RAGAZZE DIGITALI ER

Digital Girl Power

Cosa faremo oggi

Vedremo più nel dettaglio alcune funzionalità di `pygame` che ci potranno essere utili per lo sviluppo del progetto finale. Per farlo, analizzeremo un primo prototipo di gioco.

La documentazione ufficiale

Fino ad ora abbiamo presentato alcune funzionalità di `pygame` in modo molto sommario. Il posto giusto dove trovare tutti i dettagli è la documentazione ufficiale.

Outline

La finestra di gioco

```
1 WINDOW_WIDTH = 400
2 WINDOW_HEIGHT = 400
3 windowSurface = pygame.display.set_mode((WINDOW_WIDTH,
     WINDOW_HEIGHT))
4 pygame.display.set_caption('Input')
```

`pygame.display.set_mode(...)`

- La funzione `set_mode(...)` inizializza una nuova finestra
- I suoi parametri completi sono:
  - `size`: una tupla che indica la dimensione della finestra
  - `flags`: alcune opzioni (ad esempio per rendere la finestra fullscreen)
  - `depth`: la profondità di colore
  - `display`: quale display usare
  - `vsync`: booleano per abilitare o meno la sincronizzazione verticale

Il giocatore e il cibo

Creiamo le coordinate rettangolari che rappresentano il giocatore e salviamolo nella variabile player.

```
1 X_POSITION = 300
2 Y_POSITION = 100
3 PLAYER_WIDTH = 50
4 PLAYER_HEIGHT = 50
5 player = pygame.Rect(X_POSITION, Y_POSITION, PLAYER_WIDTH,
     PLAYER_HEIGHT)
```

Il giocatore e il cibo

`pygame.Rect(...)`

- La funzione `pygame.Rect(...)` crea delle coordinate rettangolari, ovvero coordinate che rappresentano un rettangolo
- I suoi parametri completi (ma esistono altre versioni) sono:
  - `left`: coordinata x del lato sinistro del rettangolo
  - `top`: coordinata y del lato superiore del rettangolo
  - `width`: larghezza del rettangolo
  - `height`: altezza del rettangolo

Il giocatore e il cibo

Creiamo un altro rettangolo, come abbiamo fatto per player. Questa volta più piccolo.

Come possiamo fare per...

dare al nuovo rettangolo una posizione (X, Y) random?
Cosa devo scrivere al posto dei ??? (ricordiamoci di aggiungere import random all'inizio del nostro file)

```
1 foodSize = 20
2 # Non costante... modificheremo il valore durante il gioco
3 food = pygame.Rect(???, ???, foodSize, foodSize)
```

Qual è la posizione (x, y) di food?

Possiamo sempre riottenere i valori delle coordinate ad esempio:

```
1 print(food.x, food.y)
```

Outline

Eventi

Come abbiamo visto nelle lezioni scorse, `pygame` ci segnala gli eventi.

Alcune tipologie di eventi sono:

- QUIT: Evento generato quando il giocatore chiude la finestra di gioco.
- KEYDOWN: Evento generato quando il giocatore inizia a premere un tasto qualsiasi.
- KEYUP: Evento generato quando il giocatore rilascia un tasto qualsiasi.
- MOUSEMOTION: Evento generato quando il mouse si muove all'interno della nostra finestra.
- MOUSEBUTTONDOWN: Evento generato quando viene premuto un tasto del mouse all'interno della nostra finestra.
- MOUSEBUTTONUP: Evento generato quando viene rilasciato un tasto del mouse all'interno della nostra finestra.

Outline

Input da tastiera

Settiamo 4 variabili, che rappresentano le diverse direzioni, con valori booleani inizialmente False. Quando il giocatore userà il tasto sinistra cambieremo il valore corrispondente in True (e faremo lo stesso per gli altri tre tasti). Quando il giocatore lascerà il tasto premuto setteremo di nuovo il valore corrispondente a False

```
1 moveLeft = False
2 moveRight = False
3 moveUp = False
4 moveDown = False
```

Gestiamo l'evento KEYDOWN (dentro il while True:)

Se l'evento scatenato è KEYDOWN allora l'evento stesso ha un attributo chiamato "key" che indica quale tasto è stato premuto.

```
for event in pygame.event.get():
    if event.type == KEYDOWN:
        if event.key == K_LEFT:
            moveRight = False
            moveLeft = True
        if event.key == K_RIGHT:
            moveLeft = False
            moveRight = True
        if event.key == K_UP:
            moveDown = False
            moveUp = True
        if event.key == K_DOWN:
            moveUp = False
            moveDown = True
```

Table 19-2: Constant Variables for Keyboard Keys

pygame constant variable	Keyboard key	pygame constant variable	Keyboard key
K_LEFT	Left arrow	K_HOME	HOME
K_RIGHT	Right arrow	K_END	END
K_UP	Up arrow	K_PAGEUP	PGUP
K_DOWN	Down arrow	K_PAGEDOWN	PGDN
K_ESCAPE	ESC	K_F1	F1
K_BACKSPACE	Backspace	K_F2	F2
K_TAB	TAB	K_F3	F3
K_RETURN	RETURN or ENTER	K_F4	F4
K_SPACE	Spacebar	K_F5	F5
K_DELETE	DEL	K_F6	F6
K_LSHIFT	Left SHIFT	K_F7	F7
K_RSHIFT	Right SHIFT	K_F8	F8
K_LCTRL	Left CTRL	K_F9	F9
K_RCTRL	Right CTRL	K_F10	F10
K_LALT	Left ALT	K_F11	F11
K_RALT	Right ALT	K_F12	F12

