37. PyGame II – Space Rocks Game – ΟΛΟΚΛΗΡΩΣΗ

Επανάληψη κεφαλαίων 21 - 23

37.0.1 Λύση του προηγούμενου project (Παιχνίδι φιδάκι)

Χρησιμοποιήστε την Pygame για να φτιάξετε το κλασικό παιχνίδι «Φιδάκι».

Δημιουργήστε ένα πλαίσιο / οθόνη για να διαδραματίζεται το παιχνίδι. Η «τροφή» του φιδιού, θα είναι ένα τετραγωνάκι της οθόνης. Το ίδιο και το ξεκίνημα του μεγέθους του φιδιού. Το φιδάκι, μεγαλώνει όσο καταφέρνει να «τρώει» τροφή (τετραγωνάκια). Με κάθε τετραγωνάκι, αυξάνονται και οι πόντοι του σε έναν πίνακα καταμέτρησης, επάνω αριστερά στον πίνακα/οθόνη του παιχνιδιού. Ο παίκτης χάνει αν το φιδάκι ακουμπήσει τα όρια του καμβά/πλαισίου/ οθόνης του παιχνιδιού.

Λύση

Μπορείτε να βρείτε την τεκμηρίωση των πλήκτρων του πληκτρολογίου στην Pygame, εδώ.

Ο σχολιασμός του project βρίσκεται μέσα στον κώδικα:

```
# Απαραίτητες εισαγωγές
import pygame
import time
import random

# Αρχικοποίηση του Pygame
pygame.init()

# Ορισμός των χρωμάτων χρησιμοποιώντας τη μορφή RGB
white = (255, 255, 255)
yellow = (255, 255, 102)
black = (0, 0, 0)
red = (213, 50, 80)
green = (0, 255, 0)
blue = (50, 153, 213)

# Ορισμός των διαστάσεων του παραθύρου του παιχνιδιού
dis_width = 600
dis_height = 400

# Δημιουργία του παραθύρου του παιχνιδιού
dis = pygame.display.set_mode((dis_width, dis_height))
pygame.display.set_caption('Φιδάκι!')

# Δημιουργία ενός ρολογιού για τον έλεγχο της ταχύτητας του
παιχνιδιού
clock = pygame.time.Clock()

# Ορισμός του μεγέθους του κεφαλιού και της ταχύτητας του
φιδιού
snake_block = 10
snake_speed = 15

# Ορισμός των γραμματοσειρών για τα μηνύματα και το σκορ
font_style = pygame.font.SysFont("bahnschrift", 25)
score_font = pygame.font.SysFont("comicsansms", 35)

# Συναρτήσεις - Μια συνάρτηση για κάθε λειτουργία του
παιχνιδιού

# Συνάρτηση που εμφανίζει το σκορ στην οθόνη
def Your_score(score):
    '''
```

```
Η Pygame δεν παρέχει άμεσο τρόπο εγγραφής κειμένου σε ένα
αντικείμενο Surface. Η μέθοδος render() χρησιμοποιείται για
τη
δημιουργία ενός αντικειμένου Surface από το κείμενο, το
οποίο
στη συνέχεια μπορεί να μεταφερθεί στην οθόνη.
Η μέθοδος render() μπορεί να αποδώσει μόνο μεμονωμένες
γραμμές.
'''
    value = score_font.render("Your Score: " + str(score),
True, yellow)
    dis.blit(value, [0, 0])


# Συνάρτηση που σχεδιάζει το σώμα του φιδιού στην οθόνη
def our_snake(snake_block, snake_list):
    # Για κάθε στοιχείο της λίστας του φιδιού
    for x in snake_list:
        # Σχεδίασε ένα ορθογώνιο (κομμάτι του φιδιού) στη
θέση του στον οριζόντιο άξονα (x[0])
        # και τον κατακόρυφο άξονα (x[1]), χρησιμοποιώντας
το χρώμα μαύρο (black)
        pygame.draw.rect(dis, black, [x[0], x[1],
snake_block, snake_block])



# Συνάρτηση που εμφανίζει μήνυμα στην οθόνη
def message(msg, color):
    mesg = font_style.render(msg, True, color)
    dis.blit(mesg, [dis_width / 6, dis_height / 3])


# Κύρια συνάρτηση του παιχνιδιού
def gameLoop():
    game_over = False
    game_close = False # Όσο ο παίκτης κερδίζει

    # Αρχικές συντεταγμένες του κεφαλιού του φιδιού
    x1 = dis_width / 2
    y1 = dis_height / 2

    # Αρχικές μεταβολές των συντεταγμένων του κεφαλιού
    x1_change = 0
```

```
    y1_change = 0

    # Αρχικοποίηση της λίστας του φιδιού και του μήκους του
φιδιού
    snake_List = []
    Length_of_snake = 1

    # Αρχικοποίηση των συντεταγμένων του φαγητού (τετράγωνα
που "τρώει" το φίδι)
    foodx = round(random.randrange(0, dis_width -
snake_block) / 10.0) * 10.0
    foody = round(random.randrange(0, dis_height -
snake_block) / 10.0) * 10.0

    while not game_over:

        while game_close == True: # Όταν ο παίκτης χάνει
            dis.fill(blue)
            message("Έχασες! C-Ξαναπαίξε ή Q-Κλείσιμο", red)
            Your_score(Length_of_snake - 1)
            pygame.display.update()

            for event in pygame.event.get():
                if event.type == pygame.KEYDOWN:
                    if event.key == pygame.K_q:
                        game_over = True
                        game_close = False
                    if event.key == pygame.K_c:
                        gameLoop()

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                game_over = True
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_LEFT:
                    x1_change = -snake_block
                    y1_change = 0
                elif event.key == pygame.K_RIGHT:
                    x1_change = snake_block
                    y1_change = 0
                elif event.key == pygame.K_UP:
                    y1_change = -snake_block
                    x1_change = 0
                elif event.key == pygame.K_DOWN:
```

```
                    y1_change = snake_block
                    x1_change = 0

        # Έλεγχος αν η κεφαλή του φιδιού βρίσκεται εκτός
οθόνης
        if x1 >= dis_width or x1 < 0 or y1 >= dis_height or
y1 < 0:
            game_close = True
        x1 += x1_change
        y1 += y1_change
        dis.fill(blue)
        '''
Σχεδιασμός ενός ορθογωνίου στην οθόνη,
που αντιπροσωπεύει το φαγητό του φιδιού.
Τα ορίσματα της συνάρτησης είναι τα εξής:

dis: Το παράθυρο του παιχνιδιού στο οποίο θα σχεδιαστεί το
ορθογώνιο.

green: Το χρώμα του ορθογωνίου, που σε αυτήν την περίπτωση
είναι το πράσινο.
[foodx, foody, snake_block, snake_block]: Το πρώτο ζευγάρι
αριθμών είναι
οι συντεταγμένες της επάνω αριστερής γωνίας του ορθογωνίου
(x, y),
και το δεύτερο ζευγάρι αριθμών είναι το πλάτος και το ύψος
του ορθογωνίου
(width, height), τα οποία είναι και τα snake_block σε κάθε
διάσταση.
'''
        pygame.draw.rect(dis, green, [foodx, foody,
snake_block, snake_block])
        snake_Head = []
        snake_Head.append(x1)
        snake_Head.append(y1)
        snake_List.append(snake_Head)
        '''
Αυτό το τμήμα κώδικα ελέγχει αν το μήκος της λίστας
snake_List,
που περιέχει τις συντεταγμένες κάθε κομματιού του φιδιού,
είναι μεγαλύτερο από το μήκος του φιδιού (Length_of_snake).
Αν αυτό ισχύει, τότε το πρώτο στοιχείο της λίστας (η ουρίτσα
του φιδιού)
διαγράφεται.
```

```
'''
        if len(snake_List) > Length_of_snake:
            del snake_List[0]

        # Έλεγχος αν η κεφαλή του φιδιού συγκρούεται με το
σώμα του
        for x in snake_List[:-1]:
            if x == snake_Head:
                game_close = True

        our_snake(snake_block, snake_List)
        Your_score(Length_of_snake - 1)

        pygame.display.update()

        # Έλεγχος αν το φίδι τρώει τα τετραφωνάκια
        if x1 == foodx and y1 == foody:
            foodx = round(random.randrange(0, dis_width -
snake_block) / 10.0) * 10.0
            foody = round(random.randrange(0, dis_height -
snake_block) / 10.0) * 10.0
            Length_of_snake += 1

        clock.tick(snake_speed)

    pygame.quit()
    quit()


# Έναρξη του παιχνιδιού
gameLoop()
```

37.1.0 Επιτάχυνση του διαστημοπλοίου – Βήμα 6.1

Στο παιχνίδι μας, όταν πατήσουμε Up, η ταχύτητα του διαστημοπλοίου θα αυξηθεί. Όταν αφήσουμε το πλήκτρο, το διαστημόπλοιο θα διατηρήσει την τρέχουσα ταχύτητά του, αλλά δεν θα πρέπει πλέον να επιταχύνει. Έτσι, για να το επιβραδύνουμε, θα πρέπει να γυρίσουμε το διαστημόπλοιο και να πατήσουμε ξανά το Up.

Η διαδικασία μπορεί να φαίνεται ήδη λίγο περίπλοκη, οπότε πριν προχωρήσουμε, ας κάνουμε μια σύντομη ανακεφαλαίωση:

• το direction είναι ένα διάνυσμα που περιγράφει το προς τα πού δείχνει το διαστημόπλοιο.

• το velocity είναι ένα διάνυσμα που περιγράφει πού κινείται το διαστημόπλοιο σε κάθε καρέ.

• ο ACCELERATION είναι ένας σταθερός αριθμός που περιγράφει πόσο γρήγορα το διαστημόπλοιο μπορεί να επιταχύνει σε κάθε καρέ.

Μπορούμε να υπολογίσουμε τη μεταβολή της ταχύτητας πολλαπλασιάζοντας το διάνυσμα direction με την τιμή της ACCELERATION και να προσθέσουμε το αποτέλεσμα στο τρέχον διάνυσμα velocity. Αυτό συμβαίνει μόνο όταν ο κινητήρας είναι αναμμένος—δηλαδή όταν παίκτης πατάει το πλήκτρο Up. Η νέα θέση του διαστημοπλοίου υπολογίζεται **προσθέτοντας την τρέχουσα** ταχύτητα στην τρέχουσα θέση του διαστημοπλοίου. Αυτό συμβαίνει σε κάθε καρέ, ανεξάρτητα από την κατάσταση του κινητήρα.

