2D Lists Ασκήσεις

Επιμέλεια: Α.Δρακόπουλος

Πίνακας περιεχομένων

Άσκηση 1 — Έλεγχος εγκυρότητας κελιού (in-bounds + obstacle check) 7
Εκφώνηση 7
Λύση 7
Σχόλια 7
Άσκηση 2 — Μέτρηση αντικειμένων σε grid 8
Εκφώνηση 8
Λύση 8
Σχόλια 8
Άσκηση 3 — Εύρεση όλων των αντικειμένων 8
Εκφώνηση 8
Λύση 8
Σχόλια 8
Άσκηση 4 — Εμφάνιση κόσμου με ένδειξη θέσης ρομπότ 9
Εκφώνηση 9
Λύση 9
Σχόλια 9
Άσκηση 5 — Λίστα με όλους τους γείτονες (4 κατευθύνσεις) 9
Εκφώνηση 9
Λύση 9
Σχόλια 10
Άσκηση 6 — Αλλαγή κελιού (marking) 10
Εκφώνηση 10
Λύση 10
Σχόλια 10
Άσκηση 7 — Δημιουργία άδειου κόσμου 10×10 10
Εκφώνηση 10
Λύση 10
Σχόλια 10
Άσκηση 8 — Μέτρηση εμποδίων 11
Εκφώνηση 11
Λύση 11
Σχόλια 11
Άσκηση 9 — Αντιστροφή στοιχείων γραμμής 11
Εκφώνηση 11
Λύση 11
Σχόλια 11
Άσκηση 10 — Έλεγχος αν υπάρχει αντικείμενο σε γραμμή ή στήλη 12
Εκφώνηση 12
Λύση 12
Σχόλια 12
Άσκηση 1 — Εντοπισμός “νησιών” εμποδίων χωρίς χρήση BFS 13
Εκφώνηση 13
Λύση 13
Σχόλια 13
Άσκηση 2 — Ανίχνευση “διαδρόμων” 14
Εκφώνηση 14
Λύση 14
Σχόλια 14
Άσκηση 3 — Εντοπισμός “αδιεξόδων” 14
Εκφώνηση 14

Λύση..... 14
Σχόλια..... 15
Άσκηση 4 — Εντοπισμός αντικειμένων που βρίσκονται σε αδιέξοδο.....15
Εκφώνηση.....15
Λύση..... 15
Σχόλια..... 15
Άσκηση 5 — Υπολογισμός απόστασης Manhattan ανάμεσα στο ρομπότ και όλα τα αντικείμενα...16
Εκφώνηση.....16
Λύση..... 16
Σχόλια..... 16
Άσκηση 6 — Εντοπισμός “ορατών” αντικειμένων σε ευθεία γραμμή.....16
Εκφώνηση.....16
Λύση..... 16
Σχόλια..... 17
Άσκηση 7 — Μετατροπή 2D grid σε heatmap εμποδίων.....17
Εκφώνηση.....17
Λύση..... 17
Σχόλια..... 17
Άσκηση 8 — Αντικατάσταση περιγράμματος grid με εμπόδια.....18
Εκφώνηση.....18
Λύση..... 18
Σχόλια..... 18
Άσκηση 9 — Μετακίνηση ρομπότ με απλή heuristic (όχι BFS).....18
Εκφώνηση.....18
Λύση..... 18
Σχόλια..... 18
Άσκηση 10 — Εντοπισμός “pockets” (περιοχών περικυκλωμένων από εμπόδια).....19
Εκφώνηση.....19
Λύση..... 19
Σχόλια..... 19
Debugging Set – 20 Ασκήσεις για world.txt.....20
Άσκηση 1 — Λάθος διαστάσεις (9×10).....20
Εκφώνηση..... 20
Απάντηση.....20
Σχόλια.....20
Άσκηση 2 — Μια γραμμή έχει 9 στοιχεία αντί για 10.....20
Εκφώνηση..... 20
Απάντηση.....20
Σχόλια.....21
Άσκηση 3 — Άκυρο σύμβολο (π.χ. “X”).....21
Εκφώνηση..... 21
Απάντηση.....21
Σχόλια.....21
Άσκηση 4 — Κενή γραμμή στο world.txt.....21
Εκφώνηση..... 21
Απάντηση.....21
Σχόλια.....21
Άσκηση 5 — Γραμμή με πολλά κενά (περισσότερα spaces).....21
Εκφώνηση..... 21
Απάντηση.....22
Σχόλια.....22
Άσκηση 6 — Λάθος encoding (UTF-16).....22

Εκφώνηση..... 22
Απάντηση..... 22
Σχόλια..... 22
Άσκηση 7 — Περίσσειες στήλες λόγω copy-paste από PDF..... 22
Εκφώνηση..... 22
Απάντηση..... 22
Σχόλια..... 22
Άσκηση 8 — Χρήση tab αντί για space..... 22
Εκφώνηση..... 22
Απάντηση..... 23
Σχόλια..... 23
Άσκηση 9 — Χρήση ελληνικών πλήκτρων για “.” (το γνωστό λάθος “·”)..... 23
Εκφώνηση..... 23
Απάντηση..... 23
Σχόλια..... 23
Άσκηση 10 — Δεν υπάρχουν καθόλου αντικείμενα “*”..... 23
Εκφώνηση..... 23
Απάντηση..... 23
Σχόλια..... 23
Άσκηση 11 — Όλα τα κελιά είναι “#”..... 23
Εκφώνηση..... 23
Απάντηση..... 24
Σχόλια..... 24
Άσκηση 12 — Το start (0,0) είναι εμπόδιο “#”..... 24
Εκφώνηση..... 24
Απάντηση..... 24
Σχόλια..... 24
Άσκηση 13 — Το αρχείο έχει 12 γραμμές λόγω copy-paste..... 24
Εκφώνηση..... 24
Απάντηση..... 24
Σχόλια..... 24
Άσκηση 14 — Αντιμετάθεση συμβόλων (“.” αντί “*”)..... 25
Εκφώνηση..... 25
Απάντηση..... 25
Σχόλια..... 25
Άσκηση 15 — Γραμμή με trailing spaces..... 25
Εκφώνηση..... 25
Απάντηση..... 25
Σχόλια..... 25
Άσκηση 16 — Εμφάνιση BOM (Byte Order Mark)..... 25
Εκφώνηση..... 25
Απάντηση..... 26
Σχόλια..... 26
Άσκηση 17 — Χρήση “O” αντί για “0” όπου γράφονται αριθμοί στη βαθμολόγηση..... 26
Εκφώνηση..... 26
Απάντηση..... 26
Σχόλια..... 26
Άσκηση 18 — Πολύ μεγάλα διαδοχικά εμπόδια..... 26
Εκφώνηση..... 26
Απάντηση..... 26
Σχόλια..... 26
Άσκηση 19 — Μόνο μία γραμμή έχει “*” αλλά ένας φοιτητής έβαλε δύο κενά ανάμεσα..... 26

