Tutorial Asteroids

Pronto ad avventurarti nello spazio con la tua astronave interstellare? Ma attento agli asteroidi !!

In questa guida creeremo il mitico gioco di Asteroids:

- L'astronave si può girare con le freccette destra e sinistra
- premendo freccia si accelera
- se tocca gli asteroidi esplode e si perde
- l'astronave può sparare raggi laser
- se i raggi laser toccano gli asteroidi, li distruggono
- si vince quando non rimangono più asteroidi

Puoi creare il codice anche online su trinket.io oppure su repl.it (creando un progetto `Python with Turtle`)

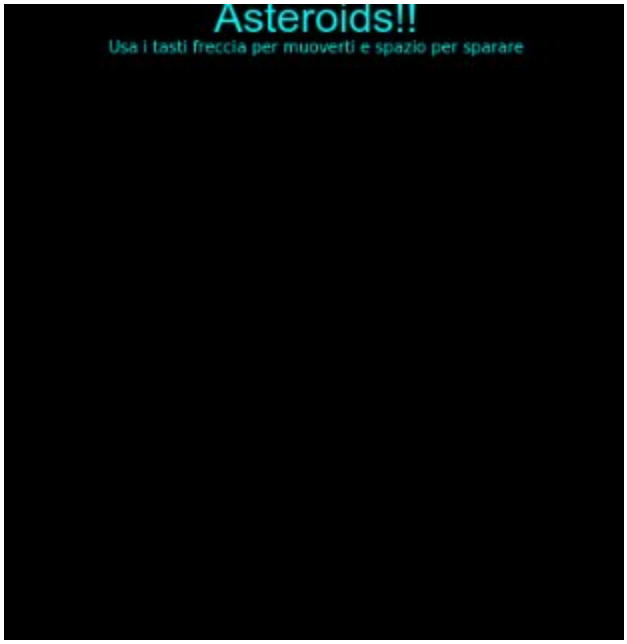
Il codice completo si trova qui: coderdojotrento.it/python1

Basato su una versione semplificata senza classi del demo asteroidi di Trinket

1. Impostiamo lo schermo

1.1 Setup schermo

```
from turtle import *
import random
import math

screen = Screen()
screen.setup(width=400,height=400)

screen_min_x = -screen.window_width()/2
screen_max_x =  screen.window_width()/2
```




```
screen_min_y = -screen.window_height()/2
screen_max_y =  screen.window_height()/2

screen.setworldcoordinates(screen_min_x,
                           screen_min_y,
                           screen_max_x,
                           screen_max_y)
screen.bgcolor("black")
```

1.2 Disegnamo titolo e istruzioni

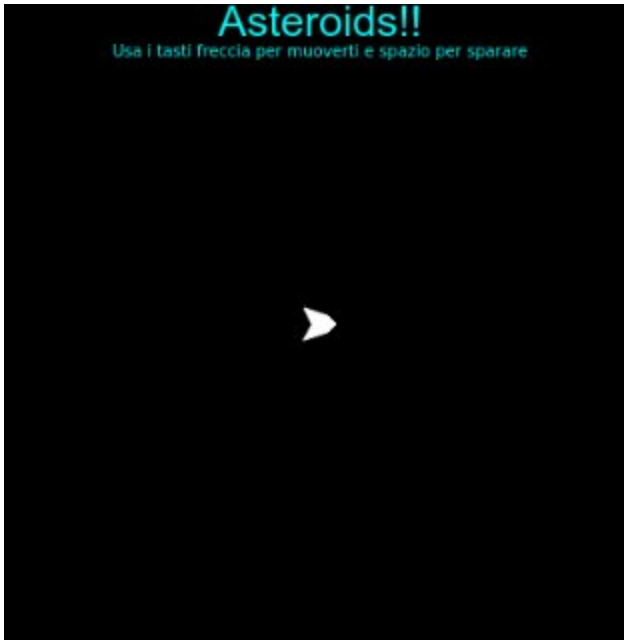
Nota che le istruzioni sono disegnate con la tartaruga predefinita fornitaci dal modulo `turtle`:

```
# Mostra titolo e istruzioni
penup()
hideturtle()
speed(0)
goto(0, screen_max_y - 20)
color('cyan')
write("Asteroids!!", align="center", font=("Arial",20))
goto(0, screen_max_y - 33)
write("Usa i tasti freccia per muoverti e spazio per sparare", align="center")
goto(0, 0)
color("lightblue")
```

2. Mostriamo l'astronave

Adesso creeremo un oggetto tartaruga appositamente per l'astronave nella funzione `ship_create`.

