Το Παιχνίδι της Αφαίρεσης

4

10 Σεπτεμβρίου 2016
10:00

Στο κεφάλαιο αυτό θα φτιάξουμε ένα απλό παιχνίδι δύο παικτών μ' ενδιαφέρουσα ιστορία. Στο τέλος, το πρόγραμμά μας θα συμμετέχει στο παιχνίδι, παίζοντας το ρόλο ενός από τους δύο παίκτες. Στην πορεία θα έχουμε την ευκαιρία να έρθουμε ξανά σε επαφή με τις αλγοριθμικές δομές που έχουμε συναντήσει μέχρι στιγμής, ενώ θα χρησιμοποιήσουμε *υποπρογράμματα*, για να κατακερματίσουμε το πρόγραμμά μας σε απλούστερα τμήματα και να διαχειριστούμε την πολυπλοκότητά του.

Έννοιες: δομή επιλογής, δομή επανάληψης, υποπρογράμματα

Το παιχνίδι που θα φτιάξουμε ονομάζεται NIM. Είναι πολύ παλιό και πιθανότατα προέρχεται από την Κίνα. Έχει πολλές παραλλαγές κι εδώ θ' ασχοληθούμε με μια απλή εκδοχή του που ονομάζεται *το παιχνίδι της αφαίρεσης*. Ένα πλήθος από αντικείμενα (π.χ. σπίρτα, ξυλάκια) τοποθετούνται στη σειρά και ο κάθε ένας από τους δύο παίκτες αφαιρεί με τη σειρά του από ένα μέχρι και τρία αντικείμενα, μέχρι να μείνει μόνο ένα. Ο παίκτης που θα μείνει με το τελευταίο αντικείμενο όταν είναι η σειρά του να παίξει *χάνει* το παιχνίδι. Στη γενικότερη εκδοχή του NIM, υπάρχουν περισσότερες σειρές από αντικείμενα.

Το NIM είναι ένα από τα πρώτα παιχνίδια που αυτοματοποιήθηκαν και, όπως θα δούμε στη συνέχεια, υπάρχει λόγος γι' αυτό. Ήδη από το 1940 η αμερικάνικη εταιρεία Westinghouse παρουσίασε ένα μηχάνημα που έπαιζε NIM, το Nimatron, ενώ στις αρχές της δεκαετίας του '50 εμφανίστηκαν ειδικού σκοπού ηλεκτρονικοί υπολογιστές που έπαιζαν NIM, όπως ο NIMROD της βρετανικής Ferranti.

Το Στήσιμο

Ας υποθέσουμε ότι τα αντικείμενα που χρησιμοποιούν οι παίκτες είναι σπίρτα. Με πόσα πρέπει να ξεκινήσουμε;

Οι κανόνες του παιχνιδιού δεν προσδιορίζουν το αρχικό πλήθος των σπίρτων. Μπορούμε λοιπόν να το ορίσουμε μόνοι μας.

```
# αρχικό πλήθος σπίρτων
matches = 7
```

Εναλλακτικά, μπορούμε να αρχικοποιήσουμε τη μεταβλητή `matches` με μια τυχαία τιμή (ας πούμε από το 7 μέχρι και το 21).

```
import random
# αρχικό πλήθος σπίρτων
matches = random.randint(7,21)
```

Καλό θα ήταν να ενημερώσουμε το χρήστη με ένα κατάλληλο μήνυμα για τον αρχικό αριθμό των σπίρτων.

```
# εμφάνιση αρχικού πλήθους σπίρτων
print("Αρχικό πλήθος σπίρτων:", matches)
```

Κάνε Ένα Γύρο

Τώρα θα πρέπει να γράψω τις εντολές για έναν γύρο του παιχνιδιού και να τις κάνω να επαναλαμβάνονται μέχρι να τελειώσει το παιχνίδι.

Το συγκεκριμένο παιχνίδι συνεχίζεται όσο απομένουν ακόμα κάποια σπίρτα – αυτή είναι και η *συνθήκη συνέχειας* της επανάληψης.

```
# επανάληψη: συνεχίζεται όσο υπάρχουν σπίρτα
while matches > 0:
```

Θα ακολουθήσουν οι εντολές που είναι εμφωλευμένες στην επαναληπτική δομή, δηλαδή οι εντολές που θα εκτελούνται ξανά και ξανά σε κάθε γύρο του παιχνιδιού.

Σε κάθε γύρο θα ζητείται από έναν παίκτη το πλήθος των σπίρτων που επιθυμεί να αφαιρέσει.

```
    # ανάγνωση σπίρτων που θα πάρει ο παίκτης
    print("Πόσα σπίρτα θέλεις;")
    removed = int(input())
```

Η μεταβλητή `matches` θα πρέπει να μειώνεται σε κάθε κύκλο κατά το πλήθος των σπίρτων που αφαιρούνται από το τραπέζι.

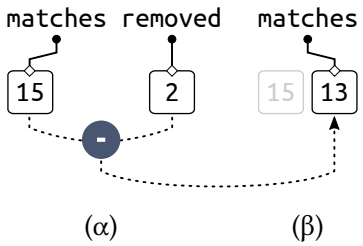
```
    # μείωση σπίρτων
    matches = matches - removed
```

Κάτι ακόμα που χρειάζεται σε κάθε κύκλο του παιχνιδιού είναι να εμφανίζουμε στους παίκτες την τιμή της μεταβλητής `matches`, ώστε να αποφασίζουν για το πλήθος των σπίρτων που θα αφαιρέσουν.

```
    # εμφάνιση πλήθους σπίρτων που απομένουν
    print("Σπίρτα που απομένουν:", matches)
```

nim/src/nim.1.py

Η τιμή που επιστρέφει η `input()` είναι *αλφαριθμητική*, πρόκειται για το κείμενο που πληκτρολόγησε ο χρήστης. Χρειάζεται να μετατρέψουμε την τιμή αυτή σε ακέραιο αριθμό, να της αλλάξουμε τον *τύπο*, και για την μετατροπή αυτή χρησιμοποιούμε την `int()`.



Σχήμα 4.1: Παράδειγμα μείωσης του πλήθους των σπίρτων: (α) υπολογίζεται η τιμή της παράστασης `matches - removed`, δηλαδή το πλήθος των σπίρτων που θ' απομείνουν στο τραπέζι αφού αφαιρεθούν τα σπίρτα που επιλέγει ο παίκτης και (β) η τιμή αυτή αποτελεί τη νέα τιμή της `matches`.

Ποιός Παίζει;

Προς το παρόν δεν υπάρχει διαφοροποίηση ανάμεσα στους παίκτες. Δεν ξέρουμε ούτε ποιος παίζει κάθε φορά, ούτε ποιος νικάει.

Σε κάθε κύκλο του παιχνιδιού, το πρόγραμμα ρωτάει πόσα σπίρτα θα αφαιρεθούν. Ωστόσο, δεν καταγράφει ποιος από τους δύο παίκτες είναι που αφαιρεί κάθε φορά τα σπίρτα κι έτσι δεν είναι σε θέση να υπολογίσει ποιος είναι ο νικητής όταν αυτά τελειώσουν.