Εκφώνηση..... 26
Απάντηση..... 27
Άσκηση 20 — Αντικείμενο εκτός ορίων (δηλαδή λανθασμένη διάσταση)..... 27
Εκφώνηση..... 27
Απάντηση..... 27
Τι μάθαμε με αυτά τα debugging tasks;..... 27
Αποφυγή Παγίδων όταν Δημιουργείτε world.txt..... 28
1. Παγίδα: Λάθος διαστάσεις (9×10, 10×9, 11×10)..... 28
Τι πάει στραβά..... 28
✔ Τι κάνουμε..... 28
2. Παγίδα: Άκυρο σύμβολο..... 28
Πρόβλημα..... 28
✔ Λύση..... 29
3. Παγίδα: Κενές γραμμές ή extra newlines..... 29
Πρόβλημα..... 29
✔ Λύση..... 29
4. Παγίδα: Αντιγραφή από PDF / Word..... 29
✔ Λύση..... 29
5. Παγίδα: UTF-16 ή ANSI encoding..... 29
Αν το αρχείο είναι UTF-16 → UnicodeDecodeError..... 30
✔ Λύση..... 30
6. Παγίδα: Tabs αντί για spaces..... 30
✔ Λύση..... 30
7. Παγίδα: Πολύ μικρά ή πολύ μεγάλα blocks εμποδίων..... 30
✔ Λύση..... 30
8. Παγίδα: Αντικείμενα σε αδιέξοδο χωρίς μονοπάτι..... 30
✔ Τι να προσέξουμε..... 31
9. Παγίδα: Το start (0,0) είναι εμπόδιο..... 31
✔ Λύση..... 31
10. Παγίδα: Το world είναι “πολύ εύκολο” ή “πολύ δύσκολο”..... 31
Τυπικά λάθη..... 31
✔ Λύση..... 31
11. Παγίδα: Εισαγωγή σχολίων στο world.txt..... 32
Προβλημα..... 32
✔ Λύση..... 32
12. Παγίδα: Hidden characters (BOM \ufeff)..... 32
✔ Λύση..... 32
13. Παγίδα: Μη σταθερό μήκος γραμμών..... 32
✔ Λύση..... 32
14. Παγίδα: Χρήση λατινικού O αντί για αριθμητικό 0..... 33
15. Παγίδα: Copy/paste από Mac Notes ή iCloud Notes..... 33
✔ Λύση..... 33
16. Παγίδα: Παρανόηση ότι πρέπει να υπάρχει συγκεκριμένος αριθμός αντικειμένων..... 33
✔ Συμβουλή..... 33
17. Παγίδα: Αντιμετάθεση χαρακτήρων κατά λάθος..... 33
✔ Σωστό format..... 33
18. Παγίδα: Ενσωμάτωση tabs στο τέλος γραμμής..... 34
19. Παγίδα: Extra symbols λόγω auto-correct (Mac / smartphones)..... 34
✔ Έλεγχος..... 34
20. Παγίδα: Ο κόσμος έχει unreachable περιοχές που δεν φαίνονται..... 34
✔ Συμβουλή..... 34
Συνολικός Οδηγός Best Practices για world.txt..... 34

✔ Χρησιμοποιήστε πάντα plain-text editor....34
✔ Υποχρεωτικά 10 γραμμές × 10 tokens....35
✔ Μόνο ".", "#", "*"....35
✔ Save as UTF-8....35
✔ Οπτικός έλεγχος πριν από testing....35
✔ Τεστ με αυτοματοποιημένο validator....35
Mini-Problem 1 — Επιστροφή όλων των έγκυρων γειτόνων....36
Εκφώνηση....36
Λύση....36
Σχόλια (Ελληνικά)....36
Mini-Problem 2 — Check αν υπάρχει τουλάχιστον ένας γείτονας....36
Εκφώνηση....36
Λύση....37
Σχόλια....37
Mini-Problem 3 — Καταμέτρηση προσβάσιμων κελιών γύρω από θέση....37
Εκφώνηση....37
Λύση....37
Σχόλια....37
Mini-Problem 4 — Σάρωση γραμμής αριστερά → δεξιά....37
Εκφώνηση....37
Λύση....37
Σχόλια....37
Mini-Problem 5 — Σάρωση στήλης πάνω → κάτω....38
Εκφώνηση....38
Λύση....38
Σχόλια....38
Mini-Problem 6 — Εύρεση της πρώτης εμφάνισης "*“....38
Εκφώνηση....38
Λύση....38
Σχόλλια....38
Mini-Problem 7 — Υπολογισμός Manhattan distance από το ρομπότ....38
Εκφώνηση....38
Λύση....39
Σχόλια....39
Mini-Problem 8 — Επιστροφή όλων των κενών κελιών....39
Εκφώνηση....39
Λύση....39
Σχόλια....39
Mini-Problem 9 — Έλεγχος αν κελί είναι dead-end....39
Εκφώνηση....39
Λύση....39
Σχόλια....40
Mini-Problem 10 — Εντοπισμός “pockets” (κελιά περικυκλωμένα από εμπόδια)....40
Εκφώνηση....40
Λύση....40
Σχόλια....40

Παρακάτω σου δίνω **10 ολοκληρωμένες ασκήσεις (εκφώνηση–απάντηση–λύση)** ειδικά σχεδιασμένες για την **Εργασία 2 – Ρομπότ Συλλέκτης** του μαθήματος , ώστε:

- να εξασκούν **2D λίστες**,
- να είναι **χρήσιμες** για το project (load_world, scanning, neighbors, object detection, obstacles, movement rules),
- αλλά **να μη δίνουν ούτε να υπονοούν τη λύση BFS**.

Κάθε άσκηση περιλαμβάνει:

✔ Εκφώνηση
✔ Λύση με κώδικα **στα αγγλικά**
✔ Αναλυτική τεκμηρίωση & σχόλια **στα ελληνικά**

Άσκηση 1 — Έλεγχος εγκυρότητας κελιού (in-bounds + obstacle check)

Εκφώνηση

Γράψτε συνάρτηση `valid(world, r, c)` που επιστρέφει `True` όταν:

- το κελί (r,c) είναι εντός ορίων 10×10, και
- **δεν** είναι εμπόδιο (#).

Αλλιώς επιστρέφει `False`.