NOTA: la funzione crea un oggetto `Turtle` e **lo ritorna**, così l'oggetto diventa disponibile a chi ha chiamato la funzione.



Dovremo aggiungere dei campi all'oggetto `ship` specifici per il nostro gioco: ci servirà tener traccia della velocità e lo faremo salvando le sue componenti orizzontale e verticale in due campi che chiameremo `dx` e 'dy'.

Aggiungi questo codice dopo quello di prima:

```
# Crea l'astronave come turtle e la RITORNA
def ship_create(dx,dy,x,y):
  ship = Turtle()
  ship.speed(0)
  ship.penup()
  ship.color("white")
  ship.goto(x,y)
  ship.dx = dx
  ship.dy = dy
  ship.shape("ship")
  return ship

# Registra le shape
screen.register_shape("ship",((-10,-10),(-5,5),(0,10),(5,5),(10,-10),(0,-5)))


# Crea l'astronave
ship = ship_create(0,   # dx
                   0,   # dy
                   0,   # x
                   0)   # y
```

3. Giriamo l'astronave

Aggiungi questo codice dopo quello di prima:

```
# Definisce i comandi da tastiera
def key_left():
```

```
  ship.left(7)

def key_right():
  ship.right(7)


screen.onkey(key_left, 'left')
screen.onkey(key_right, 'right')
screen.listen()
```

3. Acceleriamo l'astronave

```
# sposta l'astronave della velocità ship.dx ship.dy
# se l'astronave esce dallo schermo da una parte, deve rientrare dall'altra
# AGGIUNTO FUNZIONE
def ship_move(ship):
  x = ship.xcor()
  y = ship.ycor()

  x = (x + ship.dx - screen_min_x) % (screen_max_x - screen_min_x) + screen_min_x
  y = (y + ship.dy - screen_min_y) % (screen_max_y - screen_min_y) + screen_min_y

  ship.goto(x,y)

# Accelera l'astronave nella direzione corrente, cioè incrementa ship.dx e ship.dy
# di un valore pari al coseno e seno della direzione corrente, ridotto di un quinto
# AGGIUNTO FUNZIONE
def ship_accelerate(ship):
  angle = ship.heading()
  acc_x = math.cos(math.radians(angle))*0.2
  acc_y = math.sin(math.radians(angle))*0.2
  ship.dx = ship.dx + acc_x
  ship.dy = ship.dy + acc_y
```

Verso la fine del codice, aggiungi

```
# AGGIUNTO istruzioni per velocizzare la grafica
screen.tracer(0)
hideturtle()

#AGGIUNTO funzione
def play():

  ship_move(ship)
  screen.update()
  screen.ontimer(play, 30)
```

Tra i comandi da tastiera, aggiungi:

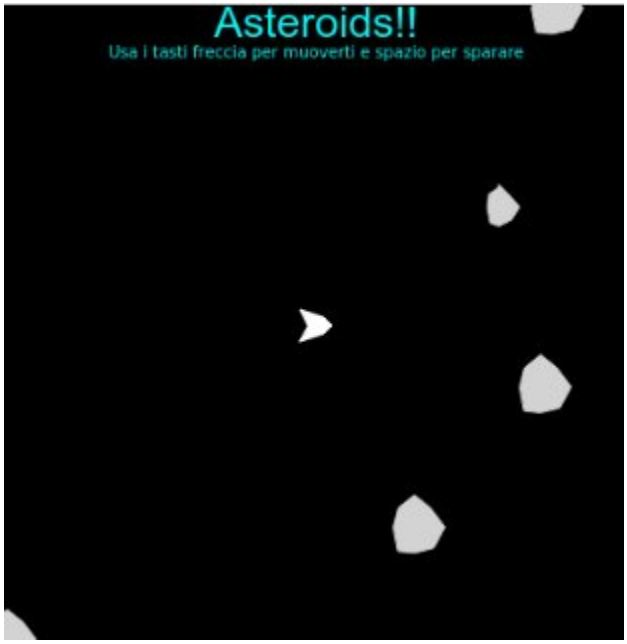
```
#AGGIUNTO funzione
def key_up():
```

```
  ship_accelerate(ship)

screen.onkey(key_up, 'up')
```

