Τρίλιζα

Στο κεφάλαιο αυτό θα υλοποιήσουμε το γνωστό παιχνίδι της τρίλιζας. Θα αναπτύξουμε τον απαραίτητο κώδικα για ένα παιχνίδι δύο παικτών και μετά θα προσθέσουμε τις απαραίτητες πινελιές ώστε τον ρόλο του ενός παίκτη να τον αναλαμβάνει το ίδιο το πρόγραμμα. Προγραμματιστικά, θα εξασκηθούμε σε όλες τις έννοιες που έχουμε ήδη συναντήσει και θα δούμε πως μπορούμε να χρησιμοποιήσουμε *λίστες* και *πλειάδες* για την αναπαράσταση των δεδομένων μας. Η τρόπος που επιλέγουμε να αναπαραστήσουμε τα δεδομένα είναι αλληλένδετος με τον τρόπο που τα επεξεργαζόμαστε και επηρρεάζει άμεσα τη διαδικασία επίλυσης ενός προβλήματος.

9 Νοεμβρίου 2016
14:43

Έννοιες: υποπρογράμματα, αναπαραστάσεις, λίστες, πλειάδες

Το 1952, ο βρετανός Sandy Douglas έγραψε, στα πλαίσια του διδακτορικού του, ένα πρόγραμμα για τον υπολογιστή EDSAC που έπαιζε τρίλιζα. Αυτό το πρόγραμμα θεωρείται ένα από τα πρώτα βιντεοπαιχνίδια. Ο πίνακας της τρίλιζας προβαλλόνταν σε μια πρωτόγονη οθόνη και ο χρήστης επέλεγε την κίνησή του χρησιμοποιώντας το περιστρεφόμενο καντράν ενός τηλεφώνου.

Ένα τέτοιο πρόγραμμα για την τρίλιζα δεν είναι ιδιαίτερα δύσκολο να γραφτεί, η τρίλιζα είναι απλό παιχνίδι. Μάλιστα θα μπορούσε να χαρακτηριστεί ακόμα και βαρετό: όσο καλά και να παίξει κανείς, δεν μπορεί να κερδίσει τον αντίπαλό του παρά μόνο αν κάνει λάθη. Το «φυσιολογικό» αποτέλεσμα είναι η ισοπαλία. Κι όμως, οι άνθρωποι παίζουν τρίλιζα μ' ενθουσιασμό. Μάλιστα, όσο μεγαλύτερος είναι ο ενθουσιασμός, τόσο ευκολότερα χάνουν...

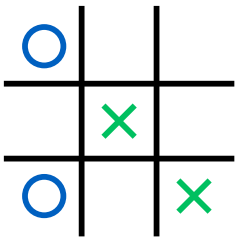
Αναπαραστάσεις

Πριν ξεκινήσουμε να γράψουμε έστω και μια γραμμή του προγράμματος, πρέπει να πάρουμε μια σημαντική απόφαση: ποια *αναπαράσταση* θα χρησιμοποιήσουμε για την τρίλιζα; Δηλαδή ποιος θα είναι ο τρόπος με τον οποίο το πρόγραμμά μας θα αποθηκεύει και θα διαχειρίζεται *εσωτερικά* τα περιεχόμενα των εννέα τετραγώνων του παιχνιδιού; Η απόφαση αυτή είναι κομβική και θα επηρρεάσει σημαντικά τη μορφή του προγράμματος που θα γράψουμε στη συνέχεια.

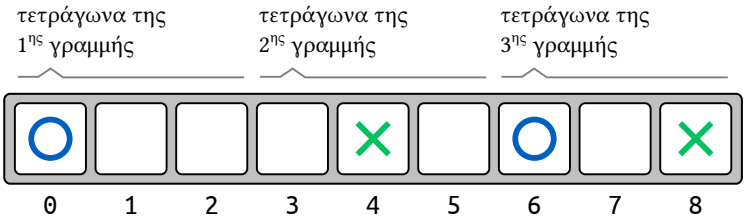
Δεν είναι βολικό ν' αποθηκεύσουμε το περιεχόμενο των εννέα τετραγώνων σε εννέα ξεχωριστές μεταβλητές. Θα καταλήξουμε με ένα δυσνόητο, εκτεταμένο πρόγραμμα με επαναλαμβανόμενα τμήματα. Αντιθέτως, θα θέλαμε τα εννέα τετράγωνα να αναπαρασταθούν ως μια *ενιαία, οργανωμένη συλλογή δεδομένων.*

Στο δικό μας πρόγραμμα θα χρησιμοποιήσουμε για τον σκοπό αυτό μια *λίστα*, δηλαδή μια συλλογή στοιχείων που είναι οργανωμένα *σειριακά*, το ένα μετά το άλλο. Σε μια λίστα, τα στοιχεία είναι *αριθμημένα* για να μπορούμε μέσω αυτής της αρίθμησης να έχουμε πρόσβαση στο περιεχόμενό τους.

Ας εξετάσουμε ένα παράδειγμα, ένα συγκεκριμένο στιγμιότυπο του πίνακα της τρίλιζας:



Η σχετική λίστα θα αποτελείται από εννέα στοιχεία, σε κάθε ένα από τα οποία θα αποθηκεύεται το περιεχόμενο του αντίστοιχου τετραγώνου της τρίλιζας. Μια απεικόνιση της λίστας φαίνεται στο σχήμα 8.1 που ακολουθεί.



Σχήμα 8.1: Παράδειγμα αναπαράστασης ενός συγκεκριμένου στιγμιοτύπου του παιχνιδιού με τη χρήση *λίστας*. Σε κάθε στοιχείο της λίστας αποθηκεύεται το περιεχόμενο ενός τετραγώνου της τρίλιζας. Η συγκεκριμένη αντιστοιχία μεταξύ τετραγώνων και στοιχείων της λίστας είναι απλά μία από τις πολλές που θα μπορούσαν να χρησιμοποιηθούν.

Στην Python, όπως και σε πολλές άλλες γλώσσες, κάθε είδους αρίθμηση ξεκινά από το `0`, όχι από το `1`. Έτσι, το πρώτο στοιχείο αυτής της λίστας που αναπαριστά τον πίνακα του παιχνιδιού βρίσκεται στη θέση `0` και το ένατο στη θέση `8`.

Γενικά, τα στοιχεία μιας λίστας μπορεί να είναι *ο,τιδήποτε*, ακόμα και άλλες λίστες.

Στη συγκεκριμένη περίπτωση, όλα τα στοιχεία της λίστας θα είναι αλφαριθμητικά και κάθε ένα από αυτά θα έχει τιμή είτε `"X"`, είτε `"O"`, είτε `" "` (κενό).

Η αναπαράσταση αυτή δεν είναι η μοναδική, υπάρχουν πολλές εναλλακτικές. Όμως δεν είναι εύκολο να γνωρίζουμε εκ των προτέρων ποια από τις πιθανές αναπαραστάσεις θ' αποδειχθεί βολικότερη. Ουσιαστικά, η επιλογή της κατάλληλης αναπαράστασης εξαρτάται από τον συγκεκριμένο τρόπο που το πρόγραμμά μας θα επεξεργάζεται τα δεδομένα. Εφόσον ακόμα δεν έχουμε γράψει το πρόγραμμα, μόνο η εμπειρία και η διαίσθηση μπορεί να μας καθοδηγήσει.

Κάτι Να "Μεταφράζει"

Εντάξει, επιλέξαμε την αναπαράσταση. Αλλά, όπως και να προχωρήσουμε, ο χρήστης θα πρέπει να βλέπει μια τρίλιζα για να παίζει, όχι εννέα τετράγωνα στη σειρά.

Ο τρόπος που το πρόγραμμα διαχειρίζεται "εσωτερικά" την αναπαράσταση της τρίλιζας δεν ενδιαφέρει το χρήστη. Στην οθόνη θα πρέπει να εμφανίζεται μια αναπαράσταση που να είναι γνώριμη σε εκείνους που παίζουν, δηλαδή ο κλασικός 3×3 πίνακας του παιχνιδιού.

Θα ξεκινήσουμε λοιπόν κατασκευάζοντας ένα υποπρόγραμμα που δέχεται σαν παράμετρο την εσωτερική αναπαράσταση `board` της τρίλιζας (εδώ πρόκειται για μια λίστα με εννέα στοιχεία) και την εμφανίζει στην οθόνη με τρόπο φιλικό προς το χρήστη.

```
def print3x3(board):
    """ Εμφανίζει σε διάταξη 3x3 τα περιεχόμενα
        μιας λίστας με 9 (τουλάχιστον) στοιχεία.
        board: Λίστα που θα εμφανιστεί σε διάταξη 3x3
    """
    print(" ", board[0], "|", board[1], "|", board[2])
    print(" ---+---+---")
    print(" ", board[3], "|", board[4], "|", board[5])
    print(" ---+---+---")
    print(" ", board[6], "|", board[7], "|", board[8])
```

Μπορούμε ν' αναφερθούμε στα στοιχεία μιας λίστας με βάση τη *θέση* τους σε αυτή. Η αρίθμηση των θέσεων ξεκινά από το `0`. Έτσι, εδώ το πρώτο στοιχείο της λίστας `board` είναι το `board[0]`, το δεύτερο είναι το `board[1]`, κ.ο.κ.

