RAGAZZE DIGITALI ER

Digital Girl Power

Cosa faremo oggi

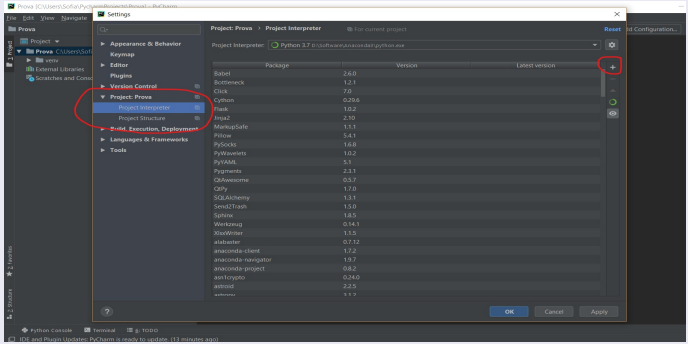
Impareremo ad usare `pygame`, un modulo che permette di creare giochi completi di animazioni, suoni e comandi con il mouse.

Outline

Installare pygame su PyCharm 1/2

Come installare `pygame`

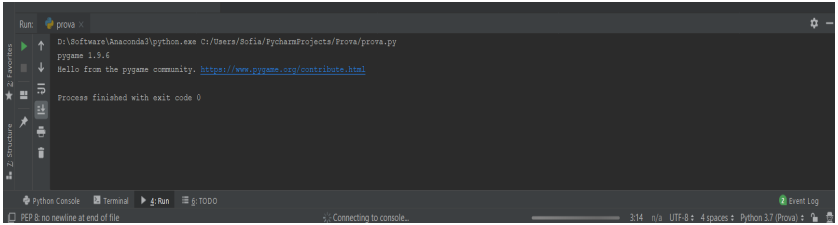
- Per installare pygame su PyCharm è sufficiente aprire il progetto corrente, andare su **File** > **Settings** > **Project:** > **Project Interpreter** (come cerchiato in figura a sinistra); a questo punto, cliccare il pulsante "+" a destra (cerchiato in figura), cercare nella barra in alto *"pygame"*, selezionarlo e cliccare *"Install Package"*



Installare pygame su PyCharm 2/2

Per controllare se è stato correttamente installato, creiamo un file all'interno del progetto, scriviamo queste due righe di codice ed eseguiamo.

```
1 import pygame
2 pygame.init()
```



Sui pc del laboratorio è già installato, quindi ora iniziamo a vedere qualche comando!

Outline

Import e impostazione della finestra

Innanzitutto importiamo e inizializziamo il modulo `pygame` nel nostro programma con queste semplici istruzioni:


```
1 import pygame, sys
2 from pygame.locals import *
3
4 pygame.init()
```

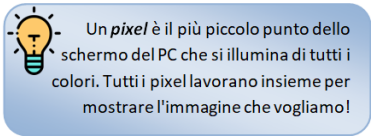
Per impostare la finestra che utilizzeremo il metodo `set_mode()` del modulo `display`, al quale ci si accede dal modulo `pygame`

```
1 windowSurface = pygame.display.set_mode((500, 400))
```

In questo modo, viene impostata una finestra larga **500** pixel e alta **400** pixel.

Import e impostazione della finestra

Innanzitutto importiamo e inizializziamo il modulo `pygame` nel nostro programma con queste semplici istruzioni:




```
1 import pygame, sys
2 from pygame.locals import *
3
4 pygame.init()
```

Per impostare la finestra che utilizzeremo il metodo `set_mode()` del modulo `display`, al quale ci si accede dal modulo `pygame`

```
1 windowSurface = pygame.display.set_mode((500, 400))
```

In questo modo, viene impostata una finestra larga **500** pixel e alta **400** pixel.

Outline

Tuple e colori 1/3

Tuple

- Una tupla è simile ad una lista, con la differenza che si usano le parentesi tonde al posto di quelle quadre e che **non possono essere modificate**.

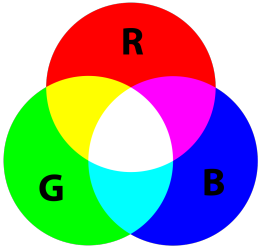
```
1 >>> spam = ('Life', 'Universe', 'Everything', 42)
2 >>> spam[0]
3 >>> spam[3]
4 >>> spam[3] = 'Hello'
```

Eseguiamo, una per volta queste righe e osserviamo il comportamento!
All'interno del metodo `set_mode()` abbiamo usato una tupla per definire la grandezza della finestra in pixel.
Vedremo che ci saranno utili in seguito!

