Μανώλης Κιαγιάς

Παιχνίδια σε Python & pygame

Χανιά, 2012

Παιχνίδια σε Python και Pygame

Μανώλης Κιαγιάς, MSc

01/09/2012

Κάθε γνήσιο αντίτυπο φέρει την υπογραφή του συγγραφέα:

Έκδοση: 1η – Χανιά, 01/09/2012

[Αριθμός αντιτύπου: Web Edition]

Είναι Ελεύθερη:

Η Διανομή – Η αναπαραγωγή, διανομή, μετάδοση και παρουσίαση του Έργου σε κοινό

Υπό τις ακόλουθες προϋποθέσεις:

Αναφορά Προέλευσης — Θα πρέπει να αναγνωρίσετε την προέλευση στο έργο σας με τον τρόπο που έχει ορίσει ο δημιουργός του ή το πρόσωπο που σας χορήγησε την άδεια (χωρίς όμως να αφήσετε να εννοηθεί ότι εγκρίνουν με οποιονδήποτε τρόπο εσάς ή τη χρήση του έργου από εσάς).

Μη Εμπορική Χρήση – Δεν μπορείτε να χρησιμοποιήσετε αυτό το έργο για εμπορικούς σκοπούς.

Παρόμοια Διανομή — Αν αλλοιώσετε, τροποποιήσετε ή δημιουργήσετε κάποιο παράγωγο έργο το οποίο βασίζεται στο παρόν έργο, μπορείτε να διανείμετε το αποτέλεσμα μόνο με την ίδια ή παρόμοια με αυτή άδεια.

Με την κατανόηση ότι:

Αποποίηση – Οποιεσδήποτε από τις παραπάνω συνθήκες μπορούν να παρακαμφθούν αν πάρετε την άδεια του δημιουργού ή κατόχου των πνευματικών δικαιωμάτων.

Άλλα Δικαιώματα – Σε καμιά περίπτωση τα ακόλουθα δικαιώματα σας, δεν επηρεάζονται από την Άδεια:

- Η δίκαιη χρήση και αντιμετώπιση του έργου
- Τα ηθικά δικαιώματα του συγγραφέα

- Τα ενδεχόμενα επί του έργου δικαιώματα τρίτων προσώπων, σχετικά με τη χρήση του έργου, όπως για παράδειγμα η δημοσιότητα ή ιδιωτικότητα.

Σημείωση – Για κάθε επαναχρησιμοποίηση ή διανομή, πρέπει να καταστήσετε σαφείς στους άλλους τους όρους της άδειας αυτού του Έργου. Ο καλύτερος τρόπος να το πράξετε αυτό, είναι να δημιουργήσετε ένα σύνδεσμο με το διαδικτυακό τόπο της παρούσας άδειας:

`http://creativecommons.org/licenses/by-nc-sa/3.0/gr/`

Το βιβλίο αυτό στοιχειοθετήθηκε σε XeLaTeX. Ο πηγαίος κώδικας του είναι διαθέσιμος στις δικτυακές τοποθεσίες που αναφέρονται παρακάτω και μέσω mercurial repository.

Επισκεφθείτε το δικτυακό τόπο του μαθήματος και κατεβάστε την τελευταία έκδοση του βιβλίου και διορθώσεις:

`http://pygamegr.wordpress.com`

Στην παραπάνω τοποθεσία θα βρείτε και όλα τα προγράμματα που περιέχονται στο παρόν βιβλίο καθώς και τα συνοδευτικά αρχεία (εικόνες και ήχους) που απαιτούνται για την εκτέλεση τους. Σε περίπτωση προβλήματος χρησιμοποιήστε το mirror site:

`http://www.freebsdworld.gr/files/python-pygame.pdf`

Αφιερώνεται στους μαθητές μου Κοντορίνη Ανδρέα και Τζωρτζάκη Εύα που υπομονετικά άντεξαν εμένα, την python και το pygame για μια ολόκληρη χρονιά!

(Κενή Σελίδα)

Πρόλογος

Οι υπολογιστές έχουν αλλάξει δραματικά από τη δεκαετία του 80

Δυστυχώς η περίφημη "ευκολία χρήσης" των σημερινών υπολογιστών έφερε μαζί της και ένα ανεπιθύμητο αποτέλεσμα: το τέλος της *δημιουργικής* χρήσης. Έχετε σκεφτεί που και πως χρησιμοποιούνται οι υπολογιστές σήμερα;

- Για πρόσβαση στο Internet (και ειδικά στα social media)
- Για παιχνίδια
- Για μουντές εργασίες γραφείου

Κάποτε όμως ο ταπεινός "χρήστης" του υπολογιστή ήταν και ο προγραμματιστής του. Στη δεκαετία του 80 δεν φοβόμασταν τις εντολές, το αντίθετο μάλιστα – τις επιζητούσαμε. Ο έλεγχος του ανθρώπου στη μηχανή, η δυνατότητα να προγραμματίσεις το μηχάνημα να κάνει κάτι πρωτότυπο, είτε υπολογισμό, είτε παιχνίδι, ήχο, γραφικά ήταν η απόλυτη πρόκληση.

Μερικά χρόνια μετά και ο περίφημος "μέσος χρήστης" χρησιμοποιείται πλέον από το μηχάνημα αντί να το χρησιμοποιεί ο ίδιος. Οι υπολογιστές παίζουν με τα νεύρα του αντί να παίζει αυτός με τις δυνατότητες τους. Ταυτόχρονα έχει αγοράσει το παραμύθι ότι ο υπολογιστής είναι μια καταναλωτική συσκευή όπως η τηλεόραση ή το ψυγείο του – και άρα είναι εύκολος στη χρήση.

Θα σας διαψεύσω: ο υπολογιστής γενικής χρήσης δεν ήταν ποτέ εύκολος στη χρήση – και ευτυχώς ούτε θα γίνει. Γιατί αν γίνει, δεν θα είναι υπολογιστής (βλέπε iPad). Αν μια πένσα είναι ένα εργαλείο που μεγεθύνει τη μυϊκή δύναμη, ο υπολογιστής είναι το αντίστοιχο εργαλείο για το μυαλό. Και το μυαλό είναι το δυσκολότερο σε χρήση εργαλείο του ανθρώπου έτσι και αλλιώς.

Για να γίνει ξανά ο υπολογιστής σας δικός σας υπηρέτης και όχι το αντίθετο, πρέπει να μάθετε να τον προγραμματίζετε. Και αυτό αποσκοπούμε μέσα από αυτό το βιβλίο – αλλά με ένα τρόπο ευχάριστο και αποφεύγοντας την πεπατημένη: με προγραμματισμό παιχνιδιών!

Ταυτόχρονα το βιβλίο αυτό είναι μια αναδρομή στη χρυσή εποχή των οικιακών υπολογιστών και των προγραμμάτων που κάποιοι από εμάς γράφαμε στην εφηβεία μας.

Τα "Παιχνίδια σε Python και Pygame" ξεκίνησαν ως μια σειρά άρθρων στο περιοδικό `http://www.deltahacker.gr` – αλλά ήταν από την πρώτη στιγμή σίγουρο ότι θα κυκλοφορούσαν και ως βιβλίο.

Γιατί, πολύ απλά, *το βιβλίο αυτό το χρώσταγα στον εαυτό μου.*

Μανώλης Κιαγιάς, Σεπτέμβριος 2012

Περιεχόμενα

Κατάλογος σχημάτων

Κεφάλαιο 1

Programming; Μα Γιατί;

1.1 Εισαγωγή

Είμαστε σίγουροι ότι οι περισσότεροι από εσάς γεννηθήκατε ή πρωτοχρησιμοποιήσατε υπολογιστή σε μια εποχή που οι δυνατότητες των μηχανημάτων ήταν ήδη μεγάλες: θεωρείτε φυσιολογικό ο υπολογιστής σας να παίζει video, να κατεβάζει ταινίες και μουσική από το Internet με μεγάλη ταχύτητα και να τρέχει τα τελευταία 3D παιχνίδια (αφήνω κατά μέρος τους εκβιασμούς στους γονείς σας να σας αγοράσουν την τελευταία και πανάκριβη κάρτα γραφικών).

Θεωρείτε δεδομένο ότι ο υπολογιστής είναι κάτι που χειριζόμαστε με το ποντίκι και οι νεότεροι από σας δεν έχουν δει ποτέ οθόνη με καθοδικό σωλήνα (εκτός ίσως σε κανένα ξεχασμένο εργαστήριο σχολείου). Αν δεν υπήρχε δε και το βιβλίο που κρατάτε στα χέρια σας, ίσως ποτέ να μην είχατε μπει στον κόπο να ασχοληθείτε με τη γραμμή εντολών ή όλα αυτά τα περίεργα λειτουργικά που θέλουν τόσο κόπο να εγκατασταθούν και τόσες χειροκίνητες ρυθμίσεις.

Όταν αφήνετε όλα αυτά κατά μέρος, ίσως γυρίζετε πίσω στην “ασφάλεια” και τη θαλπωρή των Windows που γνωρίζετε – το πρώτο άλλωστε λειτουργικό που είδατε και που μάλλον νομίζατε ότι ήταν το μοναδικό. Υπάρχουν βέβαια στιγμές που σας εκνευρίζει: Κολλάει ιούς, κρασάρει, καθυστερεί και μόνη λύση είναι να το ταΐζετε συνεχώς με περισσότερο hardware: πιο γρήγορους δίσκους και επεξεργαστές, περισσότερη RAM, ακριβότερες κάρτες γραφικών (ναι, οι γονείς σας κλαίνε στη γωνία καθώς τα γράφω αυτά) και πάει λέγοντας. Πάλι όμως, όταν σκέφτεστε τι σας προσφέρει δεν έχετε να παραπονεθείτε: Κάνει όλα αυτά που θέλετε, μπορείτε να βρείτε ένα πρόγραμμα για το κάθε τι, ότι και αν σκεφτείτε κάποιος θα το έχει γράψει. Αν δεν υπάρχει ελεύθερο… εμ θα βρείτε το κατάλληλο σπαστήρι για να το εγκαταστήσετε. Ακόμα και αν έχετε

εγκαταλείψει τα Windows και χρησιμοποιείτε κάποιο άλλο λειτουργικό, είναι αρκετά πιθανό τα έτοιμα προγράμματα που έρχονται με αυτό (ή που μπορείτε να εγκαταστήσετε) να καλύπτουν τις περισσότερες απαιτήσεις σας. Ναι, ο υπολογιστής είναι ένα κουτί που μπορεί να κάνει το κάθε τι με το κατάλληλο πρόγραμμα. Και είστε ικανοί να εντοπίσετε όποιο πρόγραμμα χρειάζεστε κάθε φορά και να το εγκαταστήσετε. Έτσι δεν είναι;

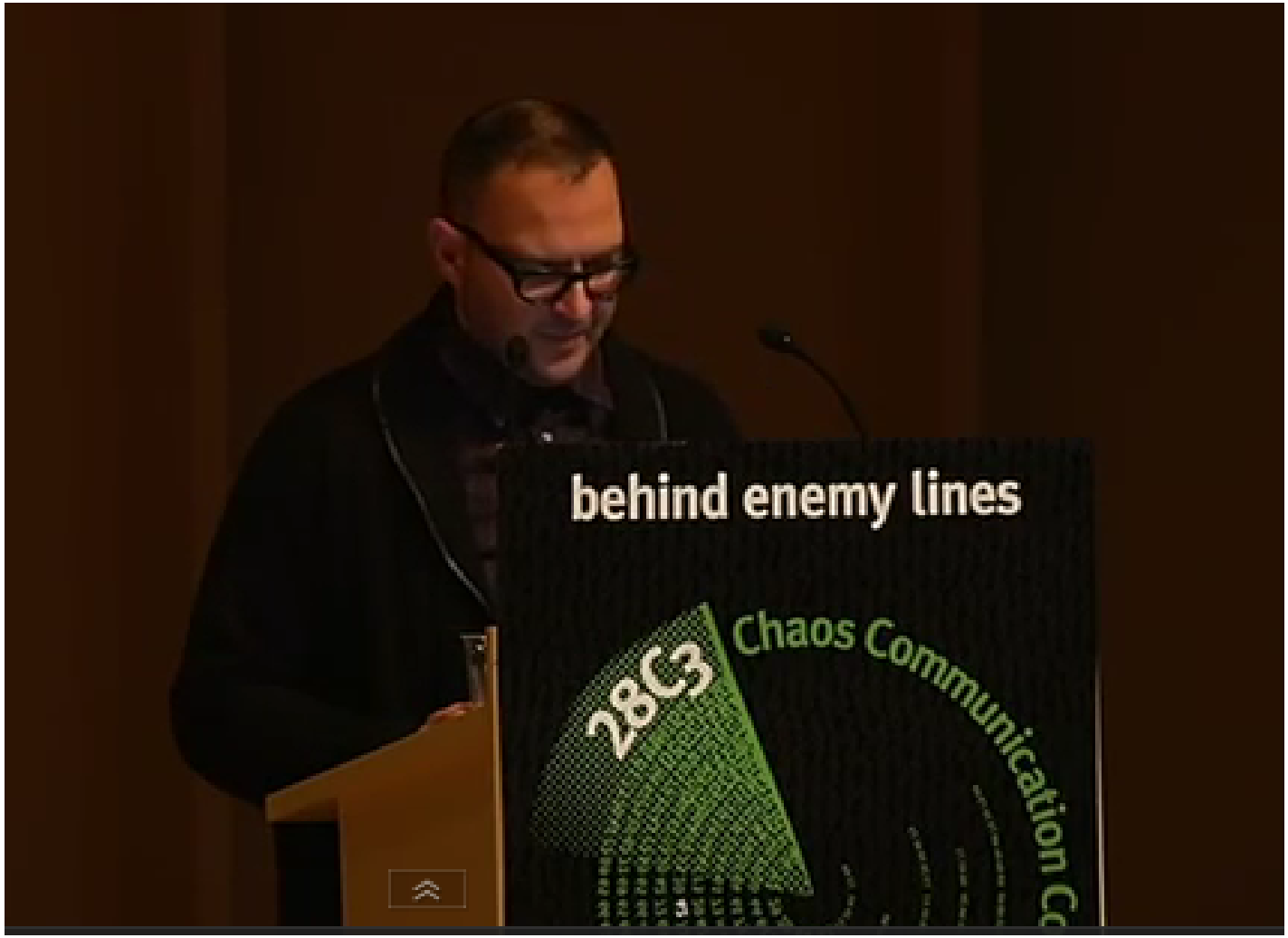
1.2 Ε Όχι, δεν Είναι Ακριβώς Έτσι

Δεν ξέρω αν το έχετε καταλάβει, αλλά όλος ο κόσμος είναι γεμάτος υπολογιστές – και δεν εννοώ τα PC. Το κινητό σας τηλέφωνο είναι ένας υπολογιστής. Η τηλεόραση σας έχει μέσα ένα υπολογιστή. Το αυτοκίνητο σας. Ο φούρνος μικροκυμάτων. Θα μου πείτε, δεν μπορώ να τρέξω το LibreOffice στο φούρνο μικροκυμάτων (αν και εγώ ευχαρίστως θα το έψηνα εκεί μέσα). Ναι, γιατί όλοι αυτοί οι υπολογιστές έχουν ένα ειδικό χαρακτηριστικό:

- Έχουν φτιαχτεί και προγραμματιστεί για μια συγκεκριμένη εργασία

Είναι λοιπόν υπολογιστές ειδικού σκοπού. Και πράγματι, λίγο ενδιαφέρον θα είχε να φτιάξουμε ένα φούρνο μικροκυμάτων που να ψήνει και αρχεία jpg. Δεν ξέρω όμως αν έχετε παρατηρήσει τώρα τελευταία τι συμβαίνει στο χώρο των υπολογιστών "γενικού σκοπού": Έχουμε γεμίσει συσκευές που ενώ είναι υπολογιστές έχουν τεχνητούς περιορισμούς. Το iPad2 που βρίσκεται δίπλα μου δεν μπορεί να τρέξει τίποτα που δεν έχει ελεγχθεί από την Apple και δεν έχει περάσει από το AppStore. Το android κινητό σας χρειάζεται jailbreak για να του πειράξετε τις πιο εσωτερικές του ρυθμίσεις. Αυτή η μανία των χρηστών να χρησιμοποιούν μόνο έτοιμα πράγματα — σε συνδυασμό με τα προβλήματα που δημιουργεί η ελευθερία — ώθησε τις εταιρίες να παράγουν ένα νέο είδος υπολογιστή: Τον **τεχνητά περιορισμένο υπολογιστή** γενικής χρήσης. Και λέμε τεχνητά, γιατί απλά δεν ξέρουμε να φτιάξουμε ένα υπολογιστή γενικής χρήσης που να τρέχει μόνο συγκεκριμένα προγράμματα. Μπορούμε να φτιάξουμε ένα ειδικό υπολογιστή, όπως αυτόν στο φούρνο μικροκυμάτων ή στο αυτοκίνητο. Αλλά ένας υπολογιστής γενικής χρήσης στον οποίο δεν έχουμε εμείς τον πλήρη έλεγχο, είναι σαν ένα φούρνος μικροκυμάτων που αρνείται να ψήσει οτιδήποτε άλλο εκτός από κοτόπουλο. Καλό για τη δίαιτα μας, αλλά σήμερα θέλω να φάω μπιφτέκια!

Δυστυχώς, οι υπολογιστές βαδίζουν προς τα εκεί. Αν δεν με πιστεύετε δείτε το video του Cory Doctorow (Εικόνα 1.1), The coming war on general computation. Πιστεύω θα σας κάνει να ανησυχήσετε.



Εικόνα 1.1: Ο Cory Doctorow παρουσιάζει την ομιλία του “Coming War on General Computation” στο Chaos Communication Congress. Αξίζει να τη δείτε και να την εμπεδώσετε:
http://www.youtube.com/watch?v=yYqkU1y0AYc

Εικόνα 1.2: Texas Instruments 99/4A. Γνωστό ως TI-99/4A. Αν δεν είχα τόσους servers ενεργούς 24/7, πιστεύω ότι αυτό (το πρώτο μου) μηχάνημα θα είχε ρεκόρ… uptime. Ναι, λειτουργεί ακόμα.
http://www.youtube.com/watch?v=b2aXw7xWzgQ

1.3 Στην Αρχή…

Οχι φυσικά, τα πράγματα δεν ήταν πάντα έτσι! Γιατί απλά, το PC με τη μορφή που ξέρετε σήμερα δεν υπήρχε στη δεκαετία του 80. Η για να το πούμε διαφορετικά το IBM PC αν και φτιάχτηκε κάπου τότε, απευθύνονταν μόνο σε επιχειρήσεις και φυσικά η τιμή του ήταν αντίστοιχη. Ήταν ένα ιδιαίτερα βαρετό μηχάνημα, χωρίς ήχο, με ασπρόμαυρη, πράσινη ή πορτοκαλί (!) οθόνη και είχε τόσο ενδιαφέρον όσο ένα ντοκιμαντέρ της τηλεόρασης που το βλέπετε σε εκατοστή επανάληψη στις πέντε το πρωί. Βέβαια το μηχάνημα αυτό δεν είχε τους σημερινούς τεχνητούς περιορισμούς στον προγραμματισμό του, αλλά μη ξεχνάτε δεν ήταν διαδεδομένο στους χομπίστες της εποχής αλλά στις εταιρίες.

Και όλοι εμείς που ξεκινήσαμε στα 80s τι χρησιμοποιούσαμε; Υπήρχε μια ολόκληρη γενιά μηχανημάτων με τα οποία μεγάλωσαν πολλοί από τους προγραμματιστές των οποίων τα προγράμματα χρησιμοποιείτε και σήμερα. Στις εικόνες 1.2, 1.3 και 1.4 θα δείτε μερικά χαρακτηριστικά μηχανήματα. Μπορείτε επίσης να επισκεφτείτε το προσωπικό μου… μουσείο:
http://museum.freebsdgr.org

Κοινό χαρακτηριστικό όλων των παραπάνω μηχανημάτων:

- Το μόνο έτοιμο πρόγραμμα που είχαν ενσωματωμένο, ήταν μια γλώσσα προγραμματισμού!

Εικόνα 1.3: Commodore 64, το μηχάνημα με το ρεκόρ πωλήσεων όλων των εποχών! Και τώρα η (νέα) Commodore το ξαναβγάζει. Αλλά με i7 μέσα. Αν του τρέξετε Windows θα το θεωρήσω ιεροσυλία!

Κατά 99%, ήταν μια διάλεκτος της γνωστής γλώσσας BASIC. Και αυτό που έκαναν όλοι οι χομπίστες της εποχής, όλη μέρα, ήταν να γράφουν προγράμματα. Δικά τους, από βιβλία, από περιοδικά. Ο προγραμματισμός βλέπετε είναι εθιστικός, γιατί περιέχει μέσα την έννοια της δημιουργίας και τον έλεγχο του ανθρώπου πάνω στη μηχανή. Κάτι που οι περισσότεροι έχουν εγκαταλείψει στις μέρες μας (έχετε γράψει κάποιο πρόγραμμα για Windows;) και που τώρα οι εταιρίες προσπαθούν να μας το απαγορέψουν έτσι και αλλιώς (βλέπε iPad).

1.4 Είναι Αγγαρεία ο Προγραμματισμός;

1.4.1 Γιατί η Νέα Γενιά δεν Μαθαίνει Προγραμματισμό;

Για διάφορους λόγους:

- Τα βρίσκει όλα έτοιμα.
- Η πληροφορική έχει ταυτιστεί με τη διδασκαλία προγραμμάτων του τύπου Word και Excel…

Εικόνα 1.4: Sinclair ZX Spectrum, το γνωστό μηχάνημα με τις γομολάστιχες πλήκτρα! Πολύ γνωστό καθώς ήταν και σε προσιτή τιμή.

- Ας μπούμε καλύτερα στο Facebook/twitter/Google plus/YouNameIt social media να περάσουμε την ώρα μας.
- Όχι, δεν θέλω άλλο να γράφω τα ακαταλαβίστικα που κάναμε στην Ανάπτυξη Εφαρμογών σε Προγραμματιστικό Περιβάλλον στο Λύκειο. Και δεν θέλω να ξαναδώ ποτέ την Γλώσσα!

1.4.2 Γιατί οι Παλιοί Μάθαιναν Προγραμματισμό;

- Δεν υπήρχε τίποτα έτοιμο. Έτσι και αλλιώς όποιος αγόραζε υπολογιστή είχε σκοπό να μάθει προγραμματισμό.
- Η MS ήταν ένα μαγαζάκι σαν το εφημεριδοπωλείο της γωνίας και η επεξεργασία κειμένου γινόταν σε γραφομηχανές.
- Το Internet ήταν καλά κρυμμένο στα όνειρα των BSDήδων που σχεδίαζαν το TCP/IP.
- Όλοι εμείς που μαθαίναμε προγραμματισμό, το κάναμε γράφοντας ενδιαφέροντα προγράμματα: Με ήχο, γραφικά, κίνηση. Παιχνίδια!

Ο προγραμματισμός δεν μπορεί να είναι κάτι βαρετό. Σίγουρα, η αλγοριθμική που κάνετε στο ΑΕΠΠ ή στον αντίστοιχο Δομημένο Προγραμματισμό των ΕΠΑΛ είναι απαραίτητη. Αλλά δεν χρειάζεται όλα σας τα προγράμματα να υπολογίζουν κλιμακωτές χρεώσεις σε λογαριασμούς ΔΕΗ και αθροίσματα αριθμών “μέχρι να δώσετε το μηδέν”. Έλεος πια! Ο προγραμματισμός έχει ρουτίνες, αλλά δεν πρέπει με τίποτα να καταντάει για εμάς ρουτίνα. Game programming FTW!

1.5 Καλώς Ήλθατε στην Python!

Εντάξει, πειστήκατε. Θα δώσετε στον προγραμματισμό ακόμα μια ευκαιρία. Αρκεί να μην έχει προγράμματα που θα αθροίζουν αριθμούς και λογαριασμούς ΔΕΗ. Βέβαια, δεν γίνεται να ξεκινήσουμε από το μηδέν γράφοντας killer προγράμματα. Σας υποσχόμαστε όμως ότι θα προσπαθήσουμε να πάμε στο στόχο μας σχετικά γρήγορα και με όσο το δυνατόν πιο ενδιαφέροντα προγράμματα.

Το όχημα μας σε αυτή την αναζήτηση θα είναι η γλώσσα python. Την επιλέξαμε επειδή:

- Είναι κατάλληλη για εκμάθηση ως πρώτη γλώσσα προγραμματισμού.
- Είναι πολύ ισχυρή – παρέχει τόσες έτοιμες βιβλιοθήκες (python modules) που μας επιτρέπουν να κάνουμε τα πάντα.
- Τρέχει σε οτιδήποτε σχεδόν μπορείτε να φανταστείτε. Ναι, ακόμα και στο iPad σας. Στο PC σας. Στα Windows. Στο Linux. Στο FreeBSD.
- Παρέχει εκείνη την ωραία αμεσότητα της BASIC των 80s!

1.5.1 Εγκατάσταση της Python

Αν χρησιμοποιείτε Linux ή FreeBSD ενδεχομένως την έχετε ήδη εγκατεστημένη. Αν όχι, εγκαταστήστε την python2.7 από το repository της διανομής σας. Π.χ. σε debianοειδή λειτουργικά θα χρειαστεί να γράψετε κάτι σαν:

```
# apt-get install python2.7
```

Στο FreeBSD μπορείτε να κάνετε εγκατάσταση από το port:

```
# cd /usr/ports/lang/python27
# make install clean
```

Στα Windows, κατεβάστε την από τη σελίδα `http://www.python.org/download/releases`. Κατεβάστε και εγκαταστήστε την τελευταία έκδοση της σειράς 2.7 για συμβατότητα με αυτά που θα φτιάχνουμε εδώ.

1.5.2 Τα Πρώτα σας Πειράματα στην Python

Η python έρχεται με ένα περιβάλλον άμεσης εκτέλεσης εντολών το *idle*, το οποίο μπορείτε να το εκτελέσετε γράφοντας απλώς idle ή επιλέγοντας το από το μενού του λειτουργικού σας. Ακόμα και αν δεν έχετε το idle, μπορείτε να γράψετε στη γραμμή εντολών:

```
$ python
```

το οποίο πάλι θα ξεκινήσει την python σε κατάσταση άμεσης εκτέλεσης εντολών. Γράψτε `exit()` για να τερματίσετε. Σε κάθε περίπτωση θα δείτε κάτι σαν το παρακάτω:

```
Python 2.7.2 (default, Jun 12 2011, 14:24:46)
Type ”copyright”, ”credits” or ”license()” for more information.
>>>
```

Για αρχή, ας δοκιμάσουμε κάτι απλό. Στην απευθείας εκτέλεση εντολών μπορείτε να κάνετε πράξεις απευθείας:

```
>>> 4/2
2

>>> 5*3
15

>>> 1/2
0
```

Μηδέν; Αν δεν το καταλάβατε, η διαίρεση που γίνεται είναι ακέραια, καθώς η python βλέπει ότι τα ορίσματα είναι ακέραια. Αν κάνετε ένα αριθμό δεκαδικό, θα πάρετε την απάντηση που περιμένατε:

```
>>> 1/2.0
0.5
```

Το υπόλοιπο της διαίρεσης, ο γνωστός τελεστής mod, στην python είναι το %:

```
>>> 1 % 2
1

>>> 5 % 3
2
```

Ενώ αν θέλετε να υψώσετε ένα αριθμό σε μια δύναμη:

```
>>> 2**24

16777216
```

Ακόμα πιο ενδιαφέρον, είναι ότι η python μπορεί να κάνει απευθείας πράξεις σε δεκαεξαδικό, οκταδικό και δυαδικό. Για δεκαεξαδικό, γράψτε τον αριθμό ξεκινώντας με **0x** και για δυαδικό με **0b**:

```
>>> 0xFF
255

>>> 0b1100
12
```

1.5.3 Μεταβλητές

Όπως πιθανώς θα ξέρετε, μια μεταβλητή είναι ένα όνομα που αντιπροσωπεύει μια τιμή. Χρησιμοποιώντας μεταβλητές είναι δυνατόν να φτιάξουμε προγράμματα που λύνουν ένα πρόβλημα με γενικό τρόπο, αφού μπορούμε να αλλάζουμε την τιμή τους σε κάθε εκτέλεση του προγράμματος. Για παράδειγμα:

```
>>> a=5
>>> a*5
25
>>> a**3
125
```

1.5.4 Είσοδος από το Χρήστη και Εμφάνιση στην Οθόνη

Με την `input` μπορούμε να ζητήσουμε από το χρήστη μια τιμή την ώρα που το πρόγραμμα εκτελείται. Η τιμή αυτή μπορεί να αποθηκευθεί απευθείας σε μια μεταβλητή όπως φαίνεται παρακάτω:

```
>>> age=input("What is your age? ")
What is your age? 24
>>> print age
24
```

Όπως καταλάβατε, η εντολή `print` χρησιμοποιείται για να τυπώσουμε κάτι στην οθόνη. Μπορεί να είναι ένα μήνυμα (σε εισαγωγικά), μια τιμή, ή μια μεταβλητή. Ή και συνδυασμός των παραπάνω:

```
>>> print age,"is not too bad!"
24 is not too bad!
```

1.5.5 Το Πρώτο σας Πρόγραμμα: HelloWorld!

Εντάξει, μια πιο καλή παραλλαγή έστω. Αν χρησιμοποιείτε το `idle`, επιλέξτε από το μενού `File => New Window` για να βρεθείτε σε ένα απλό editor. Αλλά μπορείτε να χρησιμοποιήσετε και όποιο άλλο text editor θέλετε. Γράψτε:

```
1 name=raw_input("What is your name? ")
2 age=input("What is your age? ")
3 print "Pleased to meet you ",name
4 print age, " is not too bad!"
```

Αποθηκεύστε το ως `hello.py`. Στο `idle`, απλά πιέστε `F5` για να εκτελεστεί το πρόγραμμα. Σε UNIXοειδή λειτουργικά, μπορείτε επίσης να το τρέξετε από το τερματικό:

```
$ python hello.py
```

Τι είναι το `raw_input`; Πειραματιστείτε βάζοντας σκέτο `input` και δείτε τι θα γίνει. Μετά, δώστε το όνομα σας σε εισαγωγικά. Μπορείτε να βρείτε γιατί συμβαίνει αυτό; Θα το δούμε στη συνέχεια του κεφαλαίου.

Ας γράψουμε τώρα το πρώτο μας text παιχνίδι…

1.6 Το Πρώτο Σας Παιχνίδι: Βρες τον Αριθμό

Γράψτε το παρακάτω και αποθηκεύστε το ως `guess.py` ή όπως αλλιώς θέλετε. Εκτελέστε το όπως προηγουμένως.

```
#
# Βρες τον αριθμό. Απόπειρα 1
#
import random
thenumber = random.randint(1,50)
print "Έχω σκεφτεί ένα αριθμό από το 1 ως το 50."
print "Μπορείς να τον βρεις;"
guess = 0
while guess != thenumber:
  guess = input("Δώσε τον αριθμό: ")
  if guess > thenumber:
    print "Έδωσες μεγαλύτερο αριθμό!"
  if guess < thenumber:
    print "Έδωσες μικρότερο αριθμό!"
  if guess == thenumber:
    print "Τον βρήκες!!!"
```

Προσοχή! Τα κενά είναι σημαντικά. Η python δεν έχει άγκιστρα, αγκύλες ή οτιδήποτε άλλο χρησιμοποιούν οι άλλες γλώσσες για να ξεχωρίζουν ποιες εντολές ανήκουν σε μια *σύνθετη εντολή*. Αντίθετα, χρησιμοποιεί τα κενά ή τα tabs για τον ίδιο σκοπό. Δείτε για παράδειγμα ότι όλες οι εντολές κάτω από τη `while` είναι δύο κενά διαστήματα πιο μέσα: σημαίνει ότι όλες ανήκουν στη `while`. Κάτω από κάθε `if`, η εντολή είναι δύο διαστήματα πιο μέσα: ανήκει στην `if`. Σε όλες πρακτικά τις γλώσσες πρέπει να στοιχίζουμε το πρόγραμμα μας σωστά ώστε να είναι ευανάγνωστο. Απλώς στην python αυτό είναι υποχρεωτικό καθώς η στοίχιση είναι μέρος του συντακτικού της γλώσσας!

1.6.1 Ανάλυση Γραμμή – Γραμμή

```
import random
```

Δηλώνουμε στην python ότι θέλουμε να χρησιμοποιήσουμε τη βιβλιοθήκη (module) `random`. Μέσα σε αυτή περιέχονται συναρτήσεις που παράγουν τυχαίους αριθμούς!

```
thenumber = random.randint(1,50)
```

Χρησιμοποιούμε την `randint` για να παράγουμε ένα ακέραιο τυχαίο αριθμό από το 1 ως το 50 και τον αποθηκεύουμε στη μεταβλητή `thenumber`. Παρατηρήστε πως καλούμε τη συνάρτηση `randint`.

```
print "Έχω σκεφτεί ένα αριθμό από το 1 ως το 50."
print "Μπορείς να τον βρεις;"
```

Τα γνωστά μηνύματα, δεν χρειάζονται εξήγηση.

```
guess = 0
```

Η αρχική μας πρόβλεψη. Η μεταβλητή `guess` θα κρατάει τον αριθμό που δίνουμε κάθε φορά ως μαντεψιά. Της δίνουμε μια αρχική τιμή 0 ώστε σίγουρα να μην ανήκει στο διάστημα 1 ως 50 που παράγει το πρόγραμμα. Έτσι εξασφαλίζουμε ότι η επόμενη εντολή:

```
while guess != thenumber:
```

Θα εκτελεστεί, καθώς λέει: “Κάνε τα παρακάτω, όσο ο αριθμός που έδωσε ο χρήστης δεν είναι ίδιος (!=) με αυτό που σκέφτηκε η `randint`”. Και καθώς η αρχική τιμή είναι το μηδέν, σίγουρα το πρόγραμμα θα προχωρήσει:

```
guess = input("Δώσε τον αριθμό: ")
```

Ζητάει τον αριθμό από τον παίκτη.

```
if guess > thenumber:
  print "Έδωσες μεγαλύτερο αριθμό!"
```

Αν ο χρήστης έδωσε μεγαλύτερο αριθμό, τυπώνεται το αντίστοιχο μήνυμα.

```
if guess < thenumber:
  print "Έδωσες μικρότερο αριθμό!"
```

Αν ο χρήστης έδωσε μικρότερο αριθμό, τυπώνεται πάλι αντίστοιχο μήνυμα.

```
if guess == thenumber:
  print "Τον βρήκες!!!"
```

Αν ο χρήστης βρήκε τον αριθμό, τυπώνεται το… νικητήριο μήνυμα. Παρατηρήστε ότι ο έλεγχος ισότητας γίνεται με δύο ίσον (==). Το ένα ίσον χρησιμοποιείται για να δώσουμε τιμή σε μεταβλητή.

Σε όλες τις περιπτώσεις, το πρόγραμμα επιστρέφει στην `while` όπου ξεκίνησε. Αν όμως ο αριθμός που έδωσε ο χρήστης είναι ίσος με το `thenumber`, η συνθήκη δεν ισχύει πλέον και το πρόγραμμα τερματίζει.

Ξαναδιαβάστε τώρα το πρόγραμμα προσεκτικά. Βλέπετε κάτι να περιττεύει; Αν όχι σκεφτείτε λίγο το τελευταίο `if`. Χρειάζεται στα αλήθεια; Καθώς ο βρόχος `while` τερματίζει μόνο όταν ο παίκτης βρει τον αριθμό, το τελευταίο if μπορεί να παραλειφθεί εντελώς, και το μήνυμα επιτυχίας να μπει μόνο του, αμέσως μετά το βρόχο. Φτάνουμε έτσι στην Απόπειρα 2 του προγράμματος:

```
#
# Βρες τον αριθμό. Απόπειρα 2
#
import random
thenumber = random.randint(1,50)
print "Έχω σκεφτεί ένα αριθμό από το 1 ως το 50."
print "Μπορείς να τον βρεις;"
guess = 0
while guess != thenumber:
  guess=input("Δώσε τον αριθμό: ")
  if guess > thenumber:
    print "Έδωσες μεγαλύτερο αριθμό!"
  if guess < thenumber:
    print "Έδωσες μικρότερο αριθμό!"
print "Τον βρήκες!!!"
```

1.7 Επεκτείνοντας το Παιχνίδι

Ακολουθούν μερικές βελτιώσεις για το απλό μας παιχνίδι – στην πραγματικότητα μια ευκαιρία να μάθουμε μερικά ακόμα στοιχεία της python!

1.7.1 Όνομα Χρήστη και Νικητήριο Μήνυμα

Μπορούμε εύκολα να επεκτείνουμε το παιχνίδι μας ώστε να ρωτάει το χρήστη το όνομα του στην αρχή και να το εμφανίζει στο νικητήριο μήνυμα: “Συγχαρητήρια Γιώργο το βρήκες!”

Πολύ απλό νομίζουμε, αρκεί να βάλετε την παρακάτω γραμμή πριν από το πρώτο `print`:

```
name = input("Δώσε το όνομα σου: ")
```

Και να αλλάξετε την τελευταία γραμμή σε:

```
print "Συγχαρητήρια", name, "τον βρήκες!"
```

Για εκτελέστε το όμως! Θα διαπιστώσετε με έκπληξη ότι δεν δέχεται το όνομα σας, εκτός αν το δώσετε μέσα σε εισαγωγικά. Όχι, δεν είναι ότι η python έχει προσωπικά μαζί σας. Απλώς η εντολή `input` ερμηνεύει την είσοδο ως παράσταση και ψάχνει να βρει το όνομα σας ως κάτι που έχει ήδη οριστεί στο πρόγραμμα! Αν δώσετε ως όνομα το `thenumber`, μεταβλητή που ήδη έχει ορισθεί στο πρόγραμμα μας, η python θα το δεχθεί. Δοκιμάστε και θα δείτε το αποτέλεσμα!

Φυσικά, αυτό δεν είναι κάτι που θέλουμε. Και ούτε μας αρέσει η ιδέα να βάζουμε εισαγωγικά γύρω από το όνομα μας ή οποιοδήποτε αλφαριθμητικό θέλουμε να δώσουμε στην είσοδο. Η λύση σε αυτό είναι η `raw_input`:

```
name = raw_input("Δώσε το όνομα σου: ")
```

Θα χρησιμοποιούμε από εδώ και πέρα την `raw_input` για είσοδο αλφαριθμητικών

1.7.2 Προσθήκη Μετρητή Προσπαθειών

Θα προσθέσουμε μια μεταβλητή `count` στην οποία θα δώσουμε αρχικά τιμή 0 και κάθε φορά που ο χρήστης θα δίνει ένα αριθμό θα αυξάνει κατά 1 (`count = count + 1`). Όταν ο χρήστης βρει τον αριθμό, το πρόγραμμα θα τυπώνει και πόσες προσπάθειες χρειάστηκε.

Και αυτό είναι αρκετά εύκολο. Δείτε την πλήρη τρίτη εκδοχή του προγράμματος:

```
#
# Βρες τον αριθμό. Απόπειρα 3
#
import random
thenumber = random.randint(1,50)
name = raw_input("Δώσε το όνομα σου: ")
print "Έχω σκεφτεί ένα αριθμό από το 1 ως το 50."
print "Μπορείς να τον βρεις;"
guess = 0
tries = 0
while guess != thenumber:
  tries = tries + 1
  guess=input("Δώσε τον αριθμό: ")
  if guess > thenumber:
    print "Έδωσες μεγαλύτερο αριθμό!"
  if guess < thenumber:
    print "Έδωσες μικρότερο αριθμό!"
print "Συγχαρητήρια", name, "τον βρήκες σε", tries, "προσπάθειες!"
```

Η γραμμή που αυξάνει τη μεταβλητή `tries` μπορεί να γραφεί και έτσι:

```
  tries += 1
```

Είναι μια συντομογραφία που θα συναντήσετε συχνά. Φυσικά το ίδιο μπορεί να γίνει και με τους άλλους αριθμητικούς τελεστές (για αφαίρεση, πολλαπλασιασμό κλπ) αλλά το παραπάνω είναι το πιο συχνό.

1.7.3 Κριτική στην… Ικανότητα του Παίκτη

Θα επεκτείνουμε το πρόγραμμα ώστε αν ο χρήστης βρεί τον αριθμό σε μια προσπάθεια θα του τυπώνει το μήνυμα “Beginner’s luck!”, αν το βρει σε πέντε ή λιγότερες προσπάθειες το μήνυμα “Είσαι γρήγορος” και αν το βρει σε πάνω από πέντε, το μήνυμα “Η γιαγιά μου παίζει πιο καλά!”

Μια πρώτη σκέψη ίσως είναι αυτή (οι γραμμές φυσικά μπαίνουν στο τέλος):

```
1 if tries == 1:
2   print ”Beginner’s luck!”
3 if tries <= 5:
4   print ”Είσαι γρήγορος!”
5 if tries > 5:
6   print ”Η γιαγιά μου παίζει πιο καλά!”
```

Έχουμε λίγες γραμμές κώδικα, και ήδη έχουμε bugs. Αν βρείτε τον αριθμό σε μια προσπάθεια, θα πάρετε δύο μηνύματα! “Beginner’s luck” και “Είσαι γρήγορος!” Φυσικά, καθώς το 1 είναι επίσης και μικρότερο από 5, εκτελούνται δύο εντολές `if`. Εδώ υπάρχουν δύο πιθανές λύσεις:

- Να αλλάξουμε τη δεύτερη συνθήκη ώστε να εκτελείται όταν η `tries` είναι μικρότερη από 5 και ταυτόχρονα διαφορετική του 1 (ή μεγαλύτερη του 1)
- Να φτιάξουμε την `if` με τέτοιο τρόπο ώστε αν εκτελεστεί μια συνθήκη να μην εκτελείται καμιά άλλη.

Θα μπορούσαμε να το γράψουμε έτσι:

```
1 if tries == 1:
2   print ”Beginners luck!”
3 if (tries <= 5 and tries != 1):
4   print ”Είσαι γρήγορος!”
5 if tries > 5:
6   print ”Η γιαγιά μου παίζει πιο καλά!”
```

Φτιάξαμε εδώ μια σύνθετη συνθήκη χρησιμοποιώντας τον λογικό τελεστή `and` ο οποίος πρακτικά σημαίνει ότι θα είναι αληθής και θα εκτελεστεί μόνο αν συμβαίνουν και τα δύο μέρη: και το `tries` να είναι μικρότερο ή ίσο από 5 και διαφορετικό από το 1 (θα μπορούσατε επίσης να το γράψετε και μεγαλύτερο από το 1). Καθώς φαντάζεστε υπάρχουν και οι λογικοί τελεστές `or` και `not` που θα δούμε σε παραδείγματα αργότερα.

Ένας άλλος τρόπος είναι να χρησιμοποιήσουμε μια πιο πλήρη μορφή της `if`, που περιέχει επίσης το `elif` (*else if*, αλλιώς αν) και το `else`:

```
1 if tries == 1:
2   print "Beginners luck!"
3 elif tries <= 5:
4   print "Είσαι γρήγορος!"
5 else:
6   print "Η γιαγιά μου παίζει πιο καλά!"
```

Εδώ η συνθήκη που βρίσκεται στο `elif` (αλλιώς αν) θα εκτελεστεί μόνο αν το `tries` είναι μικρότερο του 5, και δεν έχει εκτελεστεί το πρώτο `if`. Η δομή της `if-elif` εξασφαλίζει ότι μόλις εκτελεστεί μια οποιαδήποτε από τις συνθήκες (είτε ή `if`, είτε κάποια από τις `elif` - και μπορούμε να βάλουμε πολλές `elif`) οι υπόλοιπες περιπτώσεις δεν εξετάζονται. Αν υπάρχουν εντολές κάτω από τη δομή `if-elif-else`, η εκτέλεση θα συνεχιστεί από εκεί. Καθώς φαντάζεστε, η εντολή `else` εκτελείται μόνο όταν δεν εκτελεστεί καμιά από τις `elif` ή η αρχική `if`.

Μπορείτε να βρείτε το πλήρες πρόγραμμα στο παράρτημα, σελ. 163.

Κεφάλαιο 2

Νυχτερινή Περιπέτεια!

2.1 Εισαγωγή στις Λίστες

Πριν ξεκινήσουμε να γράφουμε το επόμενο μας παιχνίδι, πρέπει να μιλήσουμε λίγο για μια σημαντική δομή δεδομένων της python: Τις *λίστες*. Βλέπετε οι απλές μεταβλητές από μόνες τους δεν είναι επαρκείς για να χειριστούμε πλήθος δεδομένων. Το σημαντικότερο μειονέκτημα τους είναι ότι δεν μπορούμε να επεξεργαστούμε μαζικά και με τον ίδιο τρόπο απλές μεταβλητές που περιέχουν δεδομένα στα οποία θέλουμε να εκτελέσουμε την ίδια διαδικασία. Π.χ. να έχουμε μια σειρά από τιμές προϊόντων τις οποίες θέλουμε όλες να αυξήσουμε κατά 10%.

Αν το κάνουμε αυτό με απλές μεταβλητές, θα πρέπει να έχουμε μια μεταβλητή ανά προϊόν και να γράψουμε τον ίδιο κώδικα όσες φορές είναι και τα προϊόντα. Προφανώς αυτό δεν εξυπηρετεί.

Σε πολλές γλώσσες προγραμματισμού, βασική δομή για να επιτύχουμε μαζική επεξεργασία δεδομένων είναι ο πίνακας. Η python ωστόσο μας παρέχει άλλες δομές και μια από τις σημαντικότερες είναι η λίστα.

Η λίστα ανήκει σε ένα είδος δομής δεδομένων που ονομάζεται *sequence ή ακολουθία* αν το προτιμάτε. Είναι εύκολο να φτιάξετε μια λίστα:

```
shoppinglist = [ "Cheese" , "Rice", "Coffee", "Milk", "Camba" ]
```

Και εξίσου εύκολο να διατρέψετε όλα τα στοιχεία της:

```
for element in shoppinglist:
  print element
```

Είναι ακόμα αστεία εύκολο να δούμε αν κάτι ανήκει σε μια λίστα ή όχι:

```
shoppinglist = [ ”Cheese”, ”Rice”, ”Coffee”, ”Milk”, ”Camba” ]
element = raw_input(”What are you buying today? ”)
if element in shoppinglist:
  print ”Yes, this is on the list”
else:
  print ”Not on the list, you don’t need it”
```

Αν έχετε συνηθίσει σε άλλες γλώσσες να χρησιμοποιείτε την *for* για να δημιουργείτε βρόχο με μετρητή, στην python θα το κάνετε ως εξής:

```
for i in [1,2,3,4,5,6,7,8,9,10]:
  print i
```

Αυτό δεν βολεύει αν θέλουμε να μετρήσουμε μέχρι το 1000, οπότε θα γράφαμε:

```
for i in range(1,1001):
  print i
```

Καθώς καταλαβαίνετε, η συνάρτηση `range` παράγει μια λίστα με τα στοιχεία από 1 ως 1000 (όχι ως το 1001 όπως νομίζετε διαβάζοντας το παράδειγμα). Καλό είναι να γνωρίζετε ότι μια λίστα στην python μπορεί να περιέχει οτιδήποτε: αριθμούς, αλφαριθμητικά, αντικείμενα (που θα δούμε αργότερα) ακόμα και άλλες λίστες! Εκτός από τις λίστες, η python παρέχει επίσης και τα *tuples*, τα οποία είναι παρόμοια όμως μετά τη δημιουργία τους δεν μπορούν να μεταβληθούν. Σε μια λίστα μπορούμε όπως θα δούμε σε επόμενα άρθρα να προσθέσουμε και να διαγράψουμε στοιχεία και να εκτελέσουμε μια σειρά από χρήσιμες λειτουργίες (π.χ. ταξινόμηση).

Είμαστε τώρα έτοιμοι να ξεκινήσουμε το επόμενο μας παιχνίδι: Την Νυχτερινή Περιπέτεια ή Adventure!

2.2 Η Περιπέτεια (The Adventure)

Ότι έχουμε δει ως εδώ είναι μια μικρή εισαγωγή στον προγραμματισμό και την python. Πιο πολύ βέβαια είναι μια συνειδητή προσπάθεια από μέρους μας να σας ωθήσουμε υποσυνείδητα στον προγραμματισμό. Θυμηθείτε: κάθε φορά που τρέχετε ένα πρόγραμμα στον υπολογιστή, εκτελείτε κάτι που δημιουργήθηκε από κάποιον άλλο. Γιατί όχι από εσάς; Αν ο υπολογιστής είναι μια προγραμματιζόμενη μηχανή γενικής χρήσης, γιατί πρέπει εμείς να αφήσουμε το “προγραμματιζόμενη” στους άλλους; Μήπως είναι καλύτεροι από εμάς; Δεν σας κρύβω ότι όλοι εμείς που ξεκινήσαμε πολύ παλιά τον προγραμματισμό, είχαμε τα πλεονεκτήματα της αμεσότητας και απλότητας των μηχανημάτων. Επίσης δεν είχαμε και τίποτα άλλο να κάνουμε με αυτά! Εξ’ αρχής, τα είχαμε αγοράσει για να μάθουμε προγραμματισμό. Τώρα όμως, μέσα στο πλήθος των πληροφοριών, οι υπολογιστές έχουν υποβιβαστεί σε συσκευές. Έχουμε όλοι γυρίσει στα έτοιμα πράγματα ενώ έχουμε σημαντικά νέα πλεονεκτήματα: Γρήγορα μηχανήματα, τεράστιες μνήμες, αφθονία ελεύθερων γλωσσών προγραμματισμού και φυσικά το Internet για να ψάχνουμε! Φαίνεται όμως ότι η πολύ αφθονία βλάπτει.

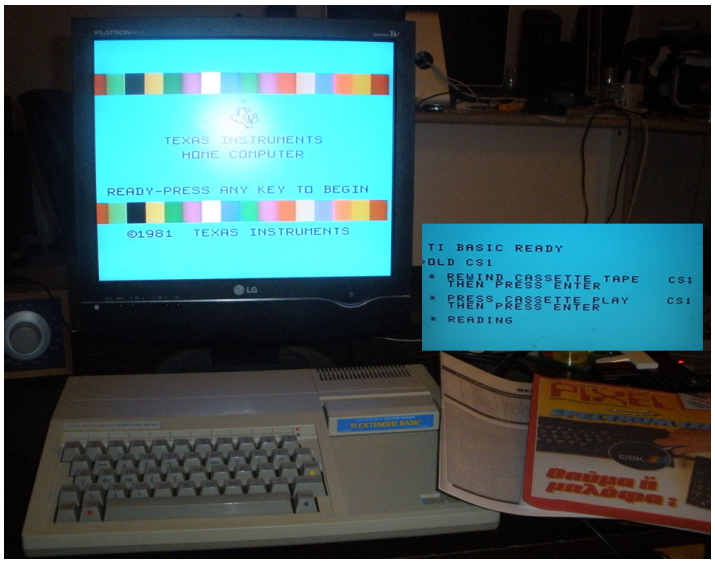
Ευτυχώς, υπάρχει η python και φυσικά το βιβλίο που κρατάτε στα χέρια σας! Γιατί θα φέρουμε κάτι από αυτή τη χαμένη αμεσότητα πίσω, γράφοντας τα παλιά καλά (θεός να τα κάνει) παιχνίδια στη σύγχρονη εκδοχή τους. Τα πλεονεκτήματα θα είναι πολλά:

- Θα μάθετε να σκέφτεστε σαν προγραμματιστής (βασικά, θα μάθετε να σκέφτεστε. Τελεία.)
- Θα μάθετε λίγη python και pygame για να συνεχίσετε να γράφετε δικά σας παιχνίδια και παραλλαγές αυτών που θα σας δείξουμε.
- Θα κολλήσετε τόσο άσχημα με τον προγραμματισμό, που θα μένετε όλη μέρα σπίτι σας, δεν θα απαντάτε στο τηλέφωνο και θα τρέφεστε αποκλειστικά με πίτσες, καφέδες και αναψυκτικά τύπου κόλα.

OK, το τελευταίο ίσως δεν είναι ιδιαίτερα καλό αλλά είμαστε σίγουροι ότι μπορείτε να το ελέγξετε. Πριν συνεχίσετε να διαβάζετε το κεφάλαιο, βεβαιωθείτε ότι έχετε κάνει και κατανοήσει τις ασκήσεις του πρώτου κεφαλαίου και φυσικά τα παραδείγματα με τις λίστες που παρουσιάσαμε προηγουμένως. Θα τα χρειαστούμε άμεσα.

2.2.1 Εισαγωγή στις Περιπέτειες Κειμένου (Text Adventures)

Μη σας φαίνεται απίστευτο, το δεύτερο μας παιχνίδι — αισθητά μεγαλύτερο από το number guess — είναι ένα παιχνίδι περιπέτειας. Όχι μη πάει ο νου σας σε κάτι με τρισδιάστατα γραφικά τύπου Final Fantasy! (αυτό θα το γράψετε εσείς, μετά που θα διαβάσετε αυτό και πολλά(!) ακόμα βιβλία). Ένα adventure στην απλούστερη του μορφή περιέχει μόνο κείμενο. Τα λεγόμενα *text adventures* ήταν πολύ της μόδας πριν μερικές δεκαετίες. Βλέπετε, υπήρχαν τότε υπολογιστές που δεν μπορούσαν να δείξουν παρά μόνο κείμενο στις οθόνες τους (όσο σοκαριστικό



Εικόνα 2.1: Το μηχάνημα στο οποία έγραψα — το 1986 — την πρώτη έκδοση του προγράμματος που φαίνεται εδώ. Ναι, το ξέθαψα από την ντουλάπα και το φόρτωσα (από την κασέτα όπως φαίνεται δεξιά, μετά από 26 χρόνια!) για χάρη σας! Και ναι, δουλεύει ακόμα. Βλέπετε και τις σελίδες του Pixel με το αντίστοιχο θέμα.

και αν ακούγεται αυτό!) Αλλά και εκείνοι που μπορούσαν να δείξουν γραφικά, ε δεν ήταν ακριβώς και… φωτογραφικής ποιότητας. Σε κάθε περίπτωση επειδή ο προγραμματιστής δεν ήταν απαραίτητα και γραφίστας, τα text adventures κυριαρχούσαν.

Η ιδέα πίσω από ένα adventure κειμένου είναι ότι ο παίκτης ξεκινάει σε ένα δωμάτιο. Ο υπολογιστής τυπώνει μια περιγραφή του χώρου με τις πιθανές εξόδους (πόρτες, σκάλες κλπ) και ο παίκτης αποφασίζει την επόμενη του κίνηση. Όταν εισέρχεται σε ένα νέο δωμάτιο, η διαδικασία επαναλαμβάνεται, μέχρι είτε ο παίκτης να κερδίσει (π.χ. να βρει το αντικείμενο που είναι ο στόχος του παιχνιδιού ή να φτάσει στο κατάλληλο δωμάτιο) είτε να χάσει (να τον πιάσει το τέρας, το alien ή η μαμά του – που ορισμένοι θεωρούν χειρότερο τέλος και από το alien). Σε μερικά adventures υπάρχουν περισσότεροι από ένας τρόποι για να κερδίσει κανείς και σίγουρα υπάρχουν πολλοί τρόποι να χάσει! Σε πιο εξελιγμένες μορφές, ένα text adventure απαιτεί από το χρήστη να συλλέγει αντικείμενα τα οποία θα χρειαστεί σε άλλα δωμάτια. Και φυσικά χρησιμοποιούνται περισσότερες εντολές για να εκπληρώσουν αυτό το σκοπό.

Το adventure που θα γράψουμε εμείς βασίζεται στο πρωτότυπο που είχε δημοσιευθεί στο άρθρο “Παιχνίδια Περιπέτειας” του Στάθη Ευθυμίου στο περιοδικό Pixel τον Ιανουάριο του 1986! Χρησιμοποιούμε τον ίδιο χάρτη και τις πρωτότυπες περιγραφές των δωματίων, αλλά φυσικά το γράφουμε σε python και όχι σε BASIC.

Δεν σας κρύβω ότι διαβάζοντας τότε το άρθρο, άρχισα να σκέφτομαι πως θα μπορούσα να το υλοποιήσω στον τότε υπολογιστή μου. Υπήρχε βέβαια ένα listing σε BASIC στο περιοδικό, αλλά εκτός ότι δεν ήταν συμβατό με το δικό μου μηχάνημα (που βλέπετε στην εικόνα 2.1),

Εικόνα 2.2: Ένα μικρό κομμάτι από το listing του αρχικού προγράμματος Adventure, © 1986! Φαίνεται επίσης και η εκτέλεση του προγράμματος όπου βλέπετε ότι δέχεται κατευθύνσεις του τύπου 1,2,3,4.

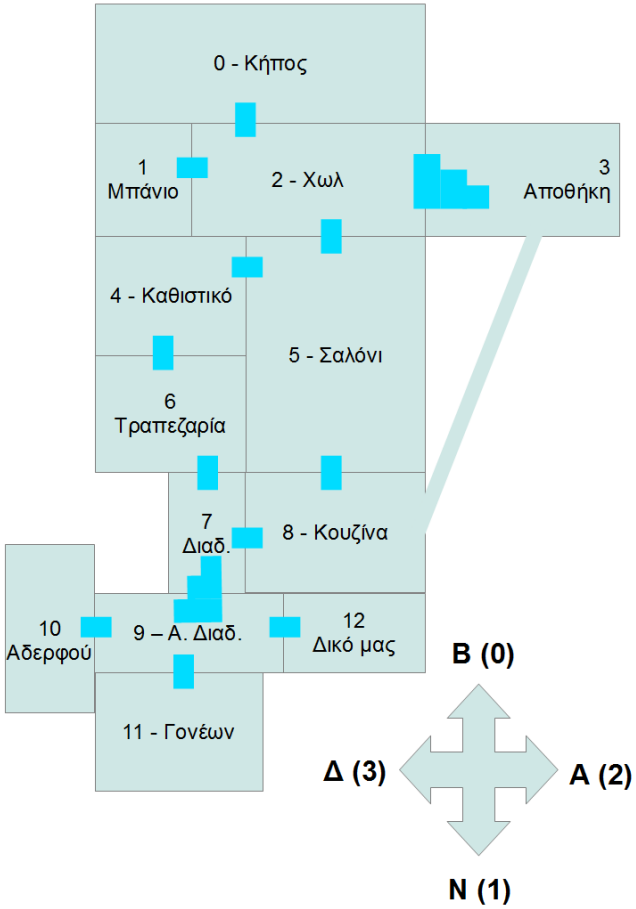
χρησιμοποιούσε και μια λογική που δεν μου άρεσε ιδιαίτερα. Ήμουν σίγουρος ότι μπορούσα να βρω μια πιο ωραία προγραμματιστικά λύση και πράγματι πέρασα μερικές μέρες σκεπτόμενος το πρόβλημα σαν ένα background task στο μυαλό μου (κατανάλωνε αρκετούς πόρους πάντως!) Ένα βράδυ πετάχτηκα από το κρεβάτι έχοντας την απάντηση που χρειαζόμουν! Η πρώτη έκδοση είχε γραφτεί μέχρι το πρωί…

Ο ίδιος αυτός τρόπος — αλλά προσαρμοσμένος στα δεδομένα της python — παρουσιάζεται εδώ. Όπως θα διαπιστώσετε κατανοώντας το πρόγραμμα, πρόκειται ουσιαστικά για μια μίνι μηχανή περιπέτειας καθώς μπορείτε να αλλάξετε τα δεδομένα μέσα στο πρόγραμμα και να δημιουργήσετε μια διαφορετική περιπέτεια.

2.2.2 Ο Χάρτης

Βασικό σημείο στα text adventures είναι ο χάρτης που δείχνει όλα τα δωμάτια και τους τρόπους σύνδεσης / επικοινωνίας μεταξύ τους. Το χάρτη αυτό θα πρέπει να τον φτιάξετε πριν αρχίσετε να γράφετε την περιπέτεια σας. Φυσικά θα έχετε σκεφτεί και κάποιο συναρπαστικό σενάριο για την περιπέτεια σας ώστε να έχετε περιγραφές για το κάθε δωμάτιο. Γεγονός είναι ότι επειδή θα έχετε γράψει εσείς την περιπέτεια μάλλον θα σας είναι εύκολο και να την κερδίσετε. Αλλά δεν πειράζει, θα παίξουν και οι φίλοι σας.

Ο δικός μας χάρτης παραμένει πιστός στο αρχικό άρθρο και φαίνεται στη εικόνα 2.3.



Εικόνα 2.3: Ο Χάρτης της Νυχτερινής Περιπέτειας

Πρόκειται προφανώς για ένα μικρού μεγέθους adventure (αλλά μπορείτε με την ίδια μηχανή να φτιάξετε ένα όσο μεγάλο θέλετε) και διαδραματίζεται σε ένα σπίτι με τα παρακάτω δωμάτια/χώρους:

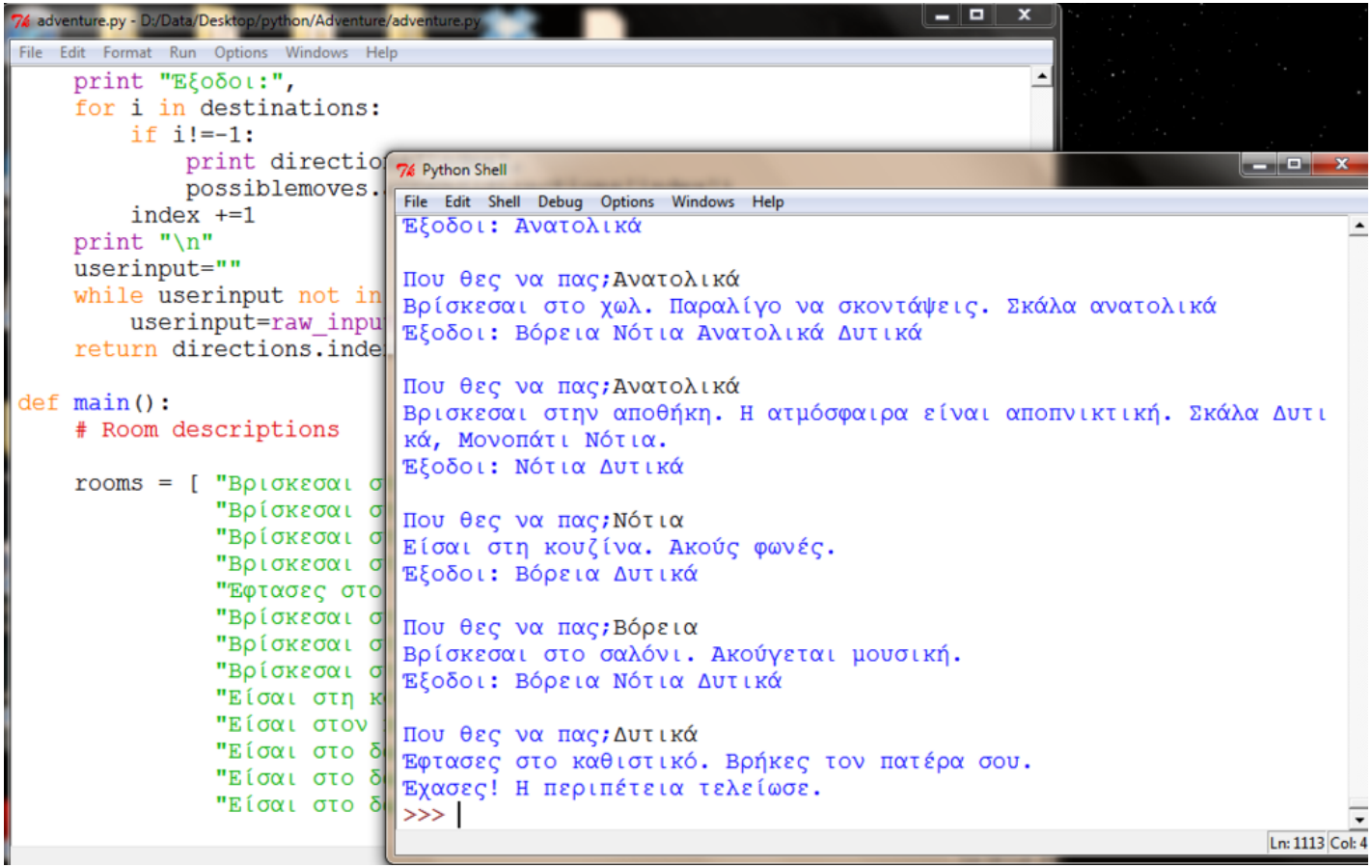
Αριθμός Δωματίου	Περιγραφή
0	Κήπος
1	Μπάνιο
2	Χωλ
3	Αποθήκη
4	Καθιστικό
5	Σαλόνι
6	Τραπεζαρία
7	Διάδρομος
8	Κουζίνα
9	Επάνω διάδρομος
10	Δωμάτιο Αδερφού
11	Δωμάτιο Γονέων
12	Δικό μας Δωμάτιο

Πίνακας 2.1: Τα δωμάτια της περιπέτειας

Δώστε έμφαση στο γεγονός ότι έχουμε κωδικοποιήσει τα δωμάτια με αριθμούς από το 0 ως το 12. Αυτό θα το χρησιμοποιήσουμε φυσικά στο πρόγραμμα και θα μας διευκολύνει πάρα πολύ.

2.2.3 Το Σενάριο

Ο αρχικός συγγραφέας ονόμασε αυτό το παιχνίδι “Νυχτερινή Περιπέτεια”. Με απλά λόγια το σενάριο είναι το παρακάτω: Είστε ένας μαθητής, μέλος μιας οικογένειας με αυστηρές ηθικές αρχές (ξέρετε, από αυτές που τα παιδιά γυρίζουν πριν τους γονείς στο σπίτι. Πριν νυχτώσει δηλαδή, όχι πριν ξημερώσει). Έχετε βγει κρυφά τη νύχτα (!) και τώρα πρέπει να επιστρέψετε στο σπίτι, στο δωμάτιο σας χωρίς να σας αντιληφθεί κανείς. Θα πρέπει να αποφύγετε τον πατέρα σας, τη μητέρα σας και τον… αδερφό σας που είναι μαρτυριάρης και θα τα πει όλα! Η μητέρα σας είναι στο δωμάτιο των γονέων, ο πατέρας σας στο καθιστικό και ο αδερφός σας στο δωμάτιο του. Θα πρέπει να βρείτε τη διαδρομή για το δωμάτιο σας χωρίς να πέσετε πάνω σε κάποιον από αυτούς. Φυσικά, υποτίθεται ότι δεν ξέρετε από πριν που είναι ο καθένας! (Μια ενδιαφέρουσα παραλλαγή θα ήταν να κάνετε το παιχνίδι να τοποθετεί αυτούς τους χαρακτήρες σε κάποια τυχαία δωμάτια ώστε κάθε φορά που θα το τρέχετε να είναι διαφορετικό. Βέβαια σε πολλές περιπτώσεις δεν θα υπάρχει λύση).



Εικόνα 2.4: Η εκτέλεση του προγράμματος στο περιβάλλον της python.

Σαν λογική είναι ιδιαίτερα απλό:

1. Ο παίκτης ξεκινάει από το δωμάτιο με αριθμό 0 (κήπος) και του εμφανίζεται η περιγραφή του χώρου και οι πιθανές έξοδοι σε μορφή κατευθύνσεων (π.χ. Νότια).

2. Ο παίκτης δίνει την κατεύθυνση, και εφόσον είναι έγκυρη, το πρόγραμμα τον πηγαίνει στο αντίστοιχο δωμάτιο.

3. Αν ο παίκτης βρεθεί σε δωμάτιο που κερδίζει ή χάνει, τυπώνεται το αντίστοιχο μήνυμα και η περιπέτεια τελειώνει, διαφορετικά…

4. …τυπώνεται η νέα περιγραφή και έξοδοι και επανερχόμαστε στο βήμα 2.

Καθώς καταλαβαίνετε το κύριο μέρος του προγράμματος είναι ένας βρόχος του τύπου:

```
“Όσο ο παίκτης δεν (έχασε ή κέρδισε)”
```

2.2.4 Προγραμματιστική Λογική

2.2.4.1 Τα Δωμάτια

Το εύκολο κομμάτι, είναι να κωδικοποιήσουμε τα δωμάτια. Θα χρησιμοποιήσουμε εδώ μια λίστα της python, την οποία, ευφάνταστα, ονομάσαμε `rooms`:

```
rooms = [  "Βρίσκεσαι στον κήπο. Παντού σκοτάδι.",
           "Βρίσκεσαι στο μπάνιο. Ακούς θόρυβο.",
           "Βρίσκεσαι στο χωλ. Παραλίγο να σκοντάψεις. Σκάλα ανατολικά",
           "Βρίσκεσαι στην αποθήκη. Η ατμόσφαιρα είναι αποπνικτική.\
            Σκάλα Δυτικά, Μονοπάτι Νότια.",
           "Έφτασες στο καθιστικό. Βρήκες τον πατέρα σου.",
           "Βρίσκεσαι στο σαλόνι. Ακούγεται μουσική.",
           "Βρίσκεσαι στην τραπεζαρία. Τα πάντα είναι ανάστατα.",
           "Βρίσκεσαι στο διάδρομο. Είναι σκοτεινά. Σκάλα Νότια",
           "Είσαι στη κουζίνα. Ακούς φωνές.",
           "Είσαι στον πάνω διάδρομο. Παντού ησυχία. Σκάλα Βόρεια.",
           "Είσαι στο δωμάτιο του αδερφού σου. Ο αδερφός σου σε μαρτυρά!",
           "Είσαι στο δωμάτιο των γονέων σου. Η μητέρα σου σε έπιασε.",
           "Είσαι στο δωμάτιο σου. Είσαι ασφαλής." ]
```

Παρατηρήστε ότι έχουμε περάσει τις περιγραφές με την ίδια σειρά (0-12) που είδαμε πριν. Αυτό μας επιτρέπει να αποθηκεύσουμε το τρέχον δωμάτιο (αυτό στο οποίο βρίσκεται ο παίκτης την δεδομένη στιγμή) σε μια μεταβλητή που θα ονομάσουμε `room`. Έτσι, αν ο χρήστης βρίσκεται στο δωμάτιο 2 (το χωλ), οι παρακάτω εντολές δίνουν την περιγραφή:

```
room = 2
print rooms[room]

"Βρίσκεσαι στο χωλ. Παραλίγο να σκοντάψεις. Σκάλα ανατολικά"
```

Πιστεύουμε ότι είναι αρκετά απλό να καταλάβετε ότι κάθε φορά το πρόγραμμα θα δίνει τη σωστή τιμή — με κάποιο τρόπο που θα δούμε — στη μεταβλητή `room` και θα χρησιμοποιεί την παραπάνω `print` για να τυπώνει το μήνυμα. Αλλά πως θα κωδικοποιήσουμε την κίνηση από το ένα δωμάτιο στο άλλο;

Εικόνα 2.5: Επειδή ξέρω ότι θέλετε (εντάξει, πεθαίνετε) να δείτε πως θα ήταν το παιχνίδι σε "γραφική" μορφή, δείτε το μέσω pygame. Η βασική μηχανή είναι φυσικά ίδια, αλλά ίσως δεν μπορέσετε να κατανοήσετε — για την ώρα — το υπόλοιπο. Εγκαταστήστε όμως το pygame για να το τρέξετε (για windows δείτε http://www.pygame.org/download.shtml, για Linux φυσικά στο repository της διανομής σας). Κατεβάστε το παιχνίδι από εδώ: http://www.freebsdworld.gr/files/adventure-pygame.zip

2.2.4.2 Οι Κατευθύνσεις και οι Προορισμοί

Δείτε ξανά το σχήμα του χάρτη. Παρατηρήστε στο κάτω μέρος τα βελάκια των κατευθύνσεων: Έχουμε κωδικοποιήσει κάθε κατεύθυνση με ένα αριθμό: Βόρεια = 0, Νότια = 1, Ανατολικά = 2, Δυτικά = 3. Φανταστείτε τώρα ότι βρίσκεστε στο δωμάτιο 9 που έχει τέσσερις πιθανές εξόδους:

Κατεύθυνση	**Δωμάτιο Προορισμού**
0 (Βόρεια)	7 (Διάδρομος
1 (Νότια)	11 (Δωμάτιο γονέων)
2 (Ανατολικά)	12 (Δικό μας δωμάτιο)
3 (Δυτικά)	10 (Δωμάτιο αδερφού)

Πίνακας 2.2: Κατευθύνσεις και προορισμοί από το δωμάτιο 9

Αν δημιουργήσουμε μια λίστα με τους αριθμούς:

```
destinations = [ 7, 11, 12, 10 ]
```

το στοιχείο `destinations[0]` θα είναι 7, το `destinations[1]` θα είναι 11 κλπ. Παρατηρείτε κάτι; Ο δείκτης της λίστας συμβολίζει την κατεύθυνση, και το αποτέλεσμα τον προορισμό! Αν λοιπόν ο χρήστης είναι στο δωμάτιο 9, και επιλέξει ως κατεύθυνση το 1 το δωμάτιο που θα βρεθεί θα είναι:

```
room = destinations[1]
```

και μετά το μόνο που μένει είναι να τυπώσουμε το μήνυμα:

```
print rooms[room]
```

Τί γίνεται σε περίπτωση που μια κατεύθυνση δεν υπάρχει σε κάποιο δωμάτιο; Αυτή την κωδικοποιούμε με το -1. Ελέγχοντας για το -1, όταν δεχόμαστε είσοδο από το χρήστη, δεν του επιτρέπουμε να δώσει κατευθύνσεις που δεν αντιστοιχούν σε έξοδο στο δωμάτιο που βρίσκεται.

Βέβαια ο χρήστης καλό θα είναι να δίνει εντολές με λέξεις και όχι αριθμούς, αλλά αυτό είναι μια λεπτομέρεια που θα δούμε αργότερα.

Αυτό που δεν είναι λεπτομέρεια είναι το εξής: Μια λίστα όπως η παραπάνω μπορεί να κρατάει μόνο τα δεδομένα κίνησης για ένα δωμάτιο. Τι γίνεται με εμάς που έχουμε 13 δωμάτια;

```
moves = [ [-1,2,-1,-1],
          [-1,-1,2,-1],
          [0,5,3,1],
          [-1,8,-1,2],
          [-1,6,5,-1],
          [2,8,-1,4],
          [4,7,-1,-1],
          [6,9,8,-1],
          [5,-1,-1,7],
          [7,11,12,10],
          [-1,-1,9,-1],
          [9,-1,-1,-1],
          [-1,-1,-1,9] ]
```

Αχά! Έχουμε μια λίστα που περιέχει λίστες! Για να δούμε αν καταλάβατε τι κάναμε, τι θα περιέχει το `moves[9]`;

Θα περιέχει τη λίστα με τους προορισμούς για κάθε κατεύθυνση από το δωμάτιο 9, δηλ:

```
[7, 11, 12, 10 ]
```

Σε οποιαδήποτε λοιπόν περίπτωση, και για οποιαδήποτε τιμή στη μεταβλητή room, το δωμάτιο δηλ. που βρισκόμαστε τη δεδομένη στιγμή:

```
destinations = moves [ room ]
```

Και όταν ο χρήστης επιλέξει μια κατεύθυνση, από 0 ως 3 την οποία έστω ότι αποθηκεύεται στην μεταβλητή `direction`, το νέο δωμάτιο θα είναι:

```
room = destinations [ direction ]
```

και έπειτα φυσικά:

```
print rooms[room]
```

Ε καθώς καταλαβαίνετε, αυτό είναι όλο! Μένουν βέβαια κάποιες λεπτομέρειες…

2.2.4.3 Ανίχνευση Νίκης / Αποτυχίας

Υπάρχει μια λίστα `winrooms` που περιέχει τα δωμάτια στα οποία το παιχνίδι τελειώνει με νίκη του παίκτη. Στο συγκεκριμένο παιχνίδι είναι μόνο ένα, αλλά θα μπορούσε να είναι περισσότερα. Αντίστοιχα υπάρχει η λίστα `lossrooms` που περιέχει τα δωμάτια στα οποία ο παίκτης χάνει! Η python μπορεί πολύ γρήγορα να ελέγξει αν κάποια τιμή υπάρχει σε μια λίστα, όπως είδαμε στην αρχή του κεφαλαίου:

```
if room in winrooms:
  print "Κέρδισες!"
```

Τελικά είναι πολύ απλό!

Το πρόγραμμα χρησιμοποιεί συναρτήσεις, για τις οποίες θα μιλήσουμε εκτενώς στο επόμενο κεφάλαιο. Εκεί θα περιγράψουμε και τη λειτουργία της συνάρτησης `getInput` που βρίσκει ποιες είναι οι έγκυρες κατευθύνσεις ανάλογα με το δωμάτιο και δέχεται (και επαληθεύει) την είσοδο του χρήστη, όχι σε αριθμούς αλλά με κανονικές εντολές.

Το πρόγραμμα μας ξεκινάει να εκτελείται από την βασική συνάρτηση `main` λόγω της εντολής:

```
if __name__ == "__main__":
  main()
```

Αν έχετε ασχοληθεί με κάποια άλλη γλώσσα προγραμματισμού (C, anyone?) θα έχετε ίσως συναντήσει την συνάρτηση με το όνομα m ain από την οποία — τυπικά — ξεκινάει η εκτέλεση ενός προγράμματος. Στην python δεν είμαστε υποχρεωμένοι να έχουμε τέτοια συνάρτηση – και στο πρώτο μας πρόγραμμα πράγματι δεν είχαμε. Είναι ωστόσο συνηθισμένο να βάζουμε σε μια συνάρτηση `main` το κύριο κομμάτι του κώδικα μας. Όταν καλούμε το πρόγραμμα από την γραμμή εντολών (ή το εκτελούμε μέσω του `idle`), η ειδική μεταβλητή `__name__` παίρνει την τιμή `__main__` επιτρέποντας μας έτσι να καλέσουμε την κύρια ρουτίνα του προγράμματος μας με τον τρόπο που βλέπετε παραπάνω.

2.2.5 Το Κύριο Πρόγραμμα

Ο βασικός βρόχος είναι:

```
room = 0
endgame = False
while not endgame:
  print rooms[room]
  if room in winrooms:
    print "Κέρδισες! Η περιπέτεια τελείωσε."
    endgame = True
  elif room in lossrooms:
    print "Έχασες! Η περιπέτεια τελείωσε."
    endgame = True
  else:
    direction = getInput(moves, room)
    destinations = moves[room]
    room = destinations[direction]
```

Η μεταβλητή `direction` έχει την κατεύθυνση που έδωσε ο παίκτης, σε μορφή αριθμού. Το τέλος του παιχνιδιού σηματοδοτείται με την μεταβλητή `endgame` η οποία γίνεται `True` όταν φτάσουμε στο `winroom` ή σε ένα από τα `lossrooms`. Η μεταβλητή `direction` μπορεί να είναι 0,1,2,3 ανάλογα με την είσοδο του χρήστη.

Μπορείτε να βρείτε το πλήρες πρόγραμμα στο παράρτημα, σελ. 164 ή να το κατεβάσετε από εδώ: `http://www.freebsdworld.gr/files/adventure.zip`.

Κεφάλαιο 3

Συναρτήσεις – Ενσωματωμένες και Δικές μας!

3.1 Εισαγωγή

Είμαστε σίγουροι ότι το adventure που δημοσιεύσαμε στο προηγούμενο κεφάλαιο σας έβαλε σε σκέψεις. Όχι γιατί ήταν δύσκολο στο… gameplay, αλλά γιατί είχε μεγάλο ενδιαφέρον από προγραμματιστικής άποψης. Αν και όλη η βασική λογική (ο αλγόριθμος αν θέλετε) παρουσιάστηκε στο προηγούμενο κεφάλαιο, αφήσαμε κάποια τμήματα που θα εξετάσουμε εδώ. Για να τα εξηγήσουμε θα πρέπει να μιλήσουμε για ένα πολύ σημαντικό στοιχείο της python (και των περισσότερων γλωσσών): Τις συναρτήσεις!

3.2 Συναρτήσεις στην Python

Αν η λέξη συνάρτηση σας θυμίζει κάτι από μαθηματικά, δεν έχετε πολύ άδικο. Βέβαια στις περισσότερες γλώσσες η έννοια της συνάρτησης είναι θα λέγαμε πιο χαλαρή σε σχέση με τα μαθηματικά. Στα μαθηματικά για παράδειγμα, σκεφτόμαστε τη συνάρτηση σαν μια σχέση στην οποία δίνουμε κάποια δεδομένα (ένα αριθμό) και μας δίνει ένα αποτέλεσμα. Αν θέλουμε να είμαστε ακριβείς, μια συνάρτηση στα μαθηματικά δεν μπορεί να δίνει διαφορετικές τιμές σαν αποτέλεσμα αν δίνουμε πάντοτε την ίδια τιμή στην είσοδο.

Σε πολλές γλώσσες προγραμματισμού, μπορούμε να ορίσουμε συναρτήσεις που να συμπεριφέρονται με ακριβώς τον ίδιο τρόπο όπως στα μαθηματικά. Για την ακρίβεια, η python — όπως και οι περισσότερες γλώσσες — διαθέτει ήδη μια μεγάλη γκάμα από συναρτήσεις που έχουν

έρθει κατευθείαν από τα μαθηματικά. Πάρτε για παράδειγμα την τετραγωνική ρίζα:

```
import math
x = math.sqrt(2)
print x
```

Ο παραπάνω κώδικας θα μας τυπώσει την τετραγωνική ρίζα του 2. Ήδη από το πρώτο μας παιχνίδι, έχετε δει πως καλούμε μια συνάρτηση και πως αποθηκεύουμε το αποτέλεσμα. Θυμηθείτε τη γραμμή:

```
number = random.randint(1,50)
```

Προφανώς λοιπόν μια συνάρτηση καλείται με τα ορίσματα μέσα σε παρένθεση και το αποτέλεσμα το λαμβάνουμε στη μεταβλητή που έχουμε πριν το ίσον. Για να χρησιμοποιήσουμε συγκεκριμένες συναρτήσεις, πρέπει πρώτα να συμπεριλάβουμε τη βιβλιοθήκη που τις περιέχει όπως φαίνεται π.χ. από το `import math` παραπάνω.

Για να πάρετε μια ιδέα τώρα το πως μια συνάρτηση στην python μπορεί να διαφοροποιείται από μια καθαρά μαθηματική συνάρτηση, θυμηθείτε την `range` που είχαμε δει στην αρχή του προηγούμενου κεφαλαίου:

```
>>> x = range(1,10)
>>> print x

[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Βλέπετε εδώ ότι η python δεν επιστρέφει ένα απλό αριθμό — κάτι που αναμένουμε συνήθως από μια μαθηματική συνάρτηση — αλλά μια ολόκληρη λίστα.

Βέβαια ακόμα πιο ενδιαφέρον είναι να αρχίσουμε να δημιουργούμε τις δικές μας συναρτήσεις!

3.3 Δημιουργία Συνάρτησης

Για ποιο λόγο να θέλουμε να κάνουμε κάτι τέτοιο; Για να δείτε πως δημιουργείται η ανάγκη για τη δημιουργία μιας συνάρτησης, ας πάρουμε την τετραγωνική ρίζα που είδαμε παραπάνω. Φανταστείτε λοιπόν ότι η γλώσσα που χρησιμοποιούσατε δεν είχε κανένα τρόπο να υπολογίζει τετραγωνικές ρίζες. Εσείς όμως, ως καταπληκτικός και wow προγραμματιστής, φτιάξατε ένα κομμάτι κώδικα που το κάνει. Και τώρα;

Θα μπορούσατε βέβαια κάθε φορά που στο κύριο πρόγραμμα σας χρειαζόσασταν μια τετραγωνική ρίζα, να κάνατε αντιγραφή – επικόλληση εκείνο το ίδιο κομμάτι κώδικα (δεν είναι τυχαίο που από τότε που εφευρέθηκε η αντιγραφή – επικόλληση χάθηκαν οι προγραμματιστές). Σύντομα όλη η ουσία του κανονικού προγράμματος σας θα χάνονταν μέσα στις… επικολλήσεις της τετραγωνικής ρίζας. Θα ήταν προφανώς προτιμότερο να δίνατε με κάποιο τρόπο ένα όνομα σε αυτό το κομμάτι κώδικα και να το καλείτε κάθε φορά που θέλετε να κάνετε τον υπολογισμό. Με λίγα λόγια:

Χρειάζεστε μια συνάρτηση!

Είναι εύκολο να καταλάβετε από τα παραπάνω ότι:

- Μια συνάρτηση τη χρησιμοποιούμε όταν χρειάζεται να εκτελέσουμε το ίδιο κομμάτι κώδικα πολλές φορές (και με άλλες ενδεχομένως τιμές)
- Μια καλογραμμένη συνάρτηση κάνει μια συγκεκριμένη εργασία και μπορεί να μας φανεί χρήσιμη σε πολλά προγράμματα
- Η σωστή χρήση συναρτήσεων κάνει το πρόγραμμα μας *δομημένο* και μπορούμε πολύ εύκολα να ακολουθήσουμε τη ροή και τη λογική του.

Και κάτι ακόμα που μάλλον υποπτεύεστε:

- Οι συναρτήσεις στην python **δεν περιορίζονται** στον αυστηρό ορισμό της μαθηματικής συνάρτησης.

Τι σημαίνει αυτό; Όπως είδατε, μπορεί να επιστρέφουν αντί για ένα αριθμό, μια λίστα. Ή ίσως μια αλφαριθμητική τιμή (ένα όνομα, κομμάτι κειμένου ή *string* όπως το λέμε συνήθως). Αλλά μπορεί να μην επιστρέφουν και… τίποτα! Ναι, καλά διαβάσατε: τίποτα! Και όχι, δεν έχουμε χάσει το μυαλό μας ακόμα (όχι εντελώς δηλαδή). Θα μου πείτε, τι μπορεί να χρησιμεύσει μια συνάρτηση που δεν επιστρέφει τίποτα;

Μα είπαμε ότι δεν πρόκειται για αυστηρό ορισμό μαθηματικής συνάρτησης. Φανταστείτε για παράδειγμα μια συνάρτηση που τυπώνει κάτι στην οθόνη. Για να μην είναι τελείως απλοϊκή, φανταστείτε ότι την έχουμε φτιάξει ώστε να της δίνουμε ως όρισμα το κείμενο που θέλουμε να τυπώσει και η συνάρτηση να το κεντράρει στα αριστερά – δεξιά περιθώρια της οθόνης. Τι επιστρέφει αυτή η συνάρτηση; Τίποτα. Κάνει μια δουλειά (τυπώνει το μήνυμα) αλλά δεν επιστρέφει κάποια τιμή στο κύριο πρόγραμμα που να προκύπτει από κάποιο υπολογισμό και να έχει κάποια χρησιμότητα.

Σε κάποιες γλώσσες προγραμματισμού, υπάρχει ή έννοια της *διαδικασίας ή procedure* για την περίπτωση που δεν υπάρχει επιστρεφόμενη τιμή. Στην python ωστόσο, χρησιμοποιούμε συναρτήσεις και για τις δύο περιπτώσεις.

Παρακάτω είναι ο κώδικας και η αντίστοιχη δοκιμή:

```
def centerText(thetext):
  linelength = 80
  textlength = len(thetext)
  spaces = (linelength - textlength)/2
  print spaces*" "+thetext
  return

centerText("Hello there!")
```

Στο παραπάνω θεωρούμε ότι το μέγεθος γραμμής της οθόνης είναι 80 χαρακτήρες (ένα τυπικό τερματικό). Η συνάρτηση τυπώνει το μήνυμα μας με τον απαραίτητο αριθμό κενών από μπροστά ώστε να κεντράρεται. Όταν την καλούμε, το κείμενο "Hello there!" αντιγράφεται ουσιαστικά μέσα στη μεταβλητή `thetext` της συνάρτησης. Θα μπορούσαμε επίσης να την καλέσουμε και έτσι:

```
mytext = "I really love Python programming"
centerText(mytext)
```

Τι γίνεται σε αυτή την περίπτωση; Όπως φαντάζεστε, η τιμή που περιέχει η μεταβλητή `mytext` αντιγράφεται στην μεταβλητή `thetext` της συνάρτησης την ώρα που την καλούμε. Σημειώστε εδώ ότι η μεταβλητή `thetext` της συνάρτησης είναι *τοπική* για τη συνάρτηση και δεν είναι γνωστή έξω από αυτή. Αν έχετε μια μεταβλητή `thetext` στη συνάρτηση σας και μια άλλη `thetext` έξω από αυτή, αυτές οι δύο δεν σχετίζονται και δεν μπερδεύονται μεταξύ τους! Για να πάμε και ένα βήμα παραπέρα, όλες οι μεταβλητές που ορίζουμε μέσα στη συνάρτηση είναι *τοπικές* για αυτή και υπάρχουν και έχουν νόημα μόνο όσο η συνάρτηση εκτελείται.

Μια ερώτηση που ίσως έχετε είναι τι γίνεται σε περίπτωση που κάνετε κάτι μέσα στη συνάρτηση και πειράξετε το ίδιο το όρισμα. Π.χ. τι θα γίνει αν αλλάξετε τη μεταβλητή `thetext`. Πιστεύετε θα γίνει και η μεταφορά αντίστροφα (από τη συνάρτηση στο κύριο πρόγραμμα, στη `mytext`);

Η απάντηση είναι **όχι**. Όπως είπαμε, κατά την κλήση της συνάρτησης, αντιγράφεται η τιμή της μεταβλητής `mytext` στην μεταβλητή `thetext`.

Αυτή είναι και η μόνη αλληλεπίδραση που υπάρχει μεταξύ τους και είναι μόνο προς τη μια κατεύθυνση. Αυτό το είδος περάσματος τιμών μεταξύ του κύριου προγράμματος και μιας συνάρτησης ονομάζεται *κλήση με τιμή ή call by value* αν προτιμάτε. Σε άλλες γλώσσες είναι πιθανή και η *κλήση κατά αναφορά ή call by reference* (που φαντάζεστε τι κάνει) αλλά στην python έχουμε άλλα… κόλπα στο τσεπάκι μας για αντίστοιχα τρικ.

Και πως θα γράφαμε μια συνάρτηση που πράγματι να επιστρέφει μια τιμή στο κύριο πρόγραμμα:

```
def cube(thenumber):
  result = thenumber ** 3
  return result

x = 3
y = cube(x)
print y
print cube(3)
print cube(y)
```

Πιστεύουμε είναι προφανές το πως λειτουργεί! Στο κύριο (θεός να το κάνει) πρόγραμμα σας δείχνουμε και διάφορους τρόπους κλήσης της συνάρτησης.

Προφανώς θα μπορούσε και η συνάρτηση μας να γραφεί πολύ πιο απλά, επί της ουσίας σε μια γραμμή:

```
def cube(thenumber):
  return thenumber**3
```

Και με αυτό το… crash course στις συναρτήσεις, είμαστε πιστεύουμε έτοιμοι να δούμε πως δουλεύει η συνάρτηση `getInput` που έχουμε στο καταπληκτικό μας Adventure!

3.4 Η Συνάρτηση getInput στο Adventure

Ας τη δούμε συνολικά και να την ερμηνεύσουμε:

```
1  def getInput(moves,room):
2    directions = [”Βόρεια”,”Νότια”,”Ανατολικά”,”Δυτικά”]
3    destinations = moves[room]
4    possiblemoves = []
5    index = 0
6    print ”Έξοδοι:”,
7    for i in destinations:
8      if i!=-1:
9        print directions[index],
10       possiblemoves.append(directions[index])
11     index +=1
12   print ”\n”
13   userinput=””
14   while userinput not in possiblemoves:
15     userinput=raw_input(”Που θες να πας; ”)
16   return directions.index(userinput)
```

Η συνάρτηση έχει δύο ορίσματα, τα `moves` και `room`. Θυμηθείτε ότι το `moves` περιέχει τις κινήσεις από το ένα δωμάτιο στο άλλο και το `room` είναι ο αριθμός του τρέχοντος δωματίου. Μη ξεχνάτε ότι τα ονόματα των μεταβλητών αυτών θα μπορούσαν να είναι διαφορετικά μέσα στη συνάρτηση και δεν υπάρχει μπέρδεμα με το κύριο πρόγραμμα!

Μέσα στην συνάρτηση ορίζεται μια λίστα `directions` που περιέχει τα ονόματα των κατευθύνσεων. Προσέξτε ότι την έχουμε φτιάξει με τέτοιο τρόπο ώστε ο δείκτης του κάθε στοιχείου να αντιστοιχεί στην κωδικοποίηση που έχουμε κάνει, δηλ. Βόρεια = 0, Νότια =1 κλπ.

Η πρώτη δουλειά της συνάρτησης είναι να βρει την λίστα των προορισμών από το συγκεκριμένο δωμάτιο:

```
  destinations = moves [room]
```

Αμέσως παρακάτω, ορίζουμε μια κενή λίστα `possiblemoves`, τυπώνουμε ένα μήνυμα στο χρήστη και εισερχόμαστε στο κύριο βρόχο `for`:

```
  index = 0
  print "Έξοδοι:",
  for i in destinations:
    if i!=-1:
      print directions[index],
      possiblemoves.append(directions[index])
    index +=1
```

Ο βρόχος διατρέχει ένα – ένα τα στοιχεία της λίστας `destinations`. Αν για παράδειγμα, βρισκόμασταν στο δωμάτιο 7, το `i` θα διατρέξει τους αριθμούς:

```
[6, 9, 8, -1]
```

Ας πάρουμε λοιπόν την πρώτη φορά. Το `i` θα έχει την τιμή 6. Έχουμε τη σύγκριση:

```
    if i!=-1:
      print directions[index],
      possiblemoves.append(directions[index])
```

Προφανώς το `i` δεν είναι -1, οπότε εκτελείται η εντολή:

```
      print directions[index]
```

και καθώς το `index` έχει την τιμή 0 (η αρχική του), το παραπάνω θα τυπώσει:

`Βόρεια`

(Το κόμμα στην άκρη της `print` εμποδίζει την αλλαγή γραμμής.) Ταυτόχρονα, στη λίστα `possiblemoves` προστίθεται η τιμή που τυπώθηκε παραπάνω:

```
      possiblemoves.append(directions[index])
```

Η `append` είναι μια μέθοδος που εφαρμόζεται σε λίστες για να προσθέσουμε ένα στοιχείο στο τέλος. Μετά την πρώτη εκτέλεση του βρόχου, η λίστα `possiblemoves` θα περιέχει την πρώτη της τιμή:

```
[ "Βόρεια" ]
```

Για τις μεθόδους θα μιλήσουμε αναλυτικότερα σε επόμενη ενότητα, αλλά πιστεύουμε κατανοείτε τι γίνεται. Σκεφτείτε τώρα ότι το ίδιο ακριβώς γίνεται και για τις άλλες τιμές που θα πάρει το `i`, ώστε στο τέλος η λίστα `possiblemoves` να περιέχει:

```
[ "Βόρεια", "Νότια", "Ανατολικά" ]
```

Και ναι δεν θα περιέχει τη λέξη "Δυτικά" καθώς η τελευταία τιμή που παίρνει το `i` είναι -1. Προσέξτε ότι σε κάθε κύκλο της `for` αυξάνουμε την τιμή του `index` κρατώντας σε συγχρονισμό το δείκτη με το αντίστοιχο στοιχείο της λίστας `destinations`.

Το υπόλοιπο κομμάτι είναι απλό. Με το τέλος του βρόχου, αλλάζουμε γραμμή τυπώνοντας τον ειδικό χαρακτήρα αλλαγής γραμμής:

```
  print "\n"
```

Τέλος, ορίζουμε μια μεταβλητή `userinput` στην οποία θα αποθηκεύσουμε την είσοδο του χρήστη. Αρχικά είναι κενή, και εισερχόμαστε στο βρόχο `while`:

```
  userinput=""
  while userinput not in possiblemoves:
    userinput=raw_input("Που θες να πας; ")
```

Όπως καταλαβαίνετε ο βρόχος ρωτάει και ξαναρωτάει το χρήστη για την κίνηση του μέχρι να λάβει μια έγκυρη απάντηση, μια απάντηση δηλ. που να βρίσκεται στη λίστα `possiblemoves` που δημιουργήσαμε προηγουμένως. Με τη λήψη έγκυρης απάντησης, η συνάρτηση επιστρέφει την παρακάτω… κρυπτική τιμή:

```
  return directions.index(userinput)
```

Και δεν είναι στην πραγματικότητα καθόλου κρυπτική: Αν ο χρήστης έδωσε "Νότια", η μέθοδος `index` επιστρέφει το δείκτη, τη θέση δηλαδή, του στοιχείου της λίστας `directions` στην οποία βρίσκεται η τιμή "Νότια". Δηλ. το 1.

Για να το καταλάβετε, το παρακάτω:

```
print directions.index("Ανατολικά")
```

Θα τύπωνε τον αριθμό 2. Ο αριθμός αυτός επιστρέφεται στο κύριο πρόγραμμα ως αποτέλεσμα της συνάρτησης. Και όπως θα παρατηρήσατε, έχουμε φτιάξει μια συνάρτηση που τυπώνει μηνύματα, δέχεται και επικυρώνει την είσοδο από το χρήστη και τελικά επιστρέφει και την τιμή έτοιμη για χρήση στο κύριο πρόγραμμα. Όπως καταλαβαίνετε μια συνάρτηση στην python σίγουρα δεν περιορίζεται στην κλασική μαθηματική έννοια της συνάρτησης!

Προφανώς ο παραπάνω τρόπος δεν είναι επίσης ο μοναδικός που θα μπορούσε να χρησιμοποιηθεί για να γραφεί η `getInput`. Θα μπορούσαμε επίσης να χρησιμοποιήσουμε τα *λεξικά ή dictionaries* της python. Επίσης θα μπορούσαμε να χωρίσουμε τη λειτουργία της `getInput` σε περισσότερες συναρτήσεις ή ακόμα και να μεταφέρουμε κάποια κομμάτια της στο κύριο πρόγραμμα. Θα μπορούσαμε π.χ. να φτιάξουμε μια `getMoves` η οποία να επιστρέφει τη λίστα `possiblemoves` και η διαχείριση της εισόδου να γίνεται στο κύριο πρόγραμμα. Και τώρα ξέρετε™ την άσκηση σας μέχρι το επόμενο κεφάλαιο! Καλό θα ήταν επίσης να προσθέσετε την `centerText` που φτιάξαμε ώστε τα μηνύματα του προγράμματος να κεντράρονται.

Πριν προχωρήσετε στο επόμενο κεφάλαιο, διαβάστε και πειραματιστείτε. Μη ξεχνάτε, ο προγραμματισμός είναι μια συνεχής ασχολία. Καλό ξεκόλλημα!

Κεφάλαιο 4

Ξεκινώντας στο Pygame

4.1 Εισαγωγή

Μέχρι τώρα σας έχουμε δείξει απλά παιχνίδια που θα μπορούσατε κάλλιστα να παίζατε σε ένα τερματικό του PDP11! Πρέπει να ομολογήσουμε ότι στις μέρες μας ένα adventure κειμένου δεν θα είχε και μεγάλη τύχη στην αγορά. Καταλαβαίνετε βέβαια ότι αυτή η εισαγωγή ήταν απαραίτητη για να μάθετε τα βασικά της python και φυσικά για να ενεργοποιήσουμε την αλγοριθμική σας σκέψη! Πιστεύουμε ότι αυτό το καταφέραμε με το text adventure.

Είναι προφανές (τουλάχιστον σε μας) ότι δεν θα μπορούσαμε να ξεκινήσουμε κατευθείαν με κάποια γραφική εφαρμογή – πολύ απλά θα χρειάζονταν να μάθετε πάρα πολλά σε λίγο χρόνο. Βλέπετε οι γραφικές εφαρμογές (στις οποίες φυσικά ανήκουν σχεδόν όλα τα παιχνίδια) έχουν πολύ διαφορετική φιλοσοφία από τις εφαρμογές κονσόλας. Σε αυτό το κεφάλαιο θα εξερευνήσουμε την δύναμη που παρέχει το pygame, η καταπληκτική βιβλιοθήκη ή module της python που μας επιτρέπει να φτιάξουμε γραφικά παιχνίδια με σχετική ευκολία. Αλλά πρώτα από όλα πρέπει να εγκαταστήσουμε το pygame και να δούμε μέσα από κάποια απλά προγράμματα τι σημαίνει να φτιάχνει κάποιος εφαρμογές που τρέχουν σε… παράθυρα (όχι απαραίτητα Windows, αλλά σε οποιοδήποτε γραφικό λειτουργικό).

4.2 Εγκατάσταση του Pygame

Είναι πολύ εύκολο να εγκαταστήσετε το pygame, αρκεί να επισκεφτείτε τη δικτυακή του τοποθεσία `http://www.pygame.org`. Αν χρησιμοποιείτε Windows, μπορείτε να κατεβάσετε την τελευταία σταθερή έκδοση για την python 2.7 που εγκαταστήσατε προηγουμένως. Τη στιγμή που γράφεται το βιβλίο, η καλύτερη επιλογή είναι:

`http://pygame.org/ftp/pygame-1.9.2a0.win32-py2.7.msi`

και αν χρησιμοποιείτε 64bit windows και την 64bit python, κατεβάστε τη δοκιμαστική έκδοση από εδώ:

`http://www.lfd.uci.edu/~gohlke/pythonlibs/#pygame`

Αν χρησιμοποιείτε κάποια debianοειδή διανομή, θα μπορέσετε να εγκαταστήσετε το pygame με μια εντολή του τύπου:

```
# apt-get install python-pygame
```

Για το FreeBSD, η εγκατάσταση γίνεται εύκολα από τα ports (με μόνο πρόβλημα την πιθανή κλιματική αλλαγή):

```
# cd /usr/ports/devel/py-game
# make install clean
```

Τέλος, για σας τους κρυφούς χρήστες του OSX (το ξέρουμε ότι είστε ανάμεσα μας!), το Lion διαθέτει ήδη την python 2.7 από τη… μαμά του και χρειάζεστε μόνο το pygame που μπορείτε να βρείτε εδώ:

`http://www.pygame.org/ftp/pygame-1.9.2pre-py2.7-macosx10.7.mpkg.zip`

Γενικά δείτε τη σελίδα `http://www.pygame.org/download.shtml` για πιθανά downloads για οποιοδήποτε λειτουργικό και έκδοση.

Και τώρα που έχετε εγκαταστήσει το module, είναι ώρα να δούμε ένα ωραίο Hello World σε pygame.

4.3 Hello Pygame!

Δεν θέλουμε να σας κρατήσουμε σε αγωνία, χρησιμοποιήστε τον συντάκτη κειμένου της προτίμησης σας (ή και το `idle` αν θέλετε) και πληκτρολογήστε το παρακάτω πρόγραμμα:

```
#
# pygame hello world
#
# Όχι, δεν είναι τόσο πολύπλοκο όσο φαίνεται.
# Μην πέσετε από το μπαλκόνι!
#
import pygame
from pygame.locals import *
from sys import exit

# Define app window width and height

windowsize = (320,140)

def centerMessage(surface):
  return (windowsize[0] - surface.get_width())/2

#
# Process the event queue
# returns true if user clicks close
#

def getQuit():
  for event in pygame.event.get():
    if event.type == QUIT:
      return True
```

```
27 def main():
28   # Initialize the pygame library
29
30   pygame.init()
31   surfacecolor = (50,80,250)
32   screen = pygame.display.set_mode(windowsize, DOUBLEBUF, 32)
33   pygame.display.set_caption("Hello Pygame!")
34   textfont = pygame.font.SysFont("Arial",48)
35   thetext = textfont.render("Hello World!", True, (255,0,0),(255,255,0))
36
37   # Initialize text position
38
39   textx = centerMessage(thetext)
40   texty = 40
41
42   # Begin main loop
43
44   endprogram = False
45
46   while not endprogram:
47
48     # fill screen with bluish tint
49
50     screen.fill(surfacecolor)
51
52     # Show text message
```

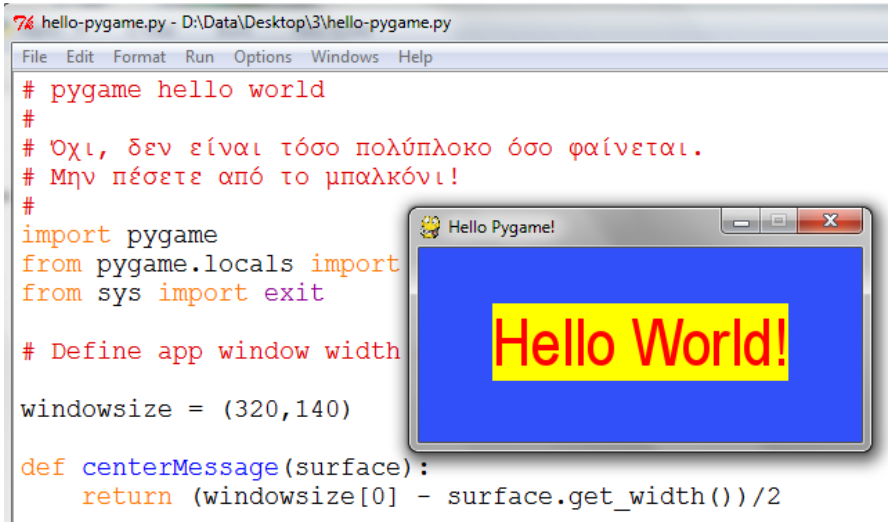
```
    screen.blit(thetext,(textx,texty))
    endprogram = getQuit()
    pygame.display.update()

  # shutdown pygame and exit program

  pygame.quit()
  exit()

# Start program
if __name__ == "__main__":
  main()
```

Όσοι ξεπεράσατε το σοκ του μεγέθους του παραπάνω προγράμματος, μπορείτε να συνεχίσετε παρακάτω – θα σας αποδείξουμε ότι δεν είναι τόσο δύσκολο όσο φαίνεται και ότι ναι, τελικά ήταν καλή ιδέα που αρχίσατε να μαθαίνετε προγραμματισμό, python και pygame. Για όσους είναι ακόμα σε κατάσταση σοκ, μη φοβάστε είναι μια απλή υπογλυκαιμία: φάτε ένα σοκολατάκι και συνεχίζουμε.

Ας το πάμε λοιπόν γραμμή – γραμμή και ταυτόχρονα θα βλέπουμε τις νέες έννοιες όπως ο *αντικειμενοστραφής προγραμματισμός*, η *επεξεργασία συμβάντων (events)* και ο προγραμματισμός που οδηγείται από συμβάντα (*event driven programming*).

```
import pygame
from pygame.locals import *
from sys import exit
```

Με την πρώτη εντολή φυσικά δηλώνουμε τη χρήση της βιβλιοθήκης pygame. Καθώς φαντάζεστε ότι χρησιμοποιούμε από pygame θα ξεχωρίζει καθώς θα είναι της μορφής `pygame.<εντολή>`. Η δεύτερη εντολή χρήζει περαιτέρω ερμηνείας: το `pygame.locals` περιέχει μια σειρά από συμβολικές σταθερές πολύ χρήσιμες όταν χρησιμοποιούμε το pygame. Για παράδειγμα δίνει ονόματα στους κωδικούς των πλήκτρων – τους αριθμούς που επιστρέφει ο ελεγκτής πληκτρολογίου στο λειτουργικό καθώς χτυπάμε τα πλήκτρα στην απέλπιδη προσπάθεια μας να γλιτώσουμε το διαστημόπλοιο μας από τα εχθρικά πυρά! Η `from` μας επιτρέπει να κρατήσουμε ότι θέλουμε από ένα module – βάζοντας * λέμε ότι θέλουμε ουσιαστικά να κάνουμε import τα πάντα ενώ ταυτόχρονα δεν χρειάζεται πλέον να βάζουμε από μπροστά το όνομα του module



Εικόνα 4.1: Το hello-world σε pygame. Ναι το ξέρουμε ότι δείχνει μεγάλο, αλλά κάνει πολύ περισσότερα πράγματα από ότι φαίνεται με την πρώτη ματιά.

όταν χρησιμοποιούμε κάτι από αυτό. Αν τώρα αναρωτιέστε γιατί δεν το κάνουμε αυτό πάντα, σας το αφήνω σαν άσκηση.

Το `exit` από το module `sys` το χρειαζόμαστε για να μπορούμε να τερματίσουμε όμορφα ένα πρόγραμμα. Σε μερικές πλατφόρμες (ονόματα δεν λέμε, windows δεν δείχνουμε) θα έχετε περίεργα μηνύματα λάθους και κολλήματα αν δεν καλέσετε την `exit()` στο τέλος του προγράμματος σας.

```
windowsize = (320,140)
```

Το `windowsize` είναι ένα tuple (θυμηθείτε, τα tuples μοιάζουν με τις λίστες αλλά έχουν κάποιους περιορισμούς) και περιέχει το μέγεθος (πλάτος, ύψος) του παραθύρου μας. Πολλές από τις συναρτήσεις του pygame δέχονται δεδομένα σε μορφή tuple.

Αφήνουμε για την ώρα την ερμηνεία της συνάρτησης `centerMessage` (που πρέπει να φαντάζεστε τι κάνει αν έχετε μελετήσει το προηγούμενο κεφάλαιο) και `getQuit` που θα δούμε μόλις κατανοήσουμε το κύριο μέρος του προγράμματος:

```
def main():
  pygame.init()
```

Η `init` είναι η πρώτη συνάρτηση του pygame που πρέπει να καλέσετε στην αρχή του προγράμματος σας για να αρχικοποιηθεί η λειτουργία του pygame.

```
surfacecolor = (50,80,250)
```

Σε αυτό το tuple ορίζουμε το χρώμα της επιφάνειας μας, δίνοντας τιμές για τα χρώματα κόκκινο, πράσινο και μπλε αντίστοιχα. Ο συνδυασμός δίνει ένα ενδιαφέρον γαλαζοπράσινο χρωματάκι που θα είναι το φόντο μας.

```
screen = pygame.display.set_mode(windowsize, DOUBLEBUF)
```

Το παραπάνω δημιουργεί την βασική μας επιφάνεια `screen`: το παράθυρο που μέσα σε αυτό θα εμφανιστούν τα πάντα. Παρατηρήστε ότι και το μέγεθος το δίνουμε σε μορφή tuple. Το `DOUBLEBUF` (μια συμβολική σταθερά που πήραμε από το `pygame.locals`) αγνοήστε το για την ώρα, θα το συζητήσουμε λεπτομερειακά παρακάτω.

```
pygame.display.set_caption("Hello Pygame!")
```

Ο τίτλος του παραθύρου. Ξέρετε, αυτό το ενοχλητικό κείμενο που εμφανίζεται στην μπάρα του παραθύρου και μας εκνευρίζει την ώρα που προσπαθούμε να μετακινήσουμε το παράθυρο με το ποντίκι μας.

```
textfont = pygame.font.SysFont("Arial",48)
```

Εδώ δημιουργούμε ένα αντικείμενο font. Για να το απλουστεύσουμε λίγο, ας πούμε ότι επιλέγουμε τη γραμματοσειρά με την οποία θα εμφανιστεί το κείμενο.

```
thetext = textfont.render("Hello Pygame!", True, (255,0,0),(255,255,0))
```

Εδώ δημιουργούμε την επιφάνεια `thetext` που περιέχει τη φράση Hello Pygame! Τα δύο tuples που βλέπετε αντιστοιχούν στο χρώμα προσκηνίου (foreground) και παρασκηνίου (background) αντίστοιχα. Πολύ απλά, κόκκινα γράμματα πάνω σε κίτρινο φόντο. Το `True` ενεργοποιεί

το *antialiasing* και θα μπορούσατε κάλλιστα να δοκιμάσετε και με `False` για να δείτε τι σας πάει καλύτερα στο μάτι.

Έχουμε όμως ήδη αναφέρει τις λέξεις "επιφάνεια" και "αντικείμενο" και δεν τις έχουμε ερμηνεύσει. Στο pygame ότι θέλουμε να εμφανίσουμε το δημιουργούμε πάνω σε μια *επιφάνεια*. Μπορούμε να βάλουμε μια επιφάνεια πάνω σε μια άλλη. Η πρώτη μας επιφάνεια είναι το παράθυρο μας, στη μεταβλητή `screen`. Μια δεύτερη επιφάνεια (που θα φροντίσουμε να μπει πάνω στην πρώτη) είναι αυτή που περιέχει το μήνυμα μας και αποθηκεύσαμε στη μεταβλητή `thetext`.

Σας βλέπω να ρωτάτε τι είδους μεταβλητές είναι αυτές οι `screen`, `thetext`, `thefont`. Μέχρι στιγμής έχετε δει μεταβλητές που περιέχουν strings, λίστες, αριθμούς. Τι ακριβώς είναι μια μεταβλητή που κρατάει μια επιφάνεια ή ένα font;

Καλό ερώτημα. Επιτρέψτε μου λοιπόν να σας εισάγω στον κόσμο των αντικειμένων και του αντικειμενοστραφούς προγραμματισμού! Ναι, πρόκειται για *αντικείμενα ή objects*. Σε σχέση με τον κλασικό προγραμματισμό ο αντικειμενοστραφής χρησιμοποιεί την έννοια του *αντικειμένου* για να αναφέρεται σε δομές που μεταξύ άλλων διαθέτουν *ιδιότητες ή χαρακτηριστικά (attributes ή properties)* καθώς και δικές τους συναρτήσεις για να χειρίζονται τα δεδομένα τους. Ξαναδείτε για παράδειγμα τη γραμμή:

```
thetext = textfont.render("Hello World!", True, (255,0,0),(255,255,0))
```

Ένα αντικείμενο της κλάσης font διαθέτει την — όπως την αποκαλούμε — μέθοδο `render` την οποία καλούμε. Η `render` δεν είναι μια συνάρτηση γενικής χρήσης που απλά χρησιμοποιεί το `textfont` ως μια ακόμα παράμετρο. Αν ήταν, θα την καλούσαμε κάπως έτσι:

```
thetext = render(textfont, "Hello World!", True, (255,0,0),(255,255,0))
```

Η `render` είναι μια συνάρτηση που ορίζεται για την κλάση των αντικειμένων font και καλείται όπως βλέπετε με τον τρόπο:

`αντικείμενο.ΌνομαΜεθόδου(παράμετροι_αν_υπάρχουν)`

Μάλιστα η συγκεκριμένη μέθοδος επιστρέφει ένα αντικείμενο που ανήκει στην κλάση επιφάνειας (surface). Κάθε κλάση μπορεί να περιέχει τις δικές της μεθόδους – θα μάθετε λεπτομέρειες για τα αντικείμενα και τον αντικειμενοστραφή προγραμματισμό στο επόμενο κεφάλαιο.

Όσοι δεν φάγατε το σοκολατάκι που αναφέραμε προηγουμένως μάλλον είναι ώρα να το φάτε τώρα. Όσοι το φάγατε, κρύψτε τα για να μη φάτε όλο το κουτί. Θα πάθετε ζάχαρο!

```
  textx = centerMessage(thetext)
  texty = 40
```

Για να εμφανιστεί το κείμενο, πρέπει να αποφασίσουμε τις συντεταγμένες του. Καθώς καταλαβαίνετε σε ένα παράθυρο γραφικών οι γραμμές και οι στήλες αναφέρονται σε pixels. Σε αντίθεση βέβαια με την κονσόλα κειμένου όπου η ελάχιστη θέση είναι ο ένας χαρακτήρας. Η γραμμή λοιπόν εδώ είναι το `texty` ενώ το `textx` υπολογίζεται από τη centerMessage ώστε το μήνυμα να κεντραριστεί. Το (0,0) είναι στην πάνω αριστερή γωνία της οθόνης και οι τιμές αυξάνονται προς τα κάτω και δεξιά – θα το δούμε αυτό καλύτερα όμως σε επόμενο παράδειγμα.

```
  endprogram = False

  while not endprogram:
    screen.fill(surfacecolor)
    screen.blit(thetext,(textx,texty))
    endprogram = getQuit()
    pygame.display.update()

  pygame.quit()
  exit()
```

Και επιτέλους φτάσαμε στο κύριο βρόχο του προγράμματος. Ας δούμε πρώτα λίγο τις εσωτερικές εντολές και μετά θα καταλάβετε γιατί μιλάμε για βρόχο, σε ένα πρόγραμμα που το μόνο που κάνει είναι να τυπώνει ένα μήνυμα!

```
    screen.fill(surfacecolor)
```

Αν έχετε πιάσει το νόημα, φαντάζεστε ότι ένα αντικείμενο τύπου επιφάνειας διαθέτει μια μέθοδο `fill` με την οποία μπορούμε να το γεμίσουμε με χρώμα. Το χρώμα `surfacecolor` θα γεμίσει την επιφάνεια `screen` που αντιπροσωπεύει το παράθυρο μας. Για να βάλουμε μια επιφάνεια πάνω σε μια άλλη, χρησιμοποιούμε τη μέθοδο `blit`. Καλώντας:

```
    screen.blit(thetext,(textx,texty))
```

βάζουμε την επιφάνεια `thetext` πάνω στην `screen` στις συντεταγμένες `textx, texty` (παρατηρήστε ότι και αυτές δίνονται σε μορφή tuple, για αυτό και η παρένθεση).

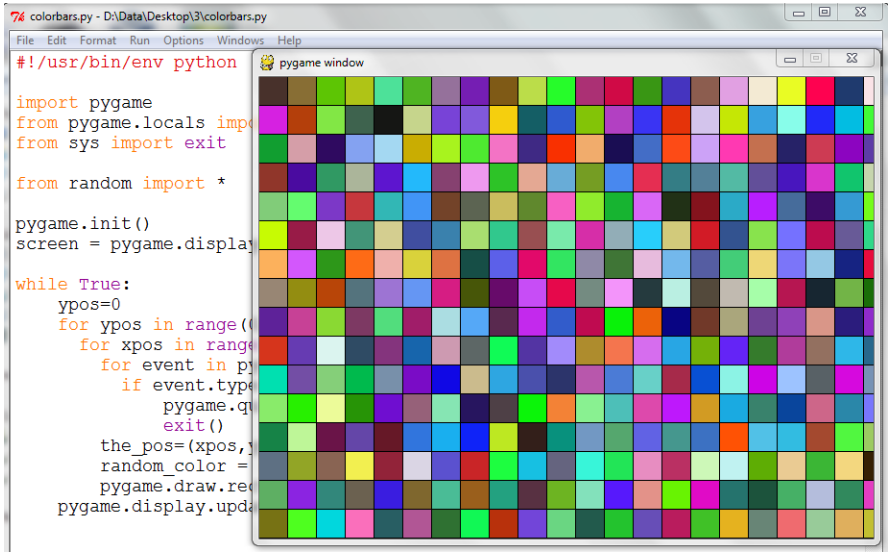
```
    pygame.display.update()
```

Αυτό και αν είναι μυστήριο! Τι update; Πολύ απλά: ότι και αν κάνετε σε οποιαδήποτε επιφάνεια δεν γίνεται ορατό πριν καλέσετε την update. Γιατί πολύ απλά όλη αυτή η σχεδίαση, το γέμισμα με χρώμα, η επιφάνεια με το κείμενο που βάλατε πάνω από τη screen δεν έγιναν απευθείας στην μνήμη που αντιπροσωπεύει την οθόνη σας (τη RAM δηλ. της κάρτας γραφικών). Έγιναν σε μια άλλη, προσωρινή περιοχή μνήμης (buffer) η οποία αντιγράφεται πολύ γρήγορα στη μνήμη οθόνης. Έχετε δηλ. δημιουργήσει πλήρως το καρέ σας πριν το εμφανίσετε, αποφεύγοντας την πιθανότητα ο χρήστης της εφαρμογής σας να δει τα περιεχόμενα να σχεδιάζονται σιγά – σιγά μπροστά του (και να κατηγορεί το πρόγραμμα σας ή το μηχάνημα του για χαμηλή ταχύτητα!). Και τώρα καταλαβαίνετε τι σημαίνει το `DOUBLEBUF` που είδαμε πριν – τις δύο περιοχές μνήμης που χρησιμοποιούμε για τη σχεδίαση της οθόνης μας!

4.4 Event Driven Programming

Αν έχετε γράψει ένα πρόγραμμα για παραθυρικό περιβάλλον (ακόμα και στη Visual Basic η οποία κρύβει πολλές λεπτομέρειες) θα ξέρετε ότι μια σημαντική διαφορά με μια εφαρμογή κονσόλας είναι ο χειρισμός των *συμβάντων, ή events*.

Οι εφαρμογές που γράφουμε στην κονσόλα δεν έχουν την έννοια των συμβάντων: εκτελούνται η μια εντολή μετά την άλλη και δέχονται είσοδο από το χρήστη μόνο όταν υπάρχει κάποια εντολή τύπου `input`. Σε μια παραθυρική εφαρμογή όμως τα πράγματα είναι διαφορετικά. Τι γίνεται για παράδειγμα αν ξαφνικά ο χρήστης κάνει κλικ στο close (X) του παραθύρου; Ίσως πιστεύετε ότι το λειτουργικό χειρίζεται αυτές τις λειτουργίες αλλά δεν είναι ακριβώς έτσι. Απλά το λειτουργικό στέλνει ένα μήνυμα στην εφαρμογή σας “Ο χρήστης πάτησε το close, μάλλον κάτι πρέπει να κάνεις για αυτό”. Η εφαρμογή σας πρέπει να είναι έτοιμη ανά πάσα στιγμή να επεξεργαστεί τα μηνύματα που δέχεται από το λειτουργικό – αν δεν το κάνει και ο χρήστης εξακολουθεί να πατάει μανιωδώς το close, το λειτουργικό θα τερματίσει την εφαρμογή αφού πρώτα δείτε το γνωστό μήνυμα “Δεν αποκρίνεται”. Εκτός φυσικά από τα μηνύματα που μας στέλνει το λειτουργικό θα πρέπει ανά τακτά διαστήματα να διαβάζουμε π.χ. και το πληκτρολόγιο με το οποίο κατευθύνουμε το παιχνίδι μας. Περισσότερα για αυτό θα δούμε σε επόμενα κεφάλαια.



Εικόνα 4.2: Αντιγράψτε αυτό το προγραμματάκι από το παράρτημα (σελ. 170) και εκτελέστε το. Έπειτα αρχίστε τις παραλλαγές! Δοκιμάστε να αλλάξετε το μέγεθος των τετραγώνων, τις αποστάσεις μεταξύ τους ή και το σχήμα τους. Κάντε τα παραλληλόγραμμα ή κύκλους. Δείτε και στο `http://www.pygame.org/docs/ref/draw.html` για βοήθεια σχετικά με τις εντολές σχεδίασης.

Ναι λοιπόν, το πρόγραμμα μας τρέχει και εκτελείται ακόμα και μετά που θα τυπωθεί το μήνυμα “Hello World!”. Αν τερμάτιζε εκεί, το παράθυρο θα έκλεινε πριν καλά καλά προλάβουμε να δούμε το μήνυμα. Το πρόγραμμα μας τερματίζει μόλις ο χρήστης πιέσει το close — τότε λαμβάνεται το μήνυμα `QUIT`:

```
def getQuit():
  for event in pygame.event.get():
    if event.type == QUIT:
      return True
```

Όλα τα συμβάντα που λαμβάνει το πρόγραμμα μας καταλήγουν σε ένα *queue* το ένα πίσω από το άλλο. Τα διαβάζουμε χρησιμοποιώντας την `pygame.event.get()` η οποία δημιουργεί μια λίστα από events τα οποία επεξεργαζόμαστε ένα – ένα με τη `for`. Παρατηρήστε ότι στο συγκεκριμένο πρόγραμμα δεν ελέγχουμε για τίποτα άλλο εκτός από το `QUIT` καθώς το πρόγραμμα μας είναι πολύ απλό.

Η επεξεργασία των events πρέπει να γίνεται συχνά. Παρατηρήστε ότι καλούμε την `getQuit` σε κάθε κύκλο που κάνει ο βρόχος `while`.

4.5 Frames και Framerate

Οι πιο παρατηρητικοί από εσάς θα είδατε ότι το κείμενο “Hello World!” τυπώνεται συνέχεια μέσα στο βρόχο while! Είναι απαραίτητο αυτό; Πόσες φορές το δευτερόλεπτο συμβαίνει; Τι θα γίνει αν μετακινήσουμε τις εντολές εμφάνισης έξω από το βρόχο και αφήσουμε σε αυτόν μόνο την επεξεργασία των συμβάντων;

Κάθε φορά που εκτελείται ο βρόχος, το πρόγραμμα μας εμφανίζει το μήνυμα στην οθόνη. Βέβαια το μήνυμα μας είναι στατικό και δεν βλέπουμε καμιά αλλαγή. Σε ένα παιχνίδι όμως κάθε εκτέλεση του βρόχου περιέχει κάποια αλλαγή — κάποια κίνηση — και δημιουργεί ένα *καρέ (frame)* του παιχνιδιού. Όσο πιο πολλά καρέ παράγονται σε ένα δευτερόλεπτο, τόσο πιο ομαλή είναι η κίνηση του παιχνιδιού. Τυπικά χρειαζόμαστε 50 καρέ το δευτερόλεπτο για να έχουμε τέλεια κίνηση. Είναι αυτό που ονομάζουμε *framerate* και για το οποίο γίνεται πάντα τόσος λόγος σε όσους παίζουν παιχνίδια first person shooters (και ξοδεύουν αντίστοιχα εκατοντάδες ευρώ για κάρτες γραφικών).

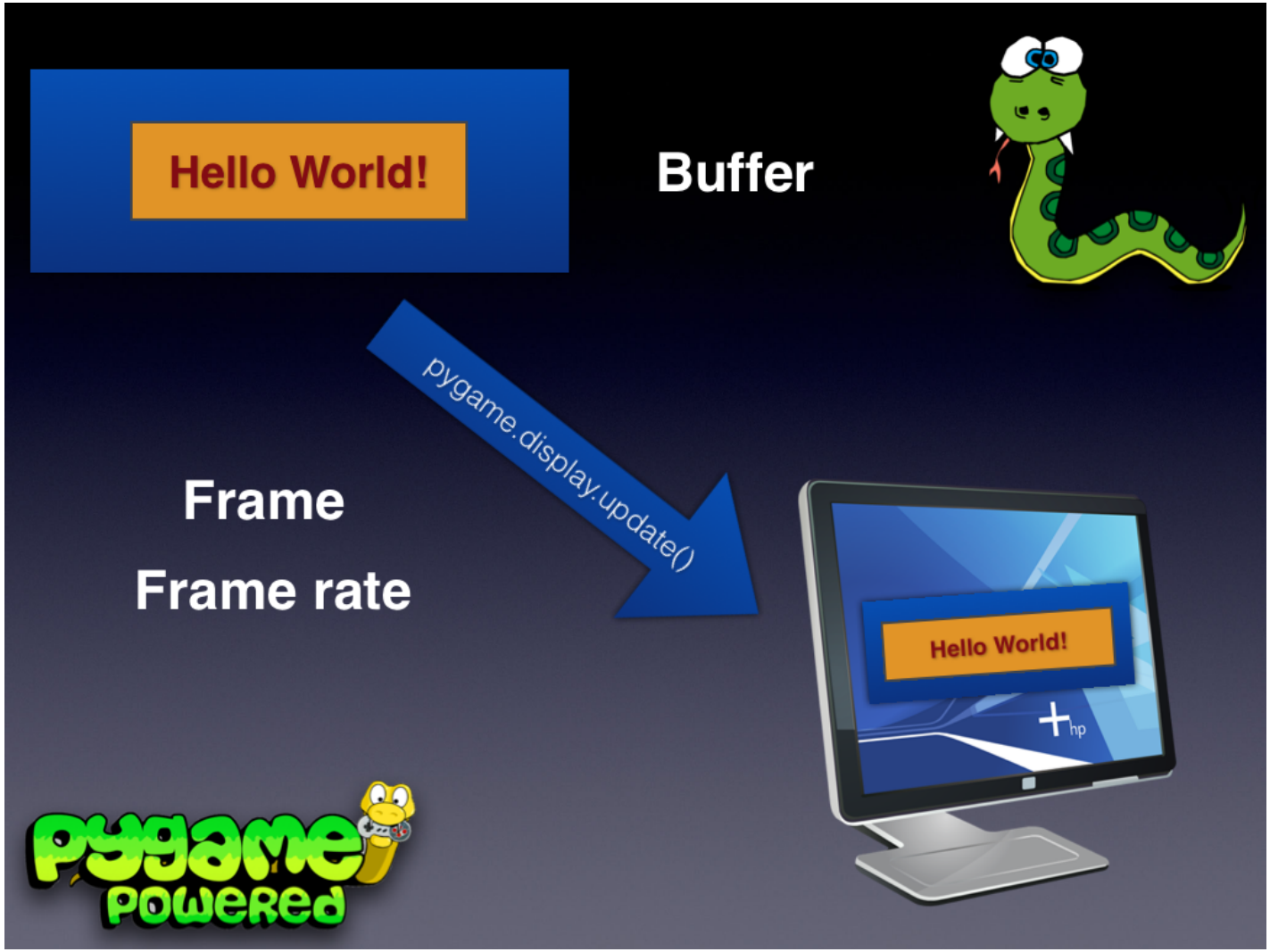
Τη δεδομένη στιγμή το “παιχνίδι” μας τρέχει με τόσα καρέ το δευτερόλεπτο όσα αντέχει ο υπολογιστής μας! Μπορούμε όμως στο pygame τόσο να μετρήσουμε το framerate που επιτυγχάνει το μηχάνημα μας (το οποίο είναι απαραίτητο όπως θα δείτε) όσο και να το περιορίσουμε σε ένα συγκεκριμένο αριθμό αν θέλουμε.

Και για να απαντήσω και στο άλλο ερώτημα σας, στο συγκεκριμένο πρόγραμμα πράγματι θα μπορούσατε να μετακινήσετε τις εντολές για το κείμενο έξω από το βρόχο. Αλλά είναι κάτι που δεν θα δείτε σε παιχνίδια!

4.6 Bouncing Ball – Επιτέλους Γραφικά και Κίνηση!

Για να δούμε επιτέλους και λίγη κίνηση! Το πρόγραμμα που σας παρουσιάζουμε εδώ είναι το γνωστό bouncing ball:

```
1 #
2 # Bouncing ball
3 #
4
5 import pygame
6 from pygame.locals import *
7 from sys import exit
```



Εικόνα 4.3: Μια εικόνα αξίζει όσο χίλιες λέξεις. Πως λειτουργεί το buffering, σχηματικά. Η οθόνη σχεδιάζεται στην ενδιάμεση μνήμη και με το update μεταφέρεται απευθείας στη μνήμη οθόνης.

```
8  def getQuit():
9    for event in pygame.event.get():
10     if event.type == QUIT:
11       return True
12   return False
13
14 def main():
15   pygame.init()
16   ballimage = 'soccer-ball.png'
17   x,y = 100.0,100.0
18   xspeed,yspeed = 50,50
19   windowsize = (640,480)
20   surfacecolor = (50,80,250)
21   screen = pygame.display.set_mode(windowsize, DOUBLEBUF)
22   ball = pygame.image.load(ballimage)
23   ballwidth = ball.get_width()
24   ballheight = ball.get_height()
25   clock = pygame.time.Clock()
26
27   # Uncomment the framerate line and change
28   # time = clock.tick() in main loop to
29   # time = clock.tick(framerate)
30   # to limit the animation to a specific framerate
31
32   #framerate = 30
33
34   textfont = pygame.font.SysFont("Arial",24)
```

```
  #
  # Main loop
  #

  endprogram = False
  while not endprogram:
    screen.fill(surfacecolor)
    screen.blit(ball, (x, y))
    time = clock.tick()
    thetext = textfont.render(str(1000/time), True, (255,0,0),(255,255,0))
    screen.blit(thetext,(0,0))
    time = time / 1000.0
    distance_x = time * xspeed
    distance_y = time * yspeed
    x = x + distance_x
    y = y + distance_y

    if (x > (640.0-ballwidth) or x<=0.0):
      xspeed = -xspeed
    if (y > (480.0-ballheight) or y<=0.0):
      yspeed = -yspeed
    pygame.display.update()
    endprogram = getQuit()

  pygame.quit()
  exit()

if __name__ == "__main__":
  main()
```

Εικόνα 4.4: Το παλιό και το καινούριο: Μια εικόνα του πρωτότυπου bouncing ball, όπως το έγραψα το 1985 στον TI-99/4A. Είναι εμφανώς… retro, αλλά φυσικά έχει πλάκα! Την εποχή εκείνη η κίνηση δεν γινόταν με καρέ (όπου συνήθως ξανασχεδιάζουμε όλη την οθόνη) αλλά με sprites ή κίνηση χαρακτήρων, όπου απλά μετακινούμε το αντικείμενο στη νέα του θέση και το σβήνουμε από την παλιά. Δεν θα ήταν άλλωστε δυνατόν για ένα μηχάνημα της εποχής να ανασχεδιάζει μια ολόκληρη οθόνη 50–60 φορές το δευτερόλεπτο. Και ούτε λόγος για… double buffering!

Για να δούμε μερικά ενδιαφέροντα σημεία:

```
x,y = 100.0,100.0
xspeed,yspeed = 50,50
```

Αρχική θέση της μπάλας και αρχική ταχύτητα στους δύο άξονες.

```
ballimage = 'soccer-ball.png'
ball = pygame.image.load(ballimage)
```

Φόρτωση της μπάλας από το αρχείο `soccer-ball.png`. Δημιουργείται μια επιφάνεια (surface) που ανατίθεται στη μεταβλητή `ball`.

```
ballwidth = ball.get_width()
ballheight = ball.get_height()
```

Μπορούμε να ρωτήσουμε ένα αντικείμενο τύπου surface να μας πει τις διαστάσεις του! Πρόκειται για τις μεθόδους `get_width()` και `get_height()`.

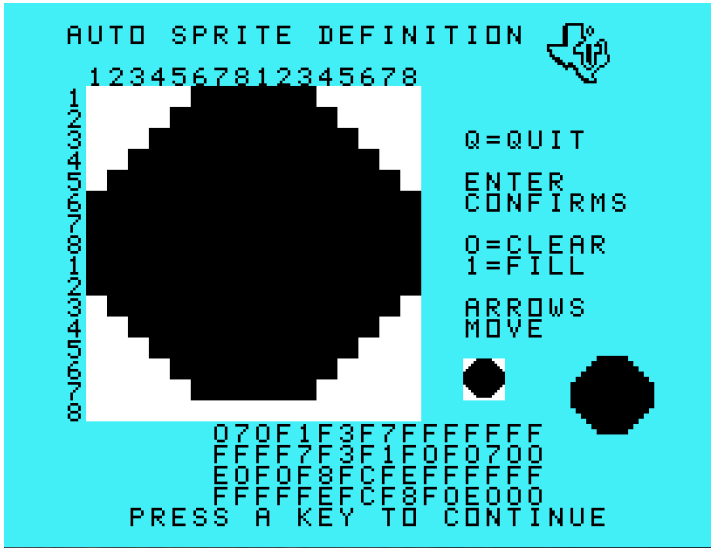
```
clock = pygame.time.Clock()
```

Το παραπάνω δημιουργεί ένα αντικείμενο τύπου `clock`. Μέσα στο βρόχο υπάρχει η εντολή:

```
    time = clock.tick()
```

Στη μεταβλητή `time` θα βρείτε σε χιλιοστά δευτερολέπτου το χρόνο που πέρασε από την προηγούμενη φορά που εκτελέστηκε η εντολή. Καθώς εκτελείται μια φορά σε κάθε κύκλο του `while`, μπορεί να χρησιμοποιηθεί για τον υπολογισμό του `framerate = 1000 / time`.

Πρέπει τόσο στο γρήγορο, όσο και στο αργό μηχάνημα η μπάλα να κινείται με σταθερή ταχύτητα – ανεξάρτητα από το framerate! Για να το επιτύχουμε πρέπει στη μονάδα του χρόνου (το 1 sec) η απόσταση που κινείται η μπάλα να είναι σταθερή. Σε ένα γρήγορο μηχάνημα η κίνηση



Εικόνα 4.5: Την παλιά καλή(;) εποχή, δεν υπήρχε το Internet για να κατεβάσει κανείς την… μπάλα της αρεσκείας του σε έτοιμο bitmap. Τα γραφικά σχεδιάζονταν στο χέρι (σε χαρτί με τετραγωνάκια!) ή σε προγράμματα που στις μέρες μας θα έμοιαζαν με icon editors. Και εννοείται ότι και αυτά τα προγράμματα τα γράφαμε μόνοι μας! Στη φώτο, το Auto Sprite Definition, μια δική μου εκδοχή σε ένα αντίστοιχο πρόγραμμα της Texas Instruments που επιτρέπει τη σχεδίαση κινούμενων γραφικών (sprites) για παιχνίδια. Άπειρα pacman και διαστημοπλοιάκια έχουν σχεδιαστεί σε αυτό το πρόγραμμα!

θα είναι ωραία και ομαλή. Σε ένα αργό όμως θα φαίνεται να κάνει πηδηματάκια! Ωστόσο, η πραγματική ταχύτητα θα είναι ίδια. Πως γίνεται αυτό; Αν θυμηθείτε λίγο τη φυσική σας, για ομαλή κίνηση ο τύπος είναι:

```
Απόσταση = Ταχύτητα * Χρόνος
```

(Ναι, τελικά η φυσική δεν είναι μόνο για να λύνετε ασκήσεις. Είναι και για να γράφετε παιχνίδια, αν και αυτό μάλλον παρέλειψαν να σας το πουν οι καθηγητές σας). Και αυτό ακριβώς κάνουμε και εμείς:

```
    time = time / 1000.0
```

Μετατροπή του χρόνου σε δευτερόλεπτα.

```
    distance_x = time * xspeed
    distance_y = time * yspeed
```

Υπολογισμός της απόστασης που κινήθηκε η μπάλα στους δύο άξονες.

Εικόνα 4.6: Οι συντεταγμένες ξεκινάνε από το (0,0) στην πάνω αριστερή γωνία της οθόνης και αυξάνονται προς τα δεξιά και προς τα κάτω. Σε ένα παράθυρο μεγέθους 640X480, μια μπάλα στη μέση θα βρίσκονταν στις συντεταγμένες (320,240).

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

Υπολογισμός της νέας θέσης της μπάλας.

Και φυσικά αντιλαμβάνεστε ότι τα `if` που ακολουθούν αναλαμβάνουν να αντιστρέψουν την ταχύτητα της μπάλας όταν χτυπήσει σε κάποιο τοίχωμα (δηλ. το border του παραθύρου!)

Το ενδιαφέρον είναι ότι το “παιχνίδι” μας τυπώνει το framerate στην οθόνη, με την εντολή:

```
    thetext = textfont.render(str(1000/time), True, (255,0,0),(255,255,0))
    screen.blit(thetext,(0,0))
```

Όσο πιο γρήγορος είναι ο υπολογιστής σας και η κάρτα γραφικών σας, τόσο μεγαλύτερος ο αριθμός που θα δείτε. Μπορούμε όμως να τον περιορίσουμε. Βγάλτε το σχόλιο από τη γραμμή:

```
  framerate = 30
```

και αλλάξτε μέσα στο βρόχο την εντολή `tick` ώστε να δείχνει:

```
    time = clock.tick(framerate)
```

Η γραμμή `time` τώρα θα προκαλεί όση καθυστέρηση χρειάζεται ώστε το πρόγραμμα σας να εκτελείται με το συγκεκριμένο framerate! Δοκιμάστε με διαφορετικούς αριθμούς για να δείτε το αποτέλεσμα. Πάνω από 60 καρέ το δευτερόλεπτο ίσως να μη βλέπετε αισθητή διαφορά. Αλλά σε χαμηλά framerates θα βλέπετε κακή ποιότητα κίνησης. Η ταχύτητα όμως της μπάλας θα είναι ίδια.

Σαν άσκηση, δοκιμάστε το πρόγραμμα με διαφορετικές ταχύτητες μπάλας και framerates. Μετατρέψτε το ώστε να κινούνται δύο μπάλες αντί για μία! (Hint: μπορείτε να κάνετε blit το ίδιο surface σε δύο διαφορετικές θέσεις) Όταν οι δύο μπάλες συναντιούνται στην οθόνη, κάποια περνάει πάνω από την άλλη. Ποια και γιατί; Μπορείτε να το βρείτε;

Σας αφήνω τώρα να παίξετε και σας προκαλώ να γράψετε τα δικά σας απλά προγραμματάκια animation με βάση αυτά που είδατε μέχρι τώρα.

Θα βρείτε τα προγράμματα αυτής της ενότητας στο Παράρτημα: Hello Pygame σελ. 168, Colorbars σελ. 170, Bouncing Ball σελ. 171.

4.7 Τα Βλέπω Διπλά! Δύο Bouncing Balls

Πως θα μετατρέψετε το bouncing ball ώστε να έχει δύο μπάλες; Ελπίζουμε να μη σας φαίνεται δύσκολο. Ας δούμε μια απλοϊκή λύση. Κάτω από την γραμμή:

```
xspeed,yspeed = 50.0 , 50.0
```

προσθέστε:

```
x2,y2 = 50.0, 50.0
xspeed2,yspeed2= 30,30
```

Βάζουμε λίγο διαφορετική ταχύτητα για να καταλαβαίνουμε ποια μπάλα είναι η πρώτη και ποια η δεύτερη. Επίσης ξεκινάμε και από άλλη αρχική θέση. Δίνουμε τώρα όλο τον κύριο βρόχο του προγράμματος (αφαιρέσαμε τα αρχικά σχόλια για συντομία):

```
  endprogram = False
  while not endprogram:
    screen.fill(surfacecolor)
    screen.blit(ball, (x, y))
    screen.blit(ball, (x2,y2))
    time = clock.tick()
    thetext = textfont.render(str(1000/time), True, (255,0,0),(255,255,0))
    screen.blit(thetext,(0,0))
    time = time / 1000.0
    distance_x = time * xspeed
    distance_y = time * yspeed
    distance_x2 = time * xspeed2
    distance_y2 = time * yspeed2
    x = x + distance_x
    y = y + distance_y
    x2 = x2 + distance_x2
    y2 = y2 + distance_y2
    if (x > (640.0-ballwidth) or x<=0.0):
      xspeed = -xspeed
    if (y > (480.0-ballheight) or y<=0.0):
      yspeed = -yspeed
```

```
    if (x2 > (640.0-ballwidth) or x2<=0.0):
      xspeed2 = -xspeed2
    if (y2 > (480.0-ballheight) or y2<=0.0):
      yspeed2 = -yspeed2

    pygame.display.update()
    endprogram = getQuit()

  pygame.quit()
  exit()
```

Είναι εύκολο, αλλά είναι και κακογραμμένο! Γιατί, αν το να βάλουμε μερικές ακόμα μεταβλητές για να φτιάξουμε μια δεύτερη μπάλα είναι απλό, τι θα λέγατε αν σας έλεγα να το φτιάξετε για 20 μπάλες; ή για 100 μπάλες; Θα πρέπει σίγουρα να αλλάξετε τρόπο σκέψης! Και φυσικά ο αντικειμενοστραφής προγραμματισμός (που θα δούμε στο επόμενο κεφάλαιο) θα μας βοηθήσει.

Α, και τώρα φυσικά που θα το τρέξετε, θα δείτε και ποια μπάλα περνάει πάνω από την άλλη. Και το αποτέλεσμα είναι απόλυτα λογικό.

Κατεβάστε τα προγράμματα αυτού του κεφαλαίου: Hello Pygame: http://www.freebsdworld.gr/files/hello-pygame.zip, Colorbars: http://www.freebsdworld.gr/files/colorbars.zip, Bouncing Ball: http://www.freebsdworld.gr/files/bouncing.zip

Κεφάλαιο 5

Εισαγωγή στον Αντικειμενοστραφή Προγραμματισμό

5.1 Εισαγωγή

Ελπίζουμε να μη ζαλιστήκατε (πολύ) με τις τόσες νέες έννοιες στο προηγούμενο κεφάλαιο! Αντικείμενα, συμβάντα, βιβλιοθήκη pygame, μέθοδοι, μέχρι και… φυσική Λυκείου είχαμε! Αλλά φυσικά μπρος στα… graphics τι είναι ο πόνος. Σίγουρα είστε διατεθειμένοι να κουραστείτε λίγο ακόμα – εξάλλου πλησιάζει η ώρα για το πρώτο μας πραγματικό γραφικό παιχνίδι. Και φυσικά, δεν χρειάζεται να μας το πείτε: το ξέρουμε ότι είστε ήδη εθισμένοι με την python, το προγραμματισμό, το pygame και όλα τα άλλα που ίσως δεν πολυκαταλαβαίνετε ακόμα. Αλλά δεν πειράζει: το ξενύχτι είναι μέρος της μαγείας! Πάμε λοιπόν να εξερευνήσουμε τον αντικειμενοστραφή προγραμματισμό.

5.2 Αντικειμενοστραφής Προγραμματισμός για… Πρωτάρηδες

Ή for Dummies θα λέγαμε αν κρατούσαμε τον Αγγλικό τίτλο της δημοφιλούς σειράς βιβλίων Βέβαια οι αναγνώστες μας δεν είναι dummies – κάθε άλλο μάλιστα. Dummies είναι όσοι χρησιμοποιούν μόνο ετοιματζίδικα πράγματα!

Είναι σχετικά δύσκολο να πάρετε μια ικανοποιητική απάντηση στην ερώτηση “Τι είναι αντικειμενοστραφής προγραμματισμός”. Είναι μάλλον πιο εύκολο να το δει κανείς στα προγράμματα παρά να προσπαθεί με ένα θεωρητικό ορισμό. Ωστόσο οι περισσότεροι συμφωνούμε στα παρακάτω:

Αντικείμενο: Είναι μια συλλογή από δεδομένα — μπορείτε να τα πείτε *χαρακτηριστικά (attributes), ιδιότητες (properties)* — με την ιδιαιτερότητα ότι έχει τις δικές του συναρτήσεις (*μεθόδους*) για να τα χειρίζεται.

Για να δημιουργήσουμε δικά μας αντικείμενα (που να περιέχουν τα δεδομένα και τις μεθόδους της επιλογής μας) γράφουμε μια *κλάση*. Σε αυτή περιγράφουμε τα δεδομένα που θα περιέχει το αντικείμενο μας καθώς και τις μεθόδους που τα χειρίζονται. Όταν δημιουργήσουμε ένα αντικείμενο που ανήκει στην κλάση μας, αυτόματα θα διαθέτει τα χαρακτηριστικά και τις μεθόδους που ορίσαμε.

Δεν είναι όμως μόνο αυτό. Στο κάτω – κάτω θα μπορούσατε να σκεφτείτε ότι δεν αλλάζει κάτι ιδιαίτερα: Αντί να έχουμε συναρτήσεις και να τους δίνουμε τα δεδομένα μας, τα ίδια τα δεδομένα μας έχουν τις δικές τους συναρτήσεις. Ε και λοιπόν; Ο αντικειμενοστραφής προγραμματισμός έχει ακόμα τρεις βασικές έννοιες:

Πολυμορφία (polymorphism): Φανταστείτε δύο αντικείμενα από διαφορετικές κλάσεις να διαθέτουν μια μέθοδο με το ίδιο όνομα. Ανάλογα με το αντικείμενο που χρησιμοποιείται όταν γίνεται η κλήση, καλείται κάθε φορά η σωστή μέθοδος!

Ενθυλάκωση (encapsulation): Άλλο πράγμα να γράψεις πως δουλεύει ένα αντικείμενο (στο κομμάτι που περιγράφεται η κλάση) και άλλο να το χρησιμοποιείς μετά για το κανονικό σου πρόγραμμα. Στη χρήση, δεν μας ενδιαφέρει πλέον πως λειτουργεί εσωτερικά το αντικείμενο, αρκεί να κάνει αυτό που θέλουμε. Η ενθυλάκωση κρύβει την εσωτερική πολυπλοκότητα του αντικειμένου όταν πλέον δεν χρειάζεται να την βλέπουμε. Σκεφτείτε το και ως εξής: Δεν ξέρετε πως λειτουργεί εσωτερικά η επιφάνεια (surface) του pygame. Μπορείτε όμως να τη χρησιμοποιήσετε γιατί το μόνο που χρειάζεστε να ξέρετε είναι οι λειτουργίες που σας παρέχουν οι μέθοδοι.

Κληρονομικότητα (inheritance): Με αυτήν την καταπληκτική δυνατότητα μπορείτε να ξεκινήσετε δημιουργώντας μια γενική κλάση αντικειμένων και από αυτή να παράγετε πιο εξειδικευμένες κλάσεις με πρόσθετα χαρακτηριστικά που να κληρονομούν όμως και τα χαρακτηριστικά και τις μεθόδους της γονικής τους κλάσης.

Ειδικά η κληρονομικότητα σε συνδυασμό με την πολυμορφία μας επιτρέπουν να κάνουμε ιδιαίτερα καλά κόλπα. Όμως αντί να σας τα λέμε θεωρητικά, θα σας τα δείξουμε με παραδείγματα. Όχι τίποτα άλλο δηλαδή, αλλά κάποιοι από εσάς αναμφίβολα μένετε σε… ψηλούς ορόφους και δεν μπορούμε να συνεχίσουμε άλλο τη θεωρία. Και πραγματικά, αν κάπου μας χάσατε με τα παραπάνω μην ανησυχείτε: όλα θα ξεκαθαρίσουν στην πορεία.

5.3 Θεός για… μια Ημέρα!

Φανταστείτε ότι είστε ο… θεός και πρόκειται να δημιουργήσετε τη ζωή πάνω στη Γη. Σε python βέβαια! Έχετε ήδη ασχοληθεί με φυτά και άλλα τέτοια… βαρετά, και έχετε φτάσει στα θηλαστικά. Σκέφτεστε λοιπόν να φτιάξετε μια κλάση θηλαστικών και από αυτή να παράγετε ότι άλλο

χρειάζεστε (γάτες, σκύλους, ανθρώπους and so on). Προφανώς ένα αντικείμενο που ανήκει σε μια τέτοια κλάση έχει διάφορα χαρακτηριστικά (attributes) και λειτουργίες (μεθόδους) αλλά για το παράδειγμα μας θα επικεντρωθούμε σε ένα: την ομιλία.

Όλα τα θηλαστικά έχουν κάποιο είδος φωνής: οι σκύλοι γαβγίζουν (bark), οι γάτες νιαουρίζουν (meow), ε και οι άνθρωποι… μερικές φορές καλύτερα να μασάνε παρά να μιλάνε. Καθώς φαντάζεστε, σαν θεός θα πρέπει να φτιάξετε μια μέθοδο `speak` που όταν την καλείτε να κάνει το σωστό, ανάλογα με το ζώο. Στο κάτω κάτω δεν υπάρχουν σκύλοι που να νιαουρίζουν και γάτες που να γαβγίζουν!

Ξεκινάτε δημιουργώντας μια κλάση που περιγράφει τα θηλαστικά γενικά:

```
class mammal:
  def speak(self):
    print ”Mpla mpla!”
```

Η δημιουργία μιας κλάσης ξεκινάει με την εντολή `class`. Για να γράψουμε μια μέθοδο που ανήκει στην κλάση, την γράφουμε ως συνάρτηση (με την `def` που ξέρουμε ήδη) μέσα στην κλάση. Το `self` που βλέπετε σημαίνει ότι το πρώτο όρισμα της συνάρτησης είναι ένα αντικείμενο της ίδιας της κλάσης που ορίζουμε.

Μπορείτε τώρα να δημιουργήσετε ένα ζώο της γενικής κατηγορίας `mammal` και να το βάλετε να μιλήσει. Όταν δημιουργούμε ένα αντικείμενο που ανήκει σε μια κλάση λέμε ότι έχουμε φτιάξει ένα *instance* αυτής της κλάσης:

```
animal = mammal()
animal.speak()
```

Το οποίο βέβαια θα σας τυπώσει:

```
Mpla mpla!
```

Βέβαια εμείς δεν ξέρουμε κανένα ζώο να έχει για φωνή το μπλα μπλα, οπότε θα φτιάξουμε τώρα την ειδική κλάση γάτας και σκύλου, η οποία προέρχεται από τη `mammal`:

```
class cat(mammal):
  def speak(self):
    print "Meow!"

class dog(mammal):
  def speak(self):
    print "Bark!"

# Δημιουργία γάτας!

thecat = cat()

# Δημιουργία σκύλου!

thedog = dog()

# Ομιλία!

thecat.speak()
thedog.speak()
```

Φτιάξαμε μια κλάση `cat` και μια `dog` που προέρχονται από τη `mammal` (το δηλώσαμε αυτό γράφοντας το όνομα της γονικής κλάσης μέσα στην παρένθεση) αλλά καθεμιά από αυτές υλοποιεί ξανά τη μέθοδο `speak`. Έτσι μια γάτα όταν καλούμε τη μέθοδο `speak` δεν λέει μπλα μπλα, αλλά meow και ένας σκύλος bark! Τώρα είμαι σίγουρος ότι αν ο γείτονας σας έχει σκύλο που γαβγίζει κάθε βράδυ, ξέρετε ποια είναι η λύση: Να του γράψετε μια μέθοδο `shutup` και να την καλέσετε!

Συνήθως δίνουμε ονόματα στις γάτες και τους σκύλους μας, οπότε σαν θεός που είστε θέλετε να το προβλέψετε αυτό στην κλάση σας. Κανένα πρόβλημα. Μόνο που δεν χρειάζεται να το γράψετε χωριστά για το σκύλο και τη γάτα. Βάλτε το απλώς στην κλάση `mammal` και θα το κληρονομήσουν:

```
class mammal:
  def speak(self):
    print "Mpla mpla!"
  def setName(self, name):
    self.name = name
  def getName(self):
    return self.name
```

Δοκιμάστε προσθέτοντας τις παρακάτω γραμμές στο κύριο πρόγραμμα:

```
thecat.setName("Catia")
thedog.setName("Lassie")
print thecat.getName()
print thedog.getName()
```

Μπορούμε επίσης να έχουμε απευθείας πρόσβαση στα δεδομένα της κλάσης, χωρίς να χρησιμοποιήσουμε τις συναρτήσεις `getName` και `setName` που φτιάξαμε:

```
print thedog.name
thecat.name="Susan"
print thecat.name
```

Σε πολλές περιπτώσεις, όταν δημιουργούμε ένα αντικείμενο θέλουμε να έχει από την αρχή κάποια δεδομένα ή ιδιότητες. Για παράδειγμα, σαν θεός αποφασίσατε ότι όλες οι γάτες θα γεννιούνται… μαύρες και όλα τα θηλαστικά θα έχουν τέσσερα πόδια! Για να δώσετε αρχική κατάσταση σε ένα αντικείμενο κατά τη δημιουργία του, χρειάζεστε μια ειδική συνάρτηση που ονομάζεται *constructor*. Για το παράδειγμα μας:

```
class mammal(object):
  def __init__(self):
    self.legs=4
  def speak(self):
    print ”Mpla mpla!”
  def setName(self, name):
    self.name = name
  def getName(self):
    return self.name

class cat(mammal):
  def __init__(self):
    super(cat,self).__init__()
    self.color=”black”
  def speak(self):
    print ”Meow!”

thecat = cat()
print thecat.color
print thecat.legs
```

Προσέξτε τη συνάρτηση `__init__` στην κλάση `mammal`: Ορίζει ότι όλα τα θηλαστικά έχουν… τέσσερα πόδια. (Θεός είστε ότι θέλετε κάνετε.) Αντίστοιχα, ο constructor για τις γάτες ορίζει ότι το χρώμα τους θα είναι μαύρο.

Με μια προσεκτικότερη ματιά, θα δείτε ότι το class `mammal` φαίνεται να προέρχεται πλέον από την γενικότερη κλάση object. Αυτό το χρειαζόμαστε για να λειτουργήσει η συνάρτηση `super` στον constructor της γάτας – και θα δούμε τώρα τι κάνει αυτή.

Όταν δημιουργείτε μια γάτα, καλείται ο constructor στην κλάση της. Προσέξτε ότι δεν καλείται ο constructor του `mammal`, όχι αυτόματα δηλαδή. Σε κάποιες γλώσσες, όταν δημιουργούμε ένα αντικείμενο μιας *υποκλάσης (subclass)*, καλείται πρώτα αυτόματα ο constructor της *γονικής κλάσης (superclass)*. Στην python αυτό δεν συμβαίνει. Θα πρέπει αν θέλουμε να καλέσουμε τον constructor της mammal να το κάνουμε μέσα από τον constructor της cat. Θα μπορούσε να γίνει ως εξής:

```
mammal.__init__(self)
```

Αλλά ο νέος Σωστός Τρόπος™ είναι:

```
super(cat,self).__init__()
```

Με την `super` ουσιαστικά λέμε να χρησιμοποιηθεί η κλάση από την οποία προέρχεται η `cat` (δηλ. η `mammal`).

5.4 Ένα Αντικειμενοστραφές… Bouncing Ball

Στο προηγούμενο κεφάλαιο, σας δώσαμε ένα μάλλον κακογραμμένο κομμάτι κώδικα το το οποίο υλοποιούσε ένα bouncing ball με δύο μπάλες. Όπως διαπιστώσατε δεν είχε σημαντικές διαφορές με το πρωτότυπο. Γιατί απλά, για δύο μπάλες χρειάζεστε:

- Το ίδιο γραφικό δύο φορές. Εύκολο, το έχουμε ήδη.
- Διπλές μεταβλητές που δείχνουν: θέση μπάλας, απόσταση που διανύθηκε, ταχύτητα σε κάθε άξονα.

Αν προσέξετε λοιπόν το listing θα διαπιστώσετε ότι οι μεταβλητές για τη μια μπάλα… κλωνοποιούνται για τη δεύτερη:

- Θέση πρώτης μπάλας: μεταβλητές `x` και `y`. Θέση δεύτερης μπάλας: `x2` και `y2`
- Ταχύτητα πρώτης μπάλας: `xspeed` και `yspeed`, δεύτερης μπάλας `xspeed2` και `yspeed2`
- Απόσταση που διάνυσε η πρώτη μπάλα: `distance_x` και `distance_y`, δεύτερη μπάλα `distance_x2` και `distance_y2`

Αν και μπορεί αρχικά να σας φάνηκε ικανοποιητική η λύση, σίγουρα κολλήσατε όταν σας ζητήσαμε να το φτιάξετε για τρεις, πέντε, δέκα μπάλες. Όχι ότι δεν γίνεται να συνεχίσετε να προσθέτετε μεταβλητές, αλλά είναι προφανές ότι η λύση αυτή δεν είναι ούτε κομψή ούτε γενική. Τι θα μπορούσατε να σκεφτείτε;

Έχετε καταλάβει σίγουρα ότι όταν έχουμε δεδομένα ίδιου τύπου, που θέλουμε να τα χειριστούμε μαζικά με τον ίδιο τρόπο, βολεύει αντί να τα αποθηκεύσουμε σε απλές μεταβλητές να χρησιμοποιήσουμε κάποια δομή που μας παρέχει η γλώσσα προγραμματισμού μας για αυτό το σκοπό. Για παράδειγμα, οι παραδοσιακές γλώσσες (όπως η BASIC από την οποία άλλωστε προέρχεται και το αρχικό μας bouncing ball) χρησιμοποιούν πίνακες. Στην python όπως θα έχετε καταλάβει βασική δομή για αυτό το σκοπό είναι οι λίστες.

Οι λίστες όπως και οι πίνακες μας επιτρέπουν μαζική επεξεργασία των δεδομένων καθώς μας επιτρέπουν να εκτελέσουμε σε κάθε στοιχείο τους την ίδια λειτουργία, διατρέχοντας τα με κάποιο βρόχο. Πράγματι μια πιθανή λύση θα ήταν να δημιουργούσαμε μια λίστα που να περιέχει μέσα τις τιμές των στοιχείων:

```
[x, y, xspeed, yspeed]
```

Επειδή μάλιστα θα χρειαζόμασταν μια τέτοια για κάθε μπάλα, θα είχαμε πολλές τέτοιες λίστες μέσα σε μια άλλη (αυτό δεν είναι κάτι καινούριο, θυμηθείτε το έχουμε κάνει στο Adventure). Π.χ.:

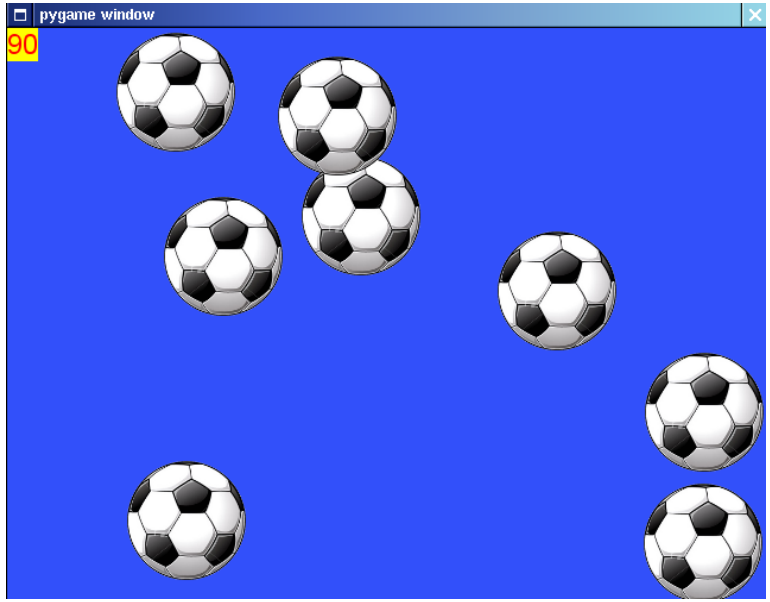
```
balls = [ [100.0, 100.0, 50.0,50.0],
          [50.0, 50.0, 30.0, 30.0], ...]
```

Πράγματι έτσι το πρόγραμμα μας θα γίνονταν αμέσως πολύ καλύτερο: Αντί να γράφουμε συνέχεια τις ίδιες εντολές με άλλα ονόματα μεταβλητών, θα είχαμε ένα βρόχο `for` που θα διέτρεχε τη λίστα, και θα εκτελούσε τις εντολές σε κάθε στοιχείο χωριστά. Μια χαρά.

Μια χαρά, μόνο που δεν είμαστε στα 80s! Είπαμε να εμπνευστούμε από τα προγράμματα του TI-99/4A, όχι να τα ξαναγράψουμε στη BASIC της εποχής! Ο κόσμος μας φωνάζει απεγνωσμένα ότι θέλει να γίνει object oriented!

Τι είναι μια μπάλα σαν αντικείμενο; Δείτε:

```
1 import pygame
2 from pygame.locals import *
3 import random
4 from sys import exit
```



Εικόνα 5.1: Το αντικειμενοστραφές Bouncing Ball δεν έχει κανένα πρόβλημα στο πόσες μπάλες θα δείξει! (από την άλλη ίσως έχει πρόβλημα το σαράβαλο που χρησιμοποιείτε για υπολογιστή. Χμμμ…) Μάλιστα το μόνο που χρειάζεται είναι να αλλάξετε ένα αριθμό στον κώδικα. Ποιο;

```
5  class Ball:
6    def __init__(self,theImage,x,y,xspeed,yspeed):
7      self.ballimage = theImage
8      self.x = x
9      self.y = y
10     self.xspeed = xspeed
11     self.yspeed = yspeed
12     self.shape = pygame.image.load(theImage)
13
14   def Show(self,surface):
15     surface.blit(self.shape, (self.x, self.y))
```

```
16  def GetWidth(self):
17    return self.shape.get_width()
18
19  def GetHeight(self):
20    return self.shape.get_height()
21
22  def Move(self, time):
23    distance_x = time * self.xspeed
24    distance_y = time * self.yspeed
25    self.x = self.x + distance_x
26    self.y = self.y + distance_y
27
28  def IsOutofX(self,xmin,xmax):
29    if (self.x >= (xmax - self.GetWidth()) or self.x <= xmin):
30      return True
31    else:
32      return False
33
34  def IsOutofY(self,ymin,ymax):
35    if (self.y >= (ymax - self.GetHeight()) or self.y <= ymin):
36      return True
37    else:
38      return False
```

Μη φωνάζετε! Είναι απλούστατο: Η μπάλα για τον εαυτό της κρατάει τις πληροφορίες που είπαμε πριν: Θέση (`x` και `y`), ταχύτητα (`xspeed` και `yspeed`) και την εικόνα (hint: όχι μόνο μπορείτε να φτιάξετε πολλαπλά bouncing balls, αλλά κάθε μπάλα να έχει και δική της μορφή). Δείτε λίγο τον constructor, τη συνάρτηση δημιουργίας ενός αντικειμένου τύπου μπάλας:

```
def __init__(self,theImage,x,y,xspeed,yspeed):
```

Ο συγκεκριμένος constructor παίρνει και κάποιες αρχικές τιμές οι οποίες αποδίδονται απευθείας στις μεταβλητές ιδιοτήτων της μπάλας μας. Τυπικά, για να δημιουργήσουμε μια μπάλα στο κύριο πρόγραμμα μας, θα γράψουμε κάτι σαν το παρακάτω:

```
MyBall = Ball(”soccer.png”,100.0,100.0,50.0,50.0)
```

Φυσικά εννοείτε ότι αντί για απευθείας (literal) τιμές, θα μπορούσαμε εδώ να έχουμε μεταβλητές. Αλλά για να δούμε τι άλλες συναρτήσεις περιέχει η κλάση μας – με λίγα λόγια τι ξέρει να κάνει η μπάλα μας από μόνη της:

```
def Show(self,surface):
```

Να δείξει τον εαυτό της σε μια επιφάνεια που θα της δώσουμε (με τη μέθοδο blit φυσικά)

```
def GetWidth(self):
def GetHeight(self):
```

Να μας πει το μέγεθος της (πλάτος και ύψος) σε pixels χρησιμοποιώντας το `get_width` και `get_height` του pygame.

```
def Move(self,time):
```

Να κινηθεί. Πολύ λογικό: γνωρίζει τόσο την ταχύτητα της — όπως τη δώσαμε στον constructor — όσο και τις προηγούμενες συντεταγμένες της. Οπότε το μόνο που χρειάζεται για τον υπολογισμό είναι και ο χρόνος που περνάμε ως παράμετρο από το κύριο πρόγραμμα.

```
def IsOutofX(self, xmin, xmax):
def IsOutofY(self, ymin, ymax):
```

Να ελέγξει αν έχει βγει εκτός ορίων της οθόνης, σε οποιοδήποτε από τους δύο άξονες. Το μόνο έξτρα που χρειάζεται είναι φυσικά τα ίδια τα όρια, τα οποία και πάλι τα περνάμε ως παραμέτρους.

Από εκεί και πέρα το πρόγραμμα είναι πολύ απλό. Θα διαπιστώσετε ότι χρησιμοποιούμε μια λίστα που αποθηκεύει… μπάλες. Στην κυριολεξία, η λίστα αυτή περιέχει μέσα αντικείμενα τύπου μπάλας, τα οποία προσθέτουμε με τον τρόπο που φαίνεται παρακάτω:

```
balls=[]
for i in range(0,8):
  balls.append(Ball(ballimage,x,y,xspeed,yspeed))
```

Βλέπετε βέβαια την απλή εκδοχή, γιατί μέσα στο βρόχο υπάρχουν και εντολές που δημιουργούν για κάθε μπάλα νέες τυχαίες τιμές θέσης και ταχύτητας. Δείτε και το υπόλοιπο πρόγραμμα και θα το καταλάβετε αμέσως.

```
def getQuit():
  for event in pygame.event.get():
    if event.type == QUIT:
      return True
  return False

def main():
  pygame.init()
  ballimage = 'soccer-ball.png'
  balls=[]
  for i in range(0,8):
    x = random.randint(80,500)
    y = random.randint(80,400)
    xspeed = 0
    while (xspeed >= -5 and xspeed <= 5):
      xspeed = random.randint(-50,50)
    yspeed = 0
```

```
    while (yspeed >= -5 and yspeed <=5):
      yspeed = random.randint(-50,50)
    balls.append(Ball(ballimage,x,y,xspeed,yspeed))
windowsize = (640,480)
surfacecolor = (50,80,250)
screen = pygame.display.set_mode(windowsize, DOUBLEBUF)
clock = pygame.time.Clock()

# Uncomment the framerate line and change
# time = clock.tick() in main loop to
# time = clock.tick(framerate)
# to limit the animation to a specific framerate

#framerate = 30

textfont = pygame.font.SysFont("Arial",24)
#
# Main loop
#

endprogram = False
while not endprogram:
  screen.fill(surfacecolor)
  for theball in balls:
    theball.Show(screen)
  time = clock.tick()
  thetext = textfont.render(str(1000/time), True, (255,0,0),(255,255,0))
  screen.blit(thetext,(0,0))
  time = time / 1000.0
```

```
47      for theball in balls:
48        theball.Move(time)
49        if theball.IsOutofX(0,640):
50          theball.xspeed = -theball.xspeed
51        if theball.IsOutofY(0,480):
52          theball.yspeed = -theball.yspeed
53      pygame.display.update()
54      endprogram = getQuit()
55
56    pygame.quit()
57    exit()
58
59  if __name__ == "__main__":
60    main()
```

Για να αντιστρέψουμε την ταχύτητα της μπάλας όταν βγει εκτός ορίων, κάνουμε κάτι σαν το παρακάτω:

```
        if theball.IsOutofX(0,640):
          theball.xspeed = -theball.xspeed
```

Αυτό μπορούμε να το κάνουμε, γιατί οι μεταβλητές `self` της μπάλας είναι προσβάσιμες και στο κύριο πρόγραμμα μας με το όνομα του αντικειμένου από μπροστά. Αλλά θα μπορούσατε επίσης να δημιουργήσετε συναρτήσεις του τύπου:

```
def GetXSpeed(self):
def GetYSpeed(self):
def SetXSpeed(self, xspeed):
def SetYSpeed(self, yspeed):
```

Θέλετε να το δοκιμάσετε; Δεν θα σας πούμε όχι :) Το πλήρες πρόγραμμα μπορείτε να το βρείτε επίσης στο παράρτημα, σελ. 173.

Κεφάλαιο 6

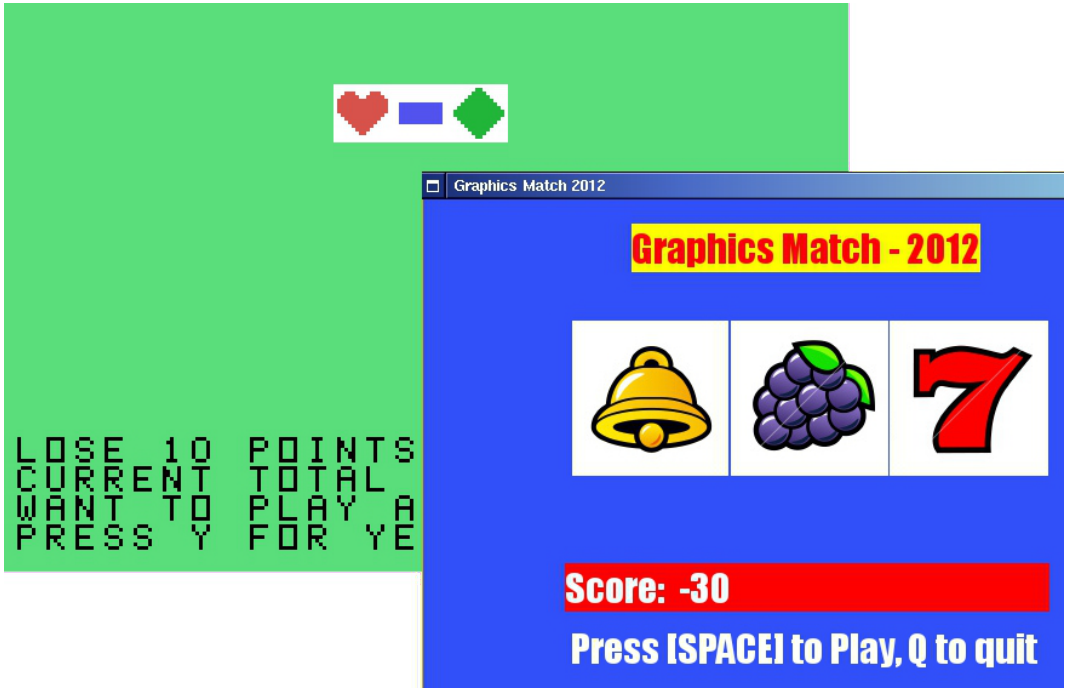
Graphics Match, το Πρώτο μας Γραφικό Παιχνίδι

6.1 Εισαγωγή

Υπομείνατε παιχνίδια με αριθμούς και text adventures, αντέξατε να μάθετε τόσα στοιχεία και εντολές της python. Μόνο στο προηγούμενο κεφάλαιο φάνηκε μια μικρή αχτίδα φωτός: σας δείξαμε επιτέλους γραφικό προγραμματισμό και pygame! Χρειάστηκε βέβαια να μάθετε πολλά νέα πράγματα – και ελπίζουμε να έχετε κατανοήσει τα περισσότερα. Ας συνοψίσουμε λίγο κάποια βασικά:

- Τα παιχνίδια σε pygame είναι *event driven*, οδηγούνται δηλ. από συμβάντα. Συμβάν είναι όταν πιέσετε το close. Συμβάν είναι όταν το πρόγραμμα σας πρέπει να ανταποκριθεί σε κάποιο πλήκτρο που πίεσε ο χρήστης.
- Το παιχνίδι αποτελείται πάντα από ένα βασικό βρόχο που εκτελείται συνέχεια μέχρι να το τερματίσουμε. Μέσα σε αυτό το βρόχο δημιουργούμε αρχικά το κάθε καρέ μας σε μια προσωρινή μνήμη και στη συνέχεια (με μια μόνο εντολή) το μεταφέρουμε στην ορατή οθόνη.
- Για να γίνει η μεταφορά χρειαζόμαστε την εντολή `pygame.display.update()` Βασικές έννοιες στην δημιουργία του κειμένου ή των γραφικών μέσα στο παράθυρο μας είναι η *επιφάνεια (surface)* και η μέθοδος `blit`.
- Κάθε φορά που εκτελείται ο κύριος βρόχος εμφανίζεται ένα καρέ. Όσο πιο γρήγορα γίνεται η εκτέλεση τόσα περισσότερα καρέ το δευτερόλεπτο θα δείχνουμε – εκτός αν τα περιορίσουμε εμείς.

Στο Κεφάλαιο 5 συνειδητοποιήσαμε ότι η python είναι αντικειμενοστραφής γλώσσα και βολεύει για παιχνίδια.



Εικόνα 6.1: Το παλιό και το καινούριο Graphics Match. Φαίνεται ξεκάθαρα ότι τα γραφικά του 1984 αποτελούνται από χαρακτήρες που έχουμε επανακαθορίσει. Το άλλο που φαίνεται ξεκάθαρα είναι ότι… χάνουμε! Καλύτερα να επενδύσετε σε γραμμάτια του Ελληνικού Δημοσίου παρά να προσπαθείτε να κερδίσετε το Graphics Match!

Η υπομονή σας επιτέλους θα ανταμειφθεί: θα φτιάξουμε το πρώτο μας κανονικό γραφικό παιχνίδι! Ένα πρόγραμμα που θα συνδυάζει πολλά από αυτά που μάθαμε μέχρι τώρα. Η υλοποίηση βέβαια είναι το τελευταίο μόνο κομμάτι του puzzle – το βασικό είναι να σκεφτούμε τον αλγόριθμο και τα δεδομένα μας.

6.2 Graphics Match – Η Σύγχρονη Εκδοχή

Το πρόγραμμα που θα φτιάξουμε ονομάζεται Graphics Match και είναι η σύγχρονη εκδοχή του προγράμματος για TI-99/4A που υπάρχει από το 1981! Από τη φωτογραφία καταλαβαίνετε ότι πρόκειται για ένα απλό παιχνίδι slot machine με φρουτάκια (και μπάρες, καμπάνες κλπ). Μόλις μάλιστα το τρέξετε θα αρχίσετε να αμφιβάλλετε για την έννοια των “τυχερών” παιχνιδιών. Μάλλον “άτυχα” θα έπρεπε να τα λένε. Ναι, ακόμα και στην python εκδοχή τους, χάνει κανείς συνέχεια!

6.2.1 Ιστορία και Θεωρία

Το “ταίριασμα των γραφικών” πρωτοεμφανίστηκε στο βιβλίο User’s Reference Guide του TI-99/4A και ίσως να είναι το πρώτο παιχνίδι που (αντ)έγραψα ποτέ! Σαν λογική δεν είναι ιδιαίτερα δύσκολο και σύντομα έφτιαξα αρκετά πιο πολύπλοκα παιχνίδια μόνος μου. Μου έδωσε όμως μερικές καλές ιδέες και με δίδαξε κάποια πράγματα που πρέπει να αποφεύγω.

Την εποχή του πρωτότυπου Graphics Match δεν υπήρχε Internet για να κατεβάσει κανείς εικόνες και γραφικά. Έπρεπε όλα να τα δημιουργούμε μόνοι μας, στο χέρι ή με κάποιο δικό μας πρόγραμμα. Για να καταλάβετε, τα γραφικά ορίζονταν σαν χαρακτήρες. Καθένα από τα φρουτάκια αποτελούνταν από τέσσερις χαρακτήρες σε μια μήτρα 2X2. Είχαμε τη δυνατότητα να αλλάζουμε τη μορφή των χαρακτήρων, κάτι σαν σημερινή σχεδίαση γραμματοσειρών αλλά σε πολύ απλή μορφή.

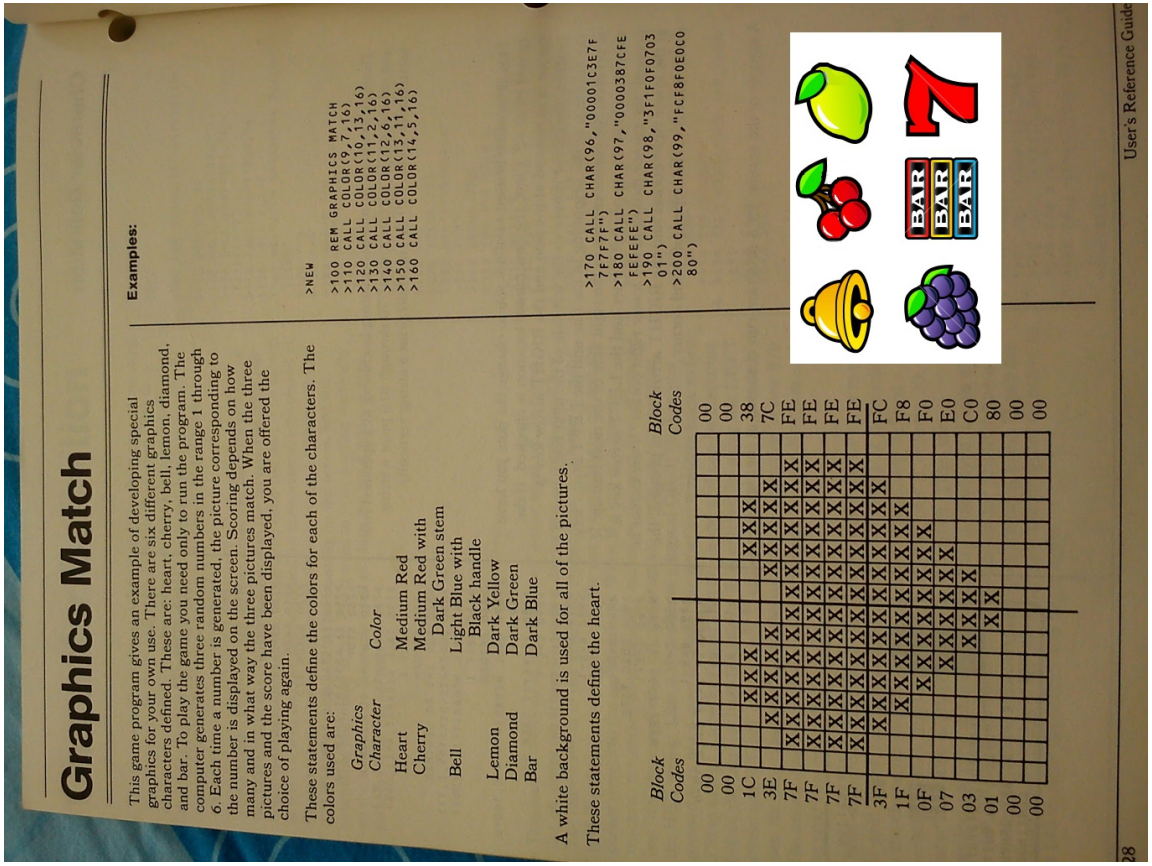
Το παιχνίδι μας χρησιμοποιεί έξι σύμβολα συνολικά.

Αριθμός	**Σύμβολο**
0	Λεμόνι
1	Μπάρα
2	Κεράσι
3	Καμπανάκι
4	Βατόμουρο
5	Το 7 (μη με ρωτήσετε γιατί)

Πίνακας 6.1: Τα γραφικά του παιχνιδιού

Το gameplay έχει ως εξής: Με την εκκίνηση του παιχνιδιού, τρία σύμβολα εναλλάσσονται γρήγορα σε μια γραμμή. Οι εναλλαγές σταματάνε μετά από 50 φορές και ανάλογα με τα σύμβολα που φαίνονται στη γραμμή υπολογίζεται ένα score. Ο τρόπος υπολογισμού είναι ο παρακάτω:

- Τρία όμοια σύμβολα σε μια γραμμή – Jackpot! Κερδίζετε 75 βαθμούς.
- Τα δύο πρώτα σύμβολα όμοια: Αν είναι λεμόνια, μπάρες ή κεράσια κερδίζετε 40 βαθμούς, αν είναι οτιδήποτε άλλο 10 βαθμούς.
- Το πρώτο και το τρίτο σύμβολο όμοια: κερδίζετε 10 βαθμούς.
- Το πρώτο και το δεύτερο όμοια ή τίποτα όμοιο, χάνετε 10 βαθμούς.



Εικόνα 6.2: Το αρχικό πρόγραμμα Graphics Match, εμφανίζεται σε TI BASIC στο User's Reference Guide του TI-99/4A το 1981. Ένα από τα πρώτα παιχνίδια που θα δοκίμαζε κανείς να γράψει μόλις αγόραζε αυτό τον υπολογιστή. Στο listing φαίνεται καθαρά πως ορίζονται οι τέσσερις χαρακτήρες που αποτελούν κάθε σύμβολο. Και ναι, όπως βλέπετε το gimp των 80s απαιτεί τη γνώση δεκαεξαδικού συστήματος! Συγκρίνετε το με τις έτοιμες εικόνες που χρησιμοποιούμε στο δικό μας πρόγραμμα. **Σημείωση:** Γυρίστε το βιβλίο στο πλάι :)

Φαντάζομαι μαντέψατε ποια περίπτωση είναι αυτή που τυχαίνει περισσότερο ε… :)

Τώρα, σε αντίθεση με το αρχικό πρόγραμμα του 1981, δεν χρειάζεται πλέον (ευτυχώς!) να σχεδιάσουμε εμείς τα λεμόνια, τα κεράσια τα καμπανάκια και όλα αυτά τα αστεία σύμβολα γενικά. Γιατί αρκεί να βρούμε αντίστοιχες εικόνες σε κατάλληλο μέγεθος από το Internet. Και δεν ξέρω αν το έχετε παρατηρήσει, αλλά υπάρχουν αρκετά "Internet καζίνο" από τα οποία μπορούμε να πάρουμε τα γραφικά! Πιστέψτε με, το μόνο πράγμα που θα πάρετε ποτέ από αυτά τα sites είναι αυτά τα γραφικά. Αν περιμένετε λεφτά, σωθήκατε.

Το καλό με τις έτοιμες εικόνες είναι ότι έχουν όλες το ίδιο στυλ και μέγεθος αφού προορίζονται για το συγκεκριμένο παιχνίδι – και αυτό μας βολεύει αφάνταστα. No photoshopping (ή… gimping) required!

Μόλις εμφανιστεί το score, ο χρήστης πιέζει το `SPACE` αν θέλει να ξαναπαίξει ή το `Q` για να τερματίσει το παιχνίδι. Το παιχνίδι φυσικά τερματίζεται και αν ο χρήστης κλείσει το παράθυρο από το close. Ξέχασα να σας πω ότι αρχικά ξεκινάτε με μηδέν βαθμούς και είναι πολύ πιθανό όταν το βαρεθείτε να φύγετε με… -1000. Αν παίξετε πολύ ώρα μπορεί να χρειαστείτε και τη βοήθεια του ΔΝΤ για να… ξελασπώσετε :)

6.2.2 Ο Αλγόριθμος

Ξεχάστε και την python και το pygame για την ώρα. Ας επικεντρωθούμε στο πως λειτουργεί αυτό το παιχνίδι και τι θα περιέχει ο βασικός βρόχος σε γενική μορφή.

Μια και έχουμε ήδη κάνει μια δυναμική εκκίνηση στον αντικειμενοστραφή προγραμματισμό ας το σκεφτούμε λίγο με αυτή τη λογική. Το πως θα την υλοποιήσουμε είναι μια άλλη ιστορία!

Φανταστείτε λοιπόν ότι έχετε δημιουργήσει μια κλάση για τη γραμμή με τα τρία περιστρεφόμενα φρουτάκια. Τι θα έπρεπε να περιέχει; Τουλάχιστον τρία πράγματα:

- Μια συνάρτηση constructor που να κατασκευάζει ένα αντικείμενο αυτού του είδους. Ο constructor είναι μια καλή ευκαιρία να φορτώσουμε και τις εικόνες.
- Μια μέθοδο `Spin` που να κάνει τα φρουτάκια να εναλλάσσονται πάνω σε μια επιφάνεια σε συγκεκριμένες συντεταγμένες που θα δώσουμε και ενδεχομένως με συγκεκριμένο framerate.
- Μια μέθοδο `GetScore` που θα την καλούμε για να μας επιστρέφει πόσο… χάσαμε.

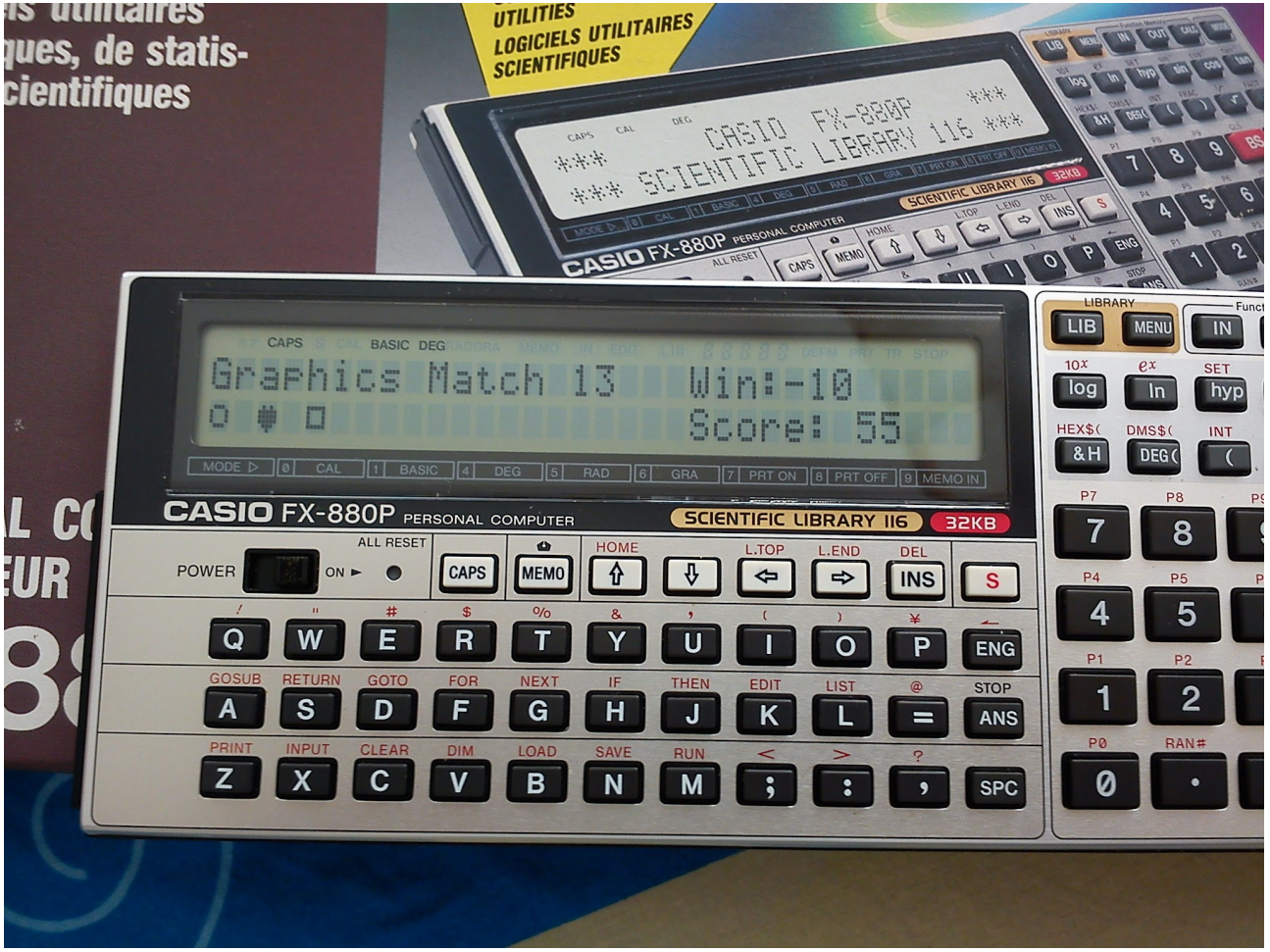
Προσέξτε το μαγικό τρικ τώρα: υποθέστε ότι όλα αυτά τα έχουμε φτιάξει και λειτουργούν! Ποιο θα ήταν το πρόγραμμα μας στον κύριο βρόχο σε γενικές γραμμές;

Ας υποθέσουμε ότι έχουμε ονομάσει την κλάση `Spinner` και θέλουμε να δημιουργήσουμε ένα αντικείμενο αυτής της κλάσης. Αυτό θα γίνονταν κάπως έτσι:

```
slotmachine = Spinner()
```

Έτσι θα κληθεί ο constructor του `Spinner` και θα δημιουργηθεί το αντικείμενο `slotmachine`. Βέβαια στο συγκεκριμένο constructor θέλουμε να δώσουμε και κάτι ακόμα (τις εικόνες) αλλά αυτό είναι μια τεχνική λεπτομέρεια που θα φροντίσουμε αργότερα. Ο κύριος βρόχος, σε μορφή — περίπου — ψευδοκώδικα θα είναι κάτι σαν το παρακάτω:

```
score = 0
endgame = False
spacepressed = True
while not endgame:

  Διάβασε πληκτρολόγιο και συμβάντα
  Αν ο χρήστης πίεσε το Q ή το close, endgame = True
  Αν ο χρήστης πίεσε SPACE, spacepressed = True

  # Κάνε τα φρουτάκια να γυρίζουν αν έχει πατηθεί το SPACE
  if spacepressed:
    slotmachine.Spin()

    # Πάρε τους βαθμούς
    points = slotmachine.GetScore()

    # Άθροισε τους στο score
    score = score + points

  Χρησιμοποίησε μια επιφάνεια για να δείξεις το score

  pygame.display.update()
```



Εικόνα 6.3: Ποιος σας είπε ότι το παιχνίδι περιορίζεται μόνο στον TI και στην python που έχετε μπροστά σας; Ο pocket υπολογιστής που βλέπετε μπροστά σας, το διάσημο Casio FX-880P (από τους καλύτερους BASIC programmable που έβγαλε ποτέ η CASIO) έχει εξαιρετική γλώσσα προγραμματισμού κατάλληλη για να γράψουμε ακόμα και παιχνίδια. Και μην ανησυχείτε καθόλου: χάνει κανείς με την ίδια συχνότητα και σε αυτό το σύστημα!

```
  spacepressed = False
```

```
Έξοδος
```

Δεν είναι πολύ απλό; Βέβαια το θέμα είναι ότι **κάποιος** πρέπει να γράψει αυτό το `Spinner` class και τις μεθόδους του. Αλλά τα περιεχόμενα των συναρτήσεων του δεν είναι κάτι που δεν έχουμε μάθει.

6.2.3 Υλοποίηση του Spinner

Τα βασικά συστατικά για το `Spinner` έχουν ως εξής:

1. Θα χρειαστούμε την `randint` να μας δίνει τυχαία νούμερα που θα αντιστοιχούν στις εικόνες που θα εμφανιστούν.
2. Θα χρειαστεί να κάνουμε `blit` τρεις εικόνες, η μια δίπλα στην άλλη. Με δεδομένη μια θέση εκκίνησης (στήλη) X για την πρώτη εικόνα, η επόμενη θα εμφανιστεί στη θέση `X + Πλάτος_Εικόνας + Κάποιο_Περιθώριο`. Για να δώσουμε την αίσθηση κίνησης, θα επαναλάβουμε αυτή τη διαδικασία μερικές φορές απεικονίζοντας αυτές τις τριάδες τη μια πάνω στην άλλη.
3. Ο υπολογισμός του score είναι μια απλή διαδικασία – απλά πρέπει να δούμε και να συγκρίνουμε μεταξύ τους τα αποτελέσματα της τελευταίας `randint`.

Ας δούμε με βάση τα παραπάνω, ποιες θα είναι οι μεταβλητές στο class που θα περιέχουν τα δεδομένα του αντικειμένου μας. Είναι όλες αυτές οι μεταβλητές που ξεκινάνε με `self.όνομα_μεταβλητής` (*data attributes* στην ορολογία της python). Τονίζουμε ότι κάθε τέτοια μεταβλητή είναι προσπελάσιμη από οποιαδήποτε συνάρτηση (μέθοδο) μέσα στην κλάση.

- Το αντικείμενο μας χρειάζεται να ξέρει τις εικόνες του. Καθώς είναι έξι σε αριθμό, μας βολεύει να τις αποθηκεύσουμε σε μια δομή σαν λίστα.
- Θα πρέπει να αποθηκεύσουμε επίσης τους τρεις αριθμούς που αντιπροσωπεύουν το τρέχον αποτέλεσμα της περιστροφής. Και εδώ βολεύει να χρησιμοποιήσουμε μια λίστα ή ένα tuple.

Ας δούμε λοιπόν το `Spinner` class. Ξεκινάμε από τον constructor:

```
class Spinner:
  def __init__(self,images):
    self.slot=[]
    for image in images:
      self.slot.append(pygame.image.load(image))
```

Σε σχέση με τα παραδείγματα που δείξαμε πριν, ο constructor μας έχει μια μικρή διαφορά: παίρνει και την παράμετρο `images` που περιέχει τα ονόματα των αρχείων εικόνων που θα χρησιμοποιηθούν. Το αντικείμενο μας στο κύριο πρόγραμμα αρχικοποιείται κάπως έτσι:

```
slot_images =   ('lemon.jpg','bar.jpg','cherry.jpg',
                 'bell.jpg','raspberry.jpg','seven.jpg')
slotmachine = Spinner(slot_images)
```

Το `self.slot` θα δημιουργηθεί ως μια λίστα αντικειμένων τύπου image, έτοιμη για χρήση από το pygame. Θυμηθείτε ότι το `append` είναι μια εντολή για να προσθέτουμε στοιχεία στο τέλος μιας λίστας. Έτσι το `slot[0]` θα είναι ένα λεμόνι, το `slot[1]` μια μπάρα κ.ο.κ. Βολικό, γιατί έτσι ξέρουμε ότι θα χρειαστούμε τυχαίους αριθμούς από το 0 ως το 5.

Ας δούμε τώρα και τη συνάρτηση `Spin`:

```
  def Spin(self, surface, x, y, framerate, clock):
    for draws in range(0,50):
      self.luck = (randint(0,5),randint(0,5), randint(0,5))
      x1=x
      for i in self.luck:
        surface.blit(self.slot[i], (x1, y))
        x1 = x1 + self.slot[i].get_width() + 3
      pygame.display.update()
      time = clock.tick(framerate)
```

Σκοπός της `Spin` είναι φυσικά να απεικονίσει το αποτέλεσμα στην οθόνη μας. Άρα θα πρέπει να γνωρίζει μια επιφάνεια που όμως την έχουμε ορίσει στο κύριο πρόγραμμα και την περνάμε ως παράμετρο. Για τον ίδιο λόγο περνάμε ως παραμέτρους και το `framerate` και `clock`.

Τα πράγματα μετά είναι απλά:

- Παράγουμε τριάδες τυχαίων αριθμών που αποθηκεύονται στο `self.luck` Τοποθετούμε με την `blit` τις αντίστοιχες εικόνες, τη μια δίπλα στην άλλη, με ένα διάκενο 3 pixels
- Το κάνουμε αυτό 50 φορές διαδοχικά για να κρατήσουμε το χρήστη σε αγωνία για μερικά δευτερόλεπτα, μέχρι να μάθει πόσο… έχασε.

Το `x` και `y` περιέχουν φυσικά τη στήλη και τη γραμμή στην οποία θα εμφανιστεί το αντικείμενο μας στην οθόνη. Ας δούμε νοητά μια εκτέλεση:

```
    self.luck = (randint(0,5),randint(0,5), randint(0,5))
```

Ας υποθέσουμε ότι αυτό δημιούργησε τις τιμές (1,3,5)

```
    x1 = x
```

Για να μη πειράξουμε την αρχική στήλη, την αποθηκεύουμε σε μια μεταβλητή η οποία θα αυξάνεται για να απεικονίζεται η μια εικόνα δίπλα στην άλλη.

```
    for i in self.luck:
```

Θα δημιουργήσει ένα βρόχο όπου το i θα διατρέξει τις τιμές (1,3,5).

```
      surface.blit(self.slot[i], (x1, y))
```

Θα τοποθετήσει την εικόνα slot[1] (μπάρα) στην επιφάνεια.

```
        x1 = x1 + self.slot[i].get_width() + 3
```

Η επόμενη εικόνα θα εμφανιστεί τρία pixels μετά το τέλος της προηγούμενης. Καθώς καταλαβαίνετε η επόμενη εικόνα θα είναι το `slot[3]` (καμπανάκι) και η τελευταία το `slot[5]` (το 7). Και φυσικά… χάσατε. Όχι, όχι ακόμα. Γιατί όλο αυτό θα επαναληφθεί 50 φορές, καθώς περικλείεται μέσα στο:

```
    for draws in range(0,50):
```

Φυσικά, το `x1` δεν αυξάνεται για πάντα: δεν θέλουμε να βγούμε έξω από την οθόνη! Όταν δείξουμε μια τριάδα εικόνων, οι επόμενες σχεδιάζονται από πάνω. Οπότε πρέπει να ξεκινήσουμε πάλι το `x1` από την αρχική στήλη όπως μας δόθηκε από τον κύριο βρόχο:

```
      x1 = x
```

Δεν σχολιάζω καθόλου εντολές όπως το `pygame.display.update` και το `time=clock.tick(framerate)` γιατί τις ξέρετε.

Είναι επίσης αυτονόητο πως λειτουργεί η `GetScore`, αρκεί να δείτε τους κανόνες βαθμολογίας στον πίνακα 6.1.

6.2.4 Events και Πληκτρολόγιο

Το κομμάτι στον κύριο βρόχο που διαπραγματεύεται τα συμβάντα και την ανάγνωση του πληκτρολογίου είναι το παρακάτω:

```
for event in pygame.event.get():
  if event.type == QUIT:
    endgame = True
  if event.type == KEYDOWN:
    keyboardinput = event.key
    if keyboardinput == K_q:
      endgame = True
    if keyboardinput == K_SPACE:
      spacepressed = True
```

Έχουμε εξηγήσει ήδη στην ενότητα 4.4 (σελίδα 56) πως λειτουργεί το σύστημα αποστολής μηνυμάτων από το λειτουργικό στην εφαρμογή μας. Όταν πιέσουμε το close (X) στο παράθυρο, αποστέλλεται το μήνυμα `QUIT`. Τι γίνεται όμως όταν πιέζουμε κάποιο πλήκτρο;

Αυτό αντιστοιχεί σε ένα event τύπου `KEYDOWN`. Αν διαπιστώσουμε ότι έχουμε τέτοιο event, μπορούμε να πάρουμε το πλήκτρο που πιέστηκε με τη εντολή:

```
    keyboardinput = event.key
```

Ευτυχώς για μας, το `pygame.locals` δίνει συμβολικά ονόματα σε κάθε πλήκτρο και δεν χρειάζεται να θυμόμαστε περίεργους κωδικούς. Έτσι για να δούμε αν ο χρήστης πίεσε το q, ελέγχουμε για το `K_q` και για το SPACE ελέγχουμε το `K_SPACE`. Εξαιρετικά απλό, αποτελεσματικό και φυσικά απαραίτητο για το επόμενο μας παιχνίδι (έκπληξη).

Μην περιμένετε όμως να σας δώσω επόμενο παιχνίδι, αν δεν μου κάνετε τις παρακάτω ασκήσεις (αρκετά χαλαρώσατε παίζοντας με μπαλίτσες και Hello World pygame):

- Προσθέστε το (κρυφό) πλήκτρο C στο παιχνίδι για cheat! Όταν το πιέζετε, αν το σκορ σας είναι αρνητικό θα το μετατρέπει σε θετικό! Π.χ. το -150 θα γίνετε 150. Αν είναι θετικό δεν θα κάνει τίποτα. Μάλλον είναι και ο μόνος τρόπος για να κερδίσει κανείς το Graphics Match!

- Φτιάξτε μια γραμμή μηνυμάτων πάνω από τη γραμμή του Score που να δείχνει πόσο έχασε ή κέρδισε ο χρήστης σε κάθε spin. Π.χ. Win 10 points, Lose 10 points κλπ. Bonus points αν το κάνετε να πανηγυρίζει για το Jackpot.
- Το ότι το αρχικό Graphics Match χρησιμοποιεί τρία slots δεν μας εμποδίζει στη σύγχρονη εκδοχή να φτιάξουμε ένα με 4 ή 5. Απλά θα πρέπει να αποφασίσετε νέους κανόνες για το ποιο συνδυασμοί κερδίζουν. Φτιάξτε το όπως το θέλετε!
- Στο πρόγραμμα μας η `Spin` αναλαμβάνει η ίδια να σχεδιάσει διαδοχικά 50 φορές τη γραμμή οπότε αναγκαστικά χρειάζεται να καλέσει την `pygame.display.update()` και την `clock.tick(framerate)` για να τα δείξει. Ίσως θα ήταν σκόπιμο αυτά να γίνονται στο κύριο πρόγραμμα. Μετατρέψτε την ώστε κάθε φορά που καλείται να παράγει μόνο ένα αποτέλεσμα και μεταφέρετε το update και το βρόχο των 50 επαναλήψεων στο κύριο πρόγραμμα.
- Διαβάστε στο `http://pygame.org` πως μπορείτε να βάλετε ήχο. Το αρχικό Graphics Match είχε ήχο!
- Ψάξτε για βελτιώσεις στον κώδικα και ξαναγράψτε τον σύμφωνα με το δικό σας στυλ. Σε κάθε προγραμματιστικό πρόβλημα υπάρχουν περισσότερες από μία υλοποιήσεις.

Και τώρα που εξασφάλισα ότι θα ξενυχτήσετε για τον επόμενο μήνα, σας αφήνω να μελετήσετε και να πειραματιστείτε με την ησυχία σας. Από το επόμενο κεφάλαιο θα ξεκινήσουμε να φτιάχνουμε σταδιακά το τελικό πρόγραμμα αυτού του βιβλίου.

Μπορείτε να βρείτε όλο το παιχνίδι στο παράρτημα, σελ. 177. Κατεβάστε το παιχνίδι από εδώ: `http://www.freebsdworld.gr/files/graphics-match.zip`

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)
```

Εικόνα 7.1: Το περίπτερο μας στην έκθεση. Ο original TI-99/4A με το cartridge TI Invaders αλλά και με δικά μου παιχνίδια σε BASIC και Extended BASIC διασκέδασε πολύ κόσμο. Τελικά το κλασικό δεν πεθαίνει ποτέ (και ειδικά το συγκεκριμένο μηχάνημα είναι φτιαγμένο να αντέξει για πάντα όπως φαίνεται). Δεξιά φαίνεται στον video-προβολέα το Pygame Invaders το οποίο φυσικά ήταν το hit της έκθεσης!

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.

Εικόνα 7.2: Ο μαθητής μας Ανδρέας Κοντορίνης παρουσιάζει το pygame στην Έκθεση Μαθητικής Δημιουργίας Πληροφορικής στα Χανιά. Ο Ανδρέας υπήρξε βασικό πειραματόζ.. ε… beta tester για τη σειρά μαθημάτων Python Game Programming.

7.5 Η Υπερκλάση Craft

Σε μια πρώτη σκέψη μπορούμε να γράψουμε την παρακάτω:

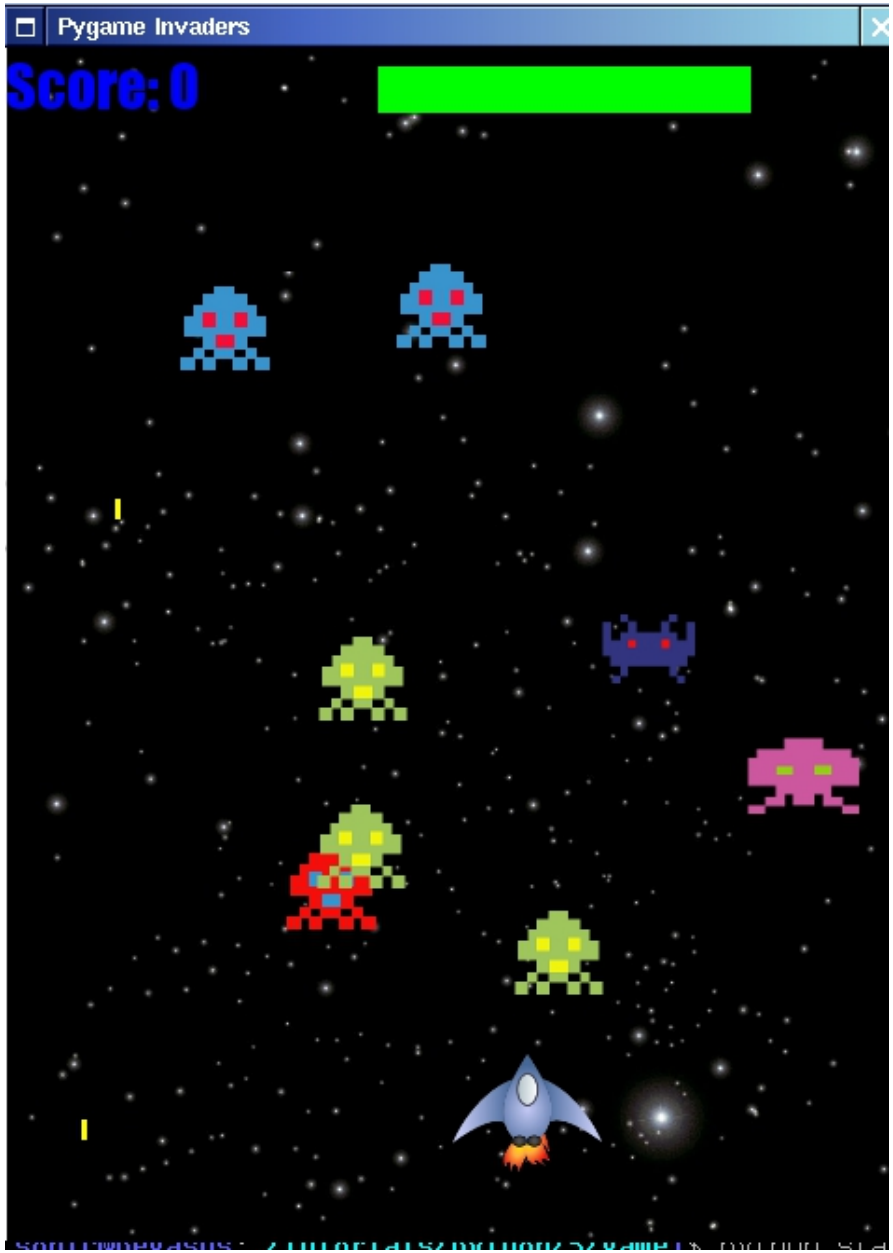
```
class Craft(object):
  def __init__ (self, imagefile, coord):
    self.shape = pygame.image.load(imagefile)
    self.ship_width = self.shape.get_width()
    self.ship_height = self.shape.get_height()
    self.rect = pygame.Rect(coord,(self.ship_width, self.ship_height))

  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)
```

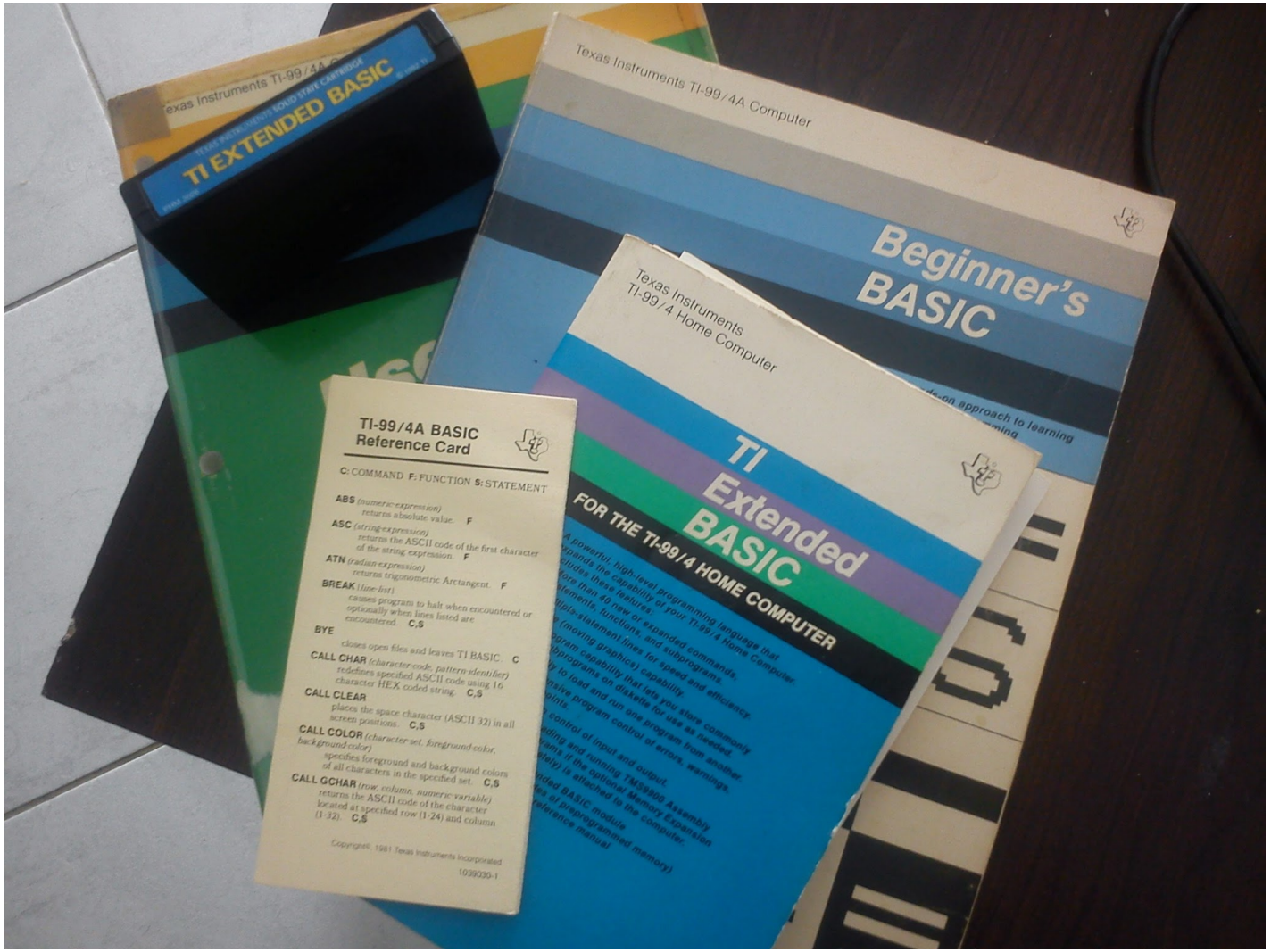
και αρκεί να την καλέσουμε οπουδήποτε λίγο πριν το `while`:

```
  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;



Εικόνα 7.4: Game programming στα 80s. Δεν υπήρχε το web άρα ξεχάστε το `http://pygame.org`. Ότι πληροφορίες μπορούσε κανείς να βρει για τη γλώσσα περιέχονταν στα εγχειρίδια του υπολογιστή του. Εδώ βλέπετε τα βασικά βιβλία του TI-99/4A καθώς και το Extended BASIC cartridge που κόστιζε μια μικρή περιουσία. Τουλάχιστον έβγαλε τα λεφτά του (δουλεύει ακόμα…)

- 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()
```

Κεφάλαιο 8

Pygame Invaders: Ready, set, fire!

8.1 Εισαγωγή

Στο προηγούμενο κεφάλαιο ξεκινήσαμε να γράφουμε το… όνειρο κάθε bedroom programmer των 80s: ένα γρήγορο διαστημικό space shooter. Ειδικά η λέξη “γρήγορο” τις περισσότερες φορές εκείνη την εποχή σήμαινε ότι θα χρησιμοποιούσαμε assembly για την εκάστοτε πλατφόρμα μας ή ότι ήμασταν από τους τυχερούς που διέθετε μηχάνημα με hardware sprites όπως η extended BASIC του TI-99 ή ο Commodore 64.

Για τους υπόλοιπους που έγραφαν σε BASIC και περιορίζονταν σε κίνηση χαρακτήρων στην οθόνη, το Space Invaders ξεκίναγε φιλόδοξα, αλλά γρήγορα κατέληγε σε… Space Invader αφού ζήτημα είναι να κατάφερνε να κινήσει παραπάνω από ένα εξωγήινο κάθε φορά και ταυτόχρονα να χρειάζεται να απεικονίζει και να υπολογίζει τόσο τις δικές μας κινήσεις όσο και τις βολές – φιλικές και εχθρικές. Και δεν έχουμε ακόμα φτάσει στα ηχητικά εφέ και την ανίχνευση συγκρούσεων.

Ευτυχώς για μας, όχι μόνο τα μηχανήματα μας είναι τόσο πολύ ταχύτερα αλλά διαθέτουμε και μια γλώσσα προγραμματισμού που μπορεί να υλοποιήσει εύκολα και καθαρά ότι σκεφτούμε.

Μέχρι τώρα έχουμε δει:

- Τις βασικές κλάσεις για το διαστημόπλοιο μας και τους εξωγήινους
- Κατεύθυνση του διαστημοπλοίου με τα βελάκια

- Μια εντελώς βασική κλάση για το φόντο
- Ήχο και μουσική στη βασική τους μορφή

Έχοντας φορτίσει τις προγραμματιστικές μας μπαταρίες (και όχι, στα 80s δεν συνηθίζαμε τις πίτσες και κόλες για αυτό το σκοπό) αισθανόμαστε έτοιμοι να αντιμετωπίσουμε τις επόμενες προκλήσεις του Pygame Invaders.

8.2 Κυλιόμενο Φόντο

Μια απίστευτη πρόκληση για τα 80s, το κυλιόμενο φόντο στο δικό μας παιχνίδι δεν είναι κάτι τόσο δύσκολο πλέον. Πως όμως θα δημιουργήσουμε μια κυλιόμενη εικόνα από μια στατική, και μάλιστα όταν το μέγεθος της στατικής είναι όσο το παράθυρο μας;

Φανταστείτε πολύ απλά, ότι παίρνετε την εικόνα του φόντου και σε κάθε καρέ τη μετατοπίζετε προς τα κάτω – με μια συγκεκριμένη ταχύτητα φυσικά. Ας ξεκινήσουμε λοιπόν με αυτή την απλή εκδοχή να γράψουμε την `Scroll`, την οποία την είχαμε αφήσει με ένα μοναχικό `pass`:

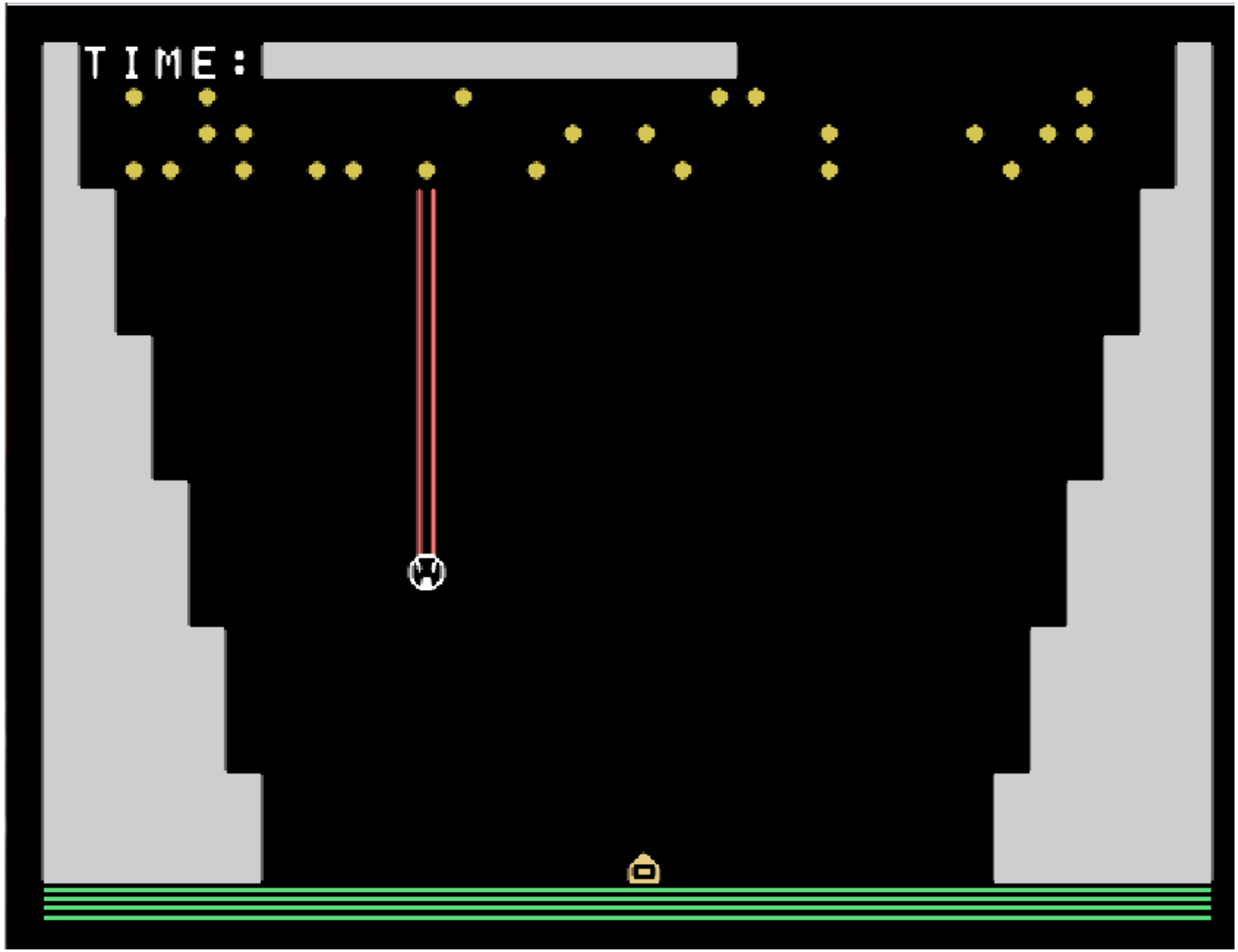
```
def Scroll(self, speed_y, time):
  distance_y = speed_y * time
  self.coord[1] += distance_y
```

Στο κύριο πρόγραμμα μας, θα προσθέσουμε φυσικά και την αντίστοιχη γραμμή πάνω από το `StarField.Show(screen)`:

```
  StarField.Scroll(backspeed, time)
```

Αρκεί να ορίσουμε κάπου και την επιθυμητή ταχύτητα, `backspeed` σε μια γραμμή που θα βάλουμε οπουδήποτε έξω από το βασικό βρόχο `while` του παιχνιδιού μας:

```
# Set the background scrolling speed

backspeed = 100
```



Εικόνα 8.1: Ο πρόγονος του προγράμματος που φτιάχνουμε εδώ θα μπορούσε να είναι το invaders που έγραψα το 1986 στον TI-99. Invaders βέβαια μόνο κατά όνομα, μιας και μετά βίας έβγαζε… ένα εξωγήινο κάθε φορά. Λογικό όμως: Laser, διαστημόπλοιο, εξωγήινος, κίνηση, ήχος και score σε… 3Mhz είναι απλά θαύμα. Και βέβαια είναι γραμμένο σε BASIC, με κίνηση χαρακτήρων, χωρίς καν sprites… Give me a break!

Αυτό είναι! Ας το τρέξουμε τώρα, και καλωσορίσατε στο…

8.2.1 Python Debugging!

Όπως έχουμε ήδη πει, όχι μόνο θα αλλάξουμε τον κώδικα μας αρκετές φορές μέχρι να πετύχουμε το αποτέλεσμα που θέλουμε, αλλά θα πρέπει να διορθώσουμε και τα προβλήματα του, ακόμα περισσότερες! Δεν υπάρχει κώδικας που είναι αλάνθαστος από την πρώτη φορά.

Αν δοκιμάσατε να τρέξετε το προηγούμενο, θα είδατε:

```
Traceback (most recent call last):
  File ”pygame-invaders.py”, line 126, in <module>
    StarField.Scroll(backspeed, time)
  File ”pygame-invaders.py”, line 58, in Scroll
    self.coord[1] += distance_y
TypeError: ’tuple’ object does not support item assignment
```

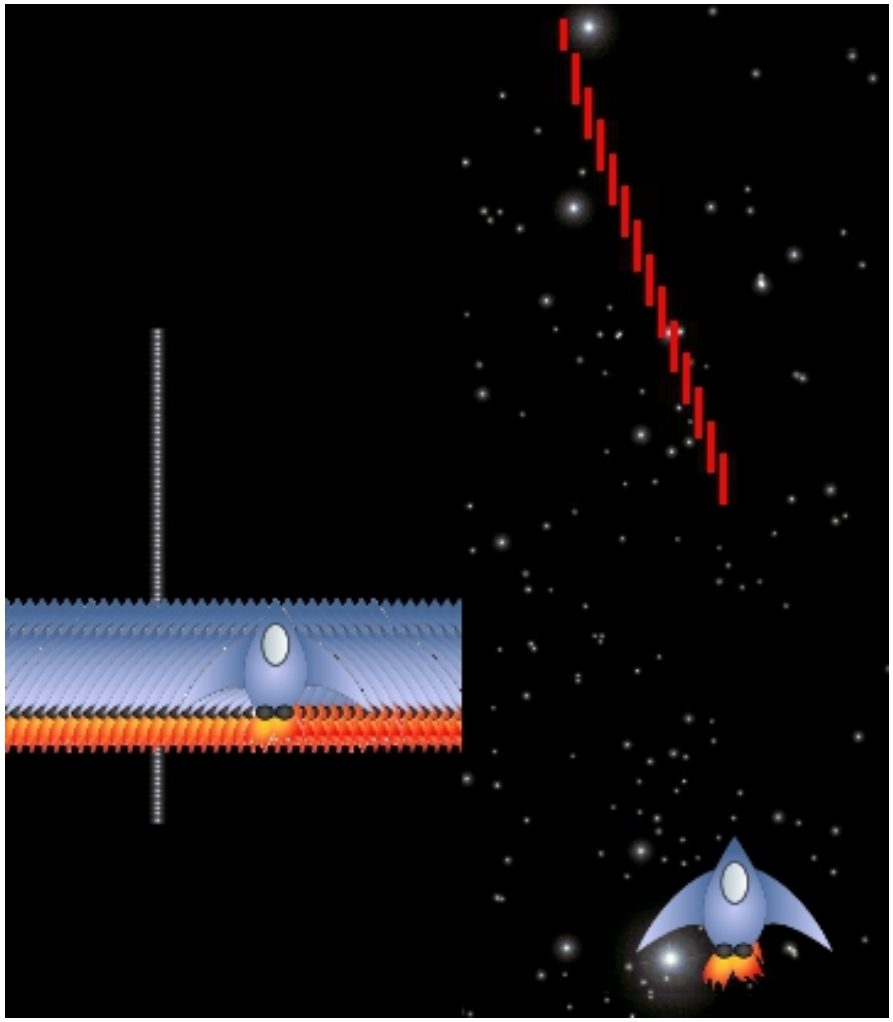
Με απλά λόγια: προσπαθήσαμε να αλλάξουμε τη μεταβλητή `self.coord`, μόνο που αυτή την είχαμε ορίσει στο βασικό μας πρόγραμμα ως tuple. Θυμηθείτε, τα tuples σε αντίθεση με τις λίστες δεν αλλάζουν (*immutable objects* τα ονομάζει η python). Ας το αλλάξουμε αυτό λοιπόν. Βρείτε τη γραμμή:

```
  StarField = SpaceBackground((0,0), ”stars.jpg”)
```

και αλλάξτε τη σε:

```
  StarField = SpaceBackground([0,0], ”stars.jpg”)
```

Αλλάζοντας τις παρενθέσεις σε αγκύλες, μετατρέψαμε το tuple σε λίστα. Μπορούμε τώρα να τρέξουμε ξανά το πρόγραμμα.



Εικόνα 8.2: Μην αφήσετε κανένα προγραμματιστή να σας πείσει ότι ο κώδικας του είναι τέλειος ή — ακόμα χειρότερα — ότι δουλεύει με την πρώτη. Αριστερά, μείναμε από… φόντο καθώς τα αστέρια κύλησαν κάτω από την οθόνη πριν γράψουμε τη ρουτίνα που κάνει την εικόνα… ρολό. Αναμέναμε ότι θα δούμε κάτι περίεργο βέβαια. Δεξιά, το διαστημόπλοιο μας ρίχνει συνεχόμενες, κολλημένες μεταξύ τους βολές. Η αρχική μας σχεδίαση είχε κάποια… εχμ… μικροπροβλήματα.

8.3 Scrolling Background, Απόπειρα II

Μη φωνάξετε “χάλια” μόλις δείτε το αποτέλεσμα… Περιμένατε μήπως δια μαγείας το φόντο να επαναλαμβάνεται; Απλά στην τωρινή εκδοχή το μόνο που γίνεται είναι να σκρολλάρει και τελικά να εξαφανίζεται στο κάτω μέρος του παραθύρου. Ακόμα πιο ενοχλητική είναι η γραμμή που μένει από εκείνο το μοναχικό αστεράκι – για να μην πούμε ότι η κίνηση του διαστημοπλοίου καταστρέφεται μόλις εξαφανιστεί το φόντο: είναι λογικό, καθώς το φόντο ουσιαστικά “σβήνει” το προηγούμενο καρέ πριν εμφανιστεί το επόμενο. Όταν αυτό δεν γίνεται πλέον, το ένα καρέ πέφτει επάνω στο άλλο, με το σουρεαλιστικό αποτέλεσμα που βλέπετε στην εικόνα 8.2. Ακόμα και ο James T. Kirk δεν είχε φανταστεί τέτοια κατάληξη για το USS Enterprise!

Εύκολο να το φτιάξουμε όμως: τι θέλουμε στην πραγματικότητα; Να κολλήσουμε δύο εικόνες φόντου, ώστε να φτιάξουμε ένα “ρολό”.

- Η πρώτη εικόνα απεικονίζεται στο [0,0] και η δεύτερη ακριβώς πάνω από αυτή – η δεύτερη δηλ. θα τελειώνει στο [0,0] και αρχικά θα είναι αόρατη!
- Η κίνηση γίνεται προς τα κάτω και για τις δύο, ταυτόχρονα.
- Όταν η πρώτη εικόνα φτάσει στο κάτω άκρο του παραθύρου, η δεύτερη πλέον εικόνα είναι σε πλήρη εμφάνιση στο παράθυρο μας.
- Η διαδικασία επαναλαμβάνεται από την αρχή και κανείς δεν καταλαβαίνει ότι είναι το ίδιο φόντο!

8.4 Scrolling Background, Απόπειρα III

Ας ξαναδούμε λοιπόν την κλάση για το background, σύμφωνα με τα παραπάνω, με σχόλια ανάμεσα στον κώδικα:

```
class SpaceBackground:
  def __init__(self, coord, coord2, imagefile):
```

Προσθέσαμε μια λίστα `coord2` που θα περιέχει τις συντεταγμένες της δεύτερης εικόνας. Αρχικά θα έχει τιμή `[0, -screenheight]`, όπως θα ορίσουμε κατά τη δημιουργία του αντικειμένου και θα είναι αόρατη.

```
    self.shape = pygame.image.load(imagefile)
    self.coord = coord
    self.coord2 = coord2
    self.y_original = coord[1]
    self.y2_original = coord2[1]
```

Αποθηκεύουμε τις αρχικές συντεταγμένες γραμμής και για τις δύο εικόνες. Η κίνηση των εικόνων αλλάζει ακριβώς αυτές τις τιμές και μόλις ολοκληρωθεί ένα πλήρες scroll πρέπει να μπορούμε να επανέλθουμε σε αυτές.

```
  def Show(self, surface):
    surface.blit(self.shape, self.coord)
    surface.blit(self.shape, self.coord2)
```

Προσθέσαμε ακόμα μια γραμμή που κάνει blit και την δεύτερη εικόνα. Τίποτα το ιδιαίτερο.

```
  def Scroll(self, speed_y, time):
    distance_y = speed_y * time
    self.coord[1] += distance_y
    self.coord2[1] += distance_y
```

Παρόμοια, εδώ προσθέσαμε μια γραμμή για να κινείται και η δεύτερη εικόνα. Αλλά όλο το παιχνίδι παίζεται εδώ:

```
    if self.coord2[1] >= 0:
      self.coord[1] = self.y_original
      self.coord2[1] = self.y2_original
```

Μόλις η δεύτερη εικόνα εμφανιστεί πλήρως στην οθόνη (φτάσει δηλ. στο [0,0] ή λόγω… κεκτημένης ταχύτητας το ξεπεράσει!), επανερχόμαστε στις αρχικές συντεταγμένες και για τις δύο εικόνες. Τώρα αν αυτό δεν είναι καμπύλωση του χωροχρόνου, δεν ξέρω τι είναι! Τρέμε

Αϊνστάιν!

Στο κύριο πρόγραμμα μας, η δημιουργία του αντικειμένου για το φόντο γίνεται πλέον με τη γραμμή:

```
StarField = SpaceBackground([0,0],[0, -screenheight], "stars.jpg")
```

8.5 Fire the Lasers!

Έχοντας επιτέλους τελειώσει με το φόντο, ήρθε η ώρα να διαλύσουμε τους εξωγήινους με το Laser μας. Το γεγονός ότι το παιχνίδι μας δεν δείχνει ακόμα εξωγήινους, είναι μια μικρή ενοχλητική λεπτομέρεια. Τι θέλετε δηλαδή, να φτιάξουμε πρώτα τους εξωγήινους και να μην έχουμε όπλα; Θα φεύγατε για πόλεμο χωρίς δοκιμαστικές βολές και εκπαίδευση;

Τώρα λοιπόν που λογικευτήκατε, ας δούμε τι θα πρέπει να περιέχει μια κλάση για το Laser:

- Κλασικά, ένα constructor. Θα ορίζει και μερικές βασικές παραμέτρους: αρχικές συντεταγμένες, χρώμα, μήκος της γραμμής, ταχύτητα.
- `Show`: Για να εμφανιστεί η βολή στην επιφάνεια της επιλογής μας
- Μια συνάρτηση `Move` για την κίνηση της βολής.

Σε μια πρώτη σκέψη, ίσως τα παραπάνω σας φαίνονται αρκετά. Να σας ρωτήσω όμως: Πότε εξαφανίζεται μια βολή; Σε δύο περιπτώσεις:

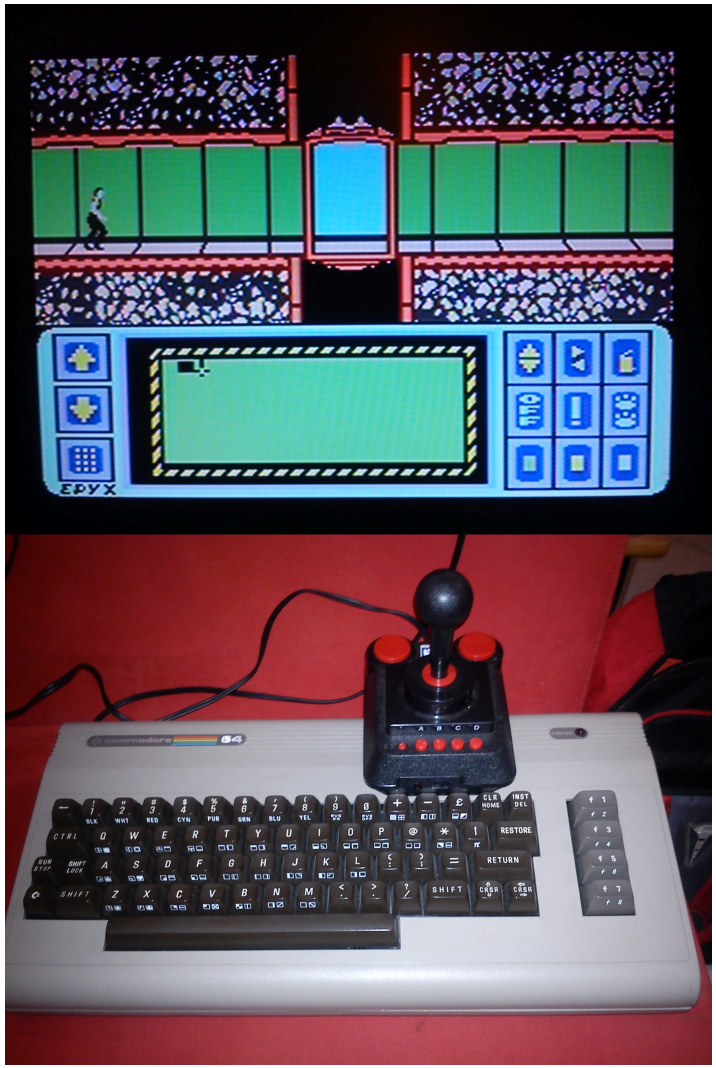
1. Όταν χτυπήσει τον επάρατο εχθρό μας
2. Όταν βγει έξω από την οθόνη μας

Δεν έχουμε ακόμα εχθρούς, αλλά χρειάζεται να κάνουμε κάτι για το (2).

Χρειαζόμαστε ακόμα μια συνάρτηση που να δείχνει αν η βολή έχει φύγει πάνω από την οθόνη. Προφανώς, αρκεί να επιστρέφει `True` ή `False`. Ας κάνουμε λοιπόν μια πρώτη απόπειρα.

8.6 Laser Class

Σχετικά εύκολα μπορούμε να υλοποιήσουμε την κλάση που περιγράψαμε. Ας δούμε τον κώδικα μας με τα σχετικά σχόλια:



Εικόνα 8.3: Το μηχάνημα είναι ο γνωστός Commodore 64, ο υπολογιστής με τις περισσότερες πωλήσεις όλων των εποχών! Αν νομίζετε όμως ότι το joystick που φαίνεται προορίζεται να συνδεθεί πάνω του, χάσατε! Δεν είναι ένα απλό joystick, είναι το C64DTV: περιέχει μέσα του ένα ολόκληρο C64 (OK, χωρίς το πληκτρολόγιο) και κάμποσα παιχνίδια (από αυτά που φορτώναμε τότε με τις ώρες) σε μια ROM. Λειτουργεί με μπαταρίες και συνδέεται απευθείας στην τηλεόραση σας. Ναι, όλος ο C64 με σημερινή τεχνολογία χωράει σε 2 τσιπάκια στο κάτω μέρος ενός joystick! Εκπληκτικό – αλλά εμείς βέβαια προτιμάμε τη μαγεία του αρχικού. Στην πάνω εικόνα το διάσημο παιχνίδι Mission Impossible του C64 από το C64DTV.

```
class Laser:
  def __init__(self, coord, color, size, speed):
    self.x1 = coord[0]
    self.y1 = coord[1]
    self.size = size
    self.color = color
    self.speed = speed
```

Ο constructor δίνει τις αρχικές συντεταγμένες (`coord`) οι οποίες μπορεί να είναι tuple ή λίστα, το χρώμα (σε μορφή λίστας RGB), το μέγεθος (`size`) και την ταχύτητα κίνησης (`speed`). Οι συντεταγμένες ανατίθενται απευθείας σε χωριστές μεταβλητές `x1` και `y1` για να τις επεξεργαζόμαστε εύκολα σε όλες τις υπόλοιπες συναρτήσεις.

```
  def Show(self, surface):
    pygame.draw.line(surface, self.color, (self.x1,self.y1),(self.x1,self.y1-self.size),3)
```

Η συνάρτηση `Show` είναι απλή: χρησιμοποιούμε την `draw` του pygame για να φτιάξουμε μια γραμμή στο κατάλληλο μέγεθος και χρώμα. Οι συντεταγμένες δίνονται με τη μορφή δύο tuples (αρχής και τέλους γραμμής) και καθώς βλέπετε το μόνο που κάνουμε είναι να αλλάζουμε το `y1` για το κατάλληλο μήκος γραμμής. Το… μυστηριώδες 3 στο τέλος είναι το πλάτος της γραμμής, τρία pixels. Αν θέλαμε να είμαστε σωστότεροι θα το ορίζαμε στον constructor (της νύχτας τον κώδικα τον βλέπει η μέρα και γελά).

```
  def Move(self, time):
    distance = self.speed * time
    self.y1 += distance
```

Δεν νομίζουμε ότι η `Move` χρειάζεται πρακτικά κάποια εξήγηση!

```
  def GoneAbove(self,y):
    if self.y1<=y:
      return True
    else:
      return False
```

Η συνάρτηση `GoneAbove` θα μας λέει αν η βολή έχει πάει πάνω από κάποιο όριο της οθόνης – αν έχει βγει από την οθόνη θα πρέπει να την σβήσουμε.

Πιστεύετε ότι έχουμε κάνει καλή δουλειά και το Laser μας θα λειτουργήσει με την πρώτη. Πόσο γελασμένοι είστε. Ευτυχώς που δεν σας έβαλα ακόμα εξωγήινους, θα σας είχαν φάει λάχανο! Αλλά ας μείνουμε λίγο ακόμα στην άγνοια μας, και πάμε να δούμε πως θα ενσωματώσουμε αυτό το — θεός να το κάνει — όπλο στο κύριο πρόγραμμα μας.

8.7 Συνάρτηση Fire στο Craft class

Η βολή είναι προφανώς μια δυνατότητα στο `Craft` class. Αν το σκεφτείτε, δεν χρειάζεται να φτιάξουμε διαφορετικό Laser class για τους εξωγήινους, άρα μας βολεύει να φτιάξουμε την μέθοδο `Fire` στο superclass. Θα πρέπει όμως να προσθέσουμε κάποια πράγματα στον constructor του `Craft`:

```
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))
    self.ship_midwidth = self.ship_width / 2
    self.firecolor=(255,0,0)
    self.firespeed = -800
    self.shotlength = 20
```

Προσθέσαμε τις τέσσερις τελευταίες γραμμές. Το `midwidth` είναι ακριβώς αυτό που λέει: το μισό πλάτος του διαστημοπλοίου. Καθώς φαντάζεστε, το Laser θα πρέπει να φεύγει από τη μύτη του διαστημοπλοίου μας που βρίσκεται στη μέση! Το χρώμα της βολής θα είναι κόκκινο (255,0,0) και η ταχύτητα -800. Μη σας παραξενεύει η αρνητική τιμή: το Laser κινείται από κάτω προς τα πάνω! Και πάμε να δούμε τη συνάρτηση `Fire`:

```
  def Fire(self):
    shot = Laser((self.rect[0]+self.ship_midwidth, self.rect[1]),
                 self.firecolor,self.shotlength,self.firespeed)
    return shot
```

Με το `midwidth` εξασφαλίζουμε ότι η βολή θα βγει από τη μέση του διαστημοπλοίου, τη μύτη του! Η συνάρτηση επιστρέφει ένα αντικείμενο τύπου… Laser. Ο λόγος θα γίνει προφανής σε λίγο.

8.7.1 Πρώτη Απόπειρα

Δείχνοντας υπερβολική εμπιστοσύνη στον εαυτό μας, είμαστε έτοιμοι για την πρώτη δοκιμή. Μένουν μόνο κάποιες γραμμές στο κύριο πρόγραμμα.

```
        if key[K_SPACE]:
          firelist.append(SpaceShip.Fire())
```

Φαντάζεστε βέβαια που βρίσκεται αυτή η γραμμή, αλλά τι είναι το `firelist`; Μα το διαστημόπλοιο μας μπορεί να ρίξει πολλές βολές. Δεν χρειάζεται να εξαφανιστεί η προηγούμενη για να ρίξουμε νέα! Πως λοιπόν θα κρατήσουμε λογαριασμό για τις βολές που πρέπει να σχεδιάσουμε στην επιφάνεια μας; Κάθε φορά που ρίχνουμε βολή το αντικείμενο που επιστρέφει η `Fire` αποθηκεύεται στην λίστα `firelist`. Προφανώς την έχουμε αρχικοποιήσει έξω από το βρόχο `while` με την εντολή:

```
  firelist = []
```

Είναι έπειτα πολύ εύκολο να την κινήσουμε και να την δείξουμε, προσθέτοντας τις παρακάτω γραμμές μετά το `SpaceShip.Show(screen)`:

```
    for theshot in firelist:
      theshot.Move(time)
```

Για κάθε βολή που έχουμε προσθέσει στη λίστα, την κινούμε.

```
      theshot.Show(screen)
```

Την δείχνουμε.

```
      if theshot.GoneAbove(0):
        firelist.remove(theshot)
```

Ερευνούμε αν έχει βγει πάνω από την οθόνη, και αν συμβαίνει αυτό την αφαιρούμε από τη λίστα.

Τρέξτε το λοιπόν και δείτε το αποτέλεσμα. Αν δεν θέλετε να ντροπιαστείτε στο Υπεργαλακτικό Συνέδριο Κβαντικής Τεχνολογίας, δείτε απλώς την εικόνα 8.2. Τι είναι αυτές οι συνεχόμενες γραμμές; Εμείς για Laser ξεκινήσαμε και μάλιστα με μήκος 20 pixels. Αλλά αν πιέζετε το SPACE

(το πλήκτρο ενεργοποίησης) συνέχεια θα έχετε μια… συνεχόμενη γραμμή προς τα πάνω (θυμίζει λίγο το Parsec, το παιχνίδι του TI-99 που φαίνεται στην εικόνα). Μη ξεχνάτε έχουμε ρυθμίσει το πληκτρολόγιο στη μέγιστη επανάληψη. Τι μπορούμε να κάνουμε;

8.7.2 Δεύτερη Απόπειρα

Εντάξει, για πρώτη απόπειρα δεν τα πήγαμε τόσο άσχημα, το φιάσκο με το φόντο ήταν σαφώς μεγαλύτερο. Χρειάστηκε να σκεφτούμε πολλά πράγματα και μας διέφυγε μόνο ένα:

- Το διαστημόπλοιο δεν πρέπει να ρίχνει νέα βολή αν η προηγούμενη δεν έχει “ταξιδέψει” μια ελάχιστη απόσταση από το αρχικό σημείο.

Και πως θα γίνει αυτό; Μα φυσικά θα φτιάξουμε μια συνάρτηση που θα δίνει την απόσταση της βολής από ένα σημείο αναφοράς (τη γραμμή που βρίσκεται το σκάφος μας τη στιγμή που ρίχνει τη βολή). Στον constructor του `Laser` θα έχουμε μια επιπλέον παράμετρο για την γραμμή αναφοράς:

```
class Laser:
  def __init__(self, coord, color, size, speed, refline):
```

και θα αποθηκεύσουμε προφανώς αυτή την τιμή:

```
    self.refline = refline
```

και η συνάρτηση μας θα είναι:

```
  def DistanceTravelled(self):
    return self.refline - self.y1
```

Η μέθοδος `Fire` θα πρέπει να ορίζει το `refline`, ως την τρέχουσα γραμμή του διαστημοπλοίου μας:

```
  def Fire(self):
    shot = Laser((self.rect[0]+self.ship_midwidth, self.rect[1]),
                 self.firecolor,self.shotlength,self.firespeed,self.rect[1])
    return shot
```

και τέλος η ανίχνευση πληκτρολογίου αλλάζει ως εξής:

```
        if key[K_SPACE]:
          if firelist:
            # Only fire if last shot has travelled a minimum distance
            if firelist[-1].DistanceTravelled() >=  150:
              firelist.append(SpaceShip.Fire())
          else:
            # or if there is no shot
            firelist.append(SpaceShip.Fire())
```

Απλά, το διαστημόπλοιο δεν ρίχνει αν η τελευταία βολή (το `firelist[-1]` πολύ βολικά μας επιστρέφει το τελευταίο στοιχείο της λίστας) δεν έχει ταξιδέψει τουλάχιστον 150 pixels από τη γραμμή βολής. Αν βέβαια δεν υπάρχει καμιά βολή (περίπτωση `else`), το διαστημόπλοιο ρίχνει αμέσως. Αλλάζοντας το 150 θα μπορείτε να ελέγξετε την “πυκνότητα” αν θέλετε των βολών (cheat mode on!)

8.8 Ήχος Βολής!

Χάρις στην `PrepareSound`, είναι εύκολο να κάνουμε το Laser μας… θορυβώδες. Τι είπατε, στο διάστημα δεν έχει αέρα και άρα δεν ακούγεται ήχος; Γιατί μήπως τα αστέρια είναι δύο εικόνες κολλημένες σε ρολό που γυρίζουν; Σας παρακαλώ, αφήστε με στην φαντασία μου! Σε ένα βολικό σημείο πριν το βασικό βρόχο γράφουμε:

```
  laser = PrepareSound("shoot.wav")
```

Εικόνα 8.4: Από τα πλέον διάσημα παιχνίδια σε cartridge του TI-99/4A, το Parsec είναι ένα διαστημικό παιχνίδι με side scrolling, σταθμούς ανεφοδιασμού, πολλαπλές πίστες και εκείνο το συνεχόμενο Laser που κάτι μας θυμίζει! Ένα ακόμα εκπληκτικό προσόν του ήταν η ομιλία – για όποιον τυχερό είχε το speech synthesizer. Για τους πραγματικά παλιούς από σας, το Parsec ήταν το επίσημο video game του εφηβικού τηλεπαιχνιδιού “Κόκκινοι γίγαντες, άσπροι νάνοι” που βλέπαμε μετά μανίας στην κρατική τηλεόραση!

Μπορούμε να προσθέσουμε τον ήχο απευθείας στη συνάρτηση `Fire` του `Craft` class:

```
    laser.play()
```

Το παιχνίδι μας επιτέλους αρχίζει να παίρνει μορφή! Έχουμε το κινούμενο φόντο, έχουμε το Laser, έχουμε κίνηση. Μένει να υποδεχτούμε μόνο τους (χαμένους από χέρι) εξωγήινους, το οποίο θα γίνει πολύ σύντομα.

8.9 Μερικές Απλές Βελτιώσεις

8.9.1 Απλούστευση του SpaceBackground

Η κλάση που έχουμε γράψει για το φόντο είναι πιο πολύπλοκη από όσο χρειάζεται! Αποφασίσαμε λοιπόν να την απλουστεύσουμε καθώς το φόντο πάντοτε ξεκινάει την κύλιση του από την αρχή του παραθύρου, το γνωστό [0,0] και καταλήγει επίσης στο `screenheight`:

```
class SpaceBackground:
  def __init__(self, screenheight, imagefile):
    self.shape = pygame.image.load(imagefile)
    self.coord = [0,0]
    self.coord2 = [0, -screenheight]
    self.y_original = self.coord[1]
    self.y2_original = self.coord2[1]
```

Και στο κύριο πρόγραμμα μας πλέον η μόνη παράμετρος που περνάμε όσο αφορά τις συντεταγμένες είναι το `screenheight`:

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

8.9.2 Βελτιστοποίηση Εμφάνισης της Βολής Laser

Παρατηρήσατε και εσείς φυσικά ότι η βολή του Laser δεν ξεκινάει ακριβώς από τη μύτη του διαστημοπλοίου, αλλά έχει offset λίγες γραμμές. Λογικό καθώς γύρω από το διαστημόπλοιο μας βρίσκεται ένα αόρατο κουτάκι που το περιέχει και η βολή εμφανίζεται από την πάνω γραμμή του – που δεν συμπίπτει αναγκαστικά με τη μύτη του κανονιού μας. Θα πρέπει να εμφανίσουμε το Laser από λίγο πιο κάτω.

Για να το κάνουμε αυτό με σωστό τρόπο™, θα δώσουμε μια επιπλέον παράμετρο `voffset` (vertical offset) στο Laser μας. Μη ξεχνάτε ότι θα το χρησιμοποιήσουμε και για τους εξωγήινους και εκεί τα πράγματα ίσως είναι διαφορετικά! Ο νέος μας constructor θα είναι κάπως έτσι:

```
class Laser:
  def __init__(self, coord, color, size, speed, refline, voffset):
    self.x1 = coord[0]
    self.y1 = coord[1] + voffset
    self.size = size
    self.color = color
    self.speed = speed
    self.refline = refline
```

Και φυσικά η κλήση που κάνουμε διαμορφώνεται κάπως έτσι:

```
    shot = Laser((self.rect[0]+self.ship_midwidth, self.rect[1]),
                 self.firecolor,self.shotlength,self.firespeed,self.rect[1],15)
```

Όπου φυσικά το μαγικό 15 είναι το offset που επέλεξα για το διαστημόπλοιο μας. Αν θέλουμε να είμαστε σωστοί βέβαια αυτό θα το βάλουμε σε μια μεταβλητή self στον constructor. Θα μου πείτε τώρα γιατί να μην κάνουμε κάτι αντίστοιχο και για το `midwidth`. Για σκεφτείτε το λίγο και ασχοληθείτε το απόγευμα, άντε μπράβο!

8.9.3 Δημιουργία Συνάρτησης Fire στο SpaceCraft Class

Σκοπός μας εδώ είναι να μεταφέρουμε τον ήχο από τη γονική συνάρτηση ώστε να εξασφαλίσουμε ότι μόνο το δικό μας (δυνατό) Laser θα ακούγεται (και όχι οι βολές των κακόμοιρων, αδύναμων και χαμένων από χέρι εξωγήινων). Και ωραία, πήγατε στο `Fire` του `Craft` class και βγάλατε το `laser.play()`. Ποια θα είναι η συνάρτηση στο δικό μας `SpaceCraft` class;

```
  def Fire(self):
    laser.play()
    return super(SpaceCraft,self).Fire()
```

Υπάρχει μόνο ένα πονηρό σημείο: Φυσικά και θα καλέσουμε με την `super` τη γονική συνάρτηση. Αλλά μη ξεχνάτε ότι αυτή επιστρέφει ένα αντικείμενο τύπου `Laser` το οποίο θα χρησιμοποιήσουμε αργότερα. Αν την καλέσετε απλώς έτσι:

```
    super(SpaceCraft,self).Fire()
```

η δική μας `Fire` θα επιστρέψει το… τίποτα. Θα προσθέσετε μετά ένα αντικείμενο του τύπου… τίποτα στη λίστα `firelist` που περιέχει τις βολές. Αυτό δεν είναι καθόλου μα καθόλου καλό, γιατί δεν υπάρχει συνάρτηση που να κινεί αντικείμενα τύπου “τίποτα”! Θα δείτε την python να παραπονιέται με το παρακάτω μήνυμα:

```
Traceback (most recent call last):
  File ”pygame-invaders.py”, line 210, in <module>
    theshot.Move(time)
AttributeError: ’NoneType’ object has no attribute ’Move’
```

Θα μπορούσατε φυσικά να καλέσετε την `Fire` και έτσι:

```
1   def Fire(self):
2     laser.play()
3     theshot = super(SpaceCraft,self).Fire()
4     return theshot
```

Αλλά είναι περιττό να αποθηκεύουμε την τιμή που δεν πρόκειται να χρησιμοποιήσουμε παρά μόνο για το `return` αμέσως μετά.

8.10 Έρχονται οι… Εξωγήινοι

Έχουμε κίνηση, Laser, scrolling background… Καιρός να βάλουμε και τους εξωγήινους στο παιχνίδι, δεν νομίζετε; Όχι τίποτα άλλο να έχουμε κάτι να πυροβολάμε άμεσα στο επόμενο κεφάλαιο. Πάμε λοιπόν να δούμε πως θα τους εμφανίσουμε.

Όπως είχαμε πει στην αρχική περιγραφή του παιχνιδιού, οι εξωγήινοι θα έρχονται σε κύματα. Καθώς ξεμπερδεύουμε με το ένα κύμα θα εμφανίζεται το επόμενο. Μια απλή σκέψη είναι φυσικά το κάθε κύμα να έχει ένα συγκεκριμένο αριθμό από εξωγήινους. Καθένας θα εμφανίζεται σε μια τυχαία θέση στην οθόνη μας και θα κινείται με σχετικά τυχαίες ταχύτητες, κάνοντας bounce στα όρια της οθόνης (σας θυμίζει κάτι;) Με βάση τα παραπάνω, μπορούμε να γράψουμε εύκολα μια αρχική υλοποίηση για την κλάση τους.

```
class Alien(Craft):
  def __init__(self, imagefile, coord, speed_x, speed_y):
    super(Alien, self).__init__(imagefile, coord)
    self.speed_x = speed_x
    self.speed_y = speed_y
    self.shot_height = 10
    self.firebaseline = self.ship_height
    self.firecolor=(255,255,0)
    self.firespeed = 200
  def Move(self, time):
    super(Alien,self).Move(self.speed_x, self.speed_y,time)
    if self.rect[0] &gt;= 440 or self.rect[0] &lt;= 10:
      self.speed_x = -self.speed_x
    if self.rect[1] &lt;= 10 or self.rect[1] &gt;= 440:
      self.speed_y = -self.speed_y
```

Τι παραπάνω έχει το `Alien` class από εμάς; Παρατηρήστε ότι ο constructor έχει από την αρχή τιμές για τις ταχύτητες του εξωγήινου και στους δύο άξονες – όπως είπαμε οι εξωγήινοι δεν είναι πολύ έξυπνοι. Ξεκινάνε με τυχαίες ταχύτητες τις οποίες θα παράγουμε στον κύριο βρόχο του

παιχνιδιού. Δείτε τι γίνεται στη συνάρτηση `Move`. Όταν φτάσουν σε κάποια όρια της οθόνης (τα οποία **κακώς** είναι hardwired στον κώδικα, κάτι που θα διορθώσετε) απλώς αλλάζουν φορά κίνησης (εμ, αυτό το bouncing ball…). Εμείς βέβαια θα κανονίσουμε να έρχονται γενικά προς το μέρος μας με τις αντίστοιχες εντολές στον κύριο βρόχο.

Πριν το `while` όμως, θα πρέπει να ετοιμάσουμε τις εικόνες που θα χρησιμοποιηθούν. Οι εξωγήινοι είναι πολλών ειδών:

```
alienimage=('alien1.png','alien2.png','alien3.png','alien4.png','alien5.png')
```

Ορίζουμε και μια μεταβλητή που μας επιτρέπει να καθορίσουμε πόσοι εξωγήινοι θα υπάρχουν σε κάθε κύμα:

```
numofaliens = 8
```

Όπως κάναμε με τις βολές Laser, θα αποθηκεύουμε τα αντικείμενα τύπου `Alien` που θα φτιάχνουμε σε μια λίστα, που αρχικά, έξω από το βρόχο, θα είναι κενή:

```
AlienShips = []
```

Μέσα στο βρόχο τώρα, θα χρησιμοποιήσουμε μια εντολή for για να δημιουργήσουμε το κύμα των χαζοεξωγήινων. Μπορείτε να βάλετε τις παρακάτω εντολές ακριβώς κάτω από την γραμμή `time` (προσέξτε το indentation):

```
1    if not AlienShips:
2      # AlienShips empty means we need to create new 'wave'
3      AlienShips = [ Alien(alienimage[randint(0,len(alienimage)-1)],
4                     [randint(20,screenwidth-80),randint(20,screenheight-140)],
5                     randint(100,150), randint(100,150))
6                     for i in range(0,numofaliens) ]
```

Αν ψάχνετε το `for`, είναι λίγο κρυμμένο: Χρησιμοποιήσαμε μια δυνατότητα της Python που ονομάζεται *list comprehension* και μας επιτρέπει πολύ εύκολα να δημιουργήσουμε μια λίστα ενσωματώνοντας την εντολή `for` με τον τρόπο που βλέπετε παραπάνω. Θα μπορούσαμε βέβαια

να το γράψουμε με τον κλασικό τρόπο:

```
      for i in range(0,numofaliens):
        Alienships.append(Alien(alienimage[randint(0,len(alienimage)-1)],
                          [randint(20,screenwidth-80),
                           randint(20,screenheight-140)],randint(100,150),
                           randint(100,150)))
```

Όπως παρατηρήσατε, χρησιμοποιούμε την `randint` για να παράγουμε αρχικές θέσεις και ταχύτητες για τους εξωγήινους. Οι τιμές έχουν επιλεγεί κατάλληλα για να μην πέφτουν εκτός οθόνης αλλά και να μη εμφανιστεί κανείς πάνω στο διαστημόπλοιο μας στο κάτω μέρος της οθόνης.

Γιατί υπάρχει το `if not AlienShips`; Αν η λίστα `AlienShips` είναι άδεια, μπορεί να συμβαίνουν δύο πράγματα:

- Είτε μόλις ξεκινήσαμε το παιχνίδι, οπότε αυτό είναι το πρώτο κύμα εξωγήινων
- Είτε μόλις έχουμε… ξεμπερδέψει με όλους τους εξωγήινους του προηγούμενου κύματος

Σε κάθε περίπτωση, σημαίνει ότι δεν έχουμε εξωγήινους στην οθόνη, άρα σίγουρα πρέπει να δημιουργήσουμε το νέο κύμα! Και ναι, αυτό είναι όλο το θέμα με την δημιουργία των εξωγήινων.

Φυσικά, οι εξωγήινοι κάπου πρέπει να φανούν και να κινηθούν. Οπότε θα έχουμε και το παρακάτω `for`. Προσέξτε ότι πρέπει να είναι μετά το `Show` για το κινούμενο φόντο μας:

```
    For AlienShip in AlienShips:
      AlienShip.Show(screen)
      AlienShip.Move(time)
```

Και τώρα μπορείτε να περάσετε μερικές ώρες ευχάριστα πυροβολώντας εξωγήινους που… δεν σκοτώνονται. Από την άλλη δεν σας πυροβολούν κιόλας. Χαρείτε αυτή την προσωρινή ανακωχή, γιατί στο επόμενο κεφάλαιο, θα διακοπεί απότομα!

Κεφάλαιο 9

Pygame Invaders: Η Εισβολή Αρχίζει!

9.1 Εισαγωγή

Καθώς φαίνεται τα ψέματα τελείωσαν. Έχουμε το σκάφος μας, το κινούμενο φόντο, το Laser και στο προηγούμενο κεφάλαιο μας ήρθαν μια… βόλτα και οι πρώτοι εξωγήινοι. Λίγο διστακτικοί εμείς, τους κοιτάξαμε, τους ρίξαμε και διαπιστώσαμε ότι δεν γίνεται τίποτα! Μα τι λείπει επιτέλους από αυτό το παιχνίδι;

Αν είχατε ξεκινήσει το programming παιχνιδιών πριν μερικά χρόνια — τότε που η δημιουργία frame based shooters ήταν απλώς άπιαστο όνειρο — και περιγράφατε το πρόβλημα σας στο φίλο σας που έγραφε εκείνη την εποχή το δικό του Invaders θα σας έλεγε δύο λέξεις: *collision detection*. Η *ανίχνευση συγκρούσεων* είναι ένα βασικό θέμα (και πρόβλημα) σε όλα τα παιχνίδια δράσης. Είναι το τμήμα του προγράμματος που αναγνωρίζει πότε ο εξωγήινος δέχτηκε τη βολή, πότε το διαστημόπλοιο έσκασε πάνω στον… κακόβουλα κινούμενο μετεωρίτη, πότε ο pacman έφαγε το φαντασματάκι.

Μια κακοσχεδιασμένη ρουτίνα ανίχνευσης συγκρούσεων οδηγεί σε τραγελαφικά αποτελέσματα:

- Χαρακτήρες που περνάνε μέσα από… τοίχους (μόνο ο κώδικας μας και τα φαντάσματα το καταφέρνουν αυτό)
- Βολές που ακουμπάνε — ή ακόμα χειρότερα περνάνε μέσα — από χαρακτήρες χωρίς να τους καταστρέφουν. Και καλά να είναι το δικό μας διαστημόπλοιο, τι γίνεται με τυχόν… απέθαντους εξωγήινους;

- Laser που καταστρέφουν πράγματα τα οποία δεν φαίνονται να έχουν αγγίξει ή — ακόμα χειρότερα — βρίσκονται στην… άλλη άκρη της οθόνης.

Σε ένα παιχνίδι που δεν έχει γραφτεί με τη frame based λογική που χρησιμοποιούμε εδώ, η ανίχνευση συγκρούσεων δεν είναι και το πιο εύκολο πράγμα: φανταστείτε ότι αν η γλώσσα παρέχει αυτόνομα κινούμενα sprites, το πρόγραμμα μας μπορεί να εκτελεί οποιαδήποτε τυχαία εντολή την ώρα που θα συγκρουστούν. Αν είμαστε τυχεροί, η εντολή που θα εκτελείται θα είναι αυτή που θα εντοπίζει τη σύγκρουση δύο sprites.

Για να αντιμετωπιστεί το πρόβλημα αυτό, σε κάποιες διαλέκτους BASIC της εποχής άρχισαν να εμφανίζονται εντολές με λογική events (συμβάντων): το πρόγραμμα εκτελούνταν κανονικά, η γλώσσα έλεγχε αυτόματα κατά τακτά διαστήματα για συγκρούσεις και έστελνε ένα event στο πρόγραμμα το οποίο μετά επεξεργαζόμασταν για να δούμε τι… σκοτώθηκε. Αυτά για τους λίγους τυχερούς βέβαια με τους αντίστοιχους υπολογιστές – εμείς ήμασταν καταδικασμένοι να πυροβολάμε δέκα φορές το κάθε καταραμένο UFO μέχρι να εκτελεστεί η σωστή εντολή!

Και τώρα που έχουμε frames τι γίνεται; Τα πράγματα έχουν απλοποιηθεί πολύ! Πάμε να τα δούμε.

9.2 Collision Detection, Part I – Εξωγήινε σε Έφαγα!

Ρίξτε λίγο μια ματιά στο super class `Craft` και θα θυμηθείτε ότι έχουμε αποθηκεύσει τις συντεταγμένες του σκάφους (είτε του δικού μας είτε του εξωγήινου) σε ένα αντικείμενο τύπου `Rect`:

```
    self.rect = pygame.Rect(coord,(self.ship_width, self.ship_height))
```

Το παραπάνω μας έχει ήδη διευκολύνει στην κίνηση, καθώς μπορούμε με ένα αντικείμενο rect να αλλάξουμε τις συντεταγμένες επιτόπου, όπως φαίνεται στην ρουτίνα κίνησης:

```
    distance_x = speed_x * time
    distance_y = speed_y * time
    self.rect.move_ip(distance_x,distance_y)
```

Δεν είναι όμως μόνο αυτό: Η κλάση `Rect` μας παρέχει την συνάρτηση `collidepoint` που ελέγχει αν κάποιες συντεταγμένες που δίνουμε βρίσκονται μέσα στο παραλληλόγραμμο (rectangle) που ελέγχουμε. Έτσι, αν `AlienShip` είναι ένα αντικείμενο τύπου “εξωγήινος” και μια

βολή βρίσκεται στις συντεταγμένες (120,248) η εντολή:

```
collision = AlienShip.rect.collidepoint((120,248))
```

θα επιστρέψει `True` αν υπάρχει "σύγκρουση" επιτρέποντας μας να κάνουμε αυτό που πάντα θέλαμε: Να ξεπαστρέψουμε τους εισβολείς! (δυστυχώς με μια μικρή παραλλαγή θα επιτρέψει και σε αυτούς να ξεπαστρέψουν εμάς). Μια αρχική εκδοχή του κώδικα μπορεί να είναι:

```
    for theshot in firelist:
      theshot.Move(time)
      theshot.Show(screen)
      if theshot.GoneAbove(0):
        firelist.remove(theshot)
      else:
        for AlienShip in AlienShips:
          if AlienShip.rect.collidepoint(theshot.GetXY()):
            explosion.play()
            if theshot in firelist:
              firelist.remove(theshot)
            AlienShips.remove(AlienShip)
```

Το πρώτο κομμάτι του κώδικα το ξέρετε ήδη: είναι αυτό που κινεί τις βολές και ελέγχει αν κάποια έχει ξεφύγει από την οθόνη για να τη διαγράψει από τη λίστα. Αν δεν έχει ξεφύγει:

- Διατρέχουμε όλη τη λίστα των `AlienShips`
- Ελέγχουμε αν η τρέχουσα βολή (`theshot`) συγκρούεται με κάποιο `AlienShip` (εντολή `collidepoint`)
- Αν αυτό συμβαίνει, ακούγεται ο ήχος της έκρηξης (σε πείσμα της φυσικής και του κενού του διαστήματος), η βολή αφαιρείται από τη λίστα `firelist` και ο αντίστοιχος εξωγήινος από τη λίστα `AlienShips`. Η διαγραφή από τη λίστα σημαίνει και εξαφάνιση τους από την οθόνη στο αμέσως επόμενο καρέ!

Η συνάρτηση `GetXY` είναι πολύ απλή. Ανήκει στο `Laser` class και απλώς επιστρέφει τις συντεταγμένες της βολής μας:

```
  def GetXY(self):
    return (self.x1,self.y1)
```

Δεν σχολιάζω καθόλου το `explosion.play()` που είναι ένα ακόμα ηχητικό εφέ που προσθέσαμε.

Και ναι, αυτό είναι όλο! Αν τρέξετε το πρόγραμμα μπορείτε να πάρετε επιτέλους εκδίκηση εξαφανίζοντας όσα κύματα εξωγήινων θέλετε. Όσο προλαβαίνετε δηλαδή, μια και δεν έχουν ακόμα αρχίσει να ρίχνουν.

Το ωραίο είναι ότι με τον τρόπο που φτιάξαμε το παιχνίδι, κάθε φορά που τελειώνουμε με ένα κύμα εξωγήινων, εμφανίζεται αμέσως το επόμενο. Θυμηθείτε το μαγικό μας κόλπο:

```
if not AlienShips:
  ...
```

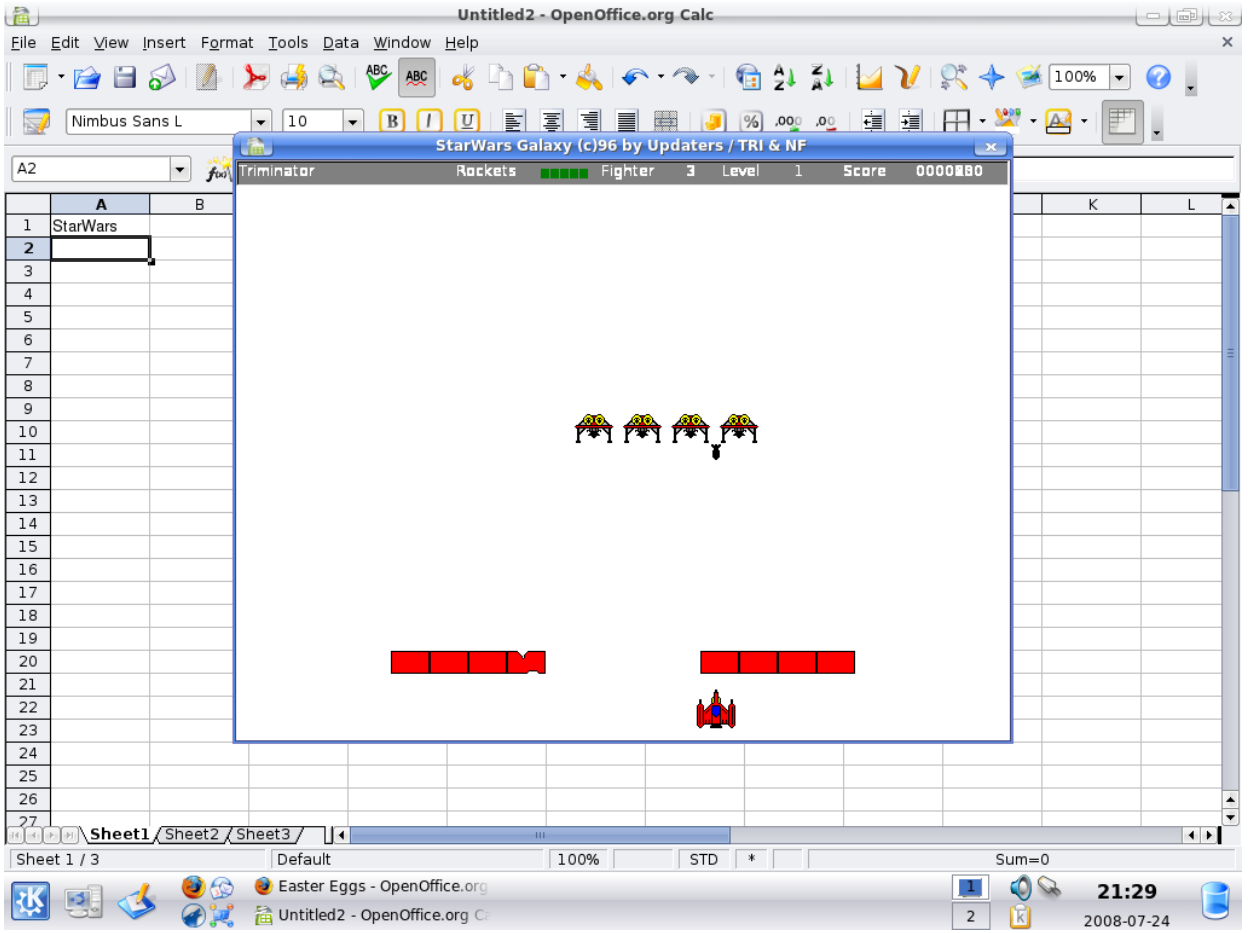
9.3 Η Αυτοκρατορία (των Χαζών) Αντεπιτίθεται!

Ένα παιχνίδι που θα πυροβολάμε μόνο εξωγήινους χωρίς να μας αντεπιτίθενται θα γίνει βαρετό πολύ σύντομα: κάτι σαν το Graphics Match όπου μόνο χάναμε. Ήρθε η ώρα αυτά τα χαζά πλάσματα να αποκτήσουν λίγη αξιοπρέπεια προβάλλοντας μια κάποια αντίσταση.

Πριν ξεκινήσουμε, ας θυμηθούμε λίγο τις παραμέτρους στον constructor του `Laser` class:

```
class Laser:
  def __init__(self, coord, color, size, speed, refline, voffset):
```

όπου φυσικά `coord` είναι οι συντεταγμένες, `color` είναι το χρώμα της βολής (διαφορετικό για τον εξωγήινο), `size` είναι το μήκος της γραμμής Laser, `refline` είναι η γραμμή αναφοράς εκκίνησης της βολής (το κάτω μέρος για τον εξωγήινο) και `voffset` είναι μια μικρή διόρθωση στον κάθετο άξονα της βολής που μάλλον δεν θα χρειαστούμε για τα `AlienShip`. Η συνάρτηση `Fire` στο `Alien` class θα είναι κάπως έτσι:



Εικόνα 9.1: Τα *easter eggs* (αν δεν τα ξέρετε) είναι μικρά προγραμματάκια κρυμμένα μέσα σε μεγαλύτερα (και πολλές φορές “σοβαρά προγράμματα”) τα οποία ενεργοποιούνται με κάποιο περίεργο, κρυφό τρόπο. Στις παλιές εκδόσεις του OpenOffice, υπήρχε κρυμμένο μέσα ένα παιχνίδι με τίτλο StarWars (μια ακόμα εκδοχή του Space Invaders στην πραγματικότητα). Για να ξεκινήσει έπρεπε κάποιος να γράψει `=GAMES(“StarWars”)` στο πρώτο κελί. Δυστυχώς στις νέες εκδόσεις του openoffice και libreoffice τα easter eggs αφαιρέθηκαν.

```
def Fire(self):
  theshot = Laser((self.rect[0]+self.ship_midwidth,
                   self.rect[1]+self.firebaseline),self.firecolor,
                   self.shot_height, self.firespeed,
                   self.rect[1]+self.firebaseline, 0)
  return theshot
```

Εννοείται ότι έχουμε ορίσει χρώμα, `firebaseline` και `shot_height` στον constructor του `Alien` class.

Καλά όλα αυτά, αλλά δεν σημαίνει ότι οι εξωγήινοι ρίχνουν κιόλας. Θα πρέπει με κάποιο τρόπο να αποφασίζουν πότε (και αν) θα ρίξει ο καθένας τους. Δεν πρόκειται να βγάλουν χέρια και να χτυπάνε το SPACE στο πληκτρολόγιο! (και αμφιβάλλουμε αν θα τους το δίνατε έτσι και αλλιώς). Για άλλη μια φορά θα μας σώσει η `randint`:

```
    for AlienShip in AlienShips:
      AlienShip.Show(screen)
      AlienShip.Move(time)
      if randint(0,10)==9:
        if alienfirelist:
          if alienfirelist[-1].DistanceTravelled()>=100:
            alienfirelist.append(AlienShip.Fire())
        else:
          alienfirelist.append(AlienShip.Fire())
```

Ο εξωγήινος ρίχνει μόνο αν το `randint(0,10)` φέρει την τιμή 9. Ναι, ουσιαστικά ο εξωγήινος… ρίχνει ζάρι για να αποφασίσει αν θα ρίξει ή όχι. Και βέβαια μπορούμε να κάνουμε το παιχνίδι πιο δύσκολο γράφοντας κάτι σαν αυτό:

```
if randint(0,10)>4:
...
```

οπότε και οι βολές θα είναι πιο συχνές. Το `alienfirelist` φυσικά είναι η λίστα που κρατάει τις εχθρικές βολές και αρχικοποιείται έξω από το βρόχο με την εντολή:

```
alienfirelist = []
```

Ότι ισχύει για τις δικές μας βολές ισχύει και για τις εχθρικές: δεν ξεκινάει νέα βολή αν η προηγούμενη δεν έχει διανύσει μια ελάχιστη απόσταση που έχουμε ορίσει ως 100 pixels και ελέγχουμε με την `DistanceTravelled()`. Δυστυχώς η `DistanceTravelled()` όπως την έχουμε φτιάξει μέχρι στιγμής δεν λειτουργεί σωστά για τις εχθρικές βολές. Μπορείτε να φανταστείτε γιατί;

Απλά, οι βολές αυτές πηγαίνουν προς τα κάτω ενώ οι δικές μας προς τα πάνω. Έτσι, η συνάρτηση παράγει αρνητικές τιμές για τις εχθρικές βολές ενώ στην ουσία μας ενδιαφέρει η απόλυτη τιμή αυτής της απόστασης. Και μόλις είπαμε την λέξη κλειδί για το πρόβλημα μας: η Python μας παρέχει την συνάρτηση απόλυτης τιμής `abs` όπως σχεδόν όλες οι γλώσσες προγραμματισμού! Άρα η `DistanceTravelled()` θα γίνει τώρα:

```
  def DistanceTravelled(self):
    return abs(self.refline - self.y1)
```

Η αλλαγή αυτή δεν επηρεάζει φυσικά καθόλου τη λειτουργία της συνάρτησης στις δικές μας βολές. Το μόνο που μένει είναι να δείξουμε και να κινήσουμε τις βολές:

```
1   def DistanceTravelled(self):
2     for theshot in alienfirelist:
3       theshot.Move(time)
4       theshot.Show(screen)
5       if theshot.GoneBelow(laserdownlimit):
6         alienfirelist.remove(theshot)
```

Αντί για `GoneAbove` έχουμε φυσικά `GoneBelow` καθώς οι εχθρικές βολές κατευθύνονται προς τα κάτω (ή προς τα πάνω μας!). Το `laserdownlimit` το έχουμε ορίσει έξω από το βασικό βρόχο ως:

```
  laserdownlimit = screenheight - 40
```

και φυσιολογικά θα έπρεπε να έχουμε και `laseruplimit = 0` αντί να είναι hardcoded μέσα στον κώδικα (hint: φτιάξτε το!). Είμαστε τώρα ένα βήμα πριν οι εξωγήινοι γίνουν επικίνδυνοι: ρίχνουν, αλλά είμαστε ακόμα άτρωτοι στις βολές τους. Αυτό θα αλλάξει άμεσα!

9.4 Collision Detection Part II – Οι Βολές Πέφτουν Βροχή!

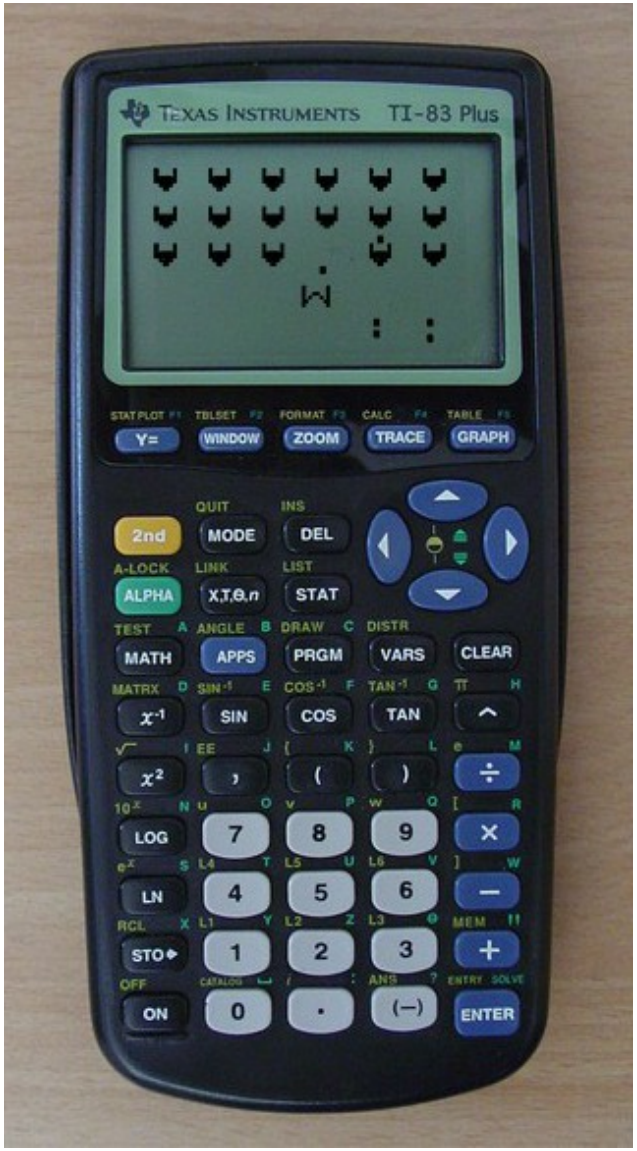
Εξασκηθείτε λίγο όσο οι εξωγήινοι είναι ακίνδυνοι: εδώ θα γράψουμε το collision detection για τις βολές που πέφτουν στο δικό μας σκάφος! Είναι αρκετά όμοιο με το part I:

```
    for theshot in alienfirelist:
      theshot.Move(time)
      theshot.Show(screen)
      if theshot.GoneBelow(laserdownlimit):
        alienfirelist.remove(theshot)
      else:
        if SpaceShip.rect.collidepoint(theshot.GetXY()):
          destroyed.play()
          if theshot in alienfirelist:
            alienfirelist.remove(theshot)
```

Για κάθε εχθρική βολή που ταξιδεύει στην οθόνη μας:

- Αν έχει βγει εκτός οθόνης εξαφανίζεται (αφαιρείται από τη λίστα `alienfirelist`)
- Διαφορετικά, αν έχει χτυπήσει το διαστημόπλοιο μας ακούγεται ο ήχος της έκρηξης (destroyed) και πάλι η βολή αφαιρείται από τη λίστα
- Αν δεν συμβαίνει τίποτα από τα δύο, η βολή απλώς συνεχίζει να κινείται.

Παρατηρήστε βέβαια ότι το διαστημόπλοιο μας δεν διαλύεται με τη βολή, όχι εμείς (θα) έχουμε ασπίδα! Χάνουμε όταν η ασπίδα μας πέφτει στο μηδέν. Όσο πετυχαίνουμε εξωγήινους η ασπίδα μας σιγά – σιγά επανέρχεται. Μεγάλο πλεονέκτημα να είσαι προγραμματιστής τελικά: ορίζεις τους κανόνες του παιχνιδιού (προς το συμφέρον σου βέβαια!)



Εικόνα 9.2: Τα παιχνίδια τύπου invaders είναι τόσο δημοφιλή που έχουν γραφτεί για τα πιο απίθανα μηχανήματα. Στη φώτο, space invaders για την προγραμματιζόμενη επιστημονική αριθμομηχανή (sic) TI-83 της Texas Instruments. Και τώρα ξέρετε τι δώρο να ζητήσετε από τους γονείς σας ως πρωτοετής φοιτητής. Για καθαρά εκπαιδευτικούς σκοπούς βέβαια!

Για την ώρα δεν έχουμε όμως ούτε ασπίδα, ούτε καν score. Και έχουμε ένα ακόμα προβληματάκι: δεν υπάρχει καμιά ένδειξη στην οθόνη ότι έχουμε χτυπηθεί – ακούγεται μόνο ένας ήχος. Όλα αυτά πρέπει φυσικά να τα φτιάξουμε.

9.5 Τήρηση Score

Αυτό είναι εύκολο. Ορίστε μια πολύ απλή κλάση για το `ScoreBoard`:

```
class ScoreBoard:
  def __init__(self,x,y,font, fontsize):
    self.x = x
    self.y = y
    self.font = pygame.font.SysFont(font,fontsize)
    self.score = 0

  def Change(self, amount):
    self.score += amount

  def Show(self,surface):
    scoretext = self.font.render("Score: "+str(self.score), True, (0,0,255))
    surface.blit(scoretext,(self.x,self.y))

  def GetValue(self):
    return self.score

  def SetValue(self, score):
    self.score = score
```

Είναι πιστεύουμε εντελώς προφανής: Το `Score` θα εμφανίζεται στην θέση `(x,y)` της οθόνης με τη γνωστή μας μέθοδο `blit`. Έχουμε συναρτήσεις `Change` για να προσθέσουμε στο score την τιμή `amount` και `GetValue / SetValue` για να διαβάσουμε την τιμή ή να

Εικόνα 9.3: Ο ZX-81, δημιούργημα της Sinclair Research, γνωστή από το ZX Spectrum, είναι ένα από τα πρώτα οικιακά micros με 1Kb RAM (και εξωτερική επέκταση στα 16Kb) με το καταπληκτικό χαρακτηριστικό ότι ήταν… ασπρόμαυρο. Με ταχύτητα 1MHz ή 2MHz (το λεγόμενο… fast mode) πάλι μπορούσε να τρέξει το Space Invaders όπως βλέπετε στη φώτο (προφανώς το συγκεκριμένο είναι γραμμένο σε assembly και όχι σε BASIC).

θέσουμε το score απευθείας σε μια τιμή της αρεσκείας μας (πάλι στο cheat πάει το μυαλό σας ε;). Στο κύριο πρόγραμμα μας θα έχουμε:

Αρχικοποίηση Score:

```
  score = ScoreBoard(0,0,"impact",32)
```

Σε κάθε επιτυχία μας:

```
          if AlienShip.rect.collidepoint(theshot.GetXY()):
            score.Change(10)
```

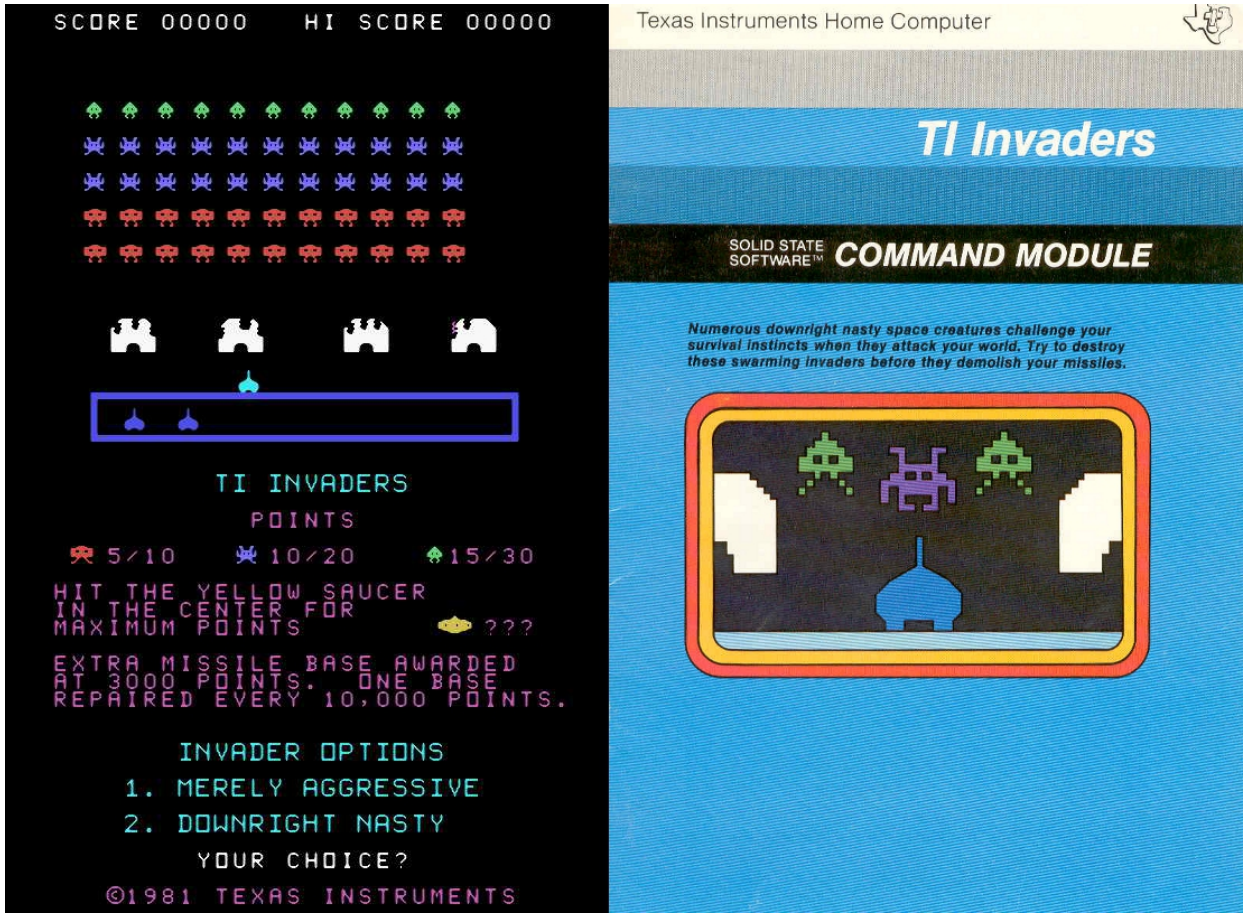
Για να δούμε όμως και αυτή την ασπίδα που λέγαμε, για να σταματήσουμε να παριστάνουμε τους… Highlanders!

9.6 ShieldMeter Class: Μια Κλάση για την Ασπίδα μας!

Το `ShieldMeter` class είναι αρκετά απλό: σχεδιάζει μια μπάρα στην οθόνη που μειώνεται καθώς πέφτει η ενέργεια της ασπίδας μας (όσο δηλ. καθόμαστε και τις τρώμε άγρια από τους εξωγήινους) ή αυξάνεται μετά από κάποιες δικές μας πετυχημένες βολές. Αν η ασπίδα μας πέσει κάτω από κάποια τιμή (`warnvalue`) το χρώμα της μπάρας αλλάζει. Προφανώς η ασπίδα έχει μια μέγιστη τιμή (`maxvalue`) πέρα από την οποία δεν αυξάνεται και εννοείται δεν μπορεί να πάει κάτω από το μηδέν (στο μηδέν βλέπουμε ήδη τα ψηφιακά ραπανάκια να φυτρώνουν από κάτω).

```
class ShieldMeter:
  def __init__(self, x, y, maxvalue, warnvalue):
```

Το μόνο ενδιαφέρον πρακτικά κομμάτι στον κώδικα είναι η σχεδίαση με την `draw.rect`, οι άλλες συναρτήσεις είναι τόσο απλές όσο και του `ScoreBoard`:



Εικόνα 9.4: Στο προηγούμενο κεφάλαιο σας έδειξα την δική μου εκδοχή του TI Invaders, σε BASIC – πριν καν μάθω ότι η TI είχε κυκλοφορήσει το Space Invaders σε cartridge. Οι δύο φώτο είναι από το παιχνίδι, ενώ βλέπετε και τις οδηγίες του. Είναι αρκετά δύσκολο και γίνεται ακόμα χειρότερο αν προσπαθείτε να παίξετε με τα original TI joysticks (έχουν ψηφιστεί μέσα στα δέκα χειρότερα όλων των εποχών).

Εικόνα 9.5: Όταν βαριόμασταν το Space Invaders σειρά είχε το Pacman. Άπειρα joysticks πρέπει να έχουν διαλυθεί εξαιτίας αυτών των δύο παιχνιδιών. Στη φώτο βλέπετε ένα από τα περιοδικά που ασχολείται με retro gaming – και πρέπει να σας πληροφορήσω ότι η retro σκηνή καλά κρατεί. Οι περισσότεροι βέβαια χρησιμοποιούν πλέον εξομοιωτές, αλλά όσοι έχουν τα original μηχανήματα δεν τα αλλάζουν με τίποτα!

```
  def Show(self, surface):
    if self.currentvalue < self.warnvalue:
      self.shieldcolor = (255,0,0)
    else:
      self.shieldcolor = (0,255,0)
    pygame.draw.rect(surface,self.shieldcolor,(self.x, self.y,  self.currentvalue,25))
```

Το “μαγικό” 25 είναι το ύψος της μπάρας στην οθόνη. Μάλλον θα θέλετε να κάνετε κάτι για αυτό στον constructor, έτσι δεν είναι;

Στο κύριο πρόγραμμα, το `shield` αρχικοποιείται ως εξής:

```
shield = ShieldMeter(200,10,250,75)
```

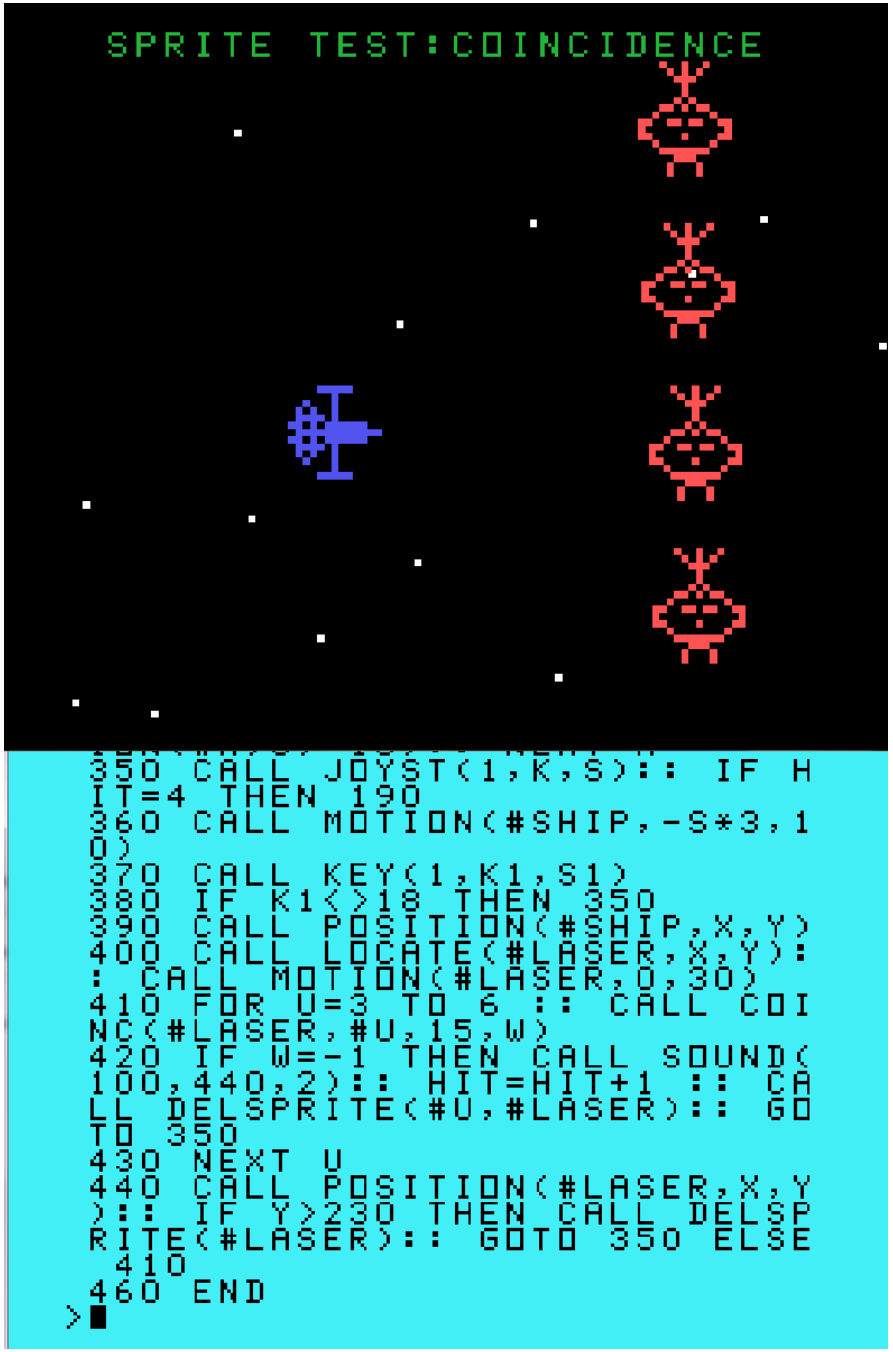
όπου το 200,10 είναι οι συντεταγμένες εμφάνισης, το 250 είναι η μέγιστη τιμή και το 75 η τιμή προειδοποίησης. Σε κάθε… τυχερή βολή των χαζοεξωγήινων χάνουμε 25 μονάδες ασπίδας:

```
      if SpaceShip.rect.collidepoint(theshot.GetXY()):
        destroyed.play()
        shield.Decrease(25)
```

ενώ κάθε φορά που το score μας διαιρείται ακριβώς με το 100, παίρνουμε 25 μονάδες ασπίδας:

```
        if AlienShip.rect.collidepoint(theshot.GetXY()):
          score.Change(10)
          explosion.play()
          if score.GetValue() % 100 == 0:
            shield.Increase(25)
```

Φυσικά μέσα στον κύριο βρόχο υπάρχει η εντολή `shield.Show(screen)` για να εμφανίζεται η ασπίδα μας στην οθόνη!



Εικόνα 9.6: Ανίχνευση συγκρούσεων με τον παλιό καλό (;) τρόπο. Τα sprites κινούνται αυτόματα και η σύγκρουση δεν γίνεται αντιληπτή αν δεν συμπέσει με την εκτέλεση της εντολής ανίχνευσης. Στην TI Extended BASIC, η εντολή αυτή είναι η CALL COINC (από το coincidence, σύμπτωση) που φαίνεται στο listing. Δείτε και το παρακάτω διασκεδαστικό video που δείχνει το πρόγραμμα και συγκρίνει το τότε με το τώρα: http://www.youtube.com/watch?v=6rK7_2zrauM

9.7 Ανίχνευση Τέλους Παιχνιδιού

Όλα τα πράγματα έχουν ένα τέλος και μοιραία κάποια στιγμή οι εξωγήινοι θα σας φάνε λάχανο! Το παιχνίδι προφανώς πρέπει να ανιχνεύει το τέλος της ασπίδας σας, να δείχνει μήνυμα Game Over και να μην επιτρέπει να συνεχίσετε. Για το μήνυμα έχουμε μια πολύ απλή συνάρτηση `GameOverShow` που κάνει `blit` τα αντίστοιχα μηνύματα:

```
def GameOverShow(screen):
  font = pygame.font.SysFont("impact", 32)
  gameovertext = font.render("Game Over!",True,(255,255,255))
  text_x = CenterMessage(screen, gameovertext)
  screen.blit (gameovertext,(text_x,280))
  gameovertext = font.render("Press R to Restart", True, (255,255,255))
  text_x = CenterMessage(screen, gameovertext)
  screen.blit(gameovertext,(text_x,320))
  return
```

Όπως βλέπετε έχουμε προσθέσει και την pygame εκδοχή της `CenterMessage` για το κεντράρισμα του κειμένου. Στον κύριο βρόχο έχουμε:

```
    if shield.GetValue() == 0:
      GameOverShow(screen)
      GameOver = True
```

Για να εξασφαλίσουμε ότι το παιχνίδι δεν παίζει όσο το `GameOver = True` αλλά και να επιτρέψουμε την επανεκκίνηση του με το πλήκτρο R, έχουμε αλλάξει κατάλληλα τις εντολές στην ανίχνευση του πληκτρολογίου:

```
        if key[K_SPACE] and not GameOver:
```

Το διαστημόπλοιο μας ρίχνει μόνο αν δεν έχει επέλθει το `GameOver`. Κινείται όμως αριστερά – δεξιά (demo mode!). Οι εξωγήινοι μας ρίχνουν, αλλά οι βολές τους περνάνε από μέσα μας χωρίς να μας πειράξουν επειδή έχουμε αλλάξει την γραμμή:

```
if SpaceShip.rect.collidepoint(theshot.GetXY()) and not GameOver:
```

Τέλος, πιέζοντας το R, ο χρήστης έχει δυνατότητα να ξαναπαίξει. Το παιχνίδι είναι τόσο εθιστικό άλλωστε, που ήταν αδύνατον να παραλείψουμε αυτό το… feature:

```
1         if key[K_r] and GameOver:
2           GameOver = False
3           shield.SetValue(250)
4           score.SetValue(0)
```

Φυσικά η μεταβλητή `GameOver` αρχικοποιείται έξω από το βρόχο ως `GameOver = False`.

Επιτέλους, μετά από όλα αυτά έχουμε ένα πλήρως λειτουργικό παιχνίδι το οποίο θα μας διασκεδάσει με τις ώρες, τόσο να το παίζουμε όσο και να το επεκτείνουμε! Γιατί φυσικά, ο σωστός προγραμματιστής διασκεδάζει περισσότερο γράφοντας το παιχνίδι, παρά παίζοντας το. Εξάλλου, όσο παίζουμε ανακαλύπτουμε δυνατότητες και λειτουργίες που θέλουμε να προσθέσουμε: ο κώδικας μας έρχεται σε κύματα (σαν τους εξωγήινους ένα πράγμα) και δεν τελειώνει ποτέ.

Σε λίγο θα δούμε πως θα κάνετε το διαστημοπλοιάκι μας να αλλάζει χρώμα όταν δέχεται βολές – ακούγεται απλό, αλλά θα διαπιστώσετε ότι χρειάζεται να μάθετε και νέα κόλπα για να γίνει σωστά. Θα συζητήσουμε ακόμα άλλες βελτιώσεις που μπορείτε να κάνετε στον κώδικα αλλά και ιδέες για να επεκτείνετε το παιχνίδι, μια πρόκληση που αφήνω αποκλειστικά στα χέρια σας και στην… καφεΐνη που θα καταναλώσετε.

Όπως σας αποδείξαμε ο προγραμματισμός όχι μόνο δεν είναι βαρετός, αλλά αντίθετα αποτελεί την πιο ενδιαφέρουσα και δημιουργική ασχολία σε ένα υπολογιστή. Ελπίζουμε τα επιχειρήματα μας να ήταν πειστικά – μην πείτε όχι: θα σας κυνηγήσω με το Laser!

9.8 Οπτική Ένδειξη LASER Hit

Τι καλά που θα ήταν αν κάθε φορά που μας χτυπάνε οι χαζοί εξωγήινοι να είχαμε μια οπτική ένδειξη. Τη δεδομένη στιγμή η μόνη ένδειξη που έχουμε είναι ένας ήχος (και φυσικά να βλέπουμε την ασπίδα μας να πέφτει). Σε πολλά shoot-em-up θα παρατηρήσετε ότι το διαστημοπλοιάκι αναβοσβήνει ή αλλάζει χρώμα όταν δέχεται μια βολή. Το δικό μας όμως τίποτα. Τι πρέπει να κάνουμε;

Με λίγη σκέψη καταλήγουμε στο συμπέρασμα ότι χρειαζόμαστε μια δεύτερη εικόνα για το διαστημόπλοιο μας, με ένα χρώμα ελαφρώς διαφορετικό.

Η γενική ιδέα είναι προφανώς ότι από τη στιγμή που θα δεχτούμε μια βολή θα προβάλλουμε εναλλάξ την κανονική εικόνα του διαστημοπλοίου και την δεύτερη για κάποιο χρονικό διάστημα. Αλλά υπάρχει πρόβλημα.

- Πόσο γρήγορα θα γίνεται η εναλλαγή; Αν αλλάζουμε εικόνα ανά καρέ που δημιουργούμε (η προγραμματιστικά εύκολη λύση) δεν θα βλέπουμε τίποτα! Μια αλλαγή που γίνεται 50-100 και παραπάνω φορές το δευτερόλεπτο είναι πρακτικά αόρατη. (Βλέπετε τις λάμπες στο σπίτι σας να αναβοσβήνουν;)

- Πως θα ρυθμίσουμε το συνολικό χρονικό διάστημα προβολής του εφέ; Θα πρέπει να ρυθμίσουμε είτε χρόνο είτε συνολικό αριθμό εναλλαγής των εικόνων μέχρι να επιστρέψουμε στην αρχική, μοναχική εικόνα.

Αφού αποφασίσαμε ότι δεν μπορούμε να εναλλάσσουμε εικόνα ανά καρέ και δεν μπορούμε να βασιστούμε στο framerate για το εφέ, υποπτευόμαστε ότι θα υπάρχει κάποιος τρόπος να το χρονομετρήσουμε. Ξέρουμε ήδη το γνωστό μας clock το οποίο μας βοηθάει να κρατάμε σταθερή την ταχύτητα των αντικειμένων μας ανεξάρτητα από το framerate. Μήπως θέλουμε και εδώ κάτι αντίστοιχο;

Ευτυχώς για μας, το pygame υποστηρίζει τη χρήση timers και μάλιστα πολλαπλών. Μπορούμε να ορίσουμε ένα timer με συγκεκριμένο χρόνο tick. Σε κάθε tick του timer θα λαμβάνουμε ένα event – αρκεί να προσθέσουμε μερικές γραμμές στη διαχείριση events που ήδη έχουμε.

9.9 Αλλαγές στις Βασικές Κλάσεις

Έχοντας μια γενική ιδέα στο μυαλό μας, πριν προσπαθήσουμε να βάλουμε τον timer πρέπει να ασχοληθούμε με πιο… πεζά θέματα. Βλέπετε, όπως έχουμε φτιάξει την κλάση `Craft` δεν επιτρέπει παρά μόνο μια εικόνα για κάθε αντικείμενο και ξαφνικά θέλουμε δύο. Ας μετατρέψουμε λίγο την κλάση `Craft`:

```
class Craft(object):
  def __init__ (self, imagefiles, coord):
    self.shape = [pygame.image.load(imagefile) for imagefile in imagefiles]
    self.ship_width = self.shape[0].get_width()
    self.ship_height = self.shape[0].get_height()
    self.rect = pygame.Rect(coord,(self.ship_width, self.ship_height))
    self.ship_midwidth = self.ship_width / 2
    self.firecolor=(255,0,0)
    self.firespeed = -800
    self.shotlength = 20

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

Το `self.shape` είναι πλέον λίστα και φορτώνουμε σε αυτό όλες τις εικόνες από τα ονόματα αρχείων που περιέχει το `imagefiles` χρησιμοποιώντας το πολύ βολικό list comprehension (το έχουμε ξαναδεί, ψάξτε λίγο στον κύριο βρόχο!) Το `imagefiles` μπορεί να είναι λίστα ή tuple.

Πρέπει όμως να δούμε και τις κλάσεις που έχουν την `Craft` ως γονική. Η `Alien` βέβαια θα λαμβάνει μόνο μια εικόνα, όμως αφού έχουμε μετατρέψει την γονική κλάση να χειρίζεται λίστα από εικόνες τι θα κάνουμε;

```
class Alien(Craft):
  def __init__(self, imagefile, coord, speed_x, speed_y):
    imagefiles = (imagefile,)
    super(Alien, self).__init__(imagefiles, coord)
```

Πήραμε το ένα όνομα αρχείου εικόνας που θα λάβει ο constructor της `Alien` από το κύριο πρόγραμμα και το μετατρέψαμε σε tuple του… ενός στοιχείου για να το δώσουμε στον γονικό constructor της `Craft`! Η αστεία αυτή μετατροπή έγινε με την εντολή:

```
    imagefiles = (imagefile,)
```

Το κόμμα δείχνει ότι δημιουργούμε tuple με ένα μόνο στοιχείο. Δεν θα μπορούσαμε να χρησιμοποιήσουμε μόνο την παρένθεση γιατί η Python δεν θα ήξερε αν προσπαθούμε να φτιάξουμε tuple ή παράσταση (όλα τα έχουν προβλέψει πια αυτοί οι Python developers!)

Πρέπει να φτιάξουμε και μια συνάρτηση `Show` για το `Alien` class, το οποίο μέχρι τώρα χρησιμοποιούσε αυτούσιο την αντίστοιχη του `Craft` class:

```
  def Show(self, surface):
    imageindex = 0
    super(Alien,self).Show(surface,imageindex)
```

Τέλειο! Αφού το `Alien` έχει πάντα σταθερή εικόνα, το `imageindex` θα είναι πάντα μηδέν και θα δείχνει τη μια και μοναδική εικόνα που έχουμε δώσει. Ενώ στο δικό μας διαστημόπλοιο, μπορούμε να αλλάζουμε το `imageindex` κατά βούληση αλλάζοντας του έτσι και τη μορφή.

Πάμε να δούμε τις αλλαγές στο κύριο πρόγραμμα. Έξω από το βρόχο ορίζουμε τις μεταβλητές `imageindex` και `flashcount`:

```
  imageindex = 0
  flashcount = 0
```

Η δημιουργία του `SpaceShip` αλλάζει ελαφρά αφού πλέον θα του δώσουμε ένα tuple με δύο εικόνες:

```
  shipimages = ('spaceship2.png', 'spaceship3.png')
  SpaceShip = SpaceCraft(shipimages,spaceship_pos, spaceship_low, spaceship_high,laser)
```

Τέλος, το μόνο άλλο που αλλάζει είναι η κλήση της `Show` για το διαστημόπλοιο μας μέσα στον κύριο βρόχο:

```
    SpaceShip.Show(screen,imageindex)
```

9.10 Προσθήκη του Timer

Το παιχνίδι μας πρέπει τώρα να λειτουργεί όπως πριν και μένει μόνο ο timer για την εναλλαγή των εικόνων. Ώρα να τον εγκαινιάσουμε:

```
        if SpaceShip.rect.collidepoint(theshot.GetXY()) and not GameOver:
          destroyed.play()
          pygame.time.set_timer(USEREVENT+1,25)
          shield.Decrease(25)
```

Όταν χτυπηθούμε, ξεκινά ένας timer ο οποίος θα στέλνει το event `USEREVENT+1` ανά 25 χιλιοστά του δευτερολέπτου. Το `USEREVENT` περιέχεται και αυτό στο `pygame.locals` και όπως φαντάζεστε είναι η αρχή μιας σειράς αριθμών που δεν χρησιμοποιούνται κάπου αλλού και μπορούμε να τα χρησιμοποιήσουμε για τα δικά μας συμβάντα (μπορούμε να έχουμε `USEREVENT+1, USEREVENT+2` κλπ).

Το μόνο που μένει τώρα είναι να αναβοσβήσουμε το διαστημόπλοιο μας δέκα φορές και μετά να απενεργοποιήσουμε το timer. Οι γραμμές προστίθενται στο βρόχο `for` διαχείρισης των συμβάντων:

```
      if event.type == USEREVENT+1:
        if flashcount < 10:
          flashcount += 1
          if imageindex == 1:
            imageindex = 0
          else:
            imageindex = 1
        else:
          imageindex = 0
          flashcount = 0
          pygame.time.set_timer(USEREVENT+1,0)
```

Κάθε φορά που λαμβάνουμε ένα event τύπου `USEREVENT+1`, ελέγχουμε το `flashcount`. Αν δεν έχει φτάσει ακόμα την τιμή 10, αλλάζουμε το `imageindex` από 0 σε 1 ή αντίστροφα. Όταν το `flashcount` γίνει 10, επιστρέφουμε στην κανονική εικόνα, μηδενίζουμε το `flashcount` και φυσικά απενεργοποιούμε το timer θέτοντας του μηδέν στο χρόνο tick. Πανεύκολο!

9.11 Επεκτείνοντας το Παιχνίδι: Μερικές Βασικές Ιδέες

Ο κώδικας είναι το καλύτερο παιχνίδι! Αν το πιστεύετε και εσείς, ορίστε μερικές ιδέες για να επεκτείνετε το Pygame Invaders:

- Κάντε μερικούς εξωγήινους να ρίχνουν δωράκια όταν τους σκοτώνετε: Power ups, αύξηση ασπίδας, πιο γρήγορο Laser για το σκάφος σας. Πάρτε ιδέες από παιχνίδια όπως το Arkanoid.
- Κάντε τη δυσκολία του παιχνιδιού να αλλάζει όσο αυξάνονται τα κύματα των εξωγήινων. Μπορεί να ρίχνουν πιο συχνά, να κινούνται πιο γρήγορα ή να έχουν κάποιο… σχέδιο να σας καταστρέψουν αντί να κάνουν bounce αριστερά – δεξιά.
- Προσθέστε πίστες: Μετά από μερικά κύματα θα μπορούσατε να κάνετε ένα γρήγορο warp (κινούμενο background έχετε!) και να βρίσκεστε κάπου με διαφορετικούς εξωγήινους.
- Βάλτε μια έξτρα πίστα με μετεωρίτες αντί για εξωγήινους. Δείτε το Parsec του TI-99/4A στο `http://www.youtube.com/watch?v=uCSQd0eJKQQ`.
- Στα εξελιγμένα διαστημικά παιχνίδια, στο τέλος εμφανίζεται ο Μεγάλος Κακός™. Φτιάξτε μια πίστα με τον τεράστιο εξωγήινο που θα πρέπει να του ρίξετε Laser, Phasers και το περίσσευμα της προχθεσινής πίτσας για να πεθάνει!
- Προσθέστε cheats για να παίζετε χωρίς να χάνετε – σε αντίθεση με τους φίλους σας (έπρεπε να το έχετε κάνει ήδη).
- Στο διαστημόπλοιο μας έχουμε ήδη και κίνηση πάνω – κάτω εκτός από αριστερά – δεξιά. Δεν ανιχνεύονται όμως συγκρούσεις μεταξύ του σκάφους μας και των εξωγήινων. Φτιάξτε το!

Θα χαρούμε πολύ να δούμε παραλλαγές και βελτιώσεις στο παιχνίδι μας – και γιατί όχι και τα δικά σας παιχνίδια! Αν θέλετε μάλιστα μπορούμε να τα δημοσιεύσουμε στο `http://pygamegr.wordpress.com` από όπου μπορείτε να κατεβάσετε και τον πλήρη και τελικό (για μας!) κώδικα του Pygame Invaders.

Το listing θα το βρείτε επίσης στο παράρτημα, σελ. 181. Μπορείτε επίσης να κατεβάσετε το πλήρες παιχνίδι από εδώ: `http://www.freebsdworld.gr/files/pygame-invaders.zip`

Παράρτημα Α

Προγράμματα

Α .1 Guess the Number

```
#/usr/bin/env python
#coding=utf-8
import random
thenumber=random.randint(1,50)
name=raw_input(”Δώσε το όνομα σου:”)
print ”Έχω σκεφτεί ένα αριθμό από 1 ως το 50”
print ”Μπορείς να τον βρεις;”
guess=0
tries=0
while guess!=thenumber:
  tries=tries+1
  guess=input(”Δώσε τον αριθμό:”)
  if guess>thenumber:
    print”Εδωσες μεγαλύτερο αριθμό!”
```

```
  if guess<thenumber:
    print″Έδωσες μικρότερο αριθμό!″

print ″Συγχαρητηρία ″,name,″ τον βρήκες σε″,tries,″προσπάθειες″
if tries==1:
  print ″Beginner’s luck!″
elif tries<=5:
  print ″Είσαι γρήγορος″
elif tries<=10:
  print ″Είσαι ένας βλάκας και μισός″
else:
  print ″My grandmother is faster (and she is dead)″
```

A .2 The Adventure

```
#
# The Python Adventure
# Based on an idea published in Pixel Magazine
# issue 18, January 1986
#
# Download this source file from http://pygamegr.wordpress.com
#

def getInput(moves,room):
  directions = [″Βόρεια″,″Νότια″,″Ανατολικά″,″Δυτικά″]
  destinations = moves[room]
  possiblemoves = []
  index = 0
  print ″Έξοδοι:″,
  for i in destinations:
```

```
    if i!=-1:
      print directions[index],
      possiblemoves.append(directions[index])
    index +=1
  print "\n"
  userinput=""
  while userinput not in possiblemoves:
    userinput=raw_input("Που θες να πας; ")
  return directions.index(userinput)

def main():
  # Room descriptions

  rooms = [ "Βρισκεσαι στον κήπο. Παντού σκοτάδι.",
            "Βρίσκεσαι στο μπάνιο. Ακούς θόρυβο.",
            "Βρίσκεσαι στο χωλ. Παραλίγο να σκοντάψεις. Σκάλα ανατολικά",
            "Βρισκεσαι στην αποθήκη. Η ατμόσφαιρα είναι αποπνικτική. Σκάλα Δυτικά, Μονοπάτι Νότια.",
            "Έφτασες στο καθιστικό. Βρήκες τον πατέρα σου.",
            "Βρίσκεσαι στο σαλόνι. Ακούγεται μουσική.",
            "Βρίσκεσαι στην τραπεζαρία. Τα πάντα είναι ανάστατα.",
            "Βρίσκεσαι στο διάδρομο. Είναι σκοτεινά. Σκάλα Νότια",
            "Είσαι στη κουζίνα. Ακούς φωνές.",
            "Είσαι στον πάνω διάδρομο. Παντού ησυχία. Σκάλα Βόρεια.",
            "Είσαι στο δωμάτιο του αδερφού σου. Ο αδερφός σου σε μαρτυρά!",
            "Είσαι στο δωμάτιο των γονέων σου. Η μητέρα σου σε έπιασε.",
            "Είσαι στο δωμάτιο σου. Είσαι ασφαλής." ]

  #
  # The following list contains lists :)
  # The list's index denotes the current room.
  # Each single list contains four numbers:
```

```
# First number -> Destination room if North (0)
# Second number -> Destination room if South(1)
# Third number -> Destination room if East (2)
# Fourth number -> Destination room if West (3)
#
# Destination is set to -1 if it does not exist
#

moves = [   [-1,2,-1,-1],
            [-1,-1,2,-1],
            [0,5,3,1],
            [-1,8,-1,2],
            [-1,6,5,-1],
            [2,8,-1,4],
            [4,7,-1,-1],
            [6,9,8,-1],
            [5,-1,-1,7],
            [7,11,12,10],
            [-1,-1,9,-1],
            [9,-1,-1,-1],
            [-1,-1,-1,9] ]

# When player reaches any of the following room(s),
# he wins the game

winrooms = [ 12 ]

# When player reaches any of the following rooms,
# he loses

lossrooms = [ 4, 10, 11 ]
```

```
  # Startup room
  # Doesn’t have to be 0, but must not be a winning or losing room

  room = 0

  print ”The Adventure!”
  print ”==============”

  endgame = False
  while not endgame:
    print rooms[room]
    if room in winrooms:
      print ”Κέρδισες! Η περιπέτεια τελείωσε.”
      endgame = True
    elif room in lossrooms:
      print ”Έχασες! Η περιπέτεια τελείωσε.”
      endgame = True
    else:
      direction = getInput(moves, room)
      destinations = moves[room]
      room = destinations[direction]

# Start program
if __name__ == ”__main__”:
    main()
```

A .3 Hello Pygame

```
#
# pygame hello world
#
# Όχι, δεν είναι τόσο πολύπλοκο όσο φαίνεται.
# Μην πέσετε από το μπαλκόνι!
#
import pygame
from pygame.locals import *
from sys import exit

# Define app window width and height

windowsize = (320,140)

def centerMessage(surface):
  return (windowsize[0] - surface.get_width())/2

#
# Process the event queue
# returns true if user clicks close
#

def getQuit():
  for event in pygame.event.get():
    if event.type == QUIT:
      return True


def main():
```

```
# Initialize the pygame library

pygame.init()
surfacecolor = (50,80,250)
screen = pygame.display.set_mode(windowsize, DOUBLEBUF, 32)
pygame.display.set_caption(”Hello Pygame!”)
textfont = pygame.font.SysFont(”Arial”,48)
thetext = textfont.render(”Hello World!”, True, (255,0,0),(255,255,0))

# Initialize text position

textx = centerMessage(thetext)
texty = 40

# Begin main loop

endprogram = False

while not endprogram:

  # fill screen with bluish tint

  screen.fill(surfacecolor)

  # Show text message

  screen.blit(thetext,(textx,texty))
  endprogram = getQuit()
  pygame.display.update()

# shutdown pygame and exit program
```

```
  pygame.quit()
  exit()

# Start program
if __name__ == ”__main__”:
  main()
```

A .4 Colorbars

```
#!/usr/bin/env python

import pygame
from pygame.locals import *
from sys import exit

from random import *

pygame.init()
screen = pygame.display.set_mode((640, 480), DOUBLEBUF)

while True:
  ypos=0
  for ypos in range(0,479,30):
    for xpos in range(0,639,30):
      for event in pygame.event.get():
        if event.type == QUIT:
          pygame.quit()
          exit()
      the_pos=(xpos,ypos,29,29)
```

```
      random_color = (randint(0,255), randint(0,255), randint(0,255))
      pygame.draw.rect(screen, random_color, the_pos)
  pygame.display.update()
```

A .5 Bouncing Ball

```
#
# Bouncing ball
# Download this source file
# and support files from http://pygamegr.wordpress.com
#

import pygame
from pygame.locals import *
from sys import exit


def getQuit():
  for event in pygame.event.get():
    if event.type == QUIT:
      return True
  return False

def main():
  pygame.init()
  ballimage = 'soccer-ball.png'
  x,y = 100.0,100.0
  xspeed,yspeed = 50,50
  windowsize = (640,480)
  surfacecolor = (50,80,250)
```

```
screen = pygame.display.set_mode(windowsize, DOUBLEBUF)
ball = pygame.image.load(ballimage)
ballwidth = ball.get_width()
ballheight = ball.get_height()
clock = pygame.time.Clock()

# Uncomment the framerate line and change
# time = clock.tick() in main loop to
# time = clock.tick(framerate)
# to limit the animation to a specific framerate

#framerate = 30

textfont = pygame.font.SysFont(”Arial”,24)

#
# Main loop
#

endprogram = False
while not endprogram:
  screen.fill(surfacecolor)
  screen.blit(ball, (x, y))
  time = clock.tick()
  thetext = textfont.render(str(1000/time), True, (255,0,0),(255,255,0))
  screen.blit(thetext,(0,0))
  time = time / 1000.0
  distance_x = time * xspeed
  distance_y = time * yspeed
  x = x + distance_x
  y = y + distance_y
```

```
    if (x > (640.0-ballwidth) or x<=0.0):
      xspeed = -xspeed
    if (y > (480.0-ballheight) or y<=0.0):
      yspeed = -yspeed
    pygame.display.update()
    endprogram = getQuit()

  pygame.quit()
  exit()

if __name__ == "__main__":
  main()
```

A .6 Αντικειμενοστραφές Bouncing Ball

```
#
# Bouncing ball - OOP :)
#

import pygame
from pygame.locals import *
import random
from sys import exit

class Ball:
  def __init__(self,theImage,x,y,xspeed,yspeed):
    self.ballimage = theImage
    self.x = x
    self.y = y
```

```
    self.xspeed = xspeed
    self.yspeed = yspeed
    self.shape = pygame.image.load(theImage)

  def Show(self,surface):
    surface.blit(self.shape, (self.x, self.y))

  def GetWidth(self):
    return self.shape.get_width()

  def GetHeight(self):
    return self.shape.get_height()

  def Move(self, time):
    distance_x = time * self.xspeed
    distance_y = time * self.yspeed
    self.x = self.x + distance_x
    self.y = self.y + distance_y

  def IsOutofX(self,xmin,xmax):
    if (self.x >= (xmax - self.GetWidth()) or self.x <= xmin):
      return True
    else:
      return False

  def IsOutofY(self,ymin,ymax):
    if (self.y >= (ymax - self.GetHeight()) or self.y <= ymin):
      return True
    else:
      return False
```

```
def getQuit():
  for event in pygame.event.get():
    if event.type == QUIT:
      return True
  return False

def main():
  pygame.init()
  ballimage = 'soccer-ball.png'
  balls=[]
  for i in range(0,8):
    x = random.randint(80,500)
    y = random.randint(80,400)
    xspeed = 0
    while (xspeed >= -5 and xspeed <= 5):
      xspeed = random.randint(-50,50)
    yspeed = 0
    while (yspeed >= -5 and yspeed <=5):
      yspeed = random.randint(-50,50)
    balls.append(Ball(ballimage,x,y,xspeed,yspeed))
  windowsize = (640,480)
  surfacecolor = (50,80,250)
  screen = pygame.display.set_mode(windowsize, DOUBLEBUF)
  clock = pygame.time.Clock()

  # Uncomment the framerate line and change
  # time = clock.tick() in main loop to
  # time = clock.tick(framerate)
  # to limit the animation to a specific framerate

  #framerate = 30
```

```
  textfont = pygame.font.SysFont(”Arial”,24)

  #
  # Main loop
  #

  endprogram = False
  while not endprogram:
    screen.fill(surfacecolor)
    for theball in balls:
      theball.Show(screen)
    time = clock.tick()
    thetext = textfont.render(str(1000/time), True, (255,0,0),(255,255,0))
    screen.blit(thetext,(0,0))
    time = time / 1000.0
    for theball in balls:
      theball.Move(time)
      if theball.IsOutofX(0,640):
        theball.xspeed = -theball.xspeed
      if theball.IsOutofY(0,480):
        theball.yspeed = -theball.yspeed
    pygame.display.update()
    endprogram = getQuit()

  pygame.quit()
  exit()

if __name__ == ”__main__”:
  main()
```

A .7 Graphics Match

```
#
# Graphics Match
# Adapted from the TI-99/4A original
# Download this source file and game support files
# from http://pygamegr.wordpress.com

import pygame
from pygame.locals import *
from random import randint
from sys import exit

class Spinner:
  def __init__(self,images):
    self.slot=[]
    for image in images:
      self.slot.append(pygame.image.load(image))

  def Spin(self, surface, x, y, framerate, clock):
    for draws in range(0,50):
      self.luck = (randint(0,5),randint(0,5), randint(0,5))
      x1=x
      for i in self.luck:
        surface.blit(self.slot[i], (x1, y))
        x1 = x1 + self.slot[i].get_width() + 3
      pygame.display.update()
      time = clock.tick(framerate)

  def GetScore(self):
    if self.luck[0] == self.luck[1] == self.luck[2]:
```

```
      points = 75
    elif self.luck[0] == self.luck[1]:
      if self.luck[0] in [0,1,2]:
        points = 40
      else:
        points = 10
    elif self.luck[0] == self.luck[2]:
      points = 10
    else:
      points = -10
    return points

#
# Initialize the pygame library
#

pygame.init()

#
# Leftmost image position
#

x = 126.0
y = 100.0

#
# Define a modest window size
#

screenwidth = 640
screenheight = 480
```

```
#
# Load slot images from files and init
# slotmachine, an object of the Spinner class
#

slot_images = ('lemon.jpg','bar.jpg','cherry.jpg','bell.jpg','raspberry.jpg','seven.jpg')
slotmachine = Spinner(slot_images)

#
# Initialize screen and caption
#

screen = pygame.display.set_mode((screenwidth, screenheight), 0, 32)
pygame.display.set_caption("Graphics Match 2012")

#
# Screen color is basically blue
#

surfacecolor = (50,80,250)
screen.fill(surfacecolor)

#
# Setup some fonts and text messages
#

font = pygame.font.SysFont("impact",32)
header_text = font.render("Graphics Match - 2012", True, (255,0,0),(255,255,0))
keys_text = font.render("Press [SPACE] to Play, Q to quit", True, (255,255,255))
```

```
#
# Create a Clock object and set framerate variable for  ticks
#

clock = pygame.time.Clock()
framerate = 25

#
# Show info message
#

screen.blit(header_text,(175,20))
screen.blit(keys_text,(125,350))

#
# Initialize score value and position
# Zero is actually a good score for this game,
# just play for a while and you'll see...
#

score = 0
scorex = 120
scorey = 300

#
# Enter main game loop
#

endgame = False
spacepressed = True
while not endgame:
```

```
  for event in pygame.event.get():
    if event.type == QUIT:
      endgame = True
    if event.type == KEYDOWN:
      keyboardinput = event.key
      if keyboardinput == K_q:
        endgame = True
      if keyboardinput == K_SPACE:
        spacepressed = True

  if spacepressed:
    slotmachine.Spin(screen,x,y,framerate,clock)
    score += slotmachine.GetScore()

  scoretext = font.render("Score:  "+str(score),True,(255,255,255))
  pygame.draw.rect(screen,(255,0,0),(scorex,scorey,400,40))
  screen.blit(scoretext,(scorex,scorey))
  pygame.display.update()
  spacepressed = False

pygame.quit()
exit()
```

A .8 Pygame Invaders

```
#
# Pygame Invaders
# Loosely Adapted from more
# than one TI-99/4A original games
# Final Stage - Invasion has began!
```

```
# Download this source file and game support files
# from http://pygamegr.wordpress.com

import pygame
import os
from pygame.locals import *
from random import randint
from sys import exit

# Center Message on surface, used for blitting info text

def CenterMessage(screen, surface):
  return (screen.get_width() - surface.get_width())/2

# A function for the background music

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

# A function for the sound effects

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

# The superclass for both our ship and the aliens

class Craft(object):
  def __init__ (self, imagefiles, coord):
    self.shape = [pygame.image.load(imagefile) for imagefile in imagefiles]
```

```
    self.ship_width = self.shape[0].get_width()
    self.ship_height = self.shape[0].get_height()
    self.rect = pygame.Rect(coord,(self.ship_width, self.ship_height))
    self.ship_midwidth = self.ship_width / 2
    self.firecolor=(255,0,0)
    self.firespeed = -800
    self.shotlength = 20

  def Show(self, surface,imageindex):
    surface.blit(self.shape[imageindex],(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):
    shot = Laser((self.rect[0]+self.ship_midwidth, self.rect[1]),
                 self.firecolor,self.shotlength,self.firespeed,self.rect[1],15)
    return shot

#
# Alien is used to create AlienShips (really?)
#

class Alien(Craft):
  def __init__(self, imagefile, coord, speed_x, speed_y):
    imagefiles = (imagefile,)
    super(Alien, self).__init__(imagefiles, coord)
    self.speed_x = speed_x
    self.speed_y = speed_y
```

```
    self.shot_height = 10
    self.firebaseline = self.ship_height
    self.firecolor=(255,255,0)
    self.firespeed = 200

  def Move(self, time):
    super(Alien,self).Move(self.speed_x, self.speed_y,time)
    if self.rect[0] >= 440 or self.rect[0] <= 10:
      self.speed_x = -self.speed_x
    if self.rect[1] <= 10 or self.rect[1] >= 440:
      self.speed_y = -self.speed_y

  def Fire(self):
    theshot = Laser((self.rect[0]+self.ship_midwidth,
                     self.rect[1]+self.firebaseline),self.firecolor,
                     self.shot_height,
                     self.firespeed,self.rect[1]+self.firebaseline, 0)
    return theshot

  def Show(self, surface):
    imageindex = 0
    super(Alien,self).Show(surface,imageindex)

# The class for our ship

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

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

  def Fire(self):
    self.lasersound.play()
    return super(SpaceCraft,self).Fire()

# This is the class for the scrolling background

class SpaceBackground:
  def __init__(self, screenheight, imagefile):
    self.shape = pygame.image.load(imagefile)
    self.coord = [0,0]
    self.coord2 = [0, -screenheight]
    self.y_original = self.coord[1]
    self.y2_original = self.coord2[1]

  def Show(self, surface):
    surface.blit(self.shape, self.coord)
    surface.blit(self.shape, self.coord2)

  def Scroll(self, speed_y, time):
    distance_y = speed_y * time
    self.coord[1] += distance_y
    self.coord2[1] += distance_y
```

```
    if self.coord2[1] >= 0:
      self.coord[1] = self.y_original
      self.coord2[1] = self.y2_original

#
# Laser class
#

class Laser:
  def __init__(self, coord, color, size, speed, refline, voffset):
    self.x1 = coord[0]
    self.y1 = coord[1] + voffset
    self.size = size
    self.color = color
    self.speed = speed
    self.refline = refline

  def Show(self, surface):
    pygame.draw.line(surface, self.color, (self.x1, self.y1),(self.x1,self.y1-self.size),3)

  def Move(self, time):
    distance = self.speed * time
    self.y1 += distance

  def DistanceTravelled(self):
    return abs(self.refline - self.y1)

  def GoneAbove(self,y):
    if self.y1<=y:
      return True
    else:
```

```
      return False

  def GoneBelow(self,y):
    if self.y1>=y:
      return True
    else:
      return False

  def GetXY(self):
    return (self.x1, self.y1)

#
# ScoreBoard - Score keeping and display
#

class ScoreBoard:
  def __init__(self,x,y,font,fontsize):
    self.x = x
    self.y = y
    self.font = pygame.font.SysFont(font,fontsize)
    self.score = 0

  def Change(self, amount):
    self.score += amount

  def Show(self,surface):
    scoretext = self.font.render("Score: "+str(self.score), True, (0,0,255))
    surface.blit(scoretext,(self.x, self.y))

  def GetValue(self):
    return self.score
```

```
  def SetValue(self, score):
    self.score = score

#
# ShieldMeter - Keep and display vintage style bar shield meter
#

class ShieldMeter:
  def __init__(self, x, y, maxvalue, warnvalue):
    self.x = x
    self.y = y
    self.maxvalue = maxvalue
    self.currentvalue = maxvalue
    self.warnvalue = warnvalue

  def Show(self, surface):
    if self.currentvalue < self.warnvalue:
      self.shieldcolor = (255,0,0)
    else:
      self.shieldcolor = (0,255,0)
    pygame.draw.rect(surface,self.shieldcolor,(self.x, self.y, self.currentvalue,25))

  def Increase(self, amount):
    self.currentvalue += amount
    if self.currentvalue > self.maxvalue:
      selfcurrentvalue = self.maxvalue

  def Decrease(self, amount):
    self.currentvalue -= amount
    if self.currentvalue < 0:
```

```
      self.currentvalue = 0

  def GetValue(self):
    return self.currentvalue

  def SetValue(self,value):
    self.currentvalue = value
    if self.currentvalue > self.maxvalue:
      self.currenvalue = self.maxvalue
    if self.currentvalue < 0:
      self.currentvalue = 0

#
# Simple function to display message for Game Over
#

def GameOverShow(screen):
  font = pygame.font.SysFont("impact", 32)
  gameovertext = font.render("Game Over!",True,(255,255,255))
  text_x = CenterMessage(screen, gameovertext)
  screen.blit (gameovertext,(text_x,280))
  gameovertext = font.render("Press R to Restart", True, (255,255,255))
  text_x = CenterMessage(screen, gameovertext)
  screen.blit(gameovertext,(text_x,320))
  return

#
# Main function, program entry point
#

def main():
```

```
# Initialize pygame library and OS environment
#The following will center the game window on the screen

os.environ['SDL_VIDEO_CENTERED']='1'
pygame.init()

# Screen size and initial spaceship position

screenwidth,screenheight = (480,640)
spaceship_pos = (240, 540)

# Initialize screen and set caption

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

# Set keyboard repeat to super fast / delay to minimum

pygame.key.set_repeat(1,1)

# Prepare Sound Effects

laser = PrepareSound("shoot.wav")
explosion = PrepareSound("invaderkilled.wav")
destroyed = PrepareSound("explosion.wav")

# Initialize the background and spaceship objects

spaceship_low = (0,0)
spaceship_high = (screenwidth, screenheight)
StarField = SpaceBackground(screenheight, "stars.jpg")
```

```
shipimages = ('spaceship2.png', 'spaceship3.png')
SpaceShip = SpaceCraft(shipimages,spaceship_pos, spaceship_low, spaceship_high,laser)

# Initialize fire lists (ours and aliens)

firelist=[]
alienfirelist = []

# Set the background scrolling speed

backspeed = 100

# Initialize some objects and game variables

score = ScoreBoard(0,0,"impact",32)
shield = ShieldMeter(200,10,250,75)
laserdownlimit = screenheight - 40
GameOver = False

# Start the background music

PlayMusic("spaceinvaders.ogg")

# Images used for the aliens

alienimage=('alien1.png','alien2.png','alien3.png','alien4.png','alien5.png')

# Number of aliens per 'wave'

numofaliens = 8
AlienShips = []
```

```
# Initialize the clock object and set framerate

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

# imageindex & flashcount used for alternating between normal
# and 'hit' images

imageindex = 0
flashcount = 0

while True:
  time = clock.tick(framerate)/1000.0

  if not AlienShips:
    # AlienShips empty means we need to create new 'wave'
    AlienShips = [ Alien(alienimage[randint(0,len(alienimage)-1)],
                         [randint(20,screenwidth-80),
                          randint(20,screenheight-140)],randint(100,150),
                         randint(100,150)) for i in range(0,numofaliens) ]

  # shipspeed is zeroed at every cycle of the loop
  shipspeed_x = 0
  shipspeed_y = 0
  for event in pygame.event.get():
    if event.type == QUIT:
      pygame.quit()
      exit()

    if event.type == USEREVENT + 1:
```

```
      if flashcount < 10:
        flashcount += 1
        if imageindex == 1:
          imageindex = 0
        else:
          imageindex = 1
      else:
        imageindex = 0
        flashcount = 0
        pygame.time.set_timer(USEREVENT+1,0)

    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_r] and GameOver:
        GameOver = False
        shield.SetValue(250)
        score.SetValue(0)
      if key[K_LEFT]:
        shipspeed_x = -300
      if key[K_RIGHT]:
        shipspeed_x = 300
      if key[K_UP]:
        shipspeed_y = -300
      if key[K_DOWN]:
        shipspeed_y = 300
```

```
    if key[K_SPACE] and not GameOver:
      if firelist:
        # Only fire if last shot has travelled
        # a minimum distance
        if firelist[-1].DistanceTravelled() >=  150:
          firelist.append(SpaceShip.Fire())
      else:
        # or if there is no shot
        firelist.append(SpaceShip.Fire())


# Move the SpaceShip by the specified amount

SpaceShip.Move(shipspeed_x, shipspeed_y, time)

# Show all the objects, background first (or it will erase everything else!)
StarField.Scroll(backspeed, time)
StarField.Show(screen)
SpaceShip.Show(screen,imageindex)

# Show and move the aliens

for AlienShip in AlienShips:
  AlienShip.Show(screen)
  AlienShip.Move(time)
  if randint(0,10)==9:
    if alienfirelist:
      if alienfirelist[-1].DistanceTravelled()>=100:
        alienfirelist.append(AlienShip.Fire())
    else:
      alienfirelist.append(AlienShip.Fire())
```

```
for theshot in firelist:
  theshot.Move(time)
  theshot.Show(screen)
  if theshot.GoneAbove(0):
      firelist.remove(theshot)
  else:
    for AlienShip in AlienShips:
      if AlienShip.rect.collidepoint(theshot.GetXY()):
        score.Change(10)
        explosion.play()
        if score.GetValue() % 100 == 0:
          shield.Increase(25)
        if theshot in firelist:
          firelist.remove(theshot)
        AlienShips.remove(AlienShip)

for theshot in alienfirelist:
  theshot.Move(time)
  theshot.Show(screen)
  if theshot.GoneBelow(laserdownlimit):
    alienfirelist.remove(theshot)
  else:
    if SpaceShip.rect.collidepoint(theshot.GetXY()) and not GameOver:
      destroyed.play()
      pygame.time.set_timer(USEREVENT+1,25)
      shield.Decrease(25)
      if theshot in alienfirelist:
        alienfirelist.remove(theshot)

score.Show(screen)
```

```
    shield.Show(screen)
    if shield.GetValue() == 0:
      GameOverShow(screen)
      GameOver = True

    pygame.display.update()

# End main loop

# Start program
if __name__ == ”__main__”:
    main()
```