Υπάρχουν αρκετοί τρόποι να λύσουμε αυτό το πρόβλημα. Μια απλή προσέγγιση είναι να χρησιμοποιήσουμε μια μεταβλητή `player`, η οποία σε κάθε γύρο θα παίρνει *εναλλάξ* την τιμή 1 ή 2, υποδεικνύοντας με αυτόν τον τρόπο ποιος παίκτης έχει σειρά να παίξει.

Σχήμα 4.2: Η τιμή της μεταβλητής `player` εναλλάσσεται σε κάθε γύρο μεταξύ του 1 και του 2, υποδεικνύοντας τον αριθμό του παίκτη που έχει σειρά να παίξει. Η αρχική της τιμή είναι το 1.

Αρχικά, πριν ξεκινήσει η διαδικασία του παιχνιδιού, ορίζουμε ποιος παίκτης ξεκινάει πρώτος: αυτός θα είναι πάντα ο παίκτης με αριθμό 1.

```
# ορισμός παίκτη που θα παίξει πρώτος
player = 1
```

Τώρα πλέον η προτροπή που εμφανίζεται σε κάθε κύκλο απευθύνεται σε συγκεκριμένο παίκτη:

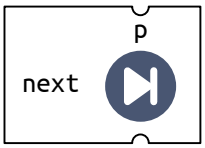
```
    # ανάγνωση σπίρτων που θα πάρει ο παίκτης
    print("Παίκτη", player, "πόσα σπίρτα θέλεις;")
    removed = int(input())
```

nim/src/nim.2.py

Πριν τελειώσει κάθε κύκλος της επανάληψης, η τιμή της μεταβλητής `player` θα πρέπει να τροποποιείται, ώστε να υποδεικνύει τον επόμενο παίκτη που έχει σειρά να παίξει.

Για τον υπολογισμό του επόμενου παίκτη θα χρησιμοποιήσουμε ένα υποπρόγραμμα, το οποίο δέχεται σαν παράμετρο τον αριθμό `p` ενός παίκτη και επιστρέφει τον αριθμό του επόμενου παίκτη.

```
def next(p):
    """ Επιστρέφει τον αριθμό του παίκτη
    που παίζει μετά τον παίκτη p.
    p: αριθμός παίκτη (1 ή 2)
    """
    if p == 1:
        return 2
    else:
        return 1
```

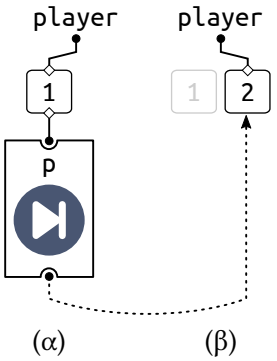


Σχήμα 4.3: Αναπαράσταση της συνάρτησης `next`, η οποία δέχεται σαν παράμετρο τον αριθμό `p` ένος παίκτη και επιστρέφει τον αριθμό του παίκτη που παίζει μετά τον παίκτη `p`.

Το υποπρόγραμμα δεν τροποποιεί την τιμή της μεταβλητής `player`, απλά επιστρέφει την επόμενη τιμή της. Στο κύριο πρόγραμμα, στο τέλος της επανάληψης, καλείται το υποπρόγραμμα και η τιμή που επιστρέφει, δηλαδή ο αριθμός του επόμενου παίκτη, γίνεται η νέα τιμή της μεταβλητής `player`.

```
    # εναλλαγή παίκτη
    player = next(player)
```

Η χρήση του υποπρογράμματος στην εναλλαγή του παίκτη διατηρεί τον κώδικά μας σύντομο και ευανάγνωστο. Με μία και μοναδική γραμμή καθορίζουμε ότι η τιμή της `player` θα μεταβληθεί ώστε ν' αντιστοιχεί στον επόμενο παίκτη. Το *πως* θα γίνει αυτό καθορίζεται εντός του υποπρογράμματος και δεν επηρρεάζει το σημείο όπου καλείται το υποπρόγραμμα.



Σχήμα 4.4: Παράδειγμα εναλλαγής παίκτη: στο τέλος κάθε γύρου: (α) καλείται η συνάρτηση `next`, με παράμετρο την τιμή της μεταβλητής `player`. Η `next` επιστρέφει μια τιμή που αντιστοιχεί στον αριθμό του παίκτη που έχει σειρά να παίξει στον επόμενο γύρο και (β) αυτή η τιμή γίνεται η νέα τιμή της `player`.

Πράγματι, δεν υπάρχει μόνο ένας τρόπος να υλοποιηθεί η `next`. Στον κώδικα που ακολουθεί δίνεται μια εναλλακτική προσέγγιση. Ανεξάρτητα από την εκδοχή της `next` που θα επιλέξουμε να χρησιμοποιήσουμε, η κλήση της θα παραμείνει η ίδια.

```
def next(p):
    """ Επιστρέφει τον αριθμό του παίκτη
    που παίζει μετά τον παίκτη p.
    p: αριθμός παίκτη (1 ή 2)
    """
    return (p % 2) + 1
```

Η έκφραση `p % 2` υπολογίζει το υπόλοιπο της ακέραιας διαίρεσης του `p` με το `2`. Η τιμή της μπορεί να είναι `0` ή `1`.

Όταν η επανάληψη τελειώσει, η μεταβλητή `player` δείχνει ποιος παίκτης θα είχε σειρά να παίξει *μετά* τον παίκτη που πήρε τα τελευταία σπίρτα. Συνεπώς, μπορούμε να χρησιμοποιήσουμε τη μεταβλητή `player` για να διαπιστώσουμε ποιος παίκτης κέρδισε.

```
28 # εμφάνιση αποτελέσματος παιχνιδιού
29 print("Παίκτη", player, "κέρδισες!")
```

nim/src/nim.3.py

Δεν υπάρχει εσοχή πριν την εντολή που εμφανίζει το αποτέλεσμα του παιχνιδιού. Η εντολή είναι στοιχισμένη πιο αριστερά απ' τις προηγούμενες κι αυτό υποδηλώνει ότι δεν ανήκει στην επανάληψη. Αντίθετα είναι η πρώτη εντολή που θα εκτελεστεί όταν η επανάληψη διακοπεί.

Μη Λέμε κι Ό,τι Θέλουμε

Το πρόγραμμά μας επιτρέπει στους παίκτες να αφαιρέσουν σ' έναν γύρο όσα σπίρτα θέλουν!

Σύμφωνα με τους κανόνες του παιχνιδιού, ένας παίκτης επιτρέπεται να αφαιρέσει από ένα μέχρι και τρία σπίρτα κάθε φορά, αρκεί τα σπίρτα που έχουν απομείνει στο τραπέζι να είναι περισσότερα από τρία. Σε διαφορετική περίπτωση, επιτρέπεται να αφαιρέσει το πολύ όσα σπίρτα έχουν απομείνει.