Γνωρίζοντας αυτό, μπορούμε να προσθέσουμε την τιμή της ACCELERATION στην κλάση Spaceship:

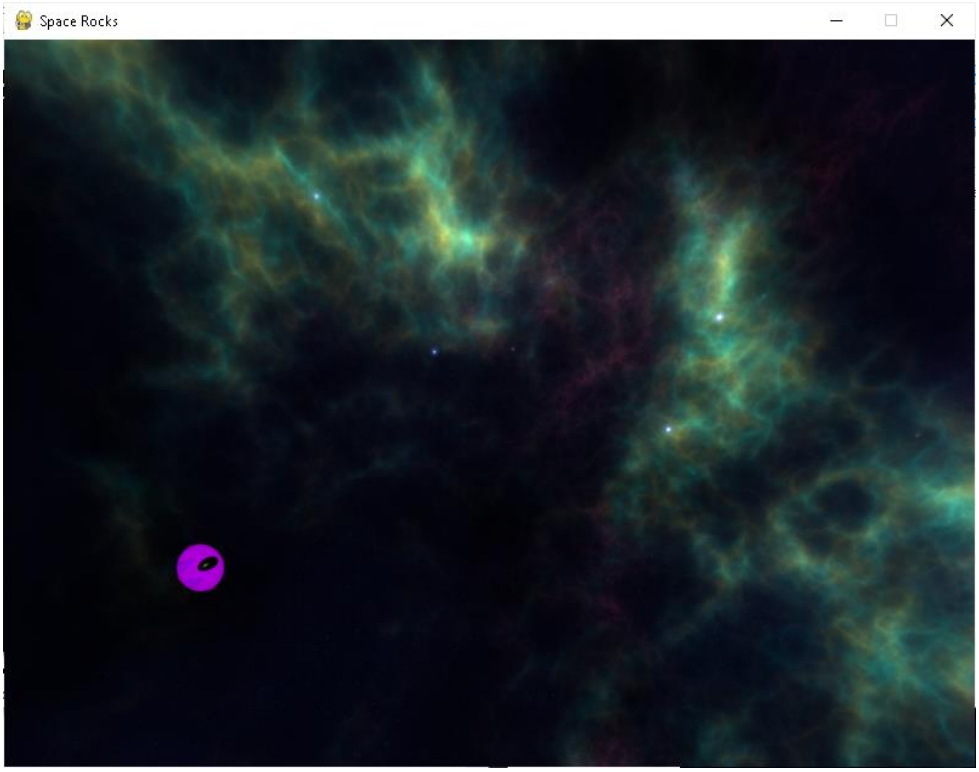
```
class Spaceship(GameObject):
MANEUVERABILITY = 3
ACCELERATION = 0.25
Στη συνέχεια, ας δημιουργήσουμε την accelerate() στην κλάση
Spaceship:
def accelerate(self):
```

```
self.velocity += self.direction * self.ACCELERATION
```

Τώρα μπορούμε να προσθέσουμε χειρισμό εισόδου _handle_input() στο SpaceRocks. Ομοίως με την περιστροφή, θα ελέγχει την τρέχουσα κατάσταση του πληκτρολογίου και όχι τα συμβάντα του πληκτρολογίου. Η σταθερά για το πλήκτρο Up είναι pygame.K_UP:

```
def _handle_input(self):
for event in pygame.event.get():
if event.type == pygame.QUIT or (
event.type == pygame.KEYDOWN and event.key ==
pygame.K_ESCAPE):
quit()
is_key_pressed = pygame.key.get_pressed()
if is_key_pressed[pygame.K_RIGHT]:
self.spaceship.rotate(clockwise=True)
elif is_key_pressed[pygame.K_LEFT]:
self.spaceship.rotate(clockwise=False)
if is_key_pressed[pygame.K_UP]:
self.spaceship.accelerate()
```

Ας δοκιμάσουμε να τρέξουμε το παιχνίδι μας , να περιστρέψουμε το διαστημόπλοιο και να ανάψουμε τον κινητήρα:



37.1.1 Συγκράτηση αντικειμένων στην οθόνη – Βήμα 6.2

Ένα σημαντικό στοιχείο αυτού του παιχνιδιού είναι να βεβαιωθούμε ότι τα αντικείμενα του παιχνιδιού δεν φεύγουν από την οθόνη. Μπορούμε είτε να τα αναπηδήσουμε από την άκρη της οθόνης, είτε να τα κάνουμε να εμφανιστούν ξανά στην αντίθετη άκρη της οθόνης.

Στο παιχνίδι μας, θα εφαρμόσουμε το τελευταίο.

Ας ξεκινήσουμε εισάγοντας την κλάση Vector2 στο αρχείο

```
space_rocks/utils.py:

from pygame.image import load
from pygame.math import Vector2
```

Στη συνέχεια, ας δημιουργήσουμε την wrap_position() στο ίδιο αρχείο:

```
def wrap_position(position, surface):
x, y = position
w, h = surface.get_size()
return Vector2(x % w, y % h)
```

Χρησιμοποιώντας τον τελεστή modulo στη γραμμή 4, βεβαιωνόμαστε ότι η θέση δεν φεύγει ποτέ από την περιοχή της δεδομένης επιφάνειας.

Στο παιχνίδι μας, αυτή η επιφάνεια θα είναι η οθόνη.

Εισάγουμε αυτήν τη νέα μέθοδο στο αρχείο space_rocks/models.py:

```
from pygame.math import Vector2
from pygame.transform import rotozoom
from utils import load_sprite, wrap_position
```

Τώρα μπορούμε να ενημερώσουμε τη move() στην κλάση GameObject:

```
def move(self, surface):
self.position = wrap_position(self.position + self.velocity,
surface)
```

Σημειώστε ότι η χρήση της wrap_position() δεν είναι η μόνη αλλαγή εδώ. Μπορούμε επίσης να προσθέσουμε ένα νέο όρισμα, το surface σε αυτήν τη μέθοδο. Αυτό συμβαίνει επειδή πρέπει να γνωρίζουμε την περιοχή γύρω από την οποία πρέπει να συγκρατήσουμε τη θέση.

Ας θυμηθούμε να ενημερώσουμε την κλήση μεθόδου και στην κλάση

SpaceRocks:

```
def _process_game_logic(self):
self.spaceship.move(self.screen)
```

Τώρα το διαστημόπλοιό μας εμφανίζεται ξανά στην άλλη πλευρά της οθόνης.

Η λογική της μετακίνησης και της περιστροφής του διαστημοπλοίου είναι έτοιμη. Όμως το πλοίο είναι ακόμα μόνο του στον κενό χώρο.

Ώρα να προσθέσουμε αστεροειδείς!

37.1.2 Δημιουργία αστεροϊδών – Βήμα 6.3

Παρόμοια με το Spaceship, θα ξεκινήσουμε δημιουργώντας μια κλάση που ονομάζεται Asteroid που κληρονομεί από την GameObject. Ας επεξεργαστούμε το αρχείο space_rocks/models.py ως εξής:

```
class Asteroid(GameObject):
def __init__(self, position):
super().__init__(position, load_sprite("asteroid"), (0, 0))
```

Όπως και πριν, ξεκινάμε καλώντας τον κατασκευαστή της GameObject με μια συγκεκριμένη εικόνα. Έχουμε προσθέσει την εικόνα σε ένα από τα προηγούμενα βήματα.

Στη συνέχεια, εισάγουμε τη νέα κλάση στο αρχείο space_rocks/game.py:

```
import pygame
from models import Asteroid, Spaceship
from utils import load_sprite
```

Τέλος, ας επεξεργαστούμε τον κατασκευαστή της κλάσης SpaceRocks στο ίδιο αρχείο για να δημιουργήσουμε έξι αστεροειδείς:

```
def __init__(self):
self._init_pygame()
self.screen = pygame.display.set_mode((800, 600))
self.background = load_sprite("space", False)
self.clock = pygame.time.Clock()
self.asteroids = [Asteroid((0, 0)) for _ in range(6)]
```

```
self.spaceship = Spaceship((400, 300))
```

Τώρα που έχουμε περισσότερα αντικείμενα στο παιχνίδι μας, θα ήταν καλή ιδέα να δημιουργήσουμε μια βοηθητική μέθοδο στην κλάση SpaceRocks που να τα επιστρέφει όλα.

Αυτή η μέθοδος θα χρησιμοποιηθεί στη συνέχεια από τη λογική σχεδίασης και κίνησης. Με αυτόν τον τρόπο, μπορούμε αργότερα να εισάγουμε νέους τύπους αντικειμένων στο παιχνίδι μας τροποποιώντας μόνο αυτήν τη μέθοδο ή μπορούμε να εξαιρέσουμε ορισμένα αντικείμενα από αυτήν την ομάδα εάν είναι απαραίτητο.

Καλούμε αυτή τη μέθοδο _get_game_objects():

```
def _get_game_objects(self):
return [*self.asteroids, self.spaceship]
```

Τώρα ας τη χρησιμοποιήσουμε για να μετακινήσουμε όλα τα αντικείμενα του παιχνιδιού σε έναν ενιαίο βρόχο με επεξεργασία της _process_game_logic():

```
def _process_game_logic(self):
for game_object in self._get_game_objects():
game_object.move(self.screen)
```

Το ίδιο ισχύει και για την _draw():

```
def _draw(self):
self.screen.blit(self.background, (0, 0))
for game_object in self._get_game_objects():
game_object.draw(self.screen)

pygame.display.flip()
self.clock.tick(60)
```

Αν εκτελέσουμε το παιχνίδι μας τώρα και θα δούμε μια οθόνη με τους

Αστεροειδείς.

37.1.3 Τυχαιοποίηση της θέσης των αστεροϊδών – Βήμα 7.1

Για να δημιουργήσουμε μια τυχαία θέση, θα πρέπει να προσθέσουμε ορισμένα import στο αρχείο space_rocks/utils.py:

```
import random

from pygame.image import load
from pygame.math import Vector2
```

Στη συνέχεια, ας δημιουργήσουμε μια μέθοδο που ονομάζεται get_random_position() στο ίδιο αρχείο:

```
def get_random_position(surface):
return Vector2(
random.randrange(surface.get_width()),
random.randrange(surface.get_height()),)
```

Αυτό θα δημιουργήσει ένα τυχαίο σύνολο συντεταγμένων σε μια δεδομένη επιφάνεια και θα επιστρέψει το αποτέλεσμα σαν ένα στιγμιότυπο της Vector2.