Alla FINE di TUTTO il codice, aggiungi:

```
#AGGIUNTO CHIAMATA A FUNZIONE
play()
```

4. Mostriamo gli asteroidi



Ogni asteroide sarà un oggetto tartaruga distinto. Per crearli, scriviamo una funzione `asteroid_create` che come la `ship_create` alla fine RITORNA la tartaruga creata rendendola disponibile a chi ha chiamato la funzione:

```
# AGGIUNTO FUNZIONE
def asteroid_create(dx,dy,x,y,size):
  asteroid = Turtle()
  asteroid.speed(0)
  asteroid.penup()
  asteroid.goto(x,y)
  asteroid.color('lightgrey')
  asteroid.radius = size*10-5
  asteroid.dx = dx
  asteroid.dy = dy
  asteroid.shape("rock" + str(size))
  return asteroid
```

Nella parte per registrare le shape, aggiungi:

```
# AGGIUNTO SHAPE ASTEROIDI
screen.register_shape("rock3",((-20, -16),(-21, 0),(-20,18),(0,27),
                               (17,15),(25,0),(16,-15),(0,-21)))
screen.register_shape("rock2",((-15, -10),(-16, 0), (-13,12),(0,19),
                               (12,10),(20,0),(12,-10),(0,-13)))
```

```
screen.register_shape("rock1",((-10,-5),(-12,0),(-8,8),(0,13),(8,6),
                               (14,0),(12,0),(8,-6),(0,-7)))
```

Salveremo gli asteroidi nella lista `asteroids`.

Dove crei la variabile `ship` (ma **fuori** dalla `ship_create`), aggiungi questo codice per creare diversi asteroidi in un ciclo `for`.

```
# AGGIUNTO VARIABILE
asteroids = []

# AGGIUNTO CREAZIONE ASTEROIDI

for k in range(5):
  dx = random.random() * 6 - 3
  dy = random.random() * 6 - 3

  # 50 così non appare sull'astronave
  x = random.randint(50,screen_max_x) * random.choice([-1,1])
  y = random.randint(50, screen_max_y) * random.choice([-1,1])

  asteroid = asteroid_create(dx,dy,x,y,random.randint(1,3))

  asteroids.append(asteroid)
```

5. Muoviamo gli asteroidi

```
# sposta l'asteroide della velocità asteroid.dx asteroid.dy
# AGGIUNTO FUNZIONE
def asteroid_move(asteroid):
  x = asteroid.xcor()
  y = asteroid.ycor()

  x = (x + asteroid.dx - screen_min_x) % (screen.window_width()) + screen_min_x
  y = (y + asteroid.dy - screen_min_y) % (screen.window_height()) + screen_min_y

  asteroid.goto(x,y)
```

In funzione `play` aggiungi for:

```
def play():

  ship_move(ship)

  # AGGIUNTO FOR
  for asteroid in asteroids:
    asteroid_move(asteroid)

  screen.update()
  screen.ontimer(play, 30)
```

6. Chi si schianta esplode

Aggiungi la funzione `intersect`:

```
# RITORNA True se i due oggetti passati sono a distanza inferiore alla
#         somma dei rispettivi radius
# AGGIUNTO FUNZIONE
def intersect(object1,object2):
  qx = (object1.xcor() - object2.xcor())**2
  qy = (object1.ycor() - object2.ycor())**2
  dist = math.sqrt(qx + qy)

  radius1 = object1.radius
  radius2 = object2.radius

  if dist <= radius1+radius2:
      return True
  else:
      return False
```

Aggiungi il campo `radius` in `ship_create`:

```
# Crea l'astronave come turtle e la RITORNA
def ship_create(dx,dy,x,y):
  ship = Turtle()
  ship.speed(0)
  ship.penup()
  ship.color("white")
  ship.goto(x,y)
  ship.dx = dx
  ship.dy = dy
  # AGGIUNTO ship.radius
  ship.radius = 10
  ship.shape("ship")
```

```
  return ship
```

Modifica `play` così:

```
def play():

  ship_move(ship)

  # AGGIUNTO VARIABILE
  gameover = False
  for asteroid in asteroids:
    asteroid_move(asteroid)
    # AGGIUNTO IF
    if intersect(ship,asteroid):
      write("BOOM!!!",font=("Arial",30),align="center")
      gameover = True

  # AGGIUNTO IF
  if gameover:
    ship.hideturtle()

  screen.update()

  # AGGIUNTO IF
  # richiama se stessa fra 30 millisecondi
  if not gameover:
    screen.ontimer(play, 30)
```