Θα φτιάξουμε ένα υποπρόγραμμα που δέχεται σαν παράμετρο το πλήθος των σπίρτων που έχουν απομείνει κι επιστρέφει το *μέγιστο* πλήθος σπίρτων που επιτρέπεται ν' αφαιρεθούν. Χρειαζόμαστε αυτό το άνω όριο, για να μπορούμε να ελέγξουμε παρακάτω αν το πλήθος των σπίρτων που επιθυμεί ν' αφαιρέσει ο παίκτης είναι έγκυρο.

```
11 def maxMatches(m):
12     """ Επιστρέφει το μέγιστο πλήθος σπίρτων
13     που επιτρέπεται να αφαιρεθούν.
14     m: πλήθος σπίρτων που απομένουν
15     """
16     # το πολύ 3 σπίρτα ή όσα απομένουν
17     if m > 3:
18         return 3
19     else:
20         return m
```

Τώρα μπορούμε να κατασκευάσουμε ένα υποπρόγραμμα το οποίο διαβάζει από τον παίκτη το πλήθος των σπίρτων που επιθυμεί ν' αφαιρέσει, *ελέγχει* την τιμή που δίνει ο παίκτης κι *εξασφαλίζει* ότι αυτή δεν παραβιάζει τους κανόνες.

Το υποπρόγραμμα δέχεται ως παραμέτρους τον αριθμό `p` του παίκτη που έχει σειρά να παίξει (για να εμφανίσει την κατάλληλη προτροπή) και το πλήθος `m` των σπίρτων που απομένουν (για να κάνει τους απαραίτητους ελέγχους).

```
21 def readMatches(p,m):
22     """ Διαβάζει από το χρήστη κι επιστρέφει
23     το πλήθος σπίρτων που θα αφαιρεθούν.
24     Εξασφαλίζει ότι η τιμή είναι έγκυρη.
25     p: αριθμός παίκτη που παίζει
26     m: πλήθος σπίρτων που απομένουν
27     """
```

Αρχικά ζητείται από τον παίκτη να πληκτρολογήσει το πλήθος των σπίρτων που επιθυμεί να αφαιρέσει, έχοντας ήδη υπολογίσει το μέγιστο επιτρεπόμενο πλήθος, με βάση τα σπίρτα που απομένουν:

```
28     # μέγιστο πλήθος σπίρτων προς αφαίρεση
29     limit = maxMatches(m)
30     # ανάγνωση σπίρτων που θα πάρει ο παίκτης
31     print("Παίκτη", p, "πόσα σπίρτα θέλεις;")
32     num = int(input())
```

Όσο η τιμή που πληκτρολογείται από τον παίκτη δεν είναι *έγκυρη* (μέσα στα επιτρεπόμενα όρια), εμφανίζεται σχετικό μήνυμα και η ανάγνωση τιμής *επαναλαμβάνεται*.

```
33     # έλεγχος και επανάληψη (σε περίπτωση λάθους)
34     while num < 1 or num > limit:
35         # μήνυμα λάθους
36         print("Πάρε από 1 μέχρι και",limit,"σπίρτα.")
37         # ανάγνωση σπίρτων που θα πάρει ο παίκτης
38         print("Παίκτη", p, "πόσα σπίρτα θέλεις;")
39         num = int(input())
```

Γιατί πρέπει ο έλεγχος να επαναλαμβάνεται; Γιατί να μην ελεγχθεί η τιμή που δίνει ο παίκτης με μια απλή δομή επιλογής; Πρέπει να σκεφτούμε ότι ο χρήστης μπορεί να πληκτρολογεί *συνεχώς* τιμές που δεν είναι έγκυρες. Χρειαζόμαστε λοιπόν την επανάληψη ώστε η ανάγνωση τιμής να εγκλωβιστεί σ' έναν κύκλο, ο οποίος διακόπτεται μόνο όταν ο χρήστης δώσει έγκυρη τιμή.

Όταν ολοκληρωθεί η επανάληψη, θα έχουμε εξασφαλίσει ότι η τιμή που θα έχει διαβαστεί από το χρήστη, η τιμή της `num`, θα είναι έγκυρη. Μπορούμε λοιπόν πλέον να επιστρέψουμε αυτή την τιμή, ως το αποτέλεσμα του υποπρογράμματος.

```
40     # επιστροφή τιμής
41     return num
```

Τώρα, στο κύριο πρόγραμμα μπορούμε να αντικαταστήσουμε την εντολή εισόδου που διαβάζει χωρίς έλεγχο τον αριθμό των σπίρτων, με την κλήση του υποπρογράμματος που κατασκευάσαμε.

```
52      # ανάγνωση σπίρτων που θα πάρει ο παίκτης
53      removed = readMatches(player, matches)
```

nim/src/nim.4.py

Αν δεν είχαμε κατασκευάσει το υποπρόγραμμα, θα είχαμε τοποθετήσει όλον τον κώδικα που υλοποιεί τους απαραίτητους ελέγχους μέσα στο κύριο πρόγραμμα, αυξάνοντας έτσι σημαντικά την πολυπλοκότητά του.

Χαζό Μηχάνημα

Βαριέμαι να παίζω μόνος μου. Δεν γίνεται να παίζω με αντίπαλο το πρόγραμμά μου;

Το πρόγραμμά μας θα πρέπει να αποφασίζει για το πλήθος των σπίρτων που θα αφαιρέσει. Δεν είναι ανάγκη να καταλήξουμε από την αρχή σε κάποια ευφυή στρατηγική, μπορούμε να ξεκινήσουμε με μια τυχαία επιλογή σπίρτων.

Θα κατασκευάσουμε ένα απλό υποπρόγραμμα το οποίο δέχεται σαν παράμετρο το πλήθος των σπίρτων που έχουν απομείνει και επιλέγει τυχαία να αφαιρέσει από 1 μέχρι το μέγιστο πλήθος σπίρτων που επιτρέπεται να αφαιρεθούν.

```
42  def randomMatches(m):
43      """ Επιλέγει κι επιστρέφει ένα τυχαίο
44      αλλά έγκυρο πλήθος σπίρτων που θα αφαιρεθούν.
45      m: πλήθος σπίρτων που απομένουν
46      """
47      return random.randint(1,maxMatches(m))
```

Οι παράμετροι που χρησιμοποιούνται κατά την κλήση μιας συνάρτησης δεν είναι μόνο σταθερές ή μεταβλητές, αλλά κι ολόκληρες εκφράσεις, που μπορεί να περιλαμβάνουν κλήσεις συναρτήσεων, όπως εδώ η `maxMatches`. Στο παράδειγμα αυτό, *πρώτα* καλείται η `maxMatches` και η τιμή που επιστρέφει χρησιμοποιείται σαν δεύτερη παράμετρος στην κλήση της `randint`.

Στην τελευταία γραμμή, η τιμή που επιστρέφει η `maxMatches()` χρησιμοποιείται ως άνω όριο για τον τυχαίο αριθμό που θα δημιουργηθεί. Είναι σχεδόν το ίδιο με τον κώδικα που ακολουθεί, αλλά χωρίς την ενδιάμεση μεταβλητή `limit`.

```
    limit = maxMatches(m)
    return random.randint(1, limit)
```

Είναι ενδιαφέρον ότι η συνάρτηση `maxMatches()` χρησιμοποιείται τώρα και από την `readMatches()` για την ανάγνωση αριθμού από τον χρήστη, αλλά και από την `randomMatches()` για την επιλογή ενός αριθμού σπίρτων από το ίδιο το πρόγραμμα. Αποδεικνύεται έτσι ένα χρήσιμο "εξάρτημα" που *επαναχρησιμοποιείται* σε διαφορετικά σημεία του προγράμματος για διαφορετικούς λόγους.