Στη συνέχεια, εισάγουμε αυτήν τη μέθοδο στο αρχείο space_rocks/game.py:

```
import pygame
from models import Asteroid, Spaceship
from utils import get_random_position, load_sprite
```

Τώρα ας χρησιμοποιήσουμε τη get_random_position() για να τοποθετήσουμε και τους έξι αστεροειδείς σε τυχαίες τοποθεσίες.

Τροποποίηση του κατασκευαστή της κλάσης SpaceRocks:

```
def __init__(self):
self._init_pygame()
self.screen = pygame.display.set_mode((800, 600))
self.background = load_sprite("space", False)
self.clock = pygame.time.Clock()

self.asteroids = [
Asteroid(get_random_position(self.screen)) for _ in range(6)
]
self.spaceship = Spaceship((400, 300))
```

Τώρα όταν τρέχουμε το παιχνίδι, θα δούμε μια ωραία, τυχαία κατανομή αστεροειδών στην οθόνη.

Αυτό φαίνεται πολύ καλύτερο, αλλά υπάρχει ένα μικρό πρόβλημα: οι αστεροειδείς δημιουργήθηκαν στην ίδια περιοχή με το διαστημόπλοιο.

Αφού προσθέσουμε συγκρούσεις, αυτό θα έκανε τον παίκτη να χάσει αμέσως μετά την έναρξη του παιχνιδιού. Αυτό θα ήταν πολύ άδικο!

Μια λύση σε αυτό το πρόβλημα είναι να ελέγξουμε εάν η θέση είναι πολύ κοντά στο διαστημόπλοιο και αν ναι, να δημιουργήσουμε μια νέα μέχρι να βρεθεί μια έγκυρη θέση.

Ας ξεκινήσουμε δημιουργώντας μια σταθερά που αντιπροσωπεύει μια περιοχή που πρέπει να παραμείνει κενή. Μια τιμή 250pixel θα πρέπει να είναι αρκετή:

```
class SpaceRocks:
MIN_ASTEROID_DISTANCE = 250
```

Τώρα μπορούμε να τροποποιήσουμε τον κατασκευαστή της κλάσης SpaceRocks για να βεβαιωθούμε ότι οι παίκτες μας έχουν πάντα την ευκαιρία να κερδίσουν:

```
def __init__(self):
self._init_pygame()
self.screen = pygame.display.set_mode((800, 600))
self.background = load_sprite("space", False)
self.clock = pygame.time.Clock()

self.asteroids = []
self.spaceship = Spaceship((400, 300))

for _ in range(6):
while True:
position = get_random_position(self.screen)
if (
position.distance_to(self.spaceship.position)
> self.MIN_ASTEROID_DISTANCE
):
break

self.asteroids.append(Asteroid(position))
```

Σε έναν βρόχο, ο κώδικάς μας ελέγχει εάν η θέση ενός αστεροειδούς είναι μεγαλύτερη από την ελάχιστη απόσταση με τον αστεροειδή. Εάν όχι, τότε ο βρόχος εκτελείται ξανά μέχρι να βρεθεί μια τέτοια θέση.

Αν εκτελέσουμε ξανά το πρόγραμμα θα δούμε ότι κανένας από τους αστεροειδείς δεν επικαλύπτεται με το διαστημόπλοιο.

37.1.4 Μετακίνηση των αστεροϊδών – Βήμα 7.2

Ας ξεκινήσουμε δημιουργώντας μια μέθοδο που ονομάζεται

get_random_velocity() στο αρχείο space_rocks/utils.py:

```
def get_random_velocity(min_speed, max_speed):
speed = random.randint(min_speed, max_speed)
angle = random.randrange(0, 360)
return Vector2(speed, 0).rotate(angle)
```

Η μέθοδος θα δημιουργήσει μια τυχαία τιμή μεταξύ min_speed και max_speed και μια τυχαία γωνία μεταξύ 0 και 360 μοιρών.

Στη συνέχεια, θα δημιουργήσει ένα διάνυσμα με αυτήν την τιμή, που θα περιστρέφεται κατά αυτή τη γωνία.

Επειδή η ταχύτητα του αστεροειδούς πρέπει να είναι τυχαία ανεξάρτητα από το πού τοποθετείται, ας χρησιμοποιήσουμε αυτήν τη μέθοδο απευθείας στην κλάση Asteroid.

Ξεκινάμε με την ενημέρωση των import στο αρχείο space_rocks/models.py:

```
from pygame.math import Vector2
from pygame.transform import rotozoom
from utils import get_random_velocity, load_sprite,
wrap_position
```

Σημειώστε ότι ορίζουμε μια τυχαία θέση σε ένα μέρος και την τυχαία ταχύτητα μας κάπου αλλού. Αυτό συμβαίνει επειδή η θέση θα πρέπει να είναι τυχαία μόνο για τους έξι αστεροειδείς με τους οποίους ξεκινάμε, επομένως τοποθετείται στο αρχείο space_rocks/game.py, όπου έχει αρχικοποιηθεί το παιχνίδι.

Ωστόσο, η ταχύτητα είναι τυχαία για κάθε αστεροειδή, οπότε την ορίζουμε στον κατασκευαστή της κλάσης Asteroid.

Στη συνέχεια χρησιμοποιούμε τη νέα μέθοδο στον κατασκευαστή της κλάσης Asteroid:

```
def __init__(self, position):
super().__init__(
position, load_sprite("asteroid"), get_random_velocity(1, 3)
)
```

Σημειώστε ότι η μέθοδος χρησιμοποιεί την ελάχιστη τιμή του 1. Αυτό συμβαίνει γιατί ο αστεροειδής πρέπει πάντα να κινείται, τουλάχιστον λίγο.

Εκτελούμε ξανά το παιχνίδι μας για να δούμε κινούμενους αστεροειδείς και βλέπουμε πως πράγματι οι αστεροειδείς κινούνται.

Μπορούμε επίσης να μετακινήσουμε το διαστημόπλοιο γύρω από την οθόνη. Δυστυχώς, όταν συναντά έναν αστεροειδή, δεν συμβαίνει τίποτα.

Ήρθε η ώρα να προσθέσουμε μερικές συγκρούσεις.

37.1.5 Δημιουργία συγκρούσεων – Βήμα 7.3

Ένα πολύ σημαντικό μέρος αυτού του παιχνιδιού είναι η πιθανότητα να καταστραφεί το διαστημόπλοιό μας από τη σύγκρουση με έναν αστεροειδή. Μπορούμε να ελέγξουμε τις συγκρούσεις χρησιμοποιώντας τη μέθοδο GameObject.collides_with() που εισήχθη προηγουμένως .

Το μόνο που χρειάζεται για να το κάνουμε είναι αυτή η μέθοδος για κάθε αστεροειδή.

Ας επεξεργαστούμε τη μέθοδο _process_game_logic() στην κλάση

SpaceRocks ως εξής:

```
def _process_game_logic(self):
for game_object in self._get_game_objects():
game_object.move(self.screen)
```

```
if self.spaceship:
for asteroid in self.asteroids:
if asteroid.collides_with(self.spaceship):
self.spaceship = None
break
```

Εάν κάποιος από τους αστεροειδείς συγκρουστεί με το διαστημόπλοιο, τότε το διαστημόπλοιο καταστρέφεται. Σε αυτό το παιχνίδι, θα το αντιπροσωπεύσουμε ορίζοντας την self.spaceship ίση με None.

Τώρα που είναι δυνατό το διαστημόπλοιο να έχει τιμή None, είναι σημαντικό να ενημερώσουμε την _get_game_objects() στην κλάση SpaceRocks για να αποφύγουμε την προσπάθεια απόδοσης ή μετακίνησης ενός κατεστραμμένου διαστημοπλοίου :

```
def _get_game_objects(self):
game_objects = [*self.asteroids]

if self.spaceship:
game_objects.append(self.spaceship)

return game_objects
```

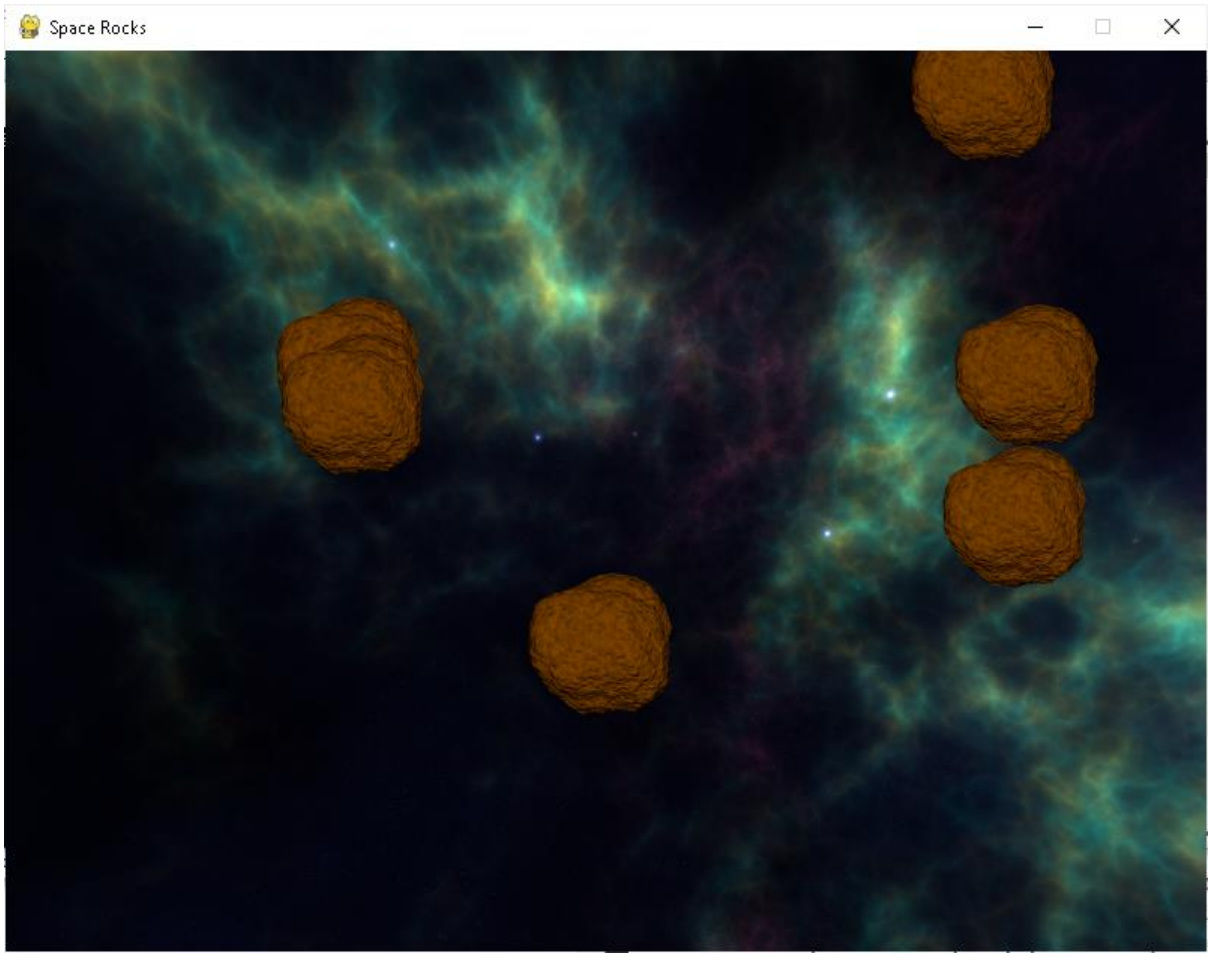
Το ίδιο ισχύει και για τον χειρισμό εισόδου:

```
def _handle_input(self):
for event in pygame.event.get():
if event.type == pygame.QUIT or (
event.type == pygame.KEYDOWN and event.key ==
pygame.K_ESCAPE
):
quit()

is_key_pressed = pygame.key.get_pressed()

if self.spaceship:
if is_key_pressed[pygame.K_RIGHT]:
```

```
self.spaceship.rotate(clockwise=True)
elif is_key_pressed[pygame.K_LEFT]:
self.spaceship.rotate(clockwise=False)
if is_key_pressed[pygame.K_UP]:
self.spaceship.accelerate()
```