Αυτή η αντιστοιχία ανάμεσα στα τετράγωνα και τις θέσεις της λίστας, όπως εξάλλου και η ίδια η αναπαράσταση με λίστα, αφορά μόνο εμάς ως προγραμματιστές και ο χρήστης ούτε τη γνωρίζει, ούτε τον αφορά. Όταν το πρόγραμμά μας θα καλεί την `print3x3`, ο χρήστης θα βλέπει απλά στην οθόνη τα τετράγωνα της τρίλιζας.

Quick and Dirty

Για αρχή, νομίζω ότι μπορώ να φτιάξω κάτι στα γρήγορα που να λειτουργεί. Δεν θα είναι τέλειο και μάλλον στην πορεία θ' αλλάξω αρκετά πράγματα, όμως τουλάχιστον θα διαπιστώσω ποια σημεία παρουσιάζουν δυσκολίες.

Αρχικές τιμές

Στο κύριο πρόγραμμα, η λίστα στην οποία θ' αποθηκεύουμε την τρέχουσα κατάσταση της τρίλιζας θα ονομάζεται `board`. Αρχικά, πριν ξεκινήσει το παιχνίδι, η τρίλιζα θα περιέχει εννέα κενά τετράγωνα.

```
# η αναπαράσταση της τρίλιζας:
# μια λίστα με 9 χαρακτήρες (" ", "X" ή "O")
# αρχικά, όλα τα τετράγωνα είναι κενά
board = 9 * [" "]
```

Όταν "πολλαπλασιάζουμε" μια λίστα με τον τελεστή `*` κατασκευάζουμε μια νέα λίστα που περιέχει πολλές φορές τα στοιχεία της αρχικής.

Εδώ, η λίστα `board` προκύπτει πολλαπλασιάζοντας επί `9` τη λίστα `[" "]`, που αποτελείται από ένα στοιχείο.

Όπως και στο Παιχνίδι της Αφαίρεσης (κεφ. 4), που ήταν επίσης παιχνίδι δύο παικτών, θα χρησιμοποιήσουμε μια μεταβλητή `player`, η οποία σε κάθε γύρο θα παίρνει *εναλλάξ* την τιμή `"X"` ή `"O"`, υποδεικνύοντας με αυτόν τον τρόπο ποιος παίκτης έχει σειρά να παίξει σε κάθε γύρο.

Κατά σύμβαση, πρώτος παίζει ο παίκτης με τα X, οπότε πριν ξεκινήσει το παιχνίδι η `player` θα πάρει την αντίστοιχη αρχική τιμή.

```
15 # ο παίκτης που θα παίξει πρώτος
16 player = "X"
```

Σχήμα 8.2: Η τιμή της μεταβλητής `player` θα εναλλάσσεται σε κάθε γύρο μεταξύ του `"X"` και του `"O"`, υποδεικνύοντας το σύμβολο του παίκτη που έχει σειρά να παίξει. Η αρχική της τιμή είναι το `"X"`.

Επανάληψη: η συνθήκη συνέχειας

Ας περάσουμε τώρα στην επαναληπτική δομή. Κάθε κύκλος της επανάληψης αντιστοιχεί στη συμπλήρωση ενός τετραγώνου από κάποιο παίκτη. Για να ολοκληρωθεί το παιχνίδι και να *τερματιστεί* η επανάληψη θα πρέπει είτε να συμπληρωθούν και τα εννέα τετράγωνα, είτε κάποιος από τους δύο παίκτες να κάνει τρίλιζα. Για να διατυπωθεί λοιπόν η συνθήκη της επανάληψης χρειάζεται το πρόγραμμα να καταμετρά το πλήθος των κενών τετραγώνων και να καταγράφει αν έχει γίνει τρίλιζα.

Για την καταμέτρηση των κενών τετραγώνων θα χρησιμοποιήσουμε τη μεταβλητή `blank`. Όταν ξεκινά το παιχνίδι, όλα τα τετράγωνα είναι κενά.

```
17 # πλήθος τετραγώνων που απομένουν κενά
18 blank = 9
```

Για να καταγράφει το πρόγραμμά μας αν έχει γίνει τρίλιζα ή όχι θα χρησιμοποιήσουμε μια λογική μεταβλητή `inarow`. Γνωρίζουμε βέβαια πως όταν ξεκινά το παιχνίδι, κανείς από τους δύο παίκτες δεν έχει κάνει τρίλιζα κι έτσι η αρχική τιμή της `inarow` θα πρέπει να είναι `False`.

```
19 # λογική μεταβλητή που δείχνει αν έχει γίνει τρίλιζα
20 inarow = False
```

Είμαστε τώρα έτοιμοι να διατυπώσουμε τη συνθήκη της επανάληψης: για να *συνεχιστεί* η επανάληψη θα πρέπει να απομένει τουλάχιστον ένα κενό τετράγωνο και ταυτόχρονα κανένας από τους δύο παίκτες να μην έχει κάνει τρίλιζα.

```
21 # επανάληψη: συνεχίζεται όσο υπάρχουν κενά τετράγωνα
22 #   και δεν έχει γίνει τρίλιζα
23 while blank > 0 and not inarow:
```

Επανάληψη: επιλογή κίνησης από τον παίκτη

Μέσα στην επανάληψη, στην αρχή κάθε νέου κύκλου, το πρόγραμμα θα εμφανίζει τον πίνακα της τρίλιζας στον παίκτη που έχει σειρά να παίξει και θα τον ρωτάει σε ποιο τετράγωνο επιθυμεί να παίξει.

```
    # εμφάνιση πίνακα τρίλιζας
    print3x3(board)
    # επιλογή θέσης από τον παίκτη
    print(player, "διάλεξε τετράγωνο:", end=" ")
    position = int(input())
```

Εδώ υπάρχει ένα πολύ λεπτό σημείο. Ζητάμε από το χρήστη να προσδιορίσει μ' έναν ακέραιο αριθμό το τετράγωνο στο οποίο θα παίξει. Υπονοείται λοιπόν ότι υπάρχει μια *αρίθμηση* των τετραγώνων της τρίλιζας, με βάση την οποία θα κάνει την επιλογή του ο παίκτης. Έχει μεγάλη σημασία ότι η αυτή η αρίθμηση των τετραγώνων αφορά την οπτική γωνία *του παίκτη*, τον τρόπο με τον οποίο θα προσδιορίσει το τετράγωνο που τον ενδιαφέρει και *δεν έχει απαραίτητα σχέση* με την εσωτερική αναπαράσταση της τρίλιζας.

Εμείς όμως προς το παρόν θα επιλέξουμε την απλούστερη δυνατή αρίθμηση, η οποία *ταυτίζεται* με την αρίθμηση της εσωτερικής αναπαράστασης και φαίνεται στο σχήμα 8.3. Ας έχουμε όμως υπόψη ότι αυτή η αρίθμηση είναι βολική για εμάς *ως προγραμματιστές* και δεν είναι απαραίτητα η βολικότερη αρίθμηση *για το χρήστη*.

0	1	2
3	4	5
6	7	8

Σχήμα 8.3: Η αρίθμηση των τετραγώνων της τρίλιζας, από την σκοπιά του παίκτη. Με βάση αυτή την αρίθμηση επιλέγει ο παίκτης το τετράγωνο στο οποίο θα παίξει κάθε φορά. Θα μπορούσαμε να έχουμε επιλέξει μια διαφορετική αρίθμηση, αυτή όμως είναι προς το παρόν η βολικότερη γιατί ταυτίζεται με την εσωτερική μας αναπαράσταση.

Όταν ο χρήστης επιλέξει αριθμό τετραγώνου, το πρόγραμμα θα συμπληρώνει την αναπαράσταση του πίνακα της τρίλιζας με το σύμβολο του παίκτη, ενώ παράλληλα θα μειώνει το πλήθος των κενών τετραγώνων.

```
    # ανακοίνωση κίνησης
    print("Ο παίκτης", player, "παίζει στο", position)
    # συμπλήρωση επιλεγμένης θέσης
    board[position] = player
    # μείωση κενών τετραγώνων
    blank = blank - 1
```

Χρησιμοποιώντας μια εντολή όπως η `board[position] = player` *τροποποιείται* το στοιχείο της λίστας `board` που βρίσκεται στη θέση `position`, καθώς μια νέα τιμή αντικαθιστά την προηγούμενη. Είναι ένα βασικό χαρακτηριστικό των λιστών ότι τα στοιχεία τους μπορούν να τροποποιηθούν.