Tuple e colori 2/3

Colori

- Ogni pixel viene illuminato da 3 colori: **rosso**, **verde** e **blu** (RGB). Combinando questi tre semplici colori, si è in grado di ottenere qualsiasi altro colore.
- In pygame, per definire i colori useremo delle tuple formate dai tre valori (**R**ed - **G**reen - **B**lue).




```
1 # Set up the colors.
2 BLACK = (0, 0, 0)
3 WHITE = (?, ?, ?)
4 RED = (255, 0, 0)
5 GREEN = (?, ?, ?)
6 BLUE = (?, ?, ?)
```

Tuple e colori 3/3

Colori

- Ogni numero della tupla rappresenta la "quantità" di ciascun colore, che può andare da 0 a 255.
- Il primo numero rappresenta la quantità di rosso, il secondo la quantità di verde e il terzo quella del blu.

Color	RGB value
Black	(0, 0, 0)
Blue	(0, 0, 255)
Gray	(128, 128, 128)
Green	(0, 128, 0)
Lime	(0, 255, 0)
Purple	(128, 0, 128)
Red	(255, 0, 0)
Teal	(0, 128, 128)
White	(255, 255, 255)
Yellow	(255, 255, 0)

```
1 # Set up the colors.
2 BLACK = (0, 0, 0)
3 WHITE = (255, 255, 255)
4 RED = (255, 0, 0)
5 GREEN = (0, 255, 0)
6 BLUE = (0, 0, 255)
```

Outline

Fonts 1/2

```
1 # Set up the fonts.
2 basicFont = pygame.font.SysFont(None, 48)
```

Impostare un font

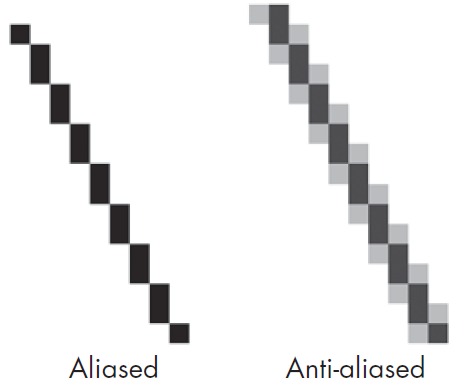
- Attraverso la funzione `pygame.font.SysFont()` è possibile impostare il font del nostro testo.
- Questa funzione prende in input due parametri: il primo è il nome del font (in questo caso viene impostato quello di default) e il secondo parametro è la grandezza.
- Salviamo il risultato di questa funzione nella variabile `basicFont`, sulla quale invocheremo il metodo `render()`

Fonts 2/2

```
1 # Set up the text.
2 text = basicFont.render('Hello world!', True, WHITE, BLUE)
```

Impostare un font

- Questo metodo permette di *renderizzare* la scritta che inseriremo in una variabile `text`.
- Il secondo parametro permette di creare un effetto *"sfumato"* al testo, chiamato anti-alias, come in figura.
- Il terzo e quarto parametro indicano il colore del testo e dello sfondo, rispettivamente.



Come disporre un testo nella finestra

```
1 textRect = text.get_rect()
2 textRect.centerx = windowSurface.get_rect().centerx
3 textRect.centery = windowSurface.get_rect().centery
```

Disposizione testo

- Il metodo `get_rect()` serve per disporre il nostro testo all'interno della finestra.
- Ciò che otteniamo con l'invocazione di questo metodo lo possiamo inserire all'interno di una variabile (`textRect`) e su di essa possiamo accedere ad alcuni valori che ci danno informazioni utili per capire e impostare la posizione del testo.

