36. PyGame – Space Rocks Game – Σχολιασμός & Αποσφαλμάτωση

Επανάληψη κεφαλαίων 21 - 23

36.0.1 Λύσεις των προηγούμενων ασκήσεων

Άσκηση 1

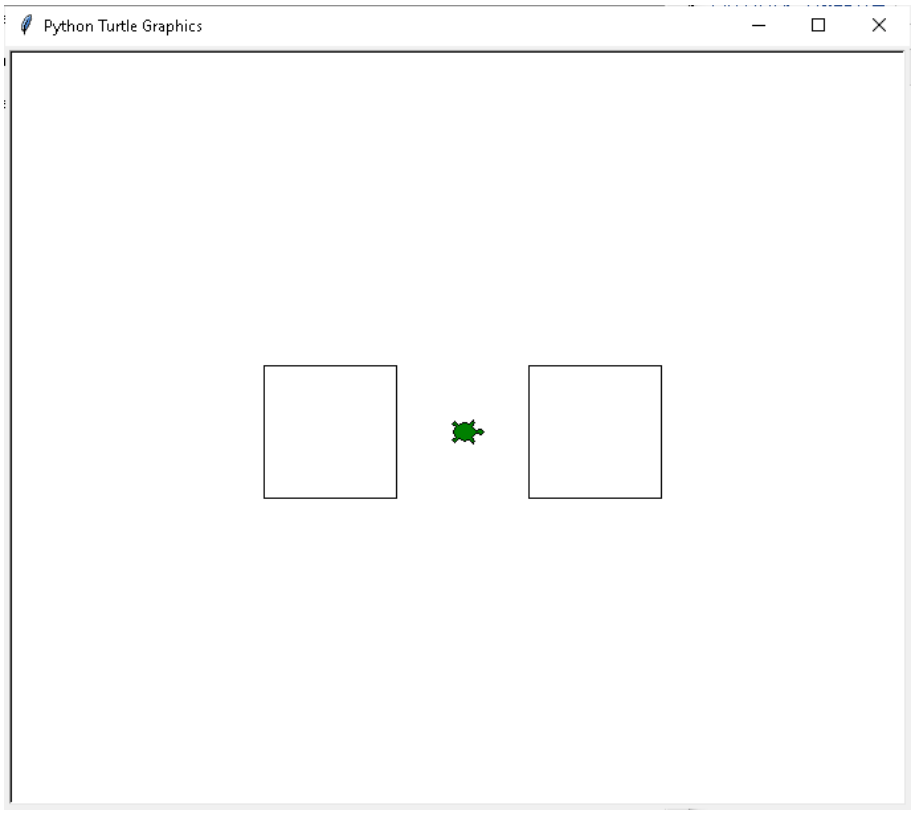
Γράψτε ένα πρόγραμμα two_squares.py που χρησιμοποιεί τη μονάδα Turtle για να σχεδιάσει δύο τετράγωνα μεγέθους 100. Σχεδιάστε τα με τέτοιο τρόπο ώστε να απέχουν 100 βήματα το ένα από το άλλο. Βεβαιωθείτε ότι το σχέδιο δεν περιέχει άλλες γραμμές. Ως τελευταίο βήμα ανάπτυξης, βεβαιωθείτε ότι το σχέδιο βρίσκεται στο κέντρο του παραθύρου.

Λύση

Δεδομένου ότι τα τετράγωνα έχουν ύψος 100 βήματα και απόσταση μεταξύ τους 100 βήματα, η κάτω αριστερή γωνία του δεξιού τετραγώνου πρέπει να βρίσκεται στο σημείο (50,-50). Θυμηθείτε να χρησιμοποιήσετε τις εντολές penup() και pendown() για να μετακινηθείτε χωρίς να σχεδιάσετε γραμμή. Στο τέλος, επιστρέφουμε στη θέση home, ώστε να είναι ξεκάθαρο που βρίσκεται το κέντρο του καμβά.

```
from turtle import *
shape('turtle')
fillcolor('green')
penup()
setposition(50,-50)
pendown()
```

```
forward(100)
left(90)
forward(100)
left(90)
forward(100)
left(90)
forward(100)
left(90)
penup()
backward(200)
pendown()
forward(100)
left(90)
forward(100)
left(90)
forward(100)
left(90)
forward(100)
left(90)
penup()
home()
```