Επανάληψη: έλεγχος για τρίλιζα

Μετά την κίνηση του παίκτη, το πρόγραμμά μας θα πρέπει να ελέγχει μήπως έγινε τρίλιζα. Προς το παρόν, θα υλοποιήσουμε αυτόν τον έλεγχο με τον πιο απλό τρόπο που μπορούμε να σκεφτούμε, εξετάζοντας όλες τις πιθανές τριάδες τετραγώνων. Στη συνέχεια θα έχουμε την ευκαιρία να διαπιστώσουμε αν υπάρχουν περιθώρια βελτίωσης.

```
    # έλεγχος για τρίλιζα:
    # για κάθε 3άδα θέσεων στις πιθανές τρίλιζες
    # ελέγχεται αν ο παίκτης έχει καταλάβει 3 θέσεις
    if board[0] == board[1] == board[2] == player:
        inarow = True
    elif board[3] == board[4] == board[5] == player:
        inarow = True
    elif board[6] == board[7] == board[8] == player:
        inarow = True
```

Μέχρι στιγμής έχουμε δει πως οι συγκριτικοί τελεστές χρησιμοποιούνται για να συγκρίνουν δύο τιμές μεταξύ τους. Στην Python μπορούν να χρησιμοποιηθούν για να συγκριθούν *πολλές* τιμές μεταξύ τους, όπως σε αυτό το παράδειγμα με τον τελεστή `==`. Αυτό είναι ένα χαρακτηριστικό που δεν το συναντάμε συχνά σε άλλες γλώσσες προγραμματισμού, με τους αντίστοιχους συγκριτικούς τελεστές.

```
44      elif board[0] == board[3] == board[6] == player:
45          inarow = True
46      elif board[1] == board[4] == board[7] == player:
47          inarow = True
48      elif board[2] == board[5] == board[8] == player:
49          inarow = True
50      elif board[0] == board[4] == board[8] == player:
51          inarow = True
52      elif board[2] == board[4] == board[6] == player:
53          inarow = True
```

Κάθε μία από τις οκτώ περιπτώσεις σε αυτή τη δομή επιλογής αντιστοιχεί σε έναν από τους οκτώ διαφορετικούς τρόπους να γίνει τρίλιζα. Και στις οκτώ περιπτώσεις εκτελείται η ίδια εντολή: η μεταβλητή `inarow` παίρνει την τιμή **`True`**, για να καταγραφεί πλέον από το πρόγραμμα ότι έχει γίνει τρίλιζα (εναλλακτικά, θα μπορούσε να χρησιμοποιηθεί μια *σύζευξη* των οκτώ επιμέρους συνθηκών). Παρατηρήστε ότι στη δομή επιλογής δεν υπάρχει **`else`** επειδή δε χρειάζεται να εκτελεστεί κάποια εντολή σε περίπτωση που δε γίνει τρίλιζα.

Επανάληψη: εναλλαγή παίκτη

Μέσα στην επανάληψη, απομένει μόνο να εναλλάσσουμε την τιμή της μεταβλητής `player`.

```
54      # εναλλαγή παίκτη
55      if player == "X":
56          player = "O"
57      else:
58          player = "X"
```

Ανακοίνωση αποτελέσματος

Όταν η επανάληψη τερματιστεί, το παιχνίδι θα έχει τελειώσει και το πρόγραμμα θα πρέπει να εμφανίσει στους παίκτες ποιο ήταν το αποτέλεσμα. Όμως υπάρχουν δύο πιθανοί λόγοι για να τελειώσει το παιχνίδι: να έχει συμπληρωθεί ο πίνακας του παιχνιδιού ή να έχει γίνει τρίλιζα (χωρίς το ένα να αποκλείει το άλλο). Αν έχει γίνει τρίλιζα, θα πρέπει να εμφανίζεται κατάλληλο μήνυμα, ενώ σε διαφορετική περίπτωση θα πρέπει να εμφανίζεται μήνυμα ότι το παιχνίδι έληξε ισόπαλο.

```
59  # εμφάνιση τελικού πίνακα τρίλιζας
60  print3x3(board)
61  # εμφάνιση αποτελέσματος
62  if inarow:
63      print("Τρίλιζα!")
64  else:
65      print("Ισοπαλία.")
```

oxo/src/oxo.1.py

Είναι *υποχρεωτικό* το πρόγραμμα να ελέγχει την τιμή της `inarow` και όχι της `blank` για να διαπιστώσει το αποτέλεσμα του παιχνιδιού. Η τιμή της `blank` *δεν* μπορεί να μας οδηγήσει σε ασφαλή συμπεράσματα. Αν μετά το τέλος του παιχνιδιού ο πίνακας είναι πλήρως συμπληρωμένος και ισχύει ότι η μεταβλητή `blank` έχει την τιμή `0`, δεν μπορούμε να συμπεράνουμε ότι το παιχνίδι έληξε ισόπαλο, αφού μπορεί η τελευταία κίνηση να συμπλήρωσε μεν τον πίνακα αλλά *και* να οδήγησε σε τρίλιζα.

Συμμάζεμα

Ξέρω ότι μπορώ να "σπάσω" το πρόγραμμά μου σε μικρότερα τμήματα, υλοποιώντας τις επιμέρους λειτουργίες του προγράμματος ως ξεχωριστά υποπρογράμματα. Από που να ξεκινήσω;

Είναι σημαντικό να διακρίνουμε (σε οποιοδήποτε πρόγραμμα) τις ομάδες εντολών που λειτουργούν ως ενιαίο σύνολο και υλοποιούν μια συγκεκριμένη λειτουργία. Αυτά τα τμήματα του προγράμματος θα αποτελέσουν τη βάση για την κατασκευή των επιμέρους υποπρογραμμάτων.

Ο τρόπος με τον οποίο μπορούμε να διαιρέσουμε ένα ενιαίο πρόγραμμα σε επιμέρους τμήματα με βάση τη λειτουργία τους δεν είναι μοναδικός. Στην πραγματικότητα υπάρχουν πολλές εναλλακτικές. Επίσης, οι εμπειρότεροι προγραμματιστές συνήθως *σχεδιάζουν* τα προγράμματά τους και τα υποπρογράμματα από τα οποία αποτελούνται *εκ των προτέρων*, χωρίς αυτό να σημαίνει ότι στη συνέχεια, καθώς υλοποιούν το πρόγραμμά τους, δεν μπορούν να αναθεωρήσουν ή να εκλεπτύνουν τον αρχικό τους σχεδιασμό.

Αλληλεπίδραση με το χρήστη

Διατρέχοντας το πρόγραμμα που έχουμε αναπτύξει μέχρι στιγμής, εντοπίζουμε μέσα στην επαναληπτική δομή μια ομάδα εντολών που αναλαμβάνει την αλληλεπίδραση με τον παίκτη: εμφανίζει τα τετράγωνα της τρίλιζας και ρωτά τον παίκτη σε ποιο τετράγωνο επιθυμεί να παίξει.

```
    # εμφάνιση πίνακα τρίλιζας
    print3x3(board)
    # επιλογή θέσης από τον παίκτη
    print(player, "διάλεξε τετράγωνο:", end=" ")
    position = int(input())
```

Αυτή η ομάδα εντολών θ' αποτελέσει τη βάση για τη συνάρτηση `readPosition`, η οποία δέχεται σαν παραμέτρους τον παίκτη `player` που έχει σειρά να παίξει και την αναπαράσταση `board` της τρίλιζας και επιστρέφει τη θέση που επέλεξε ο παίκτης για την επόμενη κίνησή του.

```
def readPosition(player, board):
    """ Διαβάζει από τον παίκτη τη θέση
        στην οποία επιθυμεί να παίξει και την
        επιστρέφει.
        player: Ο παίκτης που έχει σειρά να παίξει
        board: Μια λίστα με 9 στοιχεία (η τρίλιζα)
    """
    # εμφάνιση πίνακα τρίλιζας
    print3x3(board)
    # επιλογή θέσης από τον παίκτη
    print(player, "διάλεξε τετράγωνο:", end=" ")
    position = int(input())
    # επιστροφή επιλεγμένης θέσης
    return position
```

Πραγματοποίηση κίνησης στο επιλεγμένο τετράγωνο

Αμέσως μετά τις εντολές που ζητούν από τον παίκτη να επιλέξει το τετράγωνο στο οποίο θα παίξει, υπάρχει μια ομάδα εντολών που πραγματοποιούν την κίνηση του παίκτη.

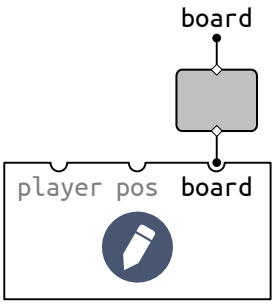
```
    # ανακοίνωση κίνησης
    print("Ο παίκτης", player, "παίζει στο", position)
    # συμπλήρωση επιλεγμένης θέσης
    board[position] = player
```

Αυτή η ομάδα εντολών θ' αποτελέσει τη βάση για τη συνάρτηση `play`, η οποία δέχεται σαν παραμέτρους τον παίκτη `player` που επέλεξε θέση, την αναπαράσταση `board` της τρίλιζας και τη θέση που επέλεξε ο παίκτης και πραγματοποιεί την κίνησή του.

```
def play(player, pos, board):
    """ Πραγματοποιεί και ανακοινώνει την κίνηση
        ενός παίκτη σε συγκεκριμένη θέση
        player: Το σύμβολο του παίκτη
        pos: Η θέση στην οποία παίζει ο παίκτης
        board: Μια λίστα με 9 στοιχεία (η τρίλιζα)
    """
    # ανακοίνωση κίνησης
    print("Ο παίκτης", player, "παίζει στο", pos)
    # συμπλήρωση επιλεγμένης θέσης
    board[pos] = player
```

Η συνάρτηση `play` δεν επιστρέφει κάποια τιμή, όμως *μεταβάλλει την αναπαράσταση της τρίλιζας* (σχήμα 8.4).

Σε αυτή, αλλά και σε όλες τις συναρτήσεις που ακολουθούν, ονομάσαμε `player` την παράμετρο που αντιστοιχεί στο σύμβολο του παίκτη και `board` την παράμετρο που αντιστοιχεί στην αναπαράσταση της τρίλιζας. Χρησιμοποιήσαμε δηλαδή τα ίδια ονόματα που χρησιμοποιούνται και στο κύριο πρόγραμμα. Αυτό δεν είναι υποχρεωτικό, θα μπορούσαμε να χρησιμοποιήσουμε άλλα ονόματα για τις παραμέτρους, απλά θεωρούμε ότι έτσι το πρόγραμμα είναι πιο κατανοητό. Πρέπει όμως να γίνει σαφές ότι *δεν πρόκειται για τις ίδιες μεταβλητές*: η `player` και η `board` του κύριου προγράμματος δεν ταυτίζονται με τις *τοπικές* `player` και `board` μέσα σε μια συνάρτηση. Μάλιστα οι τοπικές μεταβλητές των συναρτήσεων δημιουργούνται μόνο όταν καλείται η συνάρτηση και παύουν να υφίστανται όταν ολοκληρωθεί η εκτέλεση των εντολών της.



