Pygame è una libreria piuttosto consistente che si occupa di molti elementi diversi (grafica, suono, input/output ...) i suoi programmatori hanno deciso di dividerlo a sua volta in **sottomoduli**, cioè in files separati nei quali sono raggruppate le funzioni che gestiscono determinati elementi. Alcuni di questi sottomoduli:

|  |  |
| --- | --- |
| display | contiene funzioni per impostare le proprietà della finestra principale |
| image | contiene funzioni per caricare e visualizzare immagini |
| key | contiene funzioni per l'input da tastiera |

Per usare le funzioni di Pygame bisogna inserire all'inizio del nostro programma l'istruzione **import pygame** che importa contemporaneamente tutti i sottomoduli. Questo vuol dire, se ricordate bene, che prima di usare qualunque funzione dovremo anteporre il nome del package (pygame) e del sottomodulo (cursor, image ...) separati da un punto. Ad esempio:

|  |  |
| --- | --- |
| pygame.display.set\_mode() | chiama la funzione set\_mode() del sottomodulo display |
| pygame.image.load() | chiama la funzione load() del sottomodulo image |
| pygame.time.wait() | chiama la funzione wait() del sottomodulo time |

pygame.init()

La **funzione init()** di pygame: essa serve ad inizializzare tutti i sottomoduli e deve sempre essere chiamata prima di tutte le altre funzioni di pygame;

La **funzione set\_mode()** del sottomodulo display: questa funzione prende come parametro una tuple di due numeri (larghezza ed altezza) e crea la finestra principale con le dimensioni assegnate. Essa ci restituisce proprio una **Surface**, che rappresenta l'intero spazio utile della nostra finestra (e che assegneremo da ora in poi alla variabile schermo. Ricordo che in Python una tuple è una lista che non si può modificare.

pygame.display.set\_caption("Gioco dell'Oca")

La **funzione set\_caption()** del sottomodulo display: essa serve proprio a cambiare il titolo della finestra (caption in inglese) con la stringa che gli passiamo come argomento.

schermo = pygame.display.set\_mode((larghezza\_schermo, altezza\_schermo))

Il risultato della funzione viene assegnato alla variabile schermo. Ma cosa ha restituito la funzione? Bene, si tratta di un oggetto di tipo **Surface**, cioè un oggetto di pygame che rappresenta una superficie, e che corrisponde all'area della finestra appena aperta, quindi delle aree su cui disegnare.

BIANCO = (255, 255, 255)

In informatica il modello più usato è l'**RGB**, che usa come colori fondamentali il rosso (Red), verde (Green) e blu (Blue). Ogni colore è quindi rappresentato da tre numeri da 0 a 255 (cioè, in linguaggio informatico, **3 bytes**) che indicano i livelli di rosso, verde e blu che lo caratterizzano.

(0,0,0) **Nero**: assenza di tutti e tre i colori (255, 255, 255) **Bianco**: tutti e tre i colori alla massima intensità Più il numero è piccolo, più è scuro, più è alto, più è acceso,vivace

    pygame.display.flip()

La **funzione flip()** del sottomodulo display serve ad aggiornare la finestra principale. Quando disegniamo qualcosa dobbiamo sempre chiamare la funzione flip() affinchè le modifiche divengano effettive. Serve e va messo nel punto giusto per evitare il flickering, ovvero un effetto visivo fastidioso che si verifica quando un'immagine o una parte di essa lampeggia rapidamente tra due o più stati differenti. Questo fenomeno può essere causato da diverse ragioni, come problemi di refresh dello schermo o problemi nel rendering delle immagini.

    pygame.quit()

La **funzione quit()** "spegne" tutti i moduli di pygame, chiude la finestra e ritorna al sistema operativo senza errori. Va sempre chiamata per ultima, altrimenti, come abbiamo visto, la finestra di pygame non si chiuderà correttamente.

pygame.draw.rect(schermo, (0, 128, 0), rect)

Il **sottomodulo draw** di pygame contiene alcune funzioni che servono a disegnare su una Surface. In questo caso abbiamo disegnato un rettangolo con la **funzione rect()** , i cui parametri all’interno delle parentesi sono: **superficie** (Surface) - superficie su cui disegnare, **colore** (int o tuple) ovvero il colore con cui disegnare e infine il parametro **rect**, ovvero il rettangolo da disegnare, posizione e dimensioni.   
Esso restituisce un rettangolo che delimita i pixel modificati, se non viene disegnato nulla la posizione del rettangolo che delimita sarà la posizione del parametro rect specificato e la sua larghezza e altezza saranno 0

pygame.draw.line(schermo, BIANCO, (x, y), (x, y + DIMENSIONE\_QUADRATO), 1)

