Κεφάλαιο 7

Pygame Invaders: Η Αρχή!

7.1 Εισαγωγή

Αν είχατε ξεκινήσει να προγραμματίζετε στα 80s, είναι σίγουρο ότι τα πρώτα σας προγράμματα θα ήταν παρόμοια με αυτά που έχουμε γράψει ως τώρα. Πολλά μάλιστα θα τα είχατε αντιγράψει από το εγχειρίδιο προγραμματισμού του υπολογιστή σας και σίγουρα θα είχατε σπαταλήσει αρκετές ώρες για να κατανοήσετε πως λειτουργούν και να πειραματιστείτε σε παραλλαγές τους. Αν μάλιστα μας επιτρέπετε να μαντέψουμε, μετά τα πρώτα σας προγράμματα των 5-10 γραμμών, θα είχατε γράψει ένα πρόγραμμα “Guess the Number” ένα “Bouncing Ball” και ένα παιχνίδι “Graphics Match”.

Δεν ξέρω αν η πορεία αυτή σας θυμίζει κάτι, αλλά μάλλον θα πρέπει! Ο γράφων ξεκίνησε αντιγράφοντας απλά προγράμματα σαν αυτά και συνέχισε με δικές του εκδοχές και παραλλαγές οι οποίες γρήγορα κατέληξαν σε δικά του πρωτότυπα προγράμματα. Μια άλλη πηγή έμπνευσης ήταν φυσικά τα περιοδικά (Pixel, Micromad) που δημοσίευαν προγράμματα (σε μορφή listing που έπρεπε να πληκτρολογηθούν) και αποτελούσαν ανεξάντλητη πηγή ιδεών και λύσεων σε πρακτικά θέματα προγραμματισμού και αλγορίθμων (κοινώς: πως θα κάνω το καταραμένο διαστημοπλοιάκι να κινείται με τα βελάκια).

Αυτό που θέλω να τονίσω με τα παραπάνω είναι:

- Κανείς δεν γεννήθηκε γνωρίζοντας προγραμματισμό, όλοι τον μάθαμε διαβάζοντας, πειραματιζόμενοι και παίζοντας με τον κώδικα άλλων. Αυτό που μας χωρίζει με όσους δεν έμαθαν ποτέ προγραμματισμό είναι η θέληση και η πίστη ότι θα τα καταφέρουμε.

- Κατά τη συγγραφή ενός προγράμματος δεν είναι σπάνιο να αλλάξουμε τον κώδικα μας τόσες φορές που τελικά να μην έχει καμιά σχέση με τον αρχικό! Αυτό άλλωστε θα το δείτε και καθώς θα φτιάχνουμε το τελικό μας παιχνίδι – που ξεκινάει φυσικά ΤΩΡΑ!

Έχουμε μάθει λίγο προγραμματισμό και αλγοριθμική, λίγο από object oriented design και φυσικά τεχνικές που χρειάζονται για παιχνίδια όπως ανάγνωση πληκτρολογίου. Έχουμε πλέον τη γνώση και το θάρρος (!) να δοκιμάσουμε να γράψουμε ένα ολοδικό μας παιχνίδι.

Τι παιχνίδι όμως; Όπως είπα πριν, στις παλιές καλές εποχές παίρναμε ιδέες από προγράμματα άλλων αλλά και από παιχνίδια που βλέπαμε στις αίθουσες με τα ηλεκτρονικά (ουφάδικα όπως τα λέγαμε τότε). Και τα πιο πολλά από αυτά είχαν να κάνουν με εισβολή εξωγήινων, δηλαδή κλασικά shoot-em up.

Τριάντα και πλέον χρόνια μετά οι εξωγήινοι δεν ήρθαν ποτέ (αποκαλύφθηκε ότι τους γήινους πρέπει να φοβόμαστε στην πραγματικότητα) αλλά ο στόχος παραμένει: το πρώτο πραγματικό μας παιχνίδι πρέπει να είναι ένα shoot-em up. Και ο τίτλος αυτού: **Pygame Invaders!**

7.2 Στρατηγική: Πόσο Δύσκολο Είναι να Φτιαχτεί Ένα Shoot-em Up;

Πριν σκεφτούμε αυτό το ερώτημα ας περιγράψουμε λίγο το παιχνίδι μας:

- Παίζεται σε ένα παράθυρο 480X640 (portrait!)
- Το υπερσύγχρονο διαστημοπλοιάκι μας κινείται αριστερά δεξιά στο κάτω μέρος και πυροβολεί ταχύτατα.
- Οι εξωγήινοι εμφανίζονται σε κύματα στην οθόνη, ρίχνουν αδιακρίτως, δεν είναι ιδιαίτερα έξυπνοι αλλά είναι περισσότεροι.
- Το διαστημόπλοιο μας διαθέτει ασπίδα που μειώνεται με κάθε βολή που δεχόμαστε και το παιχνίδι τελειώνει όταν γίνει μηδέν. Κάθε 100 πόντους όμως κερδίζουμε μια μονάδα ασπίδας. Αντίθετα οι εξωγήινοι πεθαίνουν με μια βολή (ούτε έξυπνοι, ούτε ανθεκτικοί – αφήστε που το πρόγραμμα είναι δικό μας και τους κάνουμε ότι θέλουμε) και παίρνουμε 10 βαθμούς για κάθε ένα που… ξεπαστρεύουμε!
- Το φόντο είναι μαύρο με αστέρια (τι περιμένετε δηλαδή, στο διάστημα είναι το παιχνίδι ντε!) τα οποία θα θέλαμε να τα κάνουμε και να κινούνται (background scrolling)
- Φυσικά το παιχνίδι διαθέτει ηχητικά εφέ και μουσική.

Αν ρωτήσετε τώρα πόσο εύκολο είναι να γραφεί αυτό σήμερα σε σχέση με παλιά θα σας πω: το ίδιο εύκολο – ή ίσως το ίδιο δύσκολο! Πολύ απλά, με τον επεξεργαστή των 3Mhz του 1981 και χωρίς δυνατότητα δημιουργίας καρέ, το scrolling background γίνεται ένα απατηλό όνειρο (εκτός αν πάμε σε assembly). Στο pygame όμως μπορούμε πλέον να γράψουμε τα πάντα. Όσο αφορά τη γλώσσα προγραμματισμού, η BASIC της εποχής δεν διέθετε objects. Στην καλύτερη περίπτωση να είχαμε κάποιο υπολογιστή με δυνατότητα κίνησης γραφικών (*sprites*) οπότε

θα έπρεπε να βρούμε ένα έξυπνο τρόπο να τα χρησιμοποιήσουμε. Η απλούστερη γλώσσα προγραμματισμού σημαίνει ότι δεν χρειάζεται να θυμόμαστε πολύπλοκες εντολές και δομές. Από την άλλη όμως πρέπει συχνά να καταφύγουμε σε περίεργες και ανορθόδοξες τεχνικές για να πετύχουμε το σκοπό μας. Στο pygame μπορούμε να γράψουμε το παιχνίδι μας με αρκετά καθαρό και φυσικά object oriented τρόπο αλλά σίγουρα η python έχει κάποιο βαθμό πολυπλοκότητας μεγαλύτερο της BASIC.

7.3 Ήχος και Pygame

Το παιχνίδι μας θα έχει ηχητικά εφέ και μουσική. Αυτό είναι κάτι που δεν έχουμε εξετάσει στο pygame ακόμα και μάλλον ήρθε η ώρα. Υπάρχουν δύο βασικοί τρόποι για παραγωγή ήχου:

Το `pygame.mixer.music` χρησιμοποιείται για μουσική που τυπικά θέλουμε να ακούγεται στο background. Το αρχείο ήχου που θα δώσουμε στις σχετικές εντολές δεν φορτώνεται ολόκληρο στη μνήμη αλλά ουσιαστικά αναπαράγεται μέσω streaming. Το `pygame.mixer.Sound` χρησιμοποιείται για ηχητικά εφέ. Μας επιστρέφει ένα αντικείμενο τύπου `sound` (τι παράξενο) το οποίο διαθέτει την μέθοδο `play`. Μπορούμε να προετοιμάσουμε όσα αντικείμενα `sound` χρειαζόμαστε, το καθένα με διαφορετικό ηχητικό εφέ και να καλούμε την `play` για το καθένα στο κατάλληλο σημείο του παιχνιδιού.

Επειδή λίγες γραμμές κώδικα αξίζουν όσο εκατό περιγραφές, δείτε τα παρακάτω παραδείγματα και θα μπείτε αμέσως στο νόημα.

```
# Sound effects
import pygame
from pygame.locals import *
pygame.init()
clock = pygame.time.Clock()
laser = pygame.mixer.Sound("laser.wav")
laser.play()
while pygame.mixer.get_busy():
  clock.tick()
```

Δημιουργούμε ένα αντικείμενο `sound` με την εντολή:

```
laser = pygame.mixer.Sound("laser.wav")
```

Φυσικά το αρχείο `laser.wav` πρέπει να είναι ένα αρχείο ήχου στον τρέχοντα κατάλογο.

Καλούμε την μέθοδο `play` και εισερχόμαστε στο βρόχο `while` όπου απλά περιμένουμε να τελειώσει η αναπαραγωγή του ήχου (το `pygame.mixer.get_busy()` θα επιστρέψει `False`) για να τερματιστεί η εκτέλεση. Ο βρόχος απλά εισάγει μια καθυστέρηση για να ακουστεί το laser! Στο κανονικό παιχνίδι δεν θα χρειαστούμε το `pygame.mixer.get_busy()` καθώς έχουμε ήδη το βασικό βρόχο του παιχνιδιού που εκτελείται συνέχεια.

```
# Background music
import pygame
from pygame.locals import *
pygame.init()
clock = pygame.time.Clock()
pygame.mixer.music.load("spaceinvaders.ogg")
pygame.mixer.music.play()
while pygame.mixer.music.get_busy():
  clock.tick()
```

Παρόμοια και για τη μουσική, αλλά παρατηρήστε ότι η `pygame.mixer.music.load` δεν δημιουργεί κανένα αντικείμενο καθώς το κανάλι για streaming μουσικής είναι μόνο ένα.

Κάτι πολύ ενδιαφέρον όμως είναι ότι μπορούμε να γράψουμε:

```
pygame.mixer.music.play(6)
```

και το τραγούδι να επαναληφθεί 6 φορές. Η ακόμα:

```
pygame.mixer.music.play(-1)
```

οπότε θα επαναλαμβάνεται συνέχεια! Πολύ χρήσιμο για το παιχνίδι μας. Να σημειώσουμε εδώ ότι το pygame δεν τα πάει τόσο καλά με τα mp3 οπότε προτιμήστε αρχεία ogg (ή ασυμπίεστα wav για τα ηχητικά εφέ που είναι μικρά).

7.4 Τα Αντικείμενα του Παιχνιδιού

Έχοντας ξεκαθαρίσει κάπως τα πράγματα με τον ήχο, ώρα να σκεφτούμε λίγο τα αντικείμενα του παιχνιδιού και τι κλάσεις θα χρειαστεί να δημιουργήσουμε. Έχουμε λοιπόν:

1. Το διαστημοπλοιάκι μας
2. Τους εξωγήινους
3. Το laser μας
4. Το laser των εξωγήινων (μακριά από μας!)
5. Το φόντο

Ίσως να μπείτε στον πειρασμό να φτιάξετε μια κλάση για το καθένα από αυτά, αλλά μη βιαστείτε. Όχι ότι θα είναι λάθος αν το κάνετε, αλλά θα βρεθείτε να γράφετε τον ίδιο κώδικα δύο φορές. Γιατί αν το σκεφτείτε καλύτερα:

- Το διαστημοπλοιάκι μας και οι εξωγήινοι ουσιαστικά είναι παρόμοια αντικείμενα και χρειάζονται παρόμοιες μεθόδους.
- Οι δικές μας βολές laser ελάχιστα διαφέρουν από των εξωγήινων: ουσιαστικά στην κατεύθυνση της κίνησης και στον… στόχο.

Τι μεθόδους χρειαζόμαστε για το δικό μας σκάφος; Αν μελετήσατε την object oriented εκδοχή του bouncing ball στο Κεφάλαιο 5, θα μπορέσετε να σκεφτείτε εύκολα:

- Τη μέθοδο `Show` για να απεικονίζεται σε μια επιφάνεια που θα δώσουμε ως παράμετρο.
- Τη μέθοδο `Move` για να κινείται στην οθόνη.
- Τη μέθοδο `Fire` για να ρίχνει (ok, αυτό είναι καινούριο. Δεν έχουμε δει πουθενά bouncing balls που να πυροβολούν)
- Ένα constructor που να το δημιουργεί (να φορτώνει την εικόνα, να βρίσκει τις διαστάσεις κλπ)

Τις ίδιες όμως μεθόδους χρειάζεται και ο εξωγήινος! Η υλοποίηση θα διαφέρει σε ορισμένες μεθόδους – αλλά αν φτιάξουμε πρώτα μια γενική κλάση (*υπερκλάση ή superclass*) για τα διαστημόπλοια μπορούμε μετά να εξειδικεύσουμε τις μεθόδους όπου χρειάζεται.

7.5 Η Υπερκλάση Craft

Σε μια πρώτη σκέψη μπορούμε να γράψουμε την παρακάτω:

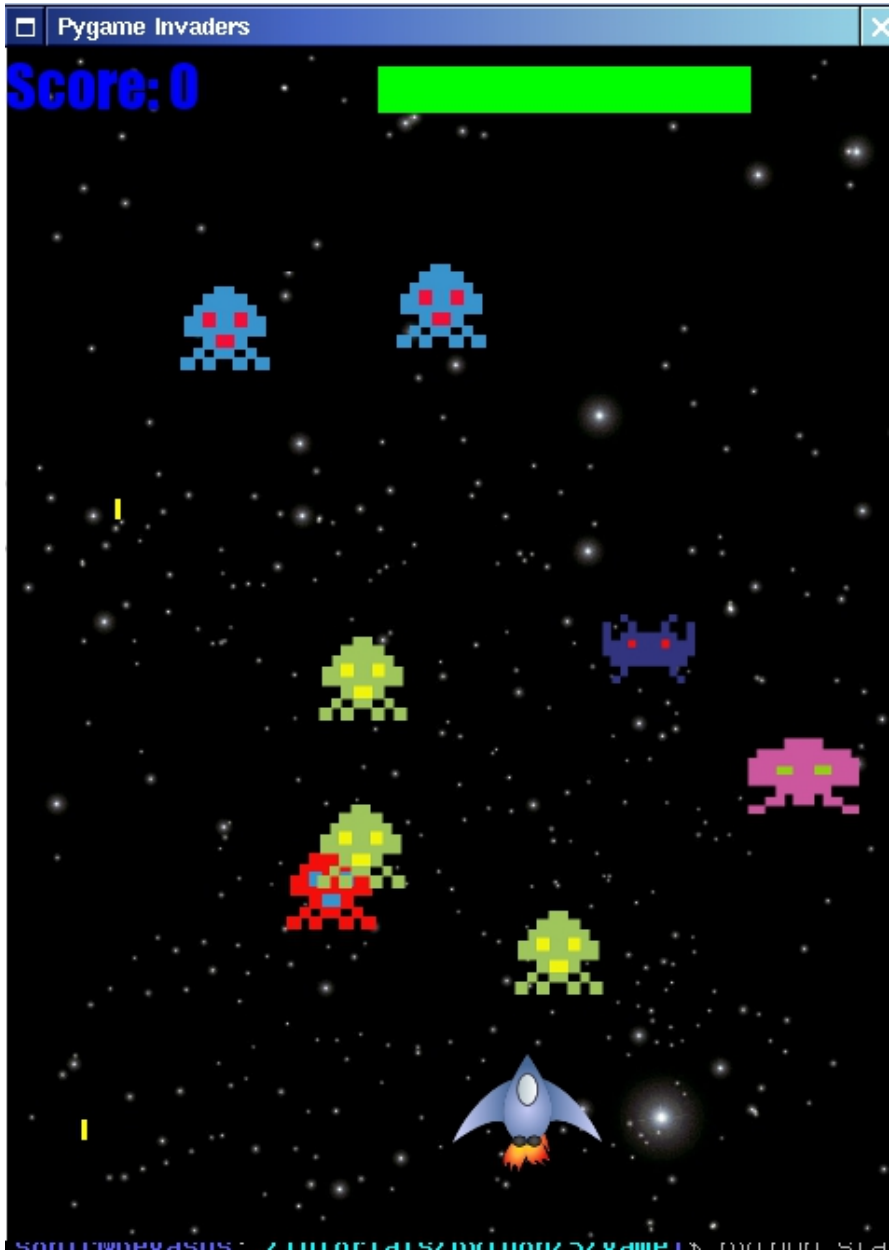
```
class Craft(object):
  def __init__ (self, imagefile, coord):
    self.shape = pygame.image.load(imagefile)
    self.ship_width = self.shape.get_width()
    self.ship_height = self.shape.get_height()
    self.rect = pygame.Rect(coord,(self.ship_width, self.ship_height))

  def Show(self, surface):
    surface.blit(self.shape,(self.rect[0],self.rect[1]))

  def Move(self,speed_x,speed_y, time):
    distance_x = speed_x * time
    distance_y = speed_y * time
    self.rect.move_ip(distance_x,distance_y)

  def Fire(self):
    pass
```

Μερικές γρήγορες παρατηρήσεις:

Αντί να χρησιμοποιούμε χωριστές μεταβλητές για να κρατάμε τις συντεταγμένες του διαστημοπλοίου, χρησιμοποιούμε το `Rect` class του pygame. Αυτό δημιουργεί αντικείμενα τύπου "παραλληλόγραμμο" τα οποία ουσιαστικά είναι λίστες με τέσσερα στοιχεία:

`[ Θέση_Χ, Θέση_Υ, Πλάτος, Ύψος ]`

Έτσι π.χ. η εντολή:



Εικόνα 7.3: Το παιχνίδι μας όπως θα δείχνει σε κάποιο προχωρημένο στάδιο. Παρατηρήστε τους retro εξωγήινους που προέρχονται από το αρχικό Space Invaders και το δικό μας υπερσύγχρονο διαστημόπλοιο. Δεν είναι τυχαίο που τελικά ποτέ δεν κυρίευσαν τη γη με αυτές τις μπακατέλες που έρχονταν. Eat my laser alien!

```
rect = pygame.Rect((240,560),(100,100))
```

φτιάχνει ένα αντικείμενο `rect` το οποίο είναι:

```
[240, 560, 100, 100]
```

Το καλό με την κλάση `Rect` είναι ότι περιέχει μεθόδους για να κινήσουμε το αντικείμενο μας — ουσιαστικά δηλαδή να αλλάξουμε τις συντεταγμένες X και Y — αλλά κυρίως για να δούμε αν ένα αντικείμενο `rect` βρίσκεται μέσα σε ένα άλλο (hint: αν έχουμε χτυπηθεί από το laser)

Μπορείτε να δείτε και να καταλάβετε εύκολα την κίνηση στην μέθοδο `Move`:

```
    self.rect.move_ip(distance_x,distance_y)
```

ουσιαστικά το παραπάνω αντικαθιστά τις εντολές:

```
    x = x + distance_x
    y = y + distance_y
```

που έχουμε δει σε προγράμματα όπως το bouncing ball. Αν πάλι κάπου χρειαζόμαστε μόνο τις τιμές των θέσεων X και Y μπορούμε να τις πάρουμε από το `rect[0]` και `rect[1]` αντίστοιχα, όπως φαίνεται και στη μέθοδο `Show`.

Η μέθοδος `Fire` περιέχει μόνο μια εντολή:

```
    pass
```

Την `pass` τη χρησιμοποιούμε στην python όταν κάπου χρειάζεται μια εντολή αλλά εμείς δεν έχουμε κάτι να γράψουμε. Προφανώς τη μέθοδο `Fire` δεν την έχουμε σκεφτεί ακόμα, βάζουμε λοιπόν `pass` προκειμένου να την ολοκληρώσουμε αργότερα.

Από την υπερκλάση `Craft` θα δημιουργήσουμε τις κλάσεις για το δικό μας διαστημόπλοιο αλλά και για τους εξωγήινους. Θα ξεκινήσουμε με μια απλή εκδοχή:

```
class SpaceCraft(Craft):
  pass
```

Καθώς καταλαβαίνετε, σίγουρα θα προσθέσουμε πράγματα σε αυτή την κλάση αλλά για την ώρα ας δούμε πως λειτουργεί στην πλέον βασική της μορφή – χωρίς καμιά διαφορά από την υπερκλάση.

7.6 Μια Κλάση για το Background (Φόντο)

Έχουμε βρει μια ωραία φώτο με αληθινά αστέρια την οποία θα χρησιμοποιήσουμε ως φόντο. Μελλοντικά θέλουμε να την κάνουμε να σκρολλάρει κατακόρυφα (και ελπίζουμε ότι μέσα στον πυρετό της μάχης ο παίκτης δεν θα παρατηρήσει ότι είναι η ίδια εικόνα που επαναλαμβάνεται!). Για την ώρα θα αρκεστούμε σε μια στατική απεικόνιση:

```
1  class SpaceBackground:
2    def __init__(self, coord, imagefile):
3      self.shape = pygame.image.load(imagefile)
4      self.coord = coord
5
6    def Show(self, surface):
7      surface.blit(self.shape,self.coord)
8
9    def Scroll(self, speed_y, time):
10     pass
```

Είναι εξαιρετικά απλή φυσικά: Ο constructor δέχεται μόνο το όνομα αρχείου της εικόνας. Έχουμε φροντίσει το μέγεθος να είναι ίδιο με το παράθυρο, οπότε η απεικόνιση θα γίνει στη θέση (0,0). Η μέθοδος `Show` κλασικά εμφανίζει το φόντο πάνω στην επιθυμητή επιφάνεια ενώ η

μέθοδος Scroll θα κάνει το φόντο να κυλάει κατακόρυφα με την επιθυμητή ταχύτητα `speed_y`. Όταν φυσικά μας έρθει η θεία επιφώτιση να την γράψουμε, γιατί για την ώρα το φόντο θα είναι στατικό.

7.7 Το Κύριο Πρόγραμμα

Για την ώρα είναι αρκετά εύκολο:

```
pygame.init()
screenwidth,screenheight = (480,640)
```

Το μέγεθος του παραθύρου μας

```
spaceship_pos = (240, 540)
```

Η αρχική θέση του διαστημοπλοίου μας

```
screen = pygame.display.set_mode((screenwidth,screenheight), DOUBLEBUF, 32)
pygame.display.set_caption("Pygame Invaders")
pygame.key.set_repeat(1,1)
```

Με το `pygame.key.set_repeat(1,1)` ρυθμίζουμε την ταχύτητα αντίδρασης και επανάληψης του πληκτρολογίου σε όσο πιο γρήγορα γίνεται!

```
StarField = SpaceBackground("stars.jpg")
```

Δημιουργούμε το αντικείμενο του φόντου.

```
SpaceShip = SpaceCraft("spaceship2.png",spaceship_pos)
```

Δημιουργούμε το σκάφος μας στην αρχική του θέση

```
clock = pygame.time.Clock()
framerate = 60
```

Δημιουργούμε το γνωστό αντικείμενο "ρολόι" και εισερχόμαστε στον κύριο βρόχο:

```
while True:
  time = clock.tick(framerate)/1000.0
  shipspeed_x = 0
  shipspeed_y = 0
```

Μηδενίζουμε την ταχύτητα του σκάφους. Για να κινηθεί πρέπει να πατήσουμε κάποιο πλήκτρο, διαφορετικά θα πρέπει να μένει ακίνητο στην τελευταία γνωστή του θέση!

```
  for event in pygame.event.get():
    if event.type == QUIT:
      pygame.quit()
      exit()
    if event.type == KEYDOWN:
      # This returns a list of True/False where the index is the
      # code of the key pressed. Fortunately pygame.locals provides
      # symbolics for these codes
      key = pygame.key.get_pressed()
```

```
        if key[K_q]:
          pygame.quit()
          exit()
        if key[K_LEFT]:
          shipspeed_x = -300
        if key[K_RIGHT]:
          shipspeed_x = 300
```

Για την διαχείριση των events, θα χρησιμοποιήσουμε την `pygame.key.get_pressed()`. Αυτή επιστρέφει μια λίστα με τιμές `True / False` όπου ο δείκτης είναι ο κωδικός αριθμός του αντίστοιχου πλήκτρου. Έτσι μπορούμε να διαβάσουμε περισσότερα από ένα πλήκτρα κάθε στιγμή (δηλ. να μετακινούμε το διαστημόπλοιο και να ρίχνουμε βολές ταυτόχρονα). Μη ξεχνάτε ότι το `pygame.locals` δίνει συμβολικά ονόματα σε κάθε κωδικό πλήκτρου, έτσι το πλήκτρο 113 είναι το `K_q` – το πλήκτρο Q που θα μπορεί να πιέσει ο χρήστης για να τερματίσει το παιχνίδι. Για να ελέγξουμε λοιπόν αν πιέζεται τη δεδομένη στιγμή:

```
        key = pygame.key.get_pressed()
        if key[K_q]:
          .... (ο παίκτης βαρέθηκε το παιχνίδι)
```

Ευτυχώς για μας, το pygame διαβάζει με τον ίδιο τρόπο και τα βελάκια (`K_LEFT, K_RIGHT`) και όποιο άλλο πλήκτρο θέλετε. Προφανώς έχουμε επιλέξει τα βελάκια για την κίνηση του σκάφους μας.

Αν πιεστεί το αριστερό βελάκι αλλάζουμε την ταχύτητα σε -300 και για το δεξιό σε 300. Αυτό ισχύει φυσικά μόνο για ένα κύκλο μέσα στο `while`, γιατί όπως είδατε στην αρχή του βρόχου η ταχύτητα μηδενίζεται. Έχουμε όμως ορίσει το repeat του πληκτρολογίου σε μεγάλη ταχύτητα – έτσι το διαστημοπλοιάκι φαίνεται να κινείται αρκετά ομαλά.

```
    SpaceShip.Move(shipspeed_x, shipspeed_y, time)
    StarField.Show(screen)
    SpaceShip.Show(screen)
    pygame.display.update()
```

Κινούμε το διαστημόπλοιο, καλούμε τις αντίστοιχες μεθόδους `Show` των αντικειμένων μας και φυσικά το γνωστό `pygame.display.update()` για να μεταφερθούν όλα στην οθόνη μας. Απλούστατο.

Απλούστατο, αλλά δεν είναι παιχνίδι ακόμα! Που είναι οι εξωγήινοι; Που είναι τα Laser; Που είναι ο ήχος και η μουσική; Για αυτό το τελευταίο θα κάνουμε κάτι αμέσως. Για τα άλλα βέβαια θα χρειαστεί να διαβάσετε τις επόμενες ενότητες.

7.8 Μουσική

Ας γράψουμε μια απλή συνάρτηση για τη μουσική του παιχνιδιού:

```
def PlayMusic(soundfile):
  pygame.mixer.music.load(soundfile)
  pygame.mixer.music.play(-1)
```

και αρκεί να την καλέσουμε οπουδήποτε λίγο πριν το `while`:

```
  PlayMusic(”spaceinvaders.ogg”)
```

7.9 Ευτυχώς, Έχουμε… Προβλήματα

Δεν φτάνει που μετά βίας έχουμε γράψει 70 γραμμές κώδικα, άρχισαν ήδη τα προβλήματα:

- Το διαστημοπλοιάκι κινείται αριστερά – δεξιά και φεύγει και εκτός οθόνης. Τι πρέπει να κάνουμε για να περιορίσουμε την κίνηση;
- Τι γραμμές χρειάζεται για να κινείται και πάνω – κάτω εκτός από αριστερά δεξιά;
- Πως θα εμφανίζονται και θα κινούνται οι εξωγήινοι;

- Πως θα πετύχουμε την κύλιση του φόντου;

Κάποια από τα παραπάνω ερωτήματα θα τα απαντήσουμε στις επόμενες ενότητες.

7.10 Κίνηση Προς Όλες τις Κατευθύνσεις

Αρχίζουμε με το δεύτερο ερώτημα που θέσαμε προηγουμένως: Πως θα κάνουμε το διαστημόπλοιο να κινείται και πάνω – κάτω εκτός από αριστερά – δεξιά. Καθώς φαντάζεστε, αυτό είναι αρκετά εύκολο, αρκεί να προσθέσετε τις παρακάτω γραμμές σε εντελώς προφανές σημείο μέσα στον κύριο βρόχο:

```
        if key[K_UP]:
          shipspeed_y = -300
        if key[K_DOWN]:
          shipspeed_y = 300
```

Ναι, δεν θέλει τίποτε άλλο! Γιατί φυσικά τη ρουτίνα κίνησης την έχουμε ήδη, το μόνο που χρειάζεται είναι να διαβάσουμε το πληκτρολόγιο και για το πάνω και κάτω βελάκι.

Εδώ όμως μπαίνει το πρόβλημα του πρώτου ερωτήματος: Πως θα περιορίσουμε την κίνηση του διαστημοπλοίου, το οποίο — εκτός ότι ήδη έχει την κακή συνήθεια να ξεφεύγει αριστερά/δεξιά — ξεφεύγει πλέον και πάνω κάτω!

Όπως φαντάζεστε οι συντεταγμένες που έχουμε για τη θέση του διαστημοπλοίου δείχνουν αυτή τη στιγμή στην πάνω αριστερή γωνία του. Με την ευκαιρία, αυτό δεν είναι απαραίτητο στο pygame – μπορούμε να ορίσουμε κάποιο άλλο σημείο ως κέντρο του αντικειμένου. Αλλά μια και το έχουμε συνηθίσει, ας συνεχίσουμε έτσι.

Είναι μάλλον εμφανές ότι οι συντεταγμένες αυτές δεν θα πρέπει να γίνουν μικρότερες από το (0,0) που αντιπροσωπεύει την πάνω αριστερή γωνία της οθόνης. Αλλά τι έχουμε να πούμε για τις μέγιστες συντεταγμένες;

Αν είστε λίγο βιαστικοί θα πείτε ότι οι μέγιστες είναι το (480,640) ή αν προτιμάτε να το εκφράσουμε στα δεδομένα του προγράμματος το `(screenwidth, screenheight)`. Δεν έχετε πολύ άδικο, αλλά όπως είπα στην αρχή: είστε λίγο βιαστικοί.

Βλέπετε, το σημείο (480,640) αντιπροσωπεύει το κάτω δεξιά άκρο του παραθύρου, ενώ το σημείο αναφοράς για το διαστημόπλοιο είναι πάνω αριστερά. Βέβαια και στο bouncing ball έχουμε κάνει κάτι αντίστοιχο: αφαιρούμε το πλάτος και το ύψος του αντικειμένου και προσαρμόζουμε

κατάλληλα: στην πραγματικότητα λοιπόν οι μέγιστες συντεταγμένες θα είναι:
`(480 - πλάτος_διαστημοπλοίου, 640-ύψος_διαστημοπλοίου)`.

Τώρα το ωραίο με τα αντικείμενα είναι ότι μπορούμε να φτιάξουμε την κλάση μας με τέτοιο τρόπο ώστε να φροντίζει αυτές τις μικρές αλλά ενοχλητικές λεπτομέρειες εσωτερικά. Δεν υπάρχει λόγος όταν στο κύριο πρόγραμμα δημιουργούμε το περίφημο μας `SpaceShip` object να μην του δώσουμε σαν μέγιστες συντεταγμένες το (480,640) και να το αφήσουμε να καταλάβει μόνο του ότι πρέπει να αναλάβει να κάνει αυτές τις πράξεις ανάλογα με το μέγεθος του. Το μόνο του βέβαια είναι τρόπος του λέγειν, καθώς φυσικά πάλι εμείς θα γράψουμε τον κώδικα.

Πως όμως θα περάσουμε αυτούς τους περιορισμούς; Μια ιδέα είναι να τις δίνουμε ως παραμέτρους κατά τη δημιουργία του αντικειμένου. Τη δεδομένη στιγμή το αντικείμενο μας δημιουργείται με την παρακάτω εντολή:

```
SpaceShip = SpaceCraft("spaceship2.png",spaceship_pos)
```

και μάλλον τώρα θέλουμε κάτι τέτοιο:

```
spaceship_low = (0,0)
spaceship_high = (screenwidth, screenheight)
SpaceShip = SpaceCraft("spaceship2.png",spaceship_pos, spaceship_low, spaceship_high)
```

Βλέπετε ότι περνάμε τις συντεταγμένες ως δύο tuples (δεν είναι κάτι που θα αλλάξουμε), απευθείας στον constructor του `SpaceCraft` class. Για μισό λεπτό όμως, το `SpaceCraft` class δεν έχει constructor: για την ακρίβεια δεν έχει τίποτα εκτός από ένα απλό `pass`. Έτσι απλά εκτελείται ο constructor του `Craft` ο οποίος δεν γνωρίζει τίποτα για συντεταγμένες low και high! Μάλλον ήρθε η ώρα να γράψουμε κάτι στην κλάση `SpaceCraft`, δεν νομίζετε; Σβήστε λοιπόν το `pass` και πάμε να φτιάξουμε constructor:

```
1 class SpaceCraft(Craft):
2   def __init__ (self, imagefile, coord, min_coord, max_coord):
3     super(SpaceCraft,self).__init__(imagefile,coord)
4     self.min_coord = min_coord
5     self.max_coord = (max_coord[0]-self.ship_width, max_coord[1]-self.ship_height)
```

Καθώς βλέπετε, το πρώτο πράγμα που θα κάνει ο constructor του `SpaceCraft` είναι να καλέσει τον αντίστοιχο της γονικής κλάσης `Craft` με την εντολή:

```
    super(SpaceCraft,self).__init__(imagefile,coord)
```

Θυμηθείτε ότι για να δουλέψει αυτό, θα πρέπει η γονική κλάση να προέρχεται από την κλάση `object`, όπως και πράγματι συμβαίνει:

```
class Craft(object):
```

Τα δεδομένα που δίνουμε για τις ελάχιστες και μέγιστες συντεταγμένες αποθηκεύονται στις μεταβλητές (tuple) `self.min_coord` και `self.max_coord`. Για τις ελάχιστες δεν απαιτείται καμιά διόρθωση, ενώ για τις μέγιστες φυσικά γίνεται το γνωστό κόλπο με το πλάτος και ύψος του αντικειμένου. Αυτά τα έχει ήδη υπολογίσει ο constructor του `Craft` για εμάς!

Το γεγονός βέβαια ότι δώσαμε περιορισμούς δεν σημαίνει ότι το διαστημόπλοιο μας τις τηρεί κιόλας. Γιατί θα πρέπει να κάνουμε κάτι και στην `Move` η οποία δεν ελέγχει αν οι τρέχοντες συντεταγμένες είναι εκτός ορίων. Και πως να το κάνει άλλωστε, αφού είναι μόνο η βασική `Move` που έχουμε γράψει για το `Craft` superclass. Ώρα λοιπόν να φτιάξουμε μια `Move` και στο `SpaceCraft`:

```
1  def Move(self, speed_x, speed_y, time):
2    super(SpaceCraft,self).Move(speed_x, speed_y, time)
3    for i in (0,1):
4      if self.rect[i] < self.min_coord[i]:
5        self.rect[i] = self.min_coord[i]
6      if self.rect[i] > self.max_coord[i]:
7        self.rect[i] = self.max_coord[i]
```

Τι κάνουμε εδώ; Μα κάτι πολύ απλό στα αλήθεια. Αφήνουμε την `Move` της γονικής κλάσης να κάνει τη δουλειά της και έπειτα βλέπουμε τις συντεταγμένες που προέκυψαν: Αν είναι μικρότερες ή μεγαλύτερες από το όριο, απλά τις φέρνουμε ακριβώς στο όριο. Πολύ απλά, αν πάτε σε μια άκρη της οθόνης και συνεχίζετε να πιέζετε το βελάκι, το διαστημόπλοιο θα μείνει εκεί, ακίνητο. No motion. No sir. Nada, nope.

Έχοντας ολοκληρώσει τη βασική κίνηση του σκάφους μας, μας μένουν ακόμα:

- Οι βολές μας
- Κάποιο εφέ που να δείχνει ότι έχουμε χτυπηθεί (ηχητικό και οπτικό)
- Η ασπίδα και η μέτρηση του score

Ακόμα δεν έχουμε αγγίξει τους εξωγήινους! Καθώς φαντάζεστε έχουμε αρκετή δουλειά ακόμα!

7.11 Μια Συνάρτηση για τον Ήχο

Αυτό είναι επίσης πολύ εύκολο:

```
def PrepareSound(filename):
  sound = pygame.mixer.Sound(filename)
  return sound
```

Το οποίο θα μπορείτε να το καλέσετε κάπως έτσι:

```
explosion = PrepareSound("explosion.wav")
```

Και φυσικά ο ήχος θα είναι έπειτα διαθέσιμος για χρήση σε οποιοδήποτε σημείο του προγράμματος τον χρειάζεστε:

```
explosion.play()
```