Qualche parola per cominciare

Python è un linguaggio di programmazione facile da imparare e ciononostante potente: la fatica per apprenderlo è ripagata appena ci si accorge di quante cose si possono fare e quanto sia facile farle, almeno a livello elementare, senza dover ricorrere a meccanismi esoterici e oscuri come a volte capita in altri linguaggi di programmazione.

PyGame è una libreria che nasce con l'intento di travasare la facilità di programmazione tipica del Python anche nella realizzazione di videogiochi, semplici e non. In effetti è la riscrittura in salsa Python di una libreria, SDL, nata per semplificare la vita a chi vuole sfruttare le potenzialità della grafica bidimensionale e del suono dei PC per realizzare videogiochi con lo stesso stile di quelli che si vedevano nelle sale giochi e negli home-computer degli anni '80.
Il risultato ottenuto con la libreria originale ha permesso la creazione di prodotti open source di buon livello, se non ottimo, ed è ancora attivamente sviluppata. Se volete vedere cosa si riesce a fare con PyGame, la cosa migliore è andare sul sito dedicato, all'indirizzo http://www.pygame.org

Quello che ci proponiamo di fare con questo tutorial è ricreare il primo videogioco commerciale di un certo successo nelle sale giochi. Era il 1972, i bar ospitavano solo flipper meccanici/elettrici e (in Italia) tavoli da calciobalilla, e la leggenda narra che un giovane Nolan Bushnell assieme ai suoi collaboratori propose a un gestore di un piccolo pub californiano uno strano televisore in bianco e nero con due manopole e una gettoniera, per sperimentare sul campo le potenzialità del gioco. Aveva già provato a lanciare commercialmente un altro apparecchio per giocare al bar, ma che aveva bisogno di un computer universitario per funzionare, e per questo era troppo costoso; questa nuova macchina era invece costruita con componenti elettronici semplici e progettata solo per eseguire il videogioco, e quindi era più abbordabile come costi di realizzazione. Durante la notte il gestore lo chiamò a casa dicendogli che la macchina non funzionava più. Bushnell, preoccupato perché da quel prototipo dipendeva il futuro della sua piccolissima società già in bolletta, si precipitò nel locale per trovare un rimedio; e si accorse che il malfunzionamento era dovuto...alla gettoniera che scoppiava di monetine! Appena quattro anni dopo venderà la sua società di videogiochi, Atari, per 28 milioni di dollari. E poi dicono che sono solo giochi…

Lo scopo del gioco è respingere una pallina che viaggia per lo schermo con una racchetta che si muove in verticale su un lato; se la cosa non riesce, l'avversario dall'altra parte guadagna un punto. Il gioco è vinto da chi riesce a raggiungere il numero di punti richiesto.

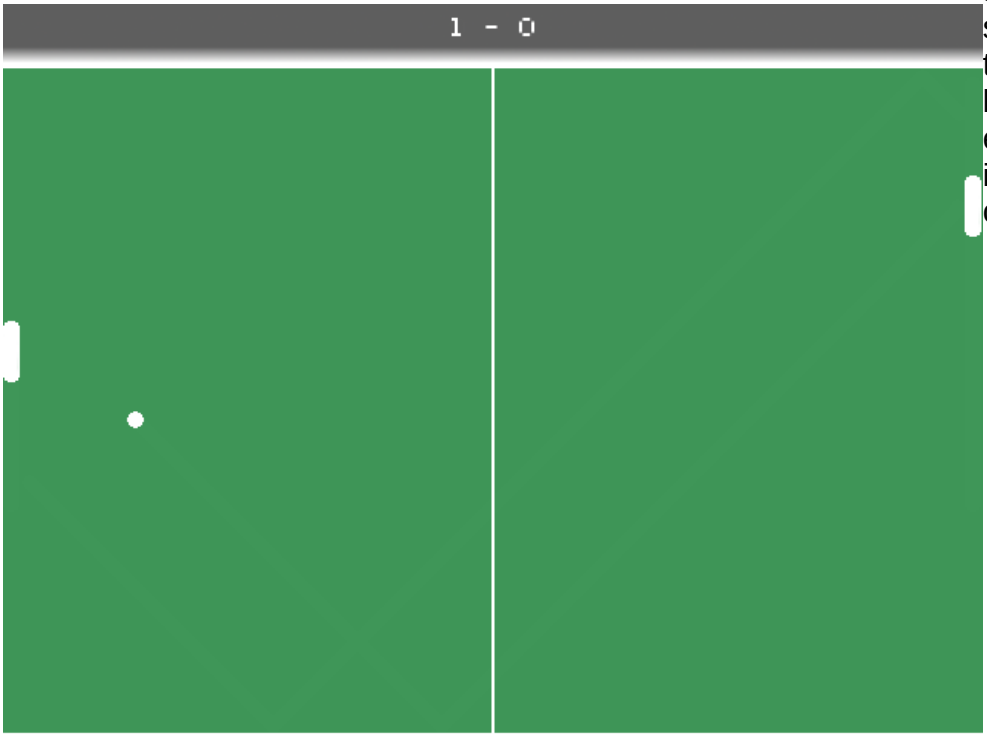
Per il gioco occorre, oltre a Python, la libreria PyGame; se Python è già installato sul sistema la cosa più semplice è installare la libreria con l'uso del comando pip. All'indirizzo https://www.pygame.org/wiki/GettingStarted troverete tutte le istruzioni i sistemi più diffusi. Inoltre serve un editor di testo, ovvero qualcosa di molto più semplice rispetto a Word o LibreOffice, ma che rappresenta lo strumento di lavoro principale di ogni buon programmatore. Se non sapete cosa usare, tra gli esempi di codice installati dal linguaggio c'è anche l'editor IDLE, oppure il suo analogo più moderno Thonny. Li dovreste trovare tra i programmi disponibili sul vostro desktop.

Esiste la possibilità di scaricare la libreria PyGame come archivio .zip dal sito, questo garantisce che assieme al codice Python arrivino anche le librerie di supporto per il sistema operativo (SDL e altre); l'indirizzo con le istruzioni per scaricare il pacchetto giusto per il vostro sistema è http://www.pygame.org/download.shtml. Se avete un sistema Linux, la libreria dovrebbe essere disponibile come pacchetto aggiuntivo dedicato alla versione della vostra distribuzione; per le distribuzioni derivate da Debian e Ubuntu è sicuramente presente la versione adatta al ramo 2.7 di Python. Comunque, il codice è stato testato sia sul ramo 2.7 che sul ramo 3.5 senza trovare incompatibilità degne di nota.