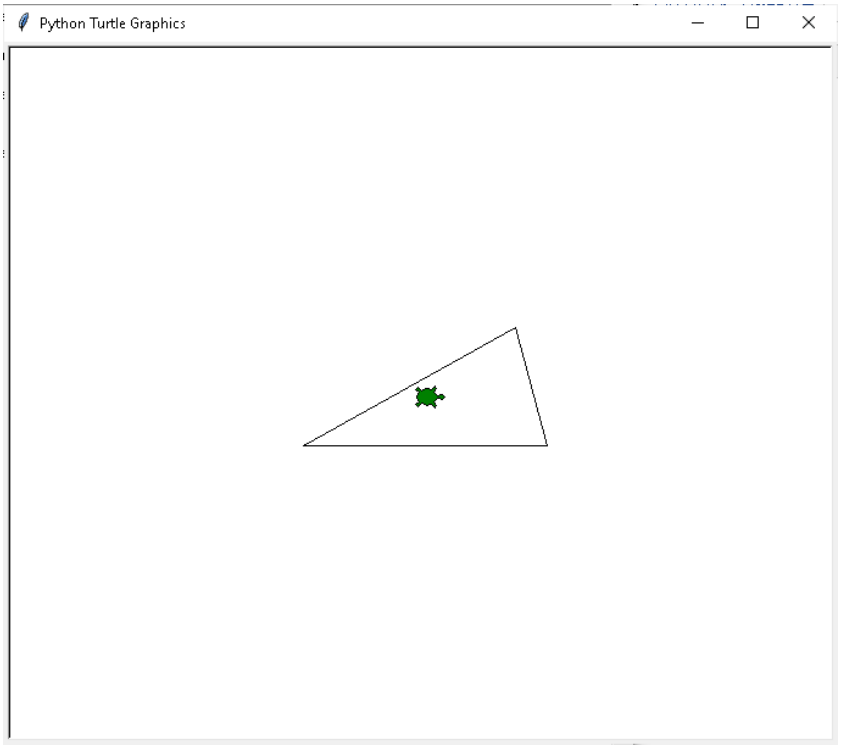
Άσκηση 2

Γράψτε ένα πρόγραμμα triangle_75.py που χρησιμοποιεί τη μονάδα Turtle για να σχεδιάσει ένα τρίγωνο που έχει μια πλευρά μήκους 200, την άλλη πλευρά μήκους 100 και η γωνία μεταξύ αυτών των δύο πλευρών είναι 75 μοίρες. Βεβαιωθείτε ότι το σχέδιο βρίσκεται (περίπου) στο κέντρο του παραθύρου Python Turtle Graphics.

Λύση

Δεδομένου ότι η οριζόντια πλευρά έχει μήκος 200 βήματα, πρέπει να μετακινηθούμε 100 βήματα προς τα αριστερά για να έχουμε το τρίγωνο κεντραρισμένο οριζόντια. Η άλλη πλευρά είναι σχεδόν κάθετη και έχει μήκος 100 βήματα, οπότε για να κεντράρουμε την εικόνα κατακόρυφα, μετακινούμαστε 40 βήματα προς τα κάτω. 45 ή ακόμα και 50 βήματα θα λειτουργούσαν επίσης.

```
from turtle import *
shape('turtle')
fillcolor('green')
penup()
setposition(-100,-40)
pendown()
forward(200)
left(105)
forward(100)
setposition(-100,-40)
penup()
home()
```



36.1.0 Η βιβλιοθήκη PyGame

Η Pygame είναι μια δημοφιλής βιβλιοθήκη Python που χρησιμοποιείται για την ανάπτυξη βιντεοπαιχνιδιών. Είναι ένας δωρεάν, ανοιχτού κώδικα wrapper πολλαπλών πλατφορμών γύρω από την απλή βιβλιοθήκη DirectMedia (SDL). Η αφαιρετικότητα (abstraction) των λειτουργιών SDL που παρέχονται από την Pygame καθιστά πολύ εύκολη την ανάπτυξη εφαρμογών πολυμέσων με την Python. Σήμερα, θα ξαναθυμηθούμε το κλασικό παιχνίδι arcade, “Asteroids”.

Θα σχολιάσουμε αναλυτικά τον κώδικα, θα προσπαθήσουμε να κάνουμε αποσφαλματώσεις καθώς και δοκιμές και γενικότερα θα αποκτήσουμε μια βαθύτερη κατανόηση για τη βιβλιοθήκη Pygame, τις κλάσεις και τον προγραμματισμό με την Python.

Μεταξύ άλλων, θα θυμηθούμε να φορτώνουμε εικόνες στην οθόνη μας, να χειριζόμαστε την είσοδο του χρήστη από το πληκτρολόγιο, να μετακινούμε αντικείμενα στην οθόνη, σύμφωνα με τη λογική του παιχνιδιού, να

ανιχνεύουμε τη σύγκρουση των αντικειμένων, να δείχνουμε κείμενο στην οθόνη και να παίζουμε ήχους.

Ήδη έχουμε ασχοληθεί με όλα τα παραπάνω, όμως τώρα, θα τα φτιάξουμε όλα με ένα τρόπο πιο «επαγγελματικό» και ακόμα πιο εξειδικευμένο στη δημιουργία παιχνιδιών, χρησιμοποιώντας τη βιβλιοθήκη pygame.

Έναν πολύ καλό οδηγό εισαγωγής στην Pygame, καθώς και γρήγορου ξεκινήματος, θα βρείτε στο Tutorialspoint.

36.1.1 Περιγραφή του Παιχνιδιού

Θα φτιάξουμε ένα κλώνο του παιχνιδιού Asteroids. Στο παιχνίδι, έχουμε

ένα σκάφος το οποίο έχει σαν σκοπό να διαλύει αστεροειδείς που έρχονται προς το μέρος του. Αν ένας αστεροειδής αγγίξει το σκάφος, τότε χάνουμε. Το παιχνίδι μας, θα είναι όπως παρακάτω.

Το παιχνίδι θα διαθέτει ένα μόνο διαστημόπλοιο. Το διαστημόπλοιο

μπορεί να περιστρέφεται, καθώς και να κινείται, αλλά και να ρίχνει

σφαίρες. Οι σφαίρες, σπάνε κάθε αστεροειδή σε μικρότερα κομμάτια.

Αν ένας αστεροειδής αγγίξει το σκάφος, τότε το σκάφος χάνεται.