Μπορούμε να εκτελέσουμε το παιχνίδι μας τώρα και να δούμε ότι το διαστημόπλοιο εξαφανίζεται μετά τη σύγκρουση με έναν αστεροειδή.

Το διαστημόπλοιό μας μπορεί τώρα να πετάξει και να καταστραφεί όταν συγκρούεται με αστεροειδείς.



Είμαστε έτοιμοι τώρα να καταστήσουμε δυνατή την καταστροφή και των αστεροειδών, αφού πρώτα δημιουργήσουμε τις σφαίρες.

37.1.6 Δημιουργία σφαιρών – Βήμα 8.1

Σε αυτό το σημείο, έχουμε μερικούς τυχαία τοποθετημένους και κινούμενους αστεροειδείς και ένα διαστημόπλοιο που μπορεί να κινηθεί και να τους αποφύγει. Στο τέλος αυτού του βήματος, το διαστημόπλοιό μας θα μπορεί επίσης να αμυνθεί εκτοξεύοντας σφαίρες.

Δημιουργία Κλάσης

Ας ξεκινήσουμε προσθέτοντας μια εικόνα μιας κουκκίδας στο assets/sprites.

Στη συνέχεια, ας επεξεργαστούμε το αρχείο space_rocks/models.py δημιουργώντας μια κλάση που ονομάζεται Bullet που κληρονομεί από τη GameObject:

```
class Bullet(GameObject):
def __init__(self, position, velocity):
super().__init__(position, load_sprite("bullet"), velocity)
```

Όπως και πριν, αυτό θα καλέσει τον κατασκευαστή της GameObject μόνο με ένα συγκεκριμένο sprite. Ωστόσο, αυτή τη φορά η ταχύτητα θα είναι ένα απαιτούμενο όρισμα επειδή μια σφαίρα πρέπει να κινηθεί.

Στη συνέχεια, θα πρέπει να προσθέσουμε έναν τρόπο για να παρακολουθούμε τις σφαίρες, παρόμοιο με αυτό που κάναμε για τους

αστεροειδείς.

Επεξεργαζόμαστε τον κατασκευαστή της κλάσης SpaceRocks στο αρχείο space_rocks/game.py:

```
def __init__(self):
self._init_pygame()
self.screen = pygame.display.set_mode((800, 600))
self.background = load_sprite("space", False)
self.clock = pygame.time.Clock()

self.asteroids = []
```

```
self.bullets = []
self.spaceship = Spaceship((400, 300))

for _ in range(6):
while True:
position = get_random_position(self.screen)
if (
position.distance_to(self.spaceship.position)

16
> self.MIN_ASTEROID_DISTANCE
):
break

self.asteroids.append(Asteroid(position))
```

Οι κουκκίδες πρέπει να αντιμετωπίζονται με τον ίδιο τρόπο με τα άλλα αντικείμενα του παιχνιδιού, επομένως ας επεξεργαστούμε τη μέθοδο _get_game_object() στην κλάση SpaceRocks:

```
def _get_game_objects(self):
game_objects = [*self.asteroids, *self.bullets]

if self.spaceship:
game_objects.append(self.spaceship)

return game_objects
```

Η λίστα με τις κουκκίδες υπάρχει, αλλά είναι κενή προς το παρόν.

Μπορούμε να το διορθώσουμε αυτό.

37.1.7 Πυροβολώντας μια σφαίρα – Βήμα 8.2

Υπάρχει ένα μικρό θέμα με την εκτόξευση των σφαιρών. Οι κουκκίδες αποθηκεύονται στο κύριο αντικείμενο του παιχνιδιού, που αντιπροσωπεύεται από την κλάση SpaceRocks.

Ωστόσο, η λογική της βολής θα πρέπει να καθορίζεται από το διαστημόπλοιο. Είναι το διαστημόπλοιο που ξέρει πώς να δημιουργεί μια νέα σφαίρα, αλλά είναι το παιχνίδι που αποθηκεύει και αργότερα ζωντανεύει τις σφαίρες.

Η κλάση Spaceship χρειάζεται έναν τρόπο να ενημερώνει την κλάση

SpaceRocks ότι έχει δημιουργηθεί μια κουκκίδα και πρέπει να παρακολουθείται.

Για να το διορθώσουμε αυτό, μπορούμε να προσθέσουμε μια συνάρτηση επανάκλησης στην κλάση Spaceship. Αυτή η λειτουργία θα παρέχεται από την κλάση SpaceRocks όταν αρχικοποιηθεί το διαστημόπλοιο.

Κάθε φορά που το διαστημόπλοιο θα δημιουργεί μια σφαίρα, θα αρχικοποιεί ένα Bullet αντικείμενο και στη συνέχεια θα καλεί την επανάκληση.

Η επανάκληση θα προσθέτει τη κουκκίδα στη λίστα όλων των κουκκίδων που είναι αποθηκευμένες στο παιχνίδι.

Ας ξεκινήσουμε προσθέτοντας μια επιστροφή κλήσης στον κατασκευαστή της κλάσης Spaceship στο αρχείο space_rocks/models.py:

```
def __init__(self, position, create_bullet_callback):
self.create_bullet_callback = create_bullet_callback
# Κάνουμε ένα αντίγραφο του αρχικού διανύσματος UP
self.direction = Vector2(UP)

super().__init__(position, load_sprite("spaceship"),
Vector2(0))
```

Θα χρειαστούμε επίσης την τιμή της ταχύτητας της σφαίρας:

```
class Spaceship(GameObject):
MANEUVERABILITY = 3
ACCELERATION = 0.25
BULLET_SPEED = 3
```

Στη συνέχεια, δημιουργούμε μια μέθοδο που ονομάζεται shoot() στην

κλάση Spaceship:

```
def shoot(self):
bullet_velocity = self.direction * self.BULLET_SPEED +
self.velocity
bullet = Bullet(self.position, bullet_velocity)
self.create_bullet_callback(bullet)
```

Ξεκινάμε υπολογίζοντας την ταχύτητα της σφαίρας. Η σφαίρα εκτοξεύεται πάντα προς τα εμπρός, επομένως χρησιμοποιούμε την κατεύθυνση του διαστημοπλοίου πολλαπλασιασμένη με την ταχύτητα της σφαίρας. Επειδή το διαστημόπλοιο δεν μένει απαραίτητα ακίνητο, προσθέτουμε την ταχύτητά του στην ταχύτητα της σφαίρας. Με αυτόν τον τρόπο, μπορούμε να δημιουργήσουμε σφαίρες υψηλής ταχύτητας εάν το διαστημόπλοιο κινείται πολύ γρήγορα.

Στη συνέχεια, δημιουργούμε ένα στιγμιότυπο της κλάσης Bullet στην ίδια θέση με το διαστημόπλοιο, χρησιμοποιώντας την ταχύτητα που μόλις υπολογίστηκε.

Τέλος, η κουκκίδα προστίθεται σε όλες τις κουκκίδες του παιχνιδιού

χρησιμοποιώντας τη μέθοδο επανάκλησης.

Τώρα ας προσθέσουμε την επανάκληση στο διαστημόπλοιο όταν δημιουργηθεί. Οι κουκκίδες αποθηκεύονται ως λίστα και το μόνο πράγμα που πρέπει να κάνει η επανάκληση είναι να προσθέσει νέα στοιχεία σε αυτήν τη λίστα.

Επομένως, η μέθοδος append() πρέπει να κάνει τη δουλειά. Ας επεξεργαστούμε τον κατασκευαστή της κλάσης SpaceRocks στο αρχείο

space_rocks/game.py:

```
def __init__(self):
self._init_pygame()
self.screen = pygame.display.set_mode((800, 600))
self.background = load_sprite("space", False)
self.clock = pygame.time.Clock()

self.asteroids = []
self.bullets = []
self.spaceship = Spaceship((400, 300), self.bullets.append)

for _ in range(6):
while True:
position = get_random_position(self.screen)
if (
position.distance_to(self.spaceship.position)
> self.MIN_ASTEROID_DISTANCE
):
break

self.asteroids.append(Asteroid(position))
```

Το τελευταίο πράγμα που πρέπει να προσθέσουμε είναι ο χειρισμός εισόδου. Η κουκκίδα πρέπει να δημιουργείται μόνο όταν πατηθεί το Space, ώστε να μπορούμε να χρησιμοποιήσουμε τον βρόχο συμβάντος.

Η σταθερά για το πλήκτρο Space είναι pygame.K_SPACE.