Σχήμα 8.4: Όταν κληθεί η συνάρτηση `play` από το κύριο πρόγραμμα, θα χρησιμοποιηθεί σαν παράμετρος η λίστα `board` του κύριου προγράμματος. Η τοπική μεταβλητή `board` της συνάρτησης `play` και η `board` του κύριου προγράμματος είναι δύο διαφορετικές μεταβλητές. Ωστόσο, είναι στην πραγματικότητα δύο διαφορετικά ονόματα για το ίδιο πράγμα: τη λίστα που αποτελεί την αναπαράσταση της τρίλιζας. Έτσι, οποιαδήποτε τροποποίηση γίνει εντός της συνάρτησης στα στοιχεία της λίστας `board`, αφορά ουσιαστικά και την λίστα `board` του κύριου προγράμματος.

Έλεγχος για τρίλιζα

Μια άλλη ομάδα εντολών που αποτελούν ενιαίο σύνολο απαρτίζεται από τη δομή επιλογής που ελέγχει τις πιθανές τριάδες τετραγώνων για να διαπιστωθεί αν ο παίκτης που έπαιξε έκανε τρίλιζα.

```
# για κάθε 3άδα θέσεων στις πιθανές τρίλιζες
# ελέγχεται αν ο παίκτης έχει καταλάβει 3 θέσεις
if board[0] == board[1] == board[2] == player:
    inarow = True
elif board[3] == board[4] == board[5] == player:
    inarow = True
...
elif board[2] == board[4] == board[6] == player:
    inarow = True
```

Αυτή η ομάδα εντολών θ' αποτελέσει τη βάση για τη συνάρτηση `check`, η οποία δέχεται σαν παραμέτρους τον παίκτη `player` που έχει σειρά να παίξει και την αναπαράσταση `board` της τρίλιζας κι επιστρέφει `True` ή `False` ανάλογα αν ο παίκτης έκανε τρίλιζα ή όχι.

```
def check(player, board):
    """ Ελέγχει αν υπάρχει 3άδα όπου ο παίκτης player
        έχει καταλάβει και τις 3 θέσεις.
        player: σύμβολο παίκτη ("X" ή "O")
        board: Μια λίστα με 9 στοιχεία (η τρίλιζα)
    """
    # για κάθε 3άδα θέσεων στις πιθανές τρίλιζες
    # αν ο παίκτης έχει καταλάβει και τις 3 θέσεις
    if board[0] == board[1] == board[2] == player:
        return True
    elif board[3] == board[4] == board[5] == player:
        return True
    elif board[6] == board[7] == board[8] == player:
        return True
    elif board[0] == board[3] == board[6] == player:
        return True
    elif board[1] == board[4] == board[7] == player:
        return True
    elif board[2] == board[5] == board[8] == player:
        return True
    elif board[0] == board[4] == board[8] == player:
        return True
    elif board[2] == board[4] == board[6] == player:
        return True
    else:
        # αν φτάσουμε μέχρι εδώ, ο έλεγχος απέτυχε
        return False
```

Εναλλαγή παίκτη

Τέλος, οι εντολές που φροντίζουν για την εναλλαγή του παίκτη στο τέλος κάθε κύκλου της επανάληψης αποτελούν επίσης ένα ενιαίο σύνολο εντολών.

```
    # εναλλαγή παίκτη
    if player == "X":
        player = "O"
    else:
        player = "X"
```

Αυτή η ομάδα εντολών θ' αποτελέσει τη βάση για τη συνάρτηση `next`, η οποία δέχεται σαν παράμετρο έναν παίκτη `player` και επιστρέφει τον παίκτη που έχει σειρά να παίξει μετά από αυτόν.

```
63 def next(player):
64     """ Επιστρέφει το σύμβολο του παίκτη που παίζει
65         μετά τον παίκτη με σύμβολο player.
66         player: σύμβολο παίκτη ("X" ή "O")
67     """
68     if player == "X":
69         return "O"
70     else:
71         return "X"
```

Το κύριο πρόγραμμα

Τώρα, το κύριο πρόγραμμα μπορεί να καλεί αυτές τις συναρτήσεις. Στο σημείο όπου το πρόγραμμα αλληλεπιδρά με το χρήστη για να τον ρωτήσει σε ποιο τετράγωνο επιθυμεί να παίξει, καλείται η συνάρτηση `readPosition`. Στο σημείο όπου πραγματοποιείται η κίνηση του παίκτη, καλείται η `play`. Στο σημείο όπου ελέγχεται αν η κίνηση του παίκτη οδηγεί σε τρίλιζα, καλείται η `check`. Τέλος, αμέσως μετά, στο σημείο όπου εναλλάσσεται ο παίκτης που έχει σειρά να παίξει, καλείται η `next`.

```
82 # επανάληψη: συνεχίζεται όσο υπάρχουν κενά τετράγωνα
83 #   και δεν έχει γίνει τρίλιζα
84 while blank > 0 and not inarow:
85     # επιλογή θέσης από τον παίκτη
86     position = readPosition(player, board)
87     # συμπλήρωση επιλεγμένης θέσης
88     play(player, position, board)
89     # μείωση κενών τετραγώνων
90     blank = blank - 1
91     # έλεγχος για τρίλιζα
92     inarow = check(player, board)
93     # εναλλαγή παίκτη
94     player = next(player)
```

Το Ζήτημα του Νικητή

Θα ήθελα το πρόγραμμα ν' ανακοινώνει ποιος παίκτης κέρδισε.

Σε περίπτωση που γίνει τρίλιζα, το πρόγραμμα εμφανίζει το μήνυμα `"Τρίλιζα!"`, χωρίς όμως ν' ανακοινώνει ποιος από τους δύο παίκτες κέρδισε. Προφανώς, πρόκειται για τον παίκτη που έπαιξε τελευταίος κι έτσι μια βιαστική λύση θα ήταν να εμφανίσουμε το νικητή τροποποιώντας τις εντολές εμφανίζουν το αποτέλεσμα ως εξής:

```
# εμφάνιση αποτελέσματος
if inarow:
    print("Τρίλιζα! Κέρδισε ο", player)
else:
    print("Ισοπαλία.")
```

Εδώ όμως υπάρχει πρόβλημα. Η μεταβλητή `player` εναλλάσσεται *στο τέλος* κάθε κύκλου της επανάληψης, ακόμα κι όταν κάποιος παίκτης κάνει τρίλιζα. Έτσι, μετά τον τερματισμό της επανάληψης, η `player` δεν αντιστοιχεί στον παίκτη που έπαιξε τελευταίος (και κέρδισε), αλλά στον επόμενο (που είναι ο ηττημένος). Επομένως, αν το πρόγραμμα απλά εμφανίσει την τιμή της `player`, θα εμφανίσει τον παίκτη που *έχασε*, όχι εκείνον που έκανε τρίλιζα.

Υπάρχουν αρκετοί τρόποι να διορθώσουμε αυτή την συμπεριφορά. Εμείς θα υιοθετήσουμε μια λύση που απαιτεί τις λιγότερες αλλαγές στο πρόγραμμά μας: ο νικητής του παιχνιδιού είναι ο παίκτης που θα έπαιζε μετά τον ηττημένο.

```
97  # εμφάνιση αποτελέσματος
98  if inarow:
99      print("Τρίλιζα! Κέρδισε ο", next(player))
100 else:
101     print("Ισοπαλία.")
```

oxo/src/oxo.2.py

Το Ζήτημα του Ελέγχου

Όταν ο χρήστης επιλέγει το τετράγωνο στο οποίο θα παίξει, το πρόγραμμα του επιτρέπει να πληκτρολογήσει οποιονδήποτε αριθμό, χωρίς κανέναν έλεγχο αν αυτός βρίσκεται μεταξύ `0` και `8`. Δεν ελέγχεται καν αν το τετράγωνο που επιλέγει ο χρήστης είναι κατειλημμένο.

Η επιλογή τετραγώνου από τον παίκτη πραγματοποιείται μέσω της συνάρτησης `readPosition`. Αυτή θα πρέπει τώρα να *επεκταθεί*, έτσι ώστε η τιμή που πληκτρολογεί ο παίκτης να ελέγχεται και, σε περίπτωση που δεν είναι έγκυρη, να ζητείται νέα τιμή.