36.1.2 Πώς παίζεται

Το παιχνίδι θα παίζεται με τα παρακάτω πλήκτρα:

Πλήκτρο	**Ενέργεια**
Δεξί βέλος →	Περιστροφή του σκάφους δεξιά
Αριστερό βέλος ←	Περιστροφή του σκάφους αριστερά
Επάνω βέλος ↑	Επιτάχυνση του σκάφους προς τα εμπρός

Space	Πυροβολώ
Escape	Διακοπή του παιχνιδιού

Λεπτομερέστερα θα έχουμε:

• Θα υπάρχουν έξι μεγάλοι αστεροειδείς στο παιχνίδι. Όταν μια σφαίρα χτυπήσει έναν μεγάλο αστεροειδή, θα χωριστεί σε δύο μεσαίους.

• Όταν μια σφαίρα χτυπήσει έναν μεσαίο αστεροειδή, θα χωριστεί σε δύο μικρούς.

• Ένας μικρός αστεροειδής δεν θα χωριστεί αλλά θα καταστραφεί από μια σφαίρα.

• Όταν ένας αστεροειδής συγκρουστεί με το διαστημόπλοιο, το διαστημόπλοιο θα καταστραφεί και το παιχνίδι θα τελειώσει με ήττα.

• Όταν χτυπηθούν όλοι οι αστεροειδείς, το παιχνίδι θα τελειώσει με νίκη!

36.1.3 Ανάλυση του project

Το project θα χωριστεί σε δέκα βήματα:

1. Ρύθμιση του Pygame για το έργο μας
2. Χειρισμός εισόδου του παίκτη στο παιχνίδι
3. Φόρτωση εικόνων και εμφάνισή τους στην οθόνη
4. Δημιουργία αντικειμένων παιχνιδιού με εικόνα, θέση και κάποια λογική
5. Μετακίνηση του διαστημοπλοίου
6. Μετακίνηση των αστεροειδών και ανίχνευση συγκρούσεων με το διαστημόπλοιο

7. Ρίξιμο σφαιρών και καταστροφή αστεροειδών

8. Διάσπαση των αστεροειδών σε μικρότερους

9. Αναπαραγωγή ήχων

10.Χειρισμός του τέλους του παιχνιδιού

Ας σχολιάσουμε τον παρακάτω κώδικα:

```
    def main_loop(self):
        while True:
            self._handle_input()# Διαχείριση των εισόδων από πλ/γιο και ποντίκι
            self._process_game_logic() #Όλη η λογική του παιχνιδιού, κανόνες, συγκρούσεις, νικητής ή χαμένος του παιχνιδιού
            self._draw()# Σχεδιασμός όλων των γραφικών στοιχείων
```

Τώρα, λαμβάνοντας υπόψη ότι θα επεκτείνουμε το παιχνίδι μας, αλλά και για την καλύτερη αναγνωσιμότητα του κώδικα και τη χρήση καλύτερων προγραμματιστικών πρακτικών, είναι καλή ιδέα να ενσωματώσουμε όλες αυτές τις λειτουργίες σε μια κλάση.

Η δημιουργία μιας σημαίνει ότι πρέπει να επιλέξουμε ένα όνομα για το παιχνίδι κι έτσι το ονομάζουμε “Space Rocks”.

Δημιουργούμε ένα αρχείο και το ονομάζουμε game.py. Εδώ θα βάλουμε την κύρια λειτουργία του παιχνιδιού μας.

Ο κώδικάς μας θα είναι:

36.1.4 Δημιουργία του παιχνιδιού – Βήμα 1

Αρχείο game.py

```
import pygame # Εισαγωγή της βιβλιοθήκης για να αποκτήσουμε όλες τις λειτουργίες της

class SpaceRocks: # Δημιουργία της κλάσης SpaceRocks
    def __init__(self): # Κατασκευαστής
        self._init_pygame() # Προετοιμασία του παιχνιδιού
        self.screen = pygame.display.set_mode((800, 600)) # Επιφάνεια/Πλαίσιο του παιχνιδιού

    def main_loop(self):
        while True:
            self._handle_input() # Διαχείριση των εισόδων από πλ/γιο και ποντίκι
            self._process_game_logic() # Όλη η λογική του παιχνιδιού, κανόνες, συγκρούσεις, νικητής ή χαμένος του παιχνιδιού
            self._draw() # Σχεδιασμός όλων των γραφικών στοιχείων

    def _init_pygame(self): # Προετοιμασία του παιχνιδιού
        pygame.init() # Αρχή του παιχνιδιού
        pygame.display.set_caption("Space Rocks") # Κείμενο/Όνομα του παιχνιδιού

    def _handle_input(self): # Χειρισμός εισόδων
        pass

    def _process_game_logic(self): # Χειρισμός λογικής
        pass

    def _draw(self): # Όλα τα γραφικά του παιχνιδιού
        self.screen.fill((0, 0, 255)) # Γέμισμα με ένα χρώμα - θα αντικατασταθεί με εικόνα παρακάτω
        pygame.display.flip() # Κλήση επαναδημιουργίας της οθόνης σε κάθε καρέ
```

Επίσης, χρειαζόμαστε ένα αρχείο __main__.py. Αυτό το αρχείο θα φροντίσει να δημιουργήσει ένα νέο στιγμιότυπο του παιχνιδιού και να το ξεκινήσει εκτελώντας το main_loop().

Αρχείο main.py

```
from game import SpaceRocks

if __name__ == "__main__":
    space_rocks = SpaceRocks()
    space_rocks.main_loop()
```

Έχουμε τώρα 2 αρχεία, τα game.py και __main__.py τα οποία

μπορούμε να βάλουμε σε ένα φάκελο, στο path της Python με το όνομα source_code_step_1.