Ας εστιάσουμε τώρα στο κύριο πρόγραμμα, όπου θα πρέπει να εισάγουμε τις απαραίτητες επεκτάσεις ώστε το ίδιο το πρόγραμμα να αναλαμβάνει πλέον το ρόλο ενός από τους δύο παίκτες. Θα χρησιμοποιήσουμε τη μεταβλητή `computer`, της οποίας η τιμή θ' αντιστοιχεί στον αριθμό του παίκτη που θα παίζει αυτοματοποιημένα. Στην αρχή του παιχνιδιού η `computer` θα παίρνει τυχαία την τιμή 1 ή 2.

```
52 # επιλογή παίκτη-υπολογιστή
53 computer = random.randint(1,2)
```

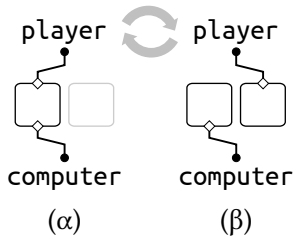
Θυμηθείτε ότι η μεταβλητή `player` εναλλάσσεται σε κάθε γύρο μεταξύ των τιμών 1 και 2, υποδεικνύοντας ποιος παίκτης έχει σειρά να παίξει στον επόμενο γύρο. Συκρίνοντας λοιπόν την `player` με την `computer` ελέγχουμε ουσιαστικά αν ο επόμενος παίκτης είναι ο άνθρωπος ή το πρόγραμμά μας, ώστε να καλέσουμε σε κάθε περίπτωση την ανάλογη συνάρτηση που θα μας επιστρέψει το πλήθος των σπίρτων που θα αφαιρεθούν.

```
58     # επιλογή κίνησης, ανάλογα με τον παίκτη
59     if player == computer:
60         # σπίρτα που θα πάρει ο υπολογιστής
61         removed = randomMatches(matches)
62         print("Ο υπολογιστής παίρνει", removed)
63     else:
64         # ανάγνωση σπίρτων που θα πάρει ο παίκτης
65         removed = readMatches(player, matches)
```

Χρειάζεται να φροντίσουμε μια ακόμα σημαντική λεπτομέρεια. Μέχρι στιγμής, στο τέλος του παιχνιδιού ανακοινώνεται ο *αριθμός* του παίκτη που κέρδισε. Τώρα που μόνο ο ένας παίκτης είναι άνθρωπος, είναι προτιμότερο να εμφανίζουμε διαφοροποιημένα μηνύματα.

```
72 # εμφάνιση αποτελέσματος παιχνιδιού
73 if player == computer:
74     print("Κέρδισε ο υπολογιστής.")
75 else:
76     print("Παίκτη", player, "κέρδισες!")
```

nim/src/nim.5.py



Σχήμα 4.5: 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.

Άνθρωπος Εναντίον Μηχανής

Όταν το πρόγραμμά μου παίζει τυχαία δεν έχει και μεγάλο ενδιαφέρον. Δεν γίνεται να το κάνουμε λίγο πιο "έξυπνο";

Θα πρέπει πρώτα εμείς να σχεδιάσουμε έναν καλύτερο τρόπο παιχνιδιού και μετά να τον περιγράψουμε με τις κατάλληλες εντολές. Είναι ευκολότερο ν' αρχίσουμε μελετώντας ποιες είναι οι ενδεδειγμένες κινήσεις όταν απομένουν 2, 3 ή 4 σπίρτα: ο παίκτης που έχει σειρά να παίξει μπορεί να αφαιρέσει αντίστοιχα 1, 2 ή 3 σπίρτα και να κερδίσει άμεσα. Αντίθετα, όταν απομένουν 5 σπίρτα η κατάσταση είναι δύσκολη: όσα σπίρτα και ν' αφαιρέσει ο παίκτης που έχει σειρά να παίξει, η έκβαση είναι στα χέρια του αντιπάλου.

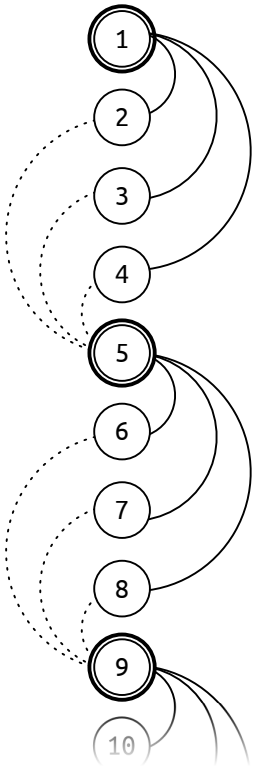
Ανάλογες δυσάρεστες καταστάσεις, που θα ονομάσουμε «ανεπιθύμητες νησίδες», αντιμετωπίζει ο παίκτης που έχει σειρά να παίξει όταν απομένουν 9, 13, 17, κ.ο.κ σπίρτα. Όσα σπίρτα κι αν αφαιρέσει, ο αντίπαλος μπορεί να τον στείλει στην επόμενη νησίδα και να τον αναγκάσει να χάσει (σχήμα 4.6).

σπίρτα `m` που απομένουν στον παίκτη	σπίρτα που πρέπει ν' αφαιρέσει ο παίκτης	σπίρτα που θ' απομείνουν στον αντίπαλο
2	1	1
3	2	1
4	3	1
5	αδιάφορο	
6	1	5
7	2	5
8	3	5
9	αδιάφορο	
10	1	9
...		

Στον πίνακα υπάρχει εμφανής *περιοδικότητα*: οι ενδεδειγμένες κινήσεις επαναλαμβάνονται *ανά τέσσερα σπίρτα.* Αυτό είναι ένδειξη ότι για να υπολογίσουμε το πλήθος των σπίρτων που θα πρέπει ν' αφαιρεθούν, θα πρέπει να χρησιμοποιήσουμε το υπόλοιπο της ακέραιας διαίρεσης του πλήθους των σπίρτων `m` με το 4. Η δεύτερη στήλη του πίνακα που ακολουθεί περιέχει την τιμή της έκφρασης `m % 4`.

σπίρτα `m` που απομένουν στον παίκτη	τιμή έκφρασης `m % 4`	τιμή έκφρασης `(m - 1) % 4`	σπίρτα που πρέπει ν' αφαιρέσει ο παίκτης
2	2	1	1
3	3	2	2
4	0	3	3
5	1	0	αδιάφορο
6	2	1	1
7	3	2	2
8	0	3	3
9	1	0	αδιάφορο
10	2	1	1
...			