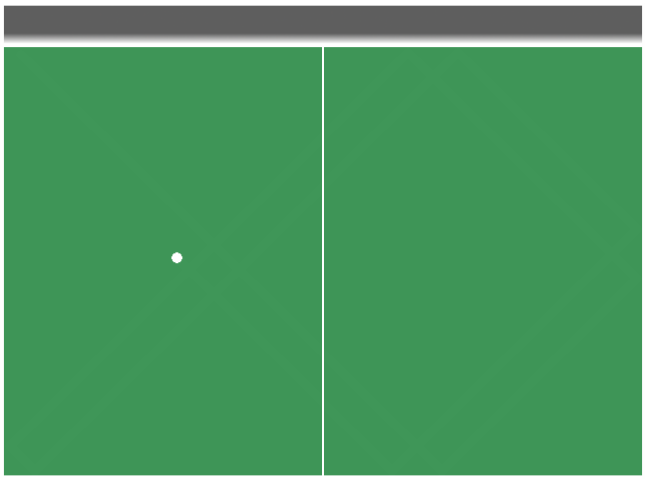
Per seguire meglio il tutorial, l'intero progetto è disponibile per il download all'indirizzo https://github.com/amigoneDaCastagneto/pong_a_2 da dove potrete scaricare tutti i componenti necessari, compresa questa documentazione in formato ODT (OpenOffice/LibreOffice) e PDF. In particolare, la directory img/ contiene elementi grafici utilizzati dal progetto, e deve essere contenuta nella stessa directory del codice.

Questo tutorial è pensato come un progetto in tre step successivi. Ciascuno di essi è però completo di per sé, ovvero al termine avrete sempre un programma completo che esegue quello che gli si richiede. Lo step successivo, poi, prevede modifiche al codice dello step precedente per aumentarne le funzionalità. Si parte con una pallina che rimbalza sullo schermo, si passa a una paletta in grado di respingere la pallina e si finisce con avere due palette che si respingono la pallina a vicenda per non far marcare un punto all'avversario. Per non perdere quanto realizzato in precedenza, meglio salvare separatamente ogni step con un nome diverso, magari già dopo aver apportato le prime modifiche per non rischiare di riscrivere codice diverso con lo stesso nome.

Nel testo, il codice sorgente è individuato con un carattere diverso e in grassetto ed è separato dalle spiegazioni. Data la particolare sintassi del linguaggio, per il quale anche l'indentazione fa parte integrante della sua grammatica, in ogni riga di codice gli spazi che precedono l'istruzione sono evidenziati con pallini grigi. Quando vi sarà richiesto di sostituire parti del codice per passare da uno step del progetto al successivo, le nuove righe saranno evidenziate con uno sfondo giallo (o grigio chiaro, se avete stampato questo tutorial in bianco e nero). Naturalmente, se al posto degli spazi volete usare delle tabulazioni siete liberi di farlo, ma dovrete usare sempre un solo sistema per tutto il codice: solo spazi o solo tabulazioni, l'interprete è tassativo in questo caso.



Passo 1: una pallina che rimbalza sullo schermo.



Il primo passo è realizzare la base del gioco sulla quale costruiremo tutto, a cominciare dalla pallina che si muove e rimbalza contro le pareti. Per portarci avanti, faremo in modo che la pallina rimbalzi effettivamente sul "campo di gioco", quindi dovrà restare all'interno della parte inferiore, in verde, visto che la fascia in alto grigia è dedicata al punteggio (e un po' anche all'estetica…).

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import pygame
```

Queste prime righe servono più all'interprete che a voi, comunque quella veramente fondamentale è la terza, che indica a Python di importare la libreria PyGame

Di seguito il codice per la pallina. È una classe, ovvero una struttura dati che contiene anche le funzioni per la loro manipolazione. Nella programmazione orientata agli oggetti le classi sono molto più di questo, ma per ora possiamo anche vederle solo così, come modelli per realizzare variabili molto complesse e modellate sui nostri scopi (gli oggetti, appunto):

```
class GameBall:

····def __init__(self,img):
········self.speed = [0,0]
········self.bitmap = pygame.image.load(img)
········self.bitmap.set_colorkey((0,0,0))

····def reset(self):
········self.x = 11
········self.y = 43
········self.speed = [0,0]

····def start(self):
········self.speed = [1,1]
```

La prima funzione (__init__()) viene richiamata appena si crea un nuovo oggetto appartenente alla classe GameBall, ovvero la nostra pallina. Creiamo le variabili interne all'oggetto impostandole a un valore prefissato e decidiamo l'immagine che rappresenterà la pallina sullo schermo.
Le funzioni interne all'oggetto sono dette più precisamente metodi, e si riconoscono perché nel codice esterno alla classe sono identificate dal formato <oggetto>.<metodo()>, mentre all'interno sono richiamate nella forma self.<metodo()>. D'ora in avanti le chiameremo metodi anche nel testo.

reset() è un metodo richiamato per inizializzare la pallina all'inizio del gioco e ogni volta che finisce al di là della racchetta di un giocatore. Più avanti vedremo come, quando introdurremo anche le racchette. start() invece imposta la velocità iniziale.

Alcuni metodi necessari per il movimento della pallina:

```
····def addX(self,val):
········new_x = self.x + val
········if new_x < 640 and new_x > 0:
············self.x += val
········else:
············if new_x < 0:
················diff = abs(new_x)
················self.x = diff
············else:
················diff = new_x - 629
················self.x = 629 - diff
············self.setSpeedX(-self.speedX())

····def addY(self,val):
········new_y = self.y + val
········if new_y <= 466 and new_y >= 42:
············self.y += val
········else:
············if new_y < 42:
················diff = 42 - new_y
················self.y = 42 + diff
············else:
················diff = new_y - 466
················self.y = 466 - diff
············self.setSpeedY(-self.speedY())

