve a prendere la posizione del mouse dello schermo che verrà poi utilizzata per il disegno del mirino nello schermo.

# CollisioniZombie

```
def CollisioniZombie(ListaZombie, listaProiettili, ZombieUccisi):
    daRimuovereZ = []
    daRimuovereP = []

    for z in ListaZombie:
        rectZ = pygame.Rect(z[0], z[1], 35, 43)
        for p in listaProiettili:
            rectP = pygame.Rect(p[0], p[1], 10, 10)
            if rectZ.colliderect(rectP):
                daRimuovereZ.append(z)
                Suonosangue.play()
                daRimuovereP.append(p)
                ListaSangue.append([z[0], z[1], pygame.time.get_ticks()])
                ZombieUccisi += 1

    for z in daRimuovereZ:
        if z in ListaZombie:
            ListaZombie.remove(z)

    for p in daRimuovereP:
        if p in listaProiettili:
            listaProiettili.remove(p)

    return ZombieUccisi
```

## Parametri di input:

ListaZombie: rappresenta una lista della posizione degli zombie formati da una coppia di coordinate x,y

ListaProiettili: rappresenta la lista che contiene tutti i proiettili sparati dal giocatore.

ZombieUccisi: rappresenta un contatore che tiene traccia di quanti zombie sono stati uccisi finora.

## Parametri di output: ZombieUccisi

**Corpo della funzione:** La funzione controlla le collisioni tra zombie e proiettili (attraverso la funzione `colliderect`), segna quelli che si scontrano, li rimuove dalle liste e aggiorna il numero di zombie uccisi.

# GestisciSangue

```
def GestisciSangue(ListaSangue):
    tempoAttuale = pygame.time.get_ticks()
    for sanguePos in ListaSangue:
        if tempoAttuale - sanguePos[2] <= 3000:
            schermo.blit(sangue, (sanguePos[0] - 40, sanguePos[1] - 43))
        else:
            ListaSangue.remove(sanguePos)
```

## Parametri di input:

ListaSangue: lista che contiene le posizioni e i tempi in cui è comparso il sangue.

## Parametri di output: Nessuno

**Corpo della funzione:** viene preso il tempo attuale (`tempoAttuale`) e per ogni macchia di sangue nella lista `ListaSangue` si controlla da quanto tempo è stata creata; se sono passati meno di 3 secondi viene disegnata sullo schermo in una posizione leggermente spostata, altrimenti viene rimossa dalla lista.

# Caricalmmagini

```
def CaricaImmagini():

    schermataTitolo = pygame.image.load("immagini/titolo.png")
    sfondoMappe = pygame.image.load("immagini/sfondoMappe.png")
    personaggio = pygame.image.load("immagini/personaggio.png")
    proiettile = pygame.transform.scale(pygame.image.load("immagini/weapon_gun.png"), (10, 10))
    zombie = pygame.image.load("immagini/zombie.png")
    sangue = pygame.image.load("immagini/sangue.png")
    sangue = pygame.transform.scale(sangue, (100, 100))
    cuore = pygame.transform.scale(pygame.image.load("immagini/cuore.png"), (70, 70))
    cuoreBonus = pygame.transform.scale(pygame.image.load("immagini/cuore.png"), (50, 50))
    fulmine = pygame.transform.scale(pygame.image.load("immagini/fulmine.png"), (50, 50))
    rifornimenti = pygame.transform.scale(pygame.image.load("immagini/risorse.png"), (40, 40))
    GameOver = pygame.transform.scale(pygame.image.load("immagini/GAMEOVER.png"), (700, 700))
    uno = pygame.transform.scale(pygame.image.load("immagini/uno.png"), (50, 50))
    due = pygame.transform.scale(pygame.image.load("immagini/due.png"), (50, 50))
    tre = pygame.transform.scale(pygame.image.load("immagini/tre.png"), (50, 50))
    quattro = pygame.transform.scale(pygame.image.load("immagini/quattro.png"), (50, 50))
    nuovaMappa = pygame.transform.scale(pygame.image.load("mappe/aggiuntaMappa.png"), (300, 250))
    boss = pygame.transform.scale(pygame.image.load("immagini/boss.png"), (100, 100))
    mirino = pygame.transform.scale(pygame.image.load("immagini/mirino.png"), (50, 50))
    SfondoMieMappe = pygame.image.load("immagini/sfondoMieMappe.png")
    bomba = pygame.transform.scale(pygame.image.load("immagini/bomba.png"), (50, 50))

    return schermataTitolo, sfondoMappe, personaggio, proiettile, zombie, sangue, cuore, cuoreBonus, rifornimenti
```

## Parametri di input:

Nessuno

La funzione viene chiamata senza parametri ed è pensata per essere eseguita una volta, all'avvio del gioco, per caricare tutte le immagini necessarie.

## Parametri di output:

Restituisce una tupla con tutte le immagini caricate e trasformate, tra cui:  
Immagini del giocatore, nemici, oggetti, interfaccia, effetti visivi e mappe.

## Struttura generale:

personaggio: sprite del giocatore

proiettile: immagine ridimensionata del proiettile (10x10)

mirino: puntatore del mouse per la mira (50x50)

zombie: sprite dello zombie base

boss: sprite del boss (100x100)

cuore: immagine grande di un cuore (70x70) per la vita

cuoreBonus: versione più piccola del cuore (50x50)

fulmine: potenziamento (50x50)

rifornimenti: munizioni (40x40)

bomba: oggetto bomba (50x50)

sangue: effetto sangue (100x100)

schermataTitolo: schermata iniziale

sfondoMappe: sfondo della schermata di selezione mappa

SfondoMieMappe: sfondo per mappe personalizzate

GameOver: immagine di game over (700x700)

nuovaMappa: bottone per aggiungere una nuova mappa (300x250)

uno, due, tre, quattro: numeri grafici (50x50) per scelte o livelli

# CaricaSuoni

```
def CaricaSuoni():

    Suonoarma = pygame.mixer.Sound('suoni/armaScarica.mp3')
    suonoRicarica = pygame.mixer.Sound('suoni/ricarica.mp3')
    Suonosangue = pygame.mixer.Sound('suoni/sangue.mp3')
    SuonoDanno = pygame.mixer.Sound('suoni/dolore.mp3')
    SuonoBomba = pygame.mixer.Sound('suoni/bomba.mp3')
    SottoInto = pygame.mixer.Sound('suoni/sottointo.mp3')

    SottoInto.set_volume(0.09)
    canaleSottoInto = pygame.mixer.Channel(0)
    canaleSottoInto.play(SottoInto, loops=-1)

    return Suonoarma, suonoRicarica, Suonosangue, SuonoDanno, SuonoBomba, SottoInto, canaleSottoInto
```

## Parametri di input:

Nessuno

La funzione viene chiamata senza parametri ed è pensata per essere eseguita una volta all'avvio del gioco per caricare e inizializzare i suoni.

## Parametri di output:

Restituisce una tupla contenente:

Tutti gli effetti sonori del gioco (Sound)

Il sottointo musicale (Sound)

Il canale mixer associato al sottointo (Channel)

## Struttura generale:

Suonoarma: rumore dello sparo (armaScarica.mp3)

suonoRicarica: suono della ricarica

Suonosangue: effetto sonoro quando si colpisce uno zombie

SuonoDanno: suono quando il giocatore subisce un danno

SuonoBomba: suono dell'esplosione della bomba

SottoInto: musica di sottointo del gioco

Volume impostato su 0.09 per non sovrastare gli altri suoni

Riproduzione automatica continua (loops=-1) sul canale 0

canaleSottoInto: canale dedicato alla musica di sottointo

Utilizzato per controllare volume separatamente

# RuotaVersoMouse

```
def RuotaVersoMouse(immagine, x, y, mouseX, mouseY):

    dx = mouseX - x
    dy = mouseY - y
    angolo = -math.degrees(math.atan2(dy, dx))
    immagineRuotata = pygame.transform.rotate(immagine, angolo)
    rett = immagineRuotata.get_rect(center=(x + immagine.get_width()//2, y + immagine.get_height()//2))
    return immagineRuotata, rett
```

## Parametri di input:

immagine: superficie (Surface) da ruotare.

x, y: coordinate dell'oggetto sullo schermo.

mouseX, mouseY: coordinate del puntatore del mouse.

