



UNIVERSITÀ
DI PARMA

animazione attori informatica e laboratorio di programmazione



```
class Actor():
    '''Interfaccia che deve essere implementata dai vari tipi di personaggi del gioco'''
    def move(self):
        '''Chiamato da Arena, ad ogni turno del personaggio'''
        raise NotImplementedError('Abstract method')

    def collide(self, other: 'Actor'):
        '''Chiamato da Arena, quando(self) è in collisione con un altro personaggio (other)'''
        raise NotImplementedError('Abstract method')

    def position(self) -> (int, int, int, int):
        '''Restituisce il rettangolo che contiene il personaggio
        tupla di 4 valori interi: (left, top, width, height)'''
        raise NotImplementedError('Abstract method')

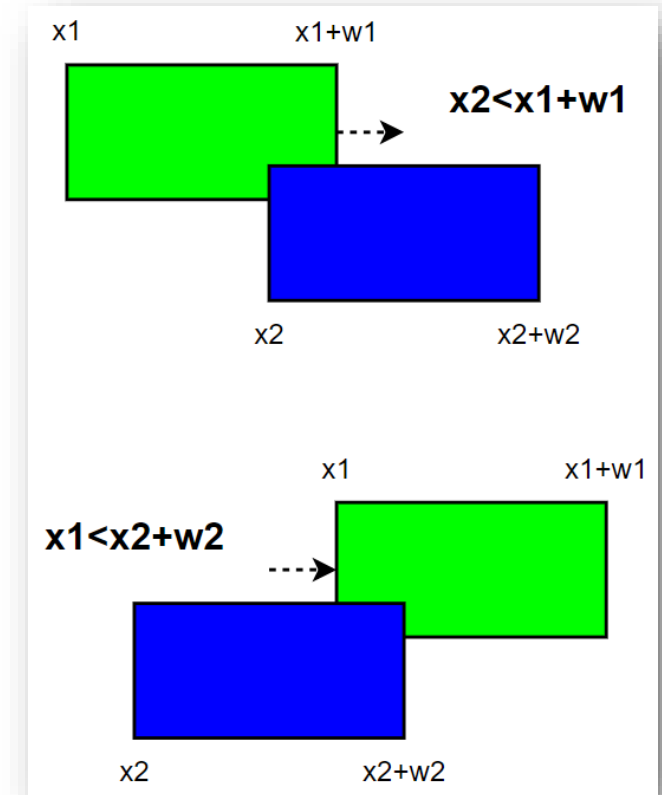
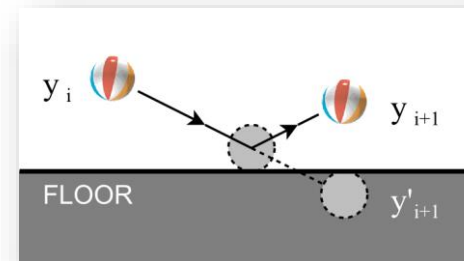
    def symbol(self) -> (int, int, int, int):
        '''Restituisce la posizione (x, y, w, h) dello sprite corrente, se l'immagine è contenuta
        in una immagine di grandi dimensioni altrimenti restituisce la tupla (0, 0, 0, 0)'''
        raise NotImplementedError('Abstract method')
```

```
class Arena():
    '''Generica 2D game, cui vengono assegnate le dimensioni e che contiene la lista dei
    personaggi del gioco '''
    def __init__(self, width: int, height: int):
        '''Crea una arena con specifica altezza e larghezza e lista di personaggi
        inizialmente vuota '''
        self._w, self._h = width, height
        self._actors = []

    def add(self, a: Actor):
        '''Aggiunge un personaggio al gioco
        I pesonaggi sono gestiti seguendo il loro ordine di inserimento '''
        if a not in self._actors:
            self._actors.append(a)

    def remove(self, a: Actor):
        '''Elimina un personaggio dal gioco '''
        if a in self._actors:
            self._actors.remove(a)
```