7. Sparare

Quando si preme spazio vengono creati proiettili

- Dopo aver percorso una certa distanza, spariscono
- per ora non possono colpire gli asteroidi

Aggiungi il campo `bullets` a `ship_create`:

```
# Crea l'astronave come turtle e la RITORNA
def ship_create(dx,dy,x,y):
  ship = Turtle()
  ship.speed(0)
  ship.penup()
  ship.color("white")
  ship.goto(x,y)
  ship.dx = dx
  ship.dy = dy
  ship.radius = 10
  # AGGIUNTO la lista bullets
  ship.bullets = []
  ship.shape("ship")
  return ship
```

Aggiungi tutte queste funzioni:

```
# Crea una proiettile come turtle e lo RITORNA
# AGGIUNTO FUNZIONE
def bullet_create(x,y,heading):
  bullet = Turtle()
  bullet.speed(0)
  bullet.penup()
  bullet.goto(x,y)
  bullet.seth(heading)
  bullet.color('yellow')
  bullet.max_distance = 500
  bullet.radius = 4
  bullet.distance = 0
  bullet.delta = 20
  bullet.shape("bullet")
  return bullet

# Muove il proiettile
#
# - se esce dallo schermo NON riappare dall'altra
# AGGIUNTO FUNZIONE
def bullet_move(bullet):
  bullet.distance = bullet.distance + bullet.delta
  bullet.forward(bullet.delta)
  if bullet_done(bullet):
    bullet.reset()

# Fa esplodere il proiettile
# AGGIUNTO FUNZIONE
def bullet_blow_up(bullet):
  bullet.goto(-600,0)

# RITORNA True se il proiettile ha superato la massima distanza percorribile,
#         False altrimenti
# AGGIUNTO FUNZIONE
```

```
def bullet_done(bullet):
  return bullet.distance >= bullet.max_distance
```

Aggiungi anche queste altre:

```
# crea un proiettile con bullet_create e lo aggiunge alla lista ship.bullets
# AGGIUNTO FUNZIONE
def ship_fire(ship):
  ship.bullets.append(bullet_create(ship.xcor(), ship.ycor(), ship.heading()))

# Per ogni proiettile
# - chiama bullet_move
# - rimuove da asteroids i proiettili che
#     - hanno colpito un asteroide
#     - oppure hanno esaurito la distanza massima percorribile
# AGGIUNTO FUNZIONE
def ship_pow_pow(ship, asteroids):
  bullets_rimasti = []
  for bullet in ship.bullets:
    bullet_move(bullet)

    if (not bullet_done(bullet)):
      bullets_rimasti.append(bullet)

  ship.bullets = bullets_rimasti
```

in `play` aggiungi chiamata a funzione `ship_pow_pow`

```
def play():

  ship_move(ship)


  gameover = False
  for asteroid in asteroids:
    asteroid_move(asteroid)

    if intersect(ship,asteroid):
      write("BOOM!!!",font=("Arial",30),align="center")
      gameover = True

  # AGGIUNTO CHIAMATA FUNZIONE
  ship_pow_pow(ship, asteroids)


  if gameover:
    ship.hideturtle()

  screen.update()

  # richiama se stessa fra 30 millisecondi
  if not gameover:
    screen.ontimer(play, 30)
```

Nella sezione tasti, aggiungi gestione tasto spazio:

```
# AGGIUNTO FUNZIONE
def space():
  ship_fire(ship)

screen.onkey(key_left, 'left')
screen.onkey(key_right, 'right')
# AGGIUNTO key space
screen.onkey(space, 'space')
screen.onkey(key_up, 'up')
```

8. Asteroidi che esplodono

Per finire, aggiungi la funzione `asteroid_blow_up`:

```
# Nasconde l'asteroide
# AGGIUNTO FUNZIONE
def asteroid_blow_up(asteroid):
  asteroid.goto(-600,0)
```