Το υποπρόγραμμα που ακολουθεί δέχεται σαν παράμετρο το πλήθος `m` των σπίρτων που απομένουν (πρώτη στήλη), υπολογίζει την τιμή της έκφρασης `m % 4` (δεύτερη στήλη) και, με βάση την αντιστοιχία του πίνακα, επιστρέφει το πλήθος των σπίρτων που χρειάζεται ν'



Σχήμα 4.6: Οι κύκλοι αντιστοιχούν στις διαφορετικές καταστάσεις του παιχνιδιού: ο αριθμός κάθε κύκλου είναι τα σπίρτα που απομένουν σε κάθε κατάσταση. Οι καταστάσεις 1, 5, 9, ... είναι οι «ανεπιθύμητες νησίδες»: όταν ένας παίκτης βρίσκεται σε μια από αυτές τότε ο αντίπαλός του μπορεί πάντα να τον οδηγήσει στην επόμενη νησίδα και, τελικά, στην ήττα.

αφαιρεθούν ώστε ο αντίπαλος να οδηγηθεί σε μια ανεπιθύμητη νησίδα (τέταρτη στήλη). Αν το πλήθος `m` αντιστοιχεί σε ανεπιθύμητη νησίδα, το υποπρόγραμμα επιστρέφει έναν τυχαίο αριθμό σπίρτων.

```
def computeMatches(m):
    """ Επιλέγει κι επιστρέφει το βέλτιστο
    πλήθος σπίρτων που θα πρέπει να αφαιρεθούν.
    Αν δεν υπάρχει, επιστρέφει μια τυχαία τιμή.
    m: πλήθος σπίρτων που απομένουν
    """
    # υπολογισμός υπολοίπου
    mod = m % 4
    if mod == 0:
        return 3
    elif mod == 1:
        # ανεπιθύμητη νησίδα: τυχαία κίνηση
        return randomMatches(m)
    elif mod == 2:
        return 1
    else:
        return 2
```

Θα μπορούσαμε να υπολογίσουμε πιο άμεσα το πλήθος των σπίρτων που χρειάζεται ν' αφαιρεθούν σε κάθε περίπτωση; Ουσιαστικά, θα θέλαμε να υπολογίσουμε ποια είναι η *απόσταση* από την κοντινότερη ανεπιθύμητη νησίδα ή, με άλλα λόγια, πόσα σπίρτα *περισσεύουν* σε σχέση με την πλησιέστερη νησίδα, ώστε να τα αφαιρέσουμε.

Μελετώντας τον προηγούμενο πίνακα, προκύπτει ότι η έκφραση που δίνει αυτή την απόσταση είναι η `(m - 1) % 4`, της οποίας η τιμή δίνεται στην τρίτη στήλη. Παρατηρήστε πως οι τιμές στην τρίτη και την τέταρτη στήλη ουσιαστικά ταυτίζονται. Οδηγούμαστε έτσι σε μια κομψότερη εναλλακτική υλοποίηση του υποπρογράμματος:

```
48 def computeMatches(m):
49     """ Επιλέγει κι επιστρέφει το βέλτιστο
50     πλήθος σπίρτων που θα πρέπει να αφαιρεθούν.
51     Αν δεν υπάρχει, επιστρέφει μια τυχαία τιμή.
52     m: πλήθος σπίρτων που απομένουν
53     """
54     # υπολογισμός υπολοίπου
55     mod = (m - 1) % 4
56     if mod == 0:
57         # ανεπιθύμητη νησίδα: τυχαία κίνηση
58         return randomMatches(m)
59     else:
60         return mod
```

Στο κύριο πρόγραμμα, η κλήση της `randomMatches()` για την επιλογή ενός τυχαίου πλήθους σπίρτων πρέπει τώρα να αντικατασταθεί από την κλήση της `computeMatches()`.

```
73          # σπίρτα που θα πάρει ο υπολογιστής
74          removed = computeMatches(matches)
75          print("Ο υπολογιστής παίρνει", removed)
```

nim/src/nim.6.py

Πλήρες Τελικό Πρόγραμμα

```
1  import random
2  def next(p):
3      """ Επιστρέφει τον αριθμό του παίκτη
4      που παίζει μετά τον παίκτη p.
5      p: αριθμός παίκτη (1 ή 2)
6      """
7      if p == 1:
8          return 2
9      else:
10         return 1
11 def maxMatches(m):
12     """ Επιστρέφει το μέγιστο πλήθος σπίρτων
13     που επιτρέπεται να αφαιρεθούν.
14     m: πλήθος σπίρτων που απομένουν
15     """
16     # το πολύ 3 σπίρτα ή όσα απομένουν
17     if m > 3:
18         return 3
19     else:
20         return m
21 def readMatches(p,m):
22     """ Διαβάζει από το χρήστη κι επιστρέφει
23     το πλήθος σπίρτων που θα αφαιρεθούν.
24     Εξασφαλίζει ότι η τιμή είναι έγκυρη.
25     p: αριθμός παίκτη που παίζει
26     m: πλήθος σπίρτων που απομένουν
27     """
28     # μέγιστο πλήθος σπίρτων προς αφαίρεση
29     limit = maxMatches(m)
30     # ανάγνωση σπίρτων που θα πάρει ο παίκτης
31     print("Παίκτη", p, "πόσα σπίρτα θέλεις;")
32     num = int(input())
33     # έλεγχος και επανάληψη (σε περίπτωση λάθους)
34     while num < 1 or num > limit:
35         # μήνυμα λάθους
36         print("Πάρε από 1 μέχρι και",limit,"σπίρτα.")
37         # ανάγνωση σπίρτων που θα πάρει ο παίκτης
38         print("Παίκτη", p, "πόσα σπίρτα θέλεις;")
39         num = int(input())
40     # επιστροφή τιμής
41     return num
```

```
42 def randomMatches(m):
43     """ Επιλέγει κι επιστρέφει ένα τυχαίο
44     αλλά έγκυρο πλήθος σπίρτων που θα αφαιρεθούν.
45     m: πλήθος σπίρτων που απομένουν
46     """
47     return random.randint(1,maxMatches(m))

48 def computeMatches(m):
49     """ Επιλέγει κι επιστρέφει το βέλτιστο
50     πλήθος σπίρτων που θα πρέπει να αφαιρεθούν.
51     Αν δεν υπάρχει, επιστρέφει μια τυχαία τιμή.
52     m: πλήθος σπίρτων που απομένουν
53     """
54     # υπολογισμός υπολοίπου
55     mod = (m - 1) % 4
56     if mod == 0:
57         # ανεπιθύμητη νησίδα: τυχαία κίνηση
58         return randomMatches(m)
59     else:
60         return mod

61 # αρχικό πλήθος σπίρτων
62 matches = random.randint(7,21)
63 # εμφάνιση αρχικού πλήθους σπίρτων
64 print("Αρχικό πλήθος σπίρτων:", matches)
65 # επιλογή παίκτη-υπολογιστή
66 computer = random.randint(1,2)
67 # ορισμός παίκτη που θα παίξει πρώτος
68 player = 1

69 # επανάληψη: συνεχίζεται όσο υπάρχουν σπίρτα
70 while matches > 0:
71     # επιλογή κίνησης, ανάλογα με τον παίκτη
72     if player == computer:
73         # σπίρτα που θα πάρει ο υπολογιστής
74         removed = computeMatches(matches)
75         print("Ο υπολογιστής παίρνει", removed)
76     else:
77         # ανάγνωση σπίρτων που θα πάρει ο παίκτης
78         removed = readMatches(player, matches)
79     # μείωση σπίρτων
80     matches = matches - removed
81     # εμφάνιση πλήθους σπίρτων που απομένουν
82     print("Σπίρτα που απομένουν:", matches)
83     # εναλλαγή παίκτη
84     player = next(player)
```

```
85  # εμφάνιση αποτελέσματος παιχνιδιού
86  if player == computer:
87      print("Κέρδισε ο υπολογιστής.")
88  else:
89      print("Παίκτη", player, "κέρδισες!")
```