- esistono molti algoritmi di *collision detection*
- casi semplici: intersezione di rettangoli
- in caso di collisione, Arena...
 - invoca il metodo *collide* di entrambi gli oggetti
 - collisione tra *personaggio self* e personaggio *other* (secondo parametro)
- possibili errori nel calcolo del rimbalzo
 - di solito accettabili
 - altrimenti ... applicare correzioni



```
def move_all(self):  
    '''chiama il metodo move di ogni personaggio  
    dopo aver effettuato il movimento verifica  
    se è avvenuta un collisione tra il personaggio  
    e un altro personaggio e in tal caso chiama  
    il metodo collide di entrambi  
    '''  
    actors = list(reversed(self._actors))  
    for a in actors:  
        previous_pos = a.position()  
        a.move()  
        if a.position() != previous_pos: # inutile per personaggi statici  
            for other in actors:  
                # reversed order, so actors drawn on top of others  
                # (towards the end of the cycle) are checked first  
                if other is not a and self.check_collision(a, other):  
                    a.collide(other)  
                    other.collide(a)
```

```
def check_collision(self, a1: Actor, a2: Actor) -> bool:
    '''Verifica se i due personaggi (parametri) sono in collisione
       (bounding-box collision detection)
    '''
    x1, y1, w1, h1 = a1.position()
    x2, y2, w2, h2 = a2.position()
    return (y2 < y1 + h1 and y1 < y2 + h2
            and x2 < x1 + w1 and x1 < x2 + w2
            and a1 in self._actors and a2 in self._actors)

def actors(self) -> list:
    '''Restituisce una copia della lista dei personaggi
    '''
    return list(self._actors)

def size(self) -> (int, int):
    '''Restituisce le dimensioni dell'arena di gioco: (width, height)
    '''
    return (self._w, self._h)
```

```
class Ball(Actor):
    '''
    Pallina che si muove in diagonale e rimbalza
    se incontra i limiti dell'arena
    e se si scontra con altri personaggi
    '''
    def __init__(self, arena, x, y):
        self._x, self._y = x, y
        self._w, self._h = 20, 20
        self._speed = 5
        self._dx, self._dy = self._speed, self._speed
        self._arena = arena
        arena.add(self)          #si aggiunge ai personaggi dell'arena

    def move(self):
        ''' Rimbalza sui bordi dell'arena '''
        arena_w, arena_h = self._arena.size()
        if not (0 <= self._x + self._dx <= arena_w - self._w):
            self._dx = -self._dx
        if not (0 <= self._y + self._dy <= arena_h - self._h):
            self._dy = -self._dy
        self._x += self._dx
        self._y += self._dy
```

```
def collide(self, other):
    ''' gestione collisioni con altri personaggi '''
    if not isinstance(other, Ghost):          # non si urtano i fantasmi!
        x, y, w, h = other.position()
        if x < self._x:
            self._dx = self._speed # rimbalzo a ds
        else:
            self._dx = -self._speed # rimbalzo a sn
        if y < self._y:
            self._dy = self._speed # rimbalzo in basso
        else:
            self._dy = -self._speed # rimbalzo in alto

def position(self):
    return self._x, self._y, self._w, self._h

def symbol(self):
    ''' sprite coordinate 0,0 20,20 '''
    return 0, 0, self._w, self._h
```



```
class Ghost(Actor):
    ''' Fantasma si muove con spostamento casuale se esce da un bordo rientra dalla
        parte opposta. Può diventare invisibile poi riapparire
    '''
    def __init__(self, arena, x, y):
        ...

    def move(self):
        dx = choice([-5, 0, 5])
        dy = choice([-5, 0, 5])
        arena_w, arena_h = self._arena.size()
        self._x = (self._x + dx) % arena_w
        self._y = (self._y + dy) % arena_h
        if randrange(100) == 0:
            self._visible = not self._visible

    def collide(self, other):
        pass

    ...

def symbol(self):
    if self._visible:
        return 20, 0, self._w, self._h #visibile
    return 20, 20, self._w, self._h # invisibile
```

```
class Turtle(Actor):
    ''' Tartaruga: movimento guidato dai tasti freccia non supera i bordi dell'arena '''

    def move(self):
        arena_w, arena_h = self._arena.size()
        self._y += self._dy
        if self._y < 0:
            self._y = 0
        elif self._y > arena_h - self._h:
            self._y = arena_h - self._h
        ...

    def go_left(self):
        self._dx, self._dy = -self._speed, 0
    ...

    def go_down(self):
        self._dx, self._dy = 0, +self._speed

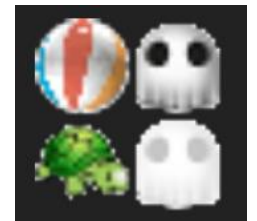
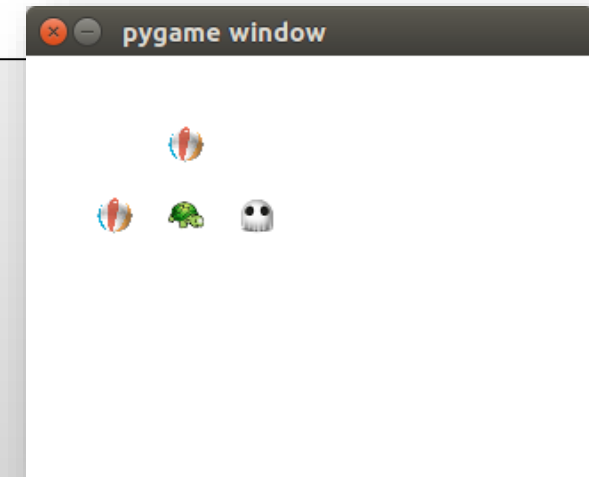
    def stay(self):
        self._dx, self._dy = 0, 0

    def collide(self, other):
        pass
```

```
...
arena = Arena(320, 240)
b1 = Ball(arena, 40, 80)
b2 = Ball(arena, 80, 40)
g = Ghost(arena, 120, 80)
turtle = Turtle(arena, 80, 80)
sprites = g2d.load_image("sprites.png")

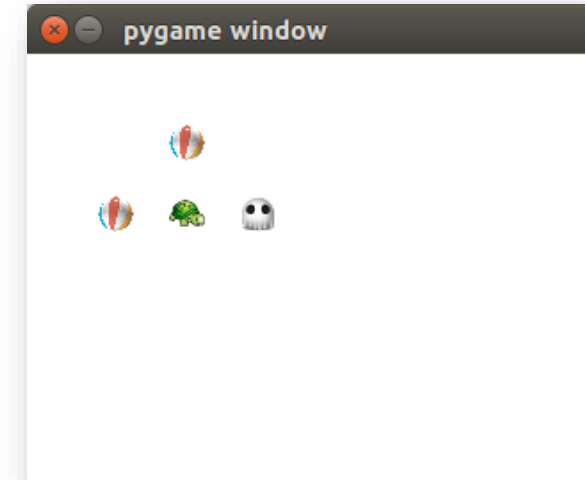
def update():
    arena.move_all() # logica del gioco

    g2d.fill_canvas((255, 255, 255))
    for a in arena.actors():
        g2d.draw_image_clip(sprites, a.position(), a.symbol())
    )
```



```
def keydown(code) :  
    if code == "ArrowUp":  
        turtle.go_up()  
    elif code == "ArrowDown":  
        turtle.go_down()  
    elif code == "ArrowLeft":  
        turtle.go_left()  
    elif code == "ArrowRight":  
        turtle.go_right()  
  
def keyup(code) :  
    turtle.stay()
```

```
def main() :  
    g2d.init_canvas(arena.size())  
    g2d.handle_keyboard(keydown, keyup)  
    g2d.main_loop(update, 1000 // 30)
```



http://www.ce.unipr.it/brython/?p3_oop_bounce_anim.py

liste e tuple
esercizi



○ **7.1 pallottole**

- all'esercizio in cui gli Alieni si muovono a serpentina
- aggiungere un attore **Bullet**
 - parte dal fondo e si muove verso l'alto
 - se esce dallo schermo, si rimuove dal gioco
 - se si scontra con un alieno, entrambi si rimuovono dal gioco
- nella funzione update, generare **casualmente** dei Bullet

http://www.ce.unipr.it/brython/?p3_oop_bullet.py