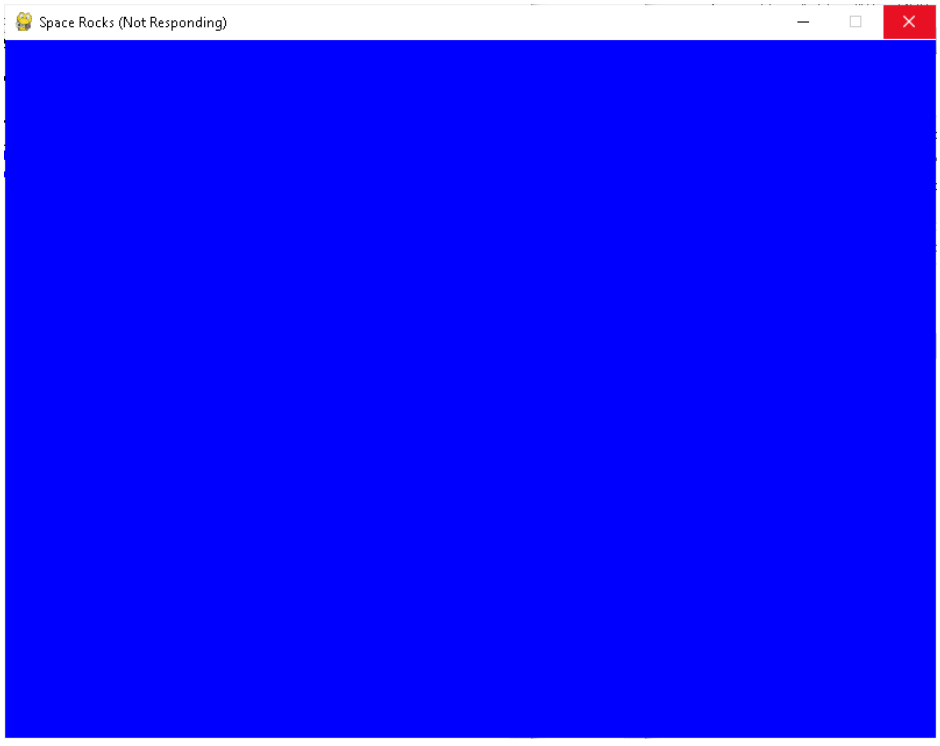
Κατόπιν, μπορούμε να τρέξουμε τον κώδικά μας από το command line

των Windows (αφού μεταβούμε στο παραπάνω directory), δίνοντας την

εντολή:

python space_rocks

και να δούμε στην οθόνη μας το παράθυρο:



36.1.5 Χειρισμός εισόδου – Βήμα 2

Σε αυτό το σημείο, έχουμε τον κύριο βρόχο του παιχνιδιού, έτοιμο να

τρέξει τη λογική του παιχνιδιού.

Η μεγαλύτερη επεξεργασία εισόδου στο Pygame γίνεται μέσα σε έναν βρόχο συμβάντων . Σε κάθε καρέ, το πρόγραμμά μας μπορεί να λάβει μια συλλογή γεγονότων που συνέβησαν από το προηγούμενο καρέ.

Αυτό περιλαμβάνει την κίνηση του ποντικιού, τα πιθανά πατήματα πλήκτρων και ούτω καθεξής.

Στη συνέχεια, ένα προς ένα, αυτά τα γεγονότα αντιμετωπίζονται. Στο Pygame, η μέθοδος απόκτησης αυτής της συλλογής είναι η pygame.event.get().

Η δήλωση που χρειαζόμαστε αυτή τη στιγμή είναι pygame.QUIT. Έτσι μπορούμε να σταματήσουμε το παιχνίδι, είτε κάνοντας κλικ στο Κλείσιμο είτε πατώντας Alt+F4 σε Windows και Linux ή Cmd+W στο macOS.

Ας τροποποιήσουμε τώρα το αρχείο game.py ξαναγράφοντας την SpaceRocks._handle_input() ως εξής:

```
def _handle_input(self):
for event in pygame.event.get():
if event.type == pygame.QUIT:
quit()
```

Μπορούμε ήδη να το δοκιμάσουμε κι εδώ είναι ευκαιρία να πούμε πως μπορούμε να συσχετίζουμε πλήκτρα του πληκτρολογίου με λειτουργίες

του παιχνιδιού, διαβάζοντας την τεκμηρίωση του pygame.

Στο παιχνίδι μας τώρα, για να κλείσουμε το παιχνίδι πατώντας Esc, θα

χρησιμοποιήσουμε το pygame.K_ESCAPE.

Ας τροποποιήσουμε ξανά τη μέθοδο _handle_input():

```
    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()
```

Τώρα το παιχνίδι μας κλείνει επίσης όταν πατάμε Esc.

Καταφέραμε να εμφανίσουμε ένα παράθυρο και να το κλείσουμε σωστά. Αλλά το παράθυρο είναι ακόμα γεμάτο με ένα μόνο χρώμα. Στη

συνέχεια, θα μάθουμε πώς να φορτώνουμε μια εικόνα και να την

εμφανίζουμε στην οθόνη.

36.1.6 Εισαγωγή εικόνων – Βήμα 3

Μέχρι εδώ, έχουμε ένα παράθυρο του παιχνιδιού που μπορούμε να το

κλείσουμε πατώντας ένα πλήκτρο. Στο τέλος αυτού του βήματος, θα

εμφανίσουμε μια εικόνα σε αυτό το παράθυρο.