## Parametri di output:

Restituisce una tupla contenente:

immagineRuotata: la superficie dell'immagine ruotata verso il mouse.

nuovoRect: il rettangolo (Rect) centrato sull'oggetto originale, aggiornato alla nuova immagine ruotata.

## Struttura generale:

Calcola la differenza tra la posizione del mouse e quella dell'immagine per determinare l'angolo di rotazione.

Usa math.atan2() per ottenere l'angolo in radianti, poi lo converte in gradi.

Ruota l'immagine con pygame.transform.rotate() secondo l'angolo calcolato.

Ottiene il nuovo Rect dell'immagine ruotata e lo centra sulla posizione originale (x, y).

# RotazioneZombie

```
def RotazioneZombie(immagine, zx, zy, giocatoreX, giocatoreY):

    dx = giocatoreX - zx
    dy = giocatoreY - zy
    angolo = -math.degrees(math.atan2(dy, dx))
    immagineRuotata = pygame.transform.rotate(immagine, angolo)
    rett = immagineRuotata.get_rect(center=(zx, zy))
    return immagineRuotata, rett
```

## Parametri di input:

immagine: immagine dello zombie da ruotare.

zx, zy: coordinate dello zombie.

giocatoreX, giocatoreY: coordinate del giocatore, verso cui lo zombie deve ruotare.

## Parametri di output:

Restituisce una tupla contenente:

immagineRuotata: la superficie dello zombie ruotata verso il giocatore.

rett: il Rect aggiornato, centrato sulle coordinate originali dello zombie.

## Struttura generale:

Calcola la differenza tra la posizione del giocatore e quella dello zombie (dx, dy).

Determina l'angolo di rotazione in gradi usando math.atan2() e math.degrees().

L'angolo viene invertito (segno meno) per adattarsi al sistema di coordinate di Pygame.

Ruota l'immagine dello zombie con pygame.transform.rotate() in base all'angolo calcolato.

Calcola il nuovo Rect con get\_rect(center=(zx, zy)) per mantenere il centro invariato.

# ScegliMappa

```
def ScegliMappa(event,nuovoPercorso):  
  
    if event.type == pygame.KEYDOWN:  
        if event.key == pygame.K_1:  
            return DizionarioMappe[1],1  
        elif event.key == pygame.K_2:  
            return DizionarioMappe[2],2  
        elif event.key == pygame.K_3:  
            return DizionarioMappe[3],3  
        elif event.key == pygame.K_4:  
            return nuovoPercorso ,4,  
    return None, None
```

## Parametri di input:

event: evento ricevuto da Pygame, in genere passato da pygame.event.get().

nuovoPercorso: stringa contenente il percorso di una mappa personalizzata (opzionale, usata se il tasto 4 viene premuto).

## Parametri di output:

Restituisce una tupla:

La mappa selezionata (presa da DizionarioMappe o da nuovoPercorso).

Il numero della mappa selezionata (1, 2, 3 o 4).

Se nessun tasto valido viene premuto, restituisce (None, None).

## Struttura generale:

Selezione mappa tramite tastiera:

Controlla se l'evento è un tasto premuto (KEYDOWN).

Se viene premuto uno dei tasti 1, 2, 3, o 4, restituisce la mappa corrispondente:

I tasti 1–3 selezionano mappe predefinite dal DizionarioMappe.

Il tasto 4 seleziona una mappa personalizzata indicata da nuovoPercorso.

# GestisciSpazio

```
def GestisciSpazio(event, SpazioPremuto):
    if not SpazioPremuto and event.type == pygame.KEYDOWN and event.key == pygame.K_SPACE:
        return True
    return SpazioPremuto
```

## Parametri di input:

event: un evento della coda di Pygame, tipicamente ottenuto con pygame.event.get().

SpazioPremuto: valore booleano che indica se il tasto SPAZIO è già stato premuto.

## Parametri di output:

Restituisce True se il tasto SPAZIO viene premuto per la prima volta.

Altrimenti, restituisce il valore originale di SpazioPremuto (non cambia nulla se il tasto non viene premuto o è già stato premuto).

## Struttura generale:

Controlla se:

SpazioPremuto è False

e l'evento è KEYDOWN

e il tasto premuto è K\_SPACE.

Se tutte queste condizioni sono vere, restituisce True, indicando che il tasto è stato appena premuto.

In tutti gli altri casi, mantiene lo stato attuale (SpazioPremuto).

# GestisciMovimento

```
def GestisciMovimento(tasti, x, y, velocita):  
  
    if tasti[pygame.K_a] and x > 0:  
        x -= velocita  
    if tasti[pygame.K_d] and x < LARGHEZZASCHERMO - 64:  
        x += velocita  
    if tasti[pygame.K_w] and y > 0:  
        y -= velocita  
    if tasti[pygame.K_s] and y < ALTEZZASCHERMO - 64:  
        y += velocita  
    return x, y
```

## Parametri di input:

tasti: dizionario booleano ottenuto da pygame.key.get\_pressed(), che indica quali tasti sono attualmente premuti.

x: coordinata X attuale del giocatore.

y: coordinata Y attuale del giocatore.

velocita: valore numerico che definisce la velocità di movimento del giocatore.

## Parametri di output:

Restituisce una tupla (x, y) aggiornata, ovvero le nuove coordinate del giocatore dopo il movimento.

## Struttura generale:

Controllo direzionale del movimento:

Controlla i tasti WASD per il movimento:

A → sinistra

D → destra

W → su

S → giù

Impedisce al giocatore di uscire dai limiti dello schermo, verificando che:

x > 0 per il bordo sinistro

x < LARGHEZZASCHERMO - 64 per il bordo destro

y > 0 per il bordo superiore

y < ALTEZZASCHERMO - 64 per il bordo inferiore

# SparaProiettile

```
def SparaProiettile(x, y, mouseX, mouseY, listaProiettili):  
  
    angoloRad = math.atan2(mouseY - (y+32), mouseX - (x+32))  
    dx = math.cos(angoloRad)  
    dy = math.sin(angoloRad)  
    distanza = 32  
    startX = x + 32 + dx * distanza  
    startY = y + 32 + dy * distanza  
    listaProiettili.append([startX, startY, dx, dy, -math.degrees(angoloRad)])
```

## Parametri di input:

x, y: coordinate del giocatore al momento dello sparo.

mouseX, mouseY: coordinate del puntatore del mouse, verso cui si spara.

listaProiettili: lista che memorizza tutti i proiettili attivi sullo schermo.

## Output:

Nessun valore di ritorno diretto.

La funzione modifica la lista listaProiettili aggiungendo un nuovo proiettile con:  
posizione iniziale calcolata in direzione del mouse,  
direzione (dx, dy) del movimento,  
angolo di rotazione in gradi per l'eventuale disegno dell'immagine.

## Struttura generale:

Calcolo direzione di sparo:

Usa math.atan2() per calcolare l'angolo tra il centro del giocatore (x+32, y+32) e il puntatore del mouse.

Calcola i versori dx, dy della direzione di sparo (coseno e seno dell'angolo).

Posizione di partenza del proiettile:

Il proiettile parte 32 pixel avanti rispetto al centro del giocatore nella direzione del mouse, per evitare che sembri uscire dal corpo.

Aggiunta alla lista:

Inserisce un nuovo elemento nella lista listaProiettili:

[startX, startY, dx, dy, angolo]

Dove:

startX, startY = posizione iniziale del proiettile

dx, dy = direzione del movimento

angolo = usato per ruotare graficamente il proiettile se necessario

# GestisciProiettili

```
def GestisciProiettili(listaProiettili, velocitaProiettile):  
  
    restanti = []  
    for p in listaProiettili:  
        p[0] += p[2] * velocitaProiettile  
        p[1] += p[3] * velocitaProiettile  
        if 0 <= p[0] and p[0] <= LARGHEZZASCHERMO and 0 <= p[1] and p[1] <= ALTEZZASCHERMO:  
            restanti.append(p)  
    listaProiettili[:] = restanti
```

## Parametri di input:

listaProiettili: una lista contenente tutti i proiettili attivi nel gioco. Ogni proiettile è rappresentato da una lista con [x, y, dx, dy, angolo].

velocitaProiettile: la velocità alla quale ogni proiettile deve muoversi ad ogni fotogramma.