Λύση

```
def valid(world, r, c):
    ROWS, COLS = len(world), len(world[0])
    if 0 <= r < ROWS and 0 <= c < COLS:
        return world[r][c] != "#"
    return False
```

Σχόλια

- Εκπαιδεύει στη βασική αποστολή του ρομπότ: *κίνηση μόνο σε επιτρεπτά κελιά*.
- Ισοδύναμη του `in_bounds` + obstacle check από το επίσημο skeleton.
- Θεμέλιο για neighbors, scan και έλεγχο μονοπατιών.

Άσκηση 2 — Μέτρηση αντικειμένων σε grid

Εκφώνηση

Να υλοποιήσετε συνάρτηση `count_items(world)` που μετρά πόσα `"*"` υπάρχουν στο πλέγμα.

Λύση

```
def count_items(world):
    total = 0
    for row in world:
        for cell in row:
            if cell == "*":
                total += 1
    return total
```

Σχόλια

- Χρήσιμη για **έλεγχο τερματισμού** (κανόνας 1 της άσκησης).
- Δείχνει πώς γίνεται πλήρης διάσχιση 2D λίστας.

Άσκηση 3 — Εύρεση όλων των αντικειμένων

Εκφώνηση

Γράψτε συνάρτηση `find_items(world)` που επιστρέφει λίστα με όλες τις θέσεις `(r,c)` όπου υπάρχει `"*"`.

Λύση

```
def find_items(world):
    items = []
    for r in range(len(world)):
        for c in range(len(world[0])):
            if world[r][c] == "*":
                items.append((r, c))
    return items
```

Σχόλια

- ικανότητα *χαρτογράφησης* του κόσμου.
- Χρήσιμο για debugging (“πού είναι τα αντικείμενα του world.txt;”).

Άσκηση 4 — Εμφάνιση κόσμου με ένδειξη θέσης ρομπότ

Εκφώνηση

Να γραφτεί συνάρτηση `show(world, r, c)` που εκτυπώνει το πλέγμα, με `"R"` στη θέση του ρομπότ.

Λύση

```
def show(world, r, c):
    for i in range(len(world)):
        line = []
        for j in range(len(world[0])):
            line.append("R" if (i,j)==(r,c) else world[i][j])
        print(" ".join(line))
    print()
```

Σχόλια

- Ισούται με `print_world` του skeleton.
- Στόχος: οπτικοποίηση 2D λιστών.

Άσκηση 5 — Λίστα με όλους τους γείτονες (4 κατευθύνσεις)

Εκφώνηση

Υλοποιήστε συνάρτηση `all_neighbors(world, r, c)` που επιστρέφει τους 4 γείτονες με σειρά **πάνω–κάτω–αριστερά–δεξιά**, μόνο αν δεν είναι εμπόδια.

Λύση

```
def all_neighbors(world, r, c):
    moves = [(-1,0), (1,0), (0,-1), (0,1)]
    result = []
    for dr,dc in moves:
        nr, nc = r+dr, c+dc
        if 0 <= nr < len(world) and 0 <= nc < len(world[0]):
            if world[nr][nc] != "#":
                result.append((nr,nc))
    return result
```

Σχόλια

- Είναι **πλήρης προετοιμασία** για BFS αλλά δεν αποκαλύπτει BFS.
- Η εργασία βασίζεται απολύτως στην έννοια των γειτόνων.

Άσκηση 6 — Αλλαγή κελιού (marking)

Εκφώνηση

Να φτιαχτεί συνάρτηση `mark(world, r, c, symbol)` που αλλάζει τιμή στο κελί `(r,c)`.

Λύση

```
def mark(world, r, c, symbol):
    world[r][c] = symbol
```

Σχόλια

- οι 2D λίστες είναι **mutable**.
- Χρήσιμο για σηματοδότηση visited, debugging κ.λπ.

Άσκηση 7 — Δημιουργία άδειου κόσμου 10×10

Εκφώνηση

Γράψτε συνάρτηση `empty_world()` που επιστρέφει 10×10 πλέγμα γεμάτο `"."`.

Λύση

```
def empty_world():
    return [["." for _ in range(10)] for _ in range(10)]
```

Σχόλια

- Ενισχύει κατανόηση **list comprehension** και κατασκευής 2D λιστών.
- Χρήσιμο για unit testing.

Άσκηση 8 — Μέτρηση εμποδίων

Εκφώνηση

Να γραφτεί συνάρτηση `count_obstacles(world)` που επιστρέφει πόσα `"#"` υπάρχουν.

Λύση

```
def count_obstacles(world):
    total = 0
    for row in world:
        for cell in row:
            if cell == "#":
                total += 1
    return total
```

Σχόλια

- Σχετικό με την προσομοίωση, γιατί διαφορετικά worlds έχουν διαφορετική δυσκολία.
- Δεν χρησιμοποιεί καμία λογική μονοπατιού.

Άσκηση 9 — Αντιστροφή στοιχείων γραμμής

Εκφώνηση

Γράψτε συνάρτηση `reverse_row(world, r)` που αντιστρέφει όλα τα στοιχεία της γραμμής `r`.

Λύση

```
def reverse_row(world, r):
    world[r] = world[r][::-1]
```

Σχόλια

- Εκπαιδεύει στη χρήση slicing και μεταβολών εντός 2D λίστας.
- Χρήσιμο για πειραματισμό/οπτική κατανόηση του grid.

Άσκηση 10 — Έλεγχος αν υπάρχει αντικείμενο σε γραμμή ή στήλη

Εκφώνηση

Γράψτε συνάρτηση `contains_item_in_row_or_col(world, r, c)` που:

- επιστρέφει `True` αν στη **γραμμή r** ή στη **στήλη c** υπάρχει τουλάχιστον ένα `"*"`.

Λύση

```
def contains_item_in_row_or_col(world, r, c):
    # check row r
    if "*" in world[r]:
        return True
    # check column c
    for row in world:
        if row[c] == "*":
            return True
    return False
```

Σχόλια

- Εξάσκηση scanning σε 2D grid κατά γραμμή και κατά στήλη.
- Χρήσιμο για κατανόηση οπτικών ελέγχων και scanning heuristics.

Άσκηση 1 — Εντοπισμός “νησιών” εμποδίων χωρίς χρήση BFS

Εκφώνηση

Δώστε συνάρτηση που μετρά πόσα “μπλοκ” εμποδίων υπάρχουν, δηλαδή πόσες ομάδες συνεχόμενων # που συνδέονται οριζόντια/κάθετα.

Απαγορεύεται BFS/DFS.
Αντί αυτού, να χρησιμοποιηθεί recursive flood-fill.