····def makeStep(self):
········self.addX(self.speed[0])
········self.addY(self.speed[1])
```

I metodi AddX() e AddY() impostano le nuove posizioni sullo schermo, rispettivamente x e y, a partire dalla posizione precedente e leggendo la velocità impostata per il rispettivo asse. I numeri utilizzati stabiliscono i limiti oltre i quali la pallina non deve andare, corrispondenti alle posizioni estreme del campo di gioco *calcolate tenendo conto dei bordi e delle dimensioni dell'immagine della pallina stessa*. Ecco perché, anche se abbiamo a disposizione il limite inferiore a 480, dobbiamo limitarci a 466. 4 pixel sono la linea laterale e (considerando che la posizione di ogni oggetto è calcolata a partire dall'angolo superiore sinistro della sua immagine) 11 pixel sono la dimensione in altezza dell'immagine, uguale alla dimensione in larghezza – e tolgo un pixel per sicurezza, servirà per accertarci più avanti che la racchetta ha effettivamente mancato la pallina.

Altri metodi per gestire la velocità e la visualizzazione a schermo aggiornata:

```
····def setSpeedX(self,val):
········if abs(val) < 3:
············self.speed[0] = val

····def speedX(self):
········return self.speed[0]
```

```
····def setSpeedY(self,val):
········if abs(val) < 3:
············self.speed[1] = val

····def speedY(self):
········return self.speed[1]

····def getX(self):
········return self.x

····def getY(self):
········return self.y

····def render(self,screen):
········self.makeStep()
········screen.blit(self.bitmap,(self.x, self.y))
```

I metodi speedX()/speedY() restituiscono rispettivamente la velocità orizzontale e verticale. Notate che possono essere negativi: in quel caso la pallina si muove rispettivamente verso l'alto e verso sinistra. Adesso, rileggendo i metodi AddX() e AddY() dovrebbe essere anche più chiaro quel che succede quando la pallina arriva a un bordo del campo di gioco: se l'incremento della posizione va oltre il limite le funzioni calcolano l'avanzo e arretrano di quel risultato la posizione della pallina, invertendo il segno della rispettiva velocità. Il risultato è un perfetto urto elastico contro la parete rappresentata dal bordo del campo. getX() e getY() riportano rispettivamente la posizione orizzontale e verticale al momento della loro chiamata. Si potrebbe anche leggere direttamente il valore della variabile interna connessa, detta attributo nell'ambito della programmazione a oggetti anche da fuori, ma pochi altri linguaggi sono così liberali come il Python nell'accesso dall'esterno alle variabili interne di un oggetto. Quindi è bene abituarsi a raggiungere gli attributi dall'esterno attraverso metodi dedicati.
Infine render() esegue l'effettivo movimento della pallina e la disegna a schermo con la funzione blit() di PyGame. O meglio, prepara il disegno della pallina, perché il disegno effettivo avverrà solo al momento della chiamata pygame.display.upgrade(), nella funzione principale, che per convenzione è main().

A proposito, passiamo al codice di quest'ultima:

```
def main(args):

····pygame.init()
····screen = pygame.display.set_mode((640,480))
····pygame.key.set_repeat(1,1)
····pygame.display.set_caption("python pong")
····backdrop = pygame.image.load('img/pong_a_2.bmp')

····ball = GameBall("img/ball_base.png")
····ball.reset()
····ball.start()
```

Le prime righe del codice inizializzano l'ambiente, impostando la dimensione della finestra di gioco (anche se qui la funzione usa un oggetto chiamato screen – volendo, e con il giusto monitor, PyGame potrebbe aprire il gioco a schermo intero…), il modo in cui

considerare la frequenza di ripetizione dei tasti se premuti a lungo, il titolo della finestra e lo sfondo del gioco.
Inoltre viene creata la nostra pallina rimbalzante tramite la sua immagine .png, posta in posizione di partenza e avviata.

Il codice che segue imposta finalmente il copione del nostro “film”, descrivendo cosa deve succedere attimo per attimo. Tutti i videogiochi interattivi si basano su quello che viene definito “il ciclo infinito”, ovvero una sequenza di istruzioni da eseguire ed eventi da gestire attimo per attimo, fino a che non succede qualcosa.
Nel nostro caso, la sequenza prevede che la pallina continui la sua corsa rimbalzando quando arriva a uno dei confini del campo di gioco, fino a che qualcuno non prema il tasto “ESC” (in alto a sinistra sulla tastiera).
Ogni passo del ciclo prevede la parte relativa alla gestione degli eventi, la parte relativa al movimento degli elementi del gioco e infine la parte di realizzazione del fotogramma a video.

```
····quit = 0

····while quit == 0:
········screen.blit(backdrop,(0,0))
```

Attenzione: se il gioco che realizziamo assomiglia a un film, occorre ricreare ogni fotogramma a partire dallo sfondo; in questo caso è proprio l’immagine che avevamo impostato come oggetto backdrop prima dell’avvio del ciclo – e che resterà la stessa dall’inizio alla fine.

```
········for ourevent in pygame.event.get():

············if ourevent.type == pygame.QUIT:
················quit = 1

············if ourevent.type == pygame.KEYDOWN:
················if ourevent.key == pygame.K_ESCAPE:
····················quit = 1
```

La gestione degli eventi è realizzata in questo ciclo for. Il metodo get() dell’oggetto event predefinito in PyGame ci restituisce l’insieme degli eventi che si verificano nel tempo necessario alla realizzazione del passo. Anche gli eventi sono oggetti, e se si esamina il loro attributo type possiamo capire a che famiglia appartengono. I vari tipi sono scritti in maiuscolo perché sono dei valori predefiniti costanti.
QUIT si capisce a cosa si riferisce: è l’evento che viene sollecitato quando viene richiesta la fine del ciclo. KEYDOWN invece si verifica quando qualcuno preme un pulsante sulla tastiera; noi per ora siamo interessati al solo tasto ‘ESC’, perché premendo questo faremo in modo da impostare la variabile quit a 1, e questo farà concludere il ciclo.
N.B.: il fatto che quit sia impostata a 1 anche nel caso di un evento QUIT garantisce che, ad esempio, il programma si chiuda anche quando decidiamo di fare click sul pulsante di chiusura della finestra. Ok, il programma si chiuderà comunque, ma in questo caso possiamo “intercettare” l’evento e fare qualcosa prima di una interruzione improvvisa.