## Parametri di output:

Nessun ritorno diretto, ma la lista listaProiettili viene aggiornata internamente con solo i proiettili ancora all'interno dello schermo.

## Struttura generale:

Aggiornamento del movimento:

Ogni proiettile si sposta in base alla sua direzione (dx, dy) moltiplicata per la velocitaProiettile.

Controllo confini schermo:

I proiettili che escono dallo schermo non vengono conservati.

Vengono tenuti solo quelli che restano entro i limiti definiti da LARGHEZZASCHERMO e ALTEZZASCHERMO.

Filtraggio e aggiornamento lista:

I proiettili validi vengono inseriti nella lista restanti.

# GestisciScritte

```
def GestisciScritte(schermo, caricatore, ricarica, ultimaRicarica, scorte, ZombieUccisi, font):

    testoColpi = font.render(F"Colpi {caricatore}", True, (255,255,255))
    schermo.blit(testoColpi, (10,10))
    if caricatore == 0:
        testoColpi = font.render(F"Colpi {caricatore}", True, (255,0,0))
        schermo.blit(testoColpi, (10,10))
    if ricarica:
        tempo = max(0, 2 - int(time.time() - ultimaRicarica))
        testoRicarica = font.render(F"Ricarica {tempo}s", True, (255,0,0))
        schermo.blit(testoRicarica, (130,10))
        schermo.blit(testoRicarica, (130,10))
    TestoScorte = font.render(F"Scorte {scorte}", True, (255,255,255))
    TestoKill = font.render(F"Punteggio {ZombieUccisi}", True, (255,255,255))
    schermo.blit(TestoKill, (600, 10))
    schermo.blit(TestoScorte, (10, 50))
```

Parametri di input:

schermo: la superficie di gioco (schermo) su cui disegnare il testo.

caricatore: il numero di colpi rimasti nel caricatore.

ricarica: un valore booleano che indica se la ricarica è in corso.

ultimaRicarica: il timestamp dell'ultima ricarica effettuata.

scorte: il numero totale di munizioni (scorte).

ZombieUccisi: il numero di zombie uccisi dal giocatore.

font: il font utilizzato per il rendering del testo.

Parametri di output:

Nessun ritorno diretto, ma disegna il testo sullo schermo, inclusi i dettagli riguardanti il caricatore, le scorte, il punteggio e la ricarica.

Struttura generale:

Testo del caricatore:

Viene disegnato il numero di colpi rimasti nel caricatore. Se il caricatore è vuoto (caricatore == 0), il testo viene visualizzato in rosso per evidenziare che è necessario ricaricare.

Testo della ricarica:

Se la ricarica è in corso (ricarica è True), viene calcolato quanto tempo manca al termine della ricarica (tempo rimanente). Viene visualizzato un testo con il countdown, che scompare quando la ricarica è completata.

Testo delle scorte:

Viene disegnato il numero di scorte di munizioni totali (non nel caricatore, ma nelle scorte generali).

Testo del punteggio:

Il numero di zombie uccisi (ZombieUccisi) viene visualizzato come punteggio.

# SpawnZombie

```
def SpawnZombie(partenzaSu, fineSu, partenzaGiu, fineGiu, partenzaSx, fineSx, partenzaDx, fineDx):
    lato = random.choice(["su", "giu", "sinistra", "destra"])
    if lato == "su":
        return [random.randint(partenzaSu, fineSu), -50]
    if lato == "giu":
        return [random.randint(partenzaGiu, fineGiu), ALTEZZASCHERMO + 50]
    if lato == "sinistra":
        return [-50, random.randint(partenzaSx, fineSx)]
    if lato == "destra":
        return [LARGHEZZASCHERMO + 50, random.randint(partenzaDx, fineDX)]
```

## Parametri di input:

partenzaSu: il limite inferiore della posizione orizzontale per lo spawn sulla parte superiore dello schermo.

fineSu: il limite superiore della posizione orizzontale per lo spawn sulla parte superiore dello schermo.

partenzaGiu: il limite inferiore della posizione orizzontale per lo spawn sulla parte inferiore dello schermo.

fineGiu: il limite superiore della posizione orizzontale per lo spawn sulla parte inferiore dello schermo.

partenzaSx: il limite inferiore della posizione verticale per lo spawn sul lato sinistro dello schermo.

fineSx: il limite superiore della posizione verticale per lo spawn sul lato sinistro dello schermo.

partenzaDx: il limite inferiore della posizione verticale per lo spawn sul lato destro dello schermo.

fineDX: il limite superiore della posizione verticale per lo spawn sul lato destro dello schermo.

## Parametri di output:

Restituisce una lista contenente due valori:

La coordinata X e la coordinata Y dove lo zombie spawnerà.

La posizione è determinata in modo casuale in base alla direzione scelta.

## Struttura generale:

La funzione seleziona una direzione casuale (su, giù, sinistra, destra) attraverso la variabile lato usando random.choice().

A seconda della direzione selezionata:

"su": lo zombie verrà spawnato sopra lo schermo, con una posizione orizzontale casuale tra partenzaSu e fineSu.

"giu": lo zombie verrà spawnato sotto lo schermo, con una posizione orizzontale casuale tra partenzaGiu e fineGiu.

"sinistra": lo zombie apparirà sulla sinistra dello schermo, con una posizione verticale casuale tra partenzaSx e fineSx.

"destra": lo zombie apparirà sulla destra dello schermo, con una posizione verticale casuale tra partenzaDx e fineDX.

La posizione Y è calcolata in modo da spawnare lo zombie al di fuori dello schermo (50 pixel sopra o sotto) e la posizione X è determinata all'interno dei limiti definiti per ogni lato.

# GestisciZombie

```
def GestisciZombie(ListaZombie, giocatoreX, giocatoreY, velocitaZombie, zombie):
    centroX = giocatoreX + 32
    centroY = giocatoreY + 32
    distanzaMinima = 40
    distanzaMinima_sq = distanzaMinima * distanzaMinima

    for i, z in enumerate(ListaZombie):
        dxGiocatore = centroX - z[0]
        dyGiocatore = centroY - z[1]
        distanza_giocatore_quadrato = dxGiocatore**2 + dyGiocatore**2

        if distanza_giocatore_quadrato != 0:
            inv_distanza = 1 / (distanza_giocatore_quadrato**0.5)
            movimentoX = dxGiocatore * inv_distanza * velocitaZombie
            movimentoY = dyGiocatore * inv_distanza * velocitaZombie
        else:
            movimentoX, movimentoY = 0, 0

        for j, altro in enumerate(ListaZombie):
            if i != j:
                dx = z[0] - altro[0]
                dy = z[1] - altro[1]
                dist_sq = dx**2 + dy**2

                if 0 < dist_sq and dist_sq < distanzaMinima_sq:
                    inv_dist = 1 / (dist_sq**0.5)
                    forza_repulsiva = (distanzaMinima_sq - dist_sq) / distanzaMinima_sq
                    movimentoX += dx * inv_dist * forza_repulsiva * velocitaZombie
                    movimentoY += dy * inv_dist * forza_repulsiva * velocitaZombie

        z[0] += movimentoX
        z[1] += movimentoY
```

