**מטלה 1**

1. הורץ.
2. נוצרו בקבצים path\_scenario\_2, path\_scenario\_3 ו-path\_scenario\_4.
3. הסברים:

* אלגוריתם דייקסטרה: אלגוריתם זה מוצא את המסלול הקצר ביותר מנקודת התחלה בגרף לבין נקודת יעד על סמך המשקלים של כל קשת. זהו אלגוריתם מדויק ולכן הבטחת הביצועים שלו היא 1.

אלגוריתם A\*: אלגוריתם חיפוש מונחה היוריסטיקה. בהינתן גרף רחב, ראשית מעריכים את המסלול שאנחנו מחפשים טרם הפעלת האלגוריתם (ע"י פונקציית הערכה h(n) – פונקציית המרחק האוקלידית/מנהטן המוערך למטרה החל מהצומת הנוכחי).

* הסבר למימוש:
  + קובץ node.py: מחלקה המייצגת צומת בגרף, שמיוצג ע"י תשעה משתנים:
    - קואורדינטה X.
    - קואורדינטה Y.
    - walkable – משתנה בוליאני שמייצג האם ניתן ללכת דרך אותו צומת.
    - h – "מחיר" מהצומת הנוכחי אל צומת היעד.
    - g – "מחיר" מצומת ההתחלה אל הצומת הנוכחי.
    - f – מרחק בין צומת ההתחלה לבין הצומת הנוכחי (f = g+h).
    - opened – כמה פעמים אותו צומת נפתח.
    - closed – האם עברנו דרכו ואי אפשר לחזור דרכו יותר.
    - parent – מאיזה צומת הגענו לצומת הנוכחי.
  + קובץ heuristic.py: מחלקה המייצגת את ההיוריסטיקות השונות (פונקציות מרחק). מכילה את המתודות הבאות:
    - Manhatten – מרחק מנהטן.
    - Euclidean – מרחק אוקלידי.
    - Chebyshev – מרחק צ'בישב.
    - Octile – מרחק אוקטיל.
  + קובץ grid.py: מחלקה שבונה את הגריד שמוצג על המסך. מכילה את המתודות הבאות:
    - build\_nodes – יוצרת את הצמתים לפי גודל הגריד. אם היא מקבלת מטריצה, היא תשמש על מנת לקבוע אילו צמתים הם walkable.

בנוסף, הקובץ מכיל מחלקה פנימית Grid שמייצגת את המפה, המכילה את המשתנים הבאים:

* + - width – רוחב הגריד.
    - height – גובה הגריד.
    - matrix – מטריצה, אם מועברת אליה, כדי לסמן אילו צמתים הם walkable.
    - nodes – הצמתים של הגרף.

בנוסף, המחלקה Grid מכילה את המתודות הבאות:

* + - node – מקבלת צומת במיקום מסוים.
    - inside – בודקת האם נקודה נמצאת בתוך המפה.
    - walkable – בודקת אם הצומת נמצא בתוך הגריד ואם הוא walkable.
    - neighbors – מחזירה מערך של כל השכנים של צומת מסוים.
    - grid\_str – יוצרת מחרוזת "צבועה" של הגריד בעזרת תווי ASCII.
  + קובץ util.py: מכיל את המתודה backtrace שמחזירה את המסלול בהתאם לרשומות של האבא.
  + קובץ diagonal\_movement.py: מכילה את המחלקה DiagonalMovement שמשמשת לדיאגנוזה של הריצה.
  + קובץ a\_star.py: מימוש של אלגוריתם \*A. מכיל את המחלקה AStarFinder שמכילה את המשתנים הבאים:
    - heuristic – פונקציית המרחק.
    - weight – משקל.
    - diagonal\_movement – בשביל הדיאגנוזה.

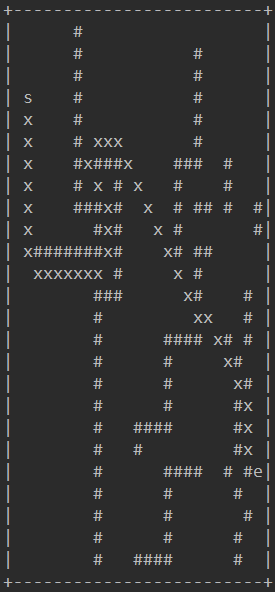
בנוסף, היא מכילה את המתודה find\_path, שמוצאת את המסלול מנקודת ההתחלה לנקודת היעד באמצעות אלגוריתם \*A.

* + קובץ dijkstra.py: מימוש של אלגוריתם דייקסטרה. מכיל את המחלקה DijkstraFinder שמכילה את המשתנים הבאים:
    - weight – משקל, תמיד 1 במקרה זה.
    - diagonal\_movement – בשביל הדיאגנוזה.