```
········if ball.getX() <= 1:
············ball.setSpeedX(-ball.speedX())
```

```
        if ball.getX() >= 629:
            ball.setSpeedX(-ball.speedX())
```

La seconda parte del codice esegue le azioni necessarie a completare il passo, siano o no dipendenti dalla gestione degli eventi appena eseguita.
In questo caso facciamo solo un controllo sulla posizione della palla in relazione ai limiti sinistro e destro del campo. I rimbalzi verticali sono già gestiti dall'oggetto palla, ma questi devono restare esterni perché nell'evoluzione del nostro progetto saranno influenzati o meno dalle racchette dei giocatori.
Notate che nel caso del rimbalzo a destra il controllo della posizione non è sul valore 640, limite destro della nostra finestra. Questo perché la posizione della pallina è rilevata sempre dall'angolo superiore sinistro dell'immagine (che misura 11x11 pixel). Se controllassimo la posizione a 640 vedremmo la pallina uscire dalla finestra quasi completamente prima di rimbalzare indietro, ma noi vogliamo un rimbalzo appena la pallina tocca il bordo.

```
        ball.render(screen)
        pygame.display.update()
        pygame.time.delay(2)
```

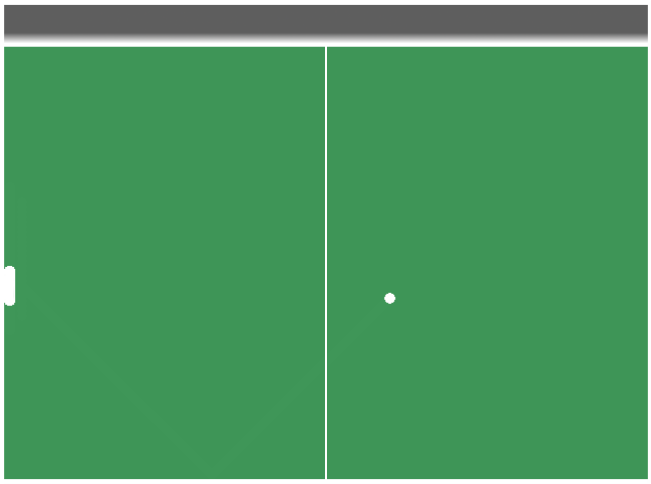
Alla fine del ciclo richiamiamo le funzioni di ridisegno della pallina nella sua nuova posizione, ridisegnamo lo schermo e aspettiamo un tempo limitato (2 millisecondi) per garantire una velocità del gioco "umana".
E infine chiudiamo la funzione.

```
    return 0
```

Ora il codice che garantisce l'esecuzione della funzione main() al lancio del programma:

```
if __name__ == '__main__':
    import sys
    sys.exit(main(sys.argv))
```

Passo 2: La prima racchetta



Una pallina che rimbalza all'infinito può dare soddisfazione sapendo di averla fatta rimbalzare noi, ma alla lunga annoia. Cominciamo a introdurre un giocatore e a rilevare quando riesce a raggiungere la pallina e quando la manca.
Suggerimento: dato che il codice riporterà d'ora in poi solo le modifiche, conviene salvare il programma realizzato finora con un nome (es. step_1.py) e riportare le modifiche su una sua copia con un nome diverso, es. step_2.py. Le nuove righe saranno colorate in modo diverso

Cominciamo dall'oggetto GameBall, le modifiche sono concentrate nel metodo reset():

```
    ....
····def reset(self,player):
········self.x = 12
········self.y = player.getY()+15
········self.speed = [0,0]
    ....
```

Cambia la dichiarazione della funzione con l'introduzione di un parametro. Questo serve perché, a differenza del precedente codice, stavolta la pallina sarà lanciata dalla racchetta. La posizione X di partenza non cambia di molto (giusto per non vedere la pallina schiacciata sulla racchetta), la posizione Y invece dipende dalla posizione della racchetta letto dal suo metodo getY() - che come vedremo è parente stretto dell'omonimo metodo dell'oggetto pallina.

Vediamo la definizione della classe per la racchetta, la parte più consistente delle modifiche:

```
class Paddle:

····def __init__(self,xpos,ypos,img,pl_id):
········self.x = xpos
········self.y = ypos
········self.oldy = ypos
········self.bitmap = pygame.image.load(img)
········self.bitmap.set_colorkey((0,0,0))
········self.pl_id = pl_id

····def getId(self):
········return self.pl_id
    ....
```

Il metodo costruttore è simile al metodo per la classe della pallina, ma gli argomenti sono vari: in definizione si impostano posizione verticale e orizzontale (perché deve essere valida per più di un giocatore), l'immagine della racchetta e un valore per distinguere

l'oggetto racchetta quando nel programma se ne creano più di uno. Di conseguenza c'è anche il metodo accessorio per leggerne il valore.

```
    ....
    def getY(self):
        return self.y

    def addY(self,val):
        if (self.y + val) < 435 and (self.y + val) > 43:
            self.oldy = self.y
            self.y += val

    def dirY(self):
        return self.y-self.oldy

    def getX(self):
        return self.x
    ....
```

Questi metodi servono per stabilire posizione e velocità della racchetta quando viene mossa durante il gioco. In particolare i metodi dirX() e dirY() sfruttano la posizione precedente della racchetta per stabilire la direzione del movimento. Certo, in questo caso non c'è un metodo che aggiorni la posizione orizzontale e tenga conto della vecchia posizione, perché le racchette si muovono solo in verticale (però, se qualcuno volesse cambiare le regole…).

```
    ....
    def diffPos(self,ball):
        diff_posx = abs(ball.getX() - self.x)
        diff_posy = ball.getY() - self.y #N.B.: qui il segno è importante...
        return (diff_posx,diff_posy)

    def render(self,screen):
        screen.blit(self.bitmap,(self.x, self.y))
```

Il penultimo metodo ci serve per stabilire la differenza tra la posizione della racchetta e la posizione della pallina. Ci servirà quando dovremo decidere se la racchetta l'ha respinta o le è sfuggita.

Passiamo alla funzione principale:

```
    ....
    pygame.display.set_caption("python pong")
    backdrop = pygame.image.load('img/pong_a_2.bmp')

    player1 = Paddle(0,292,"img/paddle_vert.png",0)

    ball = GameBall("img/ball_base.png")
    ball.reset(player1)

    quit = 0
    gamestate = 0

    while quit == 0:
    ....
```

Intanto generiamo la racchetta impostando le coordinate più o meno al cento del campo di gioco (quindi poco sotto la metà della finestra, considerando lo spazio lasciato libero in alto), assegnando l'immagine e l'id per il giocatore. Siccome l'Informatica conta a partire da 0, questo sarà l'ID per la racchetta (ma vedremo nel codice finale che il motivo è leggermente diverso).
Un altro elemento è la variabile gamestate: ci servirà per stabilire in che situazione si trova il gioco (0: palla pronta sulla racchetta prima dell'inizio – 1: palla in movimento e gioco in corso).