## Parametri di input:

ListaZombie: rappresenta una lista della posizione degli zombie formati da una coppia di coordinate x,y  
giocatoreX, giocatoreY: coordinate del giocatore

velocitaZombie: indica quanto velocemente si muovono gli zombie

zombie: rappresenta l' immagine base dello zombie

## Parametri di output:

Nessuno

**Corpo della funzione:** per ogni zombie si calcola la direzione verso il centro del giocatore e si determina un movimento in quella direzione. Poi si verifica la distanza da tutti gli altri zombie e se uno zombie è troppo vicino, si applica una forza di respulsione per allontanarli. Dopo aver sommato il movimento verso il giocatore e la respulsione dagli altri zombie, si aggiorna la posizione dello zombie. Infine, si ruota l'immagine dello zombie per far guardare il giocatore

# AumentoSpawnZombie

```
def AumentoSpawnZombie(tempoUltimoSpawn, ListaZombie, tempoUltimaOndata, durataOndata, partenzaSu, fineSu, partenzaGiù, fineGiù, partenzaSx, fineSx, partenzaDx, fineDx):
    incremento = random.randint(3,10)
    incremento = random.randint(3,10)
    if pygame.time.get_ticks() - tempoUltimaOndata >= 15000:
        for _ in range(10+incremento):
            ListaZombie.append(SpawnZombie(partenzaSu, fineSu, partenzaGiù, fineGiù, partenzaSx, fineSx, partenzaDx, fineDx))

    tempoUltimaOndata = pygame.time.get_ticks()

    if pygame.time.get_ticks() - tempoUltimoSpawn >= durataOndata:
        ListaZombie.append(SpawnZombie(partenzaSu, fineSu, partenzaGiù, fineGiù, partenzaSx, fineSx, partenzaDx, fineDx))
        tempoUltimoSpawn = pygame.time.get_ticks()

return tempoUltimoSpawn, ListaZombie, tempoUltimaOndata
```

## Parametri di input:

tempoUltimoSpawn: indica quando è stato creato l'ultimo zombie.

ListaZombie: indica la lista che contiene tutti gli zombie.

tempoUltimaOndata: indica quando è stata l'ultima ondata di zombie.

durataOndata: indica quanto tempo deve passare tra un'onda di zombie e l'altra.

partenza e fine per su, giù, sinistra, destra: indicano i limiti che definiscono dove possono apparire gli zombie.

**Parametri di output:** tempoUltimoSpawn, ListaZombie, tempoUltimaOndata

**Corpo della funzione:** la funzione AumentoSpawnZombie crea zombie a intervalli regolari. Ogni 15 secondi viene generato un gruppo di zombie, con un numero casuale tra 13 e 20. Inoltre, periodicamente, viene aggiunto uno zombie alla lista, in base alla durata definita per ogni ondata. Gli zombie appaiono in posizioni casuali, determinati dai limiti di spawn dati.

# GestisciVita

```
def GestisciVita(ListaZombie, giocatoreX, giocatoreY, cuori, tempoUltimoDanno, contatoreDanno, n):
    rectGiocatore = pygame.Rect(giocatoreX, giocatoreY, 49, 43)
    tempoAttuale = pygame.time.get_ticks()
    dannoSubito = False

    for z in ListaZombie:
        rectZombie = pygame.Rect(z[0], z[1], 35, 43)
        if rectGiocatore.colliderect(rectZombie):
            dannoSubito = True

    if dannoSubito and tempoAttuale - tempoUltimoDanno >= 2000:
        if cuori > 0:
            contatoreDanno += n
            SuonoDanno.play()
            if contatoreDanno >= 2:
                cuori -= 1
                contatoreDanno = 0

        tempoUltimoDanno = tempoAttuale

    for _ in range(3):
        if cuori>0:
            schermo.blit(cuore, (1250, 10))
        if cuori>1:
            schermo.blit(cuore, (1310, 10))
        if cuori>2:
            schermo.blit(cuore, (1370, 10))

    return cuori, tempoUltimoDanno, contatoreDanno
```

## Parametri di input:

ListaZombie: lista delle posizioni degli zombie (ogni zombie è una coppia [x, y]).

giocatoreX, giocatoreY: coordinate del giocatore.

cuori: numero di cuori (vite) del giocatore, da 0 a 3.

tempoUltimoDanno: il tempo (in millisecondi) in cui il giocatore ha subito l'ultimo danno.

contatoreDanno: un contatore che accumula il danno; serve per ridurre la vita ogni 2 colpi.

n: valore da sommare al contatore ogni volta che si subisce un danno

**Parametri di output:** cuori, tempoUltimoDanno, contatoreDanno

**Corpo della funzione:** La funzione crea un rettangolo per il giocatore e controlla se collide con uno zombie. Se c'è contatto e sono passati almeno 2 secondi dall'ultimo danno, il contatore aumenta; se arriva a 2 o più, il giocatore perde un cuore e il contatore si azzera. Viene aggiornato il tempo dell'ultimo danno. In base ai cuori rimasti, ne vengono disegnati fino a tre sullo schermo. La funzione restituisce i cuori aggiornati, il nuovo tempo dell'ultimo danno e il contatore del danno.

# CuoriCasuali

```
def CuoriCasuali(tempoUltimoCuore, CuorePos, CuoreVisible, giocatoreX, giocatoreY, cuori, maxCuori):
    if not CuoreVisible and pygame.time.get_ticks() - tempoUltimoCuore >= 20000 and cuori<=2:
        if random.random() < 0.05:
            CuorePos = (random.randint(0, LARGHEZZASCHERMO - 50), random.randint(0, ALTEZZASCHERMO - 50))
            CuoreVisible = True
            tempoUltimoCuore = pygame.time.get_ticks()

    if CuoreVisible:
        rectGiocatore = pygame.Rect(giocatoreX, giocatoreY, 49, 43)
        rectCuore = pygame.Rect(CuorePos[0], CuorePos[1], 50, 50)
        if rectGiocatore.colliderect(rectCuore):
            if cuori < maxCuori:
                cuori += 1
                CuoreVisible = False

    if CuoreVisible:
        schermo.blit(cuoreBonus, CuorePos)

    return tempoUltimoCuore, CuorePos, CuoreVisible, cuori
```

## Parametri di input:

tempoUltimoCuore: il tempo (in millisecondi) dell'ultima comparsa di un cuore.

CuorePos: una tupla con la posizione (x, y) del cuore.

CuoreVisible: booleano che indica se un cuore è attualmente visibile sullo schermo.

giocatoreX, giocatoreY: posizione X eY del giocatore.

cuori: numero attuale di cuori raccolti dal giocatore.

maxCuori: numero massimo di cuori che il giocatore può avere.

**Parametri di output:** tempoUltimoCuore, CuorePos, CuoreVisible, cuori.

**Corpo della funzione:** Se non c'è un cuore visibile, sono passati almeno 20 secondi dall'ultimo e il giocatore ha al massimo 2 cuori, con una probabilità del 5% viene generato un nuovo cuore in una posizione casuale e reso visibile. Se il cuore è visibile e il giocatore lo tocca, il numero di cuori aumenta (se sotto il massimo) e il cuore scompare. Se resta visibile, viene disegnato sullo schermo.

# FulminiCasuali

```
def FulminiCasuali(tempoUltimoFulmine, FulminePos, FulmineVisible, giocatoreX, giocatoreY, velocita, velocitaProiettile, tempoProiettili, Fulmineattivo, raccolto):  
    tempoAttuale = pygame.time.get_ticks()  
  
    if not FulmineVisible and tempoAttuale - tempoUltimoFulmine >= 20000:  
        if random.randint() < 0.02:  
            FulminePos = (random.randint(0, LARGHEZZASCHERMO - 70), random.randint(0, ALTEZZASCHERMO - 70))  
            FulmineVisible = True  
            tempoUltimoFulmine = tempoAttuale  
  
    if FulmineVisible:  
        rectGiocatore = pygame.Rect(giocatoreX, giocatoreY, 40, 40)  
        rectFulmine = pygame.Rect(FulminePos[0], FulminePos[1], 70, 70)  
  
        if rectGiocatore.colliderect(rectFulmine):  
            FulmineVisible = False  
            Fulmineattivo = tempoAttuale  
            raccolto = True  
  
    if raccolto:  
        if tempoAttuale - Fulmineattivo <= 5000:  
            velocita = 10  
            velocitaProiettile = 15  
            tempoProiettili = 0.2  
        else:  
            raccolto = False  
            velocita = 5  
            velocitaProiettile = 10  
            tempoProiettili = 0.5  
            Fulmineattivo = 0  
  
    if Fulminevisible:  
        schermo.blit(fulmine, FulminePos)  
    return tempoUltimoFulmine, FulminePos, FulmineVisible, velocita, velocitaProiettile, tempoProiettili, Fulmineattivo, raccolto
```

## Parametri di input:

tempoUltimoSpawn: indica il momento (in millisecondi) in cui è stato creato l'ultimo zombie.

