# Validate World : Έλεγχος ορθότητας αρχείου world.txt

Επιμέλεια: Α.Δρακόπουλος

# Πίνακας περιεχομένων

Ακολουθεί **ένα validator script** για την εργασία του ρομπότ, το οποίο:

- ελέγχει **αυτόματα** το world.txt πριν τρέξει η προσομοίωση,
- εντοπίζει **όλα τα γνωστά λάθη** (συμβολισμούς, διαστάσεις, κρυφά unicode, BOM, άδειες γραμμές κτλ.),
- δίνει **αναλυτικά μηνύματα λαθών** για φοιτητές,

---

# VALIDATOR SCRIPT — world_validator.py

```
import sys

VALID_SYMBOLS = {".", "#", "*"}

def validate_world(path="world.txt", rows=10, cols=10):
    """
    Validate world.txt according to the rules of the assignment.

    Returns:
        (ok, errors)
        ok: True/False
        errors: list of error-messages
    """

    errors = []

    # ----------------------------
    # STEP 1 – Διαβάζουμε το αρχείο raw
    # ----------------------------
    try:
        with open(path, "r", encoding="utf-8") as f:
            raw_lines = f.readlines()
    except UnicodeDecodeError:
        errors.append(" Encoding error: Το αρχείο ΔΕΝ είναι UTF-8.")
        return False, errors
    except FileNotFoundError:
        errors.append(" Το αρχείο world.txt δεν βρέθηκε.")
        return False, errors

    # ----------------------------
    # STEP 2 – Αφαίρεση newlines
    # ----------------------------
    lines = [line.rstrip("\n") for line in raw_lines]

    # ----------------------------
    # STEP 3 – Έλεγχος για BOM
    # ----------------------------
    if lines and lines[0].startswith("\ufeff"):
        errors.append(" Εντοπίστηκε BOM (\\ufeff) στην πρώτη γραμμή. \
Αποθηκεύστε ως UTF-8 χωρίς BOM.")
        # καθαρίζουμε για να συνεχίσει ο validator
        lines[0] = lines[0].replace("\ufeff", "")

    # ----------------------------
    # STEP 4 – Αφαίρεση κενών γραμμών
    # ----------------------------
```

```
    nonempty_lines = [line for line in lines if line.strip() != ""]

    if len(nonempty_lines) != rows:
        errors.append(f" Λάθος αριθμός γραμμών: Βρέθηκαν {len(nonempty_lines)}, \
απαιτούνται ακριβώς {rows}.")

    # ----------------------------
    # STEP 5 – Split κάθε γραμμή σε tokens
    # ----------------------------
    world = []
    for idx, line in enumerate(nonempty_lines):
        tokens = line.strip().split()
        if len(tokens) != cols:
            errors.append(
                f" Στη γραμμή {idx+1}: Βρέθηκαν {len(tokens)} tokens, \
απαιτούνται {cols}."
            )
        world.append(tokens)

    # Αν δεν έχουμε ακριβώς 10x10, σταματάμε
    if errors:
        return False, errors

    # ----------------------------
    # STEP 6 – Έλεγχος συμβόλων
    # ----------------------------
    for r in range(rows):
        for c in range(cols):
            cell = world[r][c]
            if cell not in VALID_SYMBOLS:
                errors.append(
                    f" Άκυρο σύμβολο στη θέση ({r},{c}): '{cell}'. "
                    "Επιτρέπονται μόνο '.', '#', '*'."
                )

    # ----------------------------
    # STEP 7 – Έλεγχος κρυφών unicode
    # ----------------------------
    for r in range(rows):
        for c in range(cols):
            cell = world[r][c]
            for ch in cell:
                if ord(ch) > 127:
                    errors.append(
                        f" Κρυφός unicode χαρακτήρας στη θέση ({r},{c}): \
repr={repr(ch)}"
                    )

    # ----------------------------
    # STEP 8 – Προαιρετικοί λογικοί έλεγχοι
    # (Δεν είναι υποχρεωτικοί στην εργασία, αλλά βοηθούν στο debugging)
    # ----------------------------

    # 8a: Έλεγχος unreachable blocks (όχι BFS, απλός heuristic)
    items = [(r,c) for r in range(rows) for c in range(cols) if world[r][c] == \
"*"]
    if not items:
        errors.append("⚠ Προειδοποίηση: Δεν βρέθηκαν αντικείμενα '*' στο \
world.txt.")

    # 8b: Έλεγχος αν το (0,0) είναι εμπόδιο
    if world[0][0] == "#":
        errors.append("⚠ Προειδοποίηση: Το start (0,0) είναι εμπόδιο '#'. "
```

```
                    "Το ρομπότ δεν θα μπορεί να μετακινηθεί.")

    # ----------------------------
    # ΤΕΛΙΚΟ ΑΠΟΤΕΛΕΣΜΑ
    # ----------------------------
    if errors:
        return False, errors
    return True, ["✔ Το world.txt είναι έγκυρο."]


# ----------------------------------------
# Εκτελέσιμο μέρος – επιτρέπει CLI χρήση
# ----------------------------------------
if __name__ == "__main__":
    ok, msgs = validate_world()
    for m in msgs:
        print(m)
    if not ok:
        sys.exit(1)
```

---

# ΤΕΚΜΗΡΙΩΣΗ

Το script κάνει 8 κρίσιμα validation layers:

---

## 1. File loading με ασφαλή UTF-8 ανάγνωση

Ελέγχει:

- αν το αρχείο υπάρχει
- αν έχει σωστό encoding
- αν υπάρχει BOM

Το BOM (`\ufeff`) είναι μεγάλο πρόβλημα από copy-paste PDF/Word.

---

## 2. Κανονικοποίηση γραμμών

Αφαιρεί newlines και επιτρέπει καθαρή split().