E cambia `ship_pow_pow` così:

```
def ship_pow_pow(ship, asteroids):
  bullets_rimasti = []
  for bullet in ship.bullets:
    bullet_move(bullet)

    # AGGIUNTO VARIABILE hit E IF
    hit = False
    for asteroid in asteroids:
      if intersect(asteroid, bullet):
        asteroids.remove(asteroid)
        asteroid_blow_up(asteroid)
        bullet_blow_up(bullet)
        hit = True

    # AGGIUNTO and hit
    if (not bullet_done(bullet) and not hit):
      bullets_rimasti.append(bullet)

  ship.bullets = bullets_rimasti
```

Aggiungi un `if` dentro `play`:

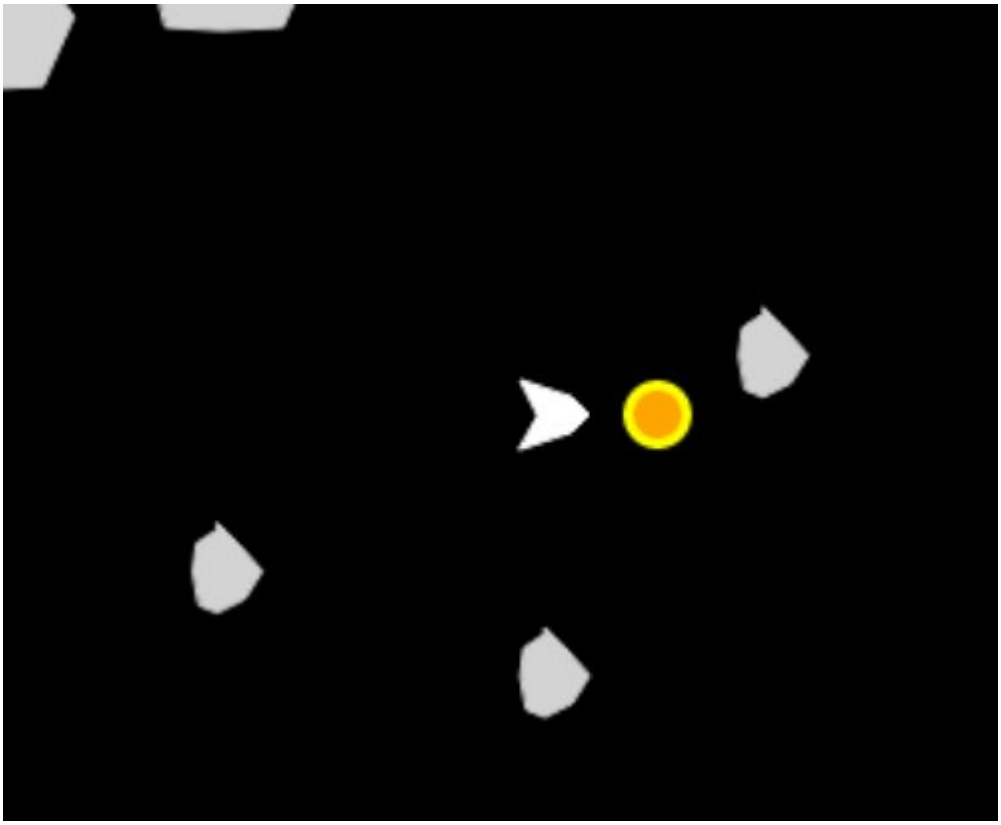
```
def play():

  ship_move(ship)


  gameover = False
  for asteroid in asteroids:
    asteroid_move(asteroid)
```

```
    if intersect(ship,asteroid):
      write("BOOM!!!",font=("Arial",30),align="center")
      gameover = True

  ship_pow_pow(ship, asteroids)

  # AGGIUNTO IF
  if not asteroids:
    color('green')
    write("Hai vinto !",font=("Arial",30),align="center")
    gameover = True

  if gameover:
    ship.hideturtle()

  screen.update()

  # richiama se stessa fra 30 millisecondi
  if not gameover:
    screen.ontimer(play, 30)
```

9. Sfide

9.1 mostra il punteggio

9.2 bomba di prossimità



Aggiungi bomba di prossimità: una volta raccolta, fa esplodere gli asteroidi entro un certo raggio `action_radius`

Creare la bomba

Per creare la bomba, implementa questa funzione:

```
# Crea una bomba di prossimità come turtle e la RITORNA
# AGGIUNTO FUNZIONE
def proximity_bomb_create(x,y, radius, action_radius):
```

- **SUGGERIMENTO**: Per disegnare il cerchio della bomba, usa il metodo `.dot` sulla tartaruga passandoci *il diametro* del della bomba (quindi dovrai moltiplicare il `radius` per due !)