ListaZombie: è la lista che contiene tutti gli zombie attualmente presenti nel gioco.

tempoUltimaOndata: indica il momento in cui è stata generata l'ultima ondata di zombie.

durataOndata: il tempo (in millisecondi) che deve passare tra un'ondata di zombie e l'altra.

partenzaSu, fineSu: definiscono i limiti di spawn verticale per la zona "sopra" (zona alta dello schermo).

partenzaGiu, fineGiu: limiti di spawn per la zona "sotto".

partenzaSinistra, fineSinistra: limiti di spawn orizzontali per la zona "sinistra".

partenzaDestra, fineDestra: limiti di spawn orizzontali per la zona "destra".

## Parametri di output:

tempoUltimoSpawn: aggiornato con il tempo dell'ultimo zombie generato.

ListaZombie: aggiornata con i nuovi zombie aggiunti.

tempoUltimaOndata: aggiornato con il tempo dell'ultima ondata creata.

**Corpo della funzione :** La funzione AumentoSpawnZombie regola la comparsa progressiva degli zombie durante la partita. Ogni 15 secondi (o secondo il tempo specificato da durataOndata), viene generata un'ondata principale composta da un numero casuale di zombie, variabile tra 13 e 20. Oltre a queste ondate, la funzione può anche aggiungere singoli zombie a intervalli regolari, in modo da mantenere una certa pressione costante sul giocatore. Gli zombie vengono posizionati in modo casuale, scegliendo tra quattro direzioni (alto, basso, sinistra, destra), e ogni direzione ha dei limiti precisi entro cui gli zombie possono comparire. Questo sistema assicura una distribuzione più dinamica e realistica sul campo di gioco, rendendo l'esperienza più imprevedibile e coinvolgente.

# ColpiCasuali

```
def ColpiCasuali(ColpiVisibili, tempoUltimoRifornimento, RifPos, giocatoreX, giocatoreY, scorte, Presi, tempoColpiRaccolti):
    tempoAttuale = pygame.time.get_ticks()

    if not ColpiVisibili and tempoAttuale - tempoUltimoRifornimento >= 25000:
        if random.random() < 0.04:
            RifPos = (random.randint(0, LARGHEZZASCHERMO - 70), random.randint(0, ALTEZZASCHERMO - 70))
            ColpiVisibili = True
            tempoUltimoRifornimento = tempoAttuale

    if ColpiVisibili:
        rectGiocatore = pygame.Rect(giocatoreX, giocatoreY, 49, 43)
        rectRifornimento = pygame.Rect(RifPos[0], RifPos[1], 70, 70)

        if rectGiocatore.colliderect(rectRifornimento):
            ColpiVisibili = False
            Presi = True
            tempoColpiRaccolti = tempoAttuale
            scorte += random.randint(10, 20)

    if ColpiVisibili:
        schermo.blit(rifornimenti, (RifPos))

    return tempoUltimoRifornimento, scorte, ColpiVisibili, RifPos, Presi, tempoColpiRaccolti
```

## Parametri di input:

ColpiVisibili: indica se un rifornimento di colpi è attualmente visibile sullo schermo.

tempoUltimoRifornimento: rappresenta il tempo (in millisecondi) in cui è stato generato l'ultimo rifornimento.

RifPos: la posizione (x, y) del rifornimento sullo schermo.

giocatoreX, giocatoreY: indicano la posizione attuale del giocatore.

scorte: quantità attuale di colpi posseduti dal giocatore.

Presi: indica se il rifornimento è stato raccolto.

tempoColpiRaccolti: momento in cui il giocatore ha raccolto il rifornimento.

## Parametri di output:

tempoUltimoRifornimento, scorte, ColpiVisibili, RifPos, Presi,

tempoColpiRaccolti

**Corpo della funzione:** La funzione ColpiCasuali gestisce la comparsa e la raccolta dei rifornimenti di munizioni durante il gioco. Ogni 25 secondi, c'è una piccola probabilità (4%) che appaia un rifornimento in una posizione casuale dello schermo. Se il giocatore entra in contatto con questo rifornimento, l'oggetto scompare e il giocatore riceve un bonus casuale di colpi, compreso tra 10 e 20. A quel punto vengono aggiornati sia il tempo dell'ultima raccolta sia un flag che indica che il rifornimento è stato preso. Se il rifornimento è attivo e non ancora raccolto, viene disegnato a schermo nella sua posizione.

# Statoiniziale

## Parametri di input:

Nessuno.

La funzione viene chiamata senza parametri ed è pensata per essere eseguita una volta, all'avvio o al reset del gioco.

## Parametri di output:

Stato del giocatore: posizione, velocità, salute, munizioni, ricarica.

Stato degli oggetti speciali: cuori, fulmini, rifornimenti, bombe.

Stato del combattimento: lista zombie, lista boss, proiettili, sangue.

Temporizzazioni: intervalli per spawn, potenziamenti, danni, boss, bombe.

Interfaccia: font nome giocatore messaggi mappe

**Stati booleani:** visibilità oggetti, condizioni di raccolta, avvio gioco, salvataggio, ecc.

### **Struttura generale:**

**Giocatore:**

Posizione iniziale (300, 300), velocità standard (velocita = 5)

Munizioni: scorte = 100, caricatore = 20, maxCaricatore = 20

Timer sparco e ricarica (ultimoColpo, ultimaRicarica)

Salute: cuori = 3, contatoreDanno, tempo, ultimoDanno

**Salute. cuori - 3,**

**Oggetti speciali:** Guerigianezza, visibilità, tempesta dell'ultima sponda

Cuore, posizione, visibilità, tempo dell'ultimo spawn  
Educazione, incisività, tenacia, tenore, attitudine

Fulmine: potenziamento temporaneo, tempo è stato raccolto

Riformimento colpi: posizione, visibilità, qualità

Bomba:

**Nemici:**

Zombie: lista, frequenza spawn, velocità, time

Boss: lista, vita, v

## **Combattimento:**

## Proiettili: lista e velocità

## Sangue: lista, visibilità, timer

## Interfaccia e stato di

## Font personalizzato

Stato partita (gioco, spazioPremuto, nom

Salvataggio (Salvato, MioFile, MieMappe)

# Inserisci Nome

```
def InserisciNome(eventi, schermo, font, nomeGiocatore, nomeInserito, Salvato):

    RectCasella = pygame.Rect(500, 90, 400, 60)
    for evento in eventi:
        if evento.type == pygame.KEYDOWN:
            if evento.key == pygame.K_RETURN and nomeGiocatore.strip() != "":
                nomeInserito = True
                Salvato = True
            elif evento.key == pygame.K_BACKSPACE:
                nomeGiocatore = nomeGiocatore[:-1]
            else:
                if len(nomeGiocatore) < 15 and evento.unicode.isprintable():
                    nomeGiocatore += evento.unicode

    Nome = font.render("COME TI CHIAMI?", True, (255,255,255))
    schermo.blit(Nome, (560, 40))

    # va a disegnare a schermo il rettangolo
    pygame.draw.rect(schermo, (184, 20, 20), RectCasella, 3, 5)

    # va a scrivere il testo dentro la casella
    Testo = font.render(nomeGiocatore, True, (255,255,255))
    schermo.blit(Testo, (505, 100))

    return nomeGiocatore.strip(), nomeInserito, Salvato
```

## Parametri di input:

eventi: indica la lista degli eventi rilevati da Pygame

schermo: indica la superficie su cui vengono disegnati elementi grafici (la finestra di gioco)

font: indica il font usato per scrivere testo sullo schermo.

nomeGiocatore: indica la stringa che contiene il nome attualmente digitato dal giocatore.

nomeInserito: indica il booleano che indica se il nome è stato confermato (con Invio).