Μέσα στη `readPosition`, οι εντολές που ζητούν από τον παίκτη να επιλέξει το τετράγωνο στο οποίο θα παίξει θα τοποθετηθούν πλέον μέσα σε μια επαναληπτική δομή. Έτσι, ο παίκτης θα "εγκλωβίζεται" σε έναν κύκλο από τον οποίο βγαίνει μόνο όταν πληκτρολογήσει μια έγκυρη τιμή. Αν ο παίκτης επιλέξει θέση που δεν είναι ανάμεσα στο

`0` και το `8` ή επιλέξει κατειλημμένη θέση τότε εμφανίζεται μήνυμα λάθους και η ανάγνωση τιμής επαναλαμβάνεται.

```
18      # εμφάνιση πίνακα τρίλιζας
19      print3x3(board)
20      # επανάληψη: όσο το τετράγωνο δεν είναι έγκυρο
21      while True:
22          # επιλογή θέσης από τον παίκτη
23          print(player, "διάλεξε τετράγωνο:", end=" ")
24          position = int(input())
25          # έλεγχος εγκυρότητας
26          if position < 0 or position > 8:
27              print("Επίλεξε τιμή μεταξύ 0 και 8.")
28          elif board[position] != " ":
29              print("Το", position, "δεν είναι κενό.")
30          else:
31              # έγκυρη τιμή, τέλος επανάληψης
32              break
33      # επιστροφή επιλεγμένης θέσης
34      return position
```

oxo/src/oxo.3.py

Στο κύριο πρόγραμμα δεν απαιτείται καμία αλλαγή. Εκεί εξακολουθεί να καλείται η `readPosition`, από την οποία προκύπτει η θέση στην οποία θα τοποθετήσει το σύμβολό του ο παίκτης. Όμως τώρα η τιμή που θα επιστρέψει η τροποποιημένη `readPosition` θα είναι σίγουρα έγκυρη.

Το Ζήτημα της Τρίλιζας

Η υλοποίηση της συνάρτησης check *μου κάθεται στο λαιμό. Δεν γίνεται να γράψουμε τον έλεγχο αν έχει γίνει τρίλιζα με πιο κομψό, συμπαγή τρόπο;*

Αν μελετήσουμε τη μεγάλη **if** που βρίσκεται μέσα στη συνάρτηση check και ελέγχει τους οκτώ διαφορετικούς τρόπους να γίνει τρίλιζα, θα παρατηρήσουμε ότι οι οκτώ συνθήκες που ελέγχονται είναι ουσιαστικά ίδιες μεταξύ τους. Κάθε συνθήκη εξετάζει μια τριάδα θέσεων και το μόνο που αλλάζει από περίπτωση σε περίπτωση είναι *οι αριθμοί των τριών θέσεων που ελέγχονται.*

Ας καταγράψουμε λοιπόν στην αρχή της συνάρτησης check τις διαφορετικές τριάδες θέσεων που εξετάζονται.

```
46  def check(player, board):
47      """ Ελέγχει αν υπάρχει 3άδα όπου ο παίκτης player
48          έχει καταλάβει και τις 3 θέσεις.
49          player: σύμβολο παίκτη ("X" ή "O")
50          board: Μια λίστα με 9 στοιχεία (η τρίλιζα)
51      """
```

```
52      # οι 3άδες των θέσεων που σχηματίζουν τρίλιζες
53      triples = (
54          (0,1,2),(3,4,5),(6,7,8),    # οριζόντια
55          (0,3,6),(1,4,7),(2,5,8),    # κάθετα
56          (0,4,8),(2,4,6))            # διαγώνια
```

Τώρα μπορούμε *επαναληπτικά*, για κάθε μια από τις οκτώ τριάδες θέσεων, να ελέγξουμε αν ο παίκτης έχει καταλάβει ολόκληρη την τριάδα.

```
57      # για κάθε 3άδα θέσεων
58      for triple in triples:
59          # p1, p2 και p3 οι τρεις θέσεις της 3άδας
60          p1, p2, p3 = triple
61          # αν ο παίκτης έχει καταλάβει και τις 3 θέσεις
62          if board[p1] == board[p2] == board[p3] == player:
63              return True
64      # αν φτάσουμε μέχρι εδώ, ο έλεγχος απέτυχε
65      return False
```

oxo/src/oxo.4.py

Αν και η συνάρτηση check ουσιαστικά ξαναγράφτηκε από την αρχή, η *λειτουργία* της παρέμεινε η ίδια: ελέγχει αν έχει γίνει τρίλιζα. Γι' αυτό και δεν απαιτείται καμία τροποποίηση στο κύριο πρόγραμμα. Αυτό που άλλαξε είναι η *υλοποίηση* αυτής της λειτουργίας, δηλαδή ο τρόπος με τον οποίο ελέγχεται αν έχει γίνει τρίλιζα.

Μια δομή δεδομένων που μοιάζει πολύ με τις λίστες είναι οι *πλειάδες* (tuples). Τα περιεχόμενα των πλειάδων περικλείονται σε παρενθέσεις και, σε αντίθεση με τις λίστες, *δεν* μπορούν να μεταβληθούν. Η πρόσβαση στα στοιχεία των πλειάδων γίνεται ακριβώς με τον ίδιο τρόπο όπως και στις λίστες.

Εδώ θέλουμε ν' αποτυπώσουμε τις διαφορετικές τριάδες θέσεων που σχηματίζουν τρίλιζες. Επειδή αυτές οι τιμές δεν πρόκειται ν' αλλάξουν, χρησιμοποιείται για την αποθήκευσή τους μια πλειάδα, η `triples`.

Η `triples` είναι μια πλειάδα που περιέχει άλλες πλειάδες. Στην περίπτωση αυτή, αποτελείται από τριάδες με τους αριθμούς των θέσεων που σχηματίζουν τρίλιζες.

Η εντολή **for** είναι μια εντολή επανάληψης που διατρέχει τα στοιχεία μιας ακολουθίας τιμών, όπως μια λίστα ή μια πλειάδα, με τη σειρά με την οποία αυτά εμφανίζονται στην ακολουθία. Σε κάθε επανάληψη η τιμή του επόμενου στοιχείου της ακολουθίας ανατίθεται στη μεταβλητή απαρίθμησης που χρησιμοποιούμε στη **for**.

Εδώ η **for** χρησιμοποιείται για να διατρεχθούν όλες οι τριάδες της πλειάδας `triples`.

Με την εντολή `p1, p2, p3 = triple` οι τρεις τιμές που βρίσκονται αποθηκευμένες στην πλειάδα `triple` αποδίδονται στις μεταβλητές `p1`, `p2` και `p3` αντίστοιχα.

Αυτό ονομάζεται *ξεπακετάρισμα* της πλειάδας (tuple unpacking) και είναι ουσιαστικά ο μηχανισμός που χρησιμοποιείται και για την επιστροφή πολλαπλών τιμών από συναρτήσεις.

Κάτι Για Παρέα

Στην αρχή είπαμε ότι θα φτιάξουμε ένα πρόγραμμα που θα παίζει τρίλιζα. Προς το παρόν και οι δύο παίκτες που συμμετέχουν στο παιχνίδι είναι άνθρωποι.

Θα κάνουμε τις απαραίτητες επεκτάσεις, ώστε το πρόγραμμα ν' αναλάβει το ρόλο ενός από τους δύο παίκτες. Δεν είναι ανάγκη να προσπαθήσουμε εξαρχής να κάνουμε το πρόγραμμα να παίζει έξυπνα. Με αυτό θα ασχοληθούμε στο επόμενο βήμα· προς το παρόν θα του επιτρέψουμε να ξεκινάει πρώτο και να επιλέγει σε κάθε γύρο μια τυχαία κίνηση.

Αφού οι κινήσεις του προγράμματος θα είναι τυχαίες, χρειάζεται να εισαγάγουμε, στην αρχή του προγράμματος, την κατάλληλη βιβλιοθήκη.

```
1   import random
```

Οι διαθέσιμες θέσεις

Για να επιλέξει το πρόγραμμα σε ποια θέση θα παίξει, θα πρέπει πρώτα να υπολογίσει ποιες από τις εννέα θέσεις είναι διαθέσιμες.

Αυτό μπορεί να γίνει με τον κώδικα που ακολουθεί, ο οποίος διατρέχει τις εννέα θέσεις της τρίλιζας και κατασκευάζει μια λίστα με τις θέσεις εκείνες που είναι διαθέσιμες.

```
# η λίστα με τις διαθέσιμες θέσεις, αρχικά κενή
positions = []
# για κάθε θέση από 0 μέχρι και 8
for s in range(9):
    # αν η θέση s είναι διαθέσιμη
    if board[s] == " ":
        # προστίθεται στη λίστα διαθέσιμων θέσεων
        positions.append(s)
```

Η `range` χρησιμοποιείται για να αναφερθούμε σε ένα *διάστημα τιμών*. Η προαιρετική πρώτη παράμετρος ορίζει το αριστερό άκρο του διαστήματος. Αν παραληφθεί, το διάστημα ξεκινά από το *μηδέν*. Η δεύτερη παράμετρος ορίζει το δεξί άκρο, το οποίο είναι ανοικτό. Η προαιρετική τρίτη παράμετρος ορίζει το *βήμα*, δηλαδή ανά πόσες τιμές του διαστήματος θα διατρέχονται.

Η μέθοδος `append()` εφαρμόζεται σε μια λίστα και προσθέτει ένα νέο στοιχείο στο τέλος της.

Εδώ η `append()` καλείται επαναληπτικά, κατασκευάζοντας στοιχείο προς στοιχείο τη λίστα με τις διαθέσιμες θέσεις.

Είναι πολύ συνηθισμένο να κατασκευάζουμε λίστες με αυτόν ακριβώς τον τρόπο, ξεκινώντας από μια κενή λίστα στην οποία προσθέτουμε σταδιακά τις τιμές που ικανοποιούν ένα συγκεκριμένο κριτήριο. Είναι τόσο συνηθισμένο αυτό το μοτίβο, που υπάρχει κι ένας εναλλακτικός, πολύ συνοπτικότερος τρόπος να δημιουργηθεί η ίδια λίστα:

```
positions = [s for s in range(9) if board[s] == " "]
```

Ουσιαστικά εδώ *περιγράφουμε* από τι αποτελείται η `positions`, όπως θα κάναμε στα μαθηματικά: είναι μια λίστα από όλες τις θέσεις `s`, μεταξύ `0` και `8`, για τις οποίες ισχύει ότι το `board[s]` είναι κενό, δηλαδή η αντίστοιχη θέση της τρίλιζας είναι διαθέσιμη.