```
    ....
            if ourevent.type == pygame.QUIT:
                quit = 1

            if ourevent.type == pygame.KEYDOWN:
                if ourevent.key == pygame.K_z:
                    player1.addY(5)
                    if gamestate == 0:
                        if ball.getY() < 445:
                            ball.addY(5)
                if ourevent.key == pygame.K_a:
                    player1.addY(-5)
                    if gamestate == 0:
                        if ball.getY() > 63:
                            ball.addY(-5)
                if ourevent.key == pygame.K_s:
                    if gamestate == 0:
                        ball.start()
                        gamestate = 1

            if ourevent.key == pygame.K_ESCAPE:
    ....
```

Nella parte relativa alla gestione degli eventi, sono aggiunti i controlli per la racchetta. Qui serve il controllo dello stato del gioco: se gamestate è azzerato il gioco è fermo e la pallina segue la racchetta nei suoi movimenti.
I tasti usati per il movimento sono 'a' e 'z', e per lanciare la pallina 's'.

Ovviamente va cambiato anche il codice che si occupa del lato sinistro del campo: stavolta c'è la racchetta, non un muro

```
    ....
                if ourevent.key == pygame.K_ESCAPE:
                    #TODO: qualcosa di più "mosso"
                    quit = 1

        if ball.getX() <= 9:
            diff_pl1_x,diff_pl1_y = player1.diffPos(ball)
            if diff_pl1_x < 10 and (diff_pl1_y < 40 and diff_pl1_y > -10):
                ball.setSpeedX(-ball.speedX())
                if (ball.speedY() * player1.dirY()) < 0: # spostamenti discordi
                    ball.setSpeedY(-ball.speedY())
            else:
                # palla persa da player1
                gamestate = 0
```

```
                ball.reset(player1)

        if ball.getX() >= 629:
            ball.setSpeedX(-ball.speedX())

        player1.render(screen)
        ball.render(screen)
    ....
```

La posizione della pallina stavolta è intercettata quando la sua ascissa arriva a 10 (perché la racchetta è larga 11 punti). Se le posizioni verticali tra racchetta e pallina sono vicine – ricordando che anche in questo caso la posizione della racchetta è quella del suo angolo superiore sinistro – la racchetta respinge la pallina, altrimenti la pallina è persa.

Passo 3: Completiamo il gioco

Cosa manca? In pratica il secondo giocatore con i suoi controlli e un meccanismo che mantenga il punteggio.

```
import pygame
import math # per il turno di battuta
    ....
```

La lista dei moduli inclusi si allunga per le funzioni matematiche. Ce ne servirà una in particolare, per il turno di battuta – ma potrebbero servircene altre per ampliare le funzionalità del gioco.

Anche la classe GameBall presenta dei cambiamenti:

```
class GameBall:

    def __init__(self,img):
        self.speed = [0,0]
        self.gamenum = 0
        self.bitmap = pygame.image.load(img)
        self.bitmap.set_colorkey((0,0,0))

    def reset(self,player):
        if player.getX() < 320:
            self.x = 12
        else:
            self.x = 618
        self.y = player.getY()+15
        self.speed = [0,0]
    ....
```

Il costruttore introduce un nuovo attributo, gamenum, che conta il numero di scambi giocati per decidere il turno di battuta. Il metodo reset(), invece, è modificato per posizionare bene la pallina quando a servire è il giocatore a destra dello schermo.

```
    ....
    def render(self,screen):
        self.makeStep()
        screen.blit(self.bitmap,(self.x, self.y))

    def getServicePl1(self):
        if self.gamenum == 0:
            return True
        else:
            if math.ceil((self.gamenum+1)/2) % 2:
                return False
            else:
                return True
```

Viene introdotto il nuovo metodo getServicePl1(), che decide se è il giocatore a sinistra a servire o no (rispondendo True o False di conseguenza). La formula per decidere si basa sul numero dello scambio, e segue la stessa regola del turno di battuta nei Tie-Break del vero tennis. Il primo servizio spetta al giocatore 1, a sinistra, dopodiché il giocatore 2, a

destra, avrà diritto a servire per due volte, quindi toccheranno di nuovo due servizi consecutivi al giocatore a sinistra, e così via.
(Ok, si potevano semplicemente alternare i servizi 1 alla volta, ma in questo caso si voleva introdurre un esempio di regola non banale)

La classe Paddle resta invariata, in quanto i suoi metodi vanno altrettanto bene sia per la racchetta di sinistra che per quella di destra. Invece, per la comunicazione del punteggio a video, introduciamo una nuova classe:

```
class Score:

    def __init__(self,goal):
        self.pnt = [0,0]
        self.goal = goal
        self.font = pygame.font.Font('img/slkscr.ttf',20)

    def displayScore(self,screen):
        textscore = str(' - ').join([str(s) for s in self.pnt])
        score_text = self.font.render(textscore,True,(255,255,255))
        score_rect = score_text.get_rect()
        score_rect.centerx = 320
        score_rect.y = 5
        screen.blit(score_text,score_rect)
        pygame.display.update(score_rect)
    ....
```

La classe Score prevede un array per il punteggio dei due giocatori, una variabile dove conservare il valore da raggiungere nel punteggio per vincere la partita, e l'oggetto font (predefinito da PyGame) con il quale scrivere a video i caratteri del punteggio; il costruttore prevede come parametro proprio il valore obbiettivo da raggiungere.