Salvato: indica il booleano che indica se il nome è stato salvato (insieme a nomeInserito).

## Parametri di output: nomeGiocatore.strip(), nomeInserito, Salvato

**Corpo della funzione:** La funzione InserisciNome gestisce l'inserimento del nome del giocatore tramite tastiera. Se il giocatore preme Invio e ha scritto qualcosa, il nome viene considerato inserito e salvato. Se preme Backspace, l'ultimo carattere viene rimosso. Se preme un altro tasto stampabile e il nome è lungo meno di 15 caratteri, il carattere viene aggiunto. Sullo schermo viene mostrata la scritta "COME TI CHIAMI?", viene disegnata una casella rossa per il testo, e dentro di essa viene visualizzato il nome digitato. Alla fine la funzione restituisce il nome ripulito da spazi, e i due stati nomeInserito e Salvato.

# CreaMappa

```
def CreaMappa(path):
    mappa = []
    try:
        f = open(path, "r", encoding="utf-8")
        for riga in f:
            riga = riga.strip().upper()
            mappa.append(riga)
        f.close()

        # Controllo che ci siano 25 righe
        if len(mappa) != 25:
            print(f"Errore: la mappa '{path}' ha {len(mappa)} righe, ma ne servono 25.")
            return None

        # Controllo che ogni riga abbia 45 caratteri
        for i, riga in enumerate(mappa):
            if len(riga) != 45:
                print(f"Errore: nella mappa '{path}' la riga {i+1} ha {len(riga)} colonne, ma ne servono 45.")
                return None

    return mappa

except Exception:
    print(F"Errore caricando la mappa {path}")
    return None
```

## Parametri di input:

path: indica il percorso del file da leggere.

## Parametri di output: Mappa

**Corpo della funzione:** La funzione prende come input il percorso del file (path) contenente la mappa. Cerca di aprire il file in modalità lettura, legge riga per riga, rimuove eventuali spazi all'inizio e alla fine, converte ogni riga in maiuscolo e la aggiunge alla lista mappa. Dopo la lettura, controlla che ci siano esattamente 25 righe e che ogni riga contenga esattamente 45 caratteri. Se i controlli falliscono, stampa un messaggio d'errore e restituisce None. Se tutto è corretto, restituisce la lista mappa.

# DisegnaMappa

```
def DisegnaMappa(mappa, mappaTile, messaggioMappaNonCorretta):
    for y, riga in enumerate(mappa):
        for x, tile in enumerate(riga):
            if tile in mappaTile:
                # Disegna il tile corrispondente alla lettera
                schermo.blit(mappaTile[tile], (x * 32, y * 32))
            else:
                messaggioMappaNonCorretta = True
    return messaggioMappaNonCorretta
```

## Parametri di input:

mappa: una lista di stringhe, ciascuna rappresenta una riga della mappa (proveniente da CreaMappa).

mappaTile: un dizionario che associa a ogni carattere (es. "X", "P", " ", ecc.) un'immagine (Surface) corrispondente al tile da disegnare.

**schermo:** la superficie di Pygame su cui disegnare la mappa.

## Parametri di output: messaggioMappaNonCorretta

**Corpo della funzione:** La funzione controlla ogni riga e ogni carattere della mappa. Se il carattere è nel dizionario mappaTile, disegna l'immagine corrispondente sullo schermo nella posizione giusta, calcolata moltiplicando x e y per 32 pixel. Se invece trova un carattere non valido, la mappa è considerata errata e la funzione restituisce True.

# DataEOrapartita

```
def DataEOrapartita():
    data = datetime.datetime.now()
    data1 = datetime.date.strftime(data, "%d/%m/%y")
    ora = datetime.date.strftime(data, "%H:%M:%S")
    return data1, ora
```

## Parametri di input:

font: indica il font usato per scrivere i nomi dei file delle mappe sullo schermo.

## Parametri di output:

Se l'utente preme un tasto numerico (1-9), restituisce il percorso del file della mappa selezionata (esempio: "MieMappe/nomefile.txt").

Se l'utente preme ESCAPE, restituisce None.

Se l'utente chiude la finestra (QUIT), restituisce "chiudi".

## Corpo della funzione:

La funzione consente al giocatore di scegliere una mappa personalizzata tra i file .txt presenti nella cartella "MieMappe". Sullo schermo viene disegnato uno sfondo fisso e mostrata una lista numerata (da 1 a 9) dei file trovati. La funzione entra in un ciclo che attende l'input del giocatore: se viene premuto un tasto numerico, viene selezionata la mappa corrispondente; se viene premuto ESC, la selezione viene annullata; se la finestra viene chiusa, si segnala che il gioco deve terminare. Il ciclo si ripete finché non viene effettuata una scelta valida o annullata.

# AggiungiGiocatoreAFile

```
def AggiungiGiocatoreAFile(nomeGiocatore, ZombieUccisi):
    data1, ora = DataOraPartita()

    file = open("file/Classifica.txt", "r", encoding="utf-8")
    contenuto = file.read()
    file.close()

    riga = f"Nome: {nomeGiocatore} - Punteggio: {ZombieUccisi} - Data: {data1} - Ora: {ora}\n"

    if nomeGiocatore not in contenuto:
        file = open("file/Classifica.txt", "a", encoding="utf-8")
        file.write(riga)
        file.close()
    else:
        file = open("file/Classifica.txt", "w", encoding="utf-8")
        for linea in contenuto.splitlines():
            if nomeGiocatore in linea:
                # Estrae le uccisioni della partita
                parti = linea.split(" - ")
                uccisioniAttuali = 0
                for parte in parti:
                    if "Punteggio:" in parte:
                        uccisioniAttuali = int(parte.replace("Punteggio:", "").strip()) # si estrae il valore solo per la riga che si sta controllando

                if ZombieUccisi > uccisioniAttuali:
                    file.write(riga)
                else:
                    file.write(linea + "\n") # Se non ci sono cambiamenti viene mantenuta la riga attuale

            else:
                file.write(linea + "\n")

        file.close()

    OrdinaClassifica()
```

## Parametri di input:

nomeGiocatore: indica il nome del giocatore che ha appena terminato la partita.

ZombieUccisi: indica il numero di zombie uccisi dal giocatore nella partita appena conclusa.

## Parametri di output:

Nessun valore di ritorno diretto. La funzione aggiorna il file "File/Classifica.txt" scrivendo o modificando i dati della classifica.

## Corpo della funzione:

La funzione AggiungiGiocatoreAFile aggiorna il file della classifica con i dati di un nuovo giocatore o aggiorna il punteggio di un giocatore già presente.

Se il nome del giocatore non esiste nel file, viene aggiunta una nuova riga con il nome, il punteggio, la data e l'ora.

Se il nome è già presente, viene confrontato il nuovo punteggio con quello registrato: se il nuovo è maggiore, la riga viene aggiornata con il punteggio migliore; se è minore o uguale, la riga esistente viene mantenuta.

Al termine, la funzione chiama OrdinaClassifica() per ordinare il file della classifica in base ai punteggi.