Αυτός ο συμβολισμός ονομάζεται *συγκέντρωση λίστας* (list comprehension). Είναι ένας εύληπτος τρόπος να δημιουργήσουμε μια λίστα περιγράφοντας τα στοιχεία της, από που αυτά προέρχονται και πως επιλέγονται.

Εδώ συγκεντρώνουμε σε μια λίστα όλα τα στοιχεία `s` που περιέχονται στο εύρος τιμών μεταξύ `0` και `8` και έχουν το χαρακτηριστικό ότι το `board[s]` έχει την τιμή `" "` (κενό).

Τώρα μπορούμε να ορίσουμε τη συνάρτηση `available`, η οποία δέχεται σαν παράμετρο την αναπαράσταση `board` της τρίλιζας και επιστρέφει *μια νέα λίστα* με τους αριθμούς των διαθέσιμων θέσεων.

```
76 def available(board):
77     """ Επιστρέφει μια λίστα με τις διαθέσιμες θέσεις
78         board: Μια λίστα με 9 στοιχεία (η τρίλιζα)
79     """
80     return [s for s in range(9) if board[s] == " "]
```

Τυχαία επιλογή θέσης

Έχοντας κατασκευάσει τη συνάρτηση `available`, η οποία επιστρέφει μια λίστα με τις διαθέσιμες θέσεις, είναι εύκολο να προχωρήσουμε και στον ορισμό της συνάρτησης `randomPosition`, η οποία δέχεται σαν παράμετρο την αναπαράσταση `board` της τρίλιζας και επιστρέφει μια τυχαία επιλεγμένη διαθέσιμη θέση.

```
81 def randomPosition(board):
82     """ Επιστρέφει μια τυχαία διαθέσιμη θέση
83         board: Μια λίστα με 9 στοιχεία (η τρίλιζα)
84     """
85     return random.choice(available(board))
```

Η συνάρτηση `choice()` της βιβλιοθήκης `random` δέχεται σαν παράμετρο μια λίστα κι επιστρέφει ένα τυχαία επιλεγμένο στοιχείο της.

Εδώ χρησιμοποιείται για την τυχαία επιλογή μιας θέσης, από τη λίστα των διαθέσιμων θέσεων.

Επεκτάσεις στο κύριο πρόγραμμα

Ως τώρα, έχουμε προσθέσει στο πρόγραμμα την απαραίτητη υποδομή ώστε να είναι δυνατή η επιλογή μιας τυχαίας θέσης. Ωστόσο, το κύριο πρόγραμμα εξακολουθεί να υλοποιεί ένα παιχνίδι δύο παικτών. Θα πρέπει λοιπόν να επεκτείνουμε το κύριο πρόγραμμα, ώστε ν' αναλαμβάνει πλέον το ρόλο ενός από τους δύο παίκτες.

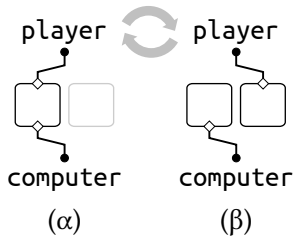
Θα χρησιμοποιήσουμε τη μεταβλητή `computer`, της οποίας η τιμή θ' αντιστοιχεί στο σύμβολο του παίκτη που θα παίζει αυτοματοποιημένα. Στην αρχή του παιχνιδιού η `computer` θα παίρνει την τιμή `"X"`, ώστε το πρόγραμμα να παίζει πρώτο.

```
92 # ο παίκτης που θα κατευθύνεται από το πρόγραμμα
93 computer = "X"
```

Θυμηθείτε ότι η μεταβλητή `player` εναλλάσσεται σε κάθε γύρο μεταξύ των τιμών `"X"` και `"O"`, υποδεικνύοντας ποιος παίκτης έχει σειρά να παίξει στον επόμενο γύρο. Συκρίνοντας λοιπόν την `player` με την `computer` ελέγχουμε ουσιαστικά αν ο επόμενος παίκτης είναι ο άνθρωπος ή το πρόγραμμά μας. Αν είναι σειρά του προγράμματος να παίξει, καλείται η `randomPosition`, ενώ σε διαφορετική περίπτωση καλείται όπως και πριν η `readPosition`, ώστε να επιλέξει ο χρήστης που επιθυμεί να παίξει.

```
101     if player == computer:
102         # επιλογή θέσης από το πρόγραμμα
103         position = randomPosition(board)
104     else:
105         # επιλογή θέσης από τον παίκτη
106         position = readPosition(player, board)
```

oxo/src/oxo.5.py



Σχήμα 8.5: Επιλογή κίνησης, ανάλογα με τον παίκτη: (α) αν η `computer` έχει ίδια τιμή με την `player`, τότε είναι σειρά του προγράμματος να παίξει, ενώ (β) σε διαφορετική περίπτωση, έχει σειρά να παίξει ο χρήστης.

Κάτι "Εξυπνότερο" Για Παρέα

Ωραία, τώρα το πρόγραμμα παίζει με τον παίκτη, αλλά οι τυχαίες κινήσεις του είναι σχεδόν αστείες. Θέλω να παίζει "εξυπνότερα".

Συνήθως τα προγράμματα που παίζουν τέτοια παιχνίδια χρειάζεται να κάνουν *αναζήτηση* για να παίξουν έξυπνα. Αυτό σημαίνει ότι δοκιμάζουν πολλούς διαφορετικούς συνδυασμούς κινήσεων για να καταλήξουν στην επόμενη κίνηση που θα επιλέξουν. Όμως η τρίλιζα είναι πολύ απλό παιχνίδι και δε χρειάζεται αναζήτηση. Αρκούν ορισμένες απλές οδηγίες για να παίξει κανείς ικανοποιητικά.

Εμείς θα δανειστούμε τις οδηγίες που περιγράφονται σε μια διασκεδαστική δραστηριότητα που ονομάζεται *Το Έξυπνο Χαρτί*. Και πάλι, θεωρούμε ότι το πρόγραμμά μας θα παίξει πρώτο, χρησιμοποιώντας το σύμβολο X.

Έξυπνο χαρτί: `pythonies.mysch.gr/ipaper.pdf` και οι επεκτάσεις του: `ipaper-ext.pdf`

> *Κίνηση 1:* Γράψε το X σε οποιαδήποτε γωνία.
>
> *Κίνηση 2:* Αν η γωνία που βρίσκεται διαγωνίως απέναντι

από το πρώτο X είναι ελεύθερη, τότε γράψε εκεί το X, αλλιώς γράψε το X σε οποιαδήποτε ελεύθερη γωνία.

Κινήσεις 3 και 4: Αν μπορείς, κάνε τρίλιζα με τα X. Διαφορετικά, έλεγξε αν ο αντίπαλος μπορεί να κάνει τρίλιζα και γράψε το X έτσι ώστε να τον εμποδίσεις. Αν κανένας δεν μπορεί να κάνει τρίλιζα, γράψε το X σε οποιαδήποτε ελεύθερη γωνία.

Κίνηση 5: Γράψε το X στο ελεύθερο τετράγωνο.

Νέοι έλεγχοι, παρόμοια εργαλεία

Για να υλοποιήσουμε τις κινήσεις 3 και 4, θα πρέπει το πρόγραμμά μας να ελέγχει πότε ένας παίκτης έχει *τη δυνατότητα* να κάνει τρίλιζα. Αυτό συμβαίνει όταν υπάρχει κάποια ελεύθερη θέση που συμμετέχει σε μια σχεδόν συμπληρωμένη τριάδα, δηλαδή *οι υπόλοιπες δύο θέσεις* της τριάδας είναι ήδη κατειλημμένες από τον παίκτη.

Η συνάρτηση `check` που περιέχει το πρόγραμμά μας ελέγχει κάτι παρεμφερές: αν υπάρχει κάποια συμπληρωμένη τριάδα, στην οποία *και οι τρεις θέσεις* είναι ήδη κατειλημμένες από έναν παίκτη.

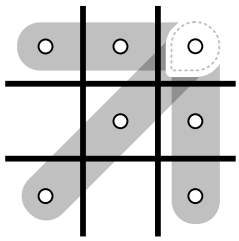
Η ομοιότητα αυτών των δύο ελέγχων οδηγεί στο εύλογο ερώτημα: μπορούμε να τροποποιήσουμε την `check`, έτσι ώστε να χρησιμοποιείται και για τους δύο ελέγχους; Η απάντηση είναι καταφατική: αρκεί η τροποποιημένη `check` να δέχεται σαν επιπρόσθετη παράμετρο μια θέση `position` και να επιστρέφει `True` αν η θέση αυτή συμμετέχει σε μια σχεδόν συμπληρωμένη τριάδα.

```
47 def check(player, board, position):
48     """ Ελέγχει αν υπάρχει 3άδα που περιέχει τη θέση
49         position και ο παίκτης player έχει καταλάβει
50         τις 2 θέσεις της 3άδας εκτός της position.
51         player: σύμβολο παίκτη ("X" ή "O")
52         board: μια λίστα με 9 στοιχεία (η τρίλιζα)
53         position: η θέση που ελέγχεται
54     """
```

Αν η παράμετρος `position` αντιστοιχεί σε μια ελεύθερη θέση και η `check` επιστρέψει την τιμή `True`, τότε η θέση συμμετέχει σε μια σχεδόν συμπληρωμένη τριάδα και ο παίκτης `player` μπορεί να κάνει τρίλιζα παίζοντας στη θέση αυτή.