Καθώς το παιχνίδι μας μεγαλώνει, είναι σημαντικό να διατηρεί τη σωστή δομή. Δημιουργούμε λοιπόν έναν φάκελο με το όνομα assets και, μέσα σε αυτόν, έναν άλλο με το όνομα sprites. Εκεί θα βάλουμε όλα τα sprites που θα χρησιμοποιεί το παιχνίδι μας.

Επίσης, δημιουργούμε ακόμα ένα αρχείο utils.py, το οποίο θα περιέχει όλες τις επαναχρησιμοποιήσιμες μεθόδους.

Μέσα σ’ αυτό το αρχείο, εφαρμόζουμε τη φόρτωση της εικόνας του φόντου μας στο παιχνίδι.

Αρχείο utils.py

```
from pygame.image import load # εισαγωγή μεθόδου load για ανάγνωση εικόνων


def load_sprite(name, with_alpha=True): # εισαγωγή εικόνας με διαφάνεια
    path = f"assets/sprites/space.png" # διαδρομή προς το αρχείο εικόνας
    loaded_sprite = load(path)

    if with_alpha:
        return loaded_sprite.convert_alpha() # χρήση της convert_alpha() για φόρτωση της εικόνας
    else:
        return loaded_sprite.convert() # χρήση της convert() για φόρτωση της εικόνας
```

Επίσης, πρέπει να αλλάξουμε το αρχείο μας game.py ώστε να φορτώσουμε το background. Θα το δούμε αυτό, στο επόμενο βήμα, μαζί με την προσθήκη κι άλλων αντικειμένων στο παιχνίδι μας.

36.1.7 Εισαγωγή αντικειμένων – Βήμα 4

Χρειαζόμαστε κι άλλα βασικά αντικείμενα στο παιχνίδι μας. Θα δημιουργήσουμε μια κλάση που αντιπροσωπεύει άλλα αντικείμενα του παιχνιδιού με δυνατότητα σχεδίασης και θα τη χρησιμοποιήσουμε για να φτιάξουμε και να χρησιμοποιήσουμε ένα διαστημόπλοιο και έναν αστεροειδή.

Κλάση GameObject

Θα ενσωματώσει κάποια γενική συμπεριφορά και δεδομένα για όλα τα άλλα αντικείμενα του παιχνιδιού. Οι κλάσεις που αντιπροσωπεύουν συγκεκριμένα αντικείμενα (όπως το διαστημόπλοιο) θα κληρονομήσουν

από αυτήν και θα την επεκτείνουν με τη δική τους συμπεριφορά και χαρακτηριστικά.

Η κλάση GameObject θα αποθηκεύσει τα ακόλουθα δεδομένα:

position: Ένα σημείο στο κέντρο του αντικειμένου στην οθόνη 2D

sprite: Μια εικόνα που χρησιμοποιείται για την εμφάνιση του αντικειμένου

radius: Μια τιμή που αντιπροσωπεύει τη ζώνη σύγκρουσης γύρω από τη θέση του αντικειμένου

velocity: Μια τιμή που χρησιμοποιείται για κίνηση

Επίσης, θα χρησιμοποιήσουμε και την έννοια διανύσματα, για τα οποία η βιβλιοθήκη Pygame, λόγω της εκτεταμένης χρήσης τους, έχει μια ειδική κλάση γι' αυτά, τη Vector2.

Αυτή η κλάση προσφέρει κάποιες προσθέτουμε λειτουργίες, όπως τον υπολογισμό της απόστασης μεταξύ των διανυσμάτων και την προσθήκη ή την αφαίρεση διανυσμάτων. Αυτά τα χαρακτηριστικά θα κάνουν τη λογική του παιχνιδιού μας πολύ πιο εύκολη στην εφαρμογή.

Στον space_rocks φάκελο, ας δημιουργήσουμε ένα νέο αρχείο που ονομάζεται models.py. Προς το παρόν, θα αποθηκεύσει την κλάση GameObject, αλλά αργότερα θα προσθέσουμε κλάσεις για αστεροειδείς, σφαίρες και το διαστημόπλοιο.

Αρχείο models.py

```
from pygame.math import Vector2


class GameObject: # Κλάση για χειρισμό των αντικειμένων του παιχνιδιού
```