Tabella di attributi di `Rect`

Table 17-2: Rect Attributes

pygame.Rect attribute	Description
`myRect.left`	Integer value of the x-coordinate of the left side of the rectangle
`myRect.right`	Integer value of the x-coordinate of the right side of the rectangle
`myRect.top`	Integer value of the y-coordinate of the top side of the rectangle
`myRect.bottom`	Integer value of the y-coordinate of the bottom side of the rectangle
`myRect.centerx`	Integer value of the x-coordinate of the center of the rectangle
`myRect.centery`	Integer value of the y-coordinate of the center of the rectangle
`myRect.width`	Integer value of the width of the rectangle
`myRect.height`	Integer value of the height of the rectangle
`myRect.size`	A tuple of two integers: `(width, height)`
`myRect.topleft`	A tuple of two integers: `(left, top)`
`myRect.topright`	A tuple of two integers: `(right, top)`
`myRect.bottomleft`	A tuple of two integers: `(left, bottom)`
`myRect.bottomright`	A tuple of two integers: `(right, bottom)`
`myRect.midleft`	A tuple of two integers: `(left, centery)`
`myRect.midright`	A tuple of two integers: `(right, centery)`
`myRect.midtop`	A tuple of two integers: `(centerx, top)`

Ecco alcuni valori a cui possiamo accedere e che potranno esserci utili in futuro.

Outline

Colorare un oggetto

`fill(...)`

- È possibile riempire un oggetto, una superficie, di colore tramite il metodo `fill()`.
- Questa funzione prende in ingresso una tupla che rappresenta un colore.
- Nel nostro esempio, abbiamo usato questa funzione per impostare lo sfondo della finestra.

```
1 # Draw the white background onto the surface.
2 windowSurface.fill(WHITE)
```

Disegnare un poligono

pygame.draw.polygon(...)

Questa funzione permette di creare qualsiasi tipo di poligono.
Vediamone gli argomenti in input:

- la finestra su cui disegnare il poligono;
- il colore;
- una tupla di n tuple (coppie di valori x,y) che indicano i punti del poligono. L'ultimo punto si collegherà automaticamente al primo;
- un intero, opzionale, che determina lo spessore del contorno. Senza di esso, il poligono è riempito.

```
1 # Draw a green polygon onto the surface.
2 pygame.draw.polygon(windowSurface, GREEN, ((146, 0), (291,
    106), (236, 277), (56, 277), (0, 106)))
```

Disegnare una linea

pygame.draw.line(...)

Questa funzione permette di creare una linea. Vediamone gli argomenti in input:

- la finestra su cui disegnare la linea;
- il colore;
- una tupla di valori x,y che indica un capo della linea;
- una tupla di valori x,y che indica l'altro capo della linea;
- un intero, opzionale, che indica lo spessore della linea.

```
1 pygame.draw.line(windowSurface, BLUE, (60, 60), (120, 60), 4)
2 pygame.draw.line(windowSurface, BLUE, (120, 60), (60, 120))
```

Disegnare un cerchio

pygame.draw.circle(...)

Questa funzione permette di creare un cerchio. Vediamone gli argomenti in input:

- la finestra su cui disegnare il cerchio;
- il colore;
- una tupla di valori x,y che indica il centro del cerchio;
- un intero che indica il raggio;
- un intero, opzionale, che indica lo spessore della linea. Se è 0, il cerchio viene riempito.

```
1 # Draw a blue circle onto the surface.
2 pygame.draw.circle(windowSurface, BLUE, (300, 50), 20, 0)
```

Disegnare un ellisse

`pygame.draw.ellipse(...)`

Questa funzione permette di creare un ellisse. Vediamone gli argomenti in input:

- la finestra su cui disegnare l'ellisse;
- il colore;
- una tupla contenente l'angolo di sinistra e in alto, la larghezza e l'altezza dell'ellisse;
- un intero, opzionale, che indica lo spessore della linea. Se è 0, l'ellisse viene riempito.

```
1 # Draw a red ellipse onto the surface.
2 pygame.draw.ellipse(windowSurface, RED, (300, 250, 40, 80), 1)
```

Disegnare un rettangolo

pygame.draw.rect(...)

Questa funzione permette di creare un rettangolo. Vediamone gli argomenti in input:

- la finestra su cui disegnare il rettangolo;
- il colore;
- una tupla con 4 valori, rispettivamente le coordinate x,y per gli spigoli alto-sinistra, larghezza e altezza.

```
1 pygame.draw.rect(windowSurface, RED, (textRect.left - 20,
2 textRect.top - 20, textRect.width + 40, textRect.height + 40))
```

Disegnare una superficie dentro l'altra

blit(...)

Questo metodo permette di disegnare due superfici, una dentro l'altra. Nel nostro esempio, infatti, abbiamo la superficie contenente la scritta e la superficie (la finestra) contenente le figure. Vediamo come sovrapporle.

```
1 # Draw the text onto the surface.
2 windowSurface.blit(text, textRect)
```

In questo caso, la superficie contenente la scritta viene inserita all'interno della finestra con le figure.
Facciamo bene attenzione quindi all'ordine con cui sovrapponiamo le due finestre!