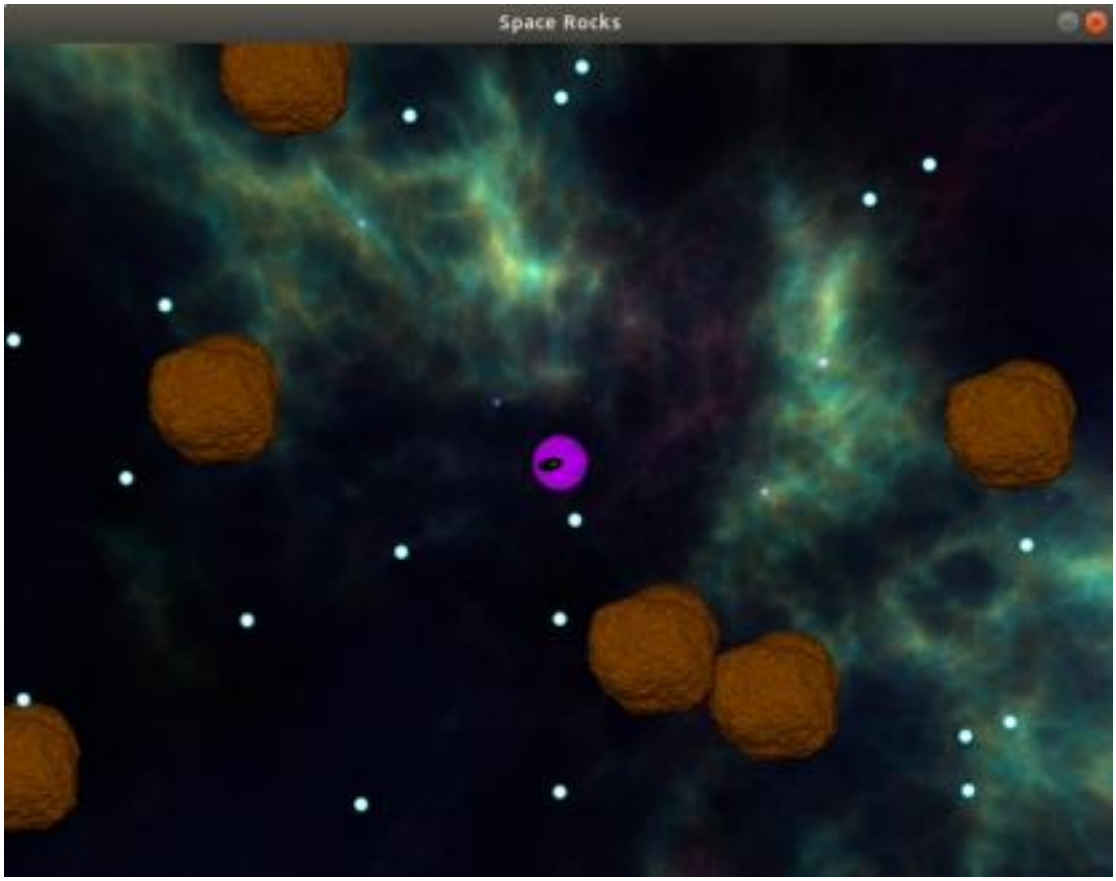
Τροποποιούμε τη μέθοδο _handle_input() στην κλάση SpaceRocks:

```
def _handle_input(self):
for event in pygame.event.get():
if event.type == pygame.QUIT or (
event.type == pygame.KEYDOWN and event.key ==
pygame.K_ESCAPE
):
quit()
elif (
self.spaceship
and event.type == pygame.KEYDOWN
and event.key == pygame.K_SPACE
):
```

```
self.spaceship.shoot()

is_key_pressed = pygame.key.get_pressed()

if self.spaceship:
if is_key_pressed[pygame.K_RIGHT]:
self.spaceship.rotate(clockwise=True)
elif is_key_pressed[pygame.K_LEFT]:
self.spaceship.rotate(clockwise=False)
if is_key_pressed[pygame.K_UP]:
self.spaceship.accelerate()
```

Σημειώνουμε ότι ο νέος χειρισμός εισόδου ελέγχει επίσης εάν το διαστημόπλοιο υπάρχει. Διαφορετικά, θα μπορούσαμε να αντιμετωπίσουμε σφάλματα όταν θα προσπαθούσαμε να καλέσουμε τη shoot() σε ένα αντικείμενο None.

Εκτελούμε το παιχνίδι μας τώρα και πυροβολούμε μερικές σφαίρες:



Το διαστημόπλοιό μας μπορεί επιτέλους να πυροβολήσει! Ωστόσο, οι σφαίρες δεν φεύγουν από την οθόνη, κάτι που μπορεί να είναι πρόβλημα.

Προς το παρόν, όλα τα αντικείμενα του παιχνιδιού συγκρατούνται στην οθόνη. Αυτό περιλαμβάνει και τις σφαίρες. Ωστόσο, λόγω αυτού του θέματος, η οθόνη γεμίζει γρήγορα με σφαίρες που πετούν προς όλες τις κατευθύνσεις. Αυτό μπορεί να κάνει το παιχνίδι υπερβολικά εύκολο!

Μπορούμε να λύσουμε αυτό το ζήτημα απενεργοποιώντας το περιτύλιγμα (wrapping) μόνο για τις κουκκίδες.

Αντικαταθιστούμε την move() στην κλάση Bullet στο αρχείο space_rocks/models.py ως εξής:

```
def move(self, surface):
 self.position = self.position + self.velocity
```

Με αυτόν τον τρόπο οι σφαίρες δεν μένουν στην οθόνη.

Ωστόσο, επίσης δεν θα καταστρέφονται. Αντίθετα, θα συνεχίζουν να πετούν στην απέραντη άβυσσο του σύμπαντος.

Σύντομα, η λίστα με τις κουκκίδες μας θα περιέχει χιλιάδες στοιχεία και όλα αυτά θα υποβάλλονται σε επεξεργασία σε κάθε καρέ, με αποτέλεσμα τη μείωση της απόδοσης του παιχνιδιού μας.

Για να αποφύγουμε αυτήν την κατάσταση, το παιχνίδι μας θα πρέπει να αφαιρεί τις σφαίρες αμέσως μόλις φύγουν από την οθόνη.

Ενημερώνουμε λοιπόν τη μέθοδο _process_game_logic() της κλάσης SpaceRocks στο αρχείο space_rocks/game.py:

```
def _process_game_logic(self):
22
 for game_object in self._get_game_objects():
 game_object.move(self.screen)
 if self.spaceship:
```

```
 for asteroid in self.asteroids:
 if asteroid.collides_with(self.spaceship):
 self.spaceship = None
 break
 for bullet in self.bullets[:]:
 if not
self.screen.get_rect().collidepoint(bullet.position):  self.
bullets.remove(bullet)
```

Παρατηρήστε ότι αντί να χρησιμοποιήσουμε την αρχική λίστα, self.bullets, δημιουργούμε ένα αντίγραφό της χρησιμοποιώντας τη μέθοδο του slicing, δηλ. γράφουμε self.bullets[:] στη γραμμή 11.

Αυτό συμβαίνει επειδή η αφαίρεση στοιχείων από μια λίστα ενώ διατρέχεται μπορεί να προκαλέσει σφάλματα.

Οι επιφάνειες στο Pygame έχουν μια μέθοδο get_rect()που επιστρέφει ένα ορθογώνιο που αντιπροσωπεύει την περιοχή τους.

Αυτό το ορθογώνιο, με τη σειρά του, έχει μια μέθοδο collidepoint() που επιστρέφει True εάν ένα σημείο περιλαμβάνεται στο ορθογώνιο και False διαφορετικά.

Χρησιμοποιώντας αυτές τις δύο μεθόδους, μπορούμε να ελέγξουμε εάν η κουκκίδα έχει φύγει από την οθόνη και, αν ναι, να την αφαιρέσουμε από τη λίστα.

Ένα κρίσιμο στοιχείο από τις σφαίρες μας εξακολουθεί να λείπει: η ικανότητα να καταστρέφουμε αστεροειδείς! Θα το διορθώσουμε παρακάτω.

37.1.8 Καταστροφή αστεροϊδών – Βήμα 8.3

Ενημερώνουμε τη μέθοδο _process_game_logic() της κλάσης SpaceRocks ως εξής:

```
def _process_game_logic(self):
 for game_object in self._get_game_objects():
 game_object.move(self.screen)
 if self.spaceship:
 for asteroid in self.asteroids:
 if asteroid.collides_with(self.spaceship):
 self.spaceship = None
 break
 for bullet in self.bullets[:]:
 for asteroid in self.asteroids[:]:
 if asteroid.collides_with(bullet):
 self.asteroids.remove(asteroid)
 self.bullets.remove(bullet)
 break
 for bullet in self.bullets[:]:
 if not
self.screen.get_rect().collidepoint(bullet.position):
 self.bullets.remove(bullet)
```

Τώρα, κάθε φορά που ανιχνεύεται μια σύγκρουση μεταξύ μιας σφαίρας και ενός αστεροειδή, και οι δύο θα αφαιρούνται από το παιχνίδι.

Παρατηρήστε ότι, όπως και πριν στον βρόχο κουκκίδων, δεν χρησιμοποιούμε τις αρχικές λίστες εδώ. Αντίθετα, δημιουργούμε αντίγραφα χρησιμοποιώντας slicing - [:] στις γραμμές 11 και 12.

Εάν τρέχουμε το παιχνίδι μας τώρα και στοχεύουμε καλά όταν πυροβολούμε, τότε θα μπορούμε να καταστρέψουμε μερικούς αστεροειδείς. Το δοκιμάζουμε!

Κάποιοι αστεροειδείς καταστράφηκαν .

Το διαστημόπλοιό μας μπορεί επιτέλους να προστατευτεί! Ωστόσο, υπάρχουν μόνο έξι μεγάλοι στόχοι στο παιχνίδι. Στη συνέχεια, θα το κάνουμε λίγο πιο δύσκολο.

Σε αυτό το σημείο, έχουμε ένα παιχνίδι με διαστημόπλοιο, αστεροειδείς και σφαίρες. Στο τέλος αυτού του βήματος, οι αστεροειδείς μας θα χωρίζονται όταν χτυπηθούν από σφαίρα.

Ένας μεγάλος αστεροειδής θα διαιρείται σε δύο μεσαίους, ένας μεσαίος θα διαιρείται σε δύο μικρούς και ένας μικρός θα εξαφανίζεται.