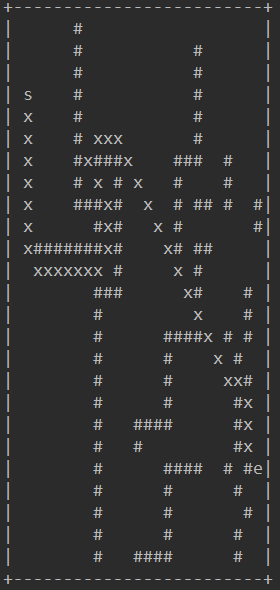
בנוסף, הconstructor מכיל את המתודה heuristic שמחזירה תמיד 0.

* + קובץ test\_path.py: הקובץ הראשי ממנו מריצים את התכנית.
* צילומי מסך:

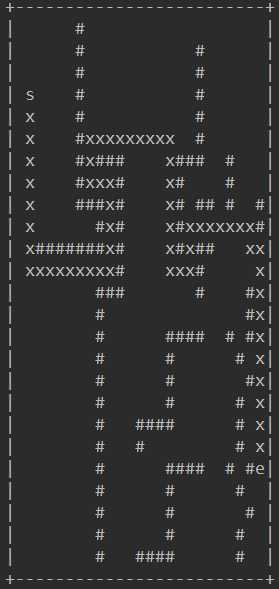
מפה בגודל 25\*25 כפי שמתוארת בקובץ path\_scenario\_2.



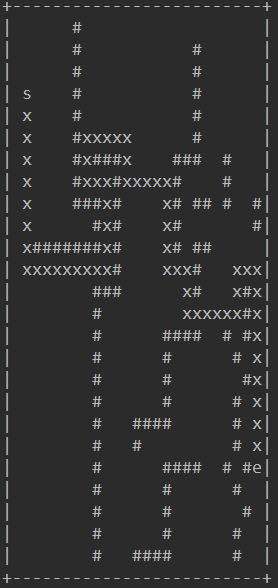
דייקסטרה כולל אלכסונים



\*A כולל אלכסונים

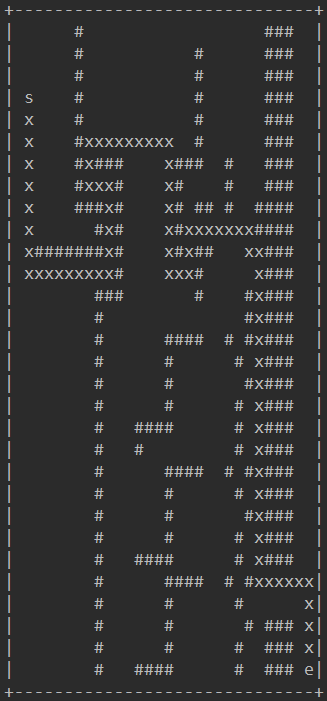


דייקסטרה לא כולל אלכסונים

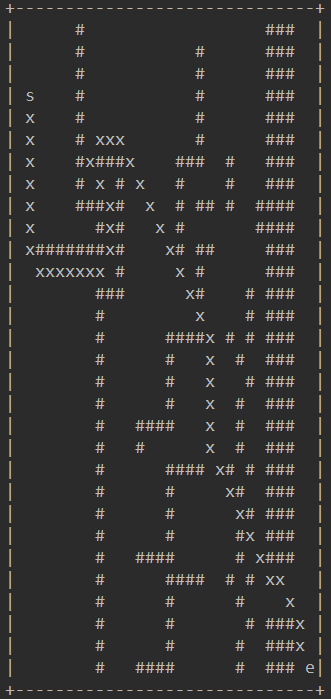


\*A לא כולל אלכסונים

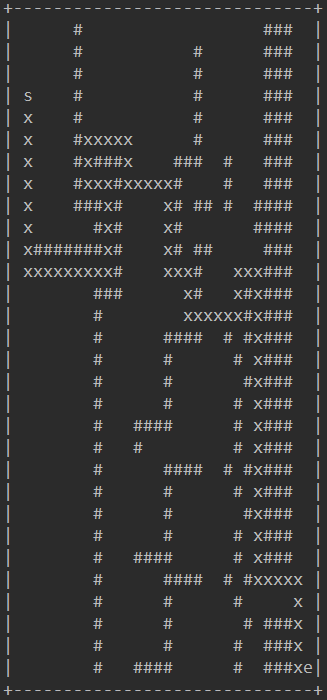
* מפה בגודל 30\*30 כפי שמתוארת בקובץ path\_scenario\_3.