nim/src/nim.final.py

Τροποποιήσεις – Επεκτάσεις

4.1 *Το τελικό πρόγραμμα συμπεριφέρεται λίγο ανόητα όταν απομείνει μόνο ένα σπίρτο: ενώ είναι βέβαιο ότι ο παίκτης που έχει σειρά να παίξει έχει χάσει, τον ρωτάει κανονικότατα πόσα σπίρτα θέλει να αφαιρέσει. Μάλιστα, αν πληκτρολογήσει έναν μη-έγκυρο αριθμό, του απαντά με το μήνυμα* `Πάρε από 1 μέχρι και 1 σπίρτα` *.*

Να τροποποιήσετε το πρόγραμμα έτσι ώστε το παιχνίδι να σταματά αυτόματα όταν απομένει μόνο ένα σπίρτο, χωρίς να ρωτάει τον παίκτη πόσα σπίρτα θέλει να αφαιρέσει.

Για να σταματάει το παιχνίδι όταν απομένει μόνο ένα σπίρτο, αρκεί μια μικρή τροποποίηση στη συνθήκη της **`while`**. Όμως το σημείο που χρειάζεται προσοχή είναι η ανακοίνωση του νικητή.

nim/exercises/subtraction-oneleft.py

4.2 *Να υλοποιήσετε την συνάρτηση* `readMatches()` *χρησιμοποιώντας μόνο την* **`break`** *για να διακόψετε τον επαναληπτικό έλεγχο. Η συνθήκη συνέχειας της* **`while`** *θα πρέπει να είναι η* **`True`**.

nim/exercises/subtraction-break.py

4.3 *Να υλοποιήσετε μια συνάρτηση η οποία ανακοινώνει το νικητή του παιχνιδιού, αφού δεχτεί τις κατάλληλες παραμέτρους. Χρησιμοποιήστε την συνάρτησή σας καλώντας την από το κύριο πρόγραμμα.*

nim/exercises/subtraction-announce.py

4.4 *Να τροποποιήσετε το παιχνίδι έτσι ώστε ο παίκτης που παίρνει τα τελευταία σπίρτα να κερδίζει. Ποιά είναι τώρα η βέλτιση στρατηγική για έναν παίκτη; Τροποποιήστε ανάλογα την συνάρτηση* `computeMatches()` *που υπολογίζει τον αριθμό σπίρτων που πρέπει ν' αφαιρεθούν σε μια δεδομένη κατάσταση.*

nim/exercises/subtraction-nonmisere.py

4.5 *Η συνάρτηση* `next` *δέχεται σαν παράμετρο τον αριθμό* `p` *ενός παίκτη και επιστρέφει τον αριθμό του παίκτη που παίζει μετά από τον* `p`*. Θεωρείται όμως δεδομένο ότι στο παιχνίδι συμμετέχουν δύο παίκτες.*

Να επεκτείνετε τη συνάρτηση `next`*, έτσι ώστε να δέχεται σαν επιπρόσθετη παράμετρο το πλήθος* `q` *των παικτών που συμμετέχουν στο παιχνίδι. Έτσι, η ίδια συνάρτηση θα μπορεί να χρησιμοποιηθεί σε διαφορετικά παιχνίδια.*

Η τιμή της παραμέτρου `q` μπορεί να είναι οποιοσδήποτε ακέραιος. Δε χρειάζεται να υποθέσετε ότι υπάρχει κάποιο όριο.

nim/exercises/next-generalized.py

Ασκήσεις

4.6 *Στο παιχνίδι «Τα Ζυγά Κερδίζουν» οι δύο παίκτες ξεκινούν με μια σειρά από αντικείμενα. Το αρχικό πλήθος των αντικειμένων πρέπει να είναι περιττός αριθμός. Κάθε ένας από τους δύο παίκτες αφαιρεί με τη σειρά του από ένα μέχρι και τέσσερα αντικείμενα, μέχρι ν' αφαιρεθούν όλα. Νικητής είναι ο παίκτης που στο τέλος του παιχνιδιού απομένει με άρτιο (ζυγό) πλήθος αντικειμένων. Να αναπτύξετε ένα πρόγραμμα που θα διαβάζει σε κάθε γύρο τον αριθμό των σπίρτων που αφαιρεί ο παίκτης που έχει σειρά και στο τέλος ανακοινώνει το νικητή.*

`nim/exercises/evenwins.py`

4.7 *Στο παιχνίδι «Τα Ζυγά Κερδίζουν» υπάρχουν επίσης ανεπιθύμητες νησίδες, δηλαδή καταστάσεις από τις οποίες ένας παίκτης δεν μπορεί ν' αποφύγει την ήττα, εφόσον ο αντίπαλός του παίξει σωστά. Αν ένας παίκτης δεν βρίσκεται σε ανεπιθύμητη νησίδα τότε μπορεί να οδηγήσει τον αντίπαλό του σε μία και να κερδίσει. Προσπαθήστε να καταγράψετε τις πιθανές καταστάσεις του παιχνιδιού όταν απομένουν λίγα σπίρτα (π.χ. από 1 μέχρι και 6) και ποια είναι η καλύτερη κίνηση σε κάθε μία απ' αυτές. Το εγχείρημα είναι δυσκολότερο απ' ότι στο Παιχνίδι της Αφαίρεσης γιατί εδώ η καλύτερη κίνηση δεν εξαρτάται μόνο από το πλήθος των σπίρτων που απομένουν. Πρέπει να λάβετε υπόψη αν ο καθένας από τους δύο παίκτες έχει συγκεντρώσει άρτιο ή περιττό πλήθος σπίρτων. Τελικός σας σκοπός είναι να τροποποιήσετε το πρόγραμμα της προηγούμενης άσκησης έτσι ώστε να παίζει με αντίπαλο το χρήστη.*

`nim/exercises/evenwins-auto.py`

4.8 *Να επεκτείνετε το παιχνίδι Ανάμεσα ή Acey Ducey (κεφ. 2), έτσι ώστε να παίζεται επαναληπτικά. Το πρόγραμμά σας θα πρέπει αρχικά να ρωτάει τον παίκτη το συνολικό ποσό με το οποίο επιθυμεί να ξεκινήσει το παιχνίδι. Με το ίδιο ακριβώς ποσό θα ξεκινήσει και η «μάνα». Σε κάθε γύρο, όταν ο παίκτης ερωτάται για το ποσό που θα στοιχηματίσει, το πρόγραμμά σας θα πρέπει να ελέγχει ότι το ποσό αυτό είναι θετικό και δεν ξεπερνά το συνολικό ποσό που διαθέτει* εκείνη την στιγμή *ο παίκτης, διαφορετικά θα πρέπει να επαναλαμβάνει την ερώτηση μέχρι το στοίχημα να είναι έγκυρο. Το παιχνίδι θα τελειώνει όταν εξαντληθούν τα χρήματα του παίκτη ή της «μάνας».*