Το μέγεθος ενός αστεροειδούς θα αντιπροσωπεύεται από έναν αριθμό:

Μέγεθος αστεροειδούς	**Τύπος αστεροειδούς**
3	Μεγάλος αστεροειδής
2	Μεσαίος αστεροειδής
1	Μικρός αστεροειδής

Κάθε φορά που χτυπιέται ένας αστεροειδής, θα παράγει δύο αστεροειδείς με μικρότερο μέγεθος. Η εξαίρεση είναι ένας αστεροειδής με μέγεθος 1, ο οποίος δεν πρέπει να παράγει νέους αστεροειδείς.

Το μέγεθος ενός αστεροειδούς θα καθορίσει επίσης το μέγεθος του sprite του, και κατά συνέπεια την ακτίνα του. Με άλλα λόγια, οι αστεροειδείς θα κλιμακωθούν ως εξής:

Μέγεθος αστεροειδούς	**Αστεροειδής κλίμακα**	**Περιγραφή**
3	1	Το προεπιλεγμένο sprite και η ακτίνα του
2	0,5	Το μισό του προεπιλεγμένου sprite και της ακτίνας

1	0,25	Το ένα τέταρτο του προεπιλεγμένου sprite και της ακτίνας

Αυτό μπορεί να φαίνεται λίγο περίπλοκο, αλλά μπορούμε να το κάνουμε με λίγες μόνο γραμμές κώδικα.

Ξαναγράφουμε τον κατασκευαστή της κλάσης Asteroid στο αρχείο

```
space_rocks/models.py:
def __init__(self, position, size=3):
 self.size = size
 size_to_scale = {
 3: 1,
 2: 0.5,
 1: 0.25,
 }
 scale = size_to_scale[size]
 sprite = rotozoom(load_sprite("asteroid"), 0, scale)
 super().__init__(
 position, sprite, get_random_velocity(1, 3)
 )
```

Αυτή η μέθοδος θα εκχωρήσει ένα μέγεθος σε έναν αστεροειδή, χρησιμοποιώντας την προεπιλεγμένη τιμή 3, η οποία αντιπροσωπεύει έναν μεγάλο αστεροειδή. Θα κλιμακώσει επίσης το αρχικό sprite χρησιμοποιώντας την rotozoom().

Το έχουμε χρησιμοποιήσει στο παρελθόν για την περιστροφή του διαστημπλοίου . Αυτή η μέθοδος μπορεί επίσης να χρησιμοποιηθεί για την κλιμάκωση εάν η γωνία είναι 0 και η κλίμακα είναι οτιδήποτε άλλο εκτός από 0. Σε αυτό το παράδειγμα, ο πίνακας size_to_scale αναζήτησης περιέχει κλίμακες για διαφορετικά μεγέθη:

Μέγεθος	**Κλίμακα**
3	1
2	0,5

1	0,25

Τέλος, περνάμε το scaled sprite στον κατασκευαστή της κλάσης GameObject, ο οποίος θα φροντίσει να υπολογίσει την ακτίνα με βάση το νέο μέγεθος εικόνας.

Η νέα μας λογική απαιτεί έναν αστεροειδή για να μπορεί να δημιουργήσει νέους αστεροειδείς. Η κατάσταση είναι παρόμοια με το διαστημόπλοιο και τις σφαίρες, επομένως μπορούμε να χρησιμοποιήσουμε μια παρόμοια λύση: μια μέθοδο επανάκλησης.

Ενημερώνουμε ξανά τον κατασκευαστή της κλάσης Asteroid:

```
def __init__(self, position, create_asteroid_callback,
size=3):  self.create_asteroid_callback =
create_asteroid_callback  self.size = size
 size_to_scale = {
 3: 1,
 2: 0.5,
 1: 0.25,
 }
 scale = size_to_scale[size]
 sprite = rotozoom(load_sprite("asteroid"), 0, scale)
 super().__init__(
 position, sprite, get_random_velocity(1, 3)
 )
```

Τώρα μπορούμε να δημιουργήσουμε μια μέθοδο που ονομάζεται split() στην ίδια κλάση:

```
def split(self):
 if self.size > 1:
 for _ in range(2):
```

```
 asteroid = Asteroid(
 self.position, self.create_asteroid_callback, self.size -
1  )
 self.create_asteroid_callback(asteroid)
```

Αυτό θα δημιουργήσει δύο νέους αστεροειδείς στην ίδια θέση με τον τρέχοντα. Κάθε ένας από αυτούς θα έχει λίγο μικρότερο μέγεθος. Αυτή η λογική θα συμβεί μόνο εάν ο τρέχων αστεροειδής είναι μεσαίος ή μεγάλος.

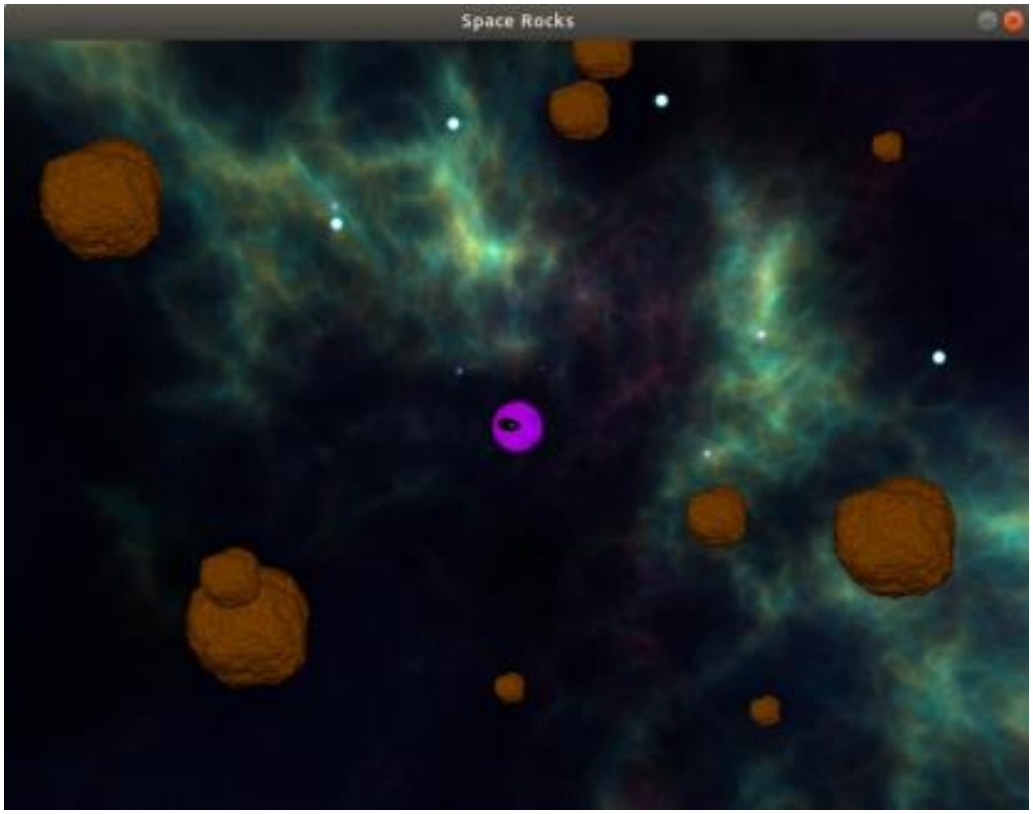
Τώρα μπορούμε να προσθέσουμε την επιστροφή κλήσης σε κάθε νεοδημιουργημένο αστεροειδή στον κατασκευαστή της κλάσης SpaceRocks.

Όπως και στην περίπτωση του διαστημοπλοίου , θα χρησιμοποιήσουμε τη μέθοδο append() της σωστής λίστας:

```
def __init__(self):
 self._init_pygame()
 self.screen = pygame.display.set_mode((800, 600))
 self.background = load_sprite("space", False)
 self.clock = pygame.time.Clock()
 self.asteroids = []
 self.bullets = []
 self.spaceship = Spaceship((400, 300),
self.bullets.append)
 for _ in range(6):
 while True:
 position = get_random_position(self.screen)
 if (
 position.distance_to(self.spaceship.position)
29
 > self.MIN_ASTEROID_DISTANCE
 ):
 break
 self.asteroids.append(Asteroid(position,
self.asteroids.append))
Ας θυμηθούμε να καλέσουμε τη split() όταν ένας αστεροειδής
χτυπηθεί από σφαίρα.
```

```
Ενημερώνουμε τη μέθοδο _process_game_logic() της
κλάσης  SpaceRocks:
def _process_game_logic(self):
 for game_object in self._get_game_objects():
 game_object.move(self.screen)
 if self.spaceship:
 for asteroid in self.asteroids:
 if asteroid.collides_with(self.spaceship):
 self.spaceship = None
 break
 for bullet in self.bullets[:]:
 for asteroid in self.asteroids[:]:
 if asteroid.collides_with(bullet):
 self.asteroids.remove(asteroid)
 self.bullets.remove(bullet)
 asteroid.split()
 break
 for bullet in self.bullets[:]:
 if not
self.screen.get_rect().collidepoint(bullet.position):  self.
bullets.remove(bullet)
```

Εάν εκτελέσουμε το παιχνίδι μας τώρα και καταρρίψουμε μερικούς αστεροειδείς, τότε θα παρατηρήσουμε ότι, αντί να εξαφανιστούν αμέσως, χωρίζονται σε μικρότερους:



Μόλις εφαρμόσαμε όλη τη λογική του παιχνιδιού! Το διαστημόπλοιο μπορεί να κινηθεί, καταστρέφεται μετά από σύγκρουση με έναν αστεροειδή, εκτοξεύει σφαίρες και οι αστεροειδείς χωρίζονται σε μικρότερους.

Αλλά το παιχνίδι είναι σιωπηλό μέχρι αυτή τη στιγμή.

37.1.9 Προσθήκη ήχου – Βήμα 9

Σε αυτό το σημείο, το πρόγραμμά μας εμφανίζει όλα τα αντικείμενα του παιχνιδιού και χειρίζεται τις αλληλεπιδράσεις μεταξύ τους. Στο τέλος αυτού του βήματος, το παιχνίδι θα παίζει επίσης ήχους.

Μέχρι τώρα, το διαστημόπλοιο ήταν εξοπλισμένο με ένα όπλο. Ωστόσο, αυτό το όπλο είναι εντελώς αθόρυβο.