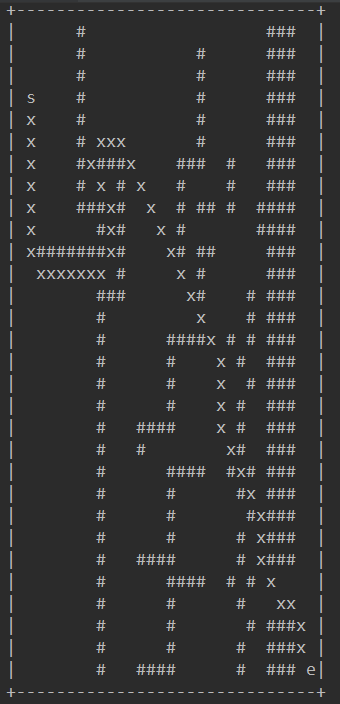
דייקסטרה לא כולל אלכסונים



דייקסטרה כולל אלכסונים

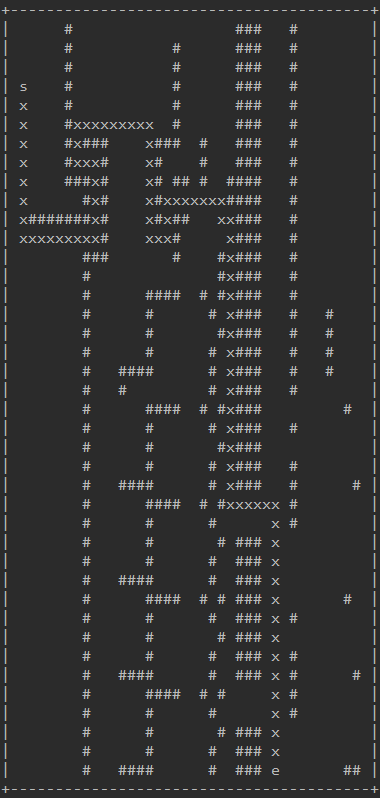


\*A לא כולל אלכסונים

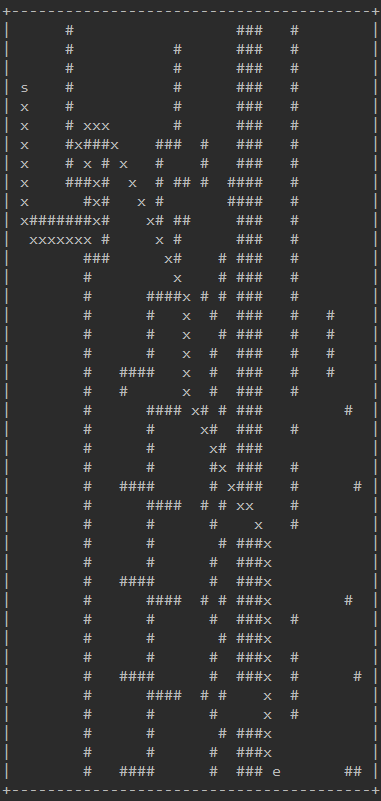


\*A כולל אלכסונים

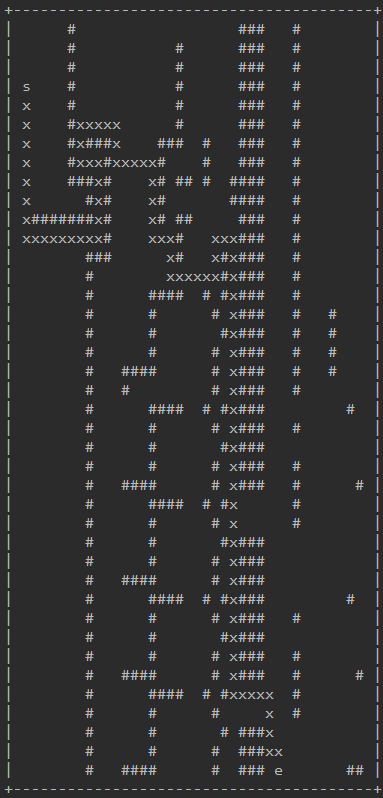
* מפה בגודל 40\*40 כפי שמתוארת בקובץ path\_scenario\_4.