Παράλληλα, αν η παράμετρος `position` αντιστοιχεί σε μια θέση που κατέχει ήδη ο παίκτης `player` και η `check` επιστρέψει την τιμή `True`, αυτό σημαίνει ότι ο παίκτης `player` έχει ήδη κάνει τρίλιζα (σχήμα 8.6).

Ας προχωρήσουμε τώρα στην υλοποίηση της `check`. Όπως και στην προηγούμενη εκδοχή της, η `check` θα διατρέχει μεν όλες τις τριάδες, αλλά θα εξετάζει μόνο εκείνες που περιλαμβάνουν τη θέση `position`.



Σχήμα 8.6: Ένα παράδειγμα του τρόπου λειτουργίας της `check`. Η επάνω δεξιά γωνία περιέχεται σε τρεις διαφορετικές τριάδες. Αν ένας παίκτης κατέχει τα δύο γραμμοσκιασμένα τετράγωνα σε κάποια από αυτές τις τριάδες τότε: (α) έχει τη δυνατότητα να κάνει τρίλιζα παίζοντας στην επάνω δεξιά γωνία, εφόσον είναι διαθέσιμη ή (β) έχει κάνει τρίλιζα αν έχει ήδη παίξει στην επάνω δεξιά γωνία.

Για κάθε μία από αυτές τις τριάδες, θα ελέγχει αν οι άλλες δύο θέσεις πλην της `position` είναι κατειλημμένες από τον παίκτη `player`.

```
60      # για κάθε 3άδα θέσεων
61      for triple in triples:
62          # p1, p2 οι υπόλοιπες θέσεις της 3άδας
63          # εκτός της position
64          if triple[0] == position:
65              p1, p2 = triple[1:]
66          elif triple[1] == position:
67              p1, p2 = triple[::2]
68          elif triple[2] == position:
69              p1, p2 = triple[:2]
70          else:
71              # η τριάδα δεν περιέχει την position
72              continue
73          # αν ο παίκτης έχει καταλάβει τις 2 θέσεις
74          if board[p1] == board[p2] == player:
75              return True
76      # αν φτάσουμε μέχρι εδώ, ο έλεγχος απέτυχε
77      return False
```

Συλλογές όπως οι λίστες, οι πλειάδες και τα αλφαριθμητικά, μπορούν να *τεμαχιστούν* (slicing). Με τον τεμαχισμό επιλέγεται ένα τμήμα μιας συλλογής, από το οποίο δημιουργείται μια νέα. Για να γίνει τεμαχισμός, πρέπει να προσδιοριστεί από ποια θέση θα ξεκινήσει, σε ποια θα σταματήσει και ανά πόσα στοιχεία θα επιλέγονται.

Η έκφραση `triple[1:]` είναι ένα παράδειγμα τεμαχισμού. Δημιουργεί μια νέα πλειάδα η οποία περιέχει τα στοιχεία της `triple` από το δεύτερο (θέση `1`) μέχρι και το τελευταίο. Γενικά, αν παραλειφθεί η θέση τερματισμού, τότε ο τεμαχισμός σταματά στο τέλος της λίστας, περιλαμβάνοντας και το τελευταίο στοιχείο.

Η έκφραση `triple[::2]` δημιουργεί μια νέα πλειάδα η οποία περιέχει *ανά δύο* τα στοιχεία της `triple` από το πρώτο μέχρι και το τελευταίο. Γενικά, όταν το βήμα τεμαχισμού δεν προσδιορίζεται, θεωρείται ότι είναι το `1`.

Η έκφραση `triple[:2]` δημιουργεί μια νέα πλειάδα η οποία περιέχει τα στοιχεία της `triple` από το πρώτο μέχρι το τρίτο (θέση `2`), χωρίς να το περιλαμβάνει. Γενικά, ο τεμαχισμός σταματά *πριν* το στοιχείο που βρίσκεται στη θέση τερματισμού. Αν παραλειφθεί η θέση εκκίνησης, τότε ο τεμαχισμός ξεκινά από την αρχή της λίστας.

Η εντολή **`continue`** διακόπτει την εκτέλεση των εντολών του τρέχοντος κύκλου της επανάληψης και συνεχίζει με τον επόμενο κύκλο.

Μια ενδιαφέρουσα εναλλακτική στον τρόπο με τον οποίο μπορούν ν' αποδοθούν τιμές στις μεταβλητές `p1` και `p2` είναι η εξής:

```
        # p1, p2 οι υπόλοιπες θέσεις της τριάδας
        # εκτός της position
        if triple[0] == position:
            _, p1, p2 = triple
        elif triple[1] == position:
            p1, _, p2 = triple
        elif triple[2] == position:
            p1, p2, _ = triple
        else:
            # η τριάδα δεν περιέχει την position
            continue
```

Το `_` είναι ένα κανονικό όνομα μεταβλητής, όμως *συνηθίζεται* να χρησιμοποιείται σαν μεταβλητή *για πέταμα* (throwaway variable), υποδηλώνοντας ότι η τιμή της δεν πρόκειται να χρησιμοποιηθεί στη συνέχεια.

Στο κύριο πρόγραμμα, η `check` καλείται για να ελεγχθεί αν έχει γίνει τρίλιζα από τον παίκτη που μόλις έπαιξε. Μετά την τροποποίησή της, η `check` απαιτεί και μια τρίτη παράμετρο: τη θέση που θα ελεγχθεί. Αυτή θα είναι η θέση `position`, στην οποία έγινε η πιο πρόσφατη κίνηση.

```
158     inarow = check(player, board, position)
159     # εναλλαγή παίκτη
```

Υλοποιώντας το Έξυπνο Χαρτί

Τώρα μπορούμε να γράψουμε τη συνάρτηση `paperPosition`, η οποία δέχεται σαν παράμετρο την αναπαράσταση `board` της τρίλιζας και επιστρέφει τη θέση στην οποία πρέπει να παίξει το πρόγραμμα, σύμ-

φωνα με τις οδηγίες του έξυπνου χαρτιού.

```
97  def paperPosition(board):
98      """ Επιστρέφει τον αριθμό της θέσης όπου πρέπει
99          να παίξει ο Χ, σύμφωνα με το "έξυπνο χαρτί".
100         board: Μια λίστα με 9 στοιχεία (η τρίλιζα)
101     """
```

Ξεκινώντας, θα χρειαστούμε μια λίστα με τις θέσεις που έχουν μείνει διαθέσιμες.

```
102     # μια λίστα με τις διαθέσιμες θέσεις
103     possible = available(board)
```

Οι οδηγίες του έξυπνου χαρτιού διαφοροποιούνται ανάλογα με το πλήθος των κινήσεων που έχει πραγματοποιήσει ο παίκτης με τα Χ.

```
104     # πόσες κινήσεις έχει κάνει ο Χ;
105     nbMoves = board.count("X")
```

Τώρα το πρόγραμμά μας μπορεί να ελέγχει πόσες κινήσεις έχει ήδη κάνει ο παίκτης με τα Χ, και να επιλέγει την επόμενη κίνησή του με βάση τις οδηγίες του έξυπνου χαρτιού.

Στην πρώτη κίνηση, το έξυπνο χαρτί προτρέπει τον παίκτη με τα Χ να επιλέξει οποιαδήποτε γωνία. Το πρόγραμμά μας θα επιλέγει την επάνω αριστερά γωνία.

```
106     if nbMoves == 0:
107         # κίνηση 1η: πάνω αριστερά γωνία (θέση 0)
108         return 0
```

Θα μπορούσε να επιλέγεται τυχαία οποιαδήποτε από τις τέσσερις γωνίες, αλλά είναι προτιμότερο να κρατήσουμε τον κώδικα απλό.

Στη δεύτερη κίνηση, ο παίκτης με τα Χ διαθέτει δύο επιλογές: Αν είναι διαθέσιμη η κάτω δεξιά γωνία (που βρίσκεται διαγωνίως απέναντι από τη γωνία που επέλεξε στην πρώτη του κίνηση), τότε θα επιλέγει να παίξει εκεί. Σε διαφορετική περίπτωση, το πρόγραμμά μας θα επιλέγει την επάνω δεξιά γωνία.

```
109     elif nbMoves == 1:
110         # κίνηση 2η: κάτω δεξιά γωνία (θέση 8), αν
111         # είναι διαθέσιμη, αλλιώς πάνω δεξιά (θέση 2)
112         if 8 in possible:
113             return 8
114         else:
115             return 2
```

Ο τελεστής `in` ελέγχει αν μια τιμή υπάρχει σε μια λίστα και επιστρέφει αντίστοιχα την τιμή `True` ή `False`.

Εδώ χρησιμοποιείται για να ελεγχθεί αν το `8` (κάτω δεξιά γωνία) βρίσκεται μέσα στη λίστα `possible` των διαθέσιμων θέσεων.