Outline

Stampare a schermo le nostre creazioni

```
1 # Draw the window onto the screen.
2 pygame.display.update()
```

Con questo semplice metodo riusciamo a rendere visibile sul nostro display tutto ciò che abbiamo creato fino ad ora!

Eventi e game loop

Eventi

- Un evento è qualcosa che accade all'interno del nostro programma, come un click o movimento del mouse o la pressione di un tasto.
- Possiamo intercettare gli eventi tramite la funzione `get()`, mentre il valore `type` (a cui accediamo tramite `event.type`) indica il tipo di evento di cui si tratta.
- L'`if` contenuto nel game loop intercetta l'evento di uscita e se si verifica, vengono invocati i metodi `pygame.quit()` e `sys.exit()`.

```
1 # Run the game loop.
2 while True:
3     for event in pygame.event.get():
4         if event.type == QUIT:
5             pygame.quit()
6             sys.exit()
```

Esempio

Proviamo l'esempio

Outline

Stato del gioco

Spesso ci servono informazioni diverse che riguardano una stessa entità. Ad esempio del giocatore vogliamo conoscere la vita, la posizione e il punteggio. Magari queste stesse informazioni ci servono per varie "copie" dell'entità, ad esempio per ognuno dei giocatori e per ognuno dei nemici.
Abbiamo già visto alcuni tipi di strutture dati come le tuple e le liste. Un'altra che ci può essere utile per raggruppare le informazioni è il **dizionario**.

Dizionari

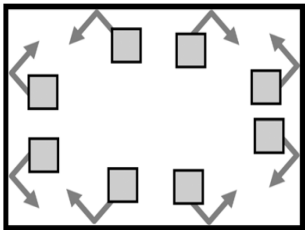
I dizionari sono simili alle liste, possono contenere elementi di vario tipo e possono essere modificati. La differenza principale è che non si accede agli elementi in base alla loro posizione bensì tramite il loro nome. Gli elementi del dizionario, infatti, sono identificati da un nome (una stringa).
I dizionari sono dichiarati usando le parentesi graffe al cui interno ci sono i vari elementi separati da virgole in modo analogo alle liste e alle tuple. Tuttavia, ogni elemento è preceduto da una stringa seguita dai due punti. Vediamo alcuni esempi.

```
1 player = {'life': 100, 'name': 'Alice', 'position': (100, 100)}
2 player['life'] = 42
3 print(player['life'])
```

Outline

Animazioni

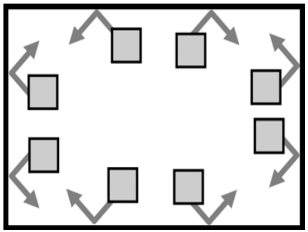
Costruiremo un programma nel quale 3 quadrati colorati diversamente si muoveranno all'interno della finestra. Ogni quadratino si muoverà in diagonale in una delle 4 direzioni. Se si scontra con un bordo della finestra, rimbalzerà in un'altra direzione, come mostrato in figura sotto.



- In quale direzione si muoverà il quadratino? Da cosa dipende?

Animazioni

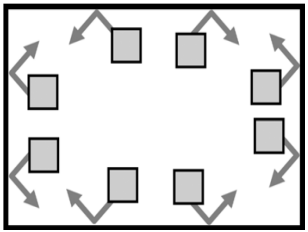
Costruiremo un programma nel quale 3 quadrati colorati diversamente si muoveranno all'interno della finestra. Ogni quadratino si muoverà in diagonale in una delle 4 direzioni. Se si scontra con un bordo della finestra, rimbalzerà in un'altra direzione, come mostrato in figura sotto.



- In quale direzione si muoverà il quadratino? Da cosa dipende?
  1) Dalla direzione in cui si muoveva prima di rimbalzare;

Animazioni

Costruiremo un programma nel quale 3 quadrati colorati diversamente si muoveranno all'interno della finestra. Ogni quadratino si muoverà in diagonale in una delle 4 direzioni. Se si scontra con un bordo della finestra, rimbalzerà in un'altra direzione, come mostrato in figura sotto.