Αυτό είναι πολύ ακριβές από την άποψη της φυσικής, αφού οι ήχοι δεν ταξιδεύουν στο ωστόσο, η χρήση ήχων στο παιχνίδι θα το έκανε πολύ πιο ενδιαφέρον.

Πρώτα, ας δημιουργήσουμε ένα φάκελο assets/sounds και ας προσθέσουμε τον ήχο λέιζερ εκεί.

Τώρα πρέπει να φορτώσουμε το αρχείο.

Στο Pygame, ένας ήχος αντιπροσωπεύεται από την κλάση Sound του module pygame.mixer.

Αν και θα χρησιμοποιήσουμε μόνο έναν ήχο σε αυτό το παιχνίδι, ίσως θέλουμε να προσθέσουμε περισσότερους αργότερα. Γι' αυτό θα δημιουργήσουμε μια βοηθητική μέθοδο για τη φόρτωση ήχων, παρόμοια με αυτή που δημιουργήσαμε για τα sprites.

Πρώτα, εισαγάγουμε την κλάση Sound στο αρχείο space_rocks/utils.py:

```
import random
from pygame.image import load
from pygame.math import Vector2
from pygame.mixer import Sound
```

Στη συνέχεια, ας δημιουργήσουμε μια μέθοδο που ονομάζεται load_sound() στο ίδιο αρχείο:

```
def load_sound(name):
 path = f"assets/sounds/{name}.wav"
 return Sound(path)
```

Η μέθοδος έχει παρόμοια λογική με την load_sprite(). Θα υποθέσει ότι ο ήχος βρίσκεται πάντα στον φάκελο assets/sounds και ότι είναι αρχείο WAV.

Τώρα μπορούμε να εισάγουμε αυτήν τη νέα μέθοδο στο αρχείο space_rocks/models.py:

```
from pygame.math import Vector2
from pygame.transform import rotozoom
from utils import get_random_velocity, load_sound,
load_sprite,  wrap_position
```

Στη συνέχεια, φορτώνουμε τον ήχο στον κατασκευαστή της κλάσης Spaceship:

```
def __init__(self, position, create_bullet_callback):
 self.create_bullet_callback =
create_bullet_callback  self.laser_sound =
load_sound("laser")
 # Make a copy of the original UP vector
 self.direction = Vector2(UP)
 super().__init__(position, load_sprite("spaceship"),
Vector2(0))
```

~~33~~

Τέλος, θα πρέπει να παίζουμε τον ήχο κάθε φορά που το διαστημόπλοιο πυροβολεί.

Ενημέρωση shoot():

```
def shoot(self):
 bullet_velocity = self.direction * self.BULLET_SPEED +
self.velocity
bullet = Bullet(self.position, bullet_velocity)
 self.create_bullet_callback(bullet)
 self.laser_sound.play()
```

Αν εκτελέσουμε τώρα το παιχνίδι θα ακούμε έναν ήχο κάθε φορά που πυροβολούμε.

Μόλις μάθαμε πώς να εργαζόμαστε με αρχεία ήχου στο Pygame!

Το μόνο που μένει είναι να εμφανιστεί ένα μήνυμα στο τέλος του παιχνιδιού.

Σε αυτό το σημείο, το παιχνίδι μας έχει σχεδόν ολοκληρωθεί, με χειρισμό εισόδου, αλληλεπιδράσεις, εικόνες, ακόμη και ήχους.

Στο τέλος αυτού του βήματος, θα εμφανίσουμε επίσης την κατάσταση του παιχνιδιού στην οθόνη.

Πολλά παιχνίδια δείχνουν κάποιες πρόσθετες πληροφορίες, τόσο κατά τη διάρκεια του παιχνιδιού όσο και μετά το τέλος του. Αυτό μπορεί να είναι ένας αριθμός εναπομεινάντων σημείων χτυπήματος, ένα επίπεδο ασπίδας, ένας αριθμός πυρομαχικών, μια συνολική βαθμολογία για την αποστολή και ούτω καθεξής. Σε αυτό το παιχνίδι, θα εμφανίσουμε την κατάσταση του παιχνιδιού.

Εάν το διαστημόπλοιο καταστραφεί από αστεροειδή, τότε το μήνυμα You lost! θα πρέπει να εμφανιστεί στην οθόνη. Αλλά αν έχουν φύγει όλοι οι αστεροειδείς και το διαστημόπλοιο είναι ακόμα εκεί, τότε θα πρέπει να εμφανίσουμε You won!

Το Pygame δεν διαθέτει προηγμένα εργαλεία για τη σχεδίαση κειμένου, πράγμα που σημαίνει περισσότερη δουλειά για τον προγραμματιστή.

Το κείμενο που αποδίδεται αντιπροσωπεύεται από μια επιφάνεια με διαφανές φόντο.

Μπορούμε να χειριστούμε αυτήν την επιφάνεια με τον ίδιο τρόπο που κάνουμε με τα sprites, για παράδειγμα χρησιμοποιώντας τη blit(). Η ίδια η επιφάνεια δημιουργείται χρησιμοποιώντας μια γραμματοσειρά.

37.1.10 Εμφάνιση κειμένου πληροφοριών – Βήμα 10.1

Δημιουργία γραμματοσειράς: Η γραμματοσειρά αντιπροσωπεύεται από την κλάση pygame.font.Font. Μπορούμε να χρησιμοποιήσουμε ένα αρχείο προσαρμοσμένης γραμματοσειράς ή μπορούμε να χρησιμοποιήσουμε την

προεπιλεγμένη γραμματοσειρά. Για αυτό το παιχνίδι, θα κάνουμε το τελευταίο.

Δημιουργία μιας επιφάνειας με το κείμενο: Αυτό γίνεται χρησιμοποιώντας την Font.render(). Προς το παρόν, γι' αυτή τη μέθοδο, αρκεί να γνωρίζουμε ότι δημιουργεί μια επιφάνεια με το κείμενο που αποδίδεται και διαφανές φόντο.

Χρήση της blit() για προσθήκη της επιφάνειας στην οθόνη:
Όπως συμβαίνει με οποιαδήποτε άλλη επιφάνεια στο Pygame, το κείμενο θα είναι ορατό μόνο εάν το φέρουμε στην οθόνη ή σε άλλη επιφάνεια που τελικά θα εμφανιστεί στην οθόνη.

Η γραμματοσειρά μας θα αποδοθεί με ένα χρώμα. Στην αρχή - αρχή, δημιουργήσαμε ένα χρώμα χρησιμοποιώντας τρεις τιμές: κόκκινο, πράσινο και μπλε. Σε αυτήν την ενότητα, θα χρησιμοποιήσουμε μια κλάση Color. Ας ξεκινήσουμε εισάγοντάς τη στο αρχείο space_rocks/utils.py:

```
import random
from pygame import Color
from pygame.image import load
from pygame.math import Vector2
from pygame.mixer import Sound
```

Στη συνέχεια, θα δημιουργήσουμε μια μέθοδο print_text() στο ίδιο αρχείο:

```
def print_text(surface, text, font,
color=Color("tomato")):  text_surface = font.render(text,
True, color)
 rect = text_surface.get_rect()
 rect.center = Vector2(surface.get_size()) / 2
 surface.blit(text_surface, rect)
```

Ας εξηγήσουμε τον κώδικα:

Η γραμμή 1 είναι η δήλωση της μεθόδου μας. Χρειάζεται μια επιφάνεια για να αποδοθεί το κείμενο, το ίδιο το κείμενο, μια γραμματοσειρά και ένα χρώμα. Η κλάση Color προσφέρει πολλά προκαθορισμένα χρώματα, τα οποία μπορούμε να βρούμε στην τεκμηρίωση της Pygame . Η μέθοδός μας θα χρησιμοποιεί ένα προεπιλεγμένο χρώμα που ονομάζεται "tomato".

Η γραμμή 2 δημιουργεί την επιφάνεια με το κείμενο χρησιμοποιώντας τη render(). Το πρώτο της όρισμα είναι το κείμενο που πρέπει να αποδοθεί. Το δεύτερο είναι μια σημαία εξομάλυνσης . Η ρύθμιση σε True θα εξομαλύνει τις άκρες του κειμένου που αποδίδεται. Το τελευταίο όρισμα είναι το χρώμα του κειμένου.

Η γραμμή 4 αφορά ένα ορθογώνιο που αντιπροσωπεύει την περιοχή της επιφάνειας με το κείμενό μας. Αυτό το ορθογώνιο είναι ένα στιγμιότυπο της κλάσης Rect και μπορεί εύκολα να μετακινηθεί και να ευθυγραμμιστεί. Μπορούμε να διαβάσουμε περισσότερα σχετικά με την ευθυγράμμιση στην τεκμηρίωση .

Η γραμμή 5 ορίζει το χαρακτηριστικό center του ορθογωνίου σε ένα σημείο στο μέσο της οθόνης. Αυτό το σημείο υπολογίζεται διαιρώντας το μέγεθος της οθόνης με το 2. Αυτή η λειτουργία διασφαλίζει ότι το κείμενό μας θα εμφανίζεται στο κέντρο της οθόνης.

Η γραμμή 7 σχεδιάζει το κείμενο στην οθόνη. Παρατηρούμε ότι αυτή τη φορά, περνάμε ένα ορθογώνιο, όχι ένα σημείο, στη blit(). Σε αυτήν την περίπτωση, η μέθοδος θα πάρει την επάνω αριστερή γωνία του δεδομένου ορθογωνίου και θα ξεκινήσει τη διαδικασία blitting εκεί.

Τώρα μπορούμε να εισάγουμε αυτήν τη μέθοδο στο αρχείο space_rocks/game.py:

```
import pygame
from models import Asteroid, Spaceship
```

```
from utils import get_random_position, load_sprite,
print_text
```

Τώρα πρέπει να δημιουργήσουμε μια γραμματοσειρά. Θα πρέπει επίσης να αποθηκεύσουμε το μήνυμα που θα εμφανιστεί.

Ας επεξεργαστούμε τον κατασκευαστή της κλάσης SpaceRocks:

```
def __init__(self):
 self._init_pygame()
 self.screen = pygame.display.set_mode((800,
600))  self.background = load_sprite("space", False)
 self.clock = pygame.time.Clock()
 self.font = pygame.font.Font(None, 64)
 self.message = ""
 self.asteroids = []
 self.bullets = []
 self.spaceship = Spaceship((400, 300),
self.bullets.append)
 for _ in range(6):
 while True:
 position = get_random_position(self.screen)
 if (
 position.distance_to(self.spaceship.position)  >
self.MIN_ASTEROID_DISTANCE
 ):
 break
 self.asteroids.append(Asteroid(position,
self.asteroids.append))
```

Ο κατασκευαστής της κλάσης Font παίρνει δύο ορίσματα:

- Το όνομα του αρχείου της γραμματοσειράς, όπου None σημαίνει ότι θα χρησιμοποιηθεί μια προεπιλεγμένη γραμματοσειρά και
- Το μέγεθος της γραμματοσειράς σε pixel

Το περιεχόμενο του μηνύματος πρέπει να ρυθμιστεί σωστά.

