22. ΚΑΛΟΚΑΙΡΙΝΑ PROJECTS IV – ASTEROIDS GAME (Β'ΜΕΡΟΣ)

Λόγω της μεγάλης χρησιμότητας των διανυσμάτων στα παιχνίδια, το Pygame έχει ήδη μια κλάση για αυτά, τη Vector2 στο module pygame.math.

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.

Το αρχείο θα πρέπει να είναι:

```
from pygame.math import Vector2


class GameObject:
    def __init__(self, position, sprite, velocity):
        self.position = Vector2(position)
        self.sprite = sprite
        self.radius = sprite.get_width() / 2
```

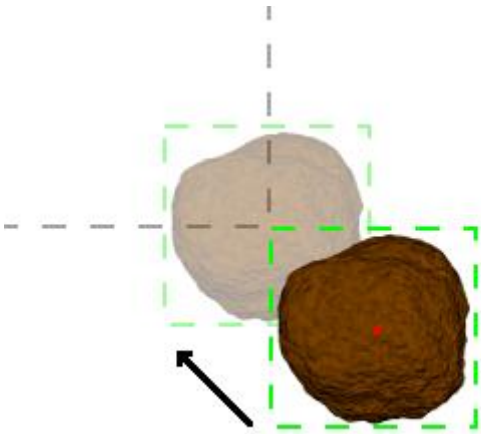
```
        self.velocity = Vector2(velocity)

    def draw(self, surface):
        blit_position = self.position - Vector2(self.radius)
        surface.blit(self.sprite, blit_position)

    def move(self):
        self.position = self.position + self.velocity

    def collides_with(self, other_obj):
        distance = self.position.distance_to(other_obj.position)
        return distance < self.radius + other_obj.radius
```

Ας δούμε τον κώδικα:

- Η γραμμή 1 εισάγει την κλάση Vector2 που αναφέρθηκε προηγουμένως.
- Η γραμμή 3 δημιουργεί την κλάση GameObject, την οποία θα χρησιμοποιήσουμε για να αναπαραστήσουμε όλα τα αντικείμενα του παιχνιδιού στο Space Rocks.
- Η γραμμή 4 είναι ο κατασκευαστής της κλάσης GameObject. Χρειάζεται τρία ορίσματα:
    1. position: Το κέντρο του αντικειμένου
    2. sprite: Η εικόνα που χρησιμοποιήθηκε για τη σχεδίαση αυτού του αντικειμένου
    3. velocity: Ενημερώνει το position του αντικειμένου σε κάθε καρέ
- Οι γραμμές 5 και 8 διασφαλίζουν ότι το position και το velocity θα αντιπροσωπεύονται πάντα ως διανύσματα για μελλοντικούς υπολογισμούς, ακόμα κι αν οι πλειάδες περαστούν στον κατασκευαστή. Το κάνουμε αυτό καλώντας τον Vector2() κατασκευαστή. Αν του δοθεί πλειάδα, τότε θα δημιουργήσει ένα νέο διάνυσμα από αυτή. Εάν του δοθεί ένα διάνυσμα, τότε θα δημιουργήσει ένα αντίγραφο αυτού του διανύσματος.

- Η γραμμή 7 υπολογίζει το radius σαν το μισό πλάτος του sprite. Σε αυτό το πρόγραμμα, τα sprites αντικειμένων του παιχνιδιού θα είναι πάντα τετράγωνα με διαφανές φόντο. Θα μπορούσαμε επίσης να χρησιμοποιήσουμε το ύψος της εικόνας - δεν θα είχε καμία διαφορά.
- Η γραμμή 10 ορίζει το draw(), το οποίο θα σχεδιάσει το sprite του αντικειμένου στην επιφάνεια και θα έχει περαστεί ως όρισμα.
- Η γραμμή 11 υπολογίζει τη σωστή θέση για το blitting της εικόνας. Η διαδικασία περιγράφεται λεπτομερέστερα παρακάτω.
- Παρατηρήστε ότι ο Vector2()κατασκευαστής λαμβάνει έναν μόνο αριθμό αντί για μια πλειάδα. Σε αυτήν την περίπτωση, θα χρησιμοποιήσει αυτόν τον αριθμό και για τις δύο τιμές. Έτσι το Vector2(self.radius) είναι ισοδύναμο με το Vector2((self.radius, self.radius)).
- Η γραμμή 12 χρησιμοποιεί τη θέση blit που υπολογίστηκε πρόσφατα για να τοποθετήσει το sprite του αντικειμένου μας στη σωστή θέση στη δεδομένη επιφάνεια.
- Η γραμμή 14 ορίζει τη move(). Θα ενημερώσει τη θέση του αντικειμένου του παιχνιδιού.
- Η γραμμή 15 προσθέτει την ταχύτητα στη θέση και λαμβάνει ένα ενημερωμένο διάνυσμα θέσης ως αποτέλεσμα. Το Pygame κάνει τον χειρισμό διανυσμάτων απλό, επιτρέποντάς μας να τα προσθέτουμε σαν αριθμούς.
- Η γραμμή 17 ορίζει την collides_with() μέθοδο που θα χρησιμοποιηθεί για τον εντοπισμό συγκρούσεων.
- Η γραμμή 18 υπολογίζει την απόσταση μεταξύ δύο αντικειμένων χρησιμοποιώντας το Vector2.distance_to().
- Η γραμμή 19 ελέγχει αν αυτή η απόσταση είναι μικρότερη από το άθροισμα των ακτίνων των αντικειμένων. Αν ναι, τα αντικείμενα συγκρούονται.

Ας λάβουμε υπόψη ότι τα αντικείμενα του παιχνιδιού μας έχουν κεντρική θέση, αλλά η blit()απαιτεί την επάνω αριστερή γωνία. Έτσι, η θέση στην οποία τοποθετεί η blit πρέπει να υπολογιστεί μετακινώντας την πραγματική θέση του αντικειμένου κατά ένα διάνυσμα:



Αυτή η διαδικασία συμβαίνει στο draw().

Μπορούμε να το δοκιμάσουμε προσθέτοντας ένα διαστημόπλοιο και έναν μόνο αστεροειδή. Αρχικά, ας αντιγράψουμε τις εικόνες του διαστημοπλοίου και του αστεροειδούς στο assets/sprites.

Τώρα τροποποιώντας το αρχείο space_rocks/game.py έχουμε:

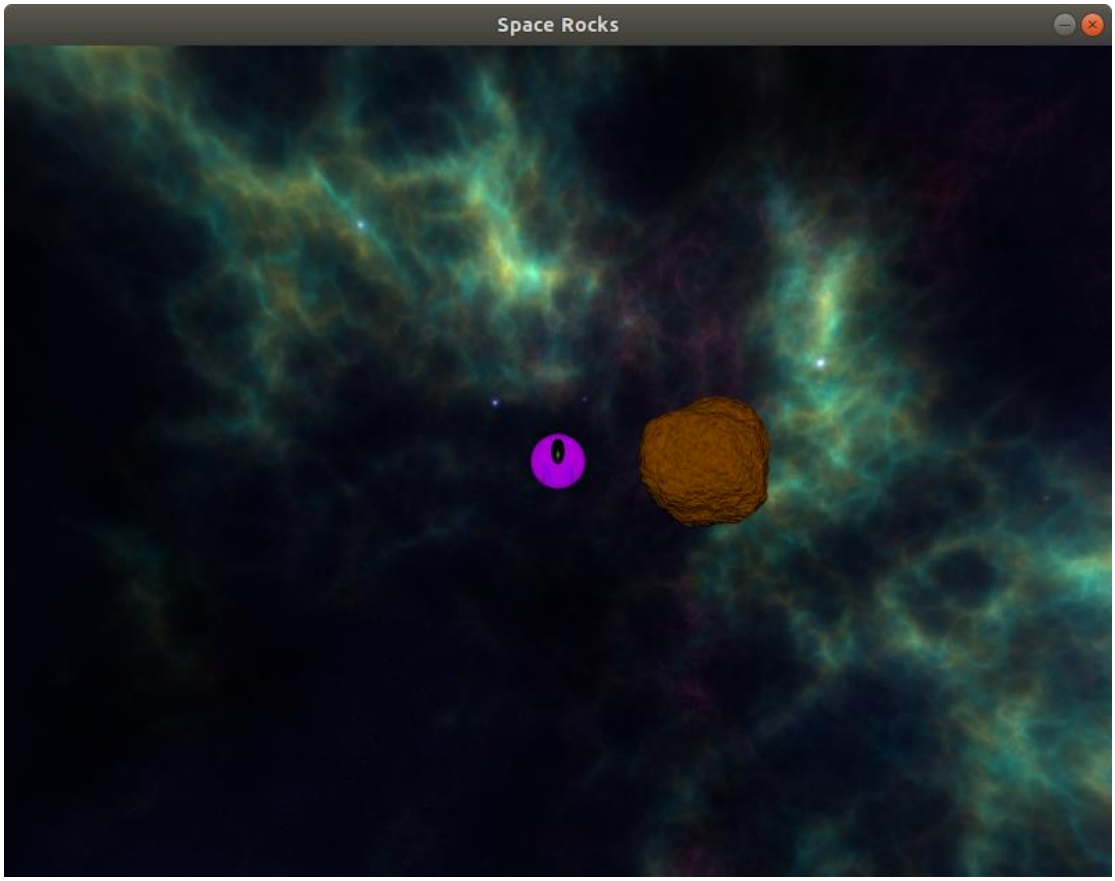
```
import pygame


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.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:



Μπορούμε επίσης να δοκιμάσουμε την collides_with() προσθέτοντας προσωρινά την παρακάτω γραμμή κώδικα στο τέλος της _draw():

print("Collides:", self.spaceship.collides_with(self.asteroid))

Στη γραμμή εντολών, θα παρατηρήσετε πώς εκτυπώνεται αρχικά η μέθοδος Trueαφού ο αστεροειδής καλύπτει το διαστημόπλοιο. Αργότερα, καθώς ο αστεροειδής μετακινείται πιο δεξιά, αρχίζει να εκτυπώνει False.

22.1.9 Έλεγχος της ταχύτητας

Τώρα που έχουμε κινούμενα αντικείμενα στην οθόνη, ήρθε η ώρα να σκεφτούμε πώς θα αποδώσει το παιχνίδι μας σε διαφορετικά μηχανήματα με διαφορετικούς επεξεργαστές. Μερικές φορές θα τρέχει πιο γρήγορα και μερικές φορές θα τρέχει πιο αργά.

Εξαιτίας αυτού, οι αστεροειδείς (και σύντομα οι σφαίρες) θα κινούνται με διαφορετική ταχύτητα, κάνοντας το παιχνίδι άλλοτε ευκολότερο και άλλοτε πιο δύσκολο. Δεν είναι κάτι που θέλουμε. Αυτό που θέλουμε

είναι το παιχνίδι μας να τρέχει με σταθερό αριθμό **καρέ ανά δευτερόλεπτο (FPS)** .

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:

```
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()
    self.clock.tick(60)
```

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.

Μόλις μάθαμε πώς να εμφανίζουμε και να μετακινούμε αντικείμενα στην οθόνη. Τώρα μπορούμε να προσθέσουμε κάποια πιο προηγμένη λογική.

Πάμε παρακάτω.

22.1.10 Το Διαστημόπλοιό μας

Σε αυτό το σημείο, θα πρέπει να έχουμε μια γενική κλάση για συρόμενα και κινούμενα αντικείμενα στο παιχνίδι μας. Στο τέλος αυτού του βήματος, θα τη χρησιμοποιήσουμε για να δημιουργήσουμε ένα ελεγχόμενο διαστημόπλοιο.

Η κλάση που δημιουργήσαμε στο προηγούμενο βήμα, η GameObject, έχει κάποια γενική λογική που μπορεί να χρησιμοποιηθεί ξανά από διαφορετικά αντικείμενα του παιχνιδιού. Ωστόσο, κάθε αντικείμενο του παιχνιδιού 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

Η εικόνα του διαστημοπλοίου βρίσκεται ήδη στο φάκελο space_rocks/assets που προσθέσαμε προηγουμένως.

Ωστόσο, νωρίτερα χρησιμοποιήθηκε στο κύριο αρχείο του παιχνιδιού και τώρα πρέπει να το φορτώσουμε σε ένα από τα μοντέλα. Για να μπορέσουμε να το κάνουμε αυτό, ας ενημερώσουμε το τμήμα import στο αρχείο space_rocks/models.py:

```
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))
```

Δεν κάνει πολλά σε αυτό το σημείο - απλώς καλεί τον κατασκευαστή της ***GameObject*** με μια συγκεκριμένη εικόνα και με μηδενική ταχύτητα.

Ωστόσο, σύντομα θα προσθέσουμε περισσότερες λειτουργίες.

Για να χρησιμοποιήσουμε αυτήν τη νέα κλάση, πρέπει πρώτα να την εισαγάγουμε.

Ας Ενημερώσουμε το import στο αρχείο space_rocks/game.py ως εξής:

```
import pygame

from models import Spaceship

from utils import load_sprite
```

Πιθανότατα παρατηρήσαμε ότι η αρχική import της κλάσης GameObject έχει φύγει. Αυτό συμβαίνει επειδή κλάση GameObject χρησιμοποιείται ως βασική κλάση που θα κληρονομηθεί από άλλες κλάσεις.

Δεν πρέπει να τις χρησιμοποιούμε απευθείας, αλλά να εισάγουμε τις κλάσεις που αντιπροσωπεύουν πραγματικά αντικείμενα του παιχνιδιού.

Αυτό σημαίνει ότι ο αστεροειδής από το προηγούμενο βήμα θα σταματήσει να λειτουργεί, αλλά αυτό δεν είναι μεγάλο θέμα. Σύντομα θα προσθέσουμε μια κατάλληλη κλάση που αντιπροσωπεύει τους αστεροειδείς.

Μέχρι τότε, θα πρέπει να εστιάσουμε στο διαστημόπλοιο.

Ας προχωρήσουμε για να επεξεργαστούμε την κλάση SpaceRocks για να γίνει:

```
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()
```

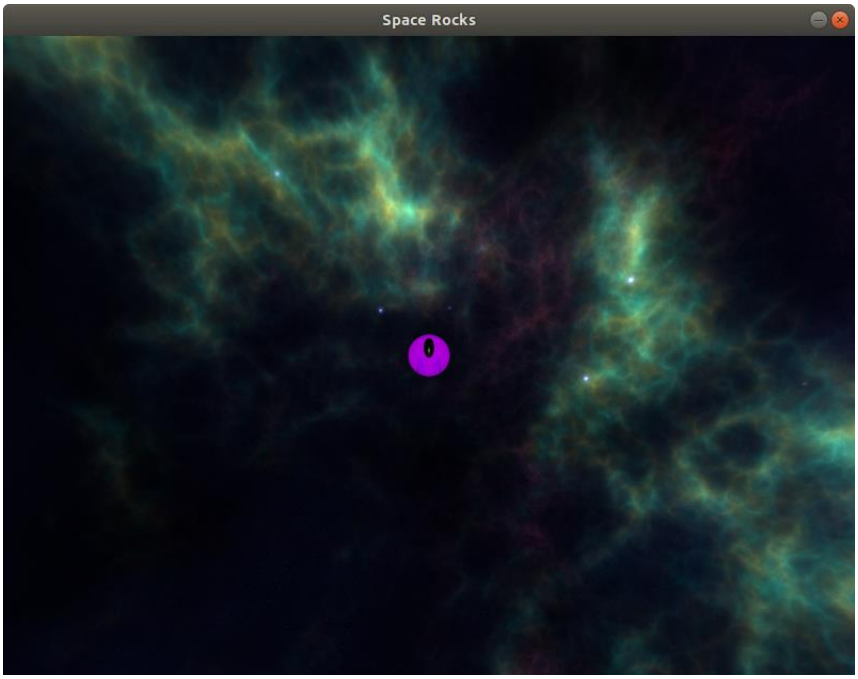
self.clock.tick(60)

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().

Εάν εκτελέσουμε τώρα το παιχνίδι, θα δούμε ένα διαστημόπλοιο στη μέση της οθόνης:



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

22.1.11 Περιστροφή του διαστημοπλοίου

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, αλλά υπάρχει ένα μικρό πρόβλημα.

Γενικά, η περιστροφή της εικόνας είναι μια πολύπλοκη διαδικασία που απαιτεί επανυπολογισμό των pixel στη νέα εικόνα. Κατά τη διάρκεια αυτού του επαναυπολογισμού, οι πληροφορίες σχετικά με τα αρχικά pixel χάνονται και η εικόνα παραμορφώνεται λίγο. Με κάθε περιστροφή, η παραμόρφωση γίνεται όλο και πιο ορατή.

Εξαιτίας αυτού, ίσως θα ήταν καλύτερη ιδέα να αποθηκεύσουμε το αρχικό sprite στην κλάση Spaceship και να έχουμε ένα άλλο sprite, το οποίο θα ενημερώνεται κάθε φορά που το διαστημόπλοιο περιστρέφεται.

Για να λειτουργήσει αυτή η προσέγγιση, θα πρέπει να γνωρίζουμε τη γωνία με την οποία περιστρέφεται το διαστημόπλοιο. Αυτό μπορεί να γίνει με δύο τρόπους:

Να διατηρήσουμε τη γωνία ως τιμή κινητής υποδιαστολής και να την ενημερώνουμε κατά την περιστροφή.

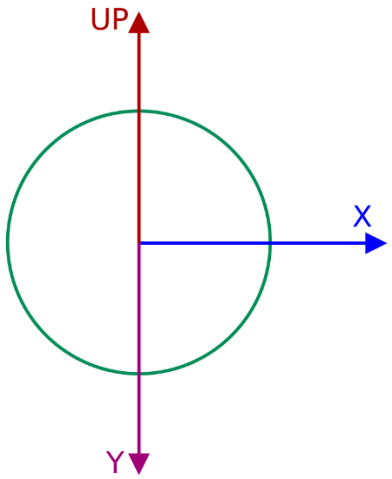
Να διατηρήσουμε το διάνυσμα που αντιπροσωπεύει την κατεύθυνση που βλέπει το διαστημόπλοιο και να υπολογίζουμε τη γωνία χρησιμοποιώντας αυτό το διάνυσμα.

Και οι δύο τρόποι είναι καλοί, αλλά πρέπει να επιλέξουμε έναν πριν προχωρήσουμε. Δεδομένου ότι η θέση και η ταχύτητα του διαστημοπλοίου είναι ήδη διανύσματα, είναι λογικό να χρησιμοποιήσουμε ένα άλλο διάνυσμα για να αναπαραστήσουμε την κατεύθυνση. Αυτό θα κάνει πιο απλό το να προσθέσουμε διανύσματα και να ενημερώσουμε τη θέση αργότερα. Ευτυχώς, με την κλάση Vector2 μπορούμε να χρησιμοποιήσουμε την περιστροφή πολύ εύκολα και το αποτέλεσμα δεν θα παραμορφώσει την εικόνα.

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)
```

Ακολουθεί μια ανάλυση βήμα προς βήμα:

Η γραμμή 2 χρησιμοποιεί τη μέθοδο angle_to() της κλάσης Vector2 για να υπολογίσει τη γωνία με την οποία πρέπει να περιστραφεί ένα διάνυσμα για να δείχνει προς την ίδια κατεύθυνση με το άλλο διάνυσμα. Αυτό καθιστά ανώδυνη τη μετάφραση της κατεύθυνσης του διαστημοπλοίου στη γωνία περιστροφής σε μοίρες.

Η γραμμή 3 περιστρέφει το sprite χρησιμοποιώντας την rotozoom(). Λαμβάνει την αρχική εικόνα, τη γωνία με την οποία πρέπει να περιστραφεί και την κλίμακα που πρέπει να εφαρμοστεί στο sprite. Σε αυτήν την περίπτωση, δεν θέλουμε να αλλάξουμε το μέγεθος, επομένως διατηρούμε την κλίμακα ως 1.0.

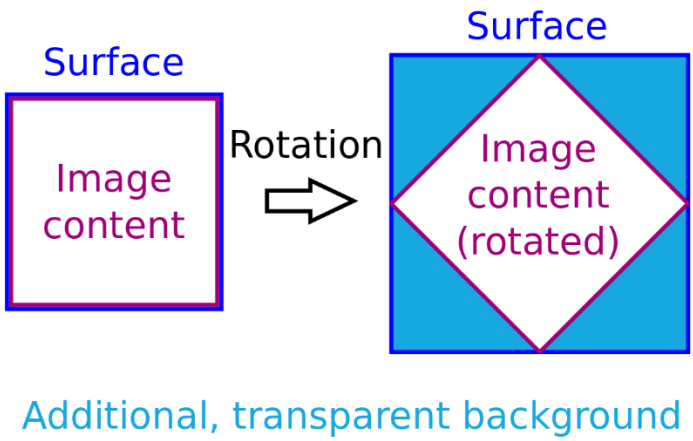
Οι γραμμές 4 και 5 υπολογίζουν εκ νέου τη θέση blit, χρησιμοποιώντας το μέγεθος rotated_surface. Η διαδικασία περιγράφεται παρακάτω.

Η γραμμή 5 περιέχει τη συνάρτηση rotated_surface_size * 0.5. Αυτό είναι ένα άλλο πράγμα που μπορούμε να κάνουμε με διανύσματα στο

Pygame. Όταν πολλαπλασιάζουμε ένα διάνυσμα με έναν αριθμό, όλες οι συντεταγμένες του πολλαπλασιάζονται με αυτόν τον αριθμό. Ως αποτέλεσμα, ο πολλαπλασιασμός με 0.5 θα επιστρέψει ένα διάνυσμα με το μισό μήκος του αρχικού.

Η γραμμή 6 χρησιμοποιεί τη θέση blit που υπολογίστηκε πρόσφατα για να τοποθετήσει την εικόνα στην οθόνη.

Σημειώστε ότι η rotozoom() επιστρέφει μια νέα επιφάνεια με μια περιστρεφόμενη εικόνα. Ωστόσο, για να διατηρηθούν όλα τα περιεχόμενα του αρχικού sprite, η νέα εικόνα μπορεί να έχει διαφορετικό μέγεθος. Σε αυτήν την περίπτωση, το Pygame θα προσθέσει κάποιο πρόσθετο, διαφανές φόντο:



Περιστρεφόμενη επιφάνεια με διαφορετικές διαστάσεις

Το μέγεθος της νέας εικόνας μπορεί να είναι σημαντικά διαφορετικό από αυτό της αρχικής εικόνας. Γι' αυτό η draw() επανυπολογίζει τη θέση blit της rotated_surface. Ας θυμηθούμε ότι η blit() ξεκινά από την επάνω αριστερή γωνία, επομένως για να κεντράρουμε την περιστρεφόμενη εικόνα, πρέπει επίσης να μετακινήσουμε τη θέση blit κατά το ήμισυ του μεγέθους της εικόνας.

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.

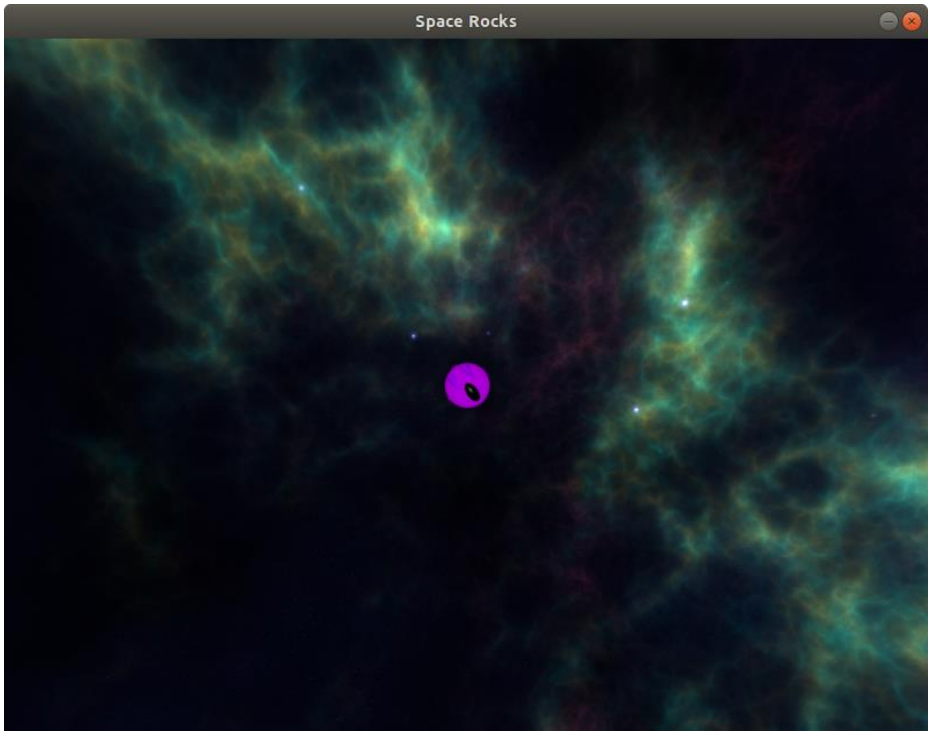
Η τρέχουσα κατάσταση του πληκτρολογίου αποθηκεύεται στο Pygame και μπορεί να ληφθεί χρησιμοποιώντας το pygame.key.get_pressed(). Επιστρέφει ένα λεξικό όπου οι σταθερές των πλήκτρων (όπως η pygame.K_ESCAPE που χρησιμοποιήσαμε προηγουμένως) είναι πλήκτρα και η τιμή είναι True εάν πατηθεί το πλήκτρο ή False 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 σας θα περιστρέφεται αριστερά και δεξιά όταν πατήσουμε τα πλήκτρα βέλους:



22.1.12 Επιτάχυνση του διαστημοπλοίου

Σε αυτήν την ενότητα, θα προσθέσουμε επιτάχυνση στο διαστημόπλοιό μας. Ας θυμηθούμε ότι, σύμφωνα με τη μηχανική του παιχνιδιού μας, το διαστημόπλοιο μπορεί να προχωρήσει μόνο μπροστά.

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
```

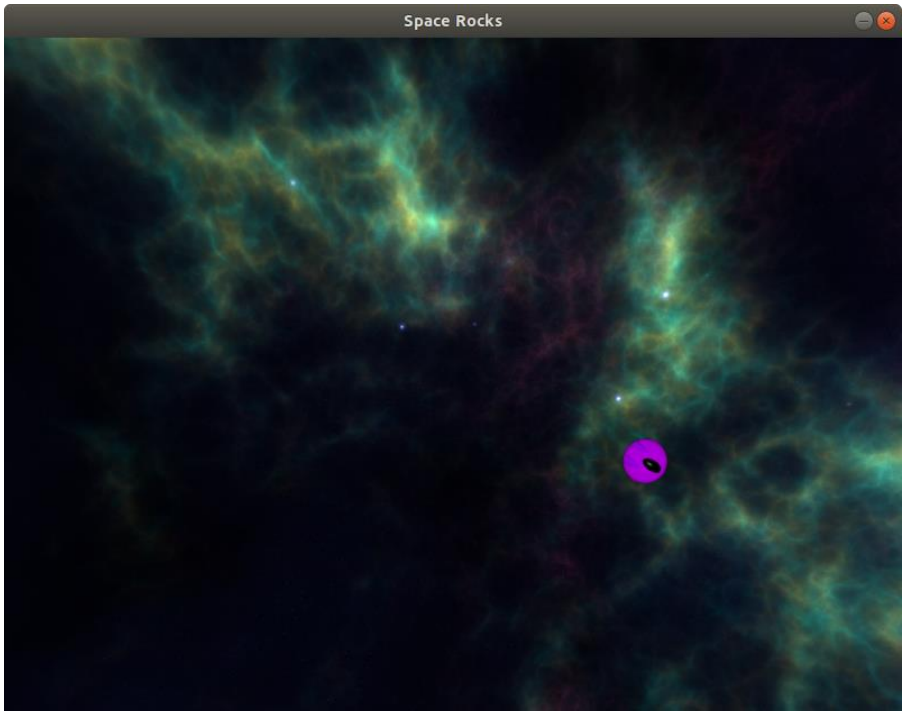
Στη συνέχεια, ας δημιουργήσουμε την accelerate() στην κλάση Spaceship:

```
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:



Το διαστημόπλοιό μας μπορεί πλέον να κινείται και να περιστρέφεται! Ωστόσο, όταν φτάσει στην άκρη της οθόνης, απλώς συνεχίζει να κινείται. Αυτό είναι κάτι που πρέπει να διορθώσουμε!

22.1.13 Συγκράτηση αντικειμένων στην οθόνη

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!

ΤΟ ΚΑΛΟΚΑΙΡΑΚΙ ΣΥΝΕΧΙΖΕΤΑΙ ΓΙΑ ΠΟΛΥ ΛΙΓΟ ΑΚΟΜΑ!

(ΔΥΣΤΥΧΩΣ!)