- In quale direzione si muoverà il quadratino? Da cosa dipende?
  1) Dalla direzione in cui si muoveva prima di rimbalzare;
  2) In che muro ha rimbalzato.

Inizializzazione gioco

```
1  import pygame, sys, time
2  from pygame.locals import *
3  # Set up pygame.
4  pygame.init()
5  # Set up the window.
6  WINDOWWIDTH = 400
7  WINDOWHEIGHT = 400
8  windowSurface = pygame.display.set_mode((WINDOWWIDTH, WINDOWHEIGHT), 0, 32)
9  pygame.display.set_caption('Animation')
10 # Set up direction constants.
11 DOWNLEFT = 'downleft'
12 DOWNRIGHT = 'downright'
13 UPLEFT = 'upleft'
14 UPRIGHT = 'upright'
15 MOVESPEED = 4
16 # Set up the colors.
17 WHITE = (?, ?, ?)
18 RED = (?, ?, ?)
19 GREEN = (?, ?, ?)
20 BLUE = (?, ?, ?)
21
22 # Set up the box data structure.
23 b1 = {'rect':pygame.Rect(300, 80, 50, 100), 'color':RED, 'dir':UPRIGHT}
24 b2 = {'rect':pygame.Rect(200, 200, 20, 20), 'color':GREEN, 'dir':UPLEFT}
25 b3 = {'rect':pygame.Rect(100, 150, 60, 60), 'color':BLUE, 'dir':DOWNLEFT}
26 boxes = [b1, b2, b3]
```

Game Loop 1/3

```
27 while True:
28     # Check for the QUIT event.
29     for ? in ?:
30                 ?
31
32     # Draw the white background onto the
           surface.
33     windowSurface.fill(WHITE)
34
35     for b in boxes:
36     # Move the box data structure.
37         if ? == ?:
38             ?
39         if ? == ?:
40             ?
41         if ? == ?:
42             ?
43         if ? == ?:
44             ?
```

All'interno del secondo `for`, inseriamo una serie di controlli che permettono alle figure di muoversi.

- **SE** la direzione è `DOWNLEFT` o `DOWNRIGHT`, va *incrementato* l'attributo `top` di una quantità pari alla velocità di movimento;
- **SE** la direzione è `UPLEFT` o `UPRIGHT`, va *decrementato* l'attributo `top` di una quantità pari alla velocità di movimento;
- **SE** la direzione è `DOWNRIGHT` o `UPRIGHT`, va *incrementato* l'attributo `left` di una quantità pari alla velocità di movimento;
- **SE** la direzione è `DOWNLEFT` o `UPLEFT`, va *decrementato* l'attributo `left` di una quantità pari alla velocità di movimento;

Game Loop 2/3

```
46      if b['rect'].top < 0:
47          # The box has moved past the top.
48          ?
49
50      if b['rect'].bottom > WINDOWHEIGHT:
51          # The box has moved past the bottom.
52          ?
53
54      if b['rect'].left < 0:
55          # The box has moved past the left side.
56          ?
57
58      if b['rect'].right > WINDOWWIDTH:
59          # The box has moved past the right side.
60          ?
```

Sempre all'interno del secondo `for`, inseriamo una serie di controlli che permettono alle figure di "rimbalzare" nel caso in cui si scontrino con il bordo.

- **SE** ha toccato il bordo superiore, se la direzione era `UPLEFT` diventerà `DOWNLEFT`, se era `UPRIGHT`, diventerà `DOWNRIGHT`
- **SE** ha toccato quello inferiore, se la direzione era `DOWNLEFT` diventerà `UPLEFT`, se era `DOWNRIGHT` diventerà `UPRIGHT`
- **SE** ha toccato quello destro, cosa quale sarà la nuova direzione? E quello sinistro?

Game Loop 3/3

```
62          # Draw the box onto the surface.
63          pygame.draw.rect(windowSurface, b['color'], b['rect'])
64
65      # Draw the window onto the screen.
66      pygame.display.update()
67      time.sleep(0.02)
```

Ultime righe di questo codice che permettono di disegnare ogni rettangolo nella nuova posizione.
Occhio che questa la prima funzione è inserita nel `for`, mentre le ultime due sono fuori dal `for` ma sempre dentro il game loop.
La funzione `display.update()` serve per aggiornare la nostra finestra, in modo che sia visibile lo spostamento delle figure!

Soluzione

Materiale rilasciato con licenza

Creative Commons - Attributions, Share-alike 4.0