In questo altro caso invece abbiamo disegnato una linea retta sulla superficie data con la **funzione line(),** i cui parametri all’interno delle parentesi sono: **superficie** (Surface) - superficie su cui disegnare, **colore** (int o tuple) ovvero il colore con cui disegnare, **pos\_iniziale**, (tupla o lista), cioè la posizione iniziale della linea(x, y),**pos\_finale** (tupla o lista), cioè la posizione finale della linea(x, y), **larghezza** (int), (facoltativo), (se larghezza >= 1, utilizzato per lo spessore della linea (il valore predefinito è 1),se larghezza < 1, non verrà disegnato nulla)

                    schermo.blit(testo\_numero\_casella,(x + 10, y + 10))

Quando creiamo una Surface per la finestra di gioco Pygame, possiamo anche crearne altre, ma le altre per apparire devono essere attivate mediante il **metodo blit().** Notate che la funzione blit() è un metodo dell'oggetto Surface, e quindi va chiamata con la sintassi **nome\_variabile.nome\_metodo(parametri).** Per esempio, in questo caso, dopo aver creato la superficie per l’inserimento del numero della casella, essa non è "disegnata" da nessuna parte: per essere visibile essa deve essere copiata sulla Surface principale (la nostra schermo). **Le regole del metodo blit()** sono: il nome della superficie destinazione(scritto a sinistra del punto), ed il nome della superficie da copiare(scritto come **primo parametro**). Il **secondo parametro** è invece una coppia di interi che rappresenta il punto della destinazione a partire dal quale deve essere copiata la seconda superficie.

                rect = pygame.Rect(x, y, DIMENSIONE\_QUADRATO, DIMENSIONE\_QUADRATO)

La **funzione costruttore Rect(),** che ha lo stesso nome dell'oggetto, è un oggetto geometrico astratto, caratterizzato da una posizione e dalla lunghezza dei suoi lati. Prende come parametri 4 numeri interi: le coordinate x, y dell'angolo in alto a sinistra, la larghezza e l'altezza.

Le **interfacce grafiche (GUI)** permettono all'utente di interagire con il computer tramite mouse, tastiera, e altri dispositivi. Il sistema operativo genera eventi per ogni azione dell'utente, come clic del mouse o pressione di un tasto. Le applicazioni ricevono questi eventi e li elaborano per rispondere alle azioni dell'utente, come cambiare l'aspetto di un'icona quando il mouse vi passa sopra.

                    if evento.type == pygame.KEYDOWN:

In **pygame esiste un oggetto apposito, l'oggetto Event**, nel quale tutte queste informazioni sono "impacchettate" sotto forma di attributi. L'attributo più importante di un Event è il type, che contiene il tipo dell'evento; i vari tipi sono identificati da delle costanti di pygame (cioè delle **variabili predefinite** in pygame stesso, scritte tutte in caratteri maiuscoli) con nomi che ricordano facilmente l'evento. Ad esempio:

|  |  |
| --- | --- |
| MOUSEMOTION | il mouse è stato mosso |
| MOUSEBUTTONDOWN | clic del mouse (è stato premuto un bottone) |
| KEYDOWN | un tasto della tastiera è stato premuto |
| KEYUP | un tasto della tastiera è stato rilasciato |
| QUIT | la finesta è stata chiusa (clic sull'icona della crocetta in alto a destra) |

        for evento in pygame.event.get():

            if evento.type == pygame.QUIT:

                pygame.quit()

Il **sottomodulo event** di pygame contiene alcune funzioni per gestire gli eventi che il sistema operativo trasmette al nostro programma. Abbiamo detto che al nostro programma è assegnata una coda di eventi (per noi invisibile perchè gestita da Windows): **la funzione pygame.event.get()** prende gli eventi dalla coda, li trasforma in oggetti Event di pygame e ci restituisce una lista di Python che contiene i nostri Event in ordine cronologico.

                pygame.quit()

**L'evento "quit"** di Pygame viene generato quando l'utente cerca di chiudere la finestra del gioco o quando viene richiesto di terminare l'applicazione. Questo evento indica al programma di interrompere l'esecuzione e di uscire in modo pulito, salvando eventuali dati o stati di gioco.

Quella è la struttura tipica di un programma scritto per un'interfaccia grafica: c'è una **prima parte di inizializzazione** in cui si definiscono tutte le variabili necessarie, **dopodichè si entra in un ciclo** (che nel seguito chiameremo ciclo principale); in questo ciclo il programma aspetta gli eventi dal sistema operativo, li esamina e poi esegue i compiti richiesti da ogni evento (per il momento abbiamo solo stampato la descrizione degli eventi ricevuti). Il ciclo si interrompe di solito solo quando l'utente chiude la finestra principale, provocando l'uscita dal programma.