Λύση

```
def count_obstacle_islands(world):
    seen = [[False]*len(world[0]) for _ in world]
    ROWS, COLS = len(world), len(world[0])

    def fill(r, c):
        if not (0 <= r < ROWS and 0 <= c < COLS):
            return
        if seen[r][c] or world[r][c] != "#":
            return
        seen[r][c] = True
        fill(r-1,c); fill(r+1,c); fill(r,c-1); fill(r,c+1)

    islands = 0
    for r in range(ROWS):
        for c in range(COLS):
            if world[r][c] == "#" and not seen[r][c]:
                islands += 1
                fill(r, c)

    return islands
```

Σχόλια

- Ενισχύει την έννοια *connected components σε grid*.
- Μαθαίνει πώς να “ξεχωρίζουν” περιοχές στο world.txt.
- Παράλληλη δεξιότητα με BFS αλλά **χωρίς τη χρήση BFS**.

Άσκηση 2 — Ανίχνευση “διαδρόμων”

Εκφώνηση

Γράψτε συνάρτηση που βρίσκει όλα τα κελιά `.` που έχουν **ακριβώς 2** προσβάσιμους γείτονες (δηλαδή αποτελούν “διάδρομο”).

Λύση

```
def corridor_cells(world):
    ROWS, COLS = len(world), len(world[0])
    corridors = []

    for r in range(ROWS):
        for c in range(COLS):
            if world[r][c] != ".":
                continue
            neighbors = 0
            for dr,dc in [(-1,0),(1,0),(0,-1),(0,1)]:
                nr, nc = r+dr, c+dc
                if 0 <= nr < ROWS and 0 <= nc < COLS:
                    if world[nr][nc] != "#":
                        neighbors += 1
            if neighbors == 2:
                corridors.append((r,c))
    return corridors
```

Σχόλια

- Χρήσιμο για να εντοπίζει ο φοιτητής στένωση/διάδρομο στο world.
- Βοηθά στην κατανόηση της δομής του grid πριν εφαρμοστεί BFS.

Άσκηση 3 — Εντοπισμός “αδιεξόδων”

Εκφώνηση

Βρείτε όλα τα κελιά `.` με **μόνο έναν** προσβάσιμο γείτονα (dead ends).

Λύση

```
def dead_ends(world):
    ends = []
    ROWS, COLS = len(world), len(world[0])

    for r in range(ROWS):
        for c in range(COLS):
            if world[r][c] != ".":
                continue
            open_neighbors = 0
            for dr,dc in [(-1,0),(1,0),(0,-1),(0,1)]:
                nr, nc = r+dr, c+dc
```

```
                if 0 <= nr < ROWS and 0 <= nc < COLS:
                    if world[nr][nc] != "#":
                        open_neighbors += 1
            if open_neighbors == 1:
                ends.append((r,c))
    return ends
```

Σχόλια

- Συσχετίζεται με “path pruning”.
- Σημαντικό για οπτική κατανόηση των μονοπατιών.

Άσκηση 4 — Εντοπισμός αντικειμένων που βρίσκονται σε αδιέξοδο

Εκφώνηση

Γράψτε συνάρτηση που επιστρέφει όλα τα αντικείμενα `"*"` που βρίσκονται σε dead-end cells.

Λύση

```
def items_in_dead_ends(world):
    result = []
    ROWS, COLS = len(world), len(world[0])

    for r in range(ROWS):
        for c in range(COLS):
            if world[r][c] != "*":
                continue
            neighbors = 0
            for dr,dc in [(-1,0),(1,0),(0,-1),(0,1)]:
                nr, nc = r+dr, c+dc
                if 0 <= nr < ROWS and 0 <= nc < COLS:
                    if world[nr][nc] != "#":
                        neighbors += 1
            if neighbors == 1:
                result.append((r,c))
    return result
```

Σχόλια

- Δείχνει *ποια αντικείμενα είναι δύσκολο να φτάσει το ρομπότ*.
- Χρήσιμο για αναλύσεις world.txt.

Άσκηση 5 — Υπολογισμός απόστασης Manhattan ανάμεσα στο ρομπότ και όλα τα αντικείμενα

Εκφώνηση

Υλοποιήστε συνάρτηση που επιστρέφει λεξικό:
`(r,c) -> Manhattan_distance_from_robot`.

Λύση

```
def manhattan_map(world, robot_r, robot_c):
    d = {}
    for r in range(len(world)):
        for c in range(len(world[0])):
            if world[r][c] == "*":
                d[(r,c)] = abs(r - robot_r) + abs(c - robot_c)
    return d
```

Σχόλια

- Η Manhattan distance συσχετίζεται έμμεσα με BFS (όχι ίδιο πράγμα!).
- Εκπαιδεύει στην αναζήτηση στόχων.

Άσκηση 6 — Εντοπισμός “ορατών” αντικειμένων σε ευθεία γραμμή

Εκφώνηση

Να γραφτεί συνάρτηση που βρίσκει αν το ρομπότ “βλέπει” κάποιο αντικείμενο `"*"` στην ίδια γραμμή **χωρίς εμπόδια ανάμεσα**.

Λύση

```
def visible_in_row(world, r, c):
    # check left
    for cc in range(c-1, -1, -1):
        if world[r][cc] == "#":
            break
        if world[r][cc] == "*":
            return True
    # check right
    for cc in range(c+1, len(world[0])):
        if world[r][cc] == "#":
            break
        if world[r][cc] == "*":
```

```
            return True
    return False
```

Σχόλια

- Σχετίζεται με concept “line-of-sight”.
- Βοηθά κατανόηση interactions εμποδίων/ορατότητας.

Άσκηση 7 — Μετατροπή 2D grid σε heatmap εμποδίων

Εκφώνηση

Να δημιουργηθεί νέα 2D λίστα ίδιων διαστάσεων, όπου κάθε θέση `(r,c)` περιέχει τον αριθμό εμποδίων στα 4 γειτονικά κελιά της.

Λύση

```
def obstacle_heatmap(world):
    ROWS, COLS = len(world), len(world[0])
    heat = [[0]*COLS for _ in range(ROWS)]

    for r in range(ROWS):
        for c in range(COLS):
            count = 0
            for dr,dc in [(-1,0),(1,0),(0,-1),(0,1)]:
                nr, nc = r+dr, c+dc
                if 0 <= nr < ROWS and 0 <= nc < COLS:
                    if world[nr][nc] == "#":
                        count += 1
            heat[r][c] = count
    return heat
```

Σχόλια

- Εξαιρετική άσκηση για “feature extraction” από grids.
- Βοηθά κατανόηση τοπολογίας.

Άσκηση 8 — Αντικατάσταση περιγράμματος grid με εμπόδια

Εκφώνηση

Δώστε συνάρτηση που μετατρέπει όλα τα κελιά του εξωτερικού περιγράμματος (r=0, r=9, c=0, c=9) σε `"#"`.