---

## 3. Έλεγχος αριθμού γραμμών

Αποκλείει:

- 9 γραμμές
- 11 γραμμές
- κενές γραμμές στη μέση

Απαιτείται **ακριβώς 10**.

---

## 4. Έλεγχος αριθμού tokens ανά γραμμή

Αποκλείει:

- γραμμές με 9 ή 11 στοιχεία
- αόρατους unicode χαρακτήρες
- λάθη copy-paste από PDF

---

## 5. Έλεγχος επιτρεπτών συμβόλων

Μόνο:

- `"."`
- `"#"`
- `"*"`

Οτιδήποτε άλλο → fail.
Αντιμετωπίζει λάθη όπως “·”, “✱”, “•”.

---

## 6. Έλεγχος για κρυφά unicode

Σαρώνει κάθε χαρακτήρα και ελέγχει:

```
if ord(ch) > 127:
```

Αυτό εντοπίζει:

- BOM
- non-breaking spaces
- ‘fancy’ punctuation
- ελληνικά σύμβολα από αυτόματη διόρθωση

---

## 7. Προαιρετικοί λογικοί έλεγχοι

Δεν ρίχνουν error, αλλά προειδοποιούν:

- world χωρίς "*"
- world που ξεκινά με εμπόδιο

- κακή διαμόρφωση

---

# Πώς να το χρησιμοποιήσεις

## 1. Ως standalone script

Αποθήκευση ως:

`world_validator.py`

Τρέχεις:

```
python world_validator.py
```

## 2. Μέσα στο project

Από run() μπορείς να κάνεις:

```
ok, errs = validate_world()
if not ok:
    print("\n".join(errs))
    return
```

## 3. Ως εργαλείο φοιτητών

Οι φοιτητές μπορούν να το τρέχουν πριν στείλουν την εργασία:

```
python world_validator.py
✔ Το world.txt είναι έγκυρο.
```

ή

```
python world_validator.py
 Άκυρο σύμβολο στη θέση (0,3): '.'
 Λάθος αριθμός tokens στη γραμμή 2: 11 tokens, απαιτούνται 10.
```

---

# Συνάρτηση  validate_world

```
import sys

VALID_SYMBOLS = {".", "#", "*"}

def validate_world(path="world_002.txt", rows=10, cols=10):
    """
    Validate world.txt according to the rules of the assignment.

    Returns:
        (ok, errors)
        ok: True/False
        errors: list of error messages (strings)
    """

    errors = []

    # ----------------------------
    # STEP 1 – Read raw file
    # ----------------------------
    try:
        with open(path, "r", encoding="utf-8") as f:
            raw_lines = f.readlines()
    except UnicodeDecodeError:
        errors.append("Encoding error: Το αρχείο ΔΕΝ είναι UTF-8.")
        return False, errors
    except FileNotFoundError:
        errors.append("Το αρχείο world.txt δεν βρέθηκε.")
        return False, errors

    # ----------------------------
    # STEP 2 – Strip newlines
    # ----------------------------
    lines = [line.rstrip("\n") for line in raw_lines]

    # ----------------------------
    # STEP 3 – BOM check
    # ----------------------------
    if lines and lines[0].startswith("\ufeff"):
        errors.append("Εντοπίστηκε BOM (\\ufeff) στην πρώτη γραμμή. Αποθηκεύστε το αρχείο ως
UTF-8 χωρίς BOM.")
        lines[0] = lines[0].replace("\ufeff", "")

    # ----------------------------
    # STEP 4 – Remove empty lines
    # ----------------------------
    nonempty_lines = [line for line in lines if line.strip() != ""]
    if len(nonempty_lines) != rows:
```

```
        errors.append(f"Λάθος αριθμός γραμμών: Βρέθηκαν {len(nonempty_lines)}, απαιτούνται ακριβώς {rows}.")

    # ----------------------------
    # STEP 5 – Tokenize rows
    # ----------------------------
    world = []
    for idx, line in enumerate(nonempty_lines):
        tokens = line.strip().split()
        if len(tokens) != cols:
            errors.append(
                f"Στη γραμμή {idx+1}: Βρέθηκαν {len(tokens)} tokens, απαιτούνται {cols}."
            )
        world.append(tokens)

    # Stop early if world is not 10x10
    if errors:
        return False, errors

    # ----------------------------
    # STEP 6 – Validate symbols
    # ----------------------------
    for r in range(rows):
        for c in range(cols):
            cell = world[r][c]
            if cell not in VALID_SYMBOLS:
                errors.append(
                    f"Άκυρο σύμβολο στη θέση ({r},{c}): '{cell}'. Επιτρέπονται μόνο '.', '#', '*'."
                )

    # ----------------------------
    # STEP 7 – Check for hidden unicode
    # ----------------------------
    for r in range(rows):
        for c in range(cols):
            cell = world[r][c]
            for ch in cell:
                if ord(ch) > 127:
                    errors.append(
                        f"Κρυφός unicode χαρακτήρας στη θέση ({r},{c}): repr={repr(ch)}"
                    )

    # ----------------------------
    # STEP 8 – Optional logical checks
    # ----------------------------
    items = [(r, c) for r in range(rows) for c in range(cols) if world[r][c] == "*"]
    if not items:
        errors.append("Προειδοποίηση: Δεν βρέθηκαν αντικείμενα '*' στο world.txt.")

    if world[0][0] == "#":
        errors.append("Προειδοποίηση: Το start (0,0) είναι εμπόδιο '#'. Το ρομπότ δεν θα μπορεί να μετακινηθεί.")
```

```
# ----------------------------
# RESULT
# ----------------------------
if errors:
    return False, errors
return True, ["To world.txt είναι έγκυρο."]
```