Gestiamo l'evento KEYUP

Se l'evento scatenato è KEYUP il nostro quadrato si dovrà fermare. Settiamo quindi la direzione del tasto rilasciato a False.

```
1  if event.type == KEYUP:
2      if event.key == K_ESCAPE:
3          pygame.quit()
4          sys.exit()
5      if event.key == K_LEFT:
6          moveLeft = False
7      if event.key == K_RIGHT:
8          moveRight = False
9      if event.key == K_UP:
10         moveUp = False
11     if event.key == K_DOWN:
12         moveDown = False
```

Outline

Muovere il giocatore

Velocità di movimento

Per muovere il giocatore cambieremo la sua posizione aggiungendo o sottraendo un valore `MOVE_SPEED` (che rappresenta la velocità di movimento) alle sue coordinate.
Dobbiamo inizializzare la costante `MOVE_SPEED` all'inizio del nostro programma, dove abbiamo inizializzato le altre costanti.

```
1 MOVE_SPEED = 6
```

Muovere il giocatore

Muovere il giocatore

Se una delle 4 variabili, `moveDown`, `moveUp`, `moveLeft` o `moveRight` ha valore True dobbiamo muovere le coordinate del giocatore (memorizzate nella variabile `player`).
In particolare possiamo muovere il giocatore verso l'alto se questo non è già arrivato all'estremo superiore della finestra, in basso se non è arrivato a quello inferiore e così via...
Vediamo un esempio per l'alto e il basso.

```
1 if moveDown and player.bottom < WINDOW_HEIGHT:
2     player.top = player.top + MOVE_SPEED
3 if moveUp and player.top > 0:
4     player.top = player.top - MOVE_SPEED
```

Muovere il giocatore

Come possiamo fare per...

aggiungere il movimento verso destra e verso sinistra? Cosa devo scrivere al posto dei ???

```
1 if moveLeft and ???:
2     player.left ??? ???
3 if moveRight and ???:
4     player.right ??? ???
```

Outline

Disegnare il giocatore e il cibo

Per adesso abbiamo solo lavorato con le coordinate del giocatore e del cibo, rispettivamente `player` e `food`, inizializzandole e cambiandole quando necessario. Non abbiamo ancora mostrato (né disegnato) nulla.

```
1 windowSurface.fill(WHITE) # A cosa serve?
2 # Notiamo la differenza tra il 'rect' come disegno
3 # e il 'rect' come coordinate rettangolari
4 pygame.draw.rect(windowSurface, BLACK, player)
5 pygame.draw.rect(windowSurface, GREEN, food)
```

Outline

Collision detection

Spesso ci serve sapere se due oggetti collidono tra loro, ad esempio il giocatore e il cibo. Uno dei possibili modi è verificare se le loro coordinate rettangolari si intersecano.

`colliderect(...)`

Ogni `Rect`, come `player`, ha un metodo `colliderect(...)` che ritorna True se collide con l'altro passato come parametro. Facciamolo tra il giocatore e il cibo.

Collision Detection

```
# se il giocatore collide con il cibo
if player.colliderect(food):
    # faccio qualcosa, ad esempio lo coloro di rosso
    pygame.draw.rect(windowSurface, RED, food)
```

O se avessimo tanti "cibi" memorizzati in una lista `foods`

```
# scorriamo una copia della lista
for food in list(foods):
    if player.colliderect(food):
        # rimuoviamo i cibi colpiti
        foods.remove(food)
```

Outline

Input da mouse

Abbiamo visto prima che esistono gli eventi legati al mouse: `MOUSEMOTION`, `MOUSEBUTTONDOWN` e `MOUSEBUTTONUP`.

Questi eventi possiedono alcuni attributi che ci possono essere utili:

- `MOUSEMOTION` possiede l'attributo `pos` che indica la posizione del mouse all'interno della finestra. La posizione è rappresentata come tupla (x, y) dove x e y sono le coordinate del mouse.
- `MOUSEBUTTONDOWN` e `MOUSEBUTTONUP` possiedono l'attributo `button` che indica quale tasto del mouse è stato, rispettivamente, premuto o rilasciato.
  1 rappresenta il tasto sinistro, 2 il tasto centrale (se presente), 3 il tasto destro, 4 se la rotella è stata mossa verso l'alto o 5 se mossa verso il basso.