Λύση

```
def add_borders(world):
    ROWS, COLS = len(world), len(world[0])
    for c in range(COLS):   # top row
        world[0][c] = "#"
    for c in range(COLS):   # bottom row
        world[ROWS-1][c] = "#"
    for r in range(ROWS):   # left col
        world[r][0] = "#"
    for r in range(ROWS):   # right col
        world[r][COLS-1] = "#"
```

Σχόλια

- να επεξεργάζονται ολόκληρες γραμμές/στήλες.
- Χρήσιμο για προσομοίωση περιορισμένων χώρων.

Άσκηση 9 — Μετακίνηση ρομπότ με απλή heuristic (όχι BFS)

Εκφώνηση

Να υλοποιηθεί συνάρτηση `step_towards(world, r, c, target_r, target_c)` που επιστρέφει **μία** κίνηση που μειώνει τη Manhattan απόσταση, χωρίς να κοιτάζει εμπόδια.

Λύση

```
def step_towards(world, r, c, tr, tc):
    if tr < r: return (r-1, c)
    if tr > r: return (r+1, c)
    if tc < c: return (r, c-1)
    if tc > c: return (r, c+1)
    return (r, c)
```

Σχόλια

- Εκπαιδεύει σε απλές στρατηγικές που **δεν εγγυώνται** shortest path.

- Διαχωρίζει heuristic από BFS.

Άσκηση 10 — Εντοπισμός “pockets” (περιοχών περικυκλωμένων από εμπόδια)

Εκφώνηση

Ορίστε συνάρτηση που εντοπίζει όλα τα κελιά **" . "** τα οποία έχουν **τουλάχιστον 3 εμπόδια** γύρω τους.

Λύση

```
def pockets(world):
    ROWS, COLS = len(world), len(world[0])
    result = []

    for r in range(ROWS):
        for c in range(COLS):
            if world[r][c] != ".":
                continue
            count = 0
            for dr,dc in [(-1,0),(1,0),(0,-1),(0,1)]:
                nr, nc = r+dr, c+dc
                if 0 <= nr < ROWS and 0 <= nc < COLS:
                    if world[nr][nc] == "#":
                        count += 1
            if count >= 3:
                result.append((r,c))
    return result
```

Σχόλια

- Ενισχύει spatial reasoning για δυσκολία προσπέλασης περιοχών.
- Σχετίζεται με “narrow passages” όπου η BFS έχει μεγαλύτερο ρόλο.

Debugging Set – 20 Ασκήσεις για world.txt

Κάθε ασκηση εξετάζει 1 πραγματικό λάθος που συχνά εμφανίζεται στα submissions των φοιτητών.

Άσκηση 1 — Λάθος διαστάσεις (9×10)

Εκφώνηση

Δίνεται world.txt με 9 γραμμές αντί για 10. Εντόπισε το πρόβλημα και εξήγησε γιατί αποτυγχάνει η `load_world`.

Απάντηση

`load_world()` περιμένει ακριβώς **10 γραμμές**.
Με 9 γραμμές, η συνθήκη:

```
assert len(world) == ROWS
```

αποτυγχάνει.

Σχόλια

- Ο evaluator σταματάει άμεσα.
- Συχνό λάθος από φοιτητές που κάνουν copy-paste μία γραμμή κατά λάθος.

Άσκηση 2 — Μια γραμμή έχει 9 στοιχεία αντί για 10

Εκφώνηση

Η 3η γραμμή περιέχει:
`. . . # . . . . *` (εννέα στοιχεία).

Απάντηση

Παραβίαση:

```
all(len(r) == COLS for r in world)
```

Άρα AssertionError “Λάθος διαστάσεις world.txt”.

Σχόλια

- Μπορεί να συμβεί όταν ένας φοιτητής ξεχάσει ένα “.” στο τέλος.

Άσκηση 3 — Άκυρο σύμβολο (π.χ. “X”)

Εκφώνηση

Σε μία θέση εμφανίζεται X. Ποιο validation αποτυγχάνει;

Απάντηση

```
assert s in valid
```

όπου `valid = {".", "#", "*"}`.

Σχόλια

- Συμβαίνει συχνά όταν κάποιος κάνει replace σε Notepad/Word.

Άσκηση 4 — Κενή γραμμή στο world.txt

Εκφώνηση

Ανάμεσα στις γραμμές υπάρχει μια κενή (empty newline).

Απάντηση

Η `.split()` δίνει κενή λίστα → δεν προστίθεται.
Αποτέλεσμα: 9 ή 11 γραμμές → AssertionError.

Σχόλια

- Οι κενές γραμμές απαγορεύονται.

Άσκηση 5 — Γραμμή με πολλά κενά (περισσότερα spaces)

Εκφώνηση

Μια γραμμή έχει σύμβολα με διπλά και τριπλά κενά.

Απάντηση

Δεν υπάρχει θέμα: `line.split()` αγνοεί πολλαπλά κενά → σωστά tokens.

Σχόλια

- Αυτή είναι *νόμιμη* μορφή world.txt.

Άσκηση 6 — Λάθος encoding (UTF-16)

Εκφώνηση

Ο φοιτητής έστειλε world.txt σε UTF-16. Τι γίνεται;

Απάντηση

Το `open(..., encoding="utf-8")` θα ρίξει `UnicodeDecodeError`.

Σχόλια

- Πολύ συχνό λάθος Windows Notepad.
- Λύση: Save as UTF-8.

Άσκηση 7 — Περίσσειες στήλες λόγω copy-paste από PDF

Εκφώνηση

Μία γραμμή περιέχει “.”, “#”, αλλά και αόρατους unicode χαρακτήρες.

Απάντηση

Το split() θα δώσει παραπάνω tokens → invalid length → AssertionError.

Σχόλια

- Χρήσιμη άσκηση γιατί πολλοί κάνουν copy από το PDF.

Άσκηση 8 — Χρήση tab αντί για space

Εκφώνηση

Οι στήλες χωρίζονται με TAB.

Απάντηση

`split()` χωρίζει και σε tabs → όλα λειτουργούν σωστά, *αν* είναι 10 tokens.

Σχόλια

- Δεν αποτελεί πρόβλημα.

Άσκηση 9 — Χρήση ελληνικών πλήκτρων για “.” (το γνωστό λάθος “·”)

Εκφώνηση

Ο φοιτητής βάζει ελληνικό middle dot (“·”) αντί για “.”.

Απάντηση

Η `assert s in valid` θα αποτύχει.

Σχόλια

- Πολύ συχνό λάθος από keyboards macOS/Windows.