Για τις επόμενες δύο κινήσεις, το πρόγραμμα θα πρέπει αρχικά θα ελέγχει αν ο παίκτης με τα Χ (δηλαδή το ίδιο το πρόγραμμα) μπορεί να κάνει τρίλιζα. Για τον σκοπό αυτό, θα πρέπει να διατρέχει όλα τα διαθέσιμα τετράγωνα και για κάθε ένα από αυτά χρησιμοποιεί την check.

```
116     elif nbMoves < 4:
117         # κίνηση 3η ή 4η: αν μπορείς κάνε τρίλιζα
118         for position in possible:
119             if check("X", board, position):
120                 return position
```

Στη συνέχεια, αν διαπιστωθεί ότι ο παίκτης με τα Χ δεν μπορεί να κάνει τρίλιζα, θα πρέπει να ελέγχεται το ίδιο για τον παίκτη με τα Ο. Και πάλι, το πρόγραμμα θα πρέπει να διατρέχει όλα τα διαθέσιμα τετράγωνα και για κάθε ένα από αυτά χρησιμοποιεί την `check`.

```
121         # αλλιώς μπλόκαρε τρίλιζα του αντιπάλου
122         for position in possible:
123             if check("O", board, position):
124                 return position
```

Αν δεν υπάρχει κάποια τρίλιζα του Ο να μπλοκαριστεί, το πρόγραμμα επιλέγει την πρώτη διαθέσιμη γωνία.

```
125         # αλλιώς παίξε σε οποιαδήποτε γωνία
126         for corner in (0,2,6,8):
```

Αν το παιχνίδι έχει φτάσει μέχρι την τελευταία κίνηση, τότε είναι δεδομένο ότι έχει μείνει μόνο μία διαθέσιμη θέση, η οποία θα αποτελεί το μοναδικό στοιχείο της λίστας `possible`. Αυτή είναι και η θέση που επιλέγει αναγκαστικά το πρόγραμμά μας.

```
129     else:
130         # κίνηση 5η: παίξε στο διαθέσιμο τετράγωνο
131         return possible[0]
```

Απομένει μόνο ν' αντικαταστήσουμε στο κύριο πρόγραμμα την κλήση της `randomPosition` με μια κλήση στην `paperPosition`, έτσι ώστε το πρόγραμμά μας να παίζει ακολουθώντας πλέον τις οδηγίες του έξυπνου χαρτιού.

```
147     if player == computer:
148         # επιλογή θέσης από το πρόγραμμα
149         position = paperPosition(board)
```

oxo/src/oxo.6.py

Τροποποιήσεις – Επεκτάσεις

8.1 *Η συνάρτηση* `readPosition` *ξεκινά εμφανίζοντας στον παίκτη την τρέχουσα κατάσταση του πίνακα του παιχνιδιού:*

```
# εμφάνιση πίνακα τρίλιζας
print3x3(board)
```

Αμέσως μετά, προσθέστε στο πρόγραμμά σας τις εντολές που ακολουθούν και εκτελέστε το πρόγραμμα για να παρατηρήσετε τι συμβαίνει.

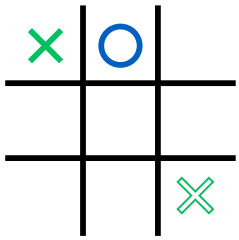
```
    print("\n")
    # εμφάνιση πιθανών κινήσεων
    p = [c if board[c]==" " else " " for c in range(9)]
    print3x3(p)
```

Τί ακριβώς περιέχει η λίστα `p` *και πως χρησιμοποιείται σε συνδυασμό με την* `print3x3`*;*

8.2 *Το έξυπνο χαρτί ξεκινά πάντα παίζοντας με τα X σε γωνιακό τετράγωνο. Αν ο παίκτης με τα O απαντήσει παίζοντας σ' ένα πλευρικό τετράγωνο (θέσεις* `1`*,* `3`*,* `5` *και* `7`*, σύμφωνα με την αρίθμησή μας) τότε το έξυπνο χαρτί θα επιλέξει τη γωνία που βρίσκεται διαγωνίως απέναντι από την αρχική (σχήμα 8.7). Αυτή η κίνηση καταλήγει σε ισοπαλία, ενώ υπάρχει καλύτερη που οδηγεί με βεβαιότητα τον παίκτη με τα X στη νίκη.*

Να επεκτείνετε τις οδηγίες του έξυπνου χαρτιού έτσι ώστε να μην παρουσιάζουν αυτή την "αδυναμία". Όταν ο παίκτης με τα O ξεκινήσει παίζοντας σε πλευρικό τετράγωνο, οι οδηγίες θα πρέπει να επιλέγουν την κατάλληλη κίνηση ώστε ο παίκτης με τα X να κερδίζει. Στη συνέχεια, να επεκτείνετε ανάλογα και τη συνάρτηση `paperPosition`*.*

`oxo/exercises/oxo-more-intelligent.py`



Σχήμα 8.7: Η 2η κίνηση του έξυπνου χαρτιού αν ο παίκτης με τα O παίξει σε πλευρικό τετράγωνο. Η κίνηση καταλήγει σε ισοπαλία, ενώ υπάρχει καλύτερη που οδηγεί αυτόματα στη νίκη.

8.3 *Μια αρίθμηση των τετραγώνων της τρίλιζας που πιθανώς να βόλευε περισσότερο τους παίκτες είναι αυτή που φαίνεται στο σχήμα 8.8. Αυτή η αρίθμηση αντιστοιχεί στη διάταξη που έχουν τα αριθμητικά πλήκτρα σε ένα τηλέφωνο ή στο πληκτρολόγιο.*

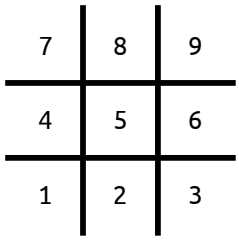
Τροποποιήστε το πρόγραμμα που αναπτύχθηκε έτσι ώστε ο παίκτης να επιλέγει το τετράγωνο στο οποίο θα παίζει με βάση αυτή την αρίθμηση.

Μια πιθανή προσέγγιση είναι να τροποποιηθεί και η εσωτερική αναπαράσταση, ώστε να υπάρχει μια φυσικότερη αντιστοιχία με την αρίθμηση που αντιλαμβάνεται ο παίκτης.

`oxo/exercises/oxo-dial-representation.py`

Εναλλακτικά, μπορεί η εσωτερική αναπαράσταση να παραμείνει ως έχει και να προστεθεί στον κώδικα ένα επίπεδο "αντιστοίχισης" ανάμεσα στην εσωτερική αναπαράσταση και την αρίθμηση από την σκοπιά του παίκτη.

`oxo/exercises/oxo-dial-mapping.py`



Σχήμα 8.8: Μια διαφορετική αρίθμηση των τετραγώνων της τρίλιζας, με βάση την οποία επιλέγει ο παίκτης το τετράγωνο στο οποίο θα παίξει. Αυτή η αρίθμηση αντιστοιχεί στο κλασικό αριθμητικό πληκτρολόγιο.

8.4 *Όταν φτάσαμε στο σημείο όπου το πρόγραμμά μας έπρεπε να αναλάβει το ρόλο ενός εκ των δύο παικτών, ήταν απαραίτητο να διαθέτουμε έναν μηχανισμό ο οποίος θα επέτρεπε στο πρόγραμμα να ελέγχει αν ένας παίκτης έχει τη δυνατότητα να κάνει τρίλιζα. Για τον σκοπό αυτό, εμείς επιλέξαμε να τροποποιήσουμε τη συνάρτηση* `check`*, η οποία έλεγχε αν ένας παίκτης είχε ήδη κάνει τρίλιζα.*

Στο βιβλίο του Invent your own computer games with Python*, ο Al Sweigart αφιερώνει επίσης ένα κεφάλαιο στην τρίλιζα και δίνει μια διαφορετική λύση σε αυτό το πρόβλημα: για να ελεγξει αν ένας παίκτης έχει τη δυνατότητα να κάνει τρίλιζα σε μια συγκεκριμένη θέση κάνει ένα* αντίγραφο *του πίνακα του παιχνιδιου, συμπληρώνει τη θέση με το σύμβολο του παίκτη και ελέγχει αν έχει γίνει τρίλιζα. Ουσιαστικά, το πρόγραμμα* δοκιμάζει *τι θα συμβεί στην επόμενη κίνηση.*

Για να δημιουργήσουμε ένα αντίγραφο μιας λίστας, μπορούμε να χρησιμοποιήσουμε τη μέθοδο `copy()`, η οποία εφαρμόζεται σε μια λίστα κι επιστρέφει ένα αντίγραφό της.

Να τροποποιήσετε το πρόγραμμα που αναπτύχθηκε έτσι ώστε να χρησιμοποιεί μια αρχική εκδοχή της `check` *που ελέγχει μόνο για τρίλιζα. Στη συνέχεια, τροποποιήστε την* `paperPosition` *ώστε να χρησιμοποιεί τη μέθοδο που περιγράψαμε για να ελέγξει αν ένας παίκτης έχει τη δυνατότητα να κάνει τρίλιζα.*

`oxo/exercises/oxo-lookahead.py`

8.5 *Το έξυπνο χαρτί παρέχει οδηγίες μόνο για λογαριασμό του παίκτη που παίζει πρώτος. Προσπαθήστε να γράψετε αντίστοιχες οδηγίες για τον παίκτη που παίζει δεύτερος και να τις υλοποιήσετε σε μια συνάρτηση αντίστοιχη της* `paperPosition`.

Για να χρησιμοποιήσετε τη συνάρτησή σας, θα χρειαστεί να κάνετε τις απαραίτητες τροποποιήσεις ώστε το πρόγραμμα να μπορεί να αναλάβει το ρόλο οποιουδήποτε από τους δύο παίκτες.

`oxo/exercises/oxo-twoplayer.py`