Quando la bomba esplode

Qando la nave tocca la bomba, devi chiamare questa funzione:

```
# Fa esplodere una bomba di prossimità annientando tutti gli asteroidi entro
# distanza action_radius
# - produce un'animazione con un cerchio di raggio action_radius
# AGGIUNTO FUNZIONE
def proximity_bomb_blow_up(bomb):
  global asteroids
```

Per trovare gli asteroidi da eliminare, verifica se il `action_radius` di `bomb` più il `radius` di un asteroide sono minori della loro distanza

- **ATTENZIONE**: qua devi usare `action_radius` della bomba, *non* il `radius` !
- **ATTENZIONE**: per modificare la lista globale `asteroids`, ci è risultato più conveniente non passarla come parametro e invece dichiararla come `global`, così all'interno della funzione `proximity_bomb_blow_up` potremo *riassegnare* `asteroids` ad una NUOVA lista SENZA gli asteroidi esplosi.

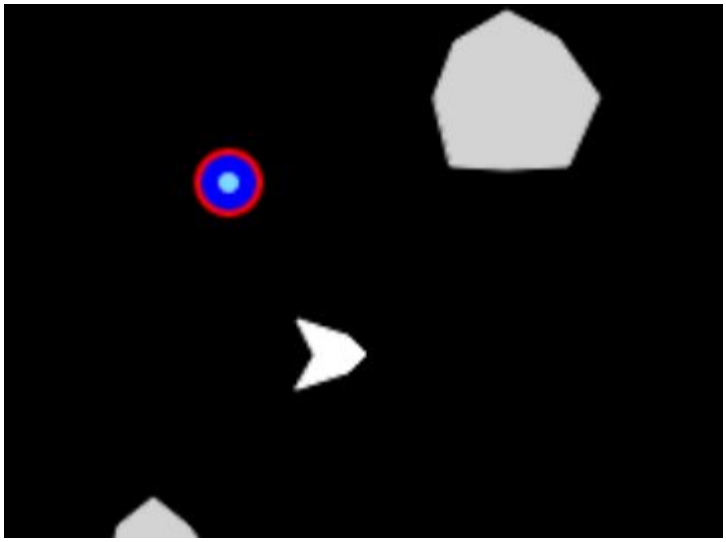
Per produrre l'animazione:

- per aspettare usa `time.sleep(0.3)` (ricordati di importare il modulo time con `import time` in cima allo script)
- per pulire il cerchio disegnato, usa il metodo `.clear()` sulla tartaruga
- dopo l'esplosione ricordati di mandare la bomba molto lontano con una chiamata al metodo `.goto` (così non potrà più collidere con l'astronave)
- **ATTENZIONE**: ricordati di aggiornare lo schermo con `screen.update()` da dentro la funzione !
- **SUGGERIMENTO**: per testare al meglio il tutto, ti conviene tenere gli asteroidi fermi commentando la linea `asteroid_move(asteroid)` nella funzione `play` e inizializzandone qualcuno vicino alla bomba (per esempio potresti forzarli tutti a stare sull'asse orizzontale ponendo la loro `y` a 0)

9.3 bomba positronica

Aggiungi bomba positronica: una volta raccolta, fa esplodere 3 asteroidi a caso. Per tenere le cose semplici, con 'a caso' intendiamo che verranno eliminati gli ultimi 3 nella lista `asteroids`

- **ATTENZIONE 1**: in questo caso alla funzione `positronic_bomb_blow_up` passiamo come parametro la lista `asteroids`, perchè non abbiamo bisogno di riassegnarla all'interno della funzione ma potremo chiamare dei metodi come `.pop()`che la MODIFICANO
- **ATTENZIONE 2**: ricordati di gestire il caso in cui gli asteroidi presenti sono meno di 3



- **SUGGERIMENTO 1**: per eliminare elementi dalla fine della lista `asteroids`, chiama ripetutamente il metodo `.pop()`
- **SUGGERIMENTO 2**: per testare al meglio il tutto, ti conviene tenere gli asteroidi fermi commentando la linea `asteroid_move(asteroid)` nella funzione `play`

Quando prendi la bomba, aggiungi animazione dell'esplosione che mostra un cerchio ridisegnato più volte, ogni volta che lo disegni ingrandiscilo e aspetta 0.3 secondi:

- per aspettare usa `time.sleep(0.3)` (ricordati di importare il modulo time con `import time` in cima allo script)
- per pulire il cerchio disegnato, usa il metodo `.clear()` sulla tartaruga della bomba
- **ATTENZIONE**: ricordati di aggiornare lo schermo con `screen.update()` da dentro la funzione !