Άσκηση 10 — Δεν υπάρχουν καθόλου αντικείμενα “*”

Εκφώνηση

Το world έχει μόνο “.” και “#”. Τι συμβαίνει;

Απάντηση

Ο κώδικας `run()` θα τρέξει 80 βήματα χωρίς να μαζέψει τίποτα.

Σχόλια

- Επιτρεπτό world, απλά σκορ = 0.

Άσκηση 11 — Όλα τα κελιά είναι “#”

Εκφώνηση

Τι συμβαίνει αν ο κόσμος είναι ένα πλέγμα γεμάτο εμπόδια;

Απάντηση

Το ρομπότ δεν μπορεί να μετακινηθεί.
`next_step_bfs` (ή sweep) θα επιστρέψει την ίδια θέση.

Σχόλια

- Επιτρεπτό world, απλά non-playable.

Άσκηση 12 — Το start (0,0) είναι εμπόδιο “#”

Εκφώνηση

Επιτρέπεται το world[0][0] = “#”; Τι συμβαίνει;

Απάντηση

Επιτρέπεται.
Το ρομπότ ξεκινά πάνω σε εμπόδιο — απλώς δεν μπορεί να φύγει.

Σχόλια

- Δεν υπάρχει validation σε αυτό στο load_world.

Άσκηση 13 — Το αρχείο έχει 12 γραμμές λόγω copy-paste

Εκφώνηση

Πώς εντοπίζεται αυτό;

Απάντηση

Η assert `len(world) == ROWS` αποτυγχάνει.

Σχόλια

- Πρέπει να διαγράψει ο φοιτητής τις επιπλέον γραμμές.

Άσκηση 14 — Αντιμετάθεση συμβόλων (“.” αντί “*”)

Εκφώνηση

Φοιτητής γράφει:

```
. . * . .
. . . . .
. # . . .
...
```

αλλά εννοεί “*” σε άλλη θέση.

Απάντηση

Δεν αποτελεί syntax error.
Το πρόβλημα είναι **λογικό** και δύσκολο να διαγνωστεί.

Σχόλια

- Debugging μέσω visualization `print_world`.

Άσκηση 15 — Γραμμή με trailing spaces

Εκφώνηση

Μία γραμμή τελειώνει με πολλά κενά.

Απάντηση

`strip()` τα αφαιρεί → split() δουλεύει κανονικά.

Σχόλια

- Νόμιμο world.txt.

Άσκηση 16 — Εμφάνιση BOM (Byte Order Mark)

Εκφώνηση

Πολλά editors κρατούν BOM στην πρώτη γραμμή. Τι συμβαίνει;

Απάντηση

Το πρώτο σύμβολο ίσως διαβαστεί ως `"\ufeff."` → άκυρο σύμβολο.

Σχόλια

- Το assertion “Άκυρο σύμβολο” θα πιάσει το λάθος.

Άσκηση 17 — Χρήση “Ο” αντί για “0” όπου γράφονται αριθμοί στη βαθμολόγηση

Εκφώνηση

Εάν κάποιος θέλει να σημειώσει από πάνω “10 items” και γράψει “IO items”, τι θα γίνει;

Απάντηση

Η γραμμή θα γίνει μέρος του world → θα αποτύχει το validation.

Σχόλια

- Το world.txt *δεν πρέπει να έχει σχόλια*.

Άσκηση 18 — Πολύ μεγάλα διαδοχικά εμπόδια

Εκφώνηση

Υπάρχουν 10 εμπόδια στη σειρά (ολόκληρη γραμμή). Είναι λάθος;

Απάντηση

Όχι, είναι απολύτως νόμιμο.

Σχόλια

- Δεν υπάρχει περιορισμός στη μορφολογία.

Άσκηση 19 — Μόνο μία γραμμή έχει “*” αλλά ένας φοιτητής έβαλε δύο κενά ανάμεσα

Εκφώνηση

Η γραμμή:

. . . * . . . .

έχει διπλά κενά. Πρόβλημα;

Απάντηση

Όχι. `split()` το διορθώνει.

Άσκηση 20 — Αντικείμενο εκτός ορίων (δηλαδή λανθασμένη διάσταση)

Εκφώνηση

Αν το world κάνει wrap-around (π.χ. 10 στοιχεία αλλά ένα είναι "κρυμμένο" στο τέλος της γραμμής), τι γίνεται;

Απάντηση

`split()` συλλαμβάνει μόνο τα tokens που χωρίζονται από whitespace.
Αν υπάρχει extra κρυφό unicode char → invalid symbol.

Τι μάθαμε με αυτά τα debugging tasks;

Οι φοιτητές αποκτούν δεξιότητες:

- ✔ Validation world.txt
- ✔ Εντοπισμός syntax errors σε 2D grids
- ✔ Κατανόηση πώς λειτουργεί `load_world` και `assert`
- ✔ Εντοπισμός unicode/encoding προβλημάτων
- ✔ Κατανόηση σωστής μορφής input για grid-based προσομοίωση
- ✔ Αποφυγή παγίδων όταν ετοιμάζουν δικά τους worlds για testing

---

Αποφυγή Παγίδων όταν Δημιουργείτε world.txt

Η εργασία βασίζεται σε **αυστηρή μορφοποίηση του world.txt**, και κάθε μικρή απόκλιση προκαλεί:

- AssertionError
- Unicode errors
- λάθος διάβασμα grid
- κακή συμπεριφορά του robot
- αδυναμία αναπαραγωγής debugging

Γι’ αυτό ακολουθεί ένας **αναλυτικός οδηγός best practices** και οι **20 πιο κρίσιμες παγίδες** με λύσεις.

---

1. Παγίδα: Λάθος διαστάσεις (9×10, 10×9, 11×10)

Τι πάει στραβά

Η `load_world()` απαιτεί ακριβώς **10 γραμμές × 10 στήλες**.
Ακόμη και 1 λιγότερο/περισσότερο στοιχείο → immediate error.

✔ Τι κάνουμε

- Μετράμε κάθε γραμμή: πρέπει να έχει ακριβώς **10 tokens**.
- Μετράμε συνολικές γραμμές: ακριβώς **10**.

---

2. Παγίδα: Άκυρο σύμβολο

(`X`, `o`, `0`, `·`, `,`, κρυφοί unicode χαρακτήρες)

Πρόβλημα

Η εργασία επιτρέπει **μόνο**:

- `"."` κενό
- `"#"` εμπόδιο
- `"*"` αντικείμενο

Οτιδήποτε άλλο → AssertionError.

✔ Λύση

- Χρησιμοποιούμε plain text editor (VSCode, Notepad++, Sublime).
- Όχι Word, όχι PDF copy-paste.
- Όχι ελληνικό “·” αντί για “.”.