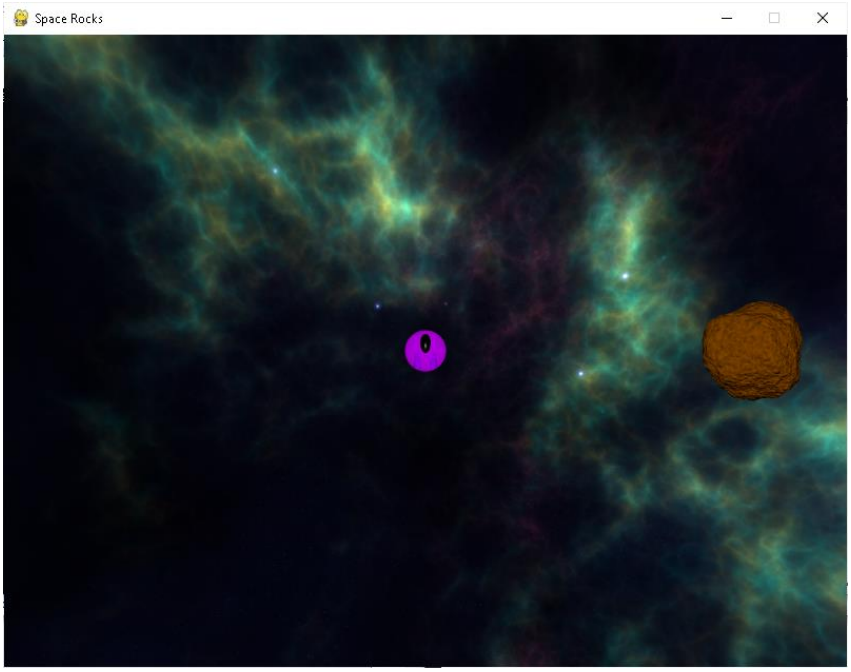
```
    def __init__(self, position, sprite, velocity): #
Κατασκευαστής με τα ανάλογα ορίσματα
        self.position = Vector2(position) # η Vector2() με
όρισμα τη θέση / κέντρο του αντικειμένου
        self.sprite = sprite # ανάθεση μεταβλητής για
εικονίδιο/αντικείμενο
        self.radius = sprite.get_width() / 2 # Ορισμός της
ακτίνας = πλάτος αντικ. / 2
        self.velocity = Vector2(velocity) # Ορισμός της
ταχύτητας και συσχέτιση με την Vector2()

    def draw(self, surface):
        blit_position = self.position - Vector2(self.radius)
# Ορισμός θέσης του αντικειμένου
        surface.blit(self.sprite, blit_position) #
Τοποθέτηση του αντικειμένου

    def move(self):
        self.position = self.position + self.velocity #
Κίνηση αντικειμένων
    def collides_with(self, other_obj): # Σύγκρουση
        distance =
self.position.distance_to(other_obj.position) # Υπολογισμός
απόστασης
        return distance < self.radius + other_obj.radius #
Επιστροφή μόνο αν υπάρχει σύγκρουση
```

Ας δούμε και το τροποποιημένο αρχείο games.py:

```
import pygame

from models import GameObject
from utils import load_sprite


class 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.spaceship = GameObject(
            (400, 300), load_sprite("spaceship"), (0, 0)
```

```
        )
        self.asteroid = GameObject((400, 300),
load_sprite("asteroid"), (1, 0))

    def main_loop(self):
        while True:
            self._handle_input()
            self._process_game_logic()
            self._draw()

    def _init_pygame(self):
        pygame.init()
        pygame.display.set_caption("Space Rocks")

    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()

    def _process_game_logic(self):
        self.spaceship.move()
        self.asteroid.move()

    def _draw(self):
        self.screen.blit(self.background, (0, 0))
        self.spaceship.draw(self.screen)
        self.asteroid.draw(self.screen)
        pygame.display.flip()
```

Και τα δύο αντικείμενα τοποθετούνται στη μέση της οθόνης, χρησιμοποιώντας τις συντεταγμένες (400, 300). Η θέση και των δύο αντικειμένων θα ενημερώνεται σε κάθε καρέ χρησιμοποιώντας την _process_game_logic(), και θα σχεδιάζονται χρησιμοποιώντας την _draw(). Αν εκτελέσουμε αυτό το πρόγραμμα θα δούμε έναν αστεροειδή να κινείται προς τα δεξιά και ένα διαστημόπλοιο να στέκεται ακίνητο στη μέση της οθόνης:



36.1.8 Έλεγχος της ταχύτητας εκτέλεσης – Βήμα 5.1

Θέλουμε το παιχνίδι μας να τρέχει με σταθερό αριθμό καρέ ανά δευτερόλεπτο (FPS) σε κάθε υπολογιστή, ανεξάρτητα από τις δυνατότητές του.

Ευτυχώς, η Pygame μπορεί να το φροντίσει αυτό.

Διαθέτει τη κλάση pygame.time.Clock με μια μέθοδο tick(). Αυτή η μέθοδος θα περιμένει τόσο ώστε να ταιριάζει με την επιθυμητή τιμή FPS, που μεταβιβάζεται ως όρισμα.

Ενημερώνουμε το αρχειάκι space_rocks/game.py.

Εάν εκτελέσουμε το παιχνίδι μας τώρα, ο αστεροειδής μπορεί να κινηθεί με διαφορετική ταχύτητα από αυτή που είχε αρχικά. Ωστόσο, μπορούμε τώρα να είμαστε σίγουροι ότι αυτή η ταχύτητα θα παραμείνει ίδια, ακόμη και σε υπολογιστές με εξαιρετικά γρήγορους επεξεργαστές.

Αυτό συμβαίνει επειδή το παιχνίδι μας θα τρέχει πάντα στα 60 FPS. Μπορούμε επίσης να πειραματιστούμε με διάφορες τιμές που μπορούμε να βάλουμε στην tick()για να δούμε τη διαφορά.

36.1.9 Δημιουργία του διαστημοπλοίου – Βήμα 5.2

Η GameObject, έχει μια γενική λογική που μπορεί να χρησιμοποιηθεί ξανά από διαφορετικά αντικείμενα του παιχνιδιού, μέσω της κληρονομικότητας. Ωστόσο, κάθε αντικείμενο θα εφαρμόσει επίσης τη δική του λογική. Το διαστημόπλοιο, για παράδειγμα, αναμένεται να περιστρέφεται και να επιταχύνει. Θα ρίχνει και σφαίρες, αλλά αυτό έρχεται αργότερα.

Δημιουργία κλάσης Spaceship

Αφού εισάγουμε και τη Vector2 για τη διαχείριση της κίνησης μέσω των διανυσμάτων των αντικειμένων, δημιουργούμε και την παραπάνω κλάση και μέχρι εδώ, απλώς καλούμε τον κατασκευαστή της GameObject με μια συγκεκριμένη εικόνα και με μηδενική ταχύτητα.