Aggiungi queste due funzioni chiamandole dove opportuno nel resto del codice:

```
# Crea una bomba positronica come turtle e la RITORNA
# AGGIUNTO FUNZIONE
def positronic_bomb_create(x,y, radius):

# Fa esplodere una bomba positronica annientando 3 asteroidi a caso
# - con 'a caso' possiamo assumere che vengono rimossi gli ultimi 3
# - produce un'animazione di cerchi che si espandono fino a
#   coprire tutto lo schermo
# AGGIUNTO FUNZIONE
def positronic_bomb_blow_up(bomb, asteroids):
```

9.4 - buco nero

Aggiungi buco nero che attira costantemente la nave verso di sè

Se l'astronave lo tocca, viene risucchiata e si perde:

Crea tartaruga e chiamala `black_hole`, aggiungendo `radius` e `mass` come campi:

```
# Crea una buco nero come turtle e lo RITORNA
# il buco nero ha posizione e radius ma non velocità perchè è fisso
# AGGIUNTO FUNZIONE
def black_hole_create(x,y,radius, mass):
  pass
```


```
# AGGIUNTO VARIABILE
# (nota: i buchi neri veri hanno massa nell'ordine di 10^30 kg)
black_hole = black_hole_create(100,   # x
                               100,   # y
                               20,    # radius
                               10e12) # massa: 10 miliardi di tonnellate
```

Aggiungi campo `massa` `ship`:

```
# Crea l'astronave come turtle e la RITORNA
# AGGIUNTO VARIABILE mass
def ship_create(dx,dy,x,y, mass):
```

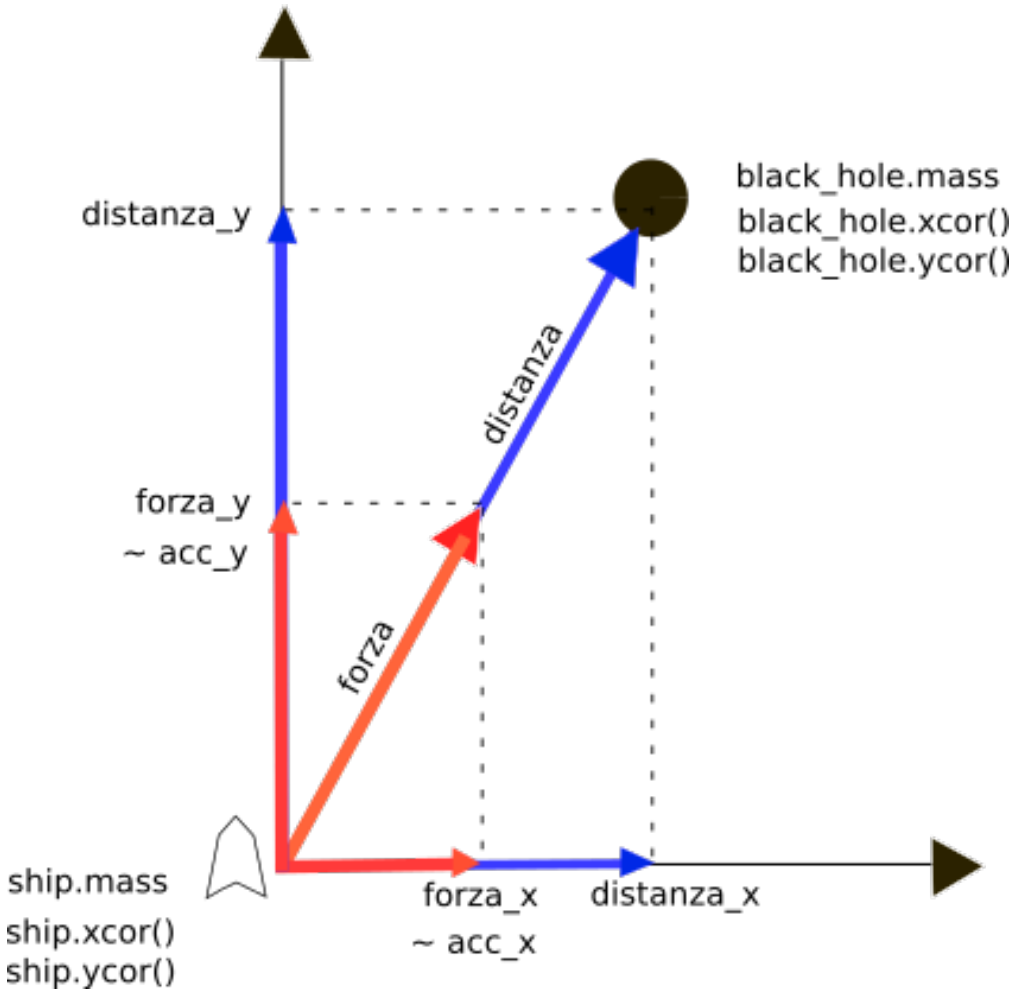
Puoi inizializzare `ship` così:

```
# Crea l'astronave

ship = ship_create(0,   # dx
                   0,   # dy
                   0,   # x
                   0,   # y
                   1000 # AGGIUNTO massa in kili
                   )
```