---

3. Παγίδα: Κενές γραμμές ή extra newlines

Πρόβλημα

Κενή γραμμή → μειώνεται ο αριθμός των tokens → failure.

✔ Λύση

- Η τελευταία γραμμή **δεν πρέπει** να έχει κενά.
- Δεν αφήνουμε κενές γραμμές στο τέλος.

---

4. Παγίδα: Αντιγραφή από PDF / Word

Δημιουργεί:

- αόρατους unicode characters
- διαφορετικά whitespace
- περίεργα newlines

✔ Λύση

Πάντα **χειροκίνητη καταχώρηση** ή επεξεργασία σε καθαρό editor (VSCode).

---

5. Παγίδα: UTF-16 ή ANSI encoding

Το skeleton ανοίγει το αρχείο ως:

```
open(path, "r", encoding="utf-8")
```

Αν το αρχείο είναι UTF-16 → UnicodeDecodeError

✔ Λύση

- File → Save with Encoding → UTF-8.

---

6. Παγίδα: Tabs αντί για spaces

Αν και επιτρεπτά, συχνά δημιουργούν extra tokens.

✔ Λύση

- Προτιμήστε πάντα **ένα space** μεταξύ συμβόλων.
- Χρησιμοποιήστε “Convert Indentation to Spaces”.

---

7. Παγίδα: Πολύ μικρά ή πολύ μεγάλα blocks εμποδίων

Αν και επιτρεπτά, δημιουργούν:

- αδιέξοδα
- unreachable περιοχές
- παράξενη συμπεριφορά sweeping robot

✔ Λύση

Ελέγχουμε structurally:

- όχι 10 εμπόδια στη σειρά χωρίς λόγο
- όχι “τοίχοι” που κόβουν τον χάρτη στα δύο
- όχι νησιά αντικειμένων μη προσβάσιμα

---

8. Παγίδα: Αντικείμενα σε αδιέξοδο χωρίς μονοπάτι

Δεν είναι λάθος world.txt, αλλά κάνει testing πολύ πιο δύσκολο.

✔ Τι να προσέξουμε

- Αν ένα `*` περιβάλλεται από 3 εμπόδια, πιθανό dead-end.
- Αν είναι πλήρως κλειστό, το BFS δεν θα φτάσει ποτέ σε αυτό.

Χρήσιμο debugging εργαλείο:

`neighbors(world,r,c)`

---

9. Παγίδα: Το start (0,0) είναι εμπόδιο

Επιτρέπεται, αλλά:

- Το robot δεν κινείται.
- Το test case είναι άχρηστο για BFS testing.

✔ Λύση

Ποτέ μην βάζετε `"#"` στο `(0,0)`.

---

10. Παγίδα: Το world είναι “πολύ εύκολο” ή “πολύ δύσκολο”

Τυπικά λάθη

- Όλα τα αντικείμενα στην ίδια γραμμή → χωρίς BFS.
- Όλα πίσω από εμπόδια → unreachable.

✔ Λύση

Φτιάξτε κόσμους με:

- 4–8 αντικείμενα,
- 2–3 moderate obstacles clusters,
- μερικά corridors,
- 1–2 dead ends (για δοκιμή robustness).

---

11. Παγίδα: Εισαγωγή σχολίων στο world.txt

Φοιτητές γράφουν:

```
. . . * . . . # . .   # row1
. * . . . # . . . .   # row2
```

Προβλημα

Τα σχόλια διαβάζονται σαν tokens → invalid symbol.

✔ Λύση

Απαγορεύονται σχόλια.
world.txt πρέπει να περιέχει **μόνο 10 γραμμές από tokens**.

---

12. Παγίδα: Hidden characters (BOM \ufeff)

Αν η πρώτη γραμμή ξεκινά με BOM:

```
\ufeff. . . * . . . . . .
```

→ το πρώτο token είναι `"\ufeff."` → invalid symbol.

✔ Λύση

- File → Save Without BOM
- VSCode: “Save with Encoding → UTF-8”.

---

13. Παγίδα: Μη σταθερό μήκος γραμμών

Παράδειγμα:

```
. . . . . . . . . *
. # . . . . . . . .
```

Η δεύτερη γραμμή έχει 9 tokens.

✔ Λύση

Χρησιμοποιήστε snippet για counting:

```
for i,row in enumerate(world):
    print(i, len(row))
```

---

14. Παγίδα: Χρήση λατινικού O αντί για αριθμητικό 0

Δεν αφορά το grid, αλλά αρκετοί φοιτητές μπερδεύουν:

- YAML paths
- logs
- filenames

Και τελικά δεν φορτώνει world.txt.

---

15. Παγίδα: Copy/paste από Mac Notes ή iCloud Notes

Παράγει fancy unicode spaces.

✔ Λύση

Πάντα plain text editor.

---

16. Παγίδα: Παρανόηση ότι πρέπει να υπάρχει συγκεκριμένος αριθμός αντικειμένων

Δεν υπάρχει περιορισμός στην εργασία.
Η τιμή score απλώς καταγράφεται.

✔ Συμβουλή

Για testing BFS, προτιμήστε 3–8 αντικείμενα.

---

17. Παγίδα: Αντιμετάθεση χαρακτήρων κατά λάθος

Π.χ. `. # . * → .#.*` επειδή αφαιρέθηκαν τα spaces.

✔ Σωστό format

Κάθε σύμβολο χωρίζεται με space.

---

18. Παγίδα: Ενσωμάτωση tabs στο τέλος γραμμής

Μερικοί editors έχουν tabs στο τέλος.
Αν το split() δεν τα αναγνωρίσει → token με TAB → invalid.

---

19. Παγίδα: Extra symbols λόγω auto-correct (Mac / smartphones)

“*” ίσως γίνεται “✱”, “.” γίνεται “•”.

✔ Έλεγχος

Αντιγράψτε το world.txt σε hex editor ή κάντε print repr:

```
print([repr(cell) for cell in row])
```

---

20. Παγίδα: Ο κόσμος έχει unreachable περιοχές που δεν φαίνονται

Μπορεί να είναι σωστό format, αλλά λογικά κακό world.

✔ Συμβουλή

Οπτικοποιήστε με χρώματα:

```
for row in world:
    print(" ".join(row))
```

ή με robovisualizer σε Python notebook.

Συνολικός Οδηγός Best Practices για world.txt

✔ Χρησιμοποιήστε πάντα plain-text editor

VSCode, PyCharm, Sublime, Notepad++.

✔ Υποχρεωτικά 10 γραμμές × 10 tokens

Με spaces ανάμεσα.