# MostraSceltaMappaPersonale

```
def MostraSceltaMappaPersonale(font):
    cartella = "MieMappe"
    file = []
    listaFile = os.listdir(cartella)

    for i in listaFile:
        if ".txt" in i:
            file.append(i)

    scegliendo = True

    while scegliendo:
        schermo.blit(SfondoMieMappe, (0, 0))

        for indice, nomefile in enumerate(file):
            testo = font.render(F"{indice+1} - {nomefile}", True, (255, 255, 255))
            schermo.blit(testo, (800, 100 + indice * 80))

        pygame.display.update()

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                return "chiudi"
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_1:
                    if len(file) >= 1:
                        return cartella + "/" + file[0]
                if event.key == pygame.K_2:
                    if len(file) >= 2:
                        return cartella + "/" + file[1]
                if event.key == pygame.K_3:
                    if len(file) >= 3:
                        return cartella + "/" + file[2]
                if event.key == pygame.K_4:
                    if len(file) >= 4:
                        return cartella + "/" + file[3]
                if event.key == pygame.K_5:
                    if len(file) >= 5:
                        return cartella + "/" + file[4]
                if event.key == pygame.K_6:
                    if len(file) >= 6:
                        return cartella + "/" + file[5]
                if event.key == pygame.K_7:
                    if len(file) >= 7:
                        return cartella + "/" + file[6]
                if event.key == pygame.K_8:
                    if len(file) >= 8:
                        return cartella + "/" + file[7]
                if event.key == pygame.K_9:
                    if len(file) >= 9:
                        return cartella + "/" + file[8]
                if event.key == pygame.K_ESCAPE:
                    return None
```

## Parametri di input:

font: indica il font usato per scrivere i nomi dei file delle mappe sullo schermo.

## Parametri di output:

Se l'utente preme un tasto numerico (1-9), restituisce il percorso del file della mappa selezionata (esempio: "MieMappe/nomefile.txt").

Se l'utente preme ESCAPE, restituisce None.

Se l'utente chiude la finestra (QUIT), restituisce "chiudi".

## Corpo della funzione:

La funzione MostraSceltaMappaPersonale permette al giocatore di scegliere una mappa personalizzata tra i file .txt presenti nella cartella "MieMappe".

Viene disegnato uno sfondo fisso e viene mostrata la lista dei file trovati, numerati da 1 a 9.

La funzione attende che il giocatore prema un tasto: se preme un numero, viene selezionato il file corrispondente; se preme ESC, si annulla la scelta; se chiude la finestra, viene segnalato di chiudere il gioco.

Il tutto avviene dentro un ciclo che si aggiorna continuamente finché non viene fatta una scelta.

# EstraiUccisioni

```
def estraiUccisioni(riga):
    parti = riga.split(" - ")
    for parte in parti:
        if "Punteggio:" in parte:
            return int(parte.replace("Punteggio:", "").strip())
    return 0
```

## Parametri di input:

riga: una stringa che rappresenta una riga del file classifica, contenente informazioni sul nome del giocatore, il punteggio, la data e l'ora.

## Parametri di output:

Restituisce un numero intero che rappresenta il punteggio (cioè il numero di zombie uccisi) estratto dalla riga. Se non viene trovato un punteggio valido, restituisce 0.

## Corpo della funzione:

La funzione estraiUccisioni prende una riga di testo, la divide in più parti separate dal trattino (-) e cerca tra queste il segmento che contiene la parola "Punteggio:". Una volta trovato, estrae il numero associato, lo converte in intero e lo restituisce. Se per qualsiasi motivo non viene trovato il punteggio, la funzione restituisce 0 come valore di default.

# OrdinaClassifica

```
def OrdinaClassifica(crescente=False):
    file = open("File/Classifica.txt", "r", encoding="utf-8")
    righe = file.readlines()
    file.close()

    for k in range(len(righe), 0, -1):
        for i in range(0, k-1):
            if crescente:
                if estraiUccisioni(righe[i]) > estraiUccisioni(righe[i+1]):
                    righe[i], righe[i+1] = righe[i+1], righe[i]
                else:
                    if estraiUccisioni(righe[i]) < estraiUccisioni(righe[i+1]):
                        righe[i], righe[i+1] = righe[i+1], righe[i]

    # vado a sovrascrivere il tutto
    file = open("File/Classifica.txt", "w", encoding="utf-8")
    for riga in righe:
        file.write(riga + "\n")
    file.close()
```

## Parametri di input:

crescente (default False): indica se ordinare la classifica in ordine crescente (True) o decrescente (False) in base al punteggio dei giocatori.

## Parametri di output:

Nessun valore di ritorno diretto. La funzione sovrascrive il file "File/Classifica.txt" ordinando le righe.

## Corpo della funzione:

La funzione OrdinaClassifica legge tutte le righe del file della classifica e le ordina usando l'algoritmo Bubble Sort, basandosi sul numero di zombie uccisi estratto da ciascuna riga.

Se il parametro crescente è True, i punteggi vengono ordinati dal più basso al più alto; se False, dal più alto al più basso.

Alla fine, il file viene riscritto completamente con le righe nell'ordine corretto.

# SpawnaBoss

```
def SpawnBoss(partenzaSu, fineSu, partenzaGiu, fineGiu, partenzaSx, fineSx, partenzaDx, fineDX, VitaBoss):
    lato = random.choice(["su", "giu", "sinistra", "destra"])
    if lato == "su":
        return [random.randint(partenzaSu, fineSu), -50, VitaBoss]
    if lato == "giu":
        return [random.randint(partenzaGiu, fineGiu), ALTEZZASCHERMO + 50, VitaBoss]
    if lato == "sinistra":
        return [-50, random.randint(partenzaSx, fineSx), VitaBoss]
    if lato == "destra":
        return [LARGHEZZASCHERMO + 50, random.randint(partenzaDx, fineDX), VitaBoss]
```

## Parametri di input:

partenzaSu, fineSu: definiscono i limiti orizzontali entro cui può spawnare il boss dalla parte superiore dello schermo.

partenzaGiu, fineGiu: definiscono i limiti orizzontali per lo spawn nella parte inferiore dello schermo.

partenzaSx, fineSx: definiscono i limiti verticali per lo spawn a sinistra dello schermo.

partenzaDx, fineDX: definiscono i limiti verticali per lo spawn a destra dello schermo.

VitaBoss: indica la quantità di vita che avrà il boss al momento della creazione.

## Parametri di output:

Restituisce una lista contenente la posizione (x, y) iniziale del boss e la sua quantità di vita [x, y, VitaBoss].

## Corpo della funzione:

La funzione SpawnBoss sceglie casualmente un lato dello schermo (alto, basso, sinistra o destra) da cui far apparire il boss.

In base al lato scelto, genera una posizione randomica lungo quel bordo e colloca il boss leggermente fuori dallo schermo (a -50 o +50 pixel), così da creare l'effetto che "entra in campo".

La funzione restituisce la posizione iniziale e la quantità di vita impostata per il boss.

# GestisciBoss

```
def GestisciBoss(ListaBoss, giocatoreX, giocatoreY, velocitaBoss, boss):
    centroX = giocatoreX + 32
    centroY = giocatoreY + 32
    distanzaMinima = 40
    distanzaMinima_sq = distanzaMinima * distanzaMinima

    for i, z in enumerate(ListaBoss):
        dxGiocatore = centroX - z[0]
        dyGiocatore = centroY - z[1]
        distanza_giocatore_quadrato = dxGiocatore**2 + dyGiocatore**2

        if distanza_giocatore_quadrato != 0:
            inv_distanza = 1 / (distanza_giocatore_quadrato**0.5)
            movimentoX = dxGiocatore * inv_distanza * velocitaBoss
            movimentoY = dyGiocatore * inv_distanza * velocitaBoss
        else:
            movimentoX, movimentoY = 0, 0

        for j, altro in enumerate(ListaBoss):
            if i != j:
                dx = z[0] - altro[0]
                dy = z[1] - altro[1]
                dist_sq = dx**2 + dy**2
                if 0 < dist_sq < distanzaMinima_sq:
                    inv_dist = 1 / (dist_sq**0.5)
                    forza_repulsiva = (distanzaMinima_sq - dist_sq) / distanzaMinima_sq
                    movimentoX += dx * inv_dist * forza_repulsiva * velocitaBoss
                    movimentoY += dy * inv_dist * forza_repulsiva * velocitaBoss

        z[0] += movimentoX
        z[1] += movimentoY

    BImg, BRect = RotazioneZombie(boss, z[0], z[1], giocatoreX, giocatoreY)
    schermo.blit(BImg, BRect.topleft)
```

## Parametri di input:

ListaBoss: una lista che contiene tutti i boss presenti nel gioco, con la loro posizione e vita.

giocatoreX, giocatoreY: le coordinate attuali del giocatore.

velocitaBoss: la velocità con cui i boss si muovono verso il giocatore.

boss: l'immagine del boss da disegnare sullo schermo.