Το νέο αρχείο models.py διαμορφώνεται ως εξής:

```
from pygame.math import Vector2
from utils import load_sprite
Τώρα μπορούμε να δημιουργήσουμε, στο ίδιο αρχείο, τη κλάση
Spaceship που κληρονομεί από την GameObject:
class Spaceship(GameObject):
def __init__(self, position):
super().__init__(position, load_sprite("spaceship"),
Vector2(0))
```

Για να χρησιμοποιήσουμε αυτή την κλάση, πρέπει να την εισάγουμε, στο αρχείο games.py

Έχουμε λοιπόν:

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

Αν επεξεργαστούμε και την κλάση SpaceRocks, θα έχουμε:

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

def main_loop(self):
while True:
self._handle_input()
self._process_game_logic()
self._draw()

def _init_pygame(self):
pygame.init()
pygame.display.set_caption("Space Rocks")

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()

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

def _draw(self):
self.screen.blit(self.background, (0, 0))
self.spaceship.draw(self.screen)
pygame.display.flip()
self.clock.tick(60)
```

Συνέβησαν δύο πράγματα:

Στη γραμμή 7, αντικαταστήσαμε τη βασική κλάση GameObject με μια

ειδική κλάση, την Spaceship.

Καταργήσαμε όλες τις αναφορές self.asteroid από τις __init__(), _process_game_logic() και _draw().

Οι αλλαγές δεν πρόσθεσαν ακόμη καμία νέα συμπεριφορά, αλλά τώρα έχουμε μια κλάση την οποία μπορούμε να επεκτείνουμε.

36.1.10 Περιστροφή του διαστημοπλοίου – Βήμα 5.3

Από προεπιλογή, το διαστημόπλοιο είναι στραμμένο προς τα επάνω, προς το επάνω μέρος της οθόνης. Οι παίκτες θα πρέπει να μπορούν να το περιστρέψουν αριστερά και δεξιά. Ευτυχώς, η Pygame έχει ενσωματωμένες μεθόδους για την περιστροφή των sprites.

Αρχικά, δημιουργούμε ένα σταθερό διάνυσμα που ονομάζεται UP στο αρχείο space_rocks/models.py. Θα το χρησιμοποιήσουμε ως αναφορά αργότερα:

```
UP = Vector2(0, -1)
```

Ας θυμηθούμε ότι ο άξονας y του Pygame πηγαίνει από πάνω προς τα κάτω, επομένως μια αρνητική τιμή δείχνει στην πραγματικότητα προς τα πάνω:

Στη συνέχεια, τροποποιούμε την κλάση Spaceship ως εξής:

```
class Spaceship(GameObject):
MANEUVERABILITY = 3
```

Η τιμή της MANEUVERABILITY καθορίζει πόσο γρήγορα μπορεί να περιστραφεί το διαστημόπλοιό μας. Μάθαμε νωρίτερα ότι τα διανύσματα στο Pygame μπορούν να περιστραφούν και αυτή η τιμή αντιπροσωπεύει μια γωνία σε μοίρες κατά την οποία η κατεύθυνση του διαστημοπλοίου μας μπορεί να περιστρέφει κάθε πλαίσιο. Η χρήση μεγαλύτερου αριθμού θα περιστρέψει το διαστημόπλοιο πιο γρήγορα, ενώ ένας μικρότερος αριθμός θα επιτρέψει πιο λεπτομερή έλεγχο της περιστροφής.

Στη συνέχεια, προσθέτουμε μια κατεύθυνση στην κλάση Spaceship

τροποποιώντας τον κατασκευαστή:

```
def __init__(self, position):
# Κάνουμε ένα αντίγραφο του αρχικού διανύσματος UP
self.direction = Vector2(UP)
super().__init__(position, load_sprite("spaceship"),
Vector2(0))
```

Το διάνυσμα κατεύθυνσης θα είναι αρχικά το ίδιο με το διάνυσμα UP.

Ωστόσο, θα τροποποιηθεί αργότερα, επομένως πρέπει να δημιουργήσουμε ένα αντίγραφό του.

Στη συνέχεια, πρέπει να δημιουργήσουμε μια νέα μέθοδο στην κλάση

Spaceship που ονομάζεται rotate():

```
def rotate(self, clockwise=True):
sign = 1 if clockwise else -1
angle = self.MANEUVERABILITY * sign
self.direction.rotate_ip(angle)
```

Αυτή η μέθοδος θα αλλάξει την κατεύθυνση περιστρέφοντάς την είτε

δεξιόστροφα είτε αριστερόστροφα. Η μέθοδος rotate_ip() της κλάσης Vector2 την περιστρέφει στη θέση της κατά μια δεδομένη γωνία σε μοίρες. Το μήκος του διανύσματος δεν αλλάζει κατά τη διάρκεια αυτής της λειτουργίας.

Το μόνο που απομένει τώρα, είναι να ενημερώσουμε το σχέδιο του Spaceship. Για να το κάνουμε αυτό, πρέπει πρώτα να εισαγάγουμε το rotozoom, το οποίο είναι υπεύθυνο για την κλιμάκωση και την περιστροφή εικόνων:

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

Στη συνέχεια, μπορούμε να παρακάμψουμε τη μέθοδο draw() στην κλάση Spaceship, δηλαδή:

```
def draw(self, surface):
angle = self.direction.angle_to(UP)
rotated_surface = rotozoom(self.sprite, angle, 1.0)
rotated_surface_size = Vector2(rotated_surface.get_size())
blit_position = self.position - rotated_surface_size * 0.5
surface.blit(rotated_surface, blit_position)
```

Τώρα πρέπει να προσθέσουμε χειρισμό εισόδου. Ωστόσο, ο βρόχος συμβάντος δεν θα λειτουργεί ακριβώς εδώ. Τα γεγονότα καταγράφονται όταν συμβαίνουν, αλλά πρέπει να ελέγχουμε συνεχώς εάν πατιέται κάποιο πλήκτρο. Άλλωστε, το διαστημόπλοιο θα πρέπει να επιταχύνει για όσο διάστημα πατάμε το Up και θα πρέπει να περιστρέφεται συνεχώς όταν πατάμε Left ή Right.

Θα μπορούσαμε να δημιουργήσουμε μια σημαία για κάθε πλήκτρο, να την ορίσουμε όταν πατάμε το πλήκτρο και να την επαναφέρουμε όταν απελευθερώνεται. Ωστόσο, υπάρχει καλύτερος τρόπος.

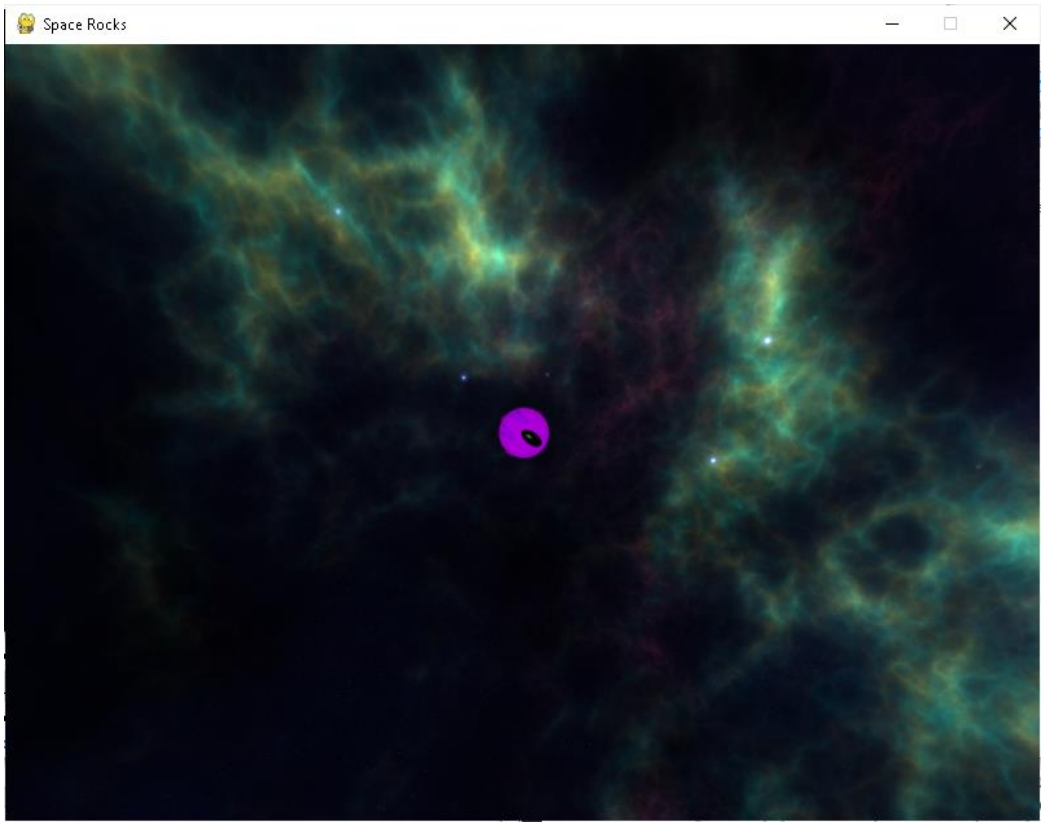
Η τρέχουσα κατάσταση του πληκτρολογίου αποθηκεύεται στο Pygame και μπορεί να ληφθεί χρησιμοποιώντας το pygame.key.get_pressed().

Επιστρέφει ένα λεξικό όπου οι σταθερές των πλήκτρων (όπως η pygame.K_ESCAPE που χρησιμοποιήσαμε προηγουμένως) είναι πλήκτρα και η τιμή είναι True εάν πατηθεί το πλήκτρο ή False αν όχι.

Γνωρίζοντας αυτό, μπορούμε να επεξεργαστούμε το αρχείο space_rocks/game.py και να ενημερώσουμε τη μέθοδο _handle_input() της κλάσης SpaceRocks. Οι σταθερές που πρέπει να χρησιμοποιήσουμε για τα πλήκτρα βέλους είναι pygame.K_RIGHTκαι pygame.K_LEFT:

```
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)
```

Τώρα το διαστημόπλοιό μας θα περιστρέφεται αριστερά και δεξιά όταν πατήσουμε τα πλήκτρα βέλους:



Και ο κώδικάς μας τώρα στο αρχείο games.py είναι:

```
import pygame

from models import Spaceship
from utils import load_sprite


class 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.spaceship = Spaceship((400, 300))

    def main_loop(self):
        while True:
            self._handle_input()
            self._process_game_logic()
            self._draw()

    def _init_pygame(self):
        pygame.init()
        pygame.display.set_caption("Space Rocks")

    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()

    def _process_game_logic(self):
```

```
        self.spaceship.move(self.screen)

    def _draw(self):
        self.screen.blit(self.background, (0, 0))
        self.spaceship.draw(self.screen)
        pygame.display.flip()
        self.clock.tick(60)
```

36.2.0 Άσκηση – Project 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.