✔ Μόνο ".", "#", "*"

Καμία παρέκκλιση.

✔ Save as UTF-8

Χωρίς BOM.

✔ Οπτικός έλεγχος πριν από testing

Με απλή εκτύπωση ή με `print_world`.

✔ Τεστ με αυτοματοποιημένο validator

(Μπορώ να σου φτιάξω script!)

Ακολουθούν **10 mini-problems για neighbors & grid traversal**, ειδικά σχεδιασμένα για την εργασία του ρομπότ.
Κάθε άσκηση περιλαμβάνει:

- ✔ **Εκφώνηση**
- ✔ **Λύση με κώδικα στα Αγγλικά**
- ✔ **Αναλυτικά σχόλια στα Ελληνικά**

Είναι **σύντομα, στοχευμένα**, και χτίζουν δεξιότητες για 2D grids, neighbors, valid moves, scanning — **χωρίς να αποκαλύπτουν BFS**.

Mini-Problem 1 — Επιστροφή όλων των έγκυρων γειτόνων

Εκφώνηση

Υλοποιήστε συνάρτηση που επιστρέφει όλους τους 4-neighbors ενός κελιού `(r,c)` που είναι εντός ορίων και δεν είναι εμπόδια.

Λύση

```
def neighbors4(world, r, c):
    moves = [(-1,0),(1,0),(0,-1),(0,1)]
    res = []
    ROWS, COLS = len(world), len(world[0])
    for dr,dc in moves:
        nr, nc = r+dr, c+dc
        if 0 <= nr < ROWS and 0 <= nc < COLS:
            if world[nr][nc] != "#":
                res.append((nr,nc))
    return res
```

Σχόλια (Ελληνικά)

- Είναι η βάση για κατανόηση grid graphs.
- Απαραίτητη δεξιότητα για BFS χωρίς να το περιέχει.

Mini-Problem 2 — Check αν υπάρχει τουλάχιστον ένας γείτονας

Εκφώνηση

Γράψτε συνάρτηση που επιστρέφει `True` αν το `(r,c)` έχει τουλάχιστον έναν νόμιμο γείτονα.

Λύση

```
def has_neighbor(world, r, c):
    return len(neighbors4(world, r, c)) > 0
```

Σχόλια

- Καλλιεργεί κατανόηση αδιεξόδων (dead-end cells).
- Χρήσιμο για debugging world.txt.

Mini-Problem 3 — Καταμέτρηση προσβάσιμων κελιών γύρω από θέση

Εκφώνηση

Επιστρέψτε πόσα από τα 4 γειτονικά κελιά είναι προσβάσιμα (όχι εμπόδια).

Λύση

```
def count_open_neighbors(world, r, c):
    return len(neighbors4(world, r, c))
```

Σχόλια

- Χρήσιμο για εντοπισμό «διαδρόμων» ή «στενών περασμάτων».

Mini-Problem 4 — Σάρωση γραμμής αριστερά → δεξιά

Εκφώνηση

Επιστρέψτε λίστα με όλα τα στοιχεία της γραμμής `r` από αριστερά προς τα δεξιά.

Λύση

```
def scan_row(world, r):
    return world[r][:]
```

Σχόλια

- Αναπτύσσει δεξιότητες γρήγορης πρόσβασης σε 2D λίστες.

Mini-Problem 5 — Σάρωση στήλης πάνω → κάτω

Εκφώνηση

Επιστρέψτε όλα τα στοιχεία της στήλης `c`.

Λύση

```
def scan_col(world, c):
    return [world[r][c] for r in range(len(world))]
```

Σχόλια

- Χρήσιμο για οπτική κατανόηση κάθετης διάταξης στο grid.

Mini-Problem 6 — Εύρεση της πρώτης εμφάνισης "*“

Εκφώνηση

Επιστρέψτε το πρώτο (r,c) όπου υπάρχει `"*"` καθώς σαρώνετε το grid row-major (γραμμές πρώτα).

Λύση

```
def first_item(world):
    for r in range(len(world)):
        for c in range(len(world[0])):
            if world[r][c] == "*":
                return (r,c)
    return None
```

Σχόλλια

- Γνωρίζουν τη σειρά row-major traversal που χρειάζεται η BFS.

Mini-Problem 7 — Υπολογισμός Manhattan distance από το ρομπότ

Εκφώνηση

Επιστρέψτε τη Manhattan απόσταση του (r,c) από το (robot_r, robot_c).

Λύση

```
def manhattan(r, c, rr, rc):
    return abs(r - rr) + abs(c - rc)
```

Σχόλια

- Βασικό μαθηματικό εργαλείο για grid heuristics.
- Δεν είναι pathfinding, άρα επιτρέπεται.

Mini-Problem 8 — Επιστροφή όλων των κενών κελιών

Εκφώνηση

Γράψτε συνάρτηση που βρίσκει όλα τα `"."` στο grid και επιστρέφει τη λίστα θέσεων.

Λύση

```
def find_empty_cells(world):
    result = []
    for r in range(len(world)):
        for c in range(len(world[0])):
            if world[r][c] == ".":
                result.append((r,c))
    return result
```

Σχόλια

- Πολύ χρήσιμο για testing patterns, οπτικοποίηση & debugging.

Mini-Problem 9 — Έλεγχος αν κελί είναι dead-end

Εκφώνηση

Ένα κελί θεωρείται dead-end αν έχει **ακριβώς 1** νόμιμο γείτονα.
Υλοποιήστε συνάρτηση που το ελέγχει.

Λύση

```
def is_dead_end(world, r, c):
    return count_open_neighbors(world, r, c) == 1
```

Σχόλια

- Καλλιεργεί κατανόηση τοπολογίας grid και στενών περασμάτων.

Mini-Problem 10 — Εντοπισμός “pockets” (κελιά περικυκλωμένα από εμπόδια)

Εκφώνηση

Επιστρέψτε όλα τα κελιά `"."` που έχουν **τουλάχιστον 3 εμπόδια** γύρω τους.

Λύση

```
def pockets(world):
    res = []
    ROWS, COLS = len(world), len(world[0])
    for r in range(ROWS):
        for c in range(COLS):
            if world[r][c] != ".":
                continue
            obst = 0
            for dr,dc in [(-1,0),(1,0),(0,-1),(0,1)]:
                nr, nc = r+dr, c+dc
                if 0 <= nr < ROWS and 0 <= nc < COLS:
                    if world[nr][nc] == "#":
                        obst += 1
            if obst >= 3:
                res.append((r,c))
    return res
```

Σχόλια

- Χρήσιμο για ανάλυση “δύσκολων περιοχών” στο grid.
- Συσχετίζεται με διάταξη εμποδίων και δυσκολία BFS.