```
    ....
    def setScore(self,player):
        return_state = 0
        self.pnt[player.getId()] += 1

        if self.pnt[player.getId()] >= self.goal:
            return_state = 2

        return return_state

    def displayWinner(self,screen):
        text_win = ""
        if self.pnt[0] == self.goal:
            text_win = "Vince la partita Player 1"
        elif self.pnt[1] == self.goal:
            text_win = "Vince la partita Player 2"

        if text_win != "":
            winner_text = self.font.render(text_win,True,(255,255,255))
            winner_rect = winner_text.get_rect()
            winner_rect.centerx = 320
            winner_rect.y = 200
            screen.blit(winner_text,winner_rect)
            pygame.display.update(winner_rect)
```

Il metodo setScore aumenta di 1 il punteggio del giocatore passato come argomento (come oggetto paddle), e restituisce come valore di uscita il valore 2 se il giocatore ha raggiunto l'obbiettivo per vincere la partita. Perché proprio 2? Come vedremo, avrà a che fare con la variabile che descrive lo stato del gioco.

Passiamo alla funzione principale, qui vedremo che le modifiche saranno in quantità, a seguito dell'introduzione della seconda racchetta, della visualizzazione del punteggio e della gestione del fine partita – come avrete notato, anche il secondo passo del progetto era sostanzialmente infinito, ogni volta che la racchetta non riusciva a raggiungere la pallina il gioco si resettava all'inizio. Qui non sarà la stessa cosa:

```
    ....
····player1 = Paddle(0,292,"img/paddle_vert.png",0)
····player2 = Paddle(628,292,"img/paddle_vert.png",1)

····ball = GameBall("img/ball_base.png")
····ball.reset(player1) # N.B.: batte player1 per il primo game
····score = Score(2)
····score.displayScore(screen)

····quit = 0
····gamestate = 0
    ....
```

Come prevedibile, la seconda racchetta ha una chiamata al metodo costruttore analoga alla prima: cambia la coordinata x per la posizione (alla destra dello schermo) e l'ID del giocatore.
Attenzione: il giocatore 2 (destra) ha ID 1, e il giocatore 1 (sinistra) ha ID 0. Come mai? Ricordate l'array usato per mantenere il punteggio nella classe Score? In Python, come in tutti i linguaggi di programmazione, ogni array o lista di elementi numerata inizia sempre con lo 0 (si dice che gli informatici sono gli unici che cominciano a contare da 0, a differenza di tutti gli altri che cominciano da 1…).
Se vogliamo riferirci al punteggio del giocatore di sinistra occorre valutare il valore della posizione 0 nell'attributo pnt di score, mentre se vogliamo conoscere il valore raggiunto dal giocatore a destra occorre leggere la posizione 1 nello stesso attributo.

```
    ....
············if ourevent.type == pygame.KEYDOWN:
················if ourevent.key == pygame.K_z and gamestate < 2:
····················player1.addY(5)
····················if gamestate == 0 and ball.getServicePl1():
························if ball.getY() < 445:
····························ball.addY(5)
················if ourevent.key == pygame.K_a and gamestate < 2:
····················player1.addY(-5)
····················if gamestate == 0 and ball.getServicePl1():
························if ball.getY() > 63:
····························ball.addY(-5)
················if ourevent.key == pygame.K_s and gamestate < 2:
····················if gamestate == 0 and ball.getServicePl1():
························ball.start()
························gamestate = 1
    ....
```

Con le modifiche ai controlli per la racchetta di sinistra ci accorgiamo del cambio di significato per la variabile che decide lo stato della partita. Adesso abbiamo 3 possibili valori:
0: Il gioco è fermo, e il giocatore di turno alla battuta può decidere da dove mettere in gioco la pallina.
1: Il gioco è in svolgimento, la pallina si muove sul campo di gioco e i giocatori devono evitare che finisca fuori dalla loro parte.
2: Il gioco è finito, uno dei due giocatori ha raggiunto il punteggio obbiettivo. In questo caso, non ha più senso muovere la racchetta.

```
    ....
················if ourevent.key == pygame.K_l and gamestate < 2:
····················player2.addY(5)
····················if gamestate == 0 and not(ball.getServicePl1()):
························if ball.getY() < 445:
····························ball.addY(5)
················if ourevent.key == pygame.K_p and gamestate < 2:
····················player2.addY(-5)
····················if gamestate == 0 and not(ball.getServicePl1()):
························if ball.getY() > 63:
····························ball.addY(-5)
················if ourevent.key == pygame.K_o and gamestate < 2:
····················if gamestate == 0 and not(ball.getServicePl1()):
························ball.start()
························gamestate = 1
    ....
```

Introduciamo i controlli anche per il secondo giocatore, il codice è quasi identico ma ovviamente sono diversi i pulsanti: ‘p’ e ‘l’ per muoversi, ‘o’ per servire la pallina quando si è di turno.

```
    ....
····if ball.getX() <= 9:
····    diff_pl1_x,diff_pl1_y = player1.diffPos(ball)
····    if diff_pl1_x < 10 and (diff_pl1_y < 40 and diff_pl1_y > -10):
················ball.setSpeedX(-ball.speedX())
················if (ball.speedY() * player1.dirY()) < 0:
····················#i due spostamenti sono discordi
····················ball.setSpeedY(-ball.speedY())
····    else:
················# palla persa da player1
················gamestate = score.setScore(player2)
················if gamestate < 2:
····················pygame.time.delay(500)
····················if ball.getServicePl1():
························ball.reset(player1)
····················else:
························ball.reset(player2)
················else:
····················ball.reset(player2)
    ....
```

Coerentemente con le modifiche del gioco, cambia anche il codice per la gestione della pallina persa dal giocatore a sinistra, che fino a ora era l’unico presente nel gioco: viene

marcato un punto per il giocatore a destra, esaminato lo stato del gioco conseguente alla marcatura e, se la partita non è finita, assegnato il servizio a che è di turno.