`nim/exercises/aceyducey-iterative.py`

4.9 *Να μετατρέψετε το «Μάντεψε τον Αριθμό» σε παιχνίδι δύο παικτών, ανάμεσα στο χρήστη και το πρόγραμμά σας. Τόσο ο χρήστης, όσο και το πρόγραμμα, θα επιλέγουν στην αρχή του παιχνιδιού έναν μυστικό αριθμό και θα προσπαθούν να μαντέψουν ο ένας τον αριθμό του άλλου, διαθέτοντας ένα συγκεκριμένο πλήθος προσπαθειών. Σε κάθε γύρο του παιχνιδιού, κάθε ένας από τους δύο παίκτες θα επιλέγει έναν αριθμό και ο άλλος παίκτης θα τον ενημερώνει αν ο μυστικός αριθμός του είναι ίσος, μικρότερος ή μεγαλύτερος από αυτόν.*

Στο παράδειγμα που ακολουθεί δίνεται η αλληλεπίδραση μεταξύ χρήστη και προγράμματος για έναν γύρο του παιχνιδιού. Πρώτα επιλέγει αριθμό ο παίκτης, προσπαθώντας να μαντέψει τον μυστικό αριθμό του

προγράμματος, και στη συνέχεια το ίδιο το πρόγραμμα, προσπαθώντας να μαντέψει τον μυστικό αριθμό του παίκτη.

```
Σου απομένουν 3 προσπάθειες.
Μάντεψε τον αριθμό μου: 25
Λάθος. Είναι μεγαλύτερος.

Μου απομένουν 3 προσπάθειες.
Επιλέγω τον αριθμό 24. Ο αριθμός σου είναι:
1. Μικρότερος 2. Μεγαλύτερος 3. Ίσος; 1
```

Το ποιος από τους δύο παίκτες θα ξεκινήσει πρώτος θα επιλέγεται τυχαία. Αν ο παίκτης που παίξει πρώτος βρει τον μυστικό αριθμό, το πρόγραμμα δεν θα πρέπει να διακόπτεται άμεσα, αλλά να ολοκληρώνει το γύρο, δίνοντας ουσιαστικά και στο δεύτερο παίκτη το ίδιο πλήθος προσπαθειών.

nim/exercises/guess-twoplayer.py

4.10 *Τα παιδιά στο δημοτικό μαθαίνουν πρόσθεση μέσα από ασκήσεις αυξανόμενης δυσκολίας. Αρχικά, τους ζητείται να προσθέσουν δύο μονοψήφιους αριθμούς, το άθροισμα των οποίων δεν ξεπερνά το 9.*

Παράδειγμα: 4 + 2 = _

Μετά τους ζητείται να προσθέσουν δύο μονοψήφιους αριθμούς, το άθροισμα των οποίων ξεπερνά το 10.

Παράδειγμα: 4 + 8 = _

Το επόμενο στάδιο είναι η πρόσθεση ενός διψήφιου κι ενός μονοψήφιου, το άθροισμα των οποίων πιθανώς να ξεπερνά τη δεκάδα του διψήφιου, αλλά όχι το 99.

Παράδειγμα: 42 + 9 = _

Μπορείτε να φτιάξετε ένα πρόγραμμα εξάσκησης για τα παιδιά του Δημοτικού. Αρχικά, το πρόγραμμά σας θα πρέπει να ρωτά ποιο είναι το επιθυμητό επίπεδο ασκήσεων και να παράγει πέντε από αυτές. Εδώ έχουμε περιγράψει τα τρία πρώτα επίπεδα, αλλά εσείς μπορείτε να φτιάξετε κι άλλα ή ακόμα και να περάσετε σε άλλες πράξεις εκτός από την πρόσθεση. Μην σας απασχολεί αν καμιά φορά τυχαίνει να εμφανίζονται διπλότυπες ασκήσεις, αυτό προς το παρόν είναι δύσκολο να το αποφύγετε. Αν το παιδί κάνει λάθος σε κάποια από τις ερωτήσεις το πρόγραμμα μπορεί να του εμφανίζει άμεσα την σωστή απάντηση ή να του δίνει κι άλλες ευκαιρίες. Ακόμα καλύτερο θα ήταν αν το πρόγραμμά σας εξηγούσε πως προκύπτει η σωστή απάντηση ή έδινε μια υπόδειξη πριν το παιδί ξαναπροσπαθήσει.

Παράδειγμα: 42 + 9 = _. Το παιδί πληκτρολογεί 52 και το πρόγραμμα του ζητά να ξαναπροσπαθήσει, υποδεικνύοντας ότι 42 + 9 = 42 + 8 + 1 = _

nim/exercises/mathgame.py

Υποπρογραμματα *Είδαμε ήδη ότι για να αντιμετωπίσουμε την αυξανόμενη έκταση και πολυπλοκότητα των προγραμμάτων χρησιμοποιούμε* υποπρογράμματα, *μικρά προγράμματα που τα συνδυάζουμε για να φτιάξουμε συνθετότερα, κατά τον ίδιο τρόπο με τον οποίο κατασκευάζουμε μια περίπλοκη μηχανή από απλούστερα εξαρτήματα, κάθε ένα από τα οποία εξυπηρετεί μια συγκεκριμένη λειτουργία. Ο κατακερματισμός των προγραμμάτων σε απλούστερα μας βοηθά να σκεφτόμαστε τμηματικά και να αναλύουμε το γενικότερο πρόβλημα σε επιμέρους απλούστερα προβλήματα. Τα υποπρογράμματα θα πρέπει να θεωρούνται όσο το δυνατόν ανεξάρτητα από το πλαίσιο μέσα στο οποίο χρησιμοποιούνται. Αντιμετωπίζουμε κάθε υποπρόγραμμα ως ένα κομμάτι κώδικα που λύνει ένα μικρό πρόβλημα με γενικό τρόπο και έτσι είναι* επαναχρησιμοποιήσιμο. *Μπορεί να κληθεί από διαφορετικά σημεία και για διαφορετικούς λόγους μέσα σ' ένα πρόγραμμα, αλλά και να χρησιμοποιηθεί και σε άλλα προγράμματα στο μέλλον. Οι εσωτερικές λεπτομέρειες της λειτουργίας κάθε υποπρογράμματος είναι κρυμμένες για εκείνους που το χρησιμοποιούν. Όποιος θέλει να χρησιμοποιήσει ένα υποπρόγραμμα δεν χρειάζεται να καταλαβαίνει πως λειτουργεί και με ποια μέθοδο υπολογίζεται το αποτέλεσμα. Ο Al Sweigart, στο εξαιρετικό του βιβλίο Invent Your Own Computer Games with Python, γράφει: «Το ωραίο με τις συναρτήσεις είναι ότι χρειάζεται μόνο να ξέρουμε τι κάνουν, αλλά όχι πως το κάνουν.» Αντίστοιχα κρυμμένες είναι και τυχόν αλλαγές στα υποπρογράμματά μας. Αν το υποπρόγραμμα τροποποιηθεί (επειδή βρήκαμε κάποιο λάθος, κάποιο τρόπο να λειτουργεί καλύτερα ή ακόμα και κάποια καλύτερη μέθοδο) οι αλλαγές θ' αφορούν μόνο το εσωτερικό του υποπρογράμματος, ενώ το υπόλοιπο πρόγραμμα θα παραμείνει αμετάβλητο.*