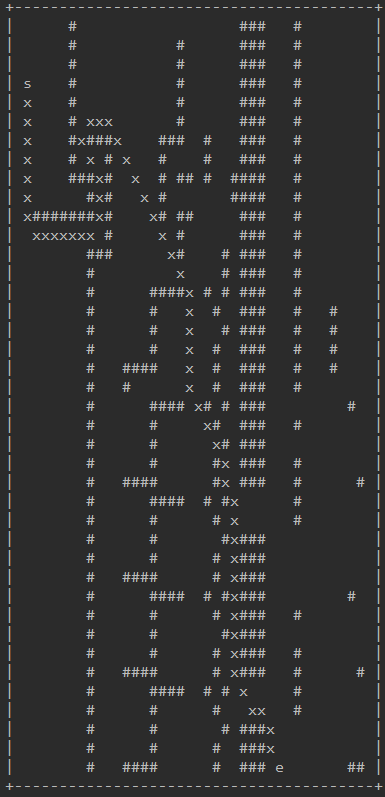
דייקסטרה לא כולל אלכסונים



דייקסטרה כולל אלכסונים



\*A לא כולל אלכסונים



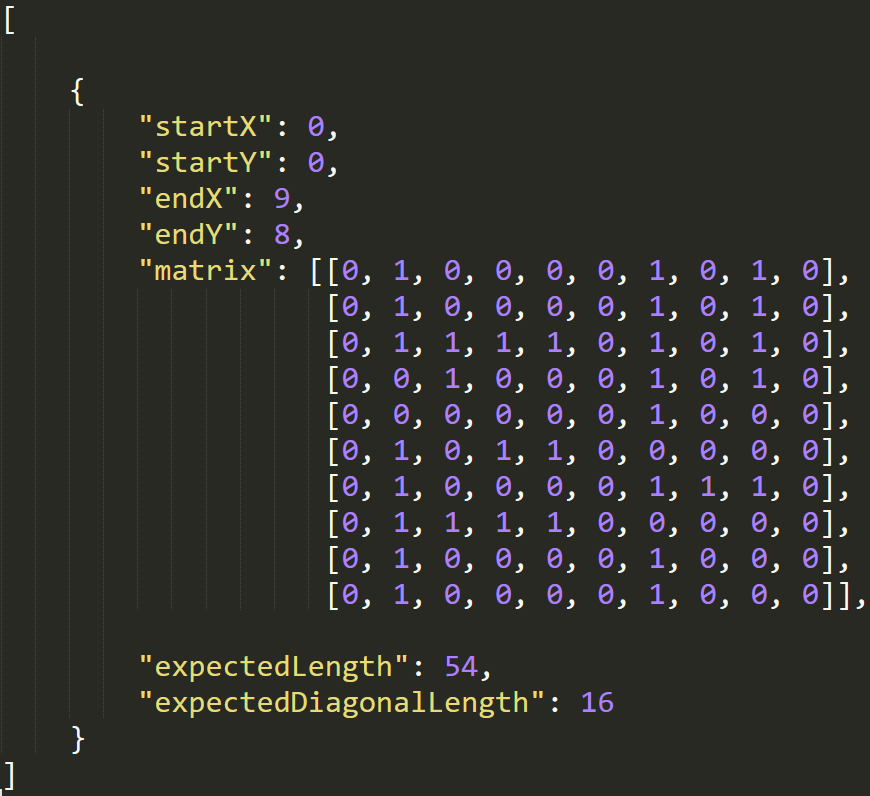
\*A כולל אלכסונים

* מסקנות:

1. כפי שניתן לראות שני האלגוריתמים , \*A ו- Dijkstra מצאו תמיד את אותו אורך מסלול.
2. אלגוריתם \*A יעיל יותר במציאת המסלול הקצר ביותר.
3. ללא מעבר באלכסון: ניתן לראות כי נדרשו לאלגוריתמים מספר גדול יותר של חישובים, וכן קיבלנו מסלולים אחרים.
4. עם מעבר באלכסון: ניתן לראות כי נדרשו לאלגוריתמים מספר קטן יותר של חישובים, וכן המסלולים שונים וכוללים מעבר באלכסון במידה ואפשר.
5. runs מייצג את כמות הריצות שהאלגוריתם ביצע עד לקבלת הפתרון.

path מייצג את המסלול הנבחר.

1. יצרנו מפה בגודל 10\*10 שמתוארת להלן (ומופיעה בקובץ path\_scenario\_5):



קובץ הפלט output.txt עם ערכי f, g, h נמצא בתיקיית המעבדה.

1. בעיות מהחיים:

* בעיה 1: ניווט במפה.

בעזרת האלגוריתמים \*A ודייקסטרה, ניתן לכתוב אפליקציות לניווט מנקודה לנקודה דרך המסלול הכי קצר שניתן, בהתאם למכשולים כגון: כבישים שסגורים לעבודות תשתית, מדרכות, רכבים שעומדים בצורה מסוכנת וכו'.

* בעיה 2: רשת אינטרנט.

בעזרת האלגוריתמים \*A ודייקסטרה, ניתן לבנות רשתות תקשורת להעברת נתונים בין אנשים. האלגוריתמים יבחרו את המסלולים הכי קצרים, שהם חוטי התקשורת כדי להעביר את המידע בצורה המהירה, הקצרה והיעילה ביותר.