Όταν το διαστημόπλοιο καταστραφεί, το ρυθμίζουμε σε "You lost!". Όταν καταστραφούν όλοι οι αστεροειδείς, ρυθμίστε το σε "You won!".

Ας επεξεργαστούμε τη μέθοδο _process_game_logic() της κλάσης SpaceRocks:

```
def _process_game_logic(self):
 for game_object in self._get_game_objects():
 game_object.move(self.screen)
 if self.spaceship:
 for asteroid in self.asteroids:
 if asteroid.collides_with(self.spaceship):
 self.spaceship = None
 self.message = "You lost!"
 break
 for bullet in self.bullets[:]:
 for asteroid in self.asteroids[:]:
 if asteroid.collides_with(bullet):
 self.asteroids.remove(asteroid)
 self.bullets.remove(bullet)
 asteroid.split()
39
 break
 for bullet in self.bullets[:]:
 if not
self.screen.get_rect().collidepoint(bullet.position):  self.
bullets.remove(bullet)
 if not self.asteroids and self.spaceship:
 self.message = "You won!"
```

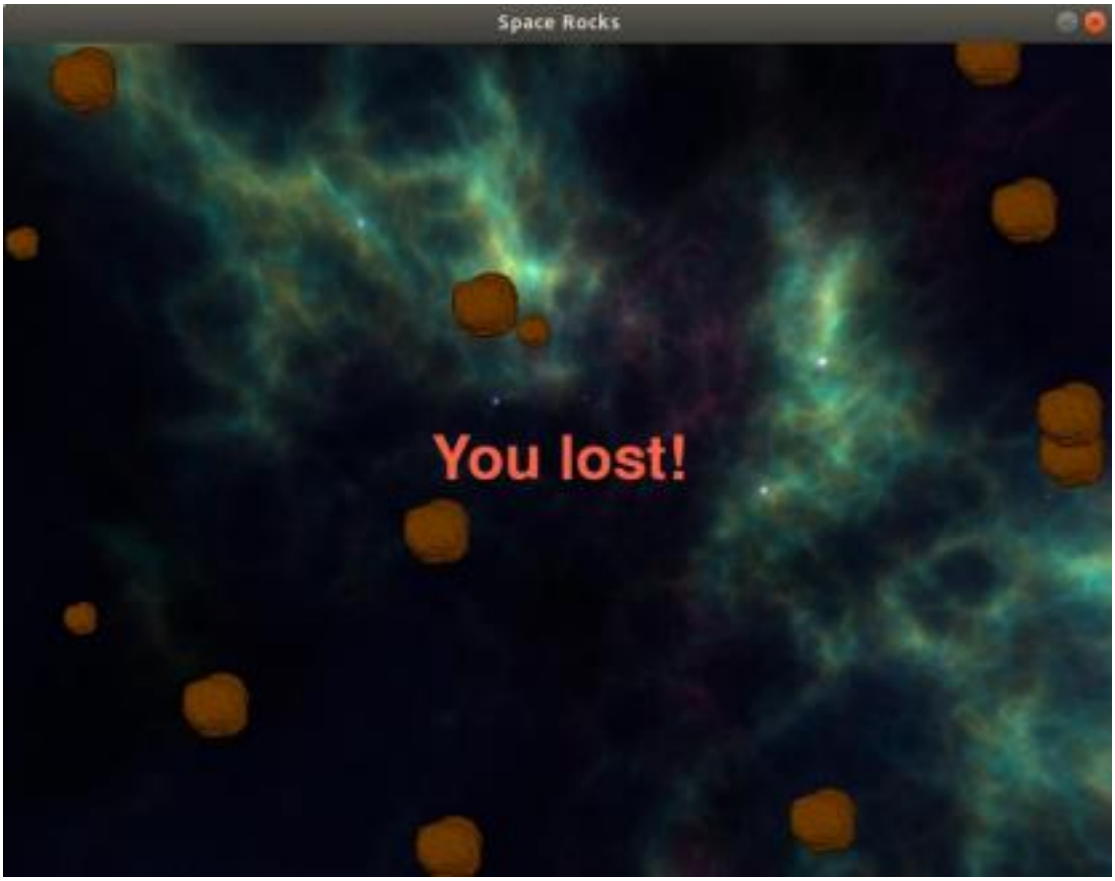
Το τελευταίο πράγμα που πρέπει να κάνουμε είναι να εμφανίσουμε το μήνυμα στην οθόνη.

Ενημερώνουμε τη μέθοδο _draw() της κλάσης SpaceRocks:

```
def _draw(self):
 self.screen.blit(self.background, (0, 0))
 for game_object in self._get_game_objects():
 game_object.draw(self.screen)
 if self.message:
 print_text(self.screen, self.message, self.font)
 pygame.display.flip()
 self.clock.tick(60)
```

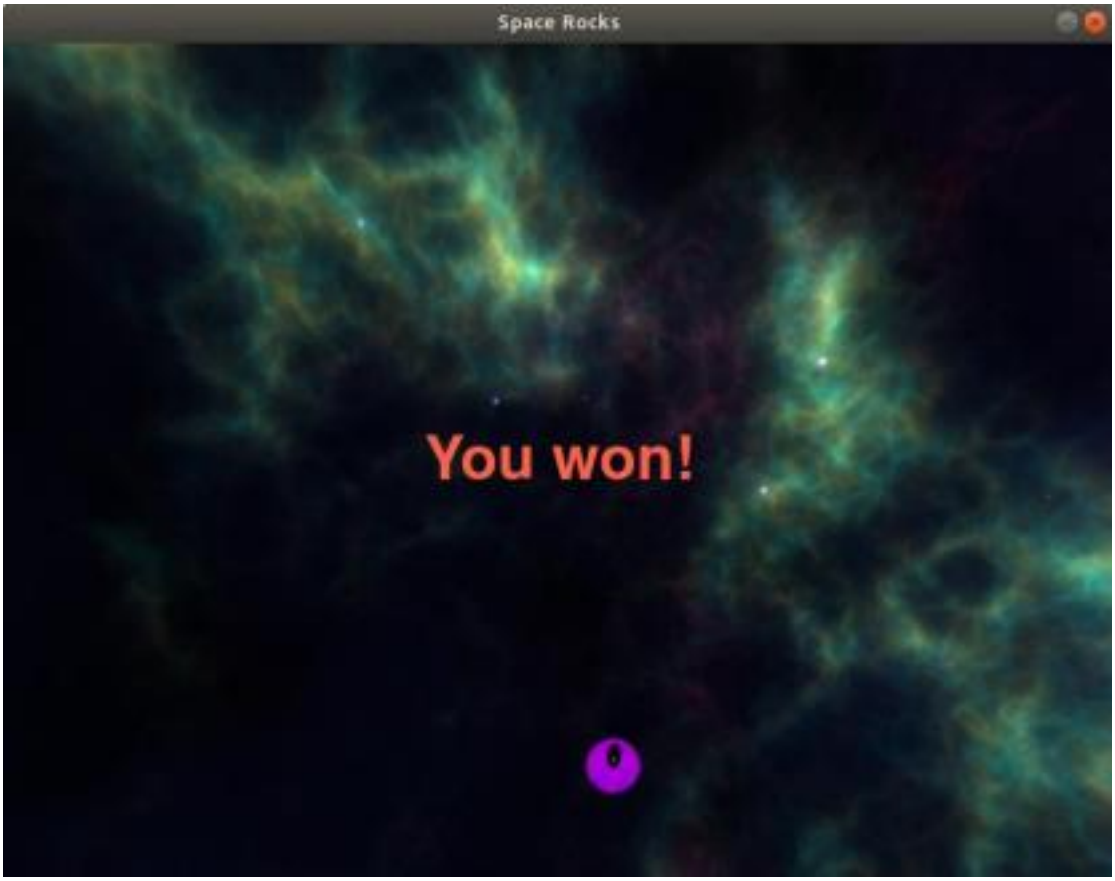
37.1.11 Ολοκλήρωση – Τελευταία μηνύματα – Βήμα 10.2

Ας το δοκιμάσουμε. Ξεκινάμε το παιχνίδι και συντρίβουμε το σκάφος μας σε έναν αστεροειδή:



Το παιχνίδι δείχνει σωστά το μήνυμα You lost!.

Τώρα αν βάλουμε περισσότερη προσπάθεια και προσπαθήσουμε να καταστρέψουμε όλους τους αστεροειδείς, θα πρέπει να δούμε μια οθόνη νίκης:



Σε αυτό το βήμα, μάθαμε πώς να εμφανίζουμε ένα μήνυμα κειμένου στην οθόνη. Αυτό ήταν το τελευταίο βήμα αυτού του project.

Το παιχνίδι μας έχει πλέον ολοκληρωθεί!

Συγχαρητήρια, μόλις δημιουργήσαμε έναν κλώνο του παιχνιδιού Asteroids με την Python!

Με το Pygame, οι γνώσεις μας για την Python μπορούν να μεταφραστούν απευθείας σε έργα ανάπτυξης παιχνιδιών.

Μάθαμε πώς να:

- Φορτώνουμε εικόνες και να τις εμφανίζουμε στην οθόνη
- Προσθέτουμε χειρισμό εισόδου στο παιχνίδι μας

- Εφαρμόζουμε τη λογική του παιχνιδιού και την ανίχνευση σύγκρουσης στην Python
  - Παίζουμε ήχους
  - Εμφανίζουμε κείμενο στην οθόνη

Πραγματοποιήσαμε όλη τη διαδικασία σχεδιασμού ενός παιχνιδιού, δόμησης αρχείων, εισαγωγής και χρήσης στοιχείων και κωδικοποίησης της λογικής. Μπορούμε να χρησιμοποιούμε όλη αυτή τη γνώση για όλα τα καταπληκτικά μελλοντικά μας έργα!

37.2.0 Άσκηση

Μελέτη των παιχνιδιών «Αστεροειδείς» και «Φιδάκι»