Attrazione universale

Aggiungi funzione `black_hole_attract`:

```
# cambia la velocità dx e dy di ship in base alla posizione di black_hole
# AGGIUNTO FUNZIONE
```


black_hole.mass
black_hole.xcor()
black_hole.ycor()
distanza_y
distanza
forza_y
~ acc_y
forza
ship.mass
ship.xcor()
ship.ycor()
forza_x
~ acc_x
distanza_x

```
def black_hole_attract(black_hole, ship):
  pass
```

Per calcolare l'accelarazione da imprimere all'astronave:

1. calcola la distanza tra astronave e buco nero
2. calcola la magnitudo della forza che spinge l'astronave verso il buco nero, usando la legge di gravitazione universale:

$$F = G\frac{m_1 m_2}{distanza^2}$$

3. Calcola le componenti `forza_x` e `forza_y` del vettore: basta fare le dovute proporzioni fra triangoli simili quindi non occorrono formule trigonometriche con angoli, seni o coseni)
4. L'accelerazione impressa sull'astronave si ottiene dalla forza
5. L'incremento di velocità per aggiornare `dx` e `dy` di `ship` si ottiene moltiplicando l'accelerazione per unità di tempo, che assumiamo essere un frame nell'animazione

Ricordati di chiamare `black_hole_attract` in `play`:

```
def play():

  ship_move(ship)

  # AGGIUNTO CHIAMATA A FUNZIONE
  black_hole_attract(black_hole, ship)
```

e verificare se l'astronave sta toccando il buco nero:

```
  # AGGIUNTO CONDIZIONE DI FINE
  if intersect(ship, black_hole):
      write("PERSO NEL BUCO NERO !!!",font=("Arial",20),align="center")
      gameover = True
```

9.5 Marble Madness

Dopo che hai completato la sfida del buco nero, potresti riusare gli stessi principi sull'attrazione per creare qualche altro gioco apparentemente diverso, come per esempio Marble Madness:

In questo gioco si controlla una biglia e bisogna evitare che precipiti in conche e altri ostacoli

- la conche di fatto si comportanto da attrattori esattamente come i buchi neri. Visto che possono essere più di una, dovrai salvare in una lista `conche`
- quando si finisce in una conca il gioco non termina ma semplicemente la biglia continua a rotolare intorno alla conca
- è difficile rappresentare la direzione in cui sta 'guardando' una biglia, perciò diversamente dall'astronave conviene cambiare il movimento in modo che con i tasti freccia ci si sposti in direzioni cardinali assolute (su giù destra sinistra)
- quando la biglia tocca i bordi, invece di passare dall'altra parte dello schermo potrebbe fermarsi (facile) o rimbalzare (un po' più complicato)
- le conche sono state sfumate disegnando diversi centri concentrici e cambiando la gradazione del colore in un `for`. Per farlo, considera che puoi specificare un colore nelle sue tre componenti `rossa verde` e `blu` con numeri da `0` a `255` inclusi, cambiando ripetutamente i valori di verde dentro un ciclo `for` dovresti riuscire ad ottenere un bell'effetto sfumato!


```
color(50,100,50)  #  poco rosso, abbastanza verde, poco blu
color(50,255,50)  #  poco rosso, tantissimo verde, poco blu
```

- prova ad aggiungere rampe ed altri ostacoli !

[ ]: