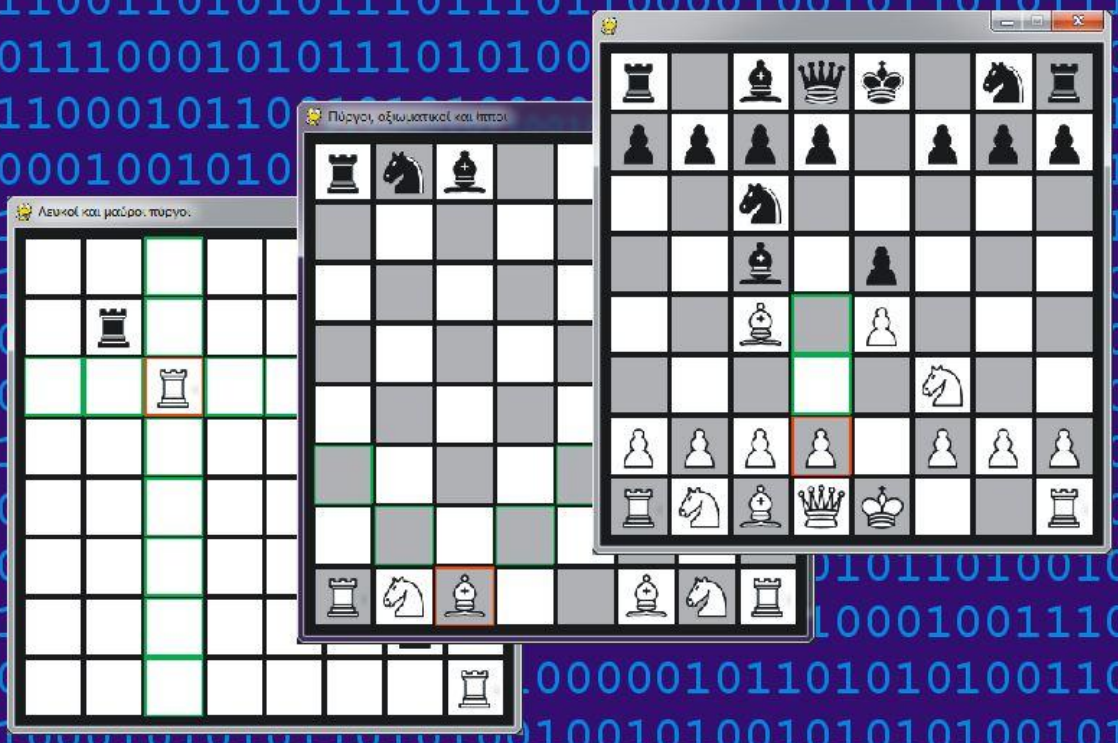
ΓΙΩΡΓΟΣ Ι. ΨΑΡΑΚΗΣ

ΠΡΟΓΡΑΜΜΑΤΙΖΟΝΤΑΣ ΕΝΑ ΣΚΑΚΙ

Βήμα βήμα προς ένα αήττητο πρόγραμμα σκακιού με τη βοήθεια της εύκολης γλώσσας προγραμματισμού Python



ΣΥΝΙΣΤΑΤΑΙ ΚΑΙ ΓΙΑ ΑΡΧΑΡΙΟΥΣ ΣΤΟΝ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟ

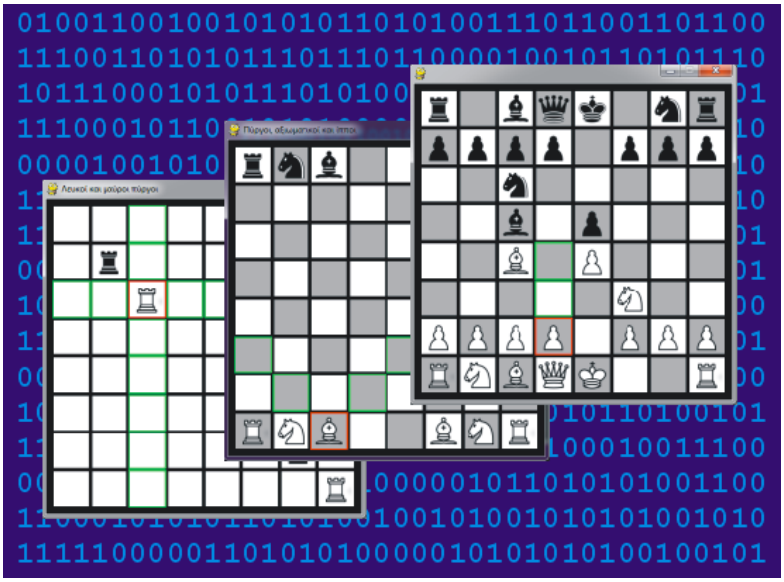
ψαρακι

e-book

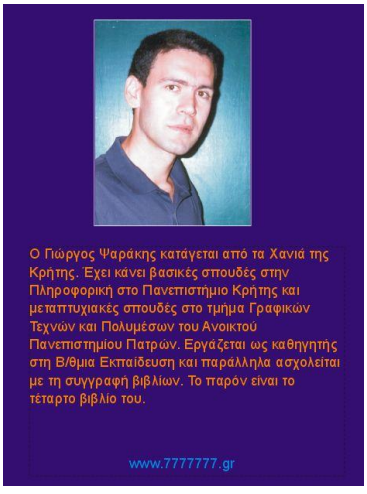
ΓΙΩΡΓΟΣ Ι. ΨΑΡΑΚΗΣ

ΠΡΟΓΡΑΜΜΑΤΙΖΟΝΤΑΣ ΕΝΑ ΣΚΑΚΙ

Βήμα βήμα προς
ένα αήττητο πρόγραμμα σκακιού
με τη βοήθεια της εύκολης
γλώσσας προγραμματισμού Python



Τίτλος βιβλίου: ΠΡΟΓΡΑΜΜΑΤΙΖΟΝΤΑΣ ΕΝΑ ΣΚΑΚΙ
Συγγραφέας: ΓΙΩΡΓΟΣ Ι. ΨΑΡΑΚΗΣ
Ιστοσελίδα βιβλίου: www.7777777.gr
ISBN: 978-960-88910-2-9



Άλλα βιβλία του ίδιου συγγραφέα:
ΘΗΣΑΥΡΟΣ ΠΡΟΒΛΗΜΑΤΩΝ, ΛΙΒΑΝΗ 2003, ISBN 978-960-14-0449-3
ΜΙΚΡΟΙ ΕΠΑΝΑΣΤΑΤΕΣ, ΨΑΡΑΚΙ 2005, ISBN 978-960-88910-0-0
ΘΗΣΑΥΡΟΣ ΓΡΙΦΩΝ, ΨΑΡΑΚΙ 2007, ISBN 978-960-88910-1-2

ψαρακι

ΔΗΜΙΟΥΡΓΙΕΣ ΕΝΤΥΠΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΗΣ ΜΟΡΦΗΣ

ΧΑΝΙΑ 2014

ΠΕΡΙΕΧΟΜΕΝΑ

ΕΥΡΕΤΗΡΙΟ ΣΕΛΙΔΩΝ ΓΙΑ ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ

ΕΙΣΑΓΩΓΗ

Ο σκοπός του βιβλίου είναι διπλός. Από τη μία επιχειρείται να καλυφθεί μια πραγματική ανάγκη στον τομέα του προγραμματισμού, που έχει να κάνει με την απαίτηση για ένα πολύ πρακτικό εγχειρίδιο, το οποίο ταχύτατα θα εισάγει κάθε ενδιαφερόμενο στον τομέα του οπτικού προγραμματισμού. Από την άλλη επιχειρείται να παρουσιαστεί ένας απλός και μεθοδικός τρόπος με τον οποίο τελικά καταλήγουμε να προγραμματίσουμε ένα αήττητο πρόγραμμα σκακιού.

Ήταν μεγάλη πρόκληση, να συνδυαστεί μια πλήρης και συστηματική μέθοδος εκμάθησης παράλληλα με απλότητα και συντομία στη διατύπωση, ώστε να επιτυγχάνονται γρήγορα αποτελέσματα απ' τους αναγνώστες. Είναι βέβαιο ότι ήταν δύσκολη αυτή η ισορροπία και μένει να κριθεί από τους αναγνώστες, αν κάτι τέτοιο επετεύχθη. Επιλέχθηκε η γλώσσα προγραμματισμού Python, η οποία είναι σχετικά απλή στην εκμάθησή της αλλά παράλληλα διαθέτει πανίσχυρες δυνατότητες δημιουργίας.

Τέλος, πρέπει να τονίσω ως προγραμματιστής την ανάγκη να είναι κανείς όσο γίνεται πιο ενημερωμένος σε θέματα προγραμματισμού. Η ανθρωπότητα έχει αποφασίσει να βαδίσει έναν «επικίνδυνο» τεχνολογικό δρόμο, όπου η σημασία του προγραμματισμού θα είναι κορυφαία. Εκτιμώ ότι σε σύντομο χρονικό διάστημα θα τρομάξουμε κυριολεκτικά από τις μαγικές δυνατότητες που ξεδιπλώνονται με εκθετικό ρυθμό στη ζωή μας, εξαιτίας ακριβώς του προγραμματισμού. Το λογισμικό (software) αποτελεί το αίμα και τα γονίδια των σύγχρονων και μελλοντικών μηχανών και δικτύων. Το λεγόμενο διαδίκτυο αντικειμένων είναι προ των πυλών, κάτι που σημαίνει ότι όλα αποκτούν ένα προγραμματιζόμενο τσιπ ελέγχου. Η μίξη βιολογίας, πληροφορικής και τεχνολογίας ίσως προκαλέσει ακόμα και νέους ανθρωπότυπους! Αυτές και άλλες μελλοντικές προκλήσεις, οφείλουμε να τις αντιμετωπίσουμε όσο γίνεται πιο ενημερωμένοι.

Ο συγγραφέας

Γιώργος Ψαράκης

Το βιβλίο αφιερώνεται στον ορθολογισμό

και σε όλους τους σκακιστές.

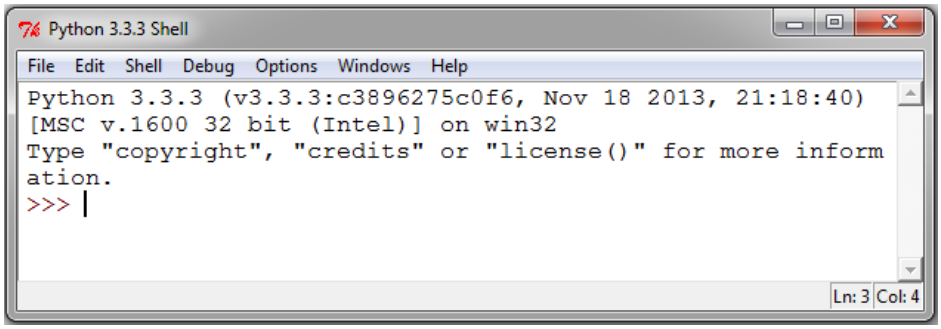
ΚΕΦΑΛΑΙΟ 0, Εγκατάσταση του περιβάλλοντος εργασίας

Μόνο δύο αρχεία είναι απαραίτητα να κατεβούν από το Διαδίκτυο, ώστε να μπορεί να ξεκινήσει κανείς να μελετά με αυτό το βιβλίο. Πρόκειται πρώτον για το αρχείο εγκατάστασης της γλώσσας προγραμματισμού Python σε περιβάλλον Windows και δεύτερον για το αρχείο επέκτασης των δυνατοτήτων της γλώσσας Python, το λεγόμενο Pygame.

Είναι απαραίτητο οι εκδόσεις Python και Pygame να είναι συνεργάσιμες μεταξύ τους, συμβατές όπως λέγεται. Στο βιβλίο γίνεται χρήση Python 3.3 και Pygame 1.9 των οποίων τα αντίστοιχα αρχεία μπορούν να κατεβούν από την ιστοσελίδα www.7777777.gr. Μπορούν επίσης να αναζητηθούν και στις ιστοσελίδες python.org, pygame.org, bitbucket.org/pygame/pygame/downloads ή και 1lyk-kissam.chan.sch.gr/downloads.

Μετά το κατέβασμα (download) των δύο αρχείων, εγκαθιστούμε πρώτα την Python και θα δημιουργηθεί ο φάκελος C:/Python33. Εγκαθιστούμε στη συνέχεια το Pygame, προσέχοντας να δηλώσουμε τον ίδιο φάκελο εγκατάστασης, δηλαδή C:/Python33 αντί για C:/PythonX που προτείνει η εγκατάσταση.

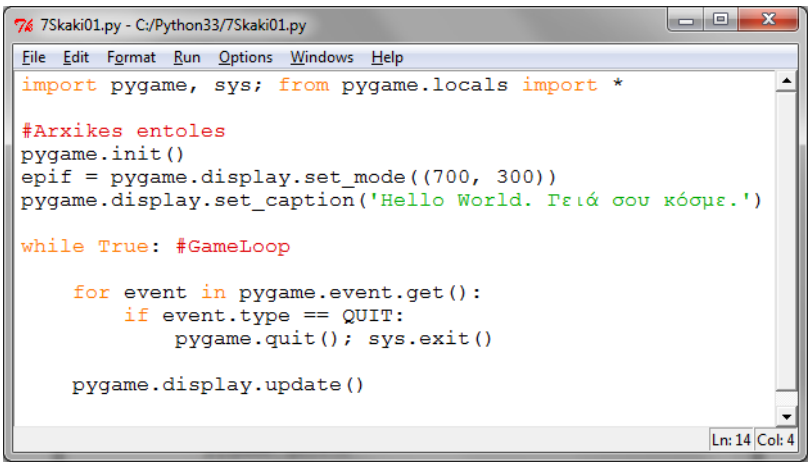
Αν γίνουν όλα σωστά, στο μενού Έναρξη/Όλα τα προγράμματα θα έχει δημιουργηθεί η επιλογή φακέλου Python 3.3. Με κλικ σε αυτόν εμφανίζεται υπομενού, και μας ενδιαφέρει πρωτίστως η επιλογή IDLE (Python GUI) με την οποία θα ανοίγει το περιβάλλον εργασίας μας. Με κλικ σε αυτήν, ανοίγει το λεγόμενο κέλυφος της Python (π.χ. Python 3.3.3 Shell).



Δίπλα στο σύμβολο προτροπής >>>, ο κατακόρυφος κέρσορας που αναβοσβήνει περιμένει τις εντολές μας. Π.χ. γράφοντας 3+4 και πατώντας Enter, θα εμφανιστεί το αποτέλεσμα 7. Όμως, δεν θα μας απασχολήσουν προς το παρόν οι εντολές κελύφους, ζήτημα που θα εξεταστεί σε επόμενα κεφάλαια κάθε φορά που θα κριθεί σκόπιμο. Μας ενδιαφέρει περισσότερο να έχουμε γρήγορα αποτελέσματα σε γραφικά. Γι' αυτό το σκοπό, από το μενού File/New window..., θα ανοίγουμε ένα δεύτερο παράθυρο, μέσα στο οποίο θα γράφουμε τα προγράμματά μας.

ΚΕΦΑΛΑΙΟ 1, Το πρώτο μας πρόγραμμα

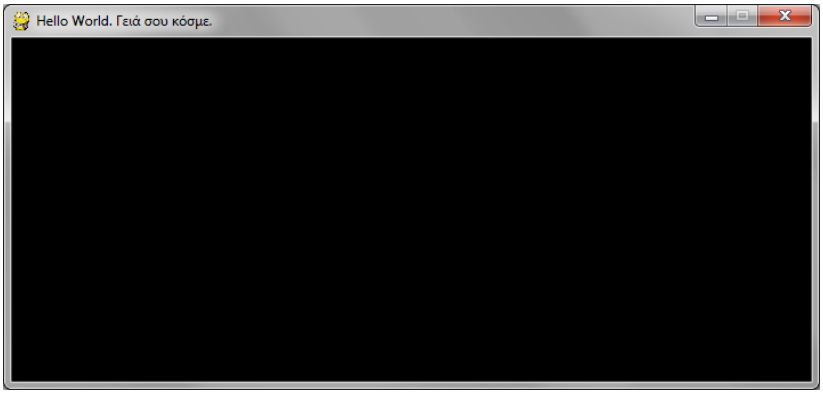
Πληκτρολογούμε τις παρακάτω εντολές, όχι στο κέλυφος αλλά σε νέο παράθυρο της Python που ανοίγουμε από File/New window... .



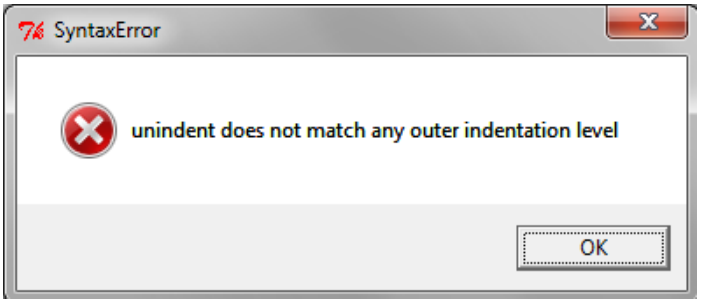
Να δοθεί προσοχή, ώστε στις γραμμές που ξεκινούν πιο δεξιά όπως η γραμμή με την εντολή for, η εσοχή να περιλαμβάνει ακριβώς τέσσερις κενούς χαρακτήρες (spacebar). Αντί για τέσσερα κενά συνιστάται να πατά κανείς το πλήκτρο Tab. Παρόμοια, η γραμμή με την εντολή if περιλαμβάνει αρχικά δύο Tab, ή αλλιώς οκτώ κενούς χαρακτήρες κτλ. Στην γλώσσα προγραμματισμού Python, αυτό το ζήτημα των Tab και των εσοχών είναι κομβικής σημασίας, πρέπει να τηρείται με ακρίβεια και θα εξηγηθεί περισσότερο αργότερα.

Αποθηκεύουμε το πρώτο πρόγραμμά μας με διαδικασία παρόμοια με αυτήν της αποθήκευσης π.χ. ενός αρχείου του κειμενογράφου Word. Από το οριζόντιο μενού επιλογών, επιλέγουμε File/Save… Στο παράθυρο αποθήκευσης που ανοίγει, πρέπει να δοθούν τρία στοιχεία: το όνομα αρχείου, ο τύπος του και η θέση στο σκληρό δίσκο που επιθυμούμε να αποθηκευτεί. Ως όνομα αρχείου δίνουμε π.χ. 7skaki01.py. Ο τύπος αρχείου ορίζεται αυτόματα και είναι Python File, χαρακτηρίζεται από την επέκταση .py και από το διπλανό σύμβολο. Ως θέση αποθήκευσης ορίζουμε το φάκελο εγκατάστασης της Python, δηλαδή C:/Python33.

Και τώρα ήρθε η ώρα να εκτελέσουμε το πρώτο μας πρόγραμμα! Να το τρέξουμε όπως λέγεται. Από το οριζόντιο μενού επιλέγουμε Run/Run Module... ή ισοδύναμα πατάμε F5. Αν δούμε να ανοίγει ένα νέο παράθυρο με μαύρη οθόνη όπως παρακάτω, τότε όλα πληκτρολογήθηκαν καλά. Η εκτέλεση προκάλεσε να ανοιχθεί και τρίτο παράθυρο, στο οποίο παρουσιάζεται η εκτέλεση του προγράμματος.



Αντιθέτως, αν δούμε στην οθόνη μας μήνυμα σαν το παρακάτω, τότε πατάμε OK και επανελέγχουμε την ορθότητα των γραμμών κώδικα που πληκτρολογήσαμε.



Όπως ήδη προαναφέρθηκε, απαιτείται απόλυτη ακρίβεια στις εσοχές!

Ας εξετάσουμε μία μία τις εντολές του κώδικα. Πρώτα απ' όλα, παρατηρούμε τις κόκκινες εντολές, οι οποίες όλες ξεκινούν με τον χαρακτήρα της δίεσης #. Όταν σε μια γραμμή κώδικα παρουσιαστεί ο χαρακτήρας #, κάθε τι μετά από αυτό αποτελεί σχόλιο του προγραμματιστή για το συγκεκριμένο τμήμα εντολών. Έχει αποδειχθεί χρήσιμο να σχολιάζουμε σύντομα την λειτουργία των εντολών μας. Βεβαίως, τα σχόλια δεν αποτελούν εντολή προς εκτέλεση από τον υπολογιστή.

Οι αρχικές εντολές **`import`** και **`from`** εισάγουν/συνδέουν κάποια χρήσιμα προγραμματιστικά στοιχεία από τα τμήματα pygame και sys της γλώσσας προγραμματισμού Python. Θα τις χρησιμοποιούμε αρχικά στα τυφλά σε όλα τα προγράμματα μας και με τον καιρό θα εμβαθύνουμε.

Η εντολή **`pygame.init()`** καθιστά έτοιμο το πρόγραμμά μας να εκτελεί λειτουργίες του τμήματος pygame, λειτουργίες που αφορούν ισχυρές δυνατότητες γραφικών, ήχου κ.α.

Η εντολή **`pygame.display.set_mode((700, 300))`** δημιουργεί το παράθυρο εκτέλεσης. Στις παρενθέσεις παρατηρούμε το ζευγάρι αριθμών (700, 300). Αυτό καθορίζει το πλάτος και το ύψος του παραθύρου εκτέλεσης σε εικονοστοιχεία (pixel). Αριστερά του συμβόλου = χρησιμοποιούμε μια μεταβλητή όπως λέγεται, συγκεκριμένα τη μεταβλητή epif, η οποία αντιπροσωπεύει το παράθυρο εκτέλεσης. Ή αλλιώς την επιφάνεια σχεδίασης που δημιουργήθηκε στο δεξί σκέλος. Θα μπορούσε να απουσιάζει αυτή η μεταβλητή και ο χαρακτήρας = και θα αρκούσε να γράψουμε **`pygame.display.set_mode((700, 300))`**. Όμως σε επόμενα παραδείγματα όπου πρέπει να γίνουν σχεδιαστικές επεμβάσεις στην αρχική επιφάνεια, θα φανεί ιδιαίτερα χρήσιμο να αναφερόμαστε σε αυτήν με κάποιο όνομα. Στην περίπτωσή μας επιλέξαμε το όνομα epif, ως αρχικά λατινικά γράμματα της λέξης επιφάνεια, όμως θα μπορούσαμε να επιλέξουμε οποιοδήποτε άλλο όνομα με λατινικά γράμματα.

Με την εντολή **`pygame.display.set_caption('Hello World.Γεια σου κόσμε.')`** δηλώνουμε τι θέλουμε να εμφανίζεται στον τίτλο του παραθύρου εκτέλεσης. Παρατηρούμε ότι μέσα στις παρενθέσεις της εντολής, πρέπει να δηλωθεί ένα όρισμα, συγκεκριμένα ένα κείμενο, το οποίο μπορεί να είναι και μικτό αγγλοελληνικό. Τα κείμενα οριοθετούνται με αποστρόφους ('), και στο πληκτρολόγιο βρίσκονται συνήθως αριστερά του πλήκτρου Enter.

Στις προηγούμενες τρεις εντολές υπάρχει κάτι κοινό. Όλες εκκινούν με το πρόθεμα **pygame** ακολουθούμενο από τελεία. Αυτό απλά σημαίνει ότι όλες ανήκουν στις εντολές του πακέτου pygame. Παρατηρούμε ότι και οι τρεις εντολές έχουν στο τέλος τους παρενθέσεις μέσα στις οποίες μπαίνουν τα κατάλληλα ορίσματα, ή αλλιώς οι κατάλληλες παράμετροι. Ειδικά η εντολή **init** δεν παίρνει παραμέτρους. Η εντολή **display.set_mode** παίρνει ως όρισμα ένα ζευγάρι τιμών, στην περίπτωσή μας το ζευγάρι (700,300) και γι αυτό το λόγο παρατηρούνται διπλές παρενθέσεις. Η εντολή **display.set_caption** παίρνει ως όρισμα μέσα στις παρενθέσεις ένα κείμενο σε αποστρόφους.

Η εντολή **while True:** ακολουθείται πάντα από ένα μπλοκ εντολών γραμμένων σε εσοχή, και δημιουργεί μια αστραπιαία και αέναη εκτέλεση των εντολών αυτών (έως και χιλιάδες φορές το δευτερόλεπτο). Όλες αυτές οι εντολές πρέπει να γραφούν σε εσοχή και αφορούν την αλληλεπίδραση δεδομένων ποντικιού, πληκτρολογίου και αντίστοιχης ανταπόκρισης στην οθόνη, δηλαδή στο παράθυρο εκτέλεσης. Ας εξετάσουμε την παρακάτω εντολή.

```
for event in pygame.event.get():
    if event.type == QUIT:
        pygame.quit(); sys.exit()
```

Η εντολή **for event**… θα ελέγχει συνεχώς για συμβάντα ποντικιού ή πληκτρολογίου. Εδώ, με την ακόλουθη **if event.type == QUIT:**, ελέγχεται τουλάχιστον ενέργεια τερματισμού του παραθύρου εκτέλεσης, αλλά αργότερα θα ελέγχονται πιθανές άλλες ενέργειες του χρήστη, όπως κλικ ποντικιού ή πληκτρολόγηση. Αργότερα θα εξηγηθεί το τμήμα αυτό περισσότερο.

Στις εντολές σε εσοχή της **while True:**, θα υπάρχει σχεδόν πάντα και η εντολή **pygame.display.update()**, η οποία στέλνει στην οθόνη τα επίκαιρα δεδομένα που σχεδιάστηκαν. Ειδικά σε αυτό το πρώτο μας πρόγραμμα, το οποίο είναι στατικό και όπου δηλαδή δεν υπάρχει αλληλεπίδραση, μπορεί και να απουσιάζει.

Όλες οι εντολές σε εσοχή της εντολής **while**, περιγράφονται και ως GameLoop, και όπως θα δούμε στη συνέχεια, αυτές θα είναι και οι πρωταγωνιστικές. Οτιδήποτε πριν την εντολή **while**, θα εκτελεστεί μόνο μια φορά. Αντίθετα, οι εντολές μετά την **while**, θα εκτελούνται ξανά και ξανά.

Κλείνουμε το παράθυρο εκτέλεσης από το κόκκινο κουμπί πάνω δεξιά. Έχουμε στο μυαλό ότι αυτό το πρώτο πρόγραμμα, θα χρησιμεύσει ως ο σκελετός για το επόμενο. Με μικρές αλλαγές θα δημιουργείται το δεύτερο πρόγραμμά μας. Παρόμοια, από το δεύτερο πρόγραμμα με μικρές αλλαγές θα δημιουργείται το τρίτο κτλ. Καλό είναι λοιπόν, με την επιλογή File/Save As... να δημιουργούμε ακριβή αντίγραφα από τα προγράμματά μας, να τα τροποποιούμε κατάλληλα και να προχωράμε. Εξάλλου, έτσι μπορούμε να δημιουργούμε αντίγραφα και να πειραματιζόμαστε άφοβα σε αυτά. Για παράδειγμα, αποθηκεύουμε το προηγούμενο πρόγραμμα και ως test.py, αλλάζουμε το κείμενο του τίτλου με την εντολή **pygame.display.set_caption** και το τρέχουμε! Ή αλλάζουμε π.χ. τις διαστάσεις ύψους και πλάτους του παραθύρου εκτέλεσης. Ή αλλάζουμε π.χ. την ονομασία της μεταβλητής epif με μια άλλη.

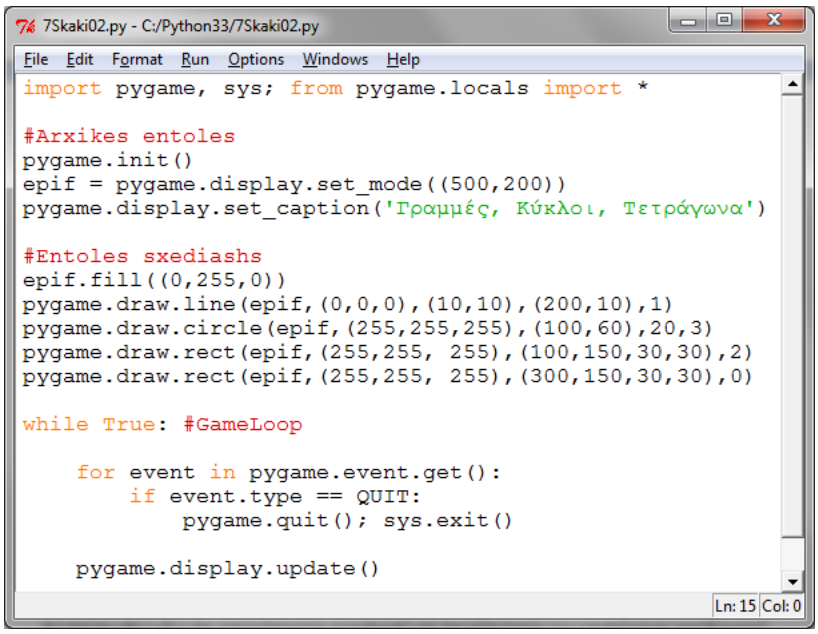
Πρέπει να προειδοποιηθεί ότι θα συναντήσουμε αρκετές αγγλικές λέξεις καθώς θα μελετάμε αυτό το βιβλίο. Προϋποτίθεται λοιπόν, όχι μόνο να γνωρίζουμε τα αγγλικά

γράμματα, αλλά να γνωρίζουμε και μερικές λέξεις όπως for, if, exit, quit, get, κάτι που θα βοηθήσει αρκετά στην κατανόηση του κώδικα. Γενικότερα, όλες οι γλώσσες προγραμματισμού χρησιμοποιούν τέτοιες λέξεις στη σύνταξή τους, ώστε ο κώδικας να γίνεται πιο προσιτός, λιγότερο αφηρημένος και κατά συνέπεια ευκολότερος στην εκμάθηση, στη διόρθωση και στη μελλοντική του επέκταση. Στο περιβάλλον που θα εργαστούμε, μερικές τέτοιες λέξεις έχουν πορτοκαλί χρωματισμό, διότι ανήκουν στις λεγόμενες λέξεις κλειδιά (keywords) της γλώσσας Python.

Και κάτι ακόμα. Αν θεωρεί κάποιος προχωρημένος αναγνώστης ότι χάνει πολύτιμο χρόνο πληκτρολογώντας τα προγράμματα του βιβλίου, μπορεί και να τα κατεβάσει από την ιστοσελίδα www.7777777.gr. Όμως αυτό δεν συνιστάται καθόλου στους νέους προγραμματιστές, οι οποίοι όσο περισσότερο κώδικα πληκτρολογήσουν, τόσο περισσότερο και θα ωφεληθούν, κυρίως από τα λάθη τους στην πληκτρολόγηση! Ίσως σε κάποιον δεν τρέχει το πρόγραμμα και δεν μπορεί να εντοπίσει τα λάθη που έχει κάνει στην πληκτρολόγηση, οπότε πάλι είναι χρήσιμο να προστρέξει στο έτοιμο και να το συγκρίνει με το δικό του. Το κυριότερο πάντως είναι, να κατανοούμε καλά κάθε πρόγραμμα που θα μελετάμε και σε αυτό βοηθάει πολύ η πληκτρολόγηση. Ξανατονίζεται ότι ακόμα και τα λάθη που θα κάνουμε έτσι, θα μας διδάξουν πολλά σε σχέση με τη σύνταξη που πρέπει να ακολουθείται στην Python. Ανεξαιρέτως, όλες οι γλώσσες προγραμματισμού απαιτούν κανόνες ορθής γραφής και αυστηρής σύνταξης των εντολών τους.

ΚΕΦΑΛΑΙΟ 2, Γραμμές κύκλοι, τετράγωνα, χρώματα, συντεταγμένες

Ήρθε η ώρα να προγραμματίσουμε κάτι πιο ενδιαφέρον, κατά συνέπεια να μάθουμε περισσότερα για τον προγραμματισμό στην Python, χρήσιμα μετέπειτα για το σκάκι. Γράφουμε τον παρακάτω κώδικα σε νέο αρχείο Python, είτε τροποποιούμε κατάλληλα το πρώτο πρόγραμμα. Αποθηκεύουμε π.χ. ως 7skaki02.py.



Τρέχουμε τον κώδικα είτε με Run/Run Module, είτε με πιο σύντομο τρόπο πατώντας το πλήκτρο συντόμευσης F5. Πρέπει να εμφανιστεί στην οθόνη κάτι σαν το παρακάτω:

Ας δούμε πώς έγινε αυτό. Στις αρχικές εντολές, σε σχέση με το πρώτο πρόγραμμα, παρατηρούμε κάποιες μικρές διαφορές στα ορίσματα. Στην εντολή **`pygame.display.set_mode()`** ορίσαμε άλλες διαστάσεις πλάτους και ύψους παραθύρου εκτέλεσης, συγκεκριμένα 500 και 200 αντί για 700 και 300 του πρώτου προγράμματός μας. Και στην εντολή **`pygame.display.set_caption()`** ορίσαμε άλλο κείμενο τίτλου, συγκεκριμένα 'Γραμμές, Κύκλοι, Τετράγωνα'. Στη συνέχεια παρατηρούμε πέντε νέες εντολές:

```
epif.fill((0,255,0))
pygame.draw.line(epif,(0,0,0),(10,10),(200,10),1)
pygame.draw.circle(epif,(255,255,255),(100,60),20,3)
pygame.draw.rect(epif,(255,255, 255),(100,150,30,30),2)
pygame.draw.rect(epif,(255,255, 255),(300,150,30,30),0)
```

Η πρώτη από αυτές γεμίζει με πράσινο χρώμα την επιφάνεια του παραθύρου εκτέλεσης, η δεύτερη σχεδιάζει μια ευθεία μαύρη γραμμή, η τρίτη έναν λευκό κύκλο, η τέταρτη ένα λευκό τετράγωνο και η πέμπτη ένα γεμισμένο λευκό τετράγωνο.

Πριν εξηγηθούν οι εντολές αυτές, είναι ανάγκη σε αυτό το σημείο να γίνει μια παρένθεση και να κατανοήσουμε καλά δύο ζητήματα: το ένα αφορά τους χρωματισμούς με τους οποίους σχεδιάζουμε, ενώ το άλλο αφορά τις θέσεις στις οποίες σχεδιάζουμε.

ΧΡΩΜΑΤΑ

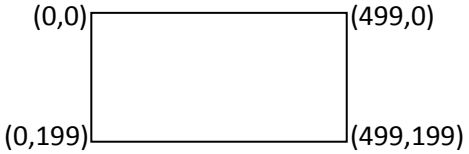
Αν εξετάσουμε προσεκτικά τα ορίσματα στις παραπάνω εντολές, θα διαπιστώσουμε ότι το ένα από αυτά είναι πάντα μια τριάδα αριθμών π.χ. (0, 255, 0). Αυτή η τριάδα καθορίζει ακριβώς το αναπαριστάμενο χρώμα, ανάλογα με την τιμή που έχουν οι τρεις αριθμοί. Οι τιμές κυμαίνονται από 0 έως και 255. Η πρώτη τιμή καθορίζει τη συμμετοχή του κόκκινου, η δεύτερη καθορίζει τη συμμετοχή του πράσινου και η τρίτη καθορίζει τη συμμετοχή του μπλε στον τελικό χρωματισμό. Έτσι, αντιλαμβανόμαστε ότι η τριάδα (0, 255, 0) σημαίνει πράσινο, ενώ οι τριάδες (0, 0, 0) και (255, 255, 255) σημαίνουν μαύρο και λευκό αντίστοιχα (ως γνωστόν η σύνθεση όλων των χρωμάτων δημιουργεί λευκό ενώ η απουσία όλων δημιουργεί μαύρο). Αυτό το σύστημα αναπαράστασης χρωμάτων, που ονομάζεται και σύστημα RGB από τα αρχικά των λέξεων Red Green Blue), χρησιμοποιείται ευρύτατα στον προγραμματισμό και θα το γνωρίσουμε καλύτερα με την εμπειρία μας. Παρακάτω βλέπουμε τους βασικούς χρωματισμούς και τις αντίστοιχες τριάδες τους.

(255,0,0)
(0,255,0)
(0,0,255)
(255,255,255)
(0,0,0)
(177,177,177)

ΘΕΣΗ ΣΧΕΔΙΑΣΗΣ

Στην εντολή **`pygame.draw.line(epif, (0, 0, 0), (10, 10), (200, 10), 1)`** που σχεδιάζει τη μαύρη γραμμή, παρατηρούμε πέντε ορίσματα: Με το πρώτο όρισμα, δηλαδή τη μεταβλητή epif, δηλώνεται σε ποια επιφάνεια θα σχεδιαστεί η γραμμή, με το δεύτερο όρισμα δηλώνεται το χρώμα, και με το πέμπτο όρισμα δηλώνεται το πάχος της γραμμής σε pixel. Παρατηρούμε ως τρίτο και τέταρτο όρισμα δύο ζευγάρια αριθμών (10, 10) και (200, 10). Αυτά, υποδηλώνουν συγκεκριμένα σημεία της επιφάνειας. Στην εντολή σχεδίασης γραμμής καθορίζουν με απόλυτη ακρίβεια τα δύο άκρα της γραμμής. Το σημείο (10, 10) απέχει 10 pixel από αριστερά και 10 pixel από επάνω, ενώ το σημείο (200, 10) απέχει 200 pixel από αριστερά και 10 pixel από επάνω.

Γενικά, κάθε σημείο της επιφάνειας έχει τις δικές του αποστάσεις από αριστερά και από επάνω, οι οποίες ονομάζονται και συντεταγμένες του σημείου. Το επάνω αριστερό σημείο δεν έχει τις φυσιολογικές συντεταγμένες (1,1) που ίσως περίμενε κανείς, αλλά (0,0). Έτσι, στο παράθυρό μας το πιο κάτω δεξί σημείο αναφέρεται με (499, 199) και όχι με (500, 200). Αυτό γίνεται κατανοητό ίσως, αν προσπαθήσει κανείς να υπολογίσει την απόσταση του πρώτου pixel από τα άκρα. Από την επάνω γραμμή απέχει πράγματι 0, ενώ από την αριστερή πλευρά απέχει επίσης 0. Επίσης, για όσους γνωρίζουν για το επίπεδο καρτεσιανό σύστημα συντεταγμένων στα μαθηματικά, ας παρατηρηθεί η σημαντική διαφορά της αρχής. Στα γραφικά των υπολογιστών έχει επικρατήσει το επάνω αριστερό σημείο ως αρχή, ενώ το επίπεδο καρτεσιανό σύστημα αναπαρίσταται με την αρχή του κάτω αριστερά!



Στην εντολή **`pygame.draw.circle(epif,(255,255,255),(100,60),20,3)`** επίσης παρατηρούμε ένα όρισμα που έχει τη μορφή συντεταγμένων σημείου, συγκεκριμένα το ζευγάρι αριθμών (100, 60). Στις εντολές σχεδίασης κύκλων, με αυτό το τρίτο όρισμα δηλώνεται το κέντρο του κύκλου. Με το τέταρτο όρισμα δηλώνεται το μέγεθος της ακτίνας του κύκλου ενώ με το πέμπτο δηλώνεται το πάχος της γραμμής του κύκλου σε pixel.

Η εντολή **`pygame.draw.rect(epif,(255,255, 255),(100,150,30,30),2)`** είναι εντολή σχεδίασης τετραγώνου και αντιλαμβάνεται βέβαια κανείς αμέσως τη χρησιμότητα που θα έχει για το σκάκι μας. Το πρώτο όρισμα (η μεταβλητή epif) αφορά την επιφάνεια που θα σχεδιαστεί το τετράγωνο. Το δεύτερο όρισμα αφορά τον χρωματισμό, ο οποίος στην περίπτωσή μας είναι το λευκό. Το τρίτο όρισμα είναι μια τετράδα αριθμών, η οποία χαρακτηρίζει τη θέση το πλάτος και το ύψος του τετραγώνου. Οι δύο πρώτοι αριθμοί της τετράδας αφορούν την απόσταση του τετραγώνου από αριστερά και από επάνω, με άλλα λόγια τις συντεταγμένες του επάνω αριστερού σημείου. Οι δύο επόμενοι αριθμοί της τετράδας αφορούν το πλάτος και το ύψος του σχεδιαζόμενου τετραγώνου. Το τελευταίο όρισμα αφορά το πάχος γραμμής σε pixel του τετραγώνου. Όταν αυτό το όρισμα είναι 0 όπως συμβαίνει με την τελευταία εντολή σχεδίασης, το τετράγωνο σχεδιάζεται γεμισμένο και αυτό ισχύει και για τη σχεδίαση κύκλων.

Ας σημειωθεί ότι επιτρέπονται αναφορές σε σημεία εκτός της επιφάνειας σχεδίασης! Για παράδειγμα, αν δοκιμάσουμε να σχεδιάσουμε γραμμή με άκρα τα σημεία (10, 10) και (800,

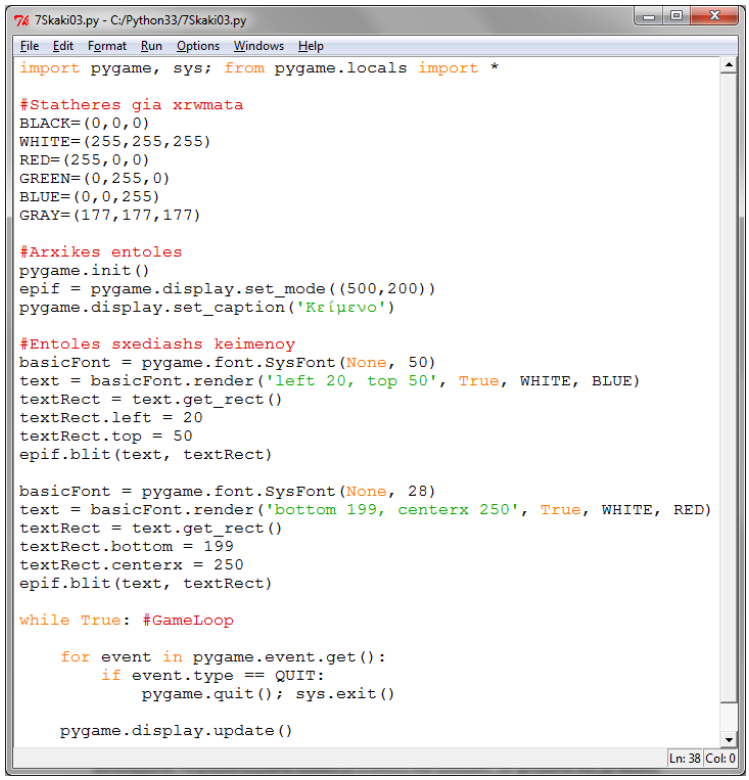
10), τότε αυτή θα σχεδιαστεί, παρόλο που το δεύτερο σημείο βρίσκεται εκτός καμβά. Οι αναφορές σε σημεία πάνω ή αριστερά από τον καμβά, γίνονται με αρνητικούς αριθμούς.

Ας σημειωθεί επίσης ότι υπάρχουν και άλλες εντολές σχεδίασης όπως π.χ. **`pygame.draw.ellipse(drawSurface,(0,0,255),(200, 100, 40, 80),2)`** ή **`pygame.draw.polygon(drawSurface,(0,0,255),((4,70),(10,100),(35,105),(45,95)),3)`**, οι οποίες σχεδιάζουν ελλείψεις και πολύγωνα αντίστοιχα. Στην έλλειψη πρέπει να δοθεί ως όρισμα και μια τετράδα, δηλαδή ένα νοητό τετράγωνο, μέσα στο οποίο θέλουμε να εμφανιστεί. Στο πολύγωνο πρέπει συν τοις άλλοις να δοθούν τα ζεύγη συντεταγμένων των κορυφών του. Αν επιθυμεί κανείς να σχεδιάσει σημείο, τότε το πετυχαίνει αυτό με την εντολή γραμμής, ορίζοντας ως αρχή και τέλος τις συντεταγμένες του ίδιου αυτού σημείου.

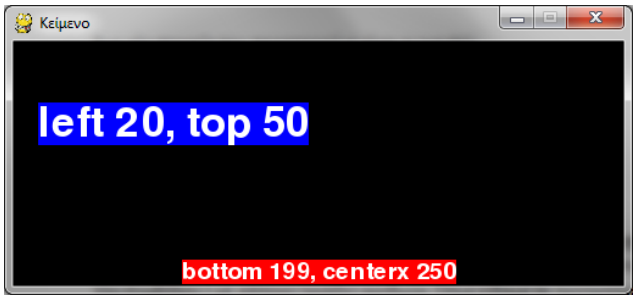
Συνιστάται ιδιαίτερα να δημιουργήσουμε αντίγραφα του παραπάνω προγράμματος και να δοκιμάσουμε διάφορες τιμές ορισμάτων στις εντολές. Κάθε κεφάλαιο, πρέπει να χωνεύεται πολύ καλά και ο καλύτερος τρόπος γι' αυτό, είναι ο συνεχής πειραματισμός και η αυτενέργεια. Πειραματιζόμαστε λοιπόν με κύκλους και γραμμές, με χρώματα και με θέσεις και φτιάχνουμε τις δικές μας δημιουργίες.

ΚΕΦΑΛΑΙΟ 3, Κείμενο

Σε αυτό το κεφάλαιο θα μάθουμε να εμφανίζουμε κείμενα κατά βούληση, όπου και όπως τα θέλουμε, κάτι που θα μας χρειαστεί αργότερα. Για το σκοπό αυτό, δημιουργούμε ένα ακριβές αντίγραφο του προηγούμενου προγράμματος και το τροποποιούμε μόνο στα κατάλληλα σημεία. Στο παράθυρο κελύφους της Python επιλέγουμε από το μενού File/Open... . Επιλέγουμε από τα αρχεία Python, εκείνο που αποθηκεύσαμε στο προηγούμενο κεφάλαιο. Όταν φορτωθεί, ήμαστε έτοιμοι να δημιουργήσουμε ένα ακριβές αντίγραφο. Επιλέγουμε File/Save As... και αποθηκεύουμε με διαφορετικό όνομα π.χ. 7skaki03.py . Σε αυτό το αντίγραφο κάνουμε τις απαραίτητες αλλαγές ώστε να προκύψει το παρακάτω πρόγραμμα.



Αποθηκεύουμε το πρόγραμμα και το τρέχουμε! Πρέπει να εμφανιστεί κάτι σαν αυτό...



Ένα νέο στοιχείο που παρατηρούμε είναι οι εντολές `BLACK=(0,0,0)`, `WHITE=(255,255,255)`, `RED=(255,0,0)`, `GREEN=(0,255,0)`, `BLUE=(0,0,255)`, `GRAY=(177,177,177)`, οι οποίες μας εισάγουν στην έννοια των σταθερών.

Έχει αποδειχθεί χρήσιμη στον προγραμματισμό η λεγόμενη τεχνική των **σταθερών**. Στην περίπτωσή μας, οι λέξεις BLACK, WHITE, RED, GREEN, BLUE και GRAY αντιστοιχίζονται μέσω του συμβόλου = με κάποιες τριάδες αριθμών. Όπως είδαμε σε προηγούμενο κεφάλαιο, αυτές οι τριάδες κωδικοποιούν χρώματα, και οι τρεις τιμές καθορίζουν τη συμμετοχή των τριών βασικών χρωμάτων. Έτσι, κάθε φορά στο εξής που είναι απαραίτητο σε κάποιες εντολές σχεδίασης να αναφερθούμε σε αυτά τα χρώματα, μπορούμε αντί για τριάδα αριθμών να δώσουμε στη θέση της την αντίστοιχη λέξη. Λέξεις με τέτοιο ρόλο, έχει καθιερωθεί να γράφονται με κεφαλαία γράμματα και να ονομάζονται σταθερές. Αντίθετα με τις μεταβλητές, το περιεχόμενό τους δηλαδή η τιμή τους, παραμένει σταθερή. Θα δούμε παρακάτω, ότι έτσι πετυχαίνουμε περισσότερο κατανοητές εντολές.

Ας δώσουμε και τις υπόλοιπες εξηγήσεις. Κάθε μπλοκ εντολών αφορά και ένα διαφορετικό κείμενο στο παράθυρο εκτέλεσης. Το πρώτο μπλοκ δημιουργεί το κείμενο πάνω αριστερά. Η εντολή **`pygame.font.SysFont(None, 50)`** δημιουργεί ένα αόριστο μέχρι στιγμής κείμενο. Το πρώτο όρισμα None, δηλώνει ότι δεν επιθυμούμε συγκεκριμένη γραμματοσειρά, και επιτρέπουμε να χρησιμοποιηθεί από το σύστημα η προεπιλεγμένη γραμματοσειρά του, η οποία συνήθως επαρκεί. Το μέγεθος γραμματοσειράς, δηλώνεται με το δεύτερο όρισμα, 50 στιγμές στην περίπτωσή μας. Για ν' αποκτήσουμε μια αίσθηση, τα κείμενα των βιβλίων γράφονται με 12 στιγμές περίπου. Όλο αυτό το αόριστο κείμενο αποθηκεύεται στη μεταβλητή basicFont.

Στη συνέχεια, με την εντολή **`text = basicFont.render('left 20, top 50', True, WHITE, BLUE)`** δημιουργείται ένα επακριβώς ορισμένο αντικείμενο κειμένου. Αυτό πλέον διαθέτει κείμενο, διαθέτει χρώμα, διαθέτει χρώμα φόντου πίσω από τα γράμματα, στοιχεία τα οποία δηλώνονται με το πρώτο, το τρίτο και το τέταρτο όρισμα αντίστοιχα. Ως δεύτερο όρισμα δηλώνουμε **`True`**. Το αντικείμενο αυτό αποθηκεύεται στην μεταβλητή text και ουσιαστικά αποτελεί μια ορθογώνια χρωματισμένη περιοχή. Ουσιαστικά ομοιάζει ο τύπος αυτού του αντικειμένου με τον τύπο της μεταβλητής epif που αντιπροσωπεύει όλη την επιφάνεια σχεδίασης. Τέτοιοι τύποι επιφανειών, ονομάζονται Surface. Στη συνέχεια, γι' αυτό το αντικείμενο δημιουργείται με την εντολή **`textRect = text.get_rect()`** ένα νοητό ορθογώνιο που ονομάζεται **`textRect`**. Αυτό είναι απαραίτητο σε όλα τα κείμενα, ώστε να μπορούμε να τους ορίσουμε ακριβή θέση που θα σχεδιαστούν. Οι εντολές **`textRect.left = 20`** και **`textRect.top = 50`** καθορίζουν το επάνω αριστερό σημείο του νοητού ορθογωνίου να απέχει 20 pixel από αριστερά και 50 pixel από επάνω. Τέλος, η εντολή **`epif.blit(text, textRect)`** αναλαμβάνει να

σχεδιάσει το αντικείμενο text πάνω στην επιφάνεια σχεδίασης στη θέση του νοητού ορθογωνίου **textRect**. Αυτή η εντολή ονομάζεται και μέθοδος **blit** του αντικειμένου epif, η οποία μέθοδος απαιτεί τα δύο ορίσματα που είδαμε πιο πριν. Αν δεν οριστεί θέση νοητού ορθογωνίου, το κείμενο εμφανίζεται στην προεπιλεγμένη θέση (0, 0).

Στο επόμενο κείμενο, το μόνο καινούργιο που βλέπουμε είναι οι εντολές **textRect.bottom=199** και **textRect.centerx=250**. Με αυτές καθορίζεται που θα βρίσκεται η βάση του νοητού ορθογωνίου καθώς και σε ποια οριζόντια θέση θα βρίσκεται το κέντρο του ίδιου ορθογωνίου. Γενικότερα, όλα τα νοητά ορθογώνια αντικείμενα έχουν ιδιότητες θέσης, τις οποίες μπορούμε να ορίζουμε. Αυτές είναι:

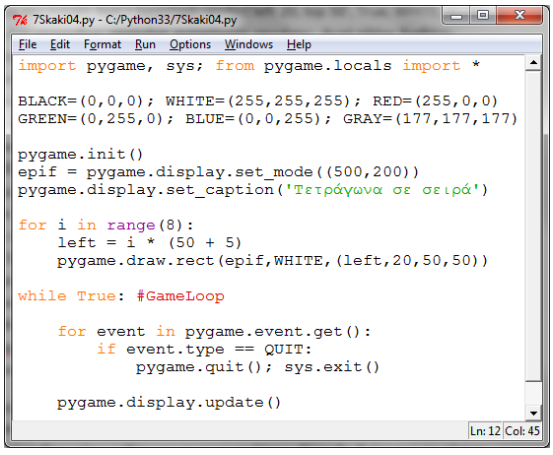
textRect.top - απόσταση από επάνω
textRect.left - απόσταση από αριστερά
textRect.bottom - απόσταση της βάσης του ορθογωνίου από επάνω
textRect.right - απόσταση της δεξιάς πλευράς του ορθογωνίου από αριστερά
textRect.centerx - απόσταση του οριζόντιου κέντρου του ορθογωνίου από αριστερά
textRect.centery - απόσταση του κατακόρυφου κέντρου του ορθογωνίου από επάνω

Σημαντικό είναι το γεγονός ότι αλλαγές στις παραπάνω ιδιότητες δεν προκαλούν αλλαγή πλάτους ή ύψους του ορθογωνίου! Προκαλείται απλά μετακίνηση, δηλαδή αλλάζουν αντίστοιχα όλες οι ιδιότητες που επηρεάζονται. Για παράδειγμα αύξηση της ιδιότητας .top προκαλεί αυτόματα ίση αύξηση των ιδιοτήτων στις ιδιότητες **.bottom** και **.centery**. Τα νοητά ορθογώνια αντικείμενα διαθέτουν και άλλες ιδιότητες όπως ύψος και πλάτος, στις οποίες μπορούμε να αναφερθούμε π.χ. ως εξής: **textRect.width** και **textRect.height**.

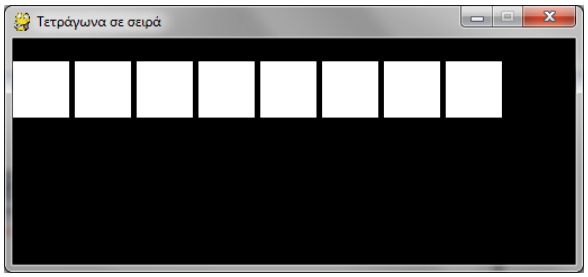
Ας κρατήσουμε στο μυαλό ότι τα διάφορα αντικείμενα προγραμματισμού διαθέτουν είτε **ιδιότητες** π.χ. **textrect.top**, **textRect.bottom**, είτε **μεθόδους** π.χ **basicFont.render(...)** ή **epif.blit(...)** ή **pygame.display.set_caption(...)**, όπου παρατηρούμε τις παρενθέσεις. Με τον καιρό θα εμβαθύνουμε σε αυτά τα θέματα, αλλά όπως είπαμε και στην αρχή, μας ενδιαφέρει πρωτίστως να πραγματοποιήσουμε γρήγορη πρόοδο στην πρακτική και δευτερευόντως στη θεωρητική γνώση, η οποία πολλές φορές κουράζει, αν αναλυθεί πλήρως και σε βάθος.

ΚΕΦΑΛΑΙΟ 4, Σχεδίαση τετραγώνων στη σειρά

Γράφουμε το παρακάτω πρόγραμμα.



Τρέχουμε το πρόγραμμα και παρατηρούμε να σχεδιάζονται 8 τετράγωνα στη σειρά!



Ας εξετάσουμε τον κώδικα. Αρχικά παρατηρούμε ότι οι εντολές ορισμού των σταθερών για τα χρώματα έχουν συμπτυχθεί σε λιγότερες γραμμές. Γενικά, επιτρέπεται αυτή η συγχώνευση απλών εντολών, αρκεί να τοποθετηθεί μεταξύ τους το σύμβολο του ερωτηματικού (;). Έτσι κερδίζει κανείς γραμμές κώδικα. Εξάλλου το ίδιο έχει γίνει στην πρώτη και στην δεκάτη όγδοη γραμμή του κώδικά μας. Παρατηρούμε επίσης την παρακάτω προσθήκη.

```
for i in range(8):
    left = i * (50 + 5)
    pygame.draw.rect(epif,WHITE,(left,20,50,50))
```

Γενικά, η **εντολή for** έχει ως αποτέλεσμα την πολλαπλή εκτέλεση των εντολών που ακολουθούν σε εσοχή. Η έκφραση δίπλα στο **`for`** καθορίζει το πλήθος των επαναλήψεων. Στην περίπτωσή μας, η έκφραση **`i in range(8)`** έχει ως αποτέλεσμα η μεταβλητή i να πάρει διαδοχικά τις τιμές 0, 1, 2, 3, 4, 5, 6, 7. Η εντολή **`range(…)`** επιστρέφει μια λίστα με ακέραιους, τόσους όσο το όρισμα. Για κάποιους λόγους, που δεν θα εξετάσουμε εδώ, οι ακέραιοι που επιστρέφονται γενικά από την **`range(n)`** δεν είναι 1, 2, 3, … n, αλλά 0, 1, … n-1. Δηλαδή περιλαμβάνεται το 0 ενώ δεν περιλαμβάνεται το n. Όταν τα ορίσματα της **`range`** είναι δύο, π.χ. **`range(1,5)`**, επιστρέφονται οι ακέραιοι από το πρώτο όρισμα συμπεριλαμβανόμενο, έως το τελευταίο, μη συμπεριλαμβανόμενο. Στο παράδειγμα θα επιστραφούν 1, 2, 3, 4. Καλό είναι να πειραματιστεί κανείς στο κέλυφος της Python με την **`range`**. Γράφοντας **`list(range(8))`** δίπλα στο σύμβολο προτροπής >>> και πατώντας Enter, θα εμφανιστεί αμέσως η λίστα με τους οκτώ αντίστοιχους αριθμούς από το 0 έως το 7. Γράφοντας **`list(range(1,10))`** θα εμφανιστούν οι αριθμοί από 1 έως 9. Είναι χρήσιμο να γνωρίζει κανείς επίσης, ότι αν δοθούν τρία ορίσματα στην range(…), τότε επιστρέφεται λίστα αριθμών με βηματισμό σύμφωνα με το τρίτο όρισμα! Π.χ. η **`range(2,102,2)`** επιστρέφει τους ζυγούς 2, 4, 6, 100. Πληκτρολογώντας στο κέλυφος **`list(range(2,102,2))`** και πατώντας Enter, το επιβεβαιώνουμε αυτό.

Οι δύο εντολές σε εσοχή που ακολουθούν την **`for`**, θα εκτελεστούν οκτώ φορές. Παρατηρούμε ότι κάθε φορά θα σχεδιάζεται ένα τετράγωνο. Όμως το όρισμα που καθορίζει την αριστερή θέση του τετραγώνου που θα σχεδιάζεται κάθε φορά, είναι μεταβλητή, συγκεκριμένα η μεταβλητή left. Ενώ, λοιπόν, το ύψος και το πλάτος των τετραγώνων είναι, όπως βλέπουμε, καθορισμένο (50 σύμφωνα με την τρίτη και τέταρτη τιμή της τετράδας), όπως επίσης καθορισμένη είναι και η απόσταση από επάνω (20, σύμφωνα με τη δεύτερη τιμή της τετράδας), δεν συμβαίνει το ίδιο με την απόσταση του τετραγώνου από αριστερά. Η απόσταση αυτή κάθε φορά μεταβάλλεται ανάλογα με την τιμή της μεταβλητής left.

Η εντολή **`left = i * (50 + 5)`** που θα εκτελείται σε κάθε επανάληψη, είναι μια **εντολή εκχώρησης** όπου στην αριστερή μεταβλητή left θα καταχωρείται η τιμή που προκύπτει από την δεξιά έκφραση. Ο αστερίσκος είναι το σύμβολο πολλαπλασιασμού. Στην πρώτη επανάληψη της for που το i έχει τιμή 0, το left θα πάρει τιμή 0. Στην δεύτερη επανάληψη που το i έχει τιμή 1, το left θα πάρει τιμή 55. Στην τρίτη επανάληψη που το i έχει τιμή 2, το left θα πάρει τιμή 110 κτλ. Στην όγδοη επανάληψη που το i έχει τιμή 7, το left θα πάρει τιμή 385. Όλες αυτές οι τιμές της left, θα χρησιμοποιηθούν επαναληπτικά στην εντολή **`pygame.draw.rect(epif, WHITE, (left, 20, 50, 50))`**, έχοντας ως αποτέλεσμα τη σχεδίαση οκτώ τετραγώνων στη σειρά. Ο λόγος που δεν γράφουμε απλά 55 αντί για την παρένθεση (50+5), είναι για να γίνεται αντιληπτό ποιο είναι το κενό μεταξύ των τετραγώνων, δηλαδή 5 pixel.

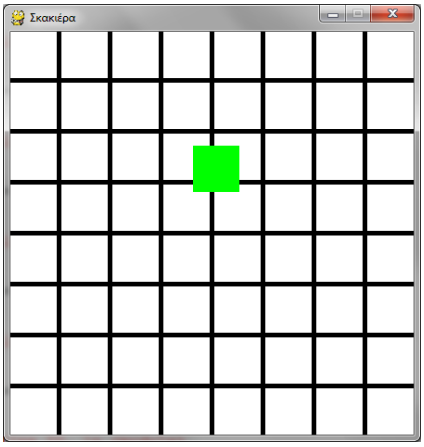
Ας δοκιμάσει κανείς να αλλάξει και τα άλλα ορίσματα στην εντολή σχεδίασης, τοποθετώντας μεταβλητές αντί για σταθερές τιμές. Θα εκπλαγεί με τα αποτελέσματα αλλά και θα χωνέψει περισσότερο αυτό το κεφάλαιο.

ΚΕΦΑΛΑΙΟ 5, Σκακιέρα και αλληλεπίδραση με ποντίκι

Σε αυτό το κεφάλαιο θα ετοιμαστεί πλέον η σκακιέρα μας με την λεγόμενη τεχνική του διπλού for. Επίσης, θα γίνει επίδειξη του πώς μπορεί ένα πρόγραμμα να ανταποκριθεί σε ενέργειες του χρήστη, συγκεκριμένα σε κλικ του ποντικιού. Γράφουμε τον παρακάτω κώδικα, ο οποίος λίγο διαφέρει από αυτόν του προηγούμενου κεφαλαίου.

```
import pygame, sys; from pygame.locals import *

BLACK=(0,0,0); WHITE=(255,255,255); RED=(255,0,0)
GREEN=(0,255,0); BLUE=(0,0,255); GRAY=(177,177,177)

pygame.init()
epif = pygame.display.set_mode((435,435))
pygame.display.set_caption('Σκακιέρα')

for i in range(8): #Sxediash skakieras
    for j in range(8):
        left = i * (50 + 5)
        top = j * (50 + 5)
        pygame.draw.rect(epif,WHITE,(left,top,50,50))

while True: #GameLoop
    mouseClicked = False
    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit(); sys.exit()
        elif event.type == MOUSEBUTTONUP: #Klik pontikioy
            mousex, mousey = event.pos
            mouseClicked = True

    if mouseClicked == True:
        pygame.draw.rect(epif,GREEN,(mousex,mousey,50,50),0)

    pygame.display.update()
```

Αποθηκεύουμε π.χ. ως 7skaki05.py και εκτελούμε.



Παρατηρούμε να εμφανίζεται στο παράθυρο εκτέλεσης ένα ταμπλό 8x8 τετραγώνων. Εξάλλου τα κλικ με το ποντίκι σχεδιάζουν πράσινα τετράγωνα στη θέση του κλικ.

Κατ' αρχήν με την **`pygame.display.set_mode((435,435))`** δημιουργήθηκε κατάλληλο παράθυρο εκτέλεσης που να χωράει τα τετράγωνα. Το 435 υπολογίζεται εύκολα καθώς αθροίζεται από οκτώ πενηντάρια (μέγεθος τετραγώνων) και επτά πεντάρια (μέγεθος κενού). Στη συνέχεια παρατηρούμε τα εξής, με τα οποία σχεδιάστηκε η σκακιέρα.

```
for i in range(8): #Sxediash skakieras
    for j in range(8):
        left = i * (50 + 5)
        top = j * (50 + 5)
        pygame.draw.rect(epif,WHITE,(left,top,50,50))
```

Η εντολή **`for`** έχει εξηγηθεί σε προηγούμενο κεφάλαιο. Εδώ πρέπει να γίνει κατανοητό ότι όταν μια εντολή **`for`** περιλαμβάνει πάλι μια **`for`**, τότε οι εντολές της εσωτερικής for θα εκτελεστούν πολλαπλά. Στην περίπτωσή μας, εξήντα τέσσερις φορές, διότι τόσο κάνει οκτώ επί οκτώ. Αρχικά το i παίρνει τιμή 0 και η εσωτερική **`for`** θα προκαλέσει οκτώ εκτελέσεις των τριών εντολών της (καθώς το j θα πάρει τις τιμές 0,1,2…7). Στη συνέχεια το i παίρνει τιμή 1 και η εσωτερική for θα προκαλέσει και πάλι οκτώ εκτελέσεις των τριών εντολών της (καθώς το j θα ξαναπάρει τις τιμές 0,1,2…7). Και ούτω καθ' εξής, και την τελευταία φορά το i θα πάρει τιμή 7 και η εσωτερική for θα προκαλέσει και πάλι οκτώ εκτελέσεις των τριών εντολών της. Άρα την πρώτη φορά οι μεταβλητές left και top παίρνουν τιμές 0, 0. Την δεύτερη φορά 0, 55. Την τρίτη φορά 0, 110 κτλ. άρα μέχρι την όγδοη επανάληψη θα σχεδιαστούν τα τετράγωνα της πρώτης στήλης. Στις επόμενες οκτώ επαναλήψεις όπου το i θα έχει την τιμή 1, θα σχεδιαστούν τα τετράγωνα της δεύτερης στήλης. Και τελικά όταν το i θα πάρει την τιμή 7, θα σχεδιαστούν τα οκτώ τετράγωνα της τελευταίας στήλης.

Ας δούμε και τις αλλαγές στο GameLoop.

```
while True: #GameLoop
    mouseClicked = False
    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit(); sys.exit()
        elif event.type == MOUSEBUTTONUP: #Klik pontikioy
            mousex, mousey = event.pos
            mouseClicked = True

    if mouseClicked == True:
        pygame.draw.rect(epif,GREEN,(mousex,mousey,50,50),0)

    pygame.display.update()
```

Παρατηρούμε μια νέα μεταβλητή ονόματος mouseClicked, στην οποία αρχικά στο GameLoop καταχωρείται η τιμή **`False`**. Μεταβλητές που δέχονται ως περιεχόμενο τις τιμές αληθείας **`False`** και **`True`** ονομάζονται λογικές μεταβλητές. Ίσως ακούγονται πιο δυσνόητες από τις μεταβλητές αριθμητικού τύπου (όπως οι left και top), όμως και αυτές το μόνο που κάνουν είναι να δέχονται διαφορετικό περιεχόμενο κατά τη διάρκεια εκτέλεσης του προγράμματος. Η συγκεκριμένη μεταβλητή mouseClicked παίρνει τιμή **`True`** λίγο παρακάτω μόνο αν συμβεί το λεγόμενο γεγονός του κλικ **`(event.type == MOUSEBUTTONUP)`**, για την ακρίβεια όταν συμβεί να αφεθεί το πατημένο ποντίκι. Ενώ έως τώρα το μόνο συμβάν που μας ενδιέφερε ήταν το κλείσιμο του παραθύρου **`(event.type == QUIT)`**, τώρα έχουμε και το κλικ. Η εντολή **`for event in pygame.event.get()`** δέχεται ακατάπαυστα τις διάφορες ενέργειες του χρήστη. Η εντολή **`if`** που ακολουθεί την **`for`**, όταν εντοπίζει τέτοιες ενέργειες, πράττει καταλλήλως. Αν π.χ. εντοπιστεί η ενέργεια QUIT, τότε εκτελούνται οι εντολές που ακολουθούν, δηλαδή οι **`pygame.quit()`** και **`sys.exit()`** που προκαλούν το κλείσιμο του προγράμματος. Αν τώρα εντοπιστεί κλικ ποντικιού, τότε γίνονται τα εξής: Καταγράφονται αυτόματα από το πρόγραμμα οι συντεταγμένες του συμβάντος αυτού και αποθηκεύονται στην ιδιότητα event.pos ως ένα ζευγάρι αριθμών. Για να είναι πιο προσιτές οι συντεταγμένες αυτές, εμείς τις τοποθετούμε στις μεταβλητές mousex και mousey. Επίσης, η μεταβλητή mouseClicked παίρνει τιμή **`True`**. Όταν τελειώσει η **`for`**, εκτελείται μια εντολή **`if`** η οποία ελέγχει αν η μεταβλητή mouseClicked έχει τιμή **`True`**. Μόνο τότε θα σχεδιαστεί ένα πράσινο τετράγωνο, το οποίο παρατηρούμε ότι θα σχεδιαστεί στο σημείο του κλικ.

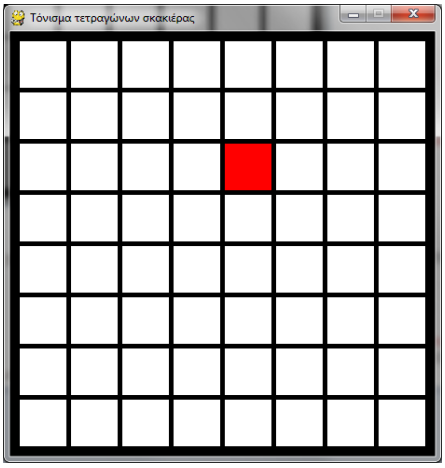
Η εντολή **`pygame.display.update()`** θα επικαιροποιήσει όλα τα δεδομένα στην οθόνη, διότι μέχρι στιγμής όλα σχεδιάζονται στη μνήμη. Καθώς φτάσαμε στο τέλος, η εκτέλεση πηγαίνει και πάλι στην αρχή των εντολών της **`while`**. Δηλαδή θα εκτελεστεί η εντολή **`mouseClicked = False`**, θα εντοπιστούν με την **`for`** ενδεχόμενες ενέργειες του χρήστη κτλ. Πρέπει να γίνει κατανοητό ότι για όσο διάστημα δεν γίνεται κλικ, στην αέναη επανάληψη των εντολών η μεταβλητή mouseClicked διατηρεί τιμή **`False`** και άρα δεν εκτελείται η σχεδίαση πράσινου τετραγώνου. Μόνο σε κλικ, η τελευταία εντολή **`if`** «ξεκλειδώνει», ώστε να επιτρέψει να εκτελεστεί η εντολή σχεδίασης **`pygame.draw.rect`**.

Γενικότερα, με αυτό το πρόγραμμα είδαμε έναν απλό τρόπο με τον οποίο μπορούμε να χειριστούμε κλικ του ποντικιού. Κάποιοι προτιμούν το συμβάν **`MOUSEBUTTONDOWN`** αντί για το συμβάν **`MOUSEBUTTONUP`** ως τελική οριστικοποίηση του κλικ. Προτείνεται να δοκιμαστούν και τα δύο, ώστε να αντιληφθεί κανείς ακριβώς τη διαφορά τους.

ΚΕΦΑΛΑΙΟ 6, Γενίκευση και τόνισμα τετραγώνων

```
import pygame, sys, time; from pygame.locals import *
TX=TY=8 #Plhthos tetragwnwn orizontia kai katheta
SZ=50 #Megethos tetragwnoy
KENO=5; PER=10 #Keno kai ekswteriko Perithwrio
BLACK=(0,0,0); WHITE=(255,255,255); RED=(255,0,0)
GREEN=(0,255,0); BLUE=(0,0,255); GRAY=(177,177,177)
pygame.init()
epif=pygame.display.set_mode((2*PER+TX*SZ+(TX-1)*KENO,2*PER+TY*SZ+(TY-1)*KENO))
pygame.display.set_caption('Τόνισμα τετραγώνων σκακιέρας')
for i in range(TX): #Sxediash skakieras
    for j in range(TY):
        left = i * (SZ + KENO)
        top = j * (SZ + KENO)
        pygame.draw.rect(epif,WHITE,(PER+left,PER+top,SZ,SZ))
while True: #GameLoop
    mouseClicked = False
    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit(); sys.exit()
        elif event.type == MOUSEBUTTONUP: #Klik pontikioy
            mousex, mousey = event.pos
            mouseClicked = True
    if mouseClicked == True:
        #Ypologizetai to tetragwno stis syntetagmenes mousex kai mousey.
        ypoloipox = ypoloipoy = SZ
        if mousex > PER and mousey > PER:
            x = int((mousex-PER)/(SZ+KENO))
            ypoloipox = (mousex-PER)-x*(SZ+KENO)
            y = int((mousey-PER)/(SZ+KENO))
            ypoloipoy = (mousey-PER)-y*(SZ+KENO)
        if ypoloipox<SZ and ypoloipoy<SZ and x<TX and y<TY: #Egkyro klik
            left = PER+x*(SZ+KENO)
            top = PER+y*(SZ+KENO)
            pygame.draw.rect(epif,RED,(left,top,SZ,SZ),0)
            pygame.display.update()
            time.sleep(0.5) #Kathysterhsh miso deyterolepto
            pygame.draw.rect(epif,WHITE,(left,top,SZ,SZ),0)
    pygame.display.update()
```

Μεταφέρουμε τον παραπάνω κώδικα σε νέο αρχείο, τον αποθηκεύουμε π.χ. ως 7skaki06.py και τον τρέχουμε. Πρέπει να εμφανιστεί κάτι σαν το παρακάτω:

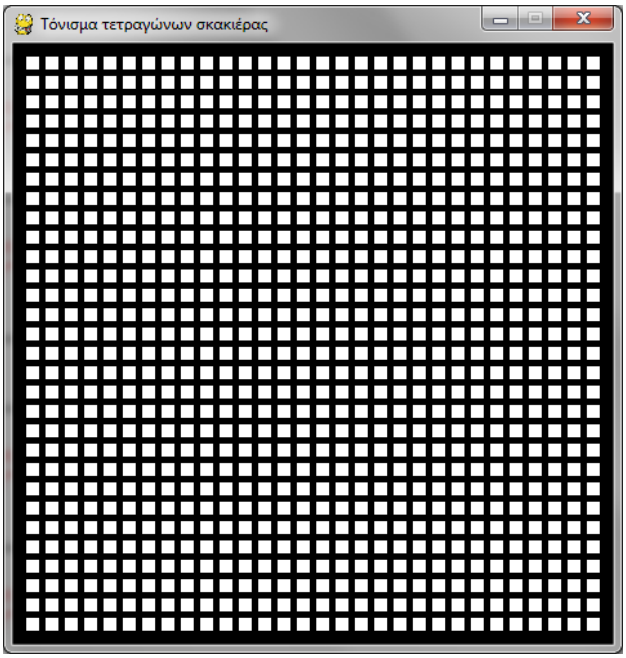


Το πρώτο που βλέπουμε είναι ότι έχει προστεθεί ένα περιθώριο γύρω από τα τετράγωνα. Επίσης, ένα κλικ έχει ως αποτέλεσμα να σχεδιαστεί κόκκινο το τετράγωνο πάνω στο οποίο έγινε το κλικ.

Ας παρατηρήσουμε ότι στις γραμμές 2 έως 4 έχουμε ορίσει πέντε νέες σταθερές, τις TX, TY, SZ, KENO και PER στις οποίες έχουμε δώσει τις τιμές που φαίνονται. Η σταθερές TX και TY αντικαθιστούν πλέον στον κώδικα το 8, δηλαδή το πλήθος τετραγώνων οριζόντια και

κάθετα. Δηλαδή σε σχέση με το προηγούμενο πρόγραμμα, κάθε εμφάνιση του 8 έχει αντικατασταθεί με τις ανάλογες σταθερές TX και TY. Παρόμοια, η σταθερά SZ αντικαθιστά το 50 στον κώδικα. Η σταθερά KENO αντικαθιστά το 5 στον κώδικα. Και τέλος, η σταθερά PER αφορά το περιθώριο σε pixel, το οποίο περιβάλλει όλη τη σκακιέρα, το οποίο ορίστηκε σε 10 pixel. Αυτές όλες οι σταθερές, συντελούν σε αυτό που λέγεται γενίκευση. Η γενίκευση των προγραμμάτων είναι μια χρήσιμη τεχνική, η οποία μας επιτρέπει με ελάχιστες τροποποιήσεις του κώδικα να αλλάξουμε εντυπωσιακά τη συμπεριφορά ενός προγράμματος.

Δοκιμάζουμε π.χ. να αλλάξουμε την τιμή της σταθεράς SZ σε 10 και τις τιμές των σταθερών TX, TY σε 30. Ξανατρέχουμε το πρόγραμμα και παρατηρούμε ότι όλα τα τετράγωνα έχουν τώρα μέγεθος 10 ενώ το πλήθος τους είναι 30 οριζόντια και 30 κάθετα.

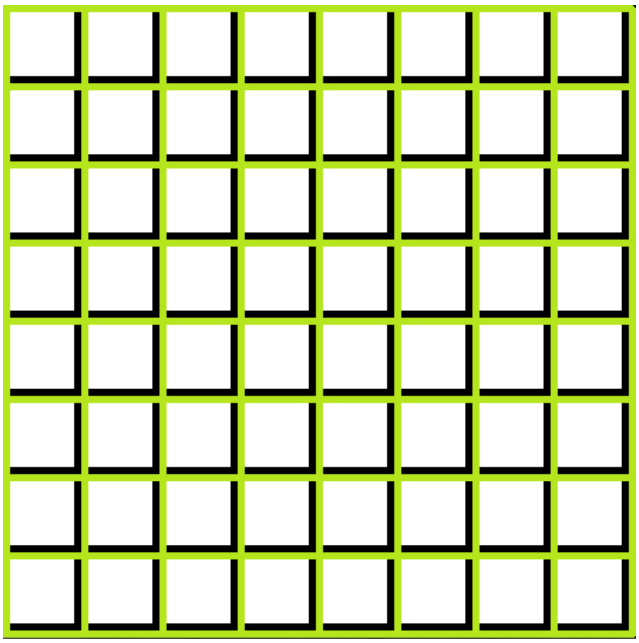


Παρατηρώντας το όρισμα της εντολής **`pygame.display.set_mode`** βλέπουμε ότι έχει εξαρτηθεί πλήρως από τις προηγούμενες σταθερές. Δηλαδή το οριζόντιο μέγεθος του παραθύρου θα προκύψει από την έκφραση **`2*PER+TX*SZ+(TX-1)*KENO`**, η οποία αθροίζει δύο περιθώρια PER, TX τετράγωνα πλάτους SZ και (TX-1) κενά πλάτους KENO. Παρόμοια υπολογίζεται και το ύψος του παραθύρου.

Παρατηρούμε επίσης ότι οι εντολές στο διπλό for, όπου σχεδιάζονται τα τετράγωνα, έχουν επίσης εξάρτηση από τις σταθερές. Οι επαναλήψεις είναι TX και TY και π.χ. η μεταβλητή left που αφορά την απόσταση από αριστερά του τετραγώνου, εξαρτάται και από την τρέχουσα τιμή του i αλλά και από τα μεγέθη SZ και KENO. Αξίζει, ο νέος προγραμματιστής να υπολογίσει το πώς αλλάζουν οι τιμές των μεταβλητών left και top στο διπλό for.

Μέσα στο GameLoop κάτω από την εντολή ελέγχου **`mouseClicked == True`** έχουν προστεθεί πολλά σε σχέση με το προηγούμενο πρόγραμμα. Αυτό που ουσιαστικά γίνεται είναι να εντοπιστεί με ειδικούς υπολογισμούς σε ποιο τετράγωνο έγινε κλικ, ώστε να σχεδιαστεί αυτό κόκκινο. Αν το κλικ δεν γίνει πάνω σε τετράγωνο αλλά σε κενό, τότε δεν σχεδιάζεται τίποτα. Επεξηγείται παρακάτω η σχετικά προχωρημένη τεχνική που πετυχαίνεται αυτό, όμως μπορεί κανείς να την χρησιμοποιεί στα τυφλά, δίχως να απαιτείται η εκμάθησή της.

Με τις κατάλληλες εντολές, από τις τιμές των μεταβλητών mousex και mousey που όπως είπαμε περιέχουν τις συντεταγμένες του κλικ, υπολογίζεται το αντίστοιχο τετράγωνο (x, y) του ταμπλό. Ο τρόπος που πετυχαίνεται αυτή η ακαριαία μετατροπή των συντεταγμένων του ποντικιού σε τετράγωνα του ταμπλό, βασίζεται στη χρήση μαθηματικών λειτουργιών της ακέραιας διαίρεσης και του υπόλοιπου της ακέραιας διαίρεσης. Θεωρείται αρχικά ως τετράγωνο η περιοχή που περιέχει τόσο το τετράγωνο όσο και τα κενά δεξιά του και από κάτω του. Δηλαδή αυτό το τετράγωνο έχει μέγεθος πλευράς SZ+KENO (πράσινο).



Με την εντολή **`x = int((mousex-PER)/(SZ+KENO))`** θα υπολογισθεί στο x η οριζόντια θέση του κλικ, θεωρώντας ταμπλό με μεγεθυμένα τετράγωνα μεγέθους SZ+KENO. Αυτό διότι διαιρώντας το **`(mousex-PER)`** με **`(SZ+KENO)`**, ουσιαστικά βλέπουμε πόσες φορές χωράει οριζόντια το μεγεθυμένο τετράγωνο στο μεγεθυμένο ταμπλό. Υπόψη ότι το πρώτο τετράγωνο δίνει τιμή 0, το δεύτερο δίνει 1,…, το όγδοο τετράγωνο δίνει 7. Όμως πώς θα ελέγξουμε ότι το κλικ δεν έγινε στην περιοχή του κενού του μεγεθυμένου τετραγώνου; Αυτό ακριβώς γίνεται στη συνέχεια. Στο ypoloipox βλέπουμε τι υπόλοιπο μένει, αν αφαιρέσουμε x μεγεθυμένα τετράγωνα από το συνολικό μήκος. Αν αυτό το υπόλοιπο είναι μικρότερο του SZ, τότε αυτό σημαίνει ότι το κλικ γίνεται πάνω σε τετράγωνο και όχι στο μέρος του δεξιού κενού (του μεγεθυμένου τετραγώνου). Η ίδια λογική εφαρμόζεται και κατακόρυφα, υπολογίζοντας το μεγεθυμένο τετράγωνο στην κατακόρυφο και ελέγχοντας μήπως το κλικ γίνεται στο κενό κάτω του τετραγώνου. Τελικά, αν και στις δύο κατευθύνσεις διαπιστώνεται ότι το κλικ δεν γίνεται στα κενά (ypoloipox<SZ and ypoloipoy<SZ), τότε πράγματι, οι συντεταγμένες x, y στις οποίες υπολογίστηκαν τα μεγεθυμένα τετράγωνα, ισχύουν και για τα κανονικά τετράγωνα. Ξανατονίζεται ότι δεν χρειάζεται να γίνουν απόλυτα κατανοητές οι εντολές αυτού του μέρους κώδικα, ειδικά από μικρότερες ηλικίες.

Σε περίπτωση έγκυρου κλικ σε τετράγωνο, με τις εντολές **`left = PER+x*(SZ+KENO)`** και **`top = PER+y*(SZ+KENO)`** υπολογίζονται οι συντεταγμένες της πάνω αριστερής γωνίας του τετραγώνου, ώστε να χρησιμοποιηθούν στη συνέχεια στην εντολή σχεδίασης του κόκκινου τετραγώνου. Η εντολή **`pygame.display.update()`** ενημερώνει αμέσως την οθόνη με το κοκκίνισμα, ενώ η εντολή **`time.sleep(0.5)`** έχει ως αποτέλεσμα μια καθυστέρηση του προγράμματος για διάρκεια μισού δευτερολέπτου. Αμέσως μετά, η

επόμενη εντολή σχεδίασης ξαναβάφει το συγκεκριμένο τετράγωνο λευκό. Σημειώνεται ότι η εντολή **time.sleep** απαιτεί να έχουμε συνδέσει στο πρόγραμμά μας το πακέτο επέκτασης εντολών time, κάτι που έγινε στην πρώτη γραμμή (**import pygame, sys, time**).

ΚΕΦΑΛΑΙΟ 7, Υποπρογράμματα

```
import pygame, sys, time; from pygame.locals import *
TX=TY=8 #Plhthos tetragwnwn orizontia kai katheta
SZ=50 #Megethos tetragwnoy
KENO=5; PER=10 #Keno kai ekswteriko Perithwrio
BLACK=(0,0,0); WHITE=(255,255,255); RED=(255,0,0)
GREEN=(0,255,0); BLUE=(0,0,255); GRAY=(177,177,177)

def xyFromMousexMousey(mousex, mousey):
    '''Epistrefontai oi syntetagmenes toy tetragwnoy (x,y) apo tis
    syntetagmenes mousex kai mousey efoson to klik ginetai panw se
    tetragwno. An den ginei klik se tetragwno, epistrefetai None.'''
    ypoloipox = ypoloipoy = SZ
    if mousex > PER and mousey > PER:
        x = int((mousex-PER)/(SZ+KENO))
        ypoloipox = (mousex-PER)-x*(SZ+KENO)
        y = int((mousey-PER)/(SZ+KENO))
        ypoloipoy = (mousey-PER)-y*(SZ+KENO)
        if ypoloipox<SZ and ypoloipoy<SZ and x<TX and y<TY:
            return x,y
    return None,None

pygame.init()
epif=pygame.display.set_mode((2*PER+TX*SZ+(TX-1)*KENO,2*PER+TY*SZ+(TY-1)*KENO))
pygame.display.set_caption('Υποπρογράμματα')
for i in range(TX): #Sxediash skakieras
    for j in range(TY):
        left = i * (SZ + KENO)
        top = j * (SZ + KENO)
        pygame.draw.rect(epif,WHITE,(PER+left,PER+top,SZ,SZ))
while True: #GameLoop
    mouseClicked = False
    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit(); sys.exit()
        elif event.type == MOUSEBUTTONUP: #Klik pontikioy
            mousex, mousey = event.pos
            mouseClicked = True
    if mouseClicked == True:
        #Ypologizetai to tetragwno stis syntetagmenes mousex kai mousey.
        x, y = xyFromMousexMousey(mousex, mousey)
        if  x != None: #Egkyro klik
            left = PER+x*(SZ+KENO)
            top = PER+y*(SZ+KENO)
            pygame.draw.rect(epif,RED,(left,top,SZ,SZ),0)
            pygame.display.update()
            time.sleep(0.5) #Kathysterhsh miso deyterolepto
            pygame.draw.rect(epif,WHITE,(left,top,SZ,SZ),0)
    pygame.display.update()
```

Πληκτρολογούμε το παραπάνω πρόγραμμα και αποθηκεύουμε π.χ. ως 7skaki07.py. Αυτό δεν παρουσιάζει καμία διαφοροποίηση κατά την εκτέλεσή του σε σχέση με το προηγούμενο. Όμως σε αυτό γίνεται χρήση της έννοιας του υποπρογράμματος, έννοια που πρέπει να γίνει σιγά σιγά αντιληπτή από νέους προγραμματιστές.

Παρατηρούμε ότι μετά τη δήλωση των σταθερών, έχει προστεθεί ένα νέο τμήμα κώδικα. Τμήματα κώδικα που ξεκινούν με την λέξη def, ονομάζονται υποπρογράμματα. Είναι σχετικά αυτοτελή τμήματα κώδικα και έχουν ως αποστολή να εκτελούν συγκεκριμένες εργασίες. Στην περίπτωσή μας, ορίσαμε το υποπρόγραμμα xyFromMousexMousey, το οποίο έχει ως αποστολή να εντοπίζει τις συντεταγμένες του τετραγώνου, πάνω στο οποίο γίνεται κλικ. Η δουλειά που προηγουμένως γινόταν χωρίς υποπρόγραμμα, τώρα θα γίνει με χρήση υποπρογράμματος. Το συγκεκριμένο υποπρόγραμμα δέχεται ως δεδομένα τις

παραμέτρους mousex, mousey (που ουσιαστικά δηλώνουν τη θέση του πίξελ όπου έγινε κλικ με το ποντίκι), και επιστρέφει στις μεταβλητές x και y ένα ζευγάρι αριθμών από (0,0) έως (7,7) που υποδηλώνουν ένα συγκεκριμένο τετράγωνο του ταμπλό. Αν δεν γίνει κλικ πάνω σε τετράγωνο (αλλά στα κενά μεταξύ των τετραγώνων ή στα περιθώρια), τότε το υποπρόγραμμα επιστρέφει τιμές None, δηλαδή το προγραμματιστικό «τίποτα». Παρατηρούμε επίσης κάποια αλλαγή μέσα στο Gameloop. Στην εντολή **`x, y = xyFromMousexMousey(mousex, mousey)`** καταχωρούνται στις μεταβλητές x και y οι επιστρεφόμενες τιμές του υποπρογράμματος. Με άλλα λόγια, όταν στο Gameloop εντοπιστεί το όνομα του υποπρογράμματος, τότε εκτελείται ο κώδικας που έχει οριστεί παραπάνω στο def. Στη συνέχεια, το **`if`** ελέγχει αν οι επιστρεφόμενες τιμές είναι έγκυρες. Αν το x είναι διαφορετικό του None, αυτό σημαίνει ότι επιστράφηκαν έγκυρες τιμές. Ακόμα και αν δεν κατανοεί κανείς απόλυτα τον κώδικα του υποπρογράμματος xyFromMousexMousey, δεν πειράζει. Μπορεί να τον χρησιμοποιεί τυφλά, συνειδητοποιώντας όμως τι κάνει και τι επιστρέφει, άσχετα της κατανόησης πώς το κάνει.

Πάντως, επειδή μπήκαμε κάπως απότομα σε βαθιά νερά, ας προσπαθήσει κανείς γράφοντας τον παρακάτω κώδικα στο κέλυφος της Python και τρέχοντάς τον, να αποκτήσει μια πιο ήρεμη επαφή με το αντικείμενο των υποπρογραμμάτων. Το παρακάτω υποπρόγραμμα μπορεί να χρησιμοποιηθεί για αθροίσματα δύο αριθμών.

```
>>> def athroisma(x,y):
        """Epistrefetai to athroisma
           dyo arithmwn"""
        return x+y

>>> print(athroisma(2,5))
7
>>>
```

Απλά μιλώντας, τα υποπρογράμματα δέχονται κάτι (τις παραμέτρους στις παρενθέσεις, οι οποίες λέγονται και ορίσματα) και επιστρέφουν κάτι άλλο, σύμφωνα με τις εντολές τους.

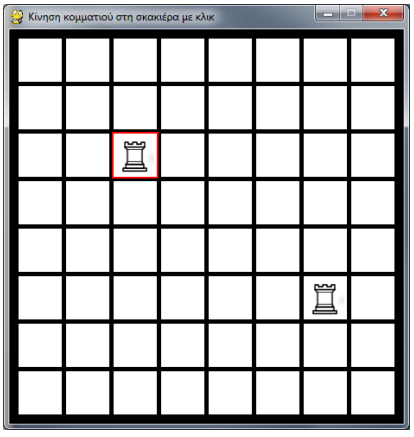
Ο παραπάνω κώδικας, είναι μια ισχυρή βάση, πάνω στην οποία μπορεί να στηθεί όχι μόνο το σκάκι αλλά οποιοδήποτε παιχνίδι με τετράγωνα όπως ντάμα κτλ. Καλό είναι να προσπαθήσουμε να τον κατανοήσουμε όσο γίνεται περισσότερο.

ΚΕΦΑΛΑΙΟ 8, Κίνηση κομματιού στη σκακιέρα με κλικ

Για την εκτέλεση του παρακάτω προγράμματος, το αρχείο εικόνας LeykosPyrgos.png είναι απαραίτητο να βρίσκεται στο φάκελο της Python, στην περίπτωσή μας στον C:/Python33 .

```
import pygame, sys, time; from pygame.locals import *
TX=TY=8; SZ=50; KENO=5; PER=10 #Plhthos tetragwnwn, megethos tetragwnoy, keno kai ekswteriko perithwrio
BLACK=(0,0,0); WHITE=(255,255,255); RED=(255,0,0); GREEN=(0,255,0); BLUE=(0,0,255); GRAY=(177,177,177)
LP = pygame.image.load('LeykosPyrgos.png');
def xyFromMousexMousey(mousex, mousey):
    '''Apo to klik, epistrefontai eite oi syntetagmenes toy tetragwnoy (x,y) eite None.'''
    ypoloipox = ypoloipoy = SZ
    if mousex > PER and mousey > PER:
        x = int((mousex-PER)/(SZ+KENO)); ypoloipox = (mousex-PER)-x*(SZ+KENO)
        y = int((mousey-PER)/(SZ+KENO)); ypoloipoy = (mousey-PER)-y*(SZ+KENO)
        if ypoloipox<SZ and ypoloipoy<SZ and x<TX and y<TY:
            return x,y
    return None,None
pygame.init()
epif=pygame.display.set_mode((2*PER+TX*SZ+(TX-1)*KENO,2*PER+TY*SZ+(TY-1)*KENO))
pygame.display.set_caption('Κίνηση κομματιού στη σκακιέρα με κλικ')
skaki = [] #Arxikopoihsh skakieras me ekxwrhsh 0 se ola ta tetragwna
for i in range(TX):
    column = []
    for j in range(TY):
        column.append(0) #0 shmainei keno tetragwno
    skaki.append(column)
skaki[2][2]=1; skaki[7][7]=1 #Leykoi pyrgoi
for i in range(TX): #Sxediash skakieras
    for j in range(TY):
        left = i * (SZ + KENO); top = j * (SZ + KENO)
        pygame.draw.rect(epif,WHITE,(PER+left,PER+top,SZ,SZ))
epif.blit(LP,(PER+2*(SZ+KENO),PER+2*(SZ+KENO),SZ,SZ)); epif.blit(LP,(PER+7*(SZ+KENO),PER+7*(SZ+KENO),SZ,SZ))
k=1
while True: #GameLoop
    mouseClicked = False
    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit(); sys.exit()
        elif event.type == MOUSEBUTTONUP: #Klik pontikioy
            mousex, mousey = event.pos; mouseClicked = True
    if mouseClicked == True:
        x, y = xyFromMousexMousey(mousex, mousey)
        if  x != None: #Egkyro klik
            if k==1 and skaki[x][y]==1: #An sthn katastash 1 ginei klik se pyrgo
                klik1x=x; klik1y=y; klik1kommati=skaki[x][y]
                pygame.draw.rect(epif,RED,(PER+x*(SZ+KENO)-2,PER+y*(SZ+KENO)-2,SZ+3,SZ+3),2)
                k=2
            elif k==2 and klik1x==x and klik1y==y: #An ginei deytero klik sto idio kommati
                pygame.draw.rect(epif,BLACK,(PER+klik1x*(SZ+KENO)-2,PER+klik1y*(SZ+KENO)-2,SZ+3,SZ+3),2)
                k=1
            elif k==2 and ((klik1x==x-1 and klik1y==y) or(klik1x==x+1 and klik1y==y) \
                  or (klik1x==x and klik1y==y-1) or(klik1x==x and klik1y==y+1) ) : #Egkyro deytero klik
                pygame.draw.rect(epif,BLACK,(PER+klik1x*(SZ+KENO)-2,PER+klik1y*(SZ+KENO)-2,SZ+3,SZ+3),2)
                pygame.draw.rect(epif,WHITE,(PER+klik1x*(SZ+KENO),PER+klik1y*(SZ+KENO),SZ,SZ),0)
                epif.blit(LP,(PER+x*(SZ+KENO),PER+y*(SZ+KENO),SZ,SZ))
                skaki[klik1x][klik1y]=0; skaki[x][y]=klik1kommati
                k=1
    pygame.display.update()
```

Γράφουμε και αποθηκεύουμε τον παραπάνω κώδικα π.χ. ως 7skaki08.py. Σε σχέση με τον προηγούμενο έχει σημαντικές αλλαγές. Τον τρέχουμε και βλέπουμε τη συμπεριφορά του.



Κλικ σε πύργο δημιουργεί κόκκινο πλαίσιο στο αντίστοιχο τετράγωνο. Στη συνέχεια δεύτερο κλικ σε γειτονικό τετράγωνο, προκαλεί κίνηση του πύργου σε αυτό. Δεύτερο κλικ στο ίδιο τετράγωνο, προκαλεί ακύρωση της πρόθεσης για κίνηση. Πρόκειται, δηλαδή, για μια αρχική υπεραπλουστευμένη εκδοχή σκακιού όπου μόνο λευκοί πύργοι μπορούν και μετακινούνται και μάλιστα μόνο κατά ένα τετράγωνο.

Σε σχέση με τον προηγούμενο κώδικα, καταρχήν παρατηρούμε ότι σε αρκετά σημεία έχουν γίνει κάποιες συγχωνεύσεις, ώστε να κερδηθούν γραμμές. Π.χ. οι αρχικές σταθερές έχουν καταλάβει πλέον λιγότερες γραμμές. Ένα νέο στοιχείο εκεί είναι μια νέα σταθερά, η LP. Με την εντολή **`LP = pygame.image.load('LeykosPyrgos.png')`**, αντιστοιχίζεται το αρχείο LeykosPyrgos.png που απεικονίζει έναν λευκό πύργο, στη σταθερά LP. Αυτό το αρχείο εικόνας, πρέπει απαραίτητα να βρίσκεται στο φάκελο της Python, όπως γενικότερα κάθε αρχείο εικόνας που θα χρησιμοποιηθεί αργότερα από το πρόγραμμα. Όπως και όλα τα αρχεία του βιβλίου αυτού, το αρχείο αυτό μπορεί κανείς να το κατεβάσει στην ιστοσελίδα www.7777777.gr. Στη συνέχεια πρέπει κανείς να το ρίξει στον φάκελο C:/Python33 και μόνο τότε μπορεί η παραπάνω εντολή να το χρησιμοποιήσει. Πλέον, κάθε αναφορά της σταθεράς LP στο πρόγραμμα, σημαίνει αναφορά αυτού του αρχείου, και στην πλειονότητα των περιπτώσεων οι αναφορές αυτές αφορούν εντολές σχεδίασης αυτού του αρχείου εικόνας σε συγκεκριμένο σημείο του παραθύρου εκτέλεσης. Βεβαίως, στη συνέχεια του βιβλίου, θα συμπεριληφθούν και άλλα αρχεία εικόνας, τα οποία θα αφορούν τα υπόλοιπα κομμάτια του σκακιού. Λίγο παρακάτω συναντούμε το εξής σετ εντολών:

```
skaki = [] #Arxikopoihsh skakieras me ekxwrhsh 0 se ola ta tetragwna
for i in range(TX):
    column = []
    for j in range(TY):
        column.append(0) #0 shmainei keno tetragwno
    skaki.append(column)
```

Αυτό που γίνεται στις παραπάνω εντολές, είναι να δημιουργείται μια δομή skaki, η οποία έχει 64 στοιχεία, διατεταγμένα 8 επί 8. Σε κάθε στοιχείο της δομής αυτής καταχωρείται αρχικά η τιμή 0. Για να είμαστε ακριβείς, στη δομή skaki πρόκειται για μια λίστα 8 στοιχείων, εκ των οποίων κάθε στοιχείο με τη σειρά του είναι μια λίστα που περιέχει 8 στοιχεία. Το παρακάτω σχήμα, μπορεί να μας διαφωτίσει περισσότερο.

$$\begin{bmatrix} \begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} & \begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} & \begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} & \begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} & \begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} & \begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} & \begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} & \begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} \end{bmatrix}$$

Στην πρώτη εντολή **`for`**, η μεταβλητή i θα πάρει διαδοχικά τις τιμές 0, 1, … 7. Σε κάθε τέτοια επανάληψη, θα δημιουργείται μια νέα λίστα στοιχείων (νέα στήλη στοιχείων), η οποία θα αποτελείται τελικά από 8 μηδενικά. Κάθε τέτοια λίστα προστίθεται ως στοιχείο στην αρχική

λίστα skaki. Ας παρατηρήσει κανείς ότι η λέξη **append** που ακολουθεί ονόματα λιστών, είναι μια εντολή-μέθοδος, η οποία εφαρμόζεται σε λίστες. Έχει ως αποτέλεσμα να προστίθεται στη λίστα που αναφέρεται πριν την τελεία, το στοιχείο που βρίσκεται ως όρισμα στις παρενθέσεις. Οι εσωτερικές λίστες περιέχουν μηδενικά, ενώ η εξωτερική λίστα περιέχει τις λίστες με τα μηδενικά. Η αναφορά σε συγκεκριμένο στοιχείο της δομής αυτής γίνεται π.χ. με την εντολή **skaki[2][2]=1** ή με την εντολή **skaki[7][7]=1**, οι οποίες ουσιαστικά αλλάζουν το περιεχόμενο των θέσεων (2,2) και (7,7) από 0 σε 1. Αυτό ακριβώς γίνεται στις εντολές της επόμενης γραμμής. Ας έχουμε κατά νου ότι το πάνω αριστερό στοιχείο της δομής skaki αναφέρεται με **skaki[0][0]** (και όχι με **skaki[1][1]**), ενώ το κάτω δεξί αναφέρεται με **skaki[7][7]** (και όχι με **skaki[8][8]**).

Αυτή η δομή skaki θα είναι η αόρατη ψυχή της σκακιέρας που θα εμφανίζεται στο παράθυρο εκτέλεσης. Κάθε μηδενικό, θα αντανακλάται με ένα άδειο τετράγωνο, ενώ κάθε άσσος θα αντανακλάται με έναν λευκό πύργο. Αργότερα θα δώσουμε κατάλληλες τιμές και στα υπόλοιπα κομμάτια του σκακιού, π.χ. 2 για ίππους. Ας προσπαθήσει κανείς να αντιλαμβάνεται κάθε στοιχείο από τα 64 της λίστας, ως μια μεταβλητή, διότι δεν θα γίνεται στη συνέχεια κάτι διαφορετικό από αυτό που γίνεται στις μεταβλητές. Τα περιεχόμενα της δομής skaki θα μεταβάλλονται ανάλογα των κινήσεων των κομματιών. Οι λίστες, κατά κάποιο τρόπο, αποτελούν μια ισχυρή επέκταση της έννοιας των μεταβλητών.

Λίγο παρακάτω, κάτω από το διπλό for που σχεδιάζει τη σκακιέρα, παρατηρούμε την εντολή **epif.blit(LP,(PER+2*(SZ+KENO),PER+2*(SZ+KENO),SZ,SZ))**. Την εντολή **blit** την έχουμε ξανασυναντήσει όταν μάθαμε για σχεδίαση αντικειμένων κειμένου πάνω σε επιφάνεια. Είναι μια μέθοδος που εφαρμόζεται γενικά σε αντικείμενα όπως η επιφάνεια σχεδίασης, όπου πάνω τους πρέπει να σχεδιαστούν άλλα πράγματα. Στην περίπτωσή μας, στην επιφάνεια epif θα σχεδιαστεί το αρχείο εικόνας που συμβολίζει η σταθερά LP, η οποία μπαίνει ως πρώτο όρισμα. Δεύτερο όρισμα μπαίνει μια τετράδα ακέραιων, η οποία καθορίζει το νοητό τετράγωνο μέσα στο οποίο θα γίνει η σχεδίαση. Οι εκφράσεις **PER+2*(SZ+KENO)** και **PER+2*(SZ+KENO)**, αφορούν τις θέσεις left και top του τετραγώνου (2,2), το οποίο είναι το τρίτο από επάνω και το τρίτο από αριστερά. Η τρίτη και τέταρτη έκφραση της τετράδας (SZ) αφορούν το πλάτος και το ύψος του νοητού τετραγώνου. Εννοείται, ότι πρέπει να έχει ληφθεί μέριμνα, ώστε το αρχείο εικόνας να έχει τις κατάλληλες διαστάσεις πλάτους και ύψους σε pixel, σύμφωνα με το χώρο που επιθυμούμε να καταλάβει στο παράθυρο εκτέλεσης. Στην περίπτωσή μας αυτό συμβαίνει. Τόσο το αρχείο εικόνας του λευκού πύργου, όσο και όλα τα αρχεία εικόνας που θα χρησιμοποιηθούν στη συνέχεια για τα υπόλοιπα κομμάτια, έχουν διαστάσεις 50x50 pixel. Η δεύτερη εντολή **epif.blit(LP,(PER+7*(SZ+KENO),PER+7*(SZ+KENO),SZ,SZ))** έχει ως αποτέλεσμα να σχεδιαστεί και ένας δεύτερος πύργος στη θέση (7,7).

Αμέσως μετά τις εντολές **blit**, παρατηρούμε την εντολή **k=1**. Η ονομασία της μεταβλητής k επιλέχθηκε ως αρχικό γράμμα της λέξης κατάσταση. Εκχώρηση με 1 σημαίνει ότι αρχικά το παιχνίδι μας βρίσκεται σε αναμονή του πρώτου κλικ. Εκχώρηση με 2 αργότερα, θα σημαίνει ότι το παιχνίδι μας θα βρίσκεται σε αναμονή του δεύτερου κλικ, το οποίο όταν γίνεται, θα ολοκληρώνει την κίνηση.

Μένει τώρα να εξηγηθεί το παρακάτω τμήμα, το οποίο έχει τροποποιηθεί αρκετά σε σχέση με το προηγούμενο πρόγραμμα και αφορά την περίπτωση έγκυρου κλικ σε κάποιο από τα τετράγωνα της σκακιέρας.

```
if k==1 and skaki[x][y]==1: #An sthn katastash 1 ginei klik se pyrgo
    klik1x=x; klik1y=y; klik1kommati=skaki[x][y]
    pygame.draw.rect(epif,RED,(PER+x*(SZ+KENO)-2,PER+y*(SZ+KENO)-2,SZ+3,SZ+3),2)
    k=2
elif k==2 and klik1x==x and klik1y==y: #An ginei deytero klik sto idio kommati
    pygame.draw.rect(epif,BLACK,(PER+klik1x*(SZ+KENO)-2,PER+klik1y*(SZ+KENO)-2,SZ+3,SZ+3),2)
    k=1
elif k==2 and ((klik1x==x-1 and klik1y==y) or(klik1x==x+1 and klik1y==y) \
       or (klik1x==x and klik1y==y-1) or(klik1x==x and klik1y==y+1) ) : #Egkyro deytero klik
    pygame.draw.rect(epif,BLACK,(PER+klik1x*(SZ+KENO)-2,PER+klik1y*(SZ+KENO)-2,SZ+3,SZ+3),2)
    pygame.draw.rect(epif,WHITE,(PER+klik1x*(SZ+KENO),PER+klik1y*(SZ+KENO),SZ,SZ),0)
    epif.blit(LP,(PER+x*(SZ+KENO),PER+y*(SZ+KENO),SZ,SZ))
    skaki[klik1x][klik1y]=0; skaki[x][y]=klik1kommati
    k=1
```

Πρέπει εδώ να γίνει μια σύντομη παρουσίαση της **εντολής if** και της σύνταξής της στη γλώσσα Python. Γενικά, η εντολή **`if`** κάνει έναν έλεγχο σε μια συνθήκη, και ανάλογα αν η τιμή της συνθήκης είναι αληθής ή ψευδής (**`True`** ή **`False`**), εκτελούνται ή όχι οι εντολές που ακολουθούν. Ακολουθείται η εξής σύνταξη: Δίπλα στη λέξη **`if`** μπαίνει μια συνθήκη. Εδώ θα κάνουμε μια μικρή παρένθεση, ώστε να εξηγηθούν πιο λεπτομερώς οι συνθήκες. Συνθήκες λέμε παραστάσεις, οι οποίες δημιουργούνται συνήθως με συγκριτικούς τελεστές π.χ. 23>20, 23==20, left > 500. Σημειώνεται εδώ ότι οι 6 συγκριτικοί τελεστές της Python είναι οι > (μεγαλύτερο από), < (μικρότερο από), >= (μεγαλύτερο ίσο από), <= (μικρότερο ίσο από), == (ίσο με), !=(όχι ίσο με). Συνθήκες δημιουργούνται επίσης με τον τελεστή **`in`** (ανήκει), π.χ. η έκφραση **`3 in range(10)`** επιστρέφει True (αληθής) καθώς το 3 ανήκει στους ακέραιους αριθμούς έως το 10. Στο θέμα των συνθηκών, ας ειπωθεί επίσης ότι μπορούν να δημιουργηθούν σύνθετες συνθήκες από δύο απλές. Π.χ. η έκφραση **`2>1 and 4>3`** επιστρέφει **`True`** (αληθής) καθώς και οι δύο απλούστερες συνθήκες είναι αληθείς. Οι τρεις τελεστές **`and`**, **`or`** και **`not`** ονομάζονται λογικοί τελεστές και χρησιμεύουν στην δημιουργία σύνθετων συνθηκών. Σύνθετη συνθήκη με λογικό τελεστή or, είναι αληθής αρκεί μια από τις δύο απλές συνθήκες να είναι αληθής. Δίπλα στην συνθήκη τοποθετείται πάντα άνω κάτω τελεία (:). Επανερχόμενοι στην σύνταξη, αν η συνθήκη δίπλα στην **`if`** είναι αληθής, τότε εκτελούνται οι εντολές που ακολουθούν σε εσοχή. Αν η συνθήκη είναι ψευδής, τότε αγνοούνται οι εντολές που ακολουθούν. Αν υπάρχει παρακάτω μια εντολή **`elif`**, τότε ελέγχεται η συνθήκη που υπάρχει δεξιά της. Αν αυτή είναι αληθής, τότε εκτελούνται οι εντολές που ακολουθούν σε εσοχή την **`elif`**. Η εντολή **`if`** και οι ενδεχόμενες **`elif`** απαιτείται να στοιχίζονται κατακόρυφα ακριβώς μεταξύ τους. Αντίθετα, οι εντολές που ακολουθούν την **`if`** ή τις **`elif`** απαιτείται να γράφονται με εσοχή. Υπάρχει περίπτωση να υπάρχει η λέξη **`else`** παρακάτω. Αυτό σημαίνει ότι αν καμία από τις παραπάνω συνθήκες στο **`if`** ή στις **`elif`** δεν ισχύει, τότε εκτελούνται οι εντολές που ακολουθούν την **`else`** σε εσοχή.

Στην περίπτωσή μας, με την **`if k==1 and skaki[x][y]==1:`** αρχικά ελέγχεται αν σε κατάσταση 1 (στο πρώτο κλικ) γίνεται κλικ σε τετράγωνο που περιέχει πύργο. Αν αυτό συμβεί, τότε στις μεταβλητές klik1x και klik1y αποθηκεύεται η θέση του τετραγώνου που έγινε το κλικ. Στην μεταβλητή klik1kommati αποθηκεύεται ο ακέραιος αριθμός που συμβολίζει το τρέχον κομμάτι, κάτι που μας το δίνει η δομή skaki στην τρέχουσα θέση (x,y), με την έκφραση **`klik1kommati=skaki[x][y]`**. Και με την **`pygame.draw.rect`** σχεδιάζεται ένα κόκκινο περίγραμμα που περιβάλλει το τρέχον τετράγωνο. Και τελικά η μεταβλητή k παίρνει τιμή 2.

Βέβαια, το GameLoop εκτελείται εκατοντάδες φορές το δευτερόλεπτο, κατά συνέπεια όταν γίνει το δεύτερο έγκυρο κλικ, αυτό αμέσως θα γίνει αντιληπτό και θα εκτελεστούν είτε οι

εντολές της πρώτης **elif**, είτε οι εντολές της δεύτερης **elif**. Οι εντολές της πρώτης **elif**, αφορούν την περίπτωση, όπου ο χρήστης κλικάρει στο ίδιο τετράγωνο που είχε κλικάρει και είχε κοκκινίσει στην αρχή. Σε τέτοια περίπτωση η συνθήκη **k==2 and klik1x==x and klik1y==y** θα είναι αληθής και το μόνο που θα γίνεται είναι να ξαναβαφεί το κόκκινο πλαίσιο με μαύρο χρώμα (δηλαδή θα επανέλθει το τετράγωνο στην αρχική του μορφή). Η δεύτερη **elif** αφορά την περίπτωση δεύτερου κλικ, το οποίο γίνεται σε ακριβώς γειτονικό τετράγωνο του πύργου. Παρατηρούμε ότι η σχετική συνθήκη είναι μεγάλη και απαιτήθηκε ο χαρακτήρας αλλαγής γραμμής (\ ), ώστε να γραφεί η συνθήκη καταλαμβάνοντας δύο γραμμές. Πρόκειται για μια χρήσιμη συντακτική σύμβαση της Python, η οποία μας επιτρέπει να μοιράζουμε μια μεγάλη εντολή σε δύο ή περισσότερες γραμμές. Οι εντολές που θα εκτελεστούν σε αυτήν την περίπτωση είναι πρώτον να βαφεί μαύρο το προηγουμένως κόκκινο βαμμένο πλαίσιο, δεύτερον να βαφεί κενό το τετράγωνο που βρισκόταν αρχικά ο πύργος (εννοείται ότι στις δύο προηγούμενες εντολές χρησιμοποιούνται οι συντεταγμένες klik1x και klik1y του πρώτου κλικ), και τρίτον θα σχεδιαστεί ο λευκός πύργος στη νέα του πλέον θέση (x,y) με χρήση της εντολής **blit**. Και βέβαια, πρέπει να ενημερωθεί κατάλληλα η δομή skaki με τις νέες τιμές στα σημεία της μετακίνησης. Μηδέν στο σημείο εκκίνησης του πύργου και ένα στο σημείο κατάληξής του. Η μεταβλητή klik1kommati περιέχει 1 όταν πρόκειται για λευκό πύργο. Εξάλλου, η μεταβλητή k πρέπει να επανέλθει στην τιμή 1, ώστε να επανέλθουμε σε κατάσταση αναμονής του πρώτου κλικ.

Δυο λόγια για τη σχεδίαση του κόκκινου πλαισίου. Παρατηρούμε ότι το πάχος του (τελευταίο όρισμα) είναι 2 pixel. Οι θέσεις left και top επίσης έχουν μετακινηθεί δύο επάνω και δύο αριστερά (έχει αφαιρεθεί το 2 στις συντεταγμένες του αντίστοιχου τετραγώνου, **PER+x*(SZ+KENO)-2,PER+y*(SZ+KENO)-2)**. Και επίσης, το μέγεθος το πλαισίου είναι SZ+3, δηλαδή κατά 3 pixel μεγαλύτερο του τετραγώνου της σκακιέρας. Αυτές ακριβώς είναι οι διαστάσεις και η θέση που πρέπει να έχει ένα πλαίσιο πάχους 2 pixel, εάν επιθυμούμε να περικλείει ακριβώς το αντίστοιχο τετράγωνο μεγέθους SZ. Γενικά, η σχεδίαση γραμμών με πάχος περιττό αριθμό pixel, γίνεται με αναφορά στο κέντρο της γραμμής. Όμως, σε σχεδίαση γραμμών με πάχος άρτιο αριθμό pixel (όπως στην περίπτωση του πλαισίου μας), επειδή υπάρχουν δύο αριθμοί γραμμών που διεκδικούν το κέντρο της γραμμής, η αναφορά πρέπει να γίνεται στην επάνω γραμμή από τις δύο όταν πρόκειται για οριζόντια σχεδίαση και στην αριστερή γραμμή από τις δύο όταν πρόκειται για κατακόρυφη σχεδίαση.

Ο βασικός κορμός για το πρόγραμμά μας είναι πλέον πανέτοιμος, καταλαμβάνοντας μόνο 54 γραμμές κώδικα, ένα αξιοσημείωτα μικρό νούμερο γραμμών. Αυτό που θα γίνει στη συνέχεια, θα είναι απλά οι κατάλληλες επεκτάσεις, ώστε να επιτύχουμε την ορθή κίνηση του πύργου (δηλαδή όχι μόνο κίνηση κατά ένα τετράγωνο αλλά κατά περισσότερα) και να συμπεριλάβουμε και την κίνηση των υπόλοιπων κομματιών του σκακιού.

ΚΕΦΑΛΑΙΟ 9, Λευκοί και μαύροι πύργοι στη σκακιέρα

Ας κάνουμε μερικές αλλαγές στον προηγούμενο κώδικα. Πρώτον, ας προσθέσουμε την εντολή **`MP = pygame.image.load('MayrosPyrgos.png')`** κάτω από την αντίστοιχη για τον λευκό πύργο. Με αυτό τον τρόπο μπαίνει στο παιχνίδι μας και ο μαύρος πύργος (αντιστοιχίζεται στη σταθερά MP) και βέβαια έχουμε φροντίσει να υπάρχει το αντίστοιχο αρχείο εικόνας MayrosPyrgos.png στο φάκελο της Python. Κατόπιν, ας προσθέσουμε το υποπρόγραμμα validMoves, ακριβώς κάτω από τον κώδικα του άλλου υποπρογράμματος xyFromMousexMousey. Θα δέχεται ως παράμετρους τη δομή skaki, την τρέχουσα θέση xx,yy του κομματιού στο ταμπλό και το είδος του κομματιού kommati (1 για λευκό πύργο, και -1 για μαύρο πύργο), και θα επιστρέφει τις επιτρεπτές κινήσεις ή χτυπήματα. Στην περίπτωσή μας, αρχικά μόνο τις επιτρεπτές κινήσεις και επιτρεπτά χτυπήματα ενός λευκού ή μαύρου πύργου.

```
def validMovesAndAttacks(skaki,xx,yy,kommati):
    '''Gia kommati poy vrisketai sto shmeio xx,yy epistrefetai lista
    me epitreptes kinhseis kathws kai lista me epitrepta xtyphmata.'''
    moves = []; attack = []
    if kommati==1 or kommati==-1: #Leykos h Mayros Pyrgos
        x1=xx-1 #Kinhseis pros aristera
        while x1>=0 and skaki[x1][yy]==0:
            moves.append([xx,yy,x1,yy])
            x1=x1-1
        if x1>=0 and skaki[x1][yy]*kommati<0: attack.append([xx,yy,x1,yy])
        x1=xx+1 #Kinhseis pros deksia
        while x1<TX and skaki[x1][yy]==0:
            moves.append([xx,yy,x1,yy])
            x1=x1+1
        if x1<TX and skaki[x1][yy]*kommati<0: attack.append([xx,yy,x1,yy])
        y1=yy-1 #Kinhseis pros epanw
        while y1>=0 and skaki[xx][y1]==0:
            moves.append([xx,yy,xx,y1])
            y1=y1-1
        if y1>=0 and skaki[xx][y1]*kommati<0: attack.append([xx,yy,xx,y1])
        y1=yy+1 #Kinhseis pros katw
        while y1<TY and skaki[xx][y1]==0:
            moves.append([xx,yy,xx,y1])
            y1=y1+1
        if y1<TY and skaki[xx][y1]*kommati<0: attack.append([xx,yy,xx,y1])
    return moves, attack
```

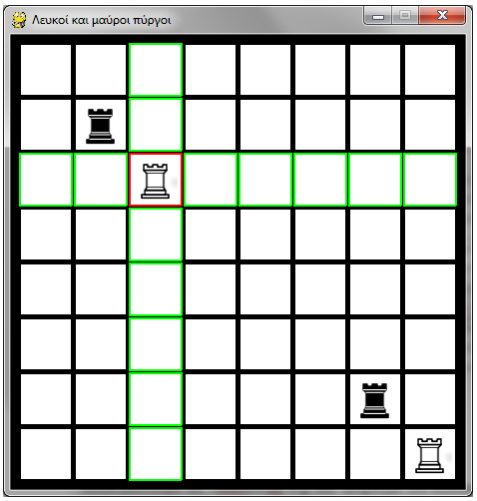
Ακριβώς κάτω από την validMovesAndAttacks προσθέτουμε τον παρακάτω κώδικα ο οποίος διαφέρει ελαφρά σε σχέση με το προηγούμενο πρόγραμμα.

```
pygame.init()
epif=pygame.display.set_mode((2*PER+TX*SZ+(TX-1)*KENO,2*PER+TY*SZ+(TY-1)*KENO))
pygame.display.set_caption('Λευκοί και μαύροι πύργοι')
skaki = [] #Arxikopoihsh skakieras me ekxwrhsh 0 se ola ta tetragwna
for i in range(TX):
    column = []
    for j in range(TY):
        column.append(0) #0 shmainei keno tetragwno
    skaki.append(column)
skaki[2][2]=1; skaki[7][7]=1 #Leykoi pyrgoi
skaki[1][1]=-1; skaki[6][6]=-1 #Mayroi pyrgoi
for i in range(TX): #Sxediash skakieras
    for j in range(TY):
        left = i * (SZ + KENO); top = j * (SZ + KENO)
        pygame.draw.rect(epif,WHITE,(PER+left,PER+top,SZ,SZ))
        if skaki[i][j]==1: epif.blit(LP,(PER+left,PER+top,SZ,SZ))
        if skaki[i][j]==-1: epif.blit(MP,(PER+left,PER+top,SZ,SZ))
k=1
turn=1 #Arxika exei seira o leykos kai turn==1. Otan paizei o mayros, turn==-1
while True: #GameLoop
```

Τροποποιούμε και τις εντολές του GameLoop όπως φαίνεται παρακάτω:

```
while True: #GameLoop
    mouseClicked = False
    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit(); sys.exit()
        elif event.type == MOUSEBUTTONUP: #Klik pontikioy
            mousex, mousey = event.pos; mouseClicked = True
    if mouseClicked == True:
        x, y = xyFromMousexMousey(mousex, mousey)
        if  x != None: #Egkyro klik
            if k==1 and skaki[x][y]*turn>0: #An sthn katastash 1 ginei klik se pyrgo
                klik1x=x; klik1y=y; klik1kommati=skaki[x][y]
                mov, att = validMovesAndAttacks(skaki,x,y,klik1kommati)
                pygame.draw.rect(epif,RED,(PER+x*(SZ+KENO)-2,PER+y*(SZ+KENO)-2,SZ+3,SZ+3),2)
                for mv in mov:
                    pygame.draw.rect(epif,GREEN,(PER+mv[2]*(SZ+KENO)-2,PER+mv[3]*(SZ+KENO)-2,SZ+3,SZ+3),2)
                for mv in att:
                    pygame.draw.rect(epif,BLUE,(PER+mv[2]*(SZ+KENO)-2,PER+mv[3]*(SZ+KENO)-2,SZ+3,SZ+3),2)
                k=2
            elif k==2 and klik1x==x and klik1y==y: #An ginei deytero klik sto idio kommati
                pygame.draw.rect(epif,BLACK,(PER+klik1x*(SZ+KENO)-2,PER+klik1y*(SZ+KENO)-2,SZ+3,SZ+3),2)
                for mv in mov:
                    pygame.draw.rect(epif,BLACK,(PER+mv[2]*(SZ+KENO)-2,PER+mv[3]*(SZ+KENO)-2,SZ+3,SZ+3),2)
                for mv in att:
                    pygame.draw.rect(epif,BLACK,(PER+mv[2]*(SZ+KENO)-2,PER+mv[3]*(SZ+KENO)-2,SZ+3,SZ+3),2)
                k=1
            elif k==2 and [klik1x,klik1y,x,y] in mov+att: #Egkyro deytero klik
                pygame.draw.rect(epif,BLACK,(PER+klik1x*(SZ+KENO)-2,PER+klik1y*(SZ+KENO)-2,SZ+3,SZ+3),2)
                pygame.draw.rect(epif,WHITE,(PER+klik1x*(SZ+KENO),PER+klik1y*(SZ+KENO),SZ,SZ),0)
                for mv in mov:
                    pygame.draw.rect(epif,BLACK,(PER+mv[2]*(SZ+KENO)-2,PER+mv[3]*(SZ+KENO)-2,SZ+3,SZ+3),2)
                for mv in att:
                    pygame.draw.rect(epif,BLACK,(PER+mv[2]*(SZ+KENO)-2,PER+mv[3]*(SZ+KENO)-2,SZ+3,SZ+3),2)
                if klik1kommati==1:
                    epif.blit(LP,(PER+x*(SZ+KENO),PER+y*(SZ+KENO),SZ,SZ))
                elif klik1kommati==-1:
                    epif.blit(MP,(PER+x*(SZ+KENO),PER+y*(SZ+KENO),SZ,SZ))
                skaki[klik1x][klik1y]=0; skaki[x][y]=klik1kommati
                k=1
                turn = turn*-1
    pygame.display.update()
```

Αποθηκεύουμε π.χ. ως 7skaki09.py και τρέχουμε τον κώδικα.



Παρατηρούμε ότι το πρόγραμμα έχει αποκτήσει μια αξιοσημείωτη λειτουργικότητα. Ξεκινά ο λευκός. Κλικ σε πύργο χρώματος που έχει σειρά να παίξει, έχει ως αποτέλεσμα να εμφανίζονται πράσινες οι επιτρεπτές κινήσεις του πύργου αυτού. Δεύτερο κλικ σε κάποιο πρασινισμένο τετράγωνο, προκαλεί τη σχετική κίνηση.

Πριν προχωρήσουμε στις επεξηγήσεις του κώδικα, είναι καλό σε αυτό το σημείο να δοθούν μερικές διευκρινήσεις για τον τρόπο χρήσης των λιστών, για τις οποίες μέχρι στιγμής ίσως δεν έχουν ειπωθεί πολλά.

ΛΙΣΤΕΣ

```
>>> a=[10,20,30]
>>> a[0]
10
>>> a[2]
30
>>>
```

```
>>> b=[[0,0],[1,1],[7,7]]
>>> b[2]
[7, 7]
>>>
```

Πηγαίνουμε στο κέλυφος της Python και πληκτρολογούμε **a=[10,20,30]**. Με αυτήν την εντολή δημιουργούμε μια λίστα όπως λέγεται, της οποίας το όνομα είναι a. Μια λίστα είναι ένα σύνολο διατεταγμένων αντικειμένων. Το πρώτο αντικείμενο της λίστας, δηλαδή το 10, προσπελαύνεται με **a[0]**, το δεύτερο με **a[1]**, το τρίτο με **a[2]**. Δοκιμάζουμε να τα προσπελάσουμε όπως φαίνεται στο παραπάνω σχήμα. Μια λίστα μπορεί να περιέχει λίστες! Δοκιμάζουμε το εξής: Πληκτρολογούμε **b=[[0,0],[1,1],[7,7]]** και στη συνέχεια **b[2]**. Εμφανίζεται **[7,7]** διότι αυτή η λίστα είναι το τρίτο στοιχείο της λίστας b.

Τώρα ίσως μπορούμε να αντιληφθούμε καλύτερα τι γίνεται μέσα στο υποπρόγραμμα validMovesAndAttacks. Οι εντολές **moves = []** και **attack = []** δημιουργούν αρχικά την κενή λίστα moves και την κενή λίστα attack. Παρακάτω, η λίστα moves θα γεμίσει με στοιχεία κάνοντας χρήση της εντολής **moves.append** και η λίστα attack το ίδιο κάνοντας χρήση της εντολής **attack.append**.

Στην πρώτη εντολή **if**, εξετάζεται αν πρόκειται για κομμάτι πύργου και μόνο τότε υπάρχει συνέχεια. Τέσσερις κατευθύνσεις θα ελεγχθούν: προς αριστερά, προς δεξιά, προς επάνω και προς τα κάτω. Ας δούμε τι γίνεται π.χ. όταν εξετάζουμε επιτρεπτές κινήσεις προς αριστερά. Αρχικά στην μεταβλητή x1 καταχωρείται η επόμενη οριζόντια συντεταγμένη προς αριστερά. Κατόπιν, η εντολή **while** ελέγχει ότι ήμαστε εντός ταμπλό (x1>=0) και ότι το προς εξέταση τετράγωνο είναι κενό (**skaki[x1][yy]==0**). Όσο ισχύουν αυτές οι δύο προϋποθέσεις, τότε επαναληπτικά εκτελούνται οι δύο παρακάτω εντολές σε εσοχή της **while**. Με την **moves.append** θα προστεθεί στη λίστα moves η κίνηση **[xx,yy,x1,yy]**, ενώ με την εντολή **x1=x1-1** θα προχωρήσουμε αριστερά στο επόμενο προς εξέταση τετράγωνο. Θα ξαναγίνει ο έλεγχος των δύο προϋποθέσεων. Αν παραμένουμε εντός ταμπλό και αν το επόμενο τετράγωνο είναι κενό, τότε προστίθεται και αυτό το τετράγωνο στη λίστα moves και προχωράμε στο επόμενο τετράγωνο. Ας σημειωθεί ότι μια κίνηση χαρακτηρίζεται ως μια τετράδα αριθμών, όπου οι δύο πρώτοι αφορούν τις συντεταγμένες εκκίνησης και οι δύο τελευταίες αφορούν τις συντεταγμένες προορισμού. xx και yy θα είναι πάντα οι συντεταγμένες εκκίνησης, x1 και yy θα είναι οι συντεταγμένες προορισμού, όπου βέβαια η συντεταγμένη x1 ενδεχομένως θα αλλάζει κάθε φορά εφόσον πρόκειται για κίνηση προς αριστερά.

Την **εντολή while**, την παρατηρήσαμε βέβαια στο GameLoop, όπου εμφανίζεται ως άκρως απαραίτητο συστατικό προκειμένου να επιτευχθεί η συνεχής αλληλεπίδραση του προγράμματος με το χρήστη. Υπενθυμίζεται ξανά ο τρόπος λειτουργίας της: δεξιά της λέξης **while** (μεταφράζεται κατάλληλα με την ελληνική λέξη Όσο), υπάρχει πάντα μια συνθήκη. Έχουμε μιλήσει για συνθήκες στο προηγούμενο κεφάλαιο 8 όπου επεξηγήθηκε η εντολή **if**. Συνθήκες παίρνουν μια τιμή αληθείας: είτε **True**, είτε **False**. Όσο η συνθήκη της εντολής **while** παραμένει αληθής, εκτελούνται ξανά και ξανά οι εντολές σε εσοχή που ακολουθούν. Όταν εκτελεστεί και η τελευταία εντολή σε εσοχή, ο έλεγχος ξαναγυρνά πίσω στην εντολή **while**, ώστε να ελεγχθεί και πάλι η συνθήκη. Όσο η συνθήκη παραμένει αληθής, οι εντολές

σε εσοχή εκτελούνται ξανά και ξανά. Όταν πάψει η συνθήκη να είναι αληθής, τότε δεν εκτελούνται πλέον οι εντολές σε εσοχή αλλά οι επόμενες, δηλαδή εκείνες που βρίσκονται στην ίδια κατακόρυφη στοίχιση με την **while**.

Στην περίπτωσή μας, εννοείται ότι κάποια στιγμή η συνθήκη θα γίνει ψευδής, το αργότερο όταν βρεθούμε αριστερά εκτός σκακιέρας καθώς η μεταβλητή x1 συνεχώς μειώνεται. Ίσως βέβαια η συνθήκη γίνει ψευδής και νωρίτερα, π.χ. αν συναντηθεί κομμάτι, δηλαδή μη κενό τετράγωνο. Αυτό ακριβώς ελέγχει η εντολή **if** που ακολουθεί την **while**. Αν φεύγοντας από τις επαναληπτικές εντολές της **while** η μεταβλητή x1 είναι ακόμα εντός ταμπλό, τότε αυτό σημαίνει ότι συναντήθηκε κομμάτι (λευκό ή μαύρο). Ελέγχεται τώρα αν στο τρέχον τετράγωνο με συντεταγμένες x1 και yy βρίσκεται αντίπαλο κομμάτι. Σε τέτοια περίπτωση πρέπει να ενημερωθεί η λίστα attack με την κίνηση χτυπήματος. Επειδή τα λευκά κομμάτια έχουν θετική τιμή, ενώ τα μαύρα αρνητική, όταν το γινόμενο **skaki[x1][yy]*kommati** είναι αρνητικό, τότε μόνο θα προστεθεί στη λίστα attack η σχετική κίνηση. Υπενθυμίζεται από τα Μαθηματικά ότι γινόμενο θετικού με αρνητικό αριθμό επιστρέφει πάντα αρνητικό αποτέλεσμα, ενώ γινόμενο ομόσημων αριθμών επιστρέφει πάντα θετικό αριθμό. Παρόμοια γίνεται η εξέταση για κινήσεις προς τα δεξιά, προς τα επάνω και προς τα κάτω και συνεχίζουν να ενημερώνονται οι λίστες moves και attack.

Ας παρακολουθήσουμε αναλυτικά πώς λειτουργεί το πρόγραμμα: Στο κυρίως πρόγραμμα όταν στην κατάσταση αναμονής του πρώτου κλικ γίνει κλικ σε τετράγωνο με κομμάτι χρώματος που έχει σειρά να παίξει, καλείται να εκτελεστεί και το υποπρόγραμμα **validMovesAndAttacks(skaki,x,y,klik1kommati)**. Οι παράμετροι με τις οποίες καλείται, είναι η δομή skaki, οι συντεταγμένες του κομματιού x και y και η τιμή klik1kommati που χαρακτηρίζει το είδος του κομματιού. Οι τιμές αυτών των παραμέτρων θα μεταφερθούν ως τιμές στις μεταβλητές skaki, xx, yy και kommati του υποπρογράμματος validMovesAndAttacks και θα μεταφερθεί η ροή εκτέλεσης εντολών στο υποπρόγραμμα. Θα εκτελεστούν οι εντολές του υποπρογράμματος validMovesAndAttacks, οι οποίες έχουν ως αποτέλεσμα δύο λίστες. Μια λίστα με επιτρεπτές κινήσεις και μια λίστα με επιτρεπτά χτυπήματα. Στη συνέχεια, η ροή εκτέλεσης εντολών φεύγει από το υποπρόγραμμα και μεταφέρεται στο σημείο που κλήθηκε το υποπρόγραμμα. Παρατηρούμε ότι οι επιστρεφόμενες κινήσεις αποθηκεύονται στις λίστες mov και att.

Η **pygame.draw.rect(epif,RED,(PER+x*(SZ+KENO)-2,PER+y*(SZ+KENO)-2,SZ+3,SZ+3),2)** σχεδιάζει ένα κόκκινο πλαίσιο στο τετράγωνο με το κομμάτι που παίζει. Στη συνέχεια, με την εντολή **for mv in mov: pygame.draw.rect(…)** σχεδιάζονται πράσινα πλαίσια σε όλα τα τετράγωνα που επιτρέπεται κίνηση και με την εντολή **for mv in att: pygame.draw.rect(…)** σχεδιάζονται μπλε πλαίσια σε όλα τα τετράγωνα που επιτρέπεται χτύπημα. Στη συνέχεια, η μεταβλητή k παίρνει την τιμή 2, κάτι που σημαίνει ότι μεταβαίνουμε σε κατάσταση αναμονής του δεύτερου κλικ.

Το Gameloop θα συνεχίσει, η μεταβλητή mouseClicked θα γίνει ψευδής (**False**) και θα ξαναγίνει παιχνίδι μόνο όταν γίνει κλικ σε κατάλληλο σημείο. Με κλικ στο ίδιο σημείο του κομματιού, επανερχόμαστε στην ίδια κατάσταση, σβήνουν το κόκκινο, τα πράσινα και τα μπλε πλαίσια και η μεταβλητή k παίρνει την αρχική της τιμή 1. Με κλικ σε επιτρεπτό τετράγωνο κίνησης ή χτυπήματος, συμβαίνουν τα εξής: Η συνθήκη **k==2 and [klik1x,klik1y,x,y] in mov+att** θα είναι αληθής (**True**), οπότε εκτελούνται οι εντολές σε εσοχή: Αρχικά αδειάζει το τετράγωνο του κομματιού με τις δύο εντολές

pygame.draw.rect(epif,BLACK,…) **pygame.draw.rect(drawSurface,WHITE…).** Στη συνέχεια με δύο εντολές **for** μαυρίζουν (επαναφέρονται) όλα τα πράσινα και μπλε πλαίσια. Κατόπιν επανασχεδιάζεται το κομμάτι στη νέα θέση [x,y] που έγινε κλικ, ανάλογα βέβαια με την τιμή της μεταβλητής klik1kommati. Με τιμή 1, η εντολή **epif.blit** σχεδιάζει λευκό πύργο και με τιμή -1, η εντολή **epif.blit** σχεδιάζει μαύρο πύργο. Κατόπιν η δομή skaki επικαιροποιείται στις θέσεις **skaki[klik1x][klik1y]** και **skaki[x][y]** με τιμές 0 και klik1kommati αντίστοιχα. Με την εντολή **k=1** επανερχόμαστε σε κατάσταση αναμονής του πρώτου κλικ και με την εντολή **turn = turn*-1** δηλώνεται ότι πλέον έχει σειρά να παίξει το άλλο χρώμα.

ΚΕΦΑΛΑΙΟ 10, Προσθέτοντας αξιωματικούς στη σκακιέρα

Οι ράγες είναι πλέον έτοιμες. Το τρενάκι μας μπορεί πλέον ν' αρχίσει να κυλάει. Προφανώς θα γίνει καλύτερο, προσθέτοντας βαγόνια. Ας προσθέσουμε λοιπόν στο πρόγραμμά μας και τη λειτουργία των αξιωματικών. Ως γνωστόν, οι αξιωματικοί προχωρούν διαγώνια προς κάθε κατεύθυνση.

Κάνουμε Αποθήκευση ως… στον προηγούμενο κώδικα και ονομάζουμε το νέο πρόγραμμα π.χ. 7skaki10.py. Η πρώτη μας αλλαγή στον κώδικα έχει βέβαια να κάνει με την προσθήκη γραμμών για ενσωμάτωση των αρχείων εικόνας με τον λευκό και τον μαύρο αξιωματικό. Στο σημείο που έχουν δηλωθεί οι πύργοι, θα πρέπει να δηλωθούν και οι αξιωματικοί.

```
LP=pygame.image.load('LeykosPyrgos.png'); MP=pygame.image.load('MayrosPyrgos.png');
LA=pygame.image.load('LeykosAksiomatikos.png'); MA=pygame.image.load('MayrosAksiomatikos.png')
```

Εννοείται ότι τα αρχεία εικόνας για τους αξιωματικούς, πρέπει να τοποθετηθούν στον φάκελο της Python και έχουμε ξαναπεί ότι όλα τα απαιτούμενα αρχεία μπορούν να κατεβούν από την ιστοσελίδα 7777777.gr . Κάθε μελλοντική αναφορά στις σταθερές LA ή MA θα αφορά αξιωματικούς, συνήθως τη σχεδίασή τους.

Η σημαντικότερη όμως προσθήκη που πρέπει να γίνει, έχει να κάνει με το υποπρόγραμμα validMovesAndAttacks. Αυτό το υποπρόγραμμα που είχε ως σκοπό του να επιστρέφει επιτρεπτές κινήσεις πύργων, τώρα πρέπει να εμπλουτιστεί, ώστε να επιστρέφει και επιτρεπτές κινήσεις αξιωματικών. Ακριβώς πριν την τελευταία εντολή return moves, attack του υποπρογράμματος, πρέπει να προστεθεί ο παρακάτω κώδικας.

```
    if y1<TY and skaki[xx][y1]*kommati<0: attack.append([xx,yy,xx,y1])
elif kommati==3 or kommati==-3: #Leykos h Mayros Aksiwmatikos
    x1=xx-1; y1=yy-1 #Kinhseis pros panw aristera
    while x1>=0 and y1>=0 and skaki[x1][y1]==0:
        moves.append([xx,yy,x1,y1])
        x1=x1-1; y1=y1-1
    if x1>=0 and y1>=0 and skaki[x1][y1]*kommati<0: attack.append([xx,yy,x1,y1])
    x1=xx+1; y1=yy-1 #Kinhseis pros panw deksia
    while x1<TX and y1>=0 and skaki[x1][y1]==0:
        moves.append([xx,yy,x1,y1])
        x1=x1+1; y1=y1-1
    if x1<TX and y1>=0 and skaki[x1][y1]*kommati<0: attack.append([xx,yy,x1,y1])
    x1=xx+1; y1=yy+1 #Kinhseis pros katw deksia
    while x1<TX and y1<TY and skaki[x1][y1]==0:
        moves.append([xx,yy,x1,y1])
        x1=x1+1; y1=y1+1
    if x1<TX and y1<TY and skaki[x1][y1]*kommati<0: attack.append([xx,yy,x1,y1])
    x1=xx-1; y1=yy+1 #Kinhseis pros katw aristera
    while x1>=0 and y1<TY and skaki[x1][y1]==0:
        moves.append([xx,yy,x1,y1])
        x1=x1-1; y1=y1+1
    if x1>=0 and y1<TY and skaki[x1][y1]*kommati<0: attack.append([xx,yy,x1,y1])
return moves, attack
```

Έχει σημασία η εντολή **elif kommati==3** … να στοιχιστεί στην ίδια απόσταση από αριστερά με την εντολή **if kommati==1** … και στην ίδια απόσταση από αριστερά με την τελική **return**, δηλαδή να υπάρχει κατακόρυφη συμφωνία στη στοίχιση. Η συνθήκη **kommati==3** δηλώνει ότι το κομμάτι που παίζει έχει κωδικό 3. Έχουμε ήδη πει ότι ο λευκός πύργος θα συμβολίζεται με κωδικό 1 και ο μαύρος πύργος με κωδικό -1. Γενικά τηρήθηκε η αρχική τοποθέτηση των κομματιών στη σκακιέρα ως κωδικοποίηση. 1 για λευκό πύργο, 2 για ίππο, 3 για αξιωματικό, 4 για βασίλισσα, 5 για βασιλιά και 6 για πιόνι. Όλοι οι κωδικοί γίνονται αρνητικοί όταν πρόκειται για μαύρα κομμάτια, π.χ. ο μαύρος αξιωματικός έχει κωδικό -3. Δηλαδή, πλέον η validMovesAndAttacks ελέγχει δύο περιπτώσεις: Πρώτον, αν το κομμάτι είναι πύργος και δεύτερον αν το κομμάτι που παίζει είναι αξιωματικός. Το κομμάτι κώδικα που προστέθηκε, γίνεται εύκολα κατανοητό αν έχει γίνει κατανοητό το πρώτο κομμάτι κώδικα που αφορά την κίνηση πύργου. Κινείται στην ίδια λογική. Το μόνο που διαφέρει είναι ότι οι μεταβλητές x1 και y1 αλλάζουν κατάλληλα, ώστε να παίρνουν την επόμενη προς εξέταση διαγώνια θέση. Με τη διαγώνια κίνηση, όσο ήμαστε εντός σκακιέρας και επίσης το τετράγωνο που εξετάζεται είναι κενό, τότε πρόκειται για επιτρεπτή κίνηση που προστίθεται στη λίστα moves. Όταν σταματήσει το **while**, ελέγχεται ο τρόπος που βγήκαμε από το **while**. Αν δεν βγήκαμε εξ' αιτίας του ότι συναντήσαμε το άκρο της σκακιέρας, αλλά εξ' αιτίας ότι συναντήσαμε αντίπαλο κομμάτι, τότε αυτό το κομμάτι μπορεί να χτυπηθεί. Έτσι, η κίνηση **[xx,yy,x1,y1]** θα προστεθεί στη λίστα των επιτρεπτών χτυπημάτων. Αυτός ο τελευταίος έλεγχος, πραγματοποιείται με την συνθήκη **skaki[x1][y1]*kommati<0**. Αυτό μας λέει ότι, αν ο κωδικός του κομματιού που βρίσκεται στην προς εξέταση θέση και ο κωδικός του κομματιού που παίζει είναι ετερόσημοι (δηλαδή ο ένας θετικός και ο άλλος αρνητικός), τότε το γινόμενο των κωδικών είναι αρνητικό και άρα πράγματι επιτρέπεται το χτύπημα. Αν το γινόμενο βγαίνει θετικό, τότε πρόκειται για κομμάτια ίδιου χρώματος και άρα απαγορεύεται τέτοια κίνηση.

Λίγο παρακάτω στον κώδικα, πρέπει να γίνουν οι παρακάτω αλλαγές.

```
skaki[1][1]=-1; skaki[6][6]=-1 #Mayroi pyrgoi
skaki[3][4]=3; skaki[3][7]=3 #Leykoi Aksiwmatikoi
skaki[1][4]=-3; skaki[1][5]=-3 #Mayroi Aksiwmatikoi
for i in range(TX): #Sxediash skakieras
    for j in range(TY):
        left = i * (SZ + KENO); top = j * (SZ + KENO)
        pygame.draw.rect(epif,WHITE,(PER+left,PER+top,SZ,SZ))
        if skaki[i][j]==1: epif.blit(LP,(PER+left,PER+top,SZ,SZ))
        elif skaki[i][j]==-1: epif.blit(MP,(PER+left,PER+top,SZ,SZ))
        elif skaki[i][j]==3: epif.blit(LA,(PER+left,PER+top,SZ,SZ))
        elif skaki[i][j]==-3: epif.blit(MA,(PER+left,PER+top,SZ,SZ))
```

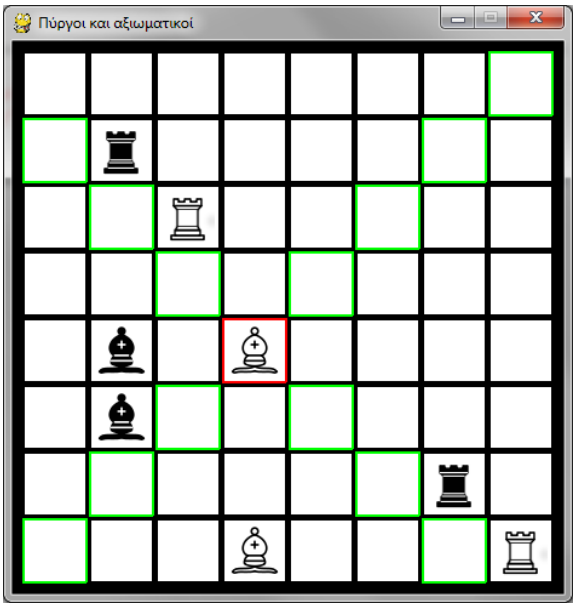
Πρώτον, προσθέτουμε δύο λευκούς και δύο μαύρους αξιωματικούς στη σκακιέρα μας. Αυτό γίνεται απλά, αλλάζοντας στην δομή skaki τους κωδικούς σε τέσσερα τετράγωνα, δίνοντας τιμές 3 ή -3. Δεύτερον, παρακάτω στην εντολή του διπλού **for** κατά τη σχεδίαση των 64 τετραγώνων, πρέπει να προστεθούν δύο γραμμές: Όταν ο κωδικός σε κάποιο τετράγωνο στη δομή skaki είναι 3, τότε στην αντίστοιχη θέση σχεδιάζεται λευκός αξιωματικός. Και όταν ο κωδικός σε κάποιο τετράγωνο στη δομή skaki είναι -3, τότε στην αντίστοιχη θέση σχεδιάζεται μαύρος αξιωματικός. Έχουμε ήδη εξηγήσει την εντολή **blit** η οποία αναλαμβάνει να κάνει αυτή τη δουλειά. Η πρώτη παράμετρός της, είναι βέβαια η σταθερά που δηλώνει το αρχείο εικόνας που θα σχεδιαστεί, δηλαδή είτε LA είτε MA, ενώ ακολουθεί μια τετράδα αριθμών που αφορά την ακριβή θέση και το μέγεθος σχεδίασης.

Τέλος, μέσα στο GameLoop, πρέπει να γίνουν οι κατάλληλες προσθήκες, ώστε όταν ολοκληρώνεται μια έγκυρη κίνηση αξιωματικού, να σχεδιάζεται αυτός στη νέα του θέση.

```
elif k==2 and [klik1x,klik1y,x,y] in mov+att: #Egkyro deytero klik
    pygame.draw.rect(epif,BLACK,(PER+klik1x*(SZ+KENO)-2,PER+klik1y*(SZ+KENO)-2,SZ+3,SZ+3),2)
    pygame.draw.rect(epif,WHITE,(PER+klik1x*(SZ+KENO),PER+klik1y*(SZ+KENO),SZ,SZ),0)
    pygame.draw.rect(epif,WHITE,(PER+x*(SZ+KENO),PER+y*(SZ+KENO),SZ,SZ),0)
    for mv in mov:
        pygame.draw.rect(epif,BLACK,(PER+mv[2]*(SZ+KENO)-2,PER+mv[3]*(SZ+KENO)-2,SZ+3,SZ+3),2)
    for mv in att:
        pygame.draw.rect(epif,BLACK,(PER+mv[2]*(SZ+KENO)-2,PER+mv[3]*(SZ+KENO)-2,SZ+3,SZ+3),2)
    if klik1kommati==1:
        epif.blit(LP,(PER+x*(SZ+KENO),PER+y*(SZ+KENO),SZ,SZ))
    elif klik1kommati==-1:
        epif.blit(MP,(PER+x*(SZ+KENO),PER+y*(SZ+KENO),SZ,SZ))
    elif klik1kommati==3:
        epif.blit(LA,(PER+x*(SZ+KENO),PER+y*(SZ+KENO),SZ,SZ))
    elif klik1kommati==-3:
        epif.blit(MA,(PER+x*(SZ+KENO),PER+y*(SZ+KENO),SZ,SZ))
    skaki[klik1x][klik1y]=0; skaki[x][y]=klik1kommati
    k=1
    turn = turn*-1
```

Παρατηρούμε τις κατάλληλες προσθήκες **`elif klik1kommati==3`** και **`elif klik1kommati==-3`** στην εντολή **`if`**. Η μεταβλητή klik1kommati περιέχει τον κωδικό του κομματιού που έχει γίνει το αρχικό πρώτο κλικ. Με βάση αυτόν τον κωδικό, γίνεται και η σχεδίαση του κομματιού στην τελική του θέση. Παρατηρούμε επίσης ότι προστέθηκε προηγουμένως και η γραμμή κώδικα **`pygame.draw.rect(epif, WHITE,(PER+x*(SZ+KENO),PER+y*(SZ+KENO),SZ,SZ),0)`**. Με αυτήν, το τετράγωνο προορισμού αρχικοποιείται ως κενό, διότι αλλιώς σε περίπτωση χτυπήματος θα σχεδιάζεται π.χ. ένας αξιωματικός πάνω από πύργο.

Τρέχοντας τον κώδικα, πρέπει να δούμε κάτι σαν το παρακάτω. Αν κάποιος το επιθυμεί, μπορεί πολύ εύκολα να επέμβει στον κώδικα και να τοποθετήσει τους πύργους και τους αξιωματικούς σε διαφορετικές θέσεις, ή μάλιστα να τοποθετήσει αρχικά και περισσότερα κομμάτια (πύργους και αξιωματικούς) στη σκακιέρα από αυτά που φαίνονται στο σχήμα.



ΚΕΦΑΛΑΙΟ 11, Υποπρόγραμμα για σχεδίαση κομματιών

Ίσως παρατηρήσει κανείς ότι τα κομμάτια κώδικα που σχεδιάζουν τους πύργους και τους αξιωματικούς, επαναλαμβάνονται στον κώδικα δύο φορές. Μια φορά στην εκκίνηση του παιχνιδιού και ξανά όταν παιχτεί έγκυρη κίνηση. Μπορεί κανείς να συντομεύσει τον κώδικα και να κάνει χρήση υποπρογράμματος που θα έχει ως αποστολή τη σχεδίαση κομματιών.

Στον προηγούμενο κώδικα κάνουμε τις παρακάτω απαραίτητες βελτιώσεις.

```
def drawKommati(xx,yy):
    ''' Sxediazetai to tetragwno kai to kommati ths theshs (xx,yy), symfwna me th domh skaki.
    An to tetragwno einai keno, sxediazetai to katallhlo xrwma tetragwnoy leyko h gkri'''
    if (xx+yy)%2==0: pygame.draw.rect(epif,WHITE,(PER+xx*(SZ+KENO),PER+yy*(SZ+KENO),SZ,SZ),0)
    else: pygame.draw.rect(epif,GRAY,(PER+xx*(SZ+KENO),PER+yy*(SZ+KENO),SZ,SZ),0)
    if skaki[xx][yy]==1: epif.blit(LP, (PER + xx*(SZ+KENO),PER + yy*(SZ+KENO),SZ,SZ))
    elif skaki[xx][yy]==-1: epif.blit(MP, (PER + xx*(SZ+KENO),PER + yy*(SZ+KENO),SZ,SZ))
    elif skaki[xx][yy]==2: epif.blit(LI, (PER + xx*(SZ+KENO),PER + yy*(SZ+KENO),SZ,SZ))
    elif skaki[xx][yy]==-2: epif.blit(MI, (PER + xx*(SZ+KENO),PER + yy*(SZ+KENO),SZ,SZ))
    elif skaki[xx][yy]==3: epif.blit(LA, (PER + xx*(SZ+KENO),PER + yy*(SZ+KENO),SZ,SZ))
    elif skaki[xx][yy]==-3: epif.blit(MA, (PER + xx*(SZ+KENO),PER + yy*(SZ+KENO),SZ,SZ))
    elif skaki[xx][yy]==4: epif.blit(LV, (PER + xx*(SZ+KENO),PER + yy*(SZ+KENO),SZ,SZ))
    elif skaki[xx][yy]==-4: epif.blit(MV, (PER + xx*(SZ+KENO),PER + yy*(SZ+KENO),SZ,SZ))
    elif skaki[xx][yy]==5: epif.blit(LR, (PER + xx*(SZ+KENO),PER + yy*(SZ+KENO),SZ,SZ))
    elif skaki[xx][yy]==-5: epif.blit(MR, (PER + xx*(SZ+KENO),PER + yy*(SZ+KENO),SZ,SZ))
    elif skaki[xx][yy]==6: epif.blit(LS, (PER + xx*(SZ+KENO),PER + yy*(SZ+KENO),SZ,SZ))
    elif skaki[xx][yy]==-6: epif.blit(MS, (PER + xx*(SZ+KENO),PER + yy*(SZ+KENO),SZ,SZ))
```

Το παραπάνω κομμάτι κώδικα, μπορεί να γραφεί οπουδήποτε, συστήνεται όμως να γραφεί μαζί με τα άλλα υποπρογράμματα, π.χ. κάτω από το υποπρόγραμμα validMovesAndAttacks. Πρόκειται για το υποπρόγραμμα drawKommati, το οποίο ανάλογα της τιμής του κωδικού κομματιού που βρίσκεται στη θέση (xx,yy) της δομής skaki, σχεδιάζει στο αντίστοιχο τετράγωνο το κατάλληλο σκιτσάκι πύργου, αξιωματικού κτλ. Παρατηρούμε ότι στο υποπρόγραμμα έχουν συμπεριληφθεί όλα τα κομμάτια του σκακιού (πιόνια, ίπποι,…), παρόλο που δεν θα γίνει χρήση τους ακόμα. Σημειώνεται ότι πριν τη σχεδίαση του κομματιού, σχεδιάζεται και το τετράγωνο στο οποίο θα πατήσει το κομμάτι και μάλιστα στο κατάλληλο χρώμα, λευκό ή μαύρο. Αυτό γίνεται στην πρώτη εντολή if. Αν το άθροισμα των συντεταγμένων του τετραγώνου διαιρείται ακριβώς με το 2 (**`if (xx+yy)%2==0`**) τότε το τετράγωνο σχεδιάζεται λευκό (δεύτερη παράμετρος WHITE), αλλιώς σχεδιάζεται μαύρο (για την ακρίβεια γκρι, δεύτερη παράμετρος GRAY).

Δυο λόγια για τους **τελεστές // και %** της Python, που ονομάζονται και τελεστής ακέραιας διαίρεσης και τελεστής υπολοίπου αντίστοιχα (ονομάζονται επίσης div και mod). Εφαρμόζονται σε δύο αριθμούς. Όταν ο // εφαρμόζεται σε δύο ακέραιους θετικούς, επιστρέφει το ακέραιο μέρος του πηλίκου τους. Όταν ο % εφαρμόζεται σε δύο ακέραιους θετικούς, επιστρέφει το ακέραιο υπόλοιπο της διαίρεσής τους. Καλό είναι να πειραματιστούμε στο κέλυφος της Python, πληκτρολογώντας π.χ. τα εξής:

```
>>> 5 // 2
2
>>> 7 // 2
3
>>> 10 // 3
3
>>> 11 // 3
3
>>> 100 // 40
2
>>>
```

```
>>> 5 % 2
1
>>> 7 % 2
1
>>> 10 % 3
1
>>> 11 % 3
2
>>> 100 % 40
20
>>>
```

Για παράδειγμα στην τελευταία πράξη div της πρώτης στήλης υπολογίζεται πόσες ακέραιες φορές χωράει το 40 στο 100 ενώ στην τελευταία πράξη mod της δεύτερης στήλης επιστρέφεται το 20 καθώς αυτό περισσεύει μετά την ακέραια διαίρεση των αριθμών 100 και 40.

Δυο λόγια για τη σταθερά χρώματος GRAY=(177,177,177), η οποία θα χρησιμοποιηθεί στον γκρι χρωματισμό των μισών τετραγώνων. Γενικότερα, όταν οι τρεις αριθμοί που δηλώνουν τις συμμετοχές των βασικών χρωμάτων κόκκινο, πράσινο και μπλε είναι ίδιοι, τότε πρόκειται για γκρι χρωματισμό. Όταν οι αριθμοί πλησιάζουν το 0, ο γκρι χρωματισμός

πλησιάζει το μαύρο, ενώ όταν οι αριθμοί πλησιάζουν το 255, ο γκρι χρωματισμός πλησιάζει το λευκό.

Λίγο παρακάτω, για τη σχεδίαση σκακιέρας αρκεί πλέον το εξής:

```
for i in range(TX): #Sxediash skakieras
    for j in range(TY):
        drawKommati(i,j)
```

Και μέσα στο GameLoop στο σημείο που ελέγχεται ότι έγινε έγκυρο δεύτερο κλικ, επίσης θα συντομευθεί ο κώδικας ως εξής:

```
elif k==2 and [klik1x,klik1y,x,y] in mov+att: #Egkyro deytero klik
    pygame.draw.rect(epif,BLACK,(PER+klik1x*(SZ+KENO)-2,PER+klik1y*(SZ+KENO)-2,SZ+3,SZ+3),2)
    for mv in mov:
        pygame.draw.rect(epif,BLACK,(PER+mv[2]*(SZ+KENO)-2,PER+mv[3]*(SZ+KENO)-2,SZ+3,SZ+3),2)
    for mv in att:
        pygame.draw.rect(epif,BLACK,(PER+mv[2]*(SZ+KENO)-2,PER+mv[3]*(SZ+KENO)-2,SZ+3,SZ+3),2)
    skaki[klik1x][klik1y]=0; skaki[x][y]=klik1kommati
    drawKommati(klik1x,klik1y)
    drawKommati(x,y)
    k=1
    turn = turn*-1
```

Δηλαδή, πρώτα ενημερώνεται η δομή skaki με τα επίκαιρα δεδομένα και στη συνέχεια σχεδιάζονται τα αντίστοιχα τετράγωνα βάση αυτών των αλλαγών. Επίσης, πριν τις εντολές **for** δεν χρειάζονται οι εντολές σχεδίασης των τετραγώνων εκκίνησης και προορισμού, διότι αυτά θα σχεδιάζονται εντός του υποπρογράμματος drawKommati.

Αποθηκεύουμε π.χ. ως 7Skaki11.py και τρέχουμε το πρόγραμμα. Λογικά, δεν πρέπει να δούμε διαφορετικά πράγματα από εκείνα που είδαμε στο πρόγραμμα 7Skaki10.py. Το κέρδος που αποκομίσαμε έχει περισσότερο να κάνει με την αύξηση της εμπειρίας μας σε σχέση με τα υποπρογράμματα και τη χρησιμότητά τους.

ΚΕΦΑΛΑΙΟ 12, Πύργοι, αξιωματικοί και ίπποι στη σκακιέρα

Ας προσθέσουμε στο πρόγραμμά μας και άλογα, τα οποία οι σκακιστές συνηθίζουν να ονομάζουν ίππους. Οι ίπποι προχωρούν δύο τετράγωνα προς οποιαδήποτε κατεύθυνση και στη συνέχεια κατά ένα τετράγωνο δεξιά ή αριστερά (λέγεται και κίνηση Γ). Μια ιδιαιτερότητα των ίππων είναι ότι μπορούν να υπερπηδούν άλλα κομμάτια.

Στον προηγούμενο κώδικα, πρέπει να γίνουν ελάχιστες αλλαγές. Πρώτον, απαιτείται προσθήκη γραμμών κώδικα για ενσωμάτωση των αρχείων εικόνας με τον λευκό και τον μαύρο ίππο. Στο σημείο που έχουν δηλωθεί οι πύργοι και οι αξιωματικοί, θα πρέπει να δηλωθούν και οι ίπποι, πάντα με προϋπόθεση τα σχετικά αρχεία εικόνας να βρίσκονται στον φάκελο της Python.

```
LI=pygame.image.load('LeykosIppos.png'); MI=pygame.image.load('MayrosIppos.png')
LA=pygame.image.load('LeykosAksiomatikos.png'); MA=pygame.image.load('MayrosAksiomatikos.png')
```

Δεύτερον απαιτείται προσθήκη κώδικα στο υποπρόγραμμα validMovesAndAttacks. Η προσθήκη αφορά βέβαια την περίπτωση που ο κωδικός kommati είναι 2 ή -2, δηλαδή κωδικός ίππου.

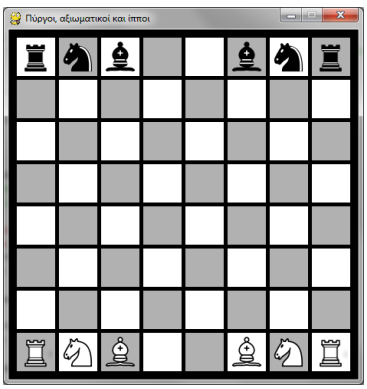
```
elif kommati==2 or kommati==-2: #Leykos h Mayros Ippos
    x1=xx-1; y1=yy-2
    if x1>=0 and y1>=0 and skaki[x1][y1]==0: moves.append([xx,yy,x1,y1])
    elif x1>=0 and y1>=0 and skaki[x1][y1]*kommati<0: attack.append([xx,yy,x1,y1])
    x1=xx-2; y1=yy-1
    if x1>=0 and y1>=0 and skaki[x1][y1]==0: moves.append([xx,yy,x1,y1])
    elif x1>=0 and y1>=0 and skaki[x1][y1]*kommati<0: attack.append([xx,yy,x1,y1])
    x1=xx-2; y1=yy+1
    if x1>=0 and y1<TY and skaki[x1][y1]==0: moves.append([xx,yy,x1,y1])
    elif x1>=0 and y1<TY and skaki[x1][y1]*kommati<0: attack.append([xx,yy,x1,y1])
    x1=xx-1; y1=yy+2
    if x1>=0 and y1<TY and skaki[x1][y1]==0: moves.append([xx,yy,x1,y1])
    elif x1>=0 and y1<TY and skaki[x1][y1]*kommati<0: attack.append([xx,yy,x1,y1])
    x1=xx+1; y1=yy+2
    if x1<TX and y1<TY and skaki[x1][y1]==0: moves.append([xx,yy,x1,y1])
    elif x1<TX and y1<TY and skaki[x1][y1]*kommati<0: attack.append([xx,yy,x1,y1])
    x1=xx+2; y1=yy+1
    if x1<TX and y1<TY and skaki[x1][y1]==0: moves.append([xx,yy,x1,y1])
    elif x1<TX and y1<TY and skaki[x1][y1]*kommati<0: attack.append([xx,yy,x1,y1])
    x1=xx+2; y1=yy-1
    if x1<TX and y1>=0 and skaki[x1][y1]==0: moves.append([xx,yy,x1,y1])
    elif x1<TX and y1>=0 and skaki[x1][y1]*kommati<0: attack.append([xx,yy,x1,y1])
    x1=xx+1; y1=yy-2
    if x1<TX and y1>=0 and skaki[x1][y1]==0: moves.append([xx,yy,x1,y1])
    elif x1<TX and y1>=0 and skaki[x1][y1]*kommati<0: attack.append([xx,yy,x1,y1])
elif kommati==3 or kommati==-3: #Leykos h Mayros Aksiwmatikos
```

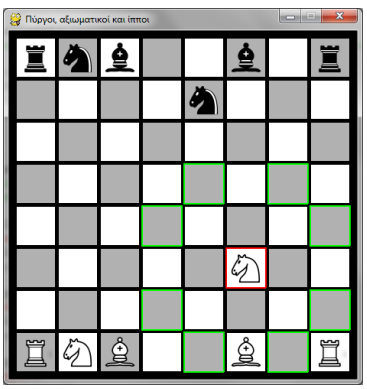
Παρατηρούμε ότι ελέγχονται οκτώ περιπτώσεις, όσες είναι οι μέγιστες δυνατές κινήσεις ενός ίππου. Οι μεταβλητές x1 και y1 δοκιμάζονται ως συντεταγμένες προορισμού. Αν η θέση (x1,y1) βρίσκεται εντός σκακιέρας και πρόκειται για κενό τετράγωνο (**`skaki[x1][y1]==0`**), τότε η κίνηση είναι επιτρεπτή και η τετράδα **`[xx,yy,x1,y1]`** προστίθεται στη λίστα moves. Αλλιώς αν η θέση (x1,y1) είναι τετράγωνο που καταλαμβάνεται από αντίπαλο κομμάτι, τότε η συνθήκη **`skaki[x1][y1]*kommati<0`** γίνεται αληθής όπως εξηγήθηκε και σε προηγούμενο κεφάλαιο, και άρα πρόκειται για επιτρεπτό χτύπημα και η τετράδα **`[xx,yy,x1,y1]`** προστίθεται στη λίστα attack.

Και τρίτον, απαιτείται κατάλληλη προσθήκη στο σημείο που δηλώνονται τα περιεχόμενα της δομής skaki. Προστέθηκαν οι ίπποι, αλλά επίσης όλα τα κομμάτια πλέον τοποθετήθηκαν στα σημεία που βρίσκονται στην αρχή μιας παρτίδας σκάκι.

```
skaki[0][7]=1; skaki[7][7]=1; skaki[0][0]=-1; skaki[7][0]=-1 #Leykoi kai mayroi pyrgoi
skaki[1][7]=2; skaki[6][7]=2; skaki[1][0]=-2; skaki[6][0]=-2 #Leykoi kai mayroi ippoi
skaki[2][7]=3; skaki[5][7]=3; skaki[2][0]=-3; skaki[5][0]=-3 #Leykoi kai mayroi aksiwmatikoi
```

Αποθηκεύουμε π.χ. ως 7Skaki12.py και τρέχουμε το πρόγραμμα. Πλέον το πρόγραμμά μας αρχίζει να παίρνει την τελική του μορφή και να αποκτά σχεδόν πλήρη λειτουργικότητα.





ΚΕΦΑΛΑΙΟ 13, Οι βασίλισσες στη σκακιέρα

Ήρθε η σειρά της βασίλισσας να ενταχθεί και αυτή στο πρόγραμμα. Ως γνωστόν μια βασίλισσα έχει τις δυνατότητες ενός πύργου και ενός αξιωματικού μαζί. Δηλαδή, κινείται οριζόντια, κάθετα και διαγώνια. Και πάλι, είναι απαραίτητο να υπάρχουν στον φάκελο της Python τα αρχεία εικόνας για τις δύο βασίλισσες. Ο τρόπος που θα εργαστούμε είναι γνωστός και σχεδόν ίδιος με τον τρόπο που εντάχθηκαν οι αξιωματικοί και οι ίπποι. Η πρώτη προσθήκη στον προηγούμενο κώδικα είναι η παρακάτω:

```
LV = pygame.image.load('LeykhVasilissa.png'); MV = pygame.image.load('MayrhVasilissa.png')
```

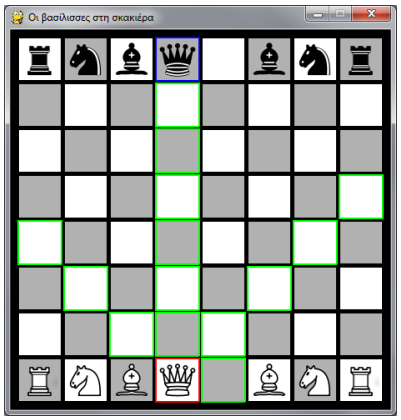
Η δεύτερη προσθήκη αφορά το υποπρόγραμμα validMovesAndAttacks.

```
    if x1>=0 and y1<8 and skaki[x1][y1]*kommati<0: attack.append([xx,yy,x1,y1])
elif kommati==4: #Leykh Vasilissa
    moves1, attack1 = validMovesAndAttacks(skaki,xx,yy,1) #Ypologismos kinhsewn
    moves3, attack3 = validMovesAndAttacks(skaki,xx,yy,3) #me texnikh ths anadromhs
    moves, attack = moves1+moves3, attack1+attack3
elif kommati==-4: #Mayrh Vasilissa
    moves_1, attack_1 = validMovesAndAttacks(skaki,xx,yy,-1)
    moves_3, attack_3 = validMovesAndAttacks(skaki,xx,yy,-3)
    moves, attack = moves_1+moves_3, attack_1+attack_3
return moves, attack
```

Και η τρίτη και τελευταία προσθήκη είναι η παρακάτω:

```
skaki[3][7]=4; skaki[3][0]=-4 #Leykh kai mayrh vasilissa
```

Η πρώτη και τρίτη προσθήκη είναι αυτονόητες. Θα εξηγηθεί παρακάτω η δεύτερη. Αποθηκεύουμε το τροποποιημένο πρόγραμμα π.χ. ως 7Skaki13.py και το τρέχουμε.



Στην validMovesAndAttacks προστέθηκαν δύο περιπτώσεις. Η μια αφορά λευκή βασίλισσα (**kommati==4**) και η άλλη αφορά μαύρη βασίλισσα (**kommati==-4**). Αν, λοιπόν, στο τετράγωνο του κλικ εντοπιστεί λευκή βασίλισσα τότε βλέπουμε να καλείται το υποπρόγραμμα μέσα από τον εαυτό του, δύο φορές! Παρατηρούμε ότι καλείται αρχικά για να εντοπιστούν οι κινήσεις της βασίλισσας ως σαν να έπαιζε ως πύργος (η τελευταία παράμετρος είναι 1 που είναι ο κωδικός του λευκού πύργου). Και στη συνέχεια καλείται για να εντοπιστούν οι κινήσεις της βασίλισσας ως σαν να έπαιζε ως αξιωματικός (η τελευταία παράμετρος είναι 3 που είναι ο κωδικός του λευκού αξιωματικού). Οι επιστρεφόμενες λίστες συγχωνεύονται και τελικά δημιουργούνται οι τελικές λίστες moves και attack με όλες

τις επιτρεπτές κινήσεις και χτυπήματα της λευκής βασίλισσας. Διευκρινίζεται ότι η έκφραση **moves1+moves3** όταν πρόκειται για λίστες επιστρέφει λίστα με τα στοιχεία των δύο λιστών. Εύκολα γίνεται αντιληπτή η περίπτωση μαύρης βασίλισσας που ακολουθεί (**kommati==-4**).

Η κλήση υποπρογράμματος μέσα από τον εαυτό του, είναι μια ισχυρή προγραμματιστική τεχνική και ονομάζεται αναδρομή. Όσο δυσνόητη και περίπλοκη ακούγεται στους αρχάριους προγραμματιστές, τόσο ισχυρή είναι. Θα της αναθέσουμε μεγαλύτερες και δυσκολότερες αποστολές όταν μιλήσουμε αργότερα για την τεχνητή νοημοσύνη.

ΚΕΦΑΛΑΙΟ 14, Βασιλιάδες και πιόνια

Δύο είδη κομματιών έχουν μείνει ακόμα να προγραμματιστούν: Ο βασιλιάς και το πιόνι. Ως γνωστόν, ο βασιλιάς μπορεί να μετακινηθεί σε όλα τα οκτώ γειτονικά τετράγωνά του, δηλαδή προχωράει κατά ένα προς κάθε κατεύθυνση, είτε κάθετα, είτε οριζόντια, είτε διαγώνια. Τα πιόνια είναι τα πιο αδύναμα κομμάτια στο σκάκι. Προχωρούν μόνο προς την κατεύθυνση του αντιπάλου κατά ένα τετράγωνο. Ειδικά τα πιόνια που βρίσκονται στην αρχή μπορούν να προχωρήσουν και κατά δύο τετράγωνα. Το πιόνια δεν χτυπούν όπως κινούνται (όπως συμβαίνει με όλα τα μεγάλα κομμάτια). Μπορούν να χτυπήσουν αντίπαλα κομμάτια μόνο αν αυτά βρεθούν ακριβώς διαγώνια μπροστά τους. Αγνοώντας προς το παρόν τους τρεις περίεργους κανόνες του σκακιού, το ροκέ, το αν πασάν και την προαγωγή, προχωράμε στον προγραμματισμό μας.

Δημιουργούμε αντίγραφο του προηγούμενου προγράμματος, το αποθηκεύουμε π.χ. ως 7Skaki14.py και κάνουμε τις παρακάτω προσθήκες στα κατάλληλα σημεία. Στο σημείο που δηλώνονται τα αρχεία εικόνας που θα ενσωματωθούν, συμπληρώνουμε τον παρακάτω κώδικα, ώστε να εμφανιστούν οι βασιλιάδες και τα πιόνια.

```
LR = pygame.image.load('LeykosVasilias.png'); MR = pygame.image.load('MayrosVasilias.png')
LS = pygame.image.load('LeykoPioni.png'); MS = pygame.image.load('MayroPioni.png');
```

Τα αρχικά των σταθερών LR και MR είναι αυθαίρετα επιλεγμένα από εμάς και σημαίνουν λευκός Ρήγας και μαύρος Ρήγας. Απαγορεύεται βέβαια να χρησιμοποιηθούν ίδια ονόματα σταθερών για διαφορετικά πράγματα (π.χ. LV και για λευκό βασιλιά και για λευκή βασίλισσα). Για τον ίδιο λόγο τα αρχικά LS και MS θα σημαίνουν λευκός και μαύρος στρατιώτης και θα αφορούν τα πιόνια. Η σταθερά LP, π.χ. για λευκό πιόνι, δεν επιτρέπεται, διότι έχει ήδη χρησιμοποιηθεί για λευκό πύργο.

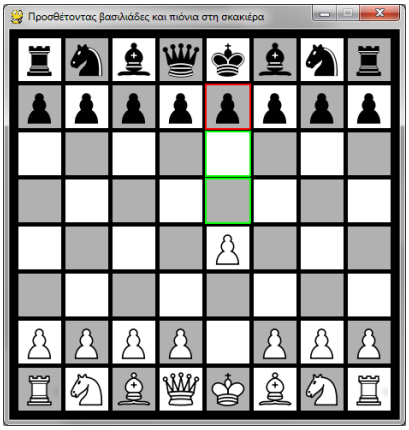
Στο σημείο που δηλώνονται τα περιεχόμενα της δομής skaki, πρέπει να δηλωθούν κωδικοί στα κατάλληλα τετράγωνα. Ο κωδικός 5 αφορά λευκό βασιλιά, ο κωδικός -5 αφορά μαύρο βασιλιά, ο κωδικός 6 αφορά λευκό πιόνι και ο κωδικός -6 αφορά μαύρο πιόνι. Δηλώνονται οι κωδικοί έτσι, ώστε τελικά να σχηματιστεί μια σκακιέρα με όλα τα κομμάτια στις αρχικές θέσεις μάχης.

```
skaki[3][7]=4; skaki[3][0]=-4; skaki[4][7]=5; skaki[4][0]=-5 #Leykh kai mayrh vasilissa, leykos kai mayros vasilias
#Leyka kai mayra pionia
skaki[0][6]=6; skaki[1][6]=6; skaki[2][6]=6; skaki[3][6]=6; skaki[4][6]=6; skaki[5][6]=6; skaki[6][6]=6; skaki[7][6]=6
skaki[0][1]=-6; skaki[1][1]=-6; skaki[2][1]=-6; skaki[3][1]=-6; skaki[4][1]=-6; skaki[5][1]=-6; skaki[6][1]=-6; skaki[7][1]=-6
```

Και βέβαια για να αποκτήσουν ζωή και κίνηση τα προηγούμενα, πρέπει να συμπληρωθεί το υποπρόγραμμα validMovesAndAttacks. Με τον παρακάτω κώδικα, συμπληρώνονται οι περιπτώσεις που το κομμάτι που κλικάρεται για παίξιμο είναι βασιλιάς ή πιόνι και το υποπρόγραμμα υπολογίζει και επιστρέφει τις επιτρεπτές κινήσεις και χτυπήματα.

```
elif kommati==5 or kommati==-5: #Leykos h Mayros Vasilias
    x1=xx; y1=yy-1
    if y1>=0 and skaki[x1][y1]==0: moves.append([xx,yy,x1,y1])
    elif y1>=0 and skaki[x1][y1]*kommati<0: attack.append([xx,yy,x1,y1])
    x1=xx; y1=yy+1
    if y1<TY and skaki[x1][y1]==0: moves.append([xx,yy,x1,y1])
    elif y1<TY and skaki[x1][y1]*kommati<0: attack.append([xx,yy,x1,y1])
    x1=xx-1; y1=yy
    if x1>=0 and skaki[x1][y1]==0: moves.append([xx,yy,x1,y1])
    elif x1>=0 and skaki[x1][y1]*kommati<0: attack.append([xx,yy,x1,y1])
    x1=xx+1; y1=yy
    if x1<TX and skaki[x1][y1]==0: moves.append([xx,yy,x1,y1])
    elif x1<TX and skaki[x1][y1]*kommati<0: attack.append([xx,yy,x1,y1])
    x1=xx-1; y1=yy-1
    if x1>=0 and y1>=0 and skaki[x1][y1]==0: moves.append([xx,yy,x1,y1])
    elif x1>=0 and y1>=0 and skaki[x1][y1]*kommati<0: attack.append([xx,yy,x1,y1])
    x1=xx-1; y1=yy+1
    if x1>=0 and y1<TY and skaki[x1][y1]==0: moves.append([xx,yy,x1,y1])
    elif x1>=0 and y1<TY and skaki[x1][y1]*kommati<0: attack.append([xx,yy,x1,y1])
    x1=xx+1; y1=yy+1
    if x1<TX and y1<TY and skaki[x1][y1]==0: moves.append([xx,yy,x1,y1])
    elif x1<TX and y1<TY and skaki[x1][y1]*kommati<0: attack.append([xx,yy,x1,y1])
    x1=xx+1; y1=yy-1
    if x1<TX and y1>=0 and skaki[x1][y1]==0: moves.append([xx,yy,x1,y1])
    elif x1<TX and y1>=0 and skaki[x1][y1]*kommati<0: attack.append([xx,yy,x1,y1])
elif kommati==6: #Leyko Pioni
    if yy-1>=0 and skaki[xx][yy-1]==0: moves.append([xx,yy,xx,yy-1])
    if yy==6 and skaki[xx][yy-1]==0 and skaki[xx][yy-2]==0: moves.append([xx,yy,xx,yy-2])
    if xx-1>=0 and yy-1>=0 and skaki[xx-1][yy-1]*kommati<0: attack.append([xx,yy,xx-1,yy-1])
    if xx+1<TX and yy-1>=0 and skaki[xx+1][yy-1]*kommati<0: attack.append([xx,yy,xx+1,yy-1])
elif kommati==-6: #Mayro Pioni
    if yy+1<TY and skaki[xx][yy+1]==0: moves.append([xx,yy,xx,yy+1])
    if yy==1 and skaki[xx][yy+1]==0 and skaki[xx][yy+2]==0: moves.append([xx,yy,xx,yy+2])
    if xx-1>=0 and yy+1<TY and skaki[xx-1][yy+1]*kommati<0: attack.append([xx,yy,xx-1,yy+1])
    if xx+1<TX and yy+1<TY and skaki[xx+1][yy+1]*kommati<0: attack.append([xx,yy,xx+1,yy+1])
return moves, attack
```

Αποθηκεύουμε και δοκιμάζουμε να τρέξουμε το πρόγραμμά μας.



Όταν η τιμή της μεταβλητής kommati είναι 5 ή -5, τότε πρόκειται για κομμάτι βασιλιά. Έτσι, πρέπει να ελεγχθούν ένα ένα όλα τα γύρω τετράγωνα. Π.χ. με τις εντολές **`x1=xx`** και **`y1=yy-1`** και τις δύο επόμενες, ελέγχεται το τετράγωνο πάνω από το βασιλιά. Αν είναι εντός ταμπλό (**`y1>=0`**) και αν είναι κενό (**`skaki[x1][y1]==0`**), τότε προστίθεται στην λίστα moves (**`moves.append([xx,yy,x1,y1])`**) . Αλλιώς αν καταλαμβάνεται από αντίπαλο κομμάτι (**`skaki[x1][y1]*kommati<0`**), τότε προστίθεται στη λίστα attack (**`attack.append([xx,yy,x1,y1])`**). Η διαδικασία επαναλαμβάνεται παρόμοια και για τα υπόλοιπα επτά τετράγωνα.

Όταν η τιμή της μεταβλητής kommati είναι 6, τότε πρόκειται για λευκό πιόνι. Ελέγχονται τέσσερις περιπτώσεις: Αν το τετράγωνο εμπρός του βρίσκεται εντός ταμπλό (**`yy-1>=0`**) και είναι κενό (**`skaki[xx][yy-1]==0`**), τότε αυτό προστίθεται στη λίστα moves. Αν το πιόνι βρίσκεται στην εκκίνηση (**`yy==6`**) και τα μπροστινά δύο τετράγωνα είναι ελεύθερα

(**skaki[xx][yy-1]==0 and skaki[xx][yy-2]==0**), τότε το πιόνι μπορεί να καταλάβει το τετράγωνο δύο μπροστά του, και άρα προστίθεται αυτή η κίνηση στη λίστα moves. Στη συνέχεια ελέγχονται και τα διαγώνια εμπρός τετράγωνα. Αν βρίσκονται εντός ταμπλό (**xx-1>=0 and yy-1>=0** ή **xx+1<TX and yy-1>=0**) και καταλαμβάνονται από αντίπαλο κομμάτι (**skaki[xx+1][yy-1]*kommati<0**), τότε τα χτυπήματα προς τα αντίστοιχα τετράγωνα προστίθενται στη λίστα attack. Παρόμοια προγραμματίζεται το μαύρο πιόνι (**kommati==-6**).

Με αυτές τις προσθήκες ολοκληρώνεται φαινομενικά το υποπρόγραμμα validMovesAndAttacks. Αν δοκιμάσουμε να τρέξουμε το πρόγραμμά μας, θα διαπιστώσουμε ότι πλέον όλα τα κομμάτια κινούνται ορθά. Όμως απέχουμε ακόμα από ένα πλήρες λειτουργικό σκάκι. Αυτό διότι π.χ. δεν έχουν προγραμματιστεί οι τρεις περίεργοι κανονισμοί του σκακιού (ροκέ, χτύπημα αν πασάν, προαγωγή πιονιών). Εξάλλου πρέπει να ληφθεί μέριμνα στις απειλές ΣΑΧ, ώστε να μην επιτρέπονται κινήσεις που να αφήνουν απροστάτευτο το βασιλιά. Όλα αυτά θα τα δούμε σε επόμενα κεφάλαια.

Ας ασχοληθούμε με μια μικρή βελτίωση του κώδικα του GameLoop. Αντί των εντολών

```
for mv in mov:
    pygame.draw.rect(epif,BLACK,(PER+mv[2]*(SZ+KENO)-2,PER+mv[3]*(SZ+KENO)-2,SZ+3,SZ+3),2)
for mv in att:
    pygame.draw.rect(epif,BLACK,(PER+mv[2]*(SZ+KENO)-2,PER+mv[3]*(SZ+KENO)-2,SZ+3,SZ+3),2)
```

… μπορούμε να γράψουμε ισοδύναμα μόνο μία εντολή:

```
for mv in mov+att:
    pygame.draw.rect(epif,BLACK,(PER+mv[2]*(SZ+KENO)-2,PER+mv[3]*(SZ+KENO)-2,SZ+3,SZ+3),2)
```

Είπαμε προηγουμένως ότι άθροισμα λιστών δημιουργεί ενιαία λίστα με τα στοιχεία των επιμέρους λιστών και αυτό συμβαίνει εδώ με την έκφραση **mov+att**. Η συγχώνευση είναι δυνατή επειδή η εντολή **pygame.draw.rect** που επαναλαμβάνεται είναι κοινή και στις δύο **for**. Μέσα στο GameLoop, η συγχώνευση θα γίνει και στο σημείο της ακύρωσης μιας κίνησης (**k==2 and klik1x==x and klik1y==y**) και στο σημείο έγκυρης κίνησης (**k==2 and [klik1x,klik1y,x,y] in mov+att**). Αποθηκεύουμε το τελικό μας πρόγραμμα.

ΚΕΦΑΛΑΙΟ 15, Βασιλιάς υπό απειλή

Για τη συνέχεια, κρίνεται απαραίτητο να μεγαλώσει η δομή skaki. Μέχρι στιγμής κρατάει μόνο στοιχεία για το τι κομμάτια βρίσκονται σε κάθε τετράγωνο. Και χρησιμοποιούνται, όπως ειπώθηκε, οι κωδικοί 1 έως 6 για τα λευκά και -1 έως -6 για τα μαύρα κομμάτια. Ας παρατηρήσει κανείς το παρακάτω σχήμα.

$$\begin{bmatrix} \begin{bmatrix}-1\\-6\\0\\0\\0\\0\\6\\1\end{bmatrix} & \begin{bmatrix}-2\\-6\\0\\0\\0\\0\\6\\2\end{bmatrix} & \begin{bmatrix}-3\\-6\\0\\0\\0\\0\\6\\3\end{bmatrix} & \begin{bmatrix}-4\\-6\\0\\0\\0\\0\\6\\4\end{bmatrix} & \begin{bmatrix}-5\\-6\\0\\0\\0\\0\\6\\5\end{bmatrix} & \begin{bmatrix}-3\\-6\\0\\0\\0\\0\\6\\3\end{bmatrix} & \begin{bmatrix}-2\\-6\\0\\0\\0\\0\\6\\2\end{bmatrix} & \begin{bmatrix}-1\\-6\\0\\0\\0\\0\\6\\1\end{bmatrix} & \begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} & \begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} \end{bmatrix}$$

Έχοντας δύο στήλες παραπάνω δηλαδή συνολικά δέκα, αυτή η νέα δομή θα αντικαταστήσει την παλιά μας. Στις πρώτες οκτώ στήλες θα τοποθετούνται όπως μέχρι τώρα οι κωδικοί των κομματιών. Στις δύο τελευταίες στήλες, θα καταχωρούνται δεδομένα που είναι χρήσιμα για τις κινήσεις ροκέ, αν πασάν που ακόμα δεν είδαμε και στοιχεία για τη θέση των βασιλιάδων που θα χρησιμεύσουν. Συμφωνούμε να χρησιμοποιείται η ένατη στήλη για στοιχεία θέσης των δύο βασιλιάδων. Έτσι, συμβατικά συμφωνούμε στα στοιχεία **skaki[8][0]** και **skaki[8][1]** να καταχωρούνται οι τρέχουσες συντεταγμένες x και y του λευκού βασιλιά ενώ στα στοιχεία **skaki[8][4]** και **skaki[8][5]** να καταχωρούνται οι συντεταγμένες x και y του μαύρου βασιλιά. Στην εκκίνηση του παιχνιδιού, προφανώς οι τιμές σε αυτές τις θέσεις θα είναι 4, 7, 4, 0. Επίσης, η αρχικοποίηση της δομής skaki, θα πρέπει να δημιουργήσει δύο στήλες επιπλέον. Δημιουργούμε αντίγραφο από το προηγούμενο πρόγραμμα, και το αποθηκεύουμε π.χ. ως 7Skaki15.py . Αρχικά στον κώδικα πρέπει να γίνουν τα εξής:

```
skaki = [] #Arxikopoihsh skakieras
for i in range(TX+2):
    column = []
    for j in range(TY):
        column.append(0)
    skaki.append(column)
```

Δηλαδή το **range** θα έχει όρισμα **TX+2** αντί για **TX**. Και επίσης πρέπει να προστεθεί ο παρακάτω κώδικας, γνωρίζοντας ότι για ταμπλό στιλ σκάκι, το **TX** έχει τιμή 8:

```
#Arxikes theseis leykoy kai mayroy vasilia
skaki[TX][0]=4; skaki[TX][1]=7; skaki[TX][4]=4; skaki[TX][5]=0
```

Για να μπορούμε να ελέγχουμε απειλές σε βασιλιά , θα χρειαστεί υποπρόγραμμα που θα ελέγχει γενικότερα αν κάποιο κομμάτι απειλείται. Προσθέτουμε λοιπόν το παρακάτω υποπρόγραμμα, συνιστάται στο σημείο που είναι και τα άλλα.

```
def attackMovesTo(skaki2,x,y,kommati):
    '''Epistrefei lista me tis kinhseis toy antipaloy,
    poy xtypoyn kommati sth thesh x,y.'''
    att = []
    skaki2[x][y]=kommati
    for i in range(1,7):
        m, a = validMovesAndAttacks(skaki2,x,y,i*int(kommati/abs(kommati)))
        for mv in a:
            if skaki2[mv[2]][mv[3]]==(-i)*int(kommati/abs(kommati)):
                tmp1, tmp2 = mv[2], mv[3]
                mv[2], mv[3] = mv[0], mv[1]
                mv[0], mv[1] = tmp1, tmp2
                att.append(mv)
    return att
```

Το υποπρόγραμμα attackMovesTo καλείται με ορίσματα τη δομή skaki2 (ουσιαστικά αντίγραφο της πραγματικής δομής skaki), τη θέση (δηλαδή τις συντεταγμένες x, y) και τον τύπο ενός κομματιού. Επιστρέφει στη λίστα att τις κινήσεις του αντιπάλου, οι οποίες μπορούν να χτυπήσουν το κομμάτι. Εννοείται, ότι αν η λίστα att επιστρέψει κενή, το κομμάτι δεν απειλείται. Ας εξετάσουμε πώς πετυχαίνεται αυτό.

Ελέγχεται απ' ευθείας το τετράγωνο x,y που απειλείται. Γίνονται έξι έλεγχοι (η range(1,7) θα δώσει στην μεταβλητή i διαδοχικά τις τιμές 1 έως 6) και καθένας έλεγχος αφορά και διαφορετικό τύπο κομματιού από πύργο έως πιόνι. Η έκφραση **int(kommati/abs(kommati))** ουσιαστικά επιστρέφει είτε 1, είτε -1, δηλαδή το πρόσημο του κομματιού kommati. Την πρώτη φορά ελέγχεται αν το κομμάτι μας παίζοντας ως

πύργος, θα είχε δυνατότητα να χτυπήσει αντίπαλο πύργο. Τη δεύτερη φορά ελέγχεται αν το κομμάτι μας παίζοντας ως ίππος, θα είχε δυνατότητα να χτυπήσει αντίπαλο ίππο. Και ούτω κάθε εξής. Έτσι, ουσιαστικά με ανάποδο τρόπο, ελέγχουμε αν το κομμάτι στη θέση x, y μπορεί να χτυπηθεί από πύργο, από ίππο κτλ. Όλες οι ζητούμενες κινήσεις, θα αποθηκευτούν τελικά στη λίστα att. Οι μεταβλητές tmp1 και tmp2 χρησιμεύουν προσωρινά ώστε οι συντεταγμένες **`mv[0]`**, **`mv[1]`** να αλλάξουν αμοιβαία θέση με τις **`mv[2]`**, **`mv[3]`** διότι οι ζητούμενες κινήσεις πρέπει να στοχεύουν στο κομμάτι και όχι να εκκινούν από αυτό. Δεν πειράζει αν δεν γίνει απόλυτα κατανοητός ο παραπάνω κώδικας, ο οποίος παρουσιάζει πιθανόν δυσκολία στην κατανόησή του, ειδικά από αρχάριους στον προγραμματισμό. Μπορεί να χρησιμοποιείται στα τυφλά και αρκεί να γνωρίζει κανείς τι δέχεται ως ορίσματα και τι επιστρέφει.

Μετά από αυτές τις προεργασίες, δηλαδή της επέκτασης της δομής skaki κατά δύο στήλες και της δημιουργίας του υποπρογράμματος attackMovesTo, ήμαστε έτοιμοι να προχωρήσουμε. Το επόμενο βήμα μας έχει να κάνει με το να μην επιτρέπει το πρόγραμμα να μένει εκτεθειμένος ο βασιλιάς σε απειλές. Ο βασιλιάς δεν τρώγεται στο σκάκι. Αν κάποιος παίκτης αγνοήσει απειλή στον βασιλιά του και τον αφήσει ακάλυπτο παίζοντας άσχετη κίνηση, τότε αυτή η κίνησή του θεωρείται αντικανονική. Αυτό που πρέπει λοιπόν να προγραμματιστεί, είναι να περιοριστεί ο σύνολο κινήσεων που επιστρέφει το validMovesAndAttacks, ώστε να μην επιτρέπονται κινήσεις που αφήνουν εκτεθειμένο το βασιλιά. Το σχέδιο είναι να δημιουργηθεί ένα υποπρόγραμμα superValidMovesAndAttacks, το οποίο θα επιστρέφει τις τελικές ορθές επιτρεπτές κινήσεις, ως ένα υποσύνολο εκείνων που επιστρέφει το υποπρόγραμμα validMovesAndAttacks.

Σε κάποιο σημείο με τα υποπρογράμματα, γράφουμε τον παρακάτω κώδικα του υποπρογράμματος superValidMovesAndAttacks:

```
def superValidMovesAndAttacks(skaki,xx,yy,kommati):
    '''To ypoprogramma ayto epistrefei ena yposynolo twn kinhsewn poy epistrefei to
    validMovesAndAttacks, kathws exoyn symperilhfthei elegxoi gia ektetheimeno vasilia.
    Gia ena kommati  poy vrisketai sto shmeio xx,yy epistrefontai 2 listes:
    H lista superMoves me orthes kinhseis kai h lista superAttack me ortha xtyphmata.'''
    moves, attack = validMovesAndAttacks(skaki,xx,yy,kommati)
    superMoves=[]; superAttack=[]
    for mv in moves+attack:
        #Dhmioyrgia antigrafoy ths domhs skaki, se eikonikh domh sk
        sk=copy.deepcopy(skaki)

        sk[mv[2]][mv[3]]=sk[mv[0]][mv[1]] #Paizetai h kinhsh mv eikonika sth domh sk
        sk[mv[0]][mv[1]]=0

        if kommati==5: #Epikairopoihsh twn eikonikwn thesewn twn vasiliadwn
            sk[TX][0]=mv[2]; sk[TX][1]=mv[3]
            saxMv=attackMovesTo(sk,sk[TX][0],sk[TX][1],5)
        elif kommati==-5:
            sk[TX][4]=mv[2]; sk[TX][5]=mv[3]
            saxMv=attackMovesTo(sk,sk[TX][4],sk[TX][5],-5)
        elif kommati in [1,2,3,4,6]:
            saxMv=attackMovesTo(sk,sk[TX][0],sk[TX][1],5)
        else:
            saxMv=attackMovesTo(sk,sk[TX][4],sk[TX][5],-5)

        if saxMv==[]: #An den menei ektetheimenos o vasilias, h mv einai orthh
            if mv in moves: superMoves.append(mv)
            else: superAttack.append(mv)
    return superMoves, superAttack
```

Στην κορυφή του προγράμματος, απαιτείται επίσης η προσθήκη του πακέτου copy.

```
import pygame, sys, time, copy; from pygame.locals import *
```

Ας εξηγηθεί το υποπρόγραμμα superValidMovesAndAttacks. Αρχικά καλείται το υποπρόγραμμα validMovesAndAttacks, με το οποίο θα επιστραφούν στις λίστες moves και

attack οι επιτρεπτές κινήσεις, δίχως πρόβλεψη για έκθεση του βασιλιά, δηλαδή πρόβλεψη για αντικανονική κίνηση. Στη συνέχεια δύο νέες λίστες αρχικοποιούνται ως κενές και είναι αυτές που τελικά θα επιστραφούν, οι superMoves και superAttack. Ακολουθεί εντολή for. Μέσα σε αυτήν την επαναληπτική εντολή θα δοκιμαστεί κάθε κίνηση για το αν είναι αντικανονική, δηλαδή για το αν μένει έκθετος ο βασιλιάς. Αρχικά με την εντολή **`sk=copy.deepcopy(skaki)`** δημιουργείται το ακριβές αντίγραφο sk της δομής skaki (εξαιτίας αυτής της εντολής απαιτήθηκε το πακέτο copy). Η δομή sk είναι μια εικονική δομή που θα παίξει το ρόλο δοκιμαστηρίου δίχως να τροποποιηθεί η πραγματική δομή skaki. Στην εικονική δομή παίζεται κάθε κίνηση που βρίσκεται στις λίστες moves και attack. Αυτό γίνεται στις εντολές **`sk[mv[2]][mv[3]]=sk[mv[0]][mv[1]]`** και **`sk[mv[0]][mv[1]]=0`**, όπου αλλάζουν οι κωδικοί κομματιών στα τετράγωνα εκκίνησης και προορισμού σύμφωνα με την κίνηση mv που εξετάζεται. Στη συνέχεια, αν η κίνηση αφορά λευκό βασιλιά, τροποποιούνται και τα σημεία της εικονικής δομής sk που κρατούν τα δεδομένα θέσης του λευκού βασιλιά και καλείται το υποπρόγραμμα attackMovesTo με το οποίο ελέγχεται αν μένει εκτεθειμένος ο βασιλιάς. Αν το attackMovesTo δεν επιστρέψει δεδομένα, τότε η λίστα saxMv θα είναι κενή. Αυτός ο έλεγχος για έκθεση του βασιλιά, γίνεται παρακάτω και στην περίπτωση κίνησης μαύρου βασιλιά αλλά και στις περιπτώσεις κίνησης άλλων κομματιών. Σε όλες τις περιπτώσεις, το υποπρόγραμμα attackMovesTo θα ελέγξει αξιόπιστα αν ο βασιλιάς θα μείνει εκτεθειμένος, επιστρέφοντας κίνηση χτυπήματος του βασιλιά στη λίστα saxMv. Αν η λίστα saxMv είναι κενή, μόνο τότε μπορούμε να θεωρήσουμε την κίνηση mv απολύτως έγκυρη. Και μόνο τότε από τις λίστες moves ή attack θα μπει και στις λίστες superMoves και superAttack. Αυτές οι λίστες κινήσεων είναι και οι τελικές που θα επιστραφούν.

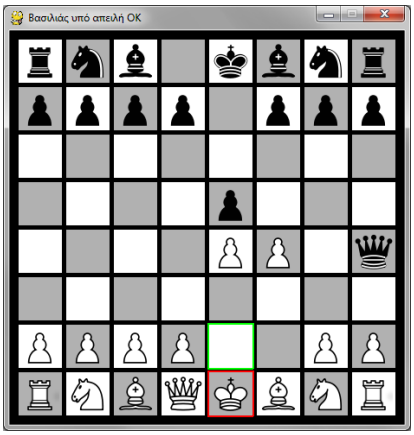
Ελάχιστες αλλαγές πρέπει να γίνουν και στο GameLoop, το οποίο παρόλ' αυτά παρουσιάζεται παρακάτω πλήρες:

```
while True: #GameLoop
    mouseClicked = False
    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit(); sys.exit()
        elif event.type == MOUSEBUTTONUP: #Klik pontikioy
            mousex, mousey = event.pos; mouseClicked = True
    if mouseClicked == True:
        x, y = xyFromMousexMousey(mousex, mousey)
        if  x != None: #Egkyro klik
            if k==1 and skaki[x][y]*turn>0: #An sthn katastash 1 ginei klik se katallhlo kommati
                klik1x=x; klik1y=y; klik1kommati=skaki[x][y]
                mov, att = superValidMovesAndAttacks(skaki,x,y,klik1kommati)
                if mov+att!=[]:
                    pygame.draw.rect(epif,RED,(PER+x*(SZ+KENO)-2,PER+y*(SZ+KENO)-2,SZ+3,SZ+3),2)
                    for mv in mov:
                        pygame.draw.rect(epif,GREEN,(PER+mv[2]*(SZ+KENO)-2,PER+mv[3]*(SZ+KENO)-2,SZ+3,SZ+3),2)
                    for mv in att:
                        pygame.draw.rect(epif,BLUE,(PER+mv[2]*(SZ+KENO)-2,PER+mv[3]*(SZ+KENO)-2,SZ+3,SZ+3),2)
                    k=2
            elif k==2 and klik1x==x and klik1y==y: #An ginei deytero klik sto idio kommati
                pygame.draw.rect(epif,BLACK,(PER+klik1x*(SZ+KENO)-2,PER+klik1y*(SZ+KENO)-2,SZ+3,SZ+3),2)
                for mv in mov+att:
                    pygame.draw.rect(epif,BLACK,(PER+mv[2]*(SZ+KENO)-2,PER+mv[3]*(SZ+KENO)-2,SZ+3,SZ+3),2)
                k=1
            elif k==2 and [klik1x,klik1y,x,y] in mov+att: #Egkyro deytero klik
                pygame.draw.rect(epif,BLACK,(PER+klik1x*(SZ+KENO)-2,PER+klik1y*(SZ+KENO)-2,SZ+3,SZ+3),2)
                for mv in mov+att:
                    pygame.draw.rect(epif,BLACK,(PER+mv[2]*(SZ+KENO)-2,PER+mv[3]*(SZ+KENO)-2,SZ+3,SZ+3),2)
                skaki[klik1x][klik1y]=0; skaki[x][y]=klik1kommati
                drawKommati(klik1x,klik1y)
                drawKommati(x,y)
                if klik1kommati==5:
                    skaki[TX][0]=x; skaki[TX][1]=y #Metakinhsh leykoy vasilia
                elif klik1kommati==-5:
                    skaki[TX][4]=x; skaki[TX][5]=y #Metakinhsh mayroy vasilia
                k=1
                turn = turn*-1
    pygame.display.update()
```

Στην περίπτωση διαχείρισης του πρώτου κλικ: Αντί του υποπρογράμματος validMovesAndAttacks, καλείται τώρα το superValidMovesAndAttacks. Στη συνέχεια προστίθεται μια εντολή if. Δηλαδή μόνο αν υπάρχει έγκυρη κίνηση σχεδιάζεται το τετράγωνο εκκίνησης με κόκκινο πλαίσιο, σχεδιάζονται τα επιτρεπτά τετράγωνα προορισμού με πράσινο πλαίσιο και η μεταβλητή κατάστασης k αλλάζει τιμή σε 2. Αν δεν υπάρχει έγκυρη κίνηση, όλα αυτά δεν γίνονται.

Στην περίπτωση διαχείρισης του δεύτερου κλικ: Προστίθεται μια εντολή if, η οποία σε περίπτωση κίνησης βασιλιά, αλλάζει στην δομή skaki τα δεδομένα θέσης του, τα οποία αποθηκεύονται όπως ειπώθηκε πριν στην ένατη στήλη.

Δοκιμάζουμε το πρόγραμμα σε καταστάσεις απειλών του βασιλιά. Για παράδειγμα, στο παρακάτω σχήμα επιτρέπονται μόνο δύο κινήσεις για το λευκό. Είτε μια κίνηση επάνω με τον βασιλιά, είτε μια κίνηση μπροστά με το πιόνι ώστε αυτό να παρεμβληθεί μεταξύ βασιλιά και βασίλισσας.



Το πρόγραμμα έχει αποκτήσει σχεδόν πλήρη λειτουργικότητα. Λείπουν όμως ακόμα το ροκέ, το χτύπημα αν πασάν, η προαγωγή πιονιών, εντοπισμός ΜΑΤ και άλλα.

ΚΕΦΑΛΑΙΟ 16, Ροκέ

Σύμφωνα με τους κανονισμούς του σκακιού, η κίνηση ροκέ εμπλέκει βασιλιά και έναν πύργο. Εφόσον ο μεταξύ τους χώρος είναι κενός από κομμάτια και εφόσον βασιλιάς και πύργος δεν έχουν κινηθεί από την αρχή της παρτίδας, τότε με το ροκέ δημιουργείται μια «χορευτική κίνηση τανγκό» όπου ο βασιλιάς θα μετακινηθεί κατά δύο τετράγωνα προς τον πύργο, ενώ ο πύργος θα μετακινηθεί στο τετράγωνο που υπερπηδήθηκε μόλις από τον βασιλιά. Μια επιπλέον τρίτη προϋπόθεση απαιτείται, ώστε να επιτρέπεται η κίνηση ροκέ: Τα τρία τετράγωνα που εμπλέκεται ο βασιλιάς (το τετράγωνο εκκίνησης, το τετράγωνο προορισμού και το τετράγωνο που υπερπηδήθηκε) δεν πρέπει να απειλούνται από τον αντίπαλο.

Δημιουργούμε αντίγραφο του προηγούμενου προγράμματος, αποθηκεύουμε π.χ. ως 7Skaki16.py και τροποποιούμε ως εξής:

Στο υποπρόγραμμα validMovesAndAttacks πρέπει να γίνουν προσθήκες στην περίπτωση κίνησης βασιλιά, ώστε να προστεθεί πιθανή δυνατότητα κίνησης ροκέ.

```
    if skaki[TX][int(-0.4*kommati+4)]==0: # if skaki[TX][2]==0 h skaki[TX][6]==0 , asaleytos vasilias
        if skaki[TX+1][int(-0.4*kommati+4)]==0 and skaki[xx+1][yy]==0 and skaki[xx+2][yy]==0: #asaleytos pyrgos,kena
            moves.append([xx,yy,xx+2,yy]) #Mikro roke
        if skaki[TX+1][int(-0.4*kommati+5)]==0 and skaki[xx-1][yy]==0 and skaki[xx-2][yy]==0 and skaki[xx-3][yy]==0:
            moves.append([xx,yy,xx-2,yy]) #Megalo roke
elif kommati==6: #Leyko Pioni
```

Συμφωνούμε, κίνηση ροκέ να σημαίνει κίνηση του βασιλιά κατά δύο τετράγωνα προς τον πύργο (**moves.append([xx,yy,xx+2,yy]**) ή **moves.append([xx,yy,xx-2,yy])**). Αυτό σημαίνει ότι σε δυνατότητα τέτοιας κίνησης ροκέ, κλικ σε βασιλιά θα πρασινίζει και το τετράγωνο δύο τετράγωνα μακριά του προς τον πύργο. Βέβαια υπάρχουν προαπαιτούμενα γι' αυτό. Το πρώτο είναι, μόνο αν ο βασιλιάς είναι ασάλευτος από την αρχή της παρτίδας, επιτρέπεται ροκέ. Γι' αυτό, στη δομή skaki, στην ένατη στήλη δεδομένων των βασιλιάδων συμφωνούμε τα εξής: Αν το τρίτο στοιχείο της στήλης είναι 0 (**skaki[TX][2]==0**), αυτό σημαίνει ότι δεν έχει μετακινηθεί ο λευκός βασιλιάς ποτέ στην παρτίδα. Αντίθετα, τιμή 1 σημαίνει ότι έχει μετακινηθεί. Παρόμοια, συμφωνούμε αν το έβδομο στοιχείο της ένατης στήλης είναι 0 (**skaki[TX][6]==0**), αυτό σημαίνει ότι δεν έχει μετακινηθεί ο μαύρος βασιλιάς ποτέ στην παρτίδα. Αντίθετα, τιμή 1 σημαίνει ότι έχει μετακινηθεί ο μαύρος βασιλιάς. Τα παραπάνω, μπορούν να γραφούν συνοπτικά με την έκφραση **skaki[TX][int(-0.4*kommati+4)]==0**. Πράγματι, αν δοκιμάσει κανείς να υπολογίσει το αποτέλεσμα της παραπάνω έκφρασης με τιμές στη μεταβλητή kommati είτε 5 είτε -5, δηλαδή τους κωδικούς των βασιλιάδων, τότε θα δει ότι οι αναφορές θα γίνονται στα κελιά **skaki[TX][2]** για λευκό βασιλιά και **skaki[TX][6]** για μαύρο.

Εννοείται, ότι αυτές οι θέσεις πρέπει να έχουν αρχικά τιμή 0 και αν μετακινηθεί βασιλιάς, να αποκτήσουν αμέσως τιμή 1 (το **skaki[TX][2]** για λευκό και το **skaki[TX][6]** για μαύρο βασιλιά). Αυτή η τιμή 1 δεν θα μπορεί να ξαναγίνει ποτέ πια 0. Αυτό θα προγραμματιστεί αργότερα στο GameLoop.

Εκτός όμως από το προαπαιτούμενο για ασάλευτο βασιλιά από την αρχή της παρτίδας, υπάρχει και το προαπαιτούμενο για ασάλευτο πύργο από την αρχή της παρτίδας. Έτσι, συμφωνούμε στη δομή skaki να έχουμε στην δέκατη στήλη δεδομένα γι' αυτό το θέμα. Στην τρίτη και έβδομη θέση αυτής της στήλης θα τοποθετούμε 0 για ασάλευτο πύργο του μικρού ροκέ, και στην τέταρτη και όγδοη θέση θα τοποθετούμε 0 για ασάλευτο πύργο του μεγάλου ροκέ. Σε περίπτωση κίνησης πύργου (ή επίσης αν φαγωθεί ο πύργος στην αρχική του θέση), η αντίστοιχη από αυτές τις θέσεις θα πάρει τιμή 1. Έτσι, ελέγχοντας αυτές τις θέσεις, μπορούμε να ελέγξουμε αν τηρείται το συγκεκριμένο προαπαιτούμενο. Αυτό γίνεται στη συνθήκη **skaki[TX+1][int(-0.4*kommati+4)]==0** η οποία ανάλογα αν το κομμάτι είναι λευκός ή μαύρος βασιλιάς, ελέγχει και τον αντίστοιχο πύργο του μικρού ροκέ.

Εκτός όμως από αυτά, πρέπει και τα τετράγωνα μεταξύ βασιλιά και πύργου να είναι κενά. Αυτό ελέγχεται με τις επόμενες συνθήκες **skaki[xx+1][yy]==0** και **skaki[xx+2][yy]==0** για μικρό ροκέ. Για μεγάλο ροκέ, ελέγχεται βέβαια ένα τετράγωνο παραπάνω (**skaki[xx-3][yy]==0**) αν είναι κενό, παράλληλα βέβαια με τη συνθήκη **skaki[TX+1][int(-0.4*kommati+5)]==0**. Η έκφραση **int(-0.4*kommati+5)** δίνει είτε 3, είτε 7 ανάλογα αν ο κωδικός kommati είναι 5 ή -5, ώστε να ελεγχθεί αν έχει κινηθεί ο πύργος του μεγάλου ροκέ. Παρατηρεί κανείς ότι με ενιαίο κώδικα ελέγχεται και ο λευκός και ο μαύρος βασιλιάς. Μπορεί κανείς να γράψει πιο απλουστευμένο και ευκολονόητο κώδικα, όμως οι γραμμές θα αυξηθούν.

Επίσης, στο υποπρόγραμμα superValidMovesAndAttacks πρέπει να γίνουν αλλαγές, ώστε να ελεγχθούν εικονικά τα ροκέ αν είναι επιτρεπτές κινήσεις. Οι αλλαγές αφορούν τις περιπτώσεις που παίζει βασιλιάς (kommati είναι 5 ή -5).

```
if kommati==5: #Epikairopoihsh twn eikonikwn thesewn twn vasiliadwn
    sk[TX][0]=mv[2]; sk[TX][1]=mv[3]; sk[TX][2]=1
    if mv[2]-mv[0]==2: #Mikro roke
        sk[7][7]=0; sk[5][7]=1 #Topothethsh kai leykoy pyrgoy
        saxMv=attackMovesTo(sk,4,7,5)+attackMovesTo(sk,5,7,5)+attackMovesTo(sk,6,7,5)
    elif mv[0]-mv[2]==2: #Megalo roke
        sk[0][7]=0; sk[3][7]=1 #Topothethsh kai leykoy pyrgoy
        saxMv=attackMovesTo(sk,4,7,5)+attackMovesTo(sk,3,7,5)+attackMovesTo(sk,2,7,5)
    else:
        saxMv=attackMovesTo(sk,sk[TX][0],sk[TX][1],5)
elif kommati==-5:
    sk[TX][4]=mv[2]; sk[TX][5]=mv[3]; sk[TX][6]=1
    if mv[2]-mv[0]==2: #Mikro roke
        sk[7][0]=0; sk[5][0]=-1 #Topothethsh kai mayroy pyrgoy
        saxMv=attackMovesTo(sk,4,0,-5)+attackMovesTo(sk,5,0,-5)+attackMovesTo(sk,6,0,-5)
    elif mv[0]-mv[2]==2: #Megalo roke
        sk[0][0]=0; sk[3][0]=-1 #Topothethsh kai mayroy pyrgoy
        saxMv=attackMovesTo(sk,4,0,-5)+attackMovesTo(sk,3,0,-5)+attackMovesTo(sk,2,0,-5)
    else:
        saxMv=attackMovesTo(sk,sk[TX][4],sk[TX][5],-5)
```

Αν παίξει ο λευκός βασιλιάς εικονικά την κίνηση mv, τώρα όχι μόνο ενημερώνεται η δομή sk για τα νέα δεδομένα θέσης (**sk[TX][4]=mv[2]** και **sk[TX][5]=mv[3]**), αλλά ενημερώνεται επίσης το σημείο **sk[TX][2]** με τιμή 1 που δηλώνει ότι ο λευκός βασιλιάς έχει κινηθεί. Σε περίπτωση μικρού ή μεγάλου ροκέ, η διαφορά των συντεταγμένων x του τετραγώνου εκκίνησης και του τετραγώνου προορισμού είναι 2. Έτσι, όταν αυτό συμβαίνει (π.χ. **mv[2]-mv[0]==2** για μικρό ροκέ), τοποθετείται εικονικά και ο πύργος στη νέα του θέση (**sk[7][7]=0 και sk[5][7]=1**).

Στη συνέχεια καλώντας τρεις φορές το υποπρόγραμμα attackMovesTo, ελέγχονται τα τρία τετράγωνα που θα περάσει ο λευκός βασιλιάς αν απειλούνται. Είπαμε ήδη ότι τότε απαγορεύεται κίνηση ροκέ. Αν δεν επιστραφούν κινήσεις, τότε η λίστα saxMv θα είναι κενή. Άρα, στο τέλος της superValidMovesAndAttacks, αυτή η κίνηση ροκέ θα καταταγεί στις έγκυρες κινήσεις superMoves. Παρόμοια συμβαίνουν για τα ροκέ του μαύρου.

Στο GameLoop θα γίνουν αλλαγές μόνο στο σημείο που παιχτεί έγκυρη κίνηση βασιλιά (**klik1kommati==5** ή **klik1kommati==-5**).

```
elif k==2 and [klik1x,klik1y,x,y] in mov+att: #Egkyro deytero klik
    pygame.draw.rect(epif,BLACK,(PER+klik1x*(SZ+KENO)-2,PER+klik1y*(SZ+KENO)-2,SZ+3,SZ+3),2)
    for mv in mov+att:
        pygame.draw.rect(epif,BLACK,(PER+mv[2]*(SZ+KENO)-2,PER+mv[3]*(SZ+KENO)-2,SZ+3,SZ+3),2)
    skaki[klik1x][klik1y]=0; skaki[x][y]=klik1kommati
    drawKommati(klik1x,klik1y); drawKommati(x,y)
    if klik1kommati==5:
        skaki[TX][0]=x; skaki[TX][1]=y; skaki[TX][2]=1 #Metakinhsh leykoy vasilia
        if x-klik1x==2: skaki[7][7]=0; skaki[5][7]=1; drawKommati(7,7); drawKommati(5,7) #Mikro roke
        elif klik1x-x==2: skaki[0][7]=0; skaki[3][7]=1; drawKommati(0,7); drawKommati(3,7) #Megalo roke
    elif klik1kommati==-5:
        skaki[TX][4]=x; skaki[TX][5]=y; skaki[TX][6]=1 #Metakinhsh mayroy vasilia
        if x-klik1x==2: skaki[7][0]=0; skaki[5][0]=-1; drawKommati(7,0); drawKommati(5,0) #Mikro roke
        elif klik1x-x==2: skaki[0][0]=0; skaki[3][0]=-1; drawKommati(0,0); drawKommati(3,0) #Megalo roke
    if (klik1x==7 and klik1y==7) or (x==7 and y==7): skaki[TX+1][2]=1 #Akyrwsh dynatothtas leykoy gia mikro roke
    if (klik1x==0 and klik1y==7) or (x==0 and y==7): skaki[TX+1][3]=1 #Akyrwsh dynatothtas leykoy gia megalo roke
    if (klik1x==7 and klik1y==0) or (x==7 and y==0): skaki[TX+1][6]=1 #Akyrwsh dynatothtas mayroy gia mikro roke
    if (klik1x==0 and klik1y==0) or (x==0 and y==0): skaki[TX+1][7]=1 #Akyrwsh dynatothtas mayroy gia megalo roke
    k=1
    turn = turn*-1
```

Εδώ, θα ενημερωθεί η πραγματική δομή skaki με τα δεδομένα που πρέπει. Γενικά αν κινηθεί βασιλιάς, θα ενημερωθούν τα δεδομένα θέσης που βρίσκονται στην ένατη στήλη **skaki[TX]**. Επίσης θα ενημερωθεί το σημείο **skaki[TX][2]** ή **skaki[TX][6]**. Τιμή 1 σημαίνει ο βασιλιάς κινήθηκε, και μας χρειάζεται αυτή η πληροφορία στους ελέγχους για έγκυρο ροκέ. Η έκφραση **if x-klik1x==2** εντός των κινήσεων του βασιλιά ελέγχει αν έγινε μικρό ροκέ, διότι πράγματι τότε τα τετράγωνα των κλικ εκκίνησης και προορισμού θα απέχουν 2. Αν έγινε μικρό ροκέ, τοποθετείται ο πύργος αυτόματα στη νέα του θέση δίπλα

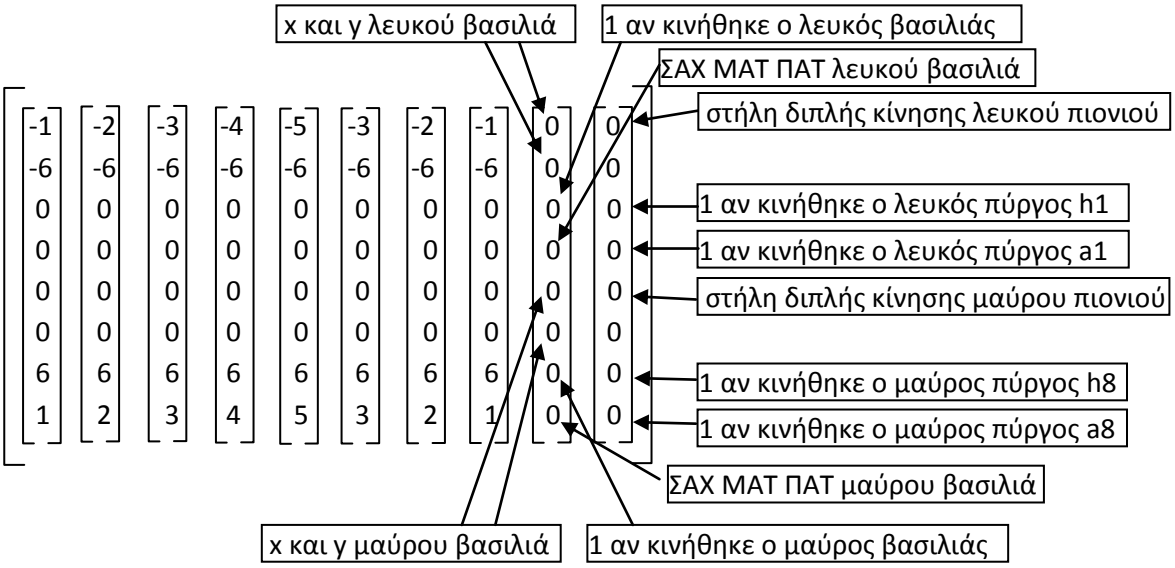
στο βασιλιά καταχωρώντας τους κατάλληλους κωδικούς (**skaki[7][7]=0** και **skaki[5][7]=1**) και σχεδιάζοντας τα αντίστοιχα τετράγωνα που εμπλέκουν τον πύργο με χρήση του υποπρογράμματος drawKommati. Παρόμοια συμβαίνουν στο μεγάλο ροκέ.

Οι τέσσερις εντολές **if** που προστέθηκαν κάτω από τις κινήσεις του βασιλιά, είναι για να ελέγχουν μήπως λόγω μετακίνησης κάποιου πύργου (η φαγώματός του στην αρχική του θέση) απωλέστηκε το δικαίωμα ροκέ από εκείνη την πλευρά. Έτσι, αν είτε οι μεταβλητές τετραγώνου εκκίνησης, είτε οι μεταβλητές τετραγώνου προορισμού αφορούν ένα από τα γωνιακά τετράγωνα, τότε σηματοδοτείται το αντίστοιχο σημείο της δομής skaki με τιμή 1. Αυτή η τιμή, θα αποτρέψει αργότερα πιθανή κίνηση ροκέ σε εκείνη τη γωνία της σκακιέρας.

Αποθηκεύουμε και τρέχουμε το πρόγραμμα. Παρατηρούμε ότι έχει προστεθεί και η λειτουργικότητα για κινήσεις ροκέ.

Το πράσινο τετράγωνο δίπλα στον πύργο, σημαίνει δυνατότητα κίνησης ροκέ. Υπενθυμίζεται ότι στο ροκέ πρέπει να γίνεται πρώτα η μετακίνηση του βασιλιά και να έπεται η μετακίνηση του πύργου. Έτσι και στο πρόγραμμά μας, δεν δίνεται δυνατότητα ροκέ αν κλικάρει κανείς πρώτα τον πύργο.

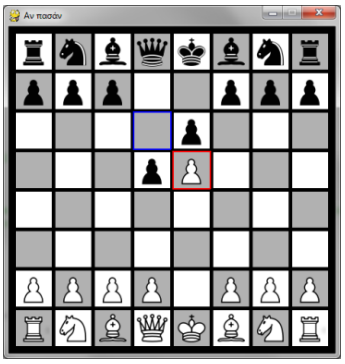
Ας συνοψίσουμε τη χρήση των δεδομένων της ένατης και δέκατης στήλης στη δομή skaki.



Παρατηρεί κανείς ότι έχουν χρησιμοποιηθεί τα σημεία **[TX][3]** και **[TX][7]**, δηλαδή το τέταρτο και το όγδοο της ένατης στήλης. Θα τα δούμε αργότερα, όπου θα χρησιμεύσουν για εντοπισμό ΣΑΧ, ΜΑΤ ή ΠΑΤ. Επίσης έχουν χρησιμοποιηθεί και τα σημεία **[TX+1][0]** και **[TX+1][4]**, δηλαδή το πρώτο και το πέμπτο της δέκατης στήλης. Θα τα δούμε στο επόμενο κεφάλαιο θα χρησιμεύσουν στη λεγόμενη κίνηση αν πασάν.

ΚΕΦΑΛΑΙΟ 17, Αν πασάν

Το αν πασάν, είναι μια ιδιαίτερη κίνηση χτυπήματος πιονιού από αντίπαλο πιόνι. Συγκεκριμένα, αν κάποιο πιόνι κάνοντας τη διπλή του κίνηση (βρισκόταν δηλαδή στην αρχή) βρεθεί δίπλα σε αντίπαλο πιόνι, τότε μπορεί να χτυπηθεί από αυτό. Το χτύπημα μπορεί να γίνει μόνο στην επόμενη κίνηση του αντιπάλου (και όχι μεταγενέστερα) και το χτύπημα γίνεται σαν να είχε προχωρήσει το πιόνι-θύμα μόνο μια κίνηση εμπρός.



Όπως είπαμε πριν, στη δέκατη στήλη της δομής skaki κρατάμε στοιχεία για το «ασάλευτο» των πύργων, χρήσιμα για τον έλεγχο εγκυρότητας ροκέ. Παρόμοια, χρειαζόμαστε στοιχεία για το αν πασάν. Μπορεί κανείς απλά να κρατάει σε ένα σημείο της δέκατης στήλης δεδομένα που να φανερώνουν αν επιτρέπεται κίνηση αν πασάν. Μια σύμβαση θα μπορούσε να είναι ότι αν κινηθεί πιόνι με διπλή κίνηση, τότε το σημείο ελέγχου αποκτά τον αριθμό της στήλης του πιονιού που κινήθηκε διπλά. Αν π.χ. κινηθεί το λευκό πιόνι του δεξιού πύργου κατά δύο τετράγωνα, το σημείο **`skaki[TX+1][0]`** θα αποκτήσει τιμή 8.

Έτσι, το validMovesAndAttacks, μπορεί να ελέγχει αυτό το σημείο. Αν κάποιο μαύρο πιόνι βρίσκεται στην τέταρτη γραμμή (**`y=4`**), αν το σημείο ελέγχου είναι διάφορο του 0 και μάλιστα είναι αριθμός που συνορεύει με τη στήλη του μαύρου πιονιού, τότε μπορεί να γίνει χτύπημα αν πασάν και το validMovesAndAttacks θα πρέπει να το συμπεριλάβει. Αυτό ακριβώς γίνεται στο κώδικα που ακολουθεί. Στο σημείο της κίνησης του λευκού πιονιού προστέθηκε μια ακόμα γραμμή. Και στο σημείο της κίνησης του μαύρου πιονιού προστέθηκε μια παρόμοια.

```
elif kommati==6: #Leyko Pioni
    if yy-1>=0 and skaki[xx][yy-1]==0: moves.append([xx,yy,xx,yy-1])
    if yy==6 and skaki[xx][yy-1]==0 and skaki[xx][yy-2]==0: moves.append([xx,yy,xx,yy-2])
    if xx-1>=0 and yy-1>=0 and skaki[xx-1][yy-1]*kommati<0: attack.append([xx,yy,xx-1,yy-1])
    if xx+1<TX and yy-1>=0 and skaki[xx+1][yy-1]*kommati<0: attack.append([xx,yy,xx+1,yy-1])
    if skaki[TX+1][4]!=0 and yy==3 and abs(skaki[TX+1][4]-1-xx)==1: attack.append([xx,yy,skaki[TX+1][4]-1,yy-1])
elif kommati==-6: #Mayro Pioni
    if yy+1<TY and skaki[xx][yy+1]==0: moves.append([xx,yy,xx,yy+1])
    if yy==1 and skaki[xx][yy+1]==0 and skaki[xx][yy+2]==0: moves.append([xx,yy,xx,yy+2])
    if xx-1>=0 and yy+1<TY and skaki[xx-1][yy+1]*kommati<0: attack.append([xx,yy,xx-1,yy+1])
    if xx+1<TX and yy+1<TY and skaki[xx+1][yy+1]*kommati<0: attack.append([xx,yy,xx+1,yy+1])
    if skaki[TX+1][0]!=0 and yy==4 and abs(skaki[TX+1][0]-1-xx)==1: attack.append([xx,yy,skaki[TX+1][0]-1,yy+1])
return moves, attack
```

Με την **`if skaki[TX+1][4]!=0`** ελέγχεται αν μπορεί να χτυπηθεί αν πασάν ο μαύρος. Αν ναι, τότε με τις άλλες δύο συνθήκες ελέγχεται αν το λευκό πιόνι βρίσκεται δίπλα στο μαύρο πιόνι. **`yy==3`** θα σήμαινε ότι το λευκό πιόνι βρίσκεται στη σωστή γραμμή, διότι μόνο σε μια γραμμή (τέταρτη από πάνω, η οποία δηλώνεται ως 3) γίνονται χτυπήματα του λευκού. Η συνθήκη **`abs(skaki[TX+1][4]-1-xx)==1`** ελέγχει ότι το λευκό πιόνι βρίσκεται και σε στήλη γειτονική. Μόνο τότε προστίθεται η κίνηση στη λίστα attack. Παρόμοια συμβαίνουν αν μπορεί να χτυπηθεί αν πασάν ο λευκός.

Επίσης, στο υποπρόγραμμα superValidMovesAndAttacks πρέπει να γίνουν αλλαγές:

```
elif kommati==6:
    if mv[1]-mv[3]==2: #An diplh kinhsh leykoy pionioy
        sk[TX+1][0]=mv[2]+1 #Markaretai to shmeio elegxoy an pasan toy leykoy
    elif sk[TX+1][4]!=0 and mv[3]==2 and mv[2]==sk[TX+1][4]-1: #Xtyphma an pasan
        sk[sk[TX+1][4]-1][3]=0 #Feygei to xtyphmeno
    saxMv=attackMovesTo(sk,sk[TX][0],sk[TX][1],5)
elif kommati==-6:
    if mv[3]-mv[1]==2: #An diplh kinhsh mayroy pionioy
        sk[TX+1][4]=mv[2]+1 #Markaretai to shmeio elegxoy an pasan toy mayroy
    elif sk[TX+1][0]!=0 and mv[3]==5 and mv[2]==sk[TX+1][0]-1: #Xtyphma an pasan
        sk[sk[TX+1][0]-1][4]=0 #Feygei to xtyphmeno
    saxMv=attackMovesTo(sk,sk[TX][4],sk[TX][5],-5)
elif kommati in [1,2,3,4]:
    saxMv=attackMovesTo(sk,sk[TX][0],sk[TX][1],5)
else:
    saxMv=attackMovesTo(sk,sk[TX][4],sk[TX][5],-5)
```

Σε περίπτωση αν πασάν, είναι πιθανό να βρεθεί ο βασιλιάς εκτεθειμένος. Άρα, πρέπει στην εικονική δομή sk όχι μόνο να τοποθετηθεί το πιόνι που τρώει στη νέα του θέση αλλά και να βγει το φαγωμένο πιόνι. Αυτό ακριβώς γίνεται παραπάνω. Έπειτα από αυτά, η κλήση του υποπρογράμματος attackMovesTo θα επιστρέψει αξιόπιστα δεδομένα.

Επίσης, στη συνέχεια ελέγχονται όλα τα υπόλοιπα κομμάτια πλην βασιλιάδων και πιονιών (**`elif kommati in [1,2,3,4]`**), δηλαδή αφαιρέθηκε και το 6.

Αν ο παίκτης επιλέξει τελικά να παίξει κίνηση αν πασάν, τότε πρέπει να γίνουν διάφορες ενέργειες. Προσθέτουμε τον παρακάτω κώδικα στο GameLoop στο σημείο αναμονής του δεύτερου κλικ όταν παιχτεί κίνηση πιονιού, ο οποίος κώδικας μοιάζει βέβαια πολύ με αυτόν που γράφτηκε στο υποπρόγραμμα superValidMovesAndAttacks.

```
elif klik1kommati==6: #Metakinhsh leykoy pionioy
    if klik1y-y==2: #An diplh kinhsh leykoy pionioy
        skaki[TX+1][0]=x+1 #Markaretai to shmeio elegxoy an pasan toy leykoy
    elif skaki[TX+1][4]!=0 and y==2 and x==skaki[TX+1][4]-1: #An xtyphma an pasan toy leykoy
        skaki[skaki[TX+1][4]-1][3]=0; drawKommati(skaki[TX+1][4]-1,3) #Feygei to xtyphmeno
elif klik1kommati==-6: #Metakinhsh mayroy pionioy
    if y-klik1y==2: #An diplh kinhsh mayroy pionioy
        skaki[TX+1][4]=x+1 #Markaretai to shmeio elegxoy an pasan toy mayroy
    elif skaki[TX+1][0]!=0 and y==5 and x==skaki[TX+1][0]-1: #An xtyphma an pasan toy mayroy
        skaki[skaki[TX+1][0]-1][4]=0; drawKommati(skaki[TX+1][0]-1,4) #Feygei to xtyphmeno
if not(klik1kommati==6 and klik1y-y==2): skaki[TX+1][0]=0 #Epanafora shmeioy elegxoy an pasan toy leykoy
if not(klik1kommati==-6 and y-klik1y==2): skaki[TX+1][4]=0 #Epanafora shmeioy elegxoy an pasan toy mayroy
```

Εύκολα ελέγχεται αν παίχτηκε διπλή κίνηση πιονιού, καθώς τότε οι διαφορές των συντεταγμένων γραμμής των τετραγώνων εκκίνησης και προορισμού θα είναι δύο (π.χ. για λευκό πιόνι **`klik1y-y==2`**). Σε τέτοια περίπτωση, τσεκάρεται το σημείο ελέγχου στη δομή skaki, το οποίο για τον λευκό είναι στην δέκατη στήλη στη θέση 4. Αν παίχτηκε πιόνι της στήλης α (x είναι 0), το σημείο τσεκάρεται με 1, αν παίχτηκε πιόνι της στήλης β (x είναι 1), τότε το σημείο τσεκάρεται με 2, … , αν παίχτηκε πιόνι της στήλης θ (x είναι 7), το σημείο τσεκάρεται με 8. Με αυτόν τον τρόπο, στην επόμενη κίνηση του μαύρου, αν αυτός έχει πιόνι στην κατάλληλη γραμμή και σε στήλη γειτονική από αυτήν που δηλώνεται σε αυτό το σημείο ελέγχου, η κίνηση αν πασάν θα προστεθεί στις επιτρεπτές.

Αν αντί για διπλή κίνηση, παιχτεί χτύπημα αν πασάν του λευκού, αυτό μπορεί να εντοπιστεί από το σημείο ελέγχου αν πασάν του μαύρου που θα είναι διάφορο του μηδέν (**`skaki[TX+1][4]!=0`**), και από τα x και y, τα οποία το μεν y θα είναι στη τρίτη γραμμή από πάνω (y==2), το δε x θα ταυτίζεται με το περιεχόμενο του σημείου ελέγχου πλην ένα (**`x==skaki[TX+1][4]-1`**). Αν και οι τρεις αυτές συνθήκες ισχύουν, τότε πρέπει να ενημερωθεί η δομή skaki για το μαύρο πιόνι που τρώγεται και να σχεδιαστεί το κατάλληλο τετράγωνο. Η κίνηση του λευκού πιονιού, θα ενημερωθεί στη δομή skaki παρακάτω όπως όλες οι άλλες κινήσεις.

Σε κάθε περίπτωση, αν δεν γίνει διπλή κίνηση πιονιού, τότε τα σημεία ελέγχου αν πασάν πρέπει να μηδενιστούν. Αυτό γίνεται στις δύο τελευταίες εντολές **if**. Αποθηκεύουμε τις παραπάνω αλλαγές π.χ. ως 7Skaki17.py και τις δοκιμάζουμε.

ΚΕΦΑΛΑΙΟ 18, Προαγωγή πιονιών

Οι κανονισμοί του σκακιού απαιτούν όταν ένα πιόνι φτάνει στην τελευταία γραμμή του αντιπάλου, αυτό να προάγεται σε βασίλισσα, πύργο, ίππο ή αξιωματικό. Για να προγραμματιστεί η προαγωγή, καλό είναι να μεγαλώσει κατ' αρχήν ο χώρος του παιχνιδιού μας, ώστε στον επιπλέον χώρο που δημιουργείται να μπορούν να σχεδιάζονται οι επιλογές. Η εντολή **epif=pygame.display.set_mode** ξαναγράφεται λοιπόν ως εξής:

```
epif=pygame.display.set_mode((2*PER+TX*SZ+(TX-1)*KENO,2*PER+TX*SZ+(TY-1)*KENO+(SZ+KENO)*2))
```

Καθώς μεγάλωσε το ύψος κατά **(SZ+KENO)*2** δημιουργήθηκε χώρος στο κάτω μέρος. Σε αυτόν θα σχεδιάζονται οι τέσσερις επιλογές κομματιών που έχει ο παίκτης που προάγει κομμάτι, με το παρακάτω υποπρόγραμμα drawKommatiaProagwghs.

```
def drawKommatiaProagwghs(turn):
    '''Sxediazontai ta 4 kommatia proagwghs sto katw meros toy tamplo'''
    for i in range(4):
        pygame.draw.rect(epif,WHITE,(PER+(i+2)*(SZ+KENO),PER+(TY+1)*(SZ+KENO),SZ,SZ),0)
    if turn==1:
        epif.blit(LP, (PER + 2*(SZ+KENO),PER + (TY+1)*(SZ+KENO),SZ,SZ))
        epif.blit(LI, (PER + 3*(SZ+KENO),PER + (TY+1)*(SZ+KENO),SZ,SZ))
        epif.blit(LA, (PER + 4*(SZ+KENO),PER + (TY+1)*(SZ+KENO),SZ,SZ))
        epif.blit(LV, (PER + 5*(SZ+KENO),PER + (TY+1)*(SZ+KENO),SZ,SZ))
    elif turn==-1:
        epif.blit(MP, (PER + 2*(SZ+KENO),PER + (TY+1)*(SZ+KENO),SZ,SZ))
        epif.blit(MI, (PER + 3*(SZ+KENO),PER + (TY+1)*(SZ+KENO),SZ,SZ))
        epif.blit(MA, (PER + 4*(SZ+KENO),PER + (TY+1)*(SZ+KENO),SZ,SZ))
        epif.blit(MV, (PER + 5*(SZ+KENO),PER + (TY+1)*(SZ+KENO),SZ,SZ))

def drawKommatiaProagwghsEnd():
    '''Epanaferetai se mayro h perioxh me ta 4 kommatia proagwghs'''
    for i in range(4):
        pygame.draw.rect(epif,BLACK,(PER+(i+2)*(SZ+KENO),PER+(TY+1)*(SZ+KENO),SZ,SZ),0)
```

Με το δεύτερο υποπρόγραμμα drawKommatiaProagwghsEnd θα επαναφέρεται σε μαύρο η περιοχή που σχεδιάστηκαν προσωρινά οι τέσσερις επιλογές κομματιών. Μένει να προγραμματιστεί μια πραγματική κίνηση προαγωγής στο GameLoop:

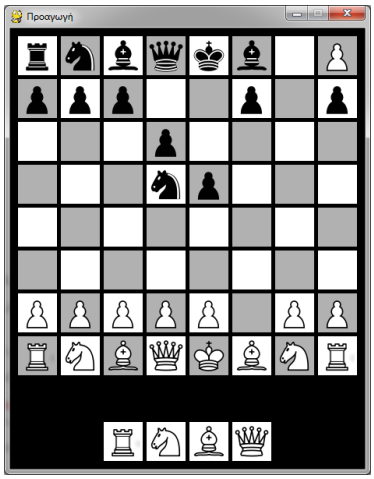
```
if klik1kommati==6: #Metakinhsh leykoy pionioy
    if klik1y-y==2: #An diplh kinhsh leykoy pionioy
        skaki[TX+1][0]=x+1 #Markaretai to shmeio elegxoy an pasan toy leykoy
    elif skaki[TX+1][4]!=0 and y==2 and x==skaki[TX+1][4]-1: #An xtyphma an pasan toy leykoy
        skaki[skaki[TX+1][4]-1][3]=0; drawKommati(skaki[TX+1][4]-1,3) #Feygei to xtyphmeno
    elif y==0: #Proagwgh leykoy pionioy
        drawKommatiaProagwghs(turn)
        pygame.display.update()
        while True:
            mouseClicked = False
            for event in pygame.event.get():
                if event.type == QUIT:
                    pygame.quit(); sys.exit()
                elif event.type == MOUSEBUTTONDOWN:
                    mousex, mousey = event.pos
                    mouseClicked = True
            if mouseClicked == True:
                iRect1 = pygame.Rect(PER + 2*(SZ+KENO),PER + (TY+1)*(SZ+KENO),SZ,SZ)
                if iRect1.collidepoint(mousex, mousey):
                    skaki[x][y]=1; drawKommati(x,y);
                iRect2 = pygame.Rect(PER + 3*(SZ+KENO),PER + (TY+1)*(SZ+KENO),SZ,SZ)
                if iRect2.collidepoint(mousex, mousey):
                    skaki[x][y]=2; drawKommati(x,y);
                iRect3 = pygame.Rect(PER + 4*(SZ+KENO),PER + (TY+1)*(SZ+KENO),SZ,SZ)
                if iRect3.collidepoint(mousex, mousey):
                    skaki[x][y]=3; drawKommati(x,y);
                iRect4 = pygame.Rect(PER + 5*(SZ+KENO),PER + (TY+1)*(SZ+KENO),SZ,SZ)
                if iRect4.collidepoint(mousex, mousey):
                    skaki[x][y]=4; drawKommati(x,y);
            if skaki[x][y]!=6: drawKommatiaProagwghsEnd(); break
elif klik1kommati==-6: #Metakinhsh mayroy pionioy
```

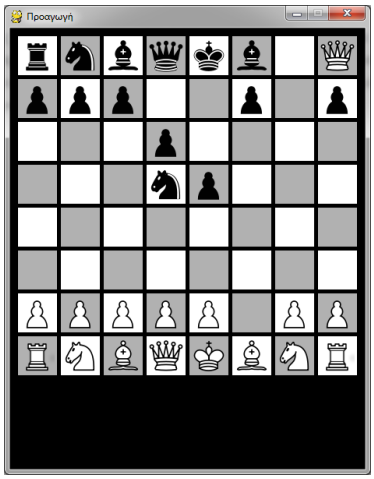
Όπως βλέπουμε παραπάνω, πρέπει να προστεθεί κώδικας όταν παίξει λευκό πιόνι και συγκεκριμένα όταν αυτό βρεθεί στην γραμμή 0 (γραμμή προαγωγής λευκού). Σε τέτοια περίπτωση σχεδιάζονται με το υποπρόγραμμα drawKommatiaProagwghs οι τέσσερις επιλογές και εκτελείται η εντολή **`pygame.display.update()`**, ώστε να έχουμε άμεσο αποτέλεσμα στην οθόνη. Στη συνέχεια, μπαίνει το πρόγραμμα σε μια επανάληψη while, από την οποία θα βγει μόνο όταν ο χρήστης κλικάρει σε μια από τις τέσσερις επιλογές κομματιών. Όταν γίνει κλικ στο ποντίκι, η μεταβλητή mouseClicked θα γίνει αληθής. Αν τώρα το κλικ έγινε πάνω σε μια από τις τέσσερις επιλογές, τότε ενημερώνεται η δομή skaki για το νέο κομμάτι, το οποίο και σχεδιάζεται. Η εντολή **`collidepoint`** είναι μια μέθοδος που εφαρμόζεται σε νοητά τετράγωνα. Παίρνει όρισμα ένα σημείο, στην περίπτωσή μας το σημείο του κλικ. Επιστρέφει αληθής (**`True`**) αν το σημείο βρίσκεται εντός του νοητού τετραγώνου. Έτσι, ένα ένα ελέγχονται όλα τα τετράγωνα, στα οποία σχεδιάστηκαν προηγουμένως οι τέσσερις επιλογές αν έγινε κλικ μέσα τους. Π.χ. η **`iRect1.collidepoint(mousex, mousey)`** επιστρέφει αληθές αν το σημείο με συντεταγμένες mousex και mousey βρίσκεται εντός του τετραγώνου iRect1, στο οποίο σχεδιάστηκε στην περίπτωσή μας ο πύργος. Στο τέλος της επανάληψης **`while`**, η εντολή **`if skaki[x][y]!=6`** ελέγχει αν άλλαξε ο κωδικός πιονιού σε κάτι άλλο. Τότε με το **`drawKommatiaProagwghsEnd()`** σχεδιάζεται μαύρη η περιοχή των επιλογών και εκτελείται η εντολή **`break`**. Αυτή η εντολή, όταν εκτελεστεί, έχει ως αποτέλεσμα την ακαριαία διακοπή της εντολής επανάληψης εντός της οποίας βρίσκεται το **`break`**. Άρα διακόπτεται η **`while`** και το παιχνίδι επανέρχεται στο γνωστό εξωτερικό GameLoop.

Το παραπάνω τμήμα, εννοείται ότι πρέπει να γραφεί και για την περίπτωση προαγωγής μαύρου πιονιού:

```
elif klik1kommati==-6: #Metakinhsh mayroy pionioy
    if y-klik1y==2: #An diplh kinhsh mayroy pionioy
        skaki[TX+1][4]=x+1 #Markaretai to shmeio elegxoy an pasan toy mayroy
    elif skaki[TX+1][0]!=0 and y==5 and x==skaki[TX+1][0]-1: #An xtyphma an pasan toy mayroy
        skaki[skaki[TX+1][0]-1][4]=0; drawKommati(skaki[TX+1][0]-1,4) #Feygei to xtyphmeno
    elif y==7: #Proagwgh mayroy pionioy
        drawKommatiaProagwghs(turn)
        pygame.display.update()
        while True:
            mouseClicked = False
            for event in pygame.event.get():
                if event.type == QUIT:
                    pygame.quit(); sys.exit()
                elif event.type == MOUSEBUTTONDOWN:
                    mousex, mousey = event.pos
                    mouseClicked = True
            if mouseClicked == True:
                iRect1 = pygame.Rect(PER + 2*(SZ+KENO),PER + (TY+1)*(SZ+KENO),SZ,SZ)
                if iRect1.collidepoint(mousex, mousey):
                    skaki[x][y]=-1; drawKommati(x,y);
                iRect2 = pygame.Rect(PER + 3*(SZ+KENO),PER + (TY+1)*(SZ+KENO),SZ,SZ)
                if iRect2.collidepoint(mousex, mousey):
                    skaki[x][y]=-2; drawKommati(x,y);
                iRect3 = pygame.Rect(PER + 4*(SZ+KENO),PER + (TY+1)*(SZ+KENO),SZ,SZ)
                if iRect3.collidepoint(mousex, mousey):
                    skaki[x][y]=-3; drawKommati(x,y);
                iRect4 = pygame.Rect(PER + 5*(SZ+KENO),PER + (TY+1)*(SZ+KENO),SZ,SZ)
                if iRect4.collidepoint(mousex, mousey):
                    skaki[x][y]=-4; drawKommati(x,y);
            if skaki[x][y]!=-6: drawKommatiaProagwghsEnd(); break
```

Αποθηκεύουμε τις αλλαγές π.χ. ως 7Skaki18.py και τις δοκιμάζουμε σε καταστάσεις προαγωγής. Όλα πρέπει να λειτουργούν επιτυχώς, δηλαδή σε περίπτωση που ένα πιόνι φτάνει στην τελευταία γραμμή, σχεδιάζονται κάτω από τη σκακιέρα οι τέσσερις δυνατότητες προαγωγής του, σε μια από τις οποίες απαιτείται κλικ.





Όλα λειτουργούν κανονικά, παρόλ' αυτά θα τροποποιήσουμε τον κώδικα σε δύο σημεία. Στο σημείο του υποπρογράμματος validMovesAndAttacks όπου εντοπίζονται κινήσεις πιονιών, ο κώδικας αλλάζει ως εξής:

```
elif kommati==6: #Leyko Pioni
    if yy-1>=0 and skaki[xx][yy-1]==0:
        if yy-1!=0: moves.append([xx,yy,xx,yy-1])
        else: moves.append([xx,yy,xx,yy-1,1]); moves.append([xx,yy,xx,yy-1,2]); moves.append([xx,yy,xx,yy-1,3]); moves.append([xx,yy,xx,yy-1,4])
    if yy==6 and skaki[xx][yy-1]==0 and skaki[xx][yy-2]==0: moves.append([xx,yy,xx,yy-2])
    if xx-1>=0 and yy-1>=0 and skaki[xx-1][yy-1]*kommati<0:
        if yy-1!=0: attack.append([xx,yy,xx-1,yy-1])
        else: attack.append([xx,yy,xx-1,yy-1,1]); attack.append([xx,yy,xx-1,yy-1,2]); attack.append([xx,yy,xx-1,yy-1,3]); attack.append([xx,yy,xx-1,yy-1,4])
    if xx+1<TX and yy-1>=0 and skaki[xx+1][yy-1]*kommati<0:
        if yy-1!=0: attack.append([xx,yy,xx+1,yy-1])
        else: attack.append([xx,yy,xx+1,yy-1,1]); attack.append([xx,yy,xx+1,yy-1,2]); attack.append([xx,yy,xx+1,yy-1,3]); attack.append([xx,yy,xx+1,yy-1,4])
    if skaki[TX+1][4]!=0 and yy==3 and abs(skaki[TX+1][4]-1-xx)==1: attack.append([xx,yy,skaki[TX+1][4]-1,yy-1])
elif kommati==-6: #Mayro Pioni
    if yy+1<TY and skaki[xx][yy+1]==0:
        if yy+1!=7: moves.append([xx,yy,xx,yy+1])
        else: moves.append([xx,yy,xx,yy+1,-1]); moves.append([xx,yy,xx,yy+1,-2]); moves.append([xx,yy,xx,yy+1,-3]); moves.append([xx,yy,xx,yy+1,-4])
    if yy==1 and skaki[xx][yy+1]==0 and skaki[xx][yy+2]==0: moves.append([xx,yy,xx,yy+2])
    if xx-1>=0 and yy+1<TY and skaki[xx-1][yy+1]*kommati<0:
        if yy+1!=7: attack.append([xx,yy,xx-1,yy+1])
        else: attack.append([xx,yy,xx-1,yy+1,-1]); attack.append([xx,yy,xx-1,yy+1,-2]); attack.append([xx,yy,xx-1,yy+1,-3]); attack.append([xx,yy,xx-1,yy+1,-4])
    if xx+1<TX and yy+1<TY and skaki[xx+1][yy+1]*kommati<0:
        if yy+1!=7: attack.append([xx,yy,xx+1,yy+1])
        else: attack.append([xx,yy,xx+1,yy+1,-1]); attack.append([xx,yy,xx+1,yy+1,-2]); attack.append([xx,yy,xx+1,yy+1,-3]); attack.append([xx,yy,xx+1,yy+1,-4])
    if skaki[TX+1][0]!=0 and yy==4 and abs(skaki[TX+1][0]-1-xx)==1: attack.append([xx,yy,skaki[TX+1][0]-1,yy+1])
```

Παρεμπιπτόντως, στο υποπρόγραμμα superValidMovesAndAttacks δεν χρειάζονται αλλαγές όταν παιχθεί πιόνι στη γραμμή προαγωγής καθώς δεν απαιτείται να ενημερωθεί η εικονική δομή sk με το κομμάτι προαγωγής. Είναι αρκετό να ελεγχθεί η προκύπτουσα θέση για ΣΑΧ με πιόνι στη θέση προαγωγής, κάτι που έχει γίνει προηγουμένως με την **`sk[mv[2]][mv[3]]=sk[mv[0]][mv[1]]`**.

Επίσης, αλλάζουμε στην περιοχή του GameLoop την συνθήκη που εντοπίζει έγκυρο δεύτερο κλικ.

```
elif k==2 and (([klik1x,klik1y,x,y] in mov+att) or ([klik1x,klik1y,x,y,turn] in mov+att)): #Egkyro deytero klik
```

Ο λόγος των δύο τελευταίων τροποποιήσεων είναι οι λίστες επιτρεπτών κινήσεων να είναι πλήρεις. Δηλαδή να περιλαμβάνουν τέσσερις αντί για μία κίνηση προαγωγής. Οι τέσσερις αυτές κινήσεις αποτελούνται πλέον από πέντε στοιχεία αντί για τέσσερα που αποτελούν τις κινήσεις μας, δηλαδή εκτός τις δύο συντεταγμένες εκκίνησης και τις δύο συντεταγμένες προορισμού περιλαμβάνεται και ο κωδικός του κομματιού προαγωγής. Π.χ. η έκφραση **`moves.append([xx,yy,xx,yy-1,1])`** σημαίνει προσθήκη της κίνησης προαγωγής του λευκού σε πύργο ενώ π.χ. η έκφραση **`moves.append([xx,yy,xx,yy-1,2])`** σημαίνει προσθήκη της κίνησης προαγωγής του λευκού σε ίππο κτλ., δηλαδή το πέμπτο στοιχείο της κίνησης αφορά τον κωδικό του κομματιού στο οποίο πρόκειται να προαχθεί το πιόνι. Κρίνονται απαραίτητες αυτές οι αλλαγές, κυρίως λόγω της τεχνητής νοημοσύνης που θα εξεταστεί σε μελλοντικά κεφάλαια, όπου είναι απαραίτητο να είναι απόλυτα διακριτές οι διάφορες κινήσεις προαγωγής.

Το παιχνίδι μας δεν είναι έτοιμο. Αυτό διότι δεν έχει γίνει προγραμματισμός του ΣΑΧ, του ΜΑΤ ή του ΠΑΤ. Αυτά θα εξεταστούν στη συνέχεια.

ΚΕΦΑΛΑΙΟ 19, Έλεγχος για ΣΑΧ, ΜΑΤ, ΠΑΤ

Η δέκατη στήλη έχει δεδομένα χρήσιμα για τα ροκέ και το αν πασάν. Από εδώ και στο εξής θα χρησιμοποιήσουμε και μια εντέκατη στήλη στη δομή skaki. Στο πρώτο της στοιχείο **`skaki[TX+2][0]`** θα κρατάμε ποιος έχει σειρά να παίξει. Αυτή τη δουλειά, μέχρι στιγμής την έκανε η μεταβλητή turn, όμως με το νέο τρόπο, η δομή skaki γίνεται πλέον συμπαγής και προσδιορίζει απόλυτα μια οποιαδήποτε σκακιστική θέση. Διαθέτει δεδομένα για τη θέση όλων των κομματιών, δεδομένα για δυνατότητες ροκέ ή αν πασάν, διαθέτει πλέον και το δεδομένο ποιος έχει σειρά να παίξει και θα δούμε στη συνέχεια ότι θα διαθέτει και πιθανά δεδομένα ΣΑΧ, ΜΑΤ και ΠΑΤ.

Κατ' αρχήν πρέπει η αρχικοποίηση της δομής skaki να δημιουργεί μια επιπλέον στήλη. Δημιουργούμε αντίγραφο του προηγούμενου προγράμματος, αποθηκεύουμε π.χ. ως 7Skaki19.py και διορθώνουμε το **`range(TX+2)`** σε **`range(TX+3)`**.

```
skaki = [] #Arxikopoihsh skakieras
for i in range(TX+3):
    column = []
    for j in range(TY):
        column.append(0)
    skaki.append(column)
```

Λίγο παρακάτω η εντολή **`turn = 1`** πρέπει να αντικατασταθεί:

```
skaki[TX+2][0]=1 #Arxika exei seira o leykos kai skaki[TX+2][0]=1.
```

Επίσης πρέπει να αντικατασταθούν και όλες οι άλλες εμφανίσεις του turn.

```
if k==1 and skaki[x][y]*skaki[TX+2][0]>0: #An sthn katastash 1 ginei klik
elif k==2 and (([klik1x,klik1y,x,y] in mov+att) or ([klik1x,klik1y,x,y,skaki[TX+2][0]] in mov+att)): #Egkyro deytero klik
    elif y==0: #Proagwgh leykoy pioniou
        drawKommatiaProagwghs(skaki[TX+2][0])
        pygame.display.update()
    elif y==7: #Proagwgh mayroy pioniou
        drawKommatiaProagwghs(skaki[TX+2][0])
        pygame.display.update()
    k=1
    skaki[TX+2][0] = skaki[TX+2][0]*-1
```

Δοκιμάζουμε το πρόγραμμα. Πρέπει να τρέχει όπως πριν.

Ανατρέχοντας στο σχήμα της δομής skaki στο κεφάλαιο 16, βλέπει κανείς ότι γενικά η ένατη στήλη αφορά δεδομένα για τους βασιλιάδες. Τα πρώτα τέσσερα στοιχεία αφορούν το λευκό βασιλιά, ενώ τα τέσσερα τελευταία αφορούν το μαύρο βασιλιά. Με αυτά, όπως ήδη

ειπώθηκε, κρατάμε δεδομένα συντεταγμένων τους και δεδομένα για το αν κινήθηκαν, χρήσιμα στο ροκέ. Το τέταρτο στοιχείο της ένατης στήλης, το οποίο δεν το έχουμε δει μέχρι στιγμής, θα χρησιμεύσει για έλεγχο ΣΑΧ, ΜΑΤ ή ΠΑΤ στον λευκό, ενώ αντίστοιχα το όγδοο στοιχείο θα χρησιμεύσει για έλεγχο ΣΑΧ, ΜΑΤ ή ΠΑΤ στον μαύρο. Συμφωνούμε, τιμή 1 να δηλώνει ΣΑΧ, τιμή 2 να δηλώνει ΜΑΤ και τιμή 3 να δηλώνει ΠΑΤ.

Σε κάθε σκακιστική θέση, για να μπορούμε να αποφανθούμε αν υπάρχει κατάσταση ΣΑΧ, ΜΑΤ ή ΠΑΤ, ακολουθείται γενικά η εξής λογική: Αρχικά εντοπίζονται όλες οι δυνατές κινήσεις του παίκτη που έχει σειρά. Αν τυχαίνει να απειλείται ο βασιλιάς του και δεν υπάρχουν δυνατές κινήσεις, η κατάσταση λέγεται ΜΑΤ. Αν τυχαίνει να απειλείται αλλά υπάρχουν δυνατές κινήσεις, η κατάσταση λέγεται ΣΑΧ. Αν τυχαίνει να μην απειλείται και ταυτοχρόνως να μην υπάρχουν δυνατές κινήσεις, η κατάσταση λέγεται ΠΑΤ. Και αν τυχαίνει να μην απειλείται και να υπάρχουν δυνατές κινήσεις, τότε πρόκειται για μια ειρηνική (δίχως ΣΑΧ) φυσιολογική κατάσταση. Αυτά που περιγράφηκαν εδώ, υλοποιούνται με τον παρακάτω κώδικα, ο οποίος μπαίνει κάπου στην αρχή του GameLoop.

```
k=1
skaki[TX+2][0]=1 #Arxika exei seira o leykos kai skaki[TX+2][0]=1. -1 otan paizei o mayros.
shmaia=True
while True: #GameLoop
    mouseClicked = False
    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit(); sys.exit()
        elif event.type == MOUSEBUTTONUP: #Klik pontikioy
            mousex, mousey = event.pos; mouseClicked = True
    if shmaia==True:
        allMoves=[]
        allAttacks=[]
        for x in range(TX):
            for y in range(TY):
                if skaki[x][y] != 0 and int(skaki[x][y]/abs(skaki[x][y]))==skaki[TX+2][0]:
                    allmov, allatt = superValidMovesAndAttacks(skaki,x,y,skaki[x][y])
                    allMoves = allMoves + allmov
                    allAttacks = allAttacks + allatt
        pygame.draw.rect(epif,BLACK,(PER,PER + TY*(SZ+KENO),TX*(SZ+KENO)-KENO,2*SZ+KENO))
        theshX=int(-0.4*5*skaki[TX+2][0]+2); theshY=int(-0.4*5*skaki[TX+2][0]+3)
        if attackMovesTo(skaki,skaki[TX][theshX],skaki[TX][theshY],5*skaki[TX+2][0]) != []:
            if allMoves+allAttacks == []:
                grapseKeimeno('MAT',50,(PER + int(TX/2)*(SZ+KENO),(TY+1)*(SZ+KENO)))
                skaki[TX][int(-0.4*5*skaki[TX+2][0]+5)]=2 #Kwdikos gia MAT
            else:
                grapseKeimeno('ΣΑΧ',50,(PER + int(TX/2)*(SZ+KENO),(TY+1)*(SZ+KENO)))
                skaki[TX][int(-0.4*5*skaki[TX+2][0]+5)]=1 #Kwdikos gia SAX
        else:
            if allMoves+allAttacks == []:
                grapseKeimeno('ΠΑΤ',50,(PER + int(TX/2)*(SZ+KENO),(TY+1)*(SZ+KENO)))
                skaki[TX][int(-0.4*5*skaki[TX+2][0]+5)]=3 #Kwdikos gia PAT
            else:
                skaki[TX][int(-0.4*5*skaki[TX+2][0]+5)]=0 #Kwdikos gia TIPOTA
        pygame.display.update(); time.sleep(0.5)
        shmaia=False
    if mouseClicked == True:
        x, y = xyFromMousexMousey(mousex, mousey)
```

Αρχικά βλέπουμε μια νέα μεταβλητή shmaia ακριβώς πριν το GameLoop, να αρχικοποιείται με **`True`**. Θα παίξει το ρόλο να γίνεται μόνο μια φορά η διαδικασία που προαναφέρθηκε. Αμέσως μετά τον έλεγχο συμβάντων (που παραμένει ως έχει), παρατηρούμε να ελέγχεται η μεταβλητή shmaia. Μόνο αν η τιμή της είναι **`True`**, θα ακολουθήσουν οι εντολές που ακολουθούν. Αρχικά, με το διπλό for εντοπίζονται όλες οι δυνατές κινήσεις του παίκτη που έχει σειρά. Κάθε τετράγωνο της σκακιέρας ελέγχεται αν «κατοικείται» από κομμάτι χρώματος που έχει σειρά να παίξει. Σε αυτό βοηθά η **`if skaki[x][y] != 0 and int(skaki[x][y]/abs(skaki[x][y]))==skaki[TX+2][0]`**.

Το υποπρόγραμμα superValidMovesAndAttacks θα κληθεί λοιπόν για όλα τα κομμάτια αυτά και τελικά θα προκύψουν οι λίστες allMoves και allAttacks. Στη συνέχεια, για λόγους αρχικοποίησης μαυρίζεται η επιφάνεια κάτω από τη σκακιέρα. Στη συνέχεια εντοπίζονται οι συντεταγμένες θέσης του βασιλιά χρώματος που έχει σειρά και αποθηκεύονται στις μεταβλητές theshX και theshY. Η τιμή της παράστασης **`int(-0.4*5*skaki[TX+2][0]+2)`** εξαρτάται από το χρώμα που έχει σειρά, δηλαδή από το **`[TX+2][0]`**, το οποίο είναι είτε 1, είτε -1. Αν είναι 1, τότε η παράσταση υπολογίζεται σε 0, ενώ αν είναι -1, η παράσταση υπολογίζεται σε 4. Παρόμοια, υπολογίζεται η τιμή της παράστασης **`int(-0.4*5*skaki[TX+2][0]+3)`** που είναι είτε 1 είτε 5. Αυτό το 0 και το 1 ή το 4 και το 5 θα χρησιμοποιηθούν στη συνέχεια από το υποπρόγραμμα attackMovesTo για να ελεγχθεί αν απειλείται ο λευκός ή ο μαύρος βασιλιάς. Διότι έχουμε πει ότι στις θέσεις 0 και 1 της ένατης στήλης βρίσκονται οι συντεταγμένες θέσης του λευκού βασιλιά, ενώ στις θέσεις 4 και 5 της ένατης στήλης βρίσκονται οι συντεταγμένες θέσης του μαύρου βασιλιά.

Αν το attackMovesTo δεν επιστρέψει κενή λίστα (δηλαδή υπάρχουν απειλές), τότε αν δεν υπάρχουν δυνατές κινήσεις (**`allMoves+allAttacks == []`**) έχουμε ΜΑΤ, το οποίο εκτυπώνεται κάτω από τη σκακιέρα. Θα εξηγηθεί παρακάτω το σχετικά απλό υποπρόγραμμα grapseKeimeno. Και επίσης, στο κατάλληλο σημείο της ένατης στήλης της δομής skaki, θα καταχωρηθεί τιμή 2 που κωδικοποιεί το ΜΑΤ. Αλλιώς, δηλαδή αν υπάρχουν απειλές αλλά υπάρχει απάντηση (δηλαδή η λίστα **`allMoves+allAttacks`** δεν είναι κενή), τότε εκτυπώνεται ΣΑΧ και επίσης στο κατάλληλο σημείο της δομής skaki καταχωρείται τιμή 1 (κωδικός ΣΑΧ).

Αν δεν υπάρχει απειλή στο βασιλιά, τότε αν δεν υπάρχουν δυνατές κινήσεις εκτυπώνεται ΠΑΤ και επίσης στο κατάλληλο σημείο της δομής skaki καταχωρείται τιμή 3 (κωδικός ΠΑΤ). Αλλιώς, το κατάλληλο σημείο που δείχνει τη κατάσταση του βασιλιά μηδενίζεται. Η τελευταία εντολή **`shmaia=False`** αποτρέπει να ξαναγίνουν όλα αυτά, διότι οι εντολές του GameLoop, εκτελούνται αέναα. Η shmaia θα ξαναπάρει τιμή **`True`** παρακάτω, όταν ολοκληρωθεί μια κίνηση, ώστε να ελεγχθεί αυτή η νέα κατάσταση:

```
                    k=1
                    skaki[TX+2][0] = skaki[TX+2][0]*-1
                    shmaia=True
    pygame.display.update()
```

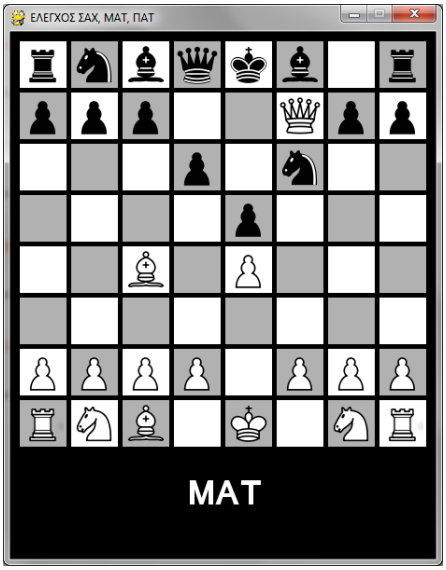
Κάπου στο σημείο των υποπρογραμμάτων, προσθέτουμε και το παρακάτω:

```
def grapseKeimeno(keimeno,size,thesh):
    font = pygame.font.SysFont(None, size)
    text = font.render(keimeno,1,WHITE,BLACK)
    textRect = text.get_rect()
    textRect.center = thesh
    epif.blit(text, textRect)
```

Έχουμε ήδη δει στο κεφάλαιο 3 τον τρόπο με τον οποίο εμφανίζεται κείμενο στην επιφάνειά μας, όπου και όπως θέλει κανείς. Ας ανατρέξει κανείς εκεί για να λύσει τυχόν απορίες του για τις σχετικές εντολές, οι οποίες σχεδόν αυτούσια έχουν χρησιμοποιηθεί

εδώ. Όμως οι εντολές αυτές, εδώ μπήκαν σε υποπρόγραμμα, ώστε να αποφύγουμε να γράφουμε τρεις φορές τα ίδια πράγματα στον κώδικα του κυρίως προγράμματος. Αντί αυτού, γίνονται τρεις κομψές κλήσεις του υποπρογράμματος grapseKeimeno.

Αποθηκεύουμε το πρόγραμμα και το τρέχουμε.



Τα μηνύματα ΣΑΧ, ΜΑΤ, ΠΑΤ βγαίνουν κανονικά κάτω από τη σκακιέρα όταν πρέπει.

ΚΕΦΑΛΑΙΟ 20, Τεχνητή νοημοσύνη (1)

Ήρθε η ώρα για ένα μεγάλο άλμα. Μέχρι τώρα, στο παιχνίδι εμπλέκονται δύο άνθρωποι. Παίζουν εναλλάξ ο καθένας τα κομμάτια του χρώματός τους και γίνεται ένα παιχνίδι που περιγράφεται και ως παιχνίδι Άνθρωπος εναντίον Ανθρώπου. Με ελάχιστες αλλαγές στον κώδικα του προηγούμενου προγράμματος, θα δημιουργήσουμε ένα σκάκι του τύπου Άνθρωπος εναντίον Υπολογιστή.

Η πιο στοιχειώδης μορφή τεχνητής νοημοσύνης είναι αυτή όπου ο υπολογιστής επιλέγει εντελώς τυχαία μια από τις δυνατές κινήσεις. Αυτό θα υλοποιηθεί εδώ, και είναι απαραίτητο το πακέτο εντολών random, το οποίο πρέπει να δηλωθεί στην πρώτη εντολή του κώδικά μας, την import. Δημιουργούμε αντίγραφο του προηγούμενου προγράμματος και αποθηκεύουμε π.χ. ως 7skaki20.py . Η πρώτη γραμμή πρέπει να αλλάξει ως εξής:

```
import pygame, sys, time, copy, random; from pygame.locals import *
```

Εντός του GameLoop, πρέπει να δημιουργηθούν δύο επιλογές: Όταν έχει σειρά ο άνθρωπος (κατά σύμβαση ο λευκός), και όταν έχει σειρά ο υπολογιστής (κατά σύμβαση ο μαύρος). Στο σημείο που ελέγχεται αν έγινε κλικ θα προστεθεί και η συνθήκη **`skaki[TX+2][0]==1`**, ενώ θα δημιουργηθεί μια παράλληλη με αυτήν επιλογή elif όπου θα μπει η συνθήκη **`skaki[TX+2][0]==-1`**. Προσθέτουμε λοιπόν κώδικα ως εξής:

```
if mouseClicked == True and skaki[TX+2][0]==1 and skaki[TX][3]!=2 and skaki[TX][3]!=3:
```

Και λίγο παρακάτω, όταν ολοκληρωθεί έγκυρη ανθρώπινη κίνηση:

```
        elif k==2 and (([klik1x,klik1y,x,y] in mov+att) or ([klik1x,klik1y,x,y,skaki[TX+2][0]] in mov+att)): #E
            pygame.draw.rect(epif,BLACK,(PER+klik1x*(SZ+KENO)-2,PER+klik1y*(SZ+KENO)-2,SZ+3,SZ+3),2)
            for mv in mov+att:
                pygame.draw.rect(epif,BLACK,(PER+mv[2]*(SZ+KENO)-2,PER+mv[3]*(SZ+KENO)-2,SZ+3,SZ+3),2)
            skaki[klik1x][klik1y]=0; skaki[x][y]=klik1kommati
            drawKommati(klik1x,klik1y); drawKommati(x,y)
            if klik1kommati==5:
                skaki[TX][0]=x; skaki[TX][1]=y; skaki[TX][2]=1 #Metakinhsh leykoy vasilia
                if x-klik1x==2: skaki[7][7]=0; skaki[5][7]=1; drawKommati(7,7); drawKommati(5,7) #Mikro roke
                elif klik1x-x==2: skaki[0][7]=0; skaki[3][7]=1; drawKommati(0,7); drawKommati(3,7) #Megalo roke
            elif klik1kommati==6: #Metakinhsh leykoy pionioy
                if klik1y-y==2: #An diplh kinhsh leykoy pionioy
                    skaki[TX+1][0]=x+1 #Markaretai to shmeio elegxoy an pasan toy leykoy
                elif skaki[TX+1][4]!=0 and y==2 and x==skaki[TX+1][4]-1: #An xtyphma an pasan toy leykoy
                    skaki[skaki[TX+1][4]-1][3]=0; drawKommati(skaki[TX+1][4]-1,3) #Feygei to xtyphmeno
                elif y==0: #Proagwgh leykoy pionioy
                    drawKommatiaProagwghs(skaki[TX+2][0])
                    pygame.display.update()
                    while True:
                        mouseClicked = False
                        for event in pygame.event.get():
                            if event.type == QUIT:
                                pygame.quit(); sys.exit()
                            elif event.type == MOUSEBUTTONDOWN:
                                mousex, mousey = event.pos
                                mouseClicked = True
                        if mouseClicked == True:
                            iRect1 = pygame.Rect(PER + 2*(SZ+KENO),PER + (TY+1)*(SZ+KENO),SZ,SZ)
                            if iRect1.collidepoint(mousex, mousey):
                                skaki[x][y]=1; drawKommati(x,y);
                            iRect2 = pygame.Rect(PER + 3*(SZ+KENO),PER + (TY+1)*(SZ+KENO),SZ,SZ)
                            if iRect2.collidepoint(mousex, mousey):
                                skaki[x][y]=2; drawKommati(x,y);
                            iRect3 = pygame.Rect(PER + 4*(SZ+KENO),PER + (TY+1)*(SZ+KENO),SZ,SZ)
                            if iRect3.collidepoint(mousex, mousey):
                                skaki[x][y]=3; drawKommati(x,y);
                            iRect4 = pygame.Rect(PER + 5*(SZ+KENO),PER + (TY+1)*(SZ+KENO),SZ,SZ)
                            if iRect4.collidepoint(mousex, mousey):
                                skaki[x][y]=4; drawKommati(x,y);
                        if skaki[x][y]!=6: drawKommatiaProagwghsEnd(); break
            if not(klik1kommati==6 and klik1y-y==2): skaki[TX+1][0]=0 #Epanafora shmeioy elegxoy an pasan toy l
            if not(klik1kommati==-6 and y-klik1y==2): skaki[TX+1][4]=0 #Epanafora shmeioy elegxoy an pasan toy
            if (klik1x==7 and klik1y==7) or (x==7 and y==7): skaki[TX+1][2]=1 #Akyrwsh dynatothtas leykoy gia m
            if (klik1x==0 and klik1y==7) or (x==0 and y==7): skaki[TX+1][3]=1 #Akyrwsh dynatothtas leykoy gia m
            if (klik1x==7 and klik1y==0) or (x==7 and y==0): skaki[TX+1][6]=1 #Akyrwsh dynatothtas mayroy gia m
            if (klik1x==0 and klik1y==0) or (x==0 and y==0): skaki[TX+1][7]=1 #Akyrwsh dynatothtas mayroy gia m
            k=1
            skaki[TX+2][0] = -1
            shmaia=True
elif skaki[TX+2][0]==-1 and skaki[TX][7]!=2 and skaki[TX][7]!=3:
```

Ο παραπάνω κώδικας όταν παίζει ο άνθρωπος, έχει εξηγηθεί λίγο πολύ σε προηγούμενα κεφάλαια. Έχουν αφαιρεθεί περιττά πράγματα. Για παράδειγμα όταν παίζει ο άνθρωπος, εφόσον έχει τα λευκά δεν χρειάζεται να υπάρχουν έλεγχοι μήπως παίχτηκε μαύρο κομμάτι. Έτσι οι επιλογές **`elif klik1kommati==-5`** και **`elif klik1kommati==-6`** απαλείφθηκαν μαζί με όλες τις εντολές που τις ακολουθούσαν. Στον κώδικα της περίπτωσης που παίζει ο υπολογιστής πρέπει να προστεθούν τα εξής:

```
            skaki[TX+2][0] = -1
            shmaia=True
elif skaki[TX+2][0]==-1 and skaki[TX][7]!=2 and skaki[TX][7]!=3:
    compMove=random.choice(allMoves+allAttacks)
    klik1x=compMove[0]; klik1y=compMove[1]; klik1kommati=skaki[compMove[0]][compMove[1]]
    x=compMove[2]; y=compMove[3]; skaki[klik1x][klik1y]=0; skaki[x][y]=klik1kommati
    pygame.draw.rect(epif,RED,(PER+klik1x*(SZ+KENO)-2,PER+klik1y*(SZ+KENO)-2,SZ+3,SZ+3),2)
    pygame.display.update(); time.sleep(0.5)
    pygame.draw.rect(epif,GREEN,(PER+x*(SZ+KENO)-2,PER+y*(SZ+KENO)-2,SZ+3,SZ+3),2)
    pygame.display.update(); time.sleep(0.5)
    drawKommati(klik1x,klik1y)
    pygame.draw.rect(epif,BLACK,(PER+klik1x*(SZ+KENO)-2,PER+klik1y*(SZ+KENO)-2,SZ+3,SZ+3),2)
    drawKommati(x,y)
    pygame.display.update(); time.sleep(0.5)
    pygame.draw.rect(epif,BLACK,(PER+x*(SZ+KENO)-2,PER+y*(SZ+KENO)-2,SZ+3,SZ+3),2)
    if klik1kommati==-5:
        skaki[TX][4]=x; skaki[TX][5]=y; skaki[TX][6]=1 #Metakinhsh mayroy vasilia
        if x-klik1x==2: skaki[7][0]=0; skaki[5][0]=-1; drawKommati(7,0); drawKommati(5,0) #Mikro roke
        elif klik1x-x==2: skaki[0][0]=0; skaki[3][0]=-1; drawKommati(0,0); drawKommati(3,0) #Megalo roke
    elif klik1kommati==-6: #Metakinhsh mayroy pionioy
        if y-klik1y==2: #An diplh kinhsh mayroy pionioy
            skaki[TX+1][4]=x+1 #Markaretai to shmeio elegxoy an pasan toy mayroy
        elif skaki[TX+1][0]!=0 and y==5 and x==skaki[TX+1][0]-1: #An xtyphma an pasan toy mayroy
            skaki[skaki[TX+1][0]-1][4]=0; drawKommati(skaki[TX+1][0]-1,4) #Feygei to xtyphmeno
        elif y==7:
            skaki[x][y]=compMove[4]; drawKommati(x,y);
    if not(klik1kommati==6 and klik1y-y==2): skaki[TX+1][0]=0 #Epanafora shmeioy elegxoy an pasan toy leykoy
    if not(klik1kommati==-6 and y-klik1y==2): skaki[TX+1][4]=0 #Epanafora shmeioy elegxoy an pasan toy mayroy
    if (klik1x==7 and klik1y==7) or (x==7 and y==7): skaki[TX+1][2]=1 #Akyrwsh dynatothtas leykoy gia mikro roke
    if (klik1x==0 and klik1y==7) or (x==0 and y==7): skaki[TX+1][3]=1 #Akyrwsh dynatothtas leykoy gia megalo roke
    if (klik1x==7 and klik1y==0) or (x==7 and y==0): skaki[TX+1][6]=1 #Akyrwsh dynatothtas mayroy gia mikro roke
    if (klik1x==0 and klik1y==0) or (x==0 and y==0): skaki[TX+1][7]=1 #Akyrwsh dynatothtas mayroy gia megalo roke
    skaki[TX+2][0] = 1
    shmaia=True

pygame.display.update()
```

Ο παραπάνω κώδικας, μοιάζει με αυτόν της περίπτωσης που παίζει ο άνθρωπος. Όμως έχουν γίνει σημαντικές προσθήκες. Η σημαντικότερη είναι η πρώτη εντολή όπου σε μια μεταβλητή compMove θα υπολογιστεί και θα καταχωρηθεί μια τυχαία κίνηση από όλες τις δυνατές. Όλες οι δυνατές κινήσεις έχουν προηγουμένως ήδη αποθηκευτεί στις δύο λίστες allMoves και allAttacks. Η εντολή **`random.choice`** είναι διαθέσιμη εφόσον εισάχθηκε αρχικά το πακέτο εντολών random. Δέχεται ως όρισμα μια λίστα με στοιχεία και επιστρέφει τυχαία ένα στοιχείο. Τα δεδομένα αυτής της κίνησης compMove, η οποία συνήθως είναι μια τετράδα αριθμών (συντεταγμένες εκκίνησης και προορισμού) καταχωρούνται αναλυτικά στις μεταβλητές klik1x, klik1y, klik1kommati, x, y, ώστε να έχουμε στη συνέχεια μια παρόμοια αντιμετώπιση της κατάστασης, με αυτήν που παίζει ο άνθρωπος. Σχετικά σπάνια η compMove θα είναι μια πεντάδα, όταν δηλαδή θα πρόκειται για κίνηση προαγωγής.

Στη συνέχεια: σχεδιάζεται με κόκκινο πλαίσιο το τετράγωνο που καταλαμβάνεται από το κομμάτι που παίζει κίνηση (βλέπε τα klik1x και klik1y στις παραμέτρους θέσης της εντολής **`pygame.draw.rect`**), με την **`pygame.display.update()`** ανανεώνεται η οθόνη, με την **`time.sleep(0.5)`** γίνεται μια μικρή διακοπή χρόνου, με την ακόλουθη **`pygame.draw.rect`** σχεδιάζεται πράσινο πλαίσιο στο τετράγωνο προορισμού (βλέπε τα x και y στις παραμέτρους θέσης), ανανεώνεται η οθόνη και γίνεται πάλι μια μικρή διακοπή χρόνου, με την **`drawKommati(klik1x,klik1y)`** αδειάζει ουσιαστικά το τετράγωνο εκκίνησης από το κομμάτι που το καταλάμβανε, με την ακόλουθη **`pygame.draw.rect`** βάφεται το κόκκινο πλαίσιο ξανά μαύρο, με την **`drawKommati(x,y)`** σχεδιάζεται το κομμάτι που παίζει στη νέα θέση του, με τις επόμενες εντολές ανανεώνεται η οθόνη και παραμένει σταθερή για λίγο χρόνο, και με την επόμενη **`pygame.draw.rect`** το πράσινο πλαίσιο στο τετράγωνο προορισμού ξαναμαυρίζει.

Στη συνέχεια απαιτούνται βέβαια εντολές που να ελέγχουν μήπως παίχτηκε ο βασιλιάς. Σε τέτοια περίπτωση, ενημερώνονται τα δεδομένα θέσης του βασιλιά **`skaki[TX][4]`** και **`skaki[TX][5]`**, το δεδομένο «ασάλευτου» βασιλιά **`skaki[TX][6]`** παίρνει τιμή 1 και αν παίχτηκε ροκέ, ενημερώνεται γι' αυτό η δομή skaki και σχεδιάζεται και ο πύργος. Εάν παίχτηκε πιόνι, αν παίχτηκε διπλή κίνηση ενημερώνεται κατάλληλα το δεδομένο αν πασάν **`skaki[TX+1][4]`**, αν παίχτηκε χτύπημα αν πασάν πρέπει να φύγει το χτυπημένο του αντιπάλου, ενημερώνοντας τόσο τη δομή skaki όσο και την οθόνη και αν παίχτηκε κίνηση προαγωγής πιονιού, τότε η εντολή **`skaki[x][y]=compMove[4]`** προσδιορίζει το κομμάτι στο οποίο γίνεται προαγωγή, το οποίο και σχεδιάζεται (**`compMove[4]`** αφορά μόνο τις κινήσεις προαγωγής και είναι το πέμπτο στοιχείο τους). Οι υπόλοιπες εντολές ενημερώνουν ενδεχομένως τη δομή skaki με δεδομένα που χρειάζονται για αν πασάν ή ροκέ.

Τέλος, η εντολή **`skaki[TX+2][0] = 1`** δίνει σειρά στο λευκό (δηλαδή στον άνθρωπο) και η εντολή **`shmaia=True`** ουσιαστικά επιτρέπει να εκτελεστούν ξανά οι εντολές που βρίσκονται στην αρχή του GameLoop και εντοπίζουν όλες τις επιτρεπτές κινήσεις και χτυπήματα allMoves και allAttacks στη δεδομένη κατάσταση της σκακιέρας.

ΚΕΦΑΛΑΙΟ 21, Τεχνητή νοημοσύνη (2)

Με το παραπάνω πρόγραμμα, ο άνθρωπος έχει πάντα τα λευκά και ο υπολογιστής τα μαύρα. Με ελάχιστες αλλαγές, το πρόγραμμα μπορεί να παίξει με τυχαία τα χρώματα για άνθρωπο και υπολογιστή. Δημιουργούμε αντίγραφο του προηγούμενου προγράμματος και αποθηκεύουμε π.χ. ως 7skaki21.py . Προσθέτουμε δύο μόνο νέες γραμμές κώδικα ακριβώς πριν το GameLoop.

```
k=1
comp=random.choice([1,-1])
anthr=-1*comp
skaki[TX+2][0]=1 #Arxika
```

Με τη βοήθεια της εντολής random.choice, η μεταβλητή comp παίρνει τυχαία μια από τις δύο τιμές 1 ή -1, προσδιορίζοντας έτσι το χρώμα του υπολογιστή, όπου 1 σημαίνει λευκά και -1 σημαίνει μαύρα. Η μεταβλητή anthr (greeklish άνθρωπος), θα παίρνει στην επόμενη εντολή την αντίθετη τιμή της comp. Μέσα στο GameLoop, οι βασικές συνθήκες επιλογής πρέπει να αλλάξουν:

```
if mouseClicked == True and skaki[TX+2][0]==anthr and \
   skaki[TX][-2*skaki[TX+2][0]+5]!=2 and skaki[TX][-2*skaki[TX+2][0]+5]!=3:
```

```
elif skaki[TX+2][0]==comp and skaki[TX][-2*skaki[TX+2][0]+5]!=2 \
     and skaki[TX][-2*skaki[TX+2][0]+5]!=3:
```

Υπενθυμίζεται ότι ο χαρακτήρας \ μπορεί να σπάσει μια πολύ μεγάλη γραμμή κώδικα σε δύο κομμάτια, κάτι που εδώ κρίθηκε σκόπιμο.

Αντί για τις τιμές 3 και 7 που προσδιορίζουν το τέταρτο και το όγδοο στοιχείο της ένατης στήλης της δομής skaki (δηλαδή τα δεδομένα για ΣΑΧ, ΜΑΤ και ΠΑΤ), τοποθετήθηκε η παράσταση **`-2*skaki[TX+2][0]+5`**. Πράγματι, ανάλογα του χρώματος που έχει σειρά (το **`skaki[TX+2][0]`** θα έχει τιμή είτε 1 είτε -1), η παράσταση υπολογίζεται είτε σε 3, είτε σε 7. Υπενθυμίζεται ότι αν εντοπιστεί ΜΑΤ ή ΠΑΤ, δεν πρέπει να συνεχίσει το παιχνίδι να δίνει δυνατότητες κίνησης. Επίσης, όταν ολοκληρώνεται κίνηση ανθρώπου ή υπολογιστή, αντί για τις εντολές **`skaki[TX+2][0] = 1`** και **`skaki[TX+2][0] = -1`** πρέπει να γραφεί:

```
skaki[TX+2][0] = -1*skaki[TX+2][0]
```

Και τέλος, πρέπει να συμπληρωθούν τμήματα κώδικα που είχαν αφαιρεθεί στο προηγούμενο πρόγραμμα ως περιττά. Διότι τώρα είναι δυνατόν ο άνθρωπος να χειρίζεται και λευκά και μαύρα κομμάτια. Έτσι, από το προπροηγούμενο πρόγραμμα ξαναφέρνουμε τα εξής δύο κομμάτια κώδικα και το τοποθετούμε στα κατάλληλα σημεία του κώδικα που παίζει ο άνθρωπος:

```
elif klik1kommati==-5:
    skaki[TX][4]=x; skaki[TX][5]=y; skaki[TX][6]=1 #Metakinhsh mayroy vasilia
    if x-klik1x==2: skaki[7][0]=0; skaki[5][0]=-1; drawKommati(7,0); drawKommati(5,0) #Mikro roke
    elif klik1x-x==2: skaki[0][0]=0; skaki[3][0]=-1; drawKommati(0,0); drawKommati(3,0) #Megalo roke
```

```
elif klik1kommati==-6: #Metakinhsh mayroy pionioy
    if y-klik1y==2: #An diplh kinhsh mayroy pionioy
        skaki[TX+1][4]=x+1 #Markaretai to shmeio elegxoy an pasan toy mayroy
    elif skaki[TX+1][0]!=0 and y==5 and x==skaki[TX+1][0]-1: #An xtyphma an pasan toy mayroy
        skaki[skaki[TX+1][0]-1][4]=0; drawKommati(skaki[TX+1][0]-1,4) #Feygei to xtyphmeno
    elif y==7: #Proagwgh mayroy pionioy
        drawKommatiaProagwghs(skaki[TX+2][0])
        pygame.display.update()
        while True:
            mouseClicked = False
            for event in pygame.event.get():
                if event.type == QUIT:
                    pygame.quit(); sys.exit()
                elif event.type == MOUSEBUTTONDOWN:
                    mousex, mousey = event.pos
                    mouseClicked = True
            if mouseClicked == True:
                iRect1 = pygame.Rect(PER + 2*(SZ+KENO),PER + (TY+1)*(SZ+KENO),SZ,SZ)
                if iRect1.collidepoint(mousex, mousey):
                    skaki[x][y]=-1; drawKommati(x,y);
                iRect2 = pygame.Rect(PER + 3*(SZ+KENO),PER + (TY+1)*(SZ+KENO),SZ,SZ)
                if iRect2.collidepoint(mousex, mousey):
                    skaki[x][y]=-2; drawKommati(x,y);
                iRect3 = pygame.Rect(PER + 4*(SZ+KENO),PER + (TY+1)*(SZ+KENO),SZ,SZ)
                if iRect3.collidepoint(mousex, mousey):
                    skaki[x][y]=-3; drawKommati(x,y);
                iRect4 = pygame.Rect(PER + 5*(SZ+KENO),PER + (TY+1)*(SZ+KENO),SZ,SZ)
                if iRect4.collidepoint(mousex, mousey):
                    skaki[x][y]=-4; drawKommati(x,y);
            if skaki[x][y]!=-6: drawKommatiaProagwghsEnd(); break
```

Για τους ίδιους λόγους, όταν παίζει ο υπολογιστής προσθέτουμε τον σχετικό κώδικα ώστε αυτός να γίνει τελικά ως εξής:

```
            shmaia=True
elif skaki[TX+2][0]==comp and skaki[TX][-2*skaki[TX+2][0]+5]!=2 \
     and skaki[TX][-2*skaki[TX+2][0]+5]!=3: #An paizei o ypologisths kai den yparxei MAT h PAT
    compMove=random.choice(allMoves+allAttacks) #Epilegetai tyxaia h kinhsh toy ypologisth
    klik1x=compMove[0]; klik1y=compMove[1]; klik1kommati=skaki[compMove[0]][compMove[1]]
    x=compMove[2]; y=compMove[3]; skaki[klik1x][klik1y]=0; skaki[x][y]=klik1kommati
    pygame.draw.rect(epif,RED,(PER+klik1x*(SZ+KENO)-2,PER+klik1y*(SZ+KENO)-2,SZ+3,SZ+3),2)
    pygame.display.update(); time.sleep(0.5)
    pygame.draw.rect(epif,GREEN,(PER+x*(SZ+KENO)-2,PER+y*(SZ+KENO)-2,SZ+3,SZ+3),2)
    pygame.display.update(); time.sleep(0.5)
    drawKommati(klik1x,klik1y)
    pygame.draw.rect(epif,BLACK,(PER+klik1x*(SZ+KENO)-2,PER+klik1y*(SZ+KENO)-2,SZ+3,SZ+3),2)
    drawKommati(x,y)
    pygame.display.update(); time.sleep(0.5)
    pygame.draw.rect(epif,BLACK,(PER+x*(SZ+KENO)-2,PER+y*(SZ+KENO)-2,SZ+3,SZ+3),2)
    if klik1kommati==5:
        skaki[TX][0]=x; skaki[TX][1]=y; skaki[TX][2]=1 #Metakinhsh leykoy vasilia
        if x-klik1x==2: skaki[7][7]=0; skaki[5][7]=1; drawKommati(7,7); drawKommati(5,7) #Mikro roke
        elif klik1x-x==2: skaki[0][7]=0; skaki[3][7]=1; drawKommati(0,7); drawKommati(3,7) #Megalo roke
    elif klik1kommati==-5:
        skaki[TX][4]=x; skaki[TX][5]=y; skaki[TX][6]=1 #Metakinhsh mayroy vasilia
        if x-klik1x==2: skaki[7][0]=0; skaki[5][0]=-1; drawKommati(7,0); drawKommati(5,0) #Mikro roke
        elif klik1x-x==2: skaki[0][0]=0; skaki[3][0]=-1; drawKommati(0,0); drawKommati(3,0) #Megalo roke
    elif klik1kommati==6: #Metakinhsh leykoy pionioy
        if klik1y-y==2: #An diplh kinhsh leykoy pionioy
            skaki[TX+1][0]=x+1 #Markaretai to shmeio elegxoy an pasan toy leykoy
        elif skaki[TX+1][4]!=0 and y==2 and x==skaki[TX+1][4]-1: #An xtyphma an pasan toy leykoy
            skaki[skaki[TX+1][4]-1][3]=0; drawKommati(skaki[TX+1][4]-1,3) #Feygei to xtyphmeno
        elif y==0: #Proagwgh leykoy pionioy
            skaki[x][y]=compMove[4]; drawKommati(x,y);
    elif klik1kommati==-6: #Metakinhsh mayroy pionioy
        if y-klik1y==2: #An diplh kinhsh mayroy pionioy
            skaki[TX+1][4]=x+1 #Markaretai to shmeio elegxoy an pasan toy mayroy
        elif skaki[TX+1][0]!=0 and y==5 and x==skaki[TX+1][0]-1: #An xtyphma an pasan toy mayroy
            skaki[skaki[TX+1][0]-1][4]=0; drawKommati(skaki[TX+1][0]-1,4) #Feygei to xtyphmeno
        elif y==7: #Proagwgh mayroy pionioy
           skaki[x][y]=compMove[4]; drawKommati(x,y);
    if not(klik1kommati==6 and klik1y-y==2): skaki[TX+1][0]=0 #Epanafora shmeioy elegxoy an pasan toy le
    if not(klik1kommati==-6 and y-klik1y==2): skaki[TX+1][4]=0 #Epanafora shmeioy elegxoy an pasan toy m
    if (klik1x==7 and klik1y==7) or (x==7 and y==7): skaki[TX+1][2]=1 #Akyrwsh dynatothtas leykoy gia mi
    if (klik1x==0 and klik1y==7) or (x==0 and y==7): skaki[TX+1][3]=1 #Akyrwsh dynatothtas leykoy gia me
    if (klik1x==7 and klik1y==0) or (x==7 and y==0): skaki[TX+1][6]=1 #Akyrwsh dynatothtas mayroy gia mi
    if (klik1x==0 and klik1y==0) or (x==0 and y==0): skaki[TX+1][7]=1 #Akyrwsh dynatothtas mayroy gia me
    skaki[TX+2][0] = -1*skaki[TX+2][0]
    shmaia=True

pygame.display.update()
```

Αποθηκεύουμε και τρέχουμε το πρόγραμμα. Τυχαία επιλέγεται ποιος θα έχει τα λευκά. Αν ξεκινήσει ο υπολογιστής τη παρτίδα θα δούμε να παίζει ως λευκός την πρώτη κίνηση. Αν δεν ξεκινήσει ο υπολογιστής, πρέπει εμείς να παίξουμε την πρώτη κίνηση ως λευκοί.

ΚΕΦΑΛΑΙΟ 22, Τεχνητή νοημοσύνη (3)

Το τρίτο μέρος της τριλογίας μας για την τεχνητή νοημοσύνη είναι το πιο ενδιαφέρον. Θα επιδειχθεί ο τρόπος, ώστε να δημιουργηθεί ένα αήττητο πρόγραμμα σκακιού. Η ιδέα που κρύβεται πίσω από ένα τέτοιο μεγαλεπήβολο σχέδιο είναι απλή: Ο υπολογιστής θα κληθεί να υπολογίζει όλες τις δυνατές θέσεις στη σκακιέρα και την αξία τους, έπειτα από αρκετό πλήθος κινήσεων του λευκού και του μαύρου, με άλλα λόγια έπειτα από εξέταση σε μεγάλο βάθος όλων των δυνατών κινήσεων ή αλλιώς βαριάντων.

Γι' αυτό το σκοπό, αναπτύχθηκε το υποπρόγραμμα xristina, το οποίο πρέπει να προστεθεί στον υπάρχοντα κώδικά μας. Δημιουργούμε αντίγραφο του προηγούμενου προγράμματος, αποθηκεύουμε π.χ. ως 7skaki22.py και προσθέτουμε το ακόλουθο υποπρόγραμμα κάπου στην περιοχή των υποπρογραμμάτων.

```
def xristina(skaki,vathos):
    '''Dexetai mia dedomenh thesh skaki kai thn exetazei plhrws se kapoio vathos.
    Epistrefontai dyo times: H axia ths theshs kai h kalyterh kinhsh!'''
    allMoves=[] #Arxika ypologizontai oles oi epitreptes kinhseis
    allAttacks=[] #kai ola ta epitrepta xtyphmata ths theshs skaki
    for x in range(TX):
        for y in range(TY):
            if skaki[x][y] != 0 and int(skaki[x][y]/abs(skaki[x][y]))==skaki[TX+2][0]:
                allmov, allatt = superValidMovesAndAttacks(skaki,x,y,skaki[x][y])
                allMoves = allMoves + allmov
                allAttacks = allAttacks + allatt
    theshX=int(-0.4*5*skaki[TX+2][0]+2); theshY=int(-0.4*5*skaki[TX+2][0]+3)
    if attackMovesTo(skaki,skaki[TX][theshX],skaki[TX][theshY],5*skaki[TX+2][0]) != []:
        if allMoves+allAttacks == []:
            skaki[TX][int(-0.4*5*skaki[TX+2][0]+5)]=2 #Kwdikos gia MAT
            return -(1000+vathos)*skaki[TX+2][0], []
    else:
        if allMoves+allAttacks == []:
            skaki[TX][int(-0.4*5*skaki[TX+2][0]+5)]=3 #Kwdikos gia PAT
            return 0, []
    if vathos==0: #An vathos==0 ypologizetai kai epistrefetai h axia ths theshs skaki
        sum = 0
        for i in range(TX):
            for j in range(TY):
                if skaki[i][j]==1: sum = sum + 5
                elif skaki[i][j]==-1: sum = sum - 5
                elif skaki[i][j]==2: sum = sum + 3
                elif skaki[i][j]==-2: sum = sum - 3
                elif skaki[i][j]==3: sum = sum + 3
                elif skaki[i][j]==-3: sum = sum - 3
                elif skaki[i][j]==4: sum = sum + 9
                elif skaki[i][j]==-4: sum = sum - 9
                elif skaki[i][j]==6: sum = sum + 1
                elif skaki[i][j]==-6: sum = sum - 1
        return sum, []
    else: #vathos > 0
        if skaki[TX+2][0]==1:
            maxAxia=-1001
            bestMoves=[]
            for mv in allMoves+allAttacks:
                sk=[skaki[0][:],skaki[1][:],skaki[2][:],skaki[3][:],skaki[4][:],skaki[5][:], \
                 skaki[6][:],skaki[7][:],skaki[8][:],skaki[9][:],skaki[10][:]]
                sk[mv[2]][mv[3]]=sk[mv[0]][mv[1]] #Paizetai h kinhsh mv eikonika sth domh sk
                sk[mv[0]][mv[1]]=0
                if sk[mv[2]][mv[3]]==5:
                    sk[TX][0]=mv[2]; sk[TX][1]=mv[3]; sk[TX][2]=1 #Epikairopoihsh theshs vasilia
                    if mv[2]-mv[0]==2: sk[7][7]=0; sk[5][7]=1 #An mikro roke topothethsh pyrgoy
                    elif mv[0]-mv[2]==2: sk[0][7]=0; sk[3][7]=1 #An megalo roke topothethsh pyrgoy
                elif sk[mv[2]][mv[3]]==6:
                    if mv[1]-mv[3]==2: sk[TX+1][0]=mv[2]+1 #An diplh kinhsh, markaretai to shmeio elegxoy an pasan
                    elif sk[TX+1][4]!=0 and mv[3]==2 and mv[2]==sk[TX+1][4]-1: sk[sk[TX+1][4]-1][3]=0 #Xtyphma an pas
                    elif mv[3]==0: sk[mv[2]][mv[3]]=mv[4] #An proagwgh pionioy, topotheteitai to kommati proagwghs
                if not(sk[mv[2]][mv[3]]==6 and mv[1]-mv[3]==2): sk[TX+1][0]=0 #Epanafora shmeioy elegxoy an pasan
                if not(sk[mv[2]][mv[3]]==-6 and mv[3]-mv[1]==2): sk[TX+1][4]=0 #Epanafora shmeioy elegxoy an pasan
                if (mv[0]==7 and mv[1]==7) or (mv[2]==7 and mv[3]==7): sk[TX+1][2]=1
```

```
                if (mv[0]==0 and mv[1]==7) or (mv[2]==0 and mv[3]==7): sk[TX+1][3]=1
                if (mv[0]==7 and mv[1]==0) or (mv[2]==7 and mv[3]==0): sk[TX+1][6]=1
                if (mv[0]==0 and mv[1]==0) or (mv[2]==0 and mv[3]==0): sk[TX+1][7]=1
                sk[TX+2][0]=-1
                ax, mo = xristina(sk, vathos-1) #Anadromikh klhsh toy ypoprogrammatos
                if ax>maxAxia:
                    maxAxia=ax
                    bestMoves=[mv]
                elif ax==maxAxia:
                    bestMoves.append(mv)
            return maxAxia, random.choice(bestMoves)
        else: #skaki[TX+2][0]==-1
            minAxia=1001
            bestMoves=[]
            for mv in allMoves+allAttacks:
                sk=[skaki[0][:],skaki[1][:],skaki[2][:],skaki[3][:],skaki[4][:],skaki[5][:], \
                 skaki[6][:],skaki[7][:],skaki[8][:],skaki[9][:],skaki[10][:]]
                sk[mv[2]][mv[3]]=sk[mv[0]][mv[1]] #Paizetai h kinhsh mv eikonika sth domh sk
                sk[mv[0]][mv[1]]=0
                if sk[mv[2]][mv[3]]==-5:
                    sk[TX][4]=mv[2]; sk[TX][5]=mv[3]; sk[TX][6]=1 #Epikairopoihsh theshs vasilia
                    if mv[2]-mv[0]==2: sk[7][0]=0; sk[5][0]=-1 #An mikro roke topothethsh pyrgoy
                    elif mv[0]-mv[2]==2: sk[0][0]=0; sk[3][0]=-1 #An megalo roke topothethsh pyrgoy
                elif sk[mv[2]][mv[3]]==-6:
                    if mv[3]-mv[1]==2: sk[TX+1][4]=mv[2]+1 #An diplh kinhsh, markaretai to shmeio elegxoy an pasan
                    elif sk[TX+1][0]!=0 and mv[3]==5 and mv[2]==sk[TX+1][0]-1: sk[sk[TX+1][0]-1][4]=0 #Xtyphma an pas
                    elif mv[3]==7: sk[mv[2]][mv[3]]=mv[4] #An proagwgh pionioy, topotheteitai to kommati proagwghs
                if not(sk[mv[2]][mv[3]]==6 and mv[1]-mv[3]==2): sk[TX+1][0]=0 #Epanafora shmeioy elegxoy an pasan
                if not(sk[mv[2]][mv[3]]==-6 and mv[3]-mv[1]==2): sk[TX+1][4]=0 #Epanafora shmeioy elegxoy an pasan
                if (mv[0]==7 and mv[1]==7) or (mv[2]==7 and mv[3]==7): sk[TX+1][2]=1
                if (mv[0]==0 and mv[1]==7) or (mv[2]==0 and mv[3]==7): sk[TX+1][3]=1
                if (mv[0]==7 and mv[1]==0) or (mv[2]==7 and mv[3]==0): sk[TX+1][6]=1
                if (mv[0]==0 and mv[1]==0) or (mv[2]==0 and mv[3]==0): sk[TX+1][7]=1
                sk[TX+2][0]=1
                ax, mo = xristina(sk, vathos-1) #Anadromikh klhsh toy ypoprogrammatos
                if ax<minAxia:
                    minAxia=ax
                    bestMoves=[mv]
                elif ax==minAxia:
                    bestMoves.append(mv)
            return minAxia, random.choice(bestMoves)
```

Επίσης πρέπει να αλλάξει μόνο μία γραμμή κώδικα, στο σημείο που υπολογίζεται η κίνηση του υπολογιστή, καθώς δεν υπολογίζεται πλέον τυχαία.

```
 and skaki[TX][-2*skaki[TX+2][0]+5]!=3: #An paizei o ypologisths kai den yparxei MAT h PAT
axia, compMove=xristina(skaki, 1) #Epilegetai h kinhsh toy ypologisth symfwna me th xristina
klik1x=compMove[0]; klik1y=compMove[1]; klik1kommati=skaki[compMove[0]][compMove[1]
```

Αποθηκεύουμε και τρέχουμε το πρόγραμμα και διαπιστώνουμε ότι ο υπολογιστής παίζει με μια στοιχειώδη λογική, δηλαδή τουλάχιστον όταν υπάρχει κάτι για φάγωμα, το βλέπει. Δοκιμάζουμε το πρόγραμμα με αλλαγμένη δεύτερη παράμετρο σε 2, δηλαδή αλλάζουμε την προηγούμενη γραμμή σε

```
axia, compMove=xristina(skaki, 2) #Epilegetai
```

Παρατηρούμε βελτιωμένο παιχνίδι από τον υπολογιστή. Εξετάζει πλέον όλες τις προκύπτουσες θέσεις σε βάθος 2 κινήσεων (μια για κάθε χρώμα) και παίζει την κίνηση που ενδείκνυται. Αν αλλάξουμε τη δεύτερη παράμετρο σε 3 και τρέξουμε το πρόγραμμα, τότε ο υπολογιστής παίζει ακόμα καλύτερα. Και ούτω καθεξής. Όμως πληρώνουμε ένα σημαντικό τίμημα: Όσο περισσότερο βάθος εξετάζει ο υπολογιστής, τόσο περισσότερο χρόνο χρειάζεται γι' αυτό! Κάτι που είναι λογικό! Σε μελλοντικούς υπερυπολογιστές, το πρόγραμμα αυτό με αρκετά μεγάλη δεύτερη παράμετρο, γίνεται αήττητο!

Ας εξετάσουμε το υποπρόγραμμα xristina, το οποίο αποτελεί βέβαια το πιο δύσκολο τμήμα του βιβλίου αυτού, και ως εκ τούτου, τυχόν δυσκολίες κατανόησης δεν πρέπει να απελπίσουν ειδικά νεότερους προγραμματιστές.

Αρχικά, στις λίστες allMoves και allAttacks θα υπολογιστούν και θα καταχωρηθούν όλες οι επιτρεπτές κινήσεις και χτυπήματα της θέσης skaki. Κάτι παρόμοιο είδαμε και στο κεφάλαιο 19 όπου με το διπλό for εντοπίζονται όλες οι δυνατές κινήσεις του παίκτη που έχει σειρά. Αν ο βασιλιάς του παίκτη που έχει σειρά βρίσκεται υπό ΣΑΧ (**`if attackMovesTo`**) τότε αν δεν υπάρχουν επιτρεπτές κινήσεις, έχουμε περίπτωση ΜΑΤ. Αν ο παίκτης που έχει σειρά δεν βρίσκεται υπό απειλή αλλά ταυτόχρονα δεν έχει επιτρεπτές κινήσεις, έχουμε περίπτωση ΠΑΤ. Είναι φανερό ότι η περίπτωση θέσης ΜΑΤ, πρέπει να βαθμολογηθεί εξαιρετικά υψηλά. Κατά σύμβαση, ΜΑΤ στον λευκό θα βαθμολογηθεί κοντά στο -1000 ενώ ΜΑΤ στον μαύρο θα βαθμολογηθεί κοντά στο 1000. Η παράσταση

`return -(1000+vathos)*skaki[TX+2][0], []`

επιστρέφει ως αξία της θέσης ΜΑΤ **`-(1000+vathos) )*skaki[TX+2][0]`** και ως καλύτερη σχετική κίνηση μια κενή λίστα καθώς δεν υπάρχουν επιτρεπτές κινήσεις. Η προηγούμενη παράσταση, επειδή το **`skaki[TX+2][0]`** περιέχει είτε 1 είτε -1, αν παίζει ο λευκός μετατρέπεται σε **`-(1000+vathos)`**, ενώ αν παίζει ο μαύρος μετατρέπεται σε (**`1000+vathos`**). Πρέπει να προστεθεί και το vathos, διότι π.χ. μια θέση όπου το ΜΑΤ προκύπτει σε 2 κινήσεις, είναι προτιμότερη από μια θέση όπου το ΜΑΤ προκύπτει σε 3 κινήσεις και άρα πρέπει να έχει καλύτερη βαθμολογία. Αν δεν γίνει αυτό, σε θέσεις όπου το ΜΑΤ προκύπτει και με βαριάντα 2 κινήσεων και με βαριάντα 3 κινήσεων, ο υπολογιστής θα τις θεωρεί ισοδύναμες και ίσως επιλέξει το χειρότερο δρόμο.

Η περίπτωση ΠΑΤ αξιολογείται με 0 που αποτελεί τη μέση τιμή μεταξύ θετικών (υπέρ του λευκού) και αρνητικών (υπέρ του μαύρου) αξιολογήσεων.

Συνηθέστερα όμως, το υποπρόγραμμα xristina δεν θα αντιμετωπίζει περιπτώσεις ΜΑΤ και ΠΑΤ. Θα παρακάμπτεται μάλλον το σχετικό κομμάτι και τα αντίστοιχα return και δύο θα είναι οι βασικοί δρόμοι: είτε το vathos θα είναι 0, είτε το vathos θα είναι μεγαλύτερο από 0.

Αν η δεύτερη παράμετρος vathos είναι 0, τότε απλά θα καταμετρώνται οι λευκές και οι μαύρες δυνάμεις στη σκακιέρα και θα επιστρέφεται το καταμετρηθέν ισοζύγιο. Κατά σύμβαση, οι λευκοί πύργοι μετρούν για 5, οι λευκοί ίπποι για 3, οι λευκοί αξιωματικοί για 3, οι λευκές βασίλισσες για 9 και τα λευκά πιόνια για 1. Αντίστοιχα, οι μαύροι πύργοι μετρούν για -5, οι μαύροι ίπποι για -3, οι μαύροι αξιωματικοί για -3, οι μαύρες βασίλισσες για -9 και τα μαύρα πιόνια για -1. Οι βασιλιάδες δεν προσμετρώνται καθώς εξισορροπούν ο ένας τον άλλο. Έτσι λοιπόν π.χ. θέση με μια λευκή βασίλισσα απέναντι σε μαύρο πύργο και μαύρο ίππο, αξιολογείται με 1 (9-5-3). Αυτό ακριβώς γίνεται στο διπλό for όπου εξετάζοντας και τα 64 τετράγωνα της σκακιέρας συναθροίζονται στην μεταβλητή sum, μία μία οι αξίες όλων των κομματιών. Θετικό αποτέλεσμα σημαίνει υπεροχή του λευκού, ενώ αρνητικό αποτέλεσμα σημαίνει υπεροχή του μαύρου. Αν λοιπόν το vathos είναι 0, επιστρέφεται απλά η αξία της θέσης και μια κενή λίστα.

Αν η παράμετρος vathos δεν είναι 0, τότε προκειμένου να γίνει η αξιολόγηση, γίνεται χρήση της λεγόμενης αναδρομής. Την τεχνική της αναδρομής την εξετάσαμε ήδη στο κεφάλαιο 13 όπου το υποπρόγραμμα validMovesAndAttacks προκειμένου να υπολογίσει τις κινήσεις της βασίλισσας, κάλεσε δύο φορές τον εαυτό του με αλλαγμένες παραμέτρους, ώστε να επιστρέψει τις κινήσεις του πύργου και του αξιωματικού. Εδώ τα πράγματα γίνονται πιο δύσκολα. Προκειμένου να αξιολογηθεί μια θέση σε βάθος π.χ. 3, δημιουργούνται όλες οι προκύπτουσες θέσεις μετά από μια κίνηση, οι οποίες αξιολογούνται με τη σειρά τους σε βάθος 2, και γι’ αυτό το σκοπό πρέπει να γίνει χρήση της τεχνικής της αναδρομής. Οι επιστρεφόμενες αξιολογήσεις όλων των θέσεων (ax) θα σχηματίσουν την καλύτερη αξιολόγηση καθώς θα συγκριθούν μεταξύ τους: Την maxAxia αν παίζει ο λευκός ή την minAxia αν παίζει ο μαύρος. Οι κινήσεις (mv) που πετυχαίνουν τις βέλτιστες αξιολογήσεις, σχηματίζουν την λίστα bestMoves. Από αυτήν τη λίστα, επιστρέφεται μια τυχαία κίνηση με την **`random.choice`**.

Ίσως αντιλαμβάνεται κανείς ότι αν γίνει κλήση με vathos 3, τότε θα πρέπει να γίνουν αρκετές κλήσεις με vathos 2, εντός αυτών πρέπει να γίνουν ακόμα περισσότερες κλήσεις με vathos 1, και εντός των τελευταίων αυτών κλήσεων για κάθε μια τους θα γίνουν κι άλλες κλήσεις με vathos 0. Στη συνέχεια, καθώς θα ολοκληρώνονται οι κλήσεις με vathos 0, θα ολοκληρώνονται και οι αντίστοιχες κλήσεις με vathos 1, στη συνέχεια θα ολοκληρώνονται οι κλήσεις με vathos 2 και τελικά θα ολοκληρωθεί και η αρχική κλήση με vathos 3. Αυτή θα επιστρέψει όχι μόνο την αξιολόγηση της θέσης, που βγήκε ως βέλτιστη των θέσεων βάθους 2, αλλά και την κίνηση με την οποία προέκυψε αυτή η βέλτιστη θέση βάθους 2.

Κάθε φορά, πριν γίνει μια αναδρομική κλήση πρέπει βεβαίως να παιχθεί μια κίνηση εικονικά, άρα να ενημερωθεί πλήρως με τα δεδομένα αυτής της κίνησης μια δεύτερη δομή του τύπου skaki, και αυτόν τον ρόλο τον παίζει η δομή sk. Σε αυτήν τη δομή, γίνονται όλες οι απαραίτητες αλλαγές προκειμένου να απεικονιστεί η επόμενη κίνηση, mv στην περίπτωσή μας. Έχουμε ξαναδεί σχετικές διεργασίες π.χ. στο υποπρόγραμμα attackMovesTo και δεν θα ξαναεξηγηθούν εδώ.

Ξανατονίζεται ότι τα παραπάνω αποτελούν το πιο δυσνόητο τμήμα του προγράμματος και δεν πρέπει να περιμένει κανείς να το κατανοήσει με την πρώτη, ούτε με τη δεύτερη ανάγνωση και άρα δεν πρέπει να απελπίσει κανέναν η τυχούσα δυσκολία του. Η τεχνική της αναδρομής όμως αποτελεί απαραίτητη προσθήκη του βιβλίου, προκειμένου αυτό να προσπαθήσει να επιτελέσει ακόμα και τους πιο δύσκολους στόχους που έθεσε.

έγχρωμο βιβλίο

ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ ΤΟΥ ΒΙΒΛΙΟΥ:

- **ΣΤΑΔΙΑΚΗ ΠΡΟΣΕΓΓΙΣΗ**
- **ΕΠΕΞΗΓΗΣΗ ΟΛΩΝ ΤΩΝ ΣΗΜΕΙΩΝ**
- **ΟΠΤΙΚΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ**
- **ΔΙΑΘΕΣΙΜΟΣ ΚΩΔΙΚΑΣ**
- **ΕΚΜΑΘΗΣΗ ΤΗΣ ΓΛΩΣΣΑΣ PYTHON**
- **ΣΥΝΙΣΤΑΤΑΙ ΚΑΙ ΓΙΑ ΑΡΧΑΡΙΟΥΣ**

«Με το πρωτότυπο αυτό βιβλίο "ΠΡΟΓΡΑΜΜΑΤΙΖΟΝΤΑΣ ΕΝΑ ΣΚΑΚΙ" ο Γιώργος Ψαράκης παρουσιάζει με πολύ εύληπτο και εξαιρετικό τρόπο ένα ενδιαφέρον σκακιστικό ζήτημα».

Γιώργος Μακρόπουλος
πρόεδρος Ελληνικής Σκακιστικής Ομοσπονδίας ΕΣΟ
αναπλ. πρόεδρος Παγκόσμιας Σκακιστικής Ομοσπονδίας FIDE

e-book

ISBN 9789608891029