## Parametri di output:

Nessun valore di ritorno diretto. La funzione aggiorna le posizioni dei boss e li disegna a schermo.

## Corpo della funzione:

La funzione GestisciBoss gestisce il movimento dei boss facendoli dirigere verso il giocatore.

Ogni boss calcola la distanza dal giocatore e si muove nella sua direzione. Se due boss si avvicinano troppo tra loro (meno di 40 pixel), una forza di repulsione li spinge a separarsi per evitare che si sovrappongano.

Dopo aver aggiornato la posizione di ogni boss, la funzione ruota l'immagine del boss per farla orientare verso il giocatore e la disegna sullo schermo.

# CollisioneBoss

```
def CollisioneBoss(ListaBoss, listaProiettili, ListaSangue, ZombieUccisi):
    daRimuovereB = []
    daRimuovereP = []

    for b in ListaBoss:
        rectB = pygame.Rect(b[0], b[1], 50, 50)
        for p in listaProiettili:
            rectP = pygame.Rect(p[0], p[1], 10, 10)
            if rectB.colliderect(rectP):
                b[2] -= 1
                daRimuovereP.append(p)

    if b[2] <= 0:
        ListaSangue.append([b[0], b[1], pygame.time.get_ticks()])
        daRimuovereB.append(b)
        ZombieUccisi += 10

    for p in daRimuovereP:
        if p in listaProiettili:
            listaProiettili.remove(p)

    for b in daRimuovereB:
        if b in ListaBoss:
            ListaBoss.remove(b)

    return ZombieUccisi
```

## Parametri di input:

ListaBoss: lista contenente tutti i boss presenti nel gioco, con la loro posizione e vita.

listaProiettili: lista contenente i proiettili sparati dal giocatore.

ListaSangue: lista dove vengono memorizzate le posizioni delle macchie di sangue create dopo la morte dei boss.

ZombieUccisi: numero attuale di zombie (e boss) uccisi dal giocatore.

## Parametri di output:

Restituisce il nuovo valore aggiornato di ZombieUccisi.

## Corpo della funzione:

La funzione CollisioneBoss gestisce la collisione tra i proiettili e i boss.

Per ogni boss viene controllato se un proiettile lo colpisce: in caso positivo, si riduce la vita del boss di 1 e il proiettile viene segnato per essere rimosso.

Se la vita del boss scende a 0 o meno, il boss viene eliminato, viene creata una macchia di sangue nella sua posizione e al punteggio del giocatore vengono aggiunti 10 punti.

Alla fine della funzione, i proiettili che hanno colpito e i boss morti vengono rimossi dalle rispettive liste.

# DisegnaMirino

```
def DisegnaMirino(mirino, mouseX, mouseY):
    rett = mirino.get_rect(center=(mouseX, mouseY))
    schermo.blit(mirino, rett)
```

## Parametri di input:

mirino: l'immagine del mirino da disegnare.

mouseX, mouseY: coordinate attuali del puntatore del mouse.

## Parametri di output:

Nessun valore di ritorno diretto. La funzione disegna il mirino sullo schermo.

## Corpo della funzione:

La funzione DisegnaMirino crea un rettangolo (rect) centrato sulle coordinate del mouse.

Successivamente disegna l'immagine del mirino sullo schermo posizionandola esattamente al centro del puntatore.

# GeneraBomba

```
def GeneraBomba(BombaVisibile, tempoUltimaBomba, BombaPos, giocatoreX, giocatoreY, BombaPresa, tempoBombaRaccolta, ListaZombie, ZombieUccisi):
    n = len(ListaZombie)
    tempoAttuale = pygame.time.get_ticks()

    if not BombaVisibile and tempoAttuale - tempoUltimaBomba >= 25000:
        if random.random() < 0.04:
            BombaPos = (random.randint(0, LARGHEZZASCHERMO - 70), random.randint(0, ALTEZZASCHERMO - 70))
            BombaVisibile = True
            tempoUltimaBomba = tempoAttuale

    if BombaVisibile == True:
        rectGiocatore = pygame.Rect(giocatoreX, giocatoreY, 49, 49)
        rectBomba = pygame.Rect(BombaPos[0], BombaPos[1], 50, 50)

        if rectGiocatore.colliderect(rectBomba):
            BombaVisibile = False
            BombaPresa = True
            tempoBombaRaccolta = tempoAttuale
            SuonoBomba.play('
                for z in ListaZombie (function) remove: Any
                    ListaZombie.remove(z)
                    ListaSangue.append([z[0], z[1], pygame.time.get_ticks()])
                    ZombieUccisi += n

            if BombaVisibile:
                schermo.blit(bomba, (BombaPos))

            GestisciSangue(ListaSangue)

            return tempoUltimaBomba, BombaVisibile, BombaPos, BombaPresa, tempoBombaRaccolta, ZombieUccisi
```

## Parametri di input:

BombaVisibile: indica se la bomba è attualmente visibile sullo schermo.

tempoUltimaBomba: indica il momento in cui è stata creata l'ultima bomba.

**BombaPos**: indica la posizione (x, y) dove si trova la bomba.

giocatoreX e giocatoreY: indicano la posizione attuale del giocatore.

BombaPresa: indica se la bomba è già stata raccolta dal giocatore.

tempoBombaRaccolta: indica il momento in cui il giocatore ha raccolto la bomba.

ListaZombie: contiene tutti gli zombie attualmente in gioco.

ZombieUccisi: indica il numero totale di zombie uccisi finora.

## Parametri di output:

tempoUltimaBomba: aggiorna il tempo di creazione dell'ultima bomba.

BombaVisibile: aggiorna se la bomba è ancora visibile o no.

BombaPos: aggiorna la posizione della bomba sullo schermo.

BombaPresa: aggiorna se la bomba è stata presa dal giocatore.

tempoBombaRaccolta: aggiorna il tempo della raccolta della bomba.

ZombieUccisi: aggiorna il conteggio degli zombie uccisi.

## Corpo della funzione:

La funzione GeneraBomba gestisce la creazione e l'interazione della bomba nel gioco.

Ogni 25 secondi, con una probabilità del 4%, viene generata una bomba in una posizione casuale sullo schermo.

Se la bomba è visibile e il giocatore entra in collisione con essa, la bomba viene raccolta: viene suonato un effetto audio, tutti gli zombie presenti vengono rimossi dalla lista e viene tracciata una macchia di sangue per ogni zombie eliminato.

Infine, la funzione aggiorna il conteggio degli zombie uccisi e mostra la bomba a schermo fino alla raccolta.

# Test Effettuati

sul gioco sono stati effettuati diversi test:

1) Test sul file esterno contenente la mappa:

- 1) se il file non è scritto solo con caratteri ammessi il gioco torna direttamente alla pagina precedente
- 2) se il file non è completo il gioco torna alla pagina precedente

2) Test sui powerUp:

- 1) i powerUp, vengono generati solo in alcuni casi, come nel momento in cui il giocatore ha meno di tot vite o è passato un determinato lasso di tempo
- 2) le vite vengono aumentate correttamente, così come i colpi, la bomba elimina correttamente tutti gli zombie

3) Test sui tasti:

- 1) nel caso in cui si premano i primi 3 tasti il gioco apre correttamente le prime 3 mappe
- 2) nel caso in cui si prema il 4 tasto si apre correttamente il menù delle mappe implementate tramite file, e quando si preme esc il gioco torna indietro
- 3) nel caso in cui si prema troppo veloce il tasto del mouse i proiettili vengono comunque sparati ogni 0.5 secondi

4) Test generali sul gioco:

- 1) gli zombie vengono generati correttamente, fanno perdere mezzo cuore, muoiono dopo 1 colpo e ne assegnano 1 al punteggio
- 2) il boss spawna dopo una range compreso tra 45-60 secondi, fa perdere un intero cuore, muore dopo 10 colpi e ne aggiunge 10 al punteggio