```
    ....
        elif ball.getX() >= 619:
            diff_pl2_x,diff_pl2_y = player2.diffPos(ball)
            if diff_pl2_x < 10 and (diff_pl2_y < 40 and diff_pl2_y > -10):
                ball.setSpeedX(-ball.speedX())
                if (ball.speedY() * player2.dirY()) < 0:
                    #i due spostamenti sono discordi
                    ball.setSpeedY(-ball.speedY())
            else:
                # palla persa da player2
                gamestate = score.setScore(player1)
                if gamestate < 2:
                    pygame.time.delay(500)
                    if ball.getServicePl1():
                        ball.reset(player1)
                    else:
                        ball.reset(player2)
                else:
                    ball.reset(player1)
    ....
```

Ora però c'è anche il giocatore a destra, quindi occorre sostituire il codice che prevede il semplice rimbalzo con gli stessi controlli che abbiamo impostato prima per il giocatore di sinistra. Potremmo anche fare una funzione per la gestione della pallina "fuori campo", perché le istruzioni sono esattamente le stesse per entrambi i giocatori: cambia il lato (quindi l'ascissa raggiunta dalla pallina) e le conseguenze del colpo mancato (nel caso del giocatore a destra il punto va al suo avversario a sinistra, esattamente il contrario di quel che abbiamo scritto poche righe sopra). Se volete provare a farla da soli, è un ottimo esercizio.

Arriviamo così alle modifiche per la generazione del fotogramma: in più adesso dobbiamo visualizzare il secondo giocatore e mostrare il punteggio con il metodo displayScore() dell'oggetto score.

```
    ....
        player1.render(screen)
        player2.render(screen)
        ball.render(screen)

        score.displayScore(screen)
        if gamestate == 2:
            winplayer = score.displayWinner(screen)

        pygame.display.update()
        pygame.time.delay(2)