**Un evento di tipo KEYDOWN e KEYUP** ha invece degli attributi che ci dicono quale tasto è stato premuto.

|  |  |
| --- | --- |
| unicode | una ***string*** di Python. Se il tasto è un carattere stampabile i (lettera, numero, punteggiatura) la string contiene il carattere; se non è stampabile (es Invio, Alt, ecc.) essa è vuota. |
| key | un ***numero intero*** che determina quale tasto è stato premuto. Ad ogni tasto è assegnato un numero diverso, e per ricordarli tutti facilmente pygame fornisce un elenco di costanti in manera simile a quello che accade con gli eventi (ne parliamo tra poco). |

             elif evento.key == pygame.K\_BACKSPACE:

                 testo\_input = testo\_input[:-1]

             elif evento.key in [pygame.K\_0, pygame.K\_1, pygame.K\_2, pygame.K\_3]:

testo\_input += evento.unicode

**L'attributo key**, come abbiamo detto, non contiene invece un carattere stampabile, ma un numero diverso per ogni possibile tasto della nostra tastiera. Come succede per gli eventi, pygame definisce nel **sottomodulo locals** una serie di costanti (variabili scritte tutte in lettere maiuscole) per ricordare facilmente il tasto: ad esempio K\_ESCAPE è il tasto Esc, K\_TAB il Tab, ecc..

(int(testo\_numero\_casella.get\_width() \* 0.5),

int(testo\_numero\_casella.get\_height() \* 0.5))

Quando si cambiano le dimensioni della finestra impostando i parametri di **pygame.display.set\_mode(),** è necessario assicurarsi che le dimensioni specificate corrispondano a quelle utilizzate in altre parti del codice che dipendono dalla dimensione della finestra. Tuttavia, per evitare questo tipo di problemi, è consigliabile utilizzare **metodi dell'oggetto Surface di Pygame** per ottenere dinamicamente le dimensioni della finestra, tipo:

|  |  |
| --- | --- |
| get\_width() | restituisce la larghezza in pixel della Surface alla quale è applicato. |
| get\_height() | restituisce l'altezza in pixel della Surface alla quale è applicato. |

    clock = pygame.time.Clock()

        clock.tick(10)

L'oggetto più usato del modulo è comunque il **Clock**, un vero orologio che misura i millesimi di secondo. Questo oggetto ha un **metodo tick()** che rappresenta un "battito" dell'orologio. **Deve essere chiamato con un argomento che rappresenta il frame rate (FPS) che si vuole ottenere**: la prima riga **crea un oggetto Clock** di Pygame che aiuta a controllare il tempo all'interno del gioco. Questo oggetto è responsabile di regolare la velocità di aggiornamento del gioco, il che significa che determina quante volte al secondo viene eseguito il ciclo principale del gioco. Useremo **il tick() come ultima istruzione del ciclo principale**.

sfondo\_benvenuto = pygame.image.load('Sfondo1.png')

pedina1\_img = pygame.image.load('Jennifer.jpeg')

Per **utilizzare le immagini** nei nostri programmi la prima cosa che dovremo fare è caricarle da un file grafico. Per fare questo si usa **la funzione load() che si trova nel sottomodulo image.** Questa funzione prende come **argomento una string** (il nome del nostro file) e ci restituisce una Surface con l'immagine del file.

font = pygame.font.Font(None, 36)

font\_regole = pygame.font.Font("Font1.ttf", 24)

Abbiamo già visto che per scrivere qualcosa sulla finestra di pygame **non possiamo usare la print()** di Python, che continua a scrivere sulla finestra di IDLE. Essendo pygame una libreria orientata alla grafica, anche il testo viene trattato come se fosse un'immagine: in pratica **quando vogliamo scrivere del testo dovremo "disegnare" la nostra scritta su una Surface**. Per fare questo esiste **un oggetto apposito, l'oggetto Font. La funzione Font(name, size) di Pygame crea un oggetto Font per visualizzare testo.** Richiede due **parametri**: **name**, il nome del file o il percorso del font, e **size**, la dimensione del font in punti. È utile per creare testo, ma può essere complicato da usare poiché richiede il percorso esatto del font nel sistema operativo. Tuttavia, **può essere semplificato utilizzando None** come nome del file per caricare il font predefinito.

         testo\_numero\_casella = font.render("START", True, BIANCO)

In Pygame, **render** è un **metodo della classe Font** utilizzato per creare una superficie con il testo renderizzato. Prende come **argomenti** il testo da renderizzare, un booleano per l'anti-aliasing e il colore del testo. Richiede una stringa come primo argomento per il testo da renderizzare. Ad esempio, **render(text, antialias, color)** restituisce una superficie con il testo disegnato nel colore specificato. L'anti-aliasing è una tecnica utilizzata per rendere più fluidi i contorni delle immagini o del testo.

pedina6\_img = pygame.transform.scale(pedina6\_img, (DIMENSIONE\_QUADRATO - 20, DIMENSIONE\_QUADRATO - 20))

La **funzione pygame.transform.scale** è una funzione di Pygame utilizzata per **ridimensionare una superficie** (come un'immagine) a una nuova dimensione specificata. Per utilizzare questa funzione, si fornisce **la superficie da ridimensionare** e le nuove dimensioni desiderate come **una tupla (larghezza, altezza).** Pygame quindi ridimensiona la superficie all'interno di un nuovo rettangolo delle dimensioni specificate, mantenendo proporzioni se necessario. Questa funzione è utile per adattare le immagini alle dimensioni desiderate all'interno di un'applicazione Pygame, come ad esempio adattare un'immagine di sfondo alle dimensioni della finestra di gioco.