Esempio di eventi provenienti dal mouse

Dentro al for di gestione degli eventi mettiamo alcune stampe per vedere cosa sta succedendo e aumentiamo la dimensione del cibo quando un pulsante del mouse è rilasciato

```
1  if event.type == MOUSEMOTION:
2      print(event.pos[0], event.pos[1])
3  if event.type == MOUSEBUTTONUP:
4      # Aumentiamo di 5 la dimensione di 'food'
5      foodSize = foodSize + 5
6      food = pygame.Rect(food.x, food.y, foodSize, foodSize)
7  if event.type == MOUSEBUTTONDOWN:
8      if event.button == 1:
9          print('Hai premuto il tasto sinistro del mouse')
10     elif event.button == 3:
11         print('Hai premuto il tasto destro del mouse')
12     else:
13         print("Hai premuto il tasto", event.button)
```

Outline

Aggiorniamo il display

`pygame.display.update()`

Le proprietà del nostro giocatore sono cambiate, come anche quelle del cibo. Il giocatore è stato spostato, il cibo potrebbe aver cambiato colore o dimensione (nella versione a più cibi invece potrebbe essere stato rimosso).
Abbiamo già ridisegnato tutto ma i disegni non sono ancora visibili, dobbiamo aggiornare il display come abbiamo visto le scorse lezioni.

```
1 pygame.display.update()
```

Creiamo un orologio

`pygame.time.Clock()`

Un oggetto `Clock()` aiuta a tenere traccia dello scorrere del tempo e a gestire la velocità di avanzamento del gioco (framerate).
Inizializziamo, all'inizio del nostro programma, una variabile globale chiamata `mainClock`.

```
1 mainClock = pygame.time.Clock()
```

Regoliamo la velocità del gioco

`tick(...)`

Questo metodo segnala all'orologio che è passato un "tick", un "frame" e va quindi chiamato alla fine di ogni ciclo del game loop. Come possiamo vedere dalla documentazione, prende un parametro opzionale `framerate` che limita la velocità di esecuzione.

```
1 mainClock.tick(40)
```

Per quale motivo è necessaria? Che differenza c'è con `time.sleep()` vista in precedenza?

Contiamo i secondi

pygame.time.get_ticks()

Vediamo ora un possibile modo di contare i secondi che passano. Useremo la funzione `pygame.time.get_ticks()` per ottenere il tempo trascorso in millisecondi.

```
1 # All'inizio del nostro codice
2 startTime = pygame.time.get_ticks()
3
4 # Alla fine del nostro codice, come ultima riga del while True:
5 elapsedTime = pygame.time.get_ticks()
6 print('Tempo trascorso:',
7     int(((elapsedTime - startTime)/1000)))
```

Outline

Immagini, carichiamole

Andiamo ora a sostituire il nostro quadrato nero (per il giocatore) e verde (per il cibo) rispettivamente con l'icona di pacman e un pezzo di pizza. Imposteremo anche un'immagine di sfondo per il background del nostro gioco.

`pygame.image.load(...)`

Carica una immagine da un file e restituisce la superficie.

- ha un parametro `filename` che indica il nome del file da caricare

Immagini, carichiamole

Per comodità lasciamo le immagini nella stessa cartella del file Python e poi carichiamole salvando le relative superfici.

```
1 backgroundImage = pygame.image.load('background.png')
2 playerImage = pygame.image.load('pacman.png')
3 foodImage = pygame.image.load('pizza.png')
```

Download background.png
Download pacman.png
Download pizza.png

Immagini, ridimensioniamole

pygame.transform.scale(...)

Questa funzione ridimensiona un'immagine (più propriamente, una superficie)

- Prende come parametri:
  - `surface`: la superficie da ridimensionare
  - `size`: la dimensione desiderata come tupla (width, height)
  - `dest_surface`: una opzionale superficie di destinazione (se questa non è specificata, una nuova superficie viene restituita)

```
1 backgroundStretchedImage =
    pygame.transform.scale(backgroundImage, (WINDOW_WIDTH,
    WINDOW_HEIGHT))
2 playerStretchedImage = pygame.transform.scale(playerImage,
    (PLAYER_WIDTH, PLAYER_HEIGHT))
3 foodStretchedImage = pygame.transform.scale(foodImage,
    (foodSize, foodSize))
```

Immagini, mostriamole

Ora non ci serve più colorare di bianco lo sfondo o disegnare i rettangoli, al loro posto dobbiamo mostrare le immagini. Avevamo già incontrato questa metodo...

`blit(...)`

Ogni superficie possiede questo metodo che disegna **l'altra superficie**, passata come parametro `source`, **su questa superficie** alle coordinate passate `dest`. Ha altri parametri che possiamo ignorare al momento.

```
1 # windowSurface.fill(WHITE)
2 windowSurface.blit(backgroundStretchedImage,
     windowSurfaceRectangle)
3 windowSurface.blit(playerStretchedImage, player)
4 windowSurface.blit(foodStretchedImage, food)
5 # pygame.draw.rect(windowSurface, BLACK, player)
6 # pygame.draw.rect(windowSurface, GREEN, food)
```

Esempi

L'esempio costruito finora, con un solo cibo e le immagini - Download

Esempio senza immagini ma con una lista di cibi che appaiono periodicamente o al click del mouse e vengono mangiati - Download

Materiale rilasciato con licenza

Creative Commons - Attributions, Share-alike 4.0