Υπολογιστες και Παιχνιδια *Στις αρχές της δεκαετίας του 1950 άρχισε να εξετάζεται συστηματικά η δυνατότητα χρήσης των ηλεκτρονικών υπολογιστών, που ήταν ακόμα στα πρώτα τους βήματα, για το παίξιμο παιχνιδιών. Ο Claude Shannon δημοσίευσε το 1950 μια εργασία με τίτλο Programming a Computer for Playing Chess. Εκεί περιέγραψε ένα σύνολο τεχνικών που θα επέτρεπαν σ' έναν υπολογιστή να απαριθμεί συστηματικά τις πιθανές κινήσεις ενός παιχνιδιού, να τις αξιολογεί και να παίρνει αποφάσεις σχετικά με τις κινήσεις που θα έπρεπε να επιλέξει. Μια παρόμοια εργασία, με τίτλο Digital Computers Applied to Games, δημοσίευσε το 1953 και ο Alan Turing. Το 1997, ένας υπολογιστής της IBM με το όνομα Deep Blue, κέρδισε τον παγκόσμιο πρωταθλητή Gary Kasparov. Σήμερα, ένας οικιακός υπολογιστής έχει την υπολογιστική ισχύ του Deep Blue και λέγεται ότι ο τρόπος παιχνιδιού των πρωταθλητών έχει αλλάξει σημαντικά από τότε που ισχυρά σκακιστικά προγράμματα χρησιμοποιούνται για προπόνηση και ανάλυση θέσεων. Αν και το σκάκι θεωρείται ο βασιλιάς των παιχνιδιών και συνδέεται παραδοσιακά με τη νοημοσύνη, δεν είναι φυσικά το μόνο παιχνίδι που έχει απασχολήσει τους ερευνητές. Ένα ιστορικό πρόγραμμα που έπαιζε ντάμα αναπτύχθηκε από τον Arthur Samuel τη δεκαετία του 1950. Το 1989 ξεκίνησε η ανάπτυξη του προγράμματος Chinook, το οποίο έπαιξε με τον παγκόσμιο πρωταθλητή το 1992. Το 2007, μετά από σχεδόν 20 χρόνια υπολογισμών, έγινε ανίκητο "λύνοντας" το παιχνίδι της ντάμας, έχοντας απαριθμήσει όλες τις πιθανές θέσεις του παιχνιδιού (περίπου 500 δισεκατομμύρια δισεκατομμύρια) και γνωρίζοντας τη βέλτιστη κίνηση για κάθε μια από αυτές. Το NIM είναι πολύ απλούστερο παιχνίδι. Οι (αποδεδειγμένα) καλύτερες δυνατές κινήσεις είναι γνωστές και υπολογίζονται εύκολα. Αυτός είναι ο λόγος που το NIM ήταν πιθανότατα το πρώτο παιχνίδι που αυτοματοποιήθηκε.*

Υπολογιστες και Νοημοσυνη *Είναι αναπόφευκτο να αναρωτηθεί κανείς πόσο "έξυπνο" είναι ένα σκακιστικό πρόγραμμα που κερδίζει παγκόσμιους πρωταθλητές. Το ερώτημα αν οι υπολογιστές έχουν τη δυνατότητα να επιδεικνύουν "νοημοσύνη" προέκυψε πολύ νωρίς και είναι τόσο δύσκολο ν' απαντηθεί όσο να ορίσει κανείς τι είναι η ευφυία και η νοημοσύνη. Η Ada Lovelace, κόρη του λόρδου Βύρωνα και στενή συνεργάτης του Charles Babbage είχε γράψει το 1842 ότι οι υπολογιστικές μηχανές μπορούν να κάνουν μόνο αυτά που εμείς μπορούμε να τις προγραμματίσουμε να κάνουν. Ο χαρισματικός Alan Turing έγραψε το 1950 ένα ιστορικό άρθρο με τίτλο Computing Machinery and Intelligence (Υπολογιστικές Μηχανές και Νοημοσύνη), το οποίο ξεκινούσε με την*

πρόταση "Προτείνω να εξετάσουμε την ερώτηση: Μπορούν οι μηχανές να σκεφτούν;" Πολύ συνοπτικά, ο Turing ισχυρίστηκε ότι η απάντηση μπορεί να είναι καταφατική όταν μια μηχανή μπορεί να συνομιλήσει μ' έναν άνθρωπο και να τον πείσει ότι δεν είναι μηχανή. Είναι σημαντικό ότι ο Turing εστίασε στον τρόπο με τον οποίο οι εξωτερικοί παρατηρητές αντιλαμβάνονται την συμπεριφορά μιας μηχανής ως εφυή, χωρίς να έχει σημασία με ποιες εσωτερικές διεργασίες επιτυγχάνεται αυτό. Οι απόψεις του Turing δεν είναι καθολικά αποδεκτές, το αντίθετο μάλιστα. Είναι πάντως σημαντικό να θυμάστε ότι οι υπολογιστές είναι σχεδιασμένοι για να κάνουν μόνο ένα *πράγμα: να* εκτελούν προγράμματα*, τα οποία επεξεργάζονται τις τιμές που δέχονται στην είσοδό τους και επιστρέφουν αποτελέσματα στην έξοδό τους. Με την πάροδο του χρόνου, τα εξαρτήματά των υπολογιστών εξελίσσονται, οι ταχύτητες με τις οποίες λειτουργούν αυξάνονται και οι συσκευές εισόδου-εξόδου που διαθέτουν επιτρέπουν πλέον μια πολύ φυσική αλληλεπίδραση με τους χρήστες. Όμως το γεγονός παραμένει πως για κάθε τι που κάνουν, υπάρχει ένα πρόγραμμα, γραμμένο από ανθρώπους, που τους υπαγορεύει με απόλυτη ακρίβεια* πως *να το κάνουν. Υπάρχουν βέβαια και προγράμματα που μαθαίνουν και προσαρμόζουν την συμπεριφορά τους. Υπό αυτή την έννοια, η συμπεριφορά τους δεν είναι προγραμματισμένη, αλλά ο μηχανισμός μέσω του οποίου την τροποποιούν καθώς μαθαίνουν είναι. Έτσι, τα αυτοκίνητα της Google κινούνται χωρίς οδηγό για χιλιάδες μίλια χωρίς ατυχήματα, το ρομπότ της NASA Curiosity πλοηγείται αυτόνομα στην επιφάνεια του Άρη, το σύστημα Watson της IBM κερδίζει σε τηλεπαιχνίδια και βοηθά στη διάγνωση ασθενειών, επειδή ομάδες ανθρώπων καθόρισαν (με πολύ κόπο) πως ακριβώς μπορούν να γίνουν όλα αυτά.*