    return 0
```

Attenzione però, se il gioco è terminato dobbiamo anche darne notizia con l'altro metodo displayWinner(); naturalmente

l’oggetto sa da solo chi ha vinto perché abbiamo inserito il punteggio proprio tra i suoi attributi, dove ci faceva più comodo.

E con questo abbiamo finito. Se adesso fate partire il programma (e non avete fatto errori), potrete divertirvi a fare delle sfide uno contro l’altro, su qualunque sistema dove siano presenti Python e la libreria PyGame – oltre naturalmente a uno schermo e una tastiera...

Step 3: alcuni suggerimenti per migliorare il progetto

Arrivati a questo punto il gioco è completo, ma ci sono varie possibilità per migliorarlo ulteriormente, a patto di sapere dove intervenire con nuova programmazione.
Di seguito ci sono alcuni suggerimenti per personalizzare e rendere il gioco ancora più divertente:

Proposta n.1: Modificare l'angolo di rimbalzo della pallina sulla racchetta.
Vi sarete accorti che la pallina percorre l'area di gioco sempre in diagonale con lo stesso angolo di 45°. L'unico modo per farle cambiare direzione è colpirla con la racchetta in movimento contrario. In questo caso la pallina invece di rimbalzare nella stessa direzione verticale tornerà indietro. Ma questo è un po' poco per impostare strategie di gioco vincenti, no?
Se guardate attentamente il codice relativo al rilevamento dell'intercettazione della palla per ciascuno dei due giocatori, noterete che l'unico intervento sul moto della pallina è il cambio di segno delle componenti della sua velocità: solo la x nel caso la racchetta sia ferma o si muova nella stessa direzione della pallina, anche la y nel caso i rispettivi movimenti siano contrari l'uno all'altro.
Si potrebbe rimpiazzare questo gruppo di istruzioni con una funzione, o meglio ancora un metodo della classe Paddle, che esamina il punto della racchetta su cui la pallina è rimbalzata e varia l'angolo di conseguenza: per far questo avrete però bisogno di qualche conoscenza di fisica per l'implementazione delle giuste formule di modifica delle componenti orizzontale e verticale della velocità per la pallina.

Nota per i più curiosi/appassionati di matematica:
In realtà, esiste un metodo più semplice per impostare il cambio dell'angolo: si basa sull'implementazione in Python della matematica per i cosiddetti numeri complessi. Che questi siano legati alle radici quadrate e in genere pari di numeri negativi non vi deve interessare – a meno che non facciate corsi di matematica alle scuole superiori e all'Università (Teorema fondamentale dell'Algebra e sue conseguenze, ecc.). Quel che è interessante è che questi numeri sono rappresentati su un piano cartesiano perché sono la generalizzazione dei numeri reali normalmente usati, e rappresentati di solito su una retta. Questi numeri per motivi matematici possono essere identificati sia con le loro componenti x (parte reale) e y (parte immaginaria) sia con la distanza dal centro del riferimento cartesiano ("modulo", per i più curiosi) accoppiata all'angolo misurato tra l'asse delle x e il segmento che congiunge l'origine del riferimento cartesiano con la posizione del numero complesso.
Possiamo sfruttare le funzioni Python per la conversione tra le due rappresentazioni: sono funzioni presenti nel modulo math. Creiamo un numero complesso con le componenti x e y della velocità della pallina, convertiamo il numero nella rappresentazione modulo/angolo, cambiamo l'angolo secondo le regole che abbiamo stabilito per il rimbalzo sulla racchetta e riconvertiamo il risultato nella rappresentazione x,y per ottenere le componenti della velocità adatte al gioco. Semplice, no?

Proposta n.2: Impostare una velocità progressiva.
Se i due giocatori sono bravi, può capitare che lo scambio tra i due diventi molto lungo, soprattutto perché i rimbalzi, come si diceva alla proposta precedente, hanno un angolo prefissato.
Per mettere un po' di pepe allora conviene ricorrere a una variante presente anche nel gioco originale di 40 anni fa: aumentare progressivamente la velocità della pallina.

Non è necessario che le componenti della velocità siano necessariamente numeri interi: le posizioni della pallina saranno comunque arrotondate all'intero più vicino e comunque la velocità del gioco non permetterà di notare piccolissime oscillazioni nella posizione (soprattutto se aprite la finestra 640x480 su uno schermo FullHD…).
Si può cominciare con una velocità leggermente inferiore all'attuale, diciamo 1.5 per ciascuna componente, per poi aumentare di 0.2 ogni 20 scambi. Per fare questo basta aggiungere un nuovo attributo all'oggetto Gameball, incrementarlo ogni volta che una racchetta respinge la palla e far scattare l'aumento quando il totale dei rimbalzi è divisibile per 20, ad esempio.
Solo un'accortezza: meglio stabilire un massimo perché ricordate che le racchette continueranno a muoversi alla stessa velocità. All'inizio sarà facile raggiungere la pallina, ma quando la velocità raggiungerà certi valori sarà quasi impossibile essere nel posto giusto al momento giusto...

Proposta n.3: Usare racchette personalizzate (e/o pallina diversa, e/o campo diverso...)
Questa è solo una modifica estetica, ma è abbastanza facile. Siete stanchi del solito bastoncino bianco, e volete un tocco di colore? Basta cambiare l'immagine caricata al momento della creazione dell'oggetto.
Potrebbe essere divertente giocare con una racchetta rossa e una verde, oppure con forme differenti. Ma potete anche sostituire l'immagine della pallina attuale con qualcosa di diverso; magari un cuore per i romantici o un teschio per gli...alternativi.
Stesso discorso per il campo da gioco. Non vi piace quello attuale? E allora ridisegnatelo come volete.
Un'avvertenza, però: qualunque sia l'immagine che decidete di sostituire, dovete comunque rispettare le dimensioni originali. Quindi la vostra nuova pallina dovrà comunque essere 11x11 pixel, e ogni racchetta dovrà misurare 11x40 pixel. Anche il campo dovrà comunque prevedere l'area di gioco più in basso di 40 pixel per garantire lo spazio riservato al punteggio. Questo per via dei valori scritti nel codice python del gioco.
Potete anche provare a fare i furbi e implementare una racchetta più lunga, ma non funzionerà finché non cambierete il codice: la parte valida per far rimbalzare la palla sarà sempre limitata a 40 pixel verticali e 11 pixel orizzontali – i più vicini al bordo dello schermo, peraltro.

Nota per i più smanettoni
In realtà, esistono funzioni in PyGame che permettono di leggere l'effettiva dimensione in pixel di ogni immagine caricata, quindi con un po' di lavoro si potrebbe sostituire i numeri fissi usati per definire tutti i rimbalzi e cambiarli con valori derivati dalle dimensioni effettive delle immagini usate.
Questo potrebbe portare ad implementare una fase introduttiva del gioco nella quale ogni giocatore può scegliere la racchetta preferita. Certo, sorge il problema delle dimensioni differenti, ma questo in realtà permetterebbe a chi è meno bravo di giocare con qualche speranza in più di battere un avversario particolarmente abile. Oppure si potrebbe impostare una velocità di spostamento per le racchette differente in base alla loro dimensione, per riequilibrare un po' le possibilità.

Proposta n.4: Passare dal Tennis allo Squash
Per questa variante l'idea delle racchette differenziate potrebbe essere in effetti molto utile. Chi ha detto che i giocatori devono stare ai lati opposti del gioco? Riutilizzando il concetto del muro laterale a destra, possiamo piazzare anche il giocatore numero due a sinistra, e alternare l'obbligo di respingere la pallina in base al numero di rimbalzi sul muro. Volendo,

si può aggiungere anche un terzo giocatore, e cambiare la modalità di punteggio partendo da un certo numero di “vite” e stabilendo la fine del gioco appena uno dei tre giocatori arriva a 0. Qui le modifiche necessarie cominciano a essere molte, ma se ci si pensa bene si possono ancora gestire col codice a disposizione: il punteggio diventa un lista di 3 numeri, a cui si sottrae 1 per marcare un punto anziché aggiungere; il metodo che stabilisce il servizio risponde 1,2,3 invece che True o False per indicare chi deve battere – e la battuta è rigidamente a turni, dal giocatore 1 al giocatore 2 al 3, e così via; l’oggetto pallina ha un attributo ulteriore per stabilire chi deve respingere; questo è incrementato di 1 ad ogni rimbalzo a destra, cosicché il calcolo del resto della sua divisione per tre decide il giocatore...

Proposta n.5: Ricreare ‘Breakout’
Per quanto lo step 2 permetta di giocare anche da soli per allenarsi, la cosa può diventare noiosa senza incentivo. Allora si può ricostruire il gioco Breakout, in pratica il primo gioco derivato dal Pong originale. Al posto di un avversario in carne e ossa, una fila di mattoncini da abbattere per ottenere punti in base al loro colore.
Ogni mattoncino può essere realizzato come un altro oggetto, la sua classe può essere scritta a partire da parte del codice usato per pallina e racchetta, con la ovvia differenza che stavolta il nuovo oggetto non si muove.
L’urto della pallina con un mattoncino è identico all’urto contro una parete, con la differenza che il mattone sparisce e il punteggio è incrementato del valore del mattone. Attenzione che in questo caso occorre prevedere un urto su tutti e 4 i lati del mattone, e in questo caso diventa obbligatorio rendere dinamici i valori per le “coordinate d’urto” ipotizzate per la proposta precedente delle racchette custom: mica vorrete tenere i conti con tutte le coordinate di tutti i mattoni e per ciascuno di essi scrivere un codice separato?

Proposta n.6: Giocare in quattro
Come nel tennis esiste il doppio, anche in questo caso si può prevedere l’aggiunta di altri due giocatori con due racchette a metà strada ciascuna tra il bordo presidiato dal proprio compagno di squadra e la linea di metà schermo. In questo caso, se la pallina oltrepassa la posizione orizzontale di un giocatore in mezzo al campo non succede niente, il controllo per il punto da assegnare resta ai bordi del campo. Però potrebbe essere divertente ipotizzare che succede quando una risposta di un giocatore a bordo campo colpisce per sbaglio la racchetta del suo compagno. In questo caso occorre prevedere per le racchette centrali l’urto con la pallina su entrambi i lati (e occhio: il lato sinistro respinge la pallina quando arriva da sinistra, e viceversa).
Unica possibile controindicazione: la tastiera potrebbe diventare un po’ troppo affollata con quattro mani contemporaneamente. Tuttavia PyGame può gestire anche periferiche aggiuntive, come joystick, joypad e mouse/trackpad; basta dare un’occhiata alla documentazione.

Proposta n.7: Muovere le racchette anche in orizzontale
Come accennato alla proposta precedente, si può pensare anche metodi di controllo alternativi alla tastiera, e così potrebbe venire l’idea di provare a muovere le racchette anche in orizzontale oltre che in verticale. In questo caso il codice da modificare riguarda non solo la gestione degli eventi per il controllo delle racchette, ma anche il codice per rilevare l’urto con la pallina, perché a questo punto non è più detto che avvenga vicino al bordo del campo. D’altra parte, il punto potrà essere assegnato solo quando la pallina sarà effettivamente arrivata laggiù, il giocatore potrebbe tornare indietro in tempo per la respinta – sempre che la pallina non sia troppo veloce, vedi la proposta n.2.