תוכן עניינים:

מבוא

בקורס אלגוריתמים היוריסטיים ומקורבים נחשפנו לאלגוריתמים שנועדו לפתור בעיות בצורה מהירה יותר, קלה יותר ע"י ויתור של אופטימיזציה, דיוק, השתמשנו באלגוריתמים אלו על מנת לפתור בעיות NP קשות שידוע שאותם לא ניתן לפתור בצורה מהירה ומדויקת.

בשיטות היוריסטיות מעבר ממצב אחד למצב הבא נעזר בשיקולים יוריסטים, המציינים הערכה לכדאיות של המעבר למצב הבא שנבחר שממנו ניתן להגיע ליעד.
השיקולים היוריסטים ניתנים להציג על ידי פונקציות, הפונקציות מתאימה לכל מצב ציון שמייצג את הכדאיות לעבור למצב הבא דרכו.

בקורס ראינו מימושים שונים בשפת Python , אשר כלל גם A* ו Dijkstra שעליהם אדבר בהרחבה בהמשך.

רובוטים בחיי היום יום

"משחר ההיסטוריה חיפש האדם לייצר כלים שיסייעו לו במלאכתו ויקלו את חייו. מאבני הצור עמן חתך את מזונו, דרך החץ והקשת שאפשרו לו לצוד ביתר יעילות, הפטישים שאפשרו לפסל את יצירות האמנות המרהיבות של הרנסנס ועד המחשב, המכיל מידע שפעם אכלס בניינים שלמים, עליו נכתבות שורות אלה מבלי ללכלך את הידיים בדיו – הדחף לשכלל ולשפר את הטכנולוגיה"
/https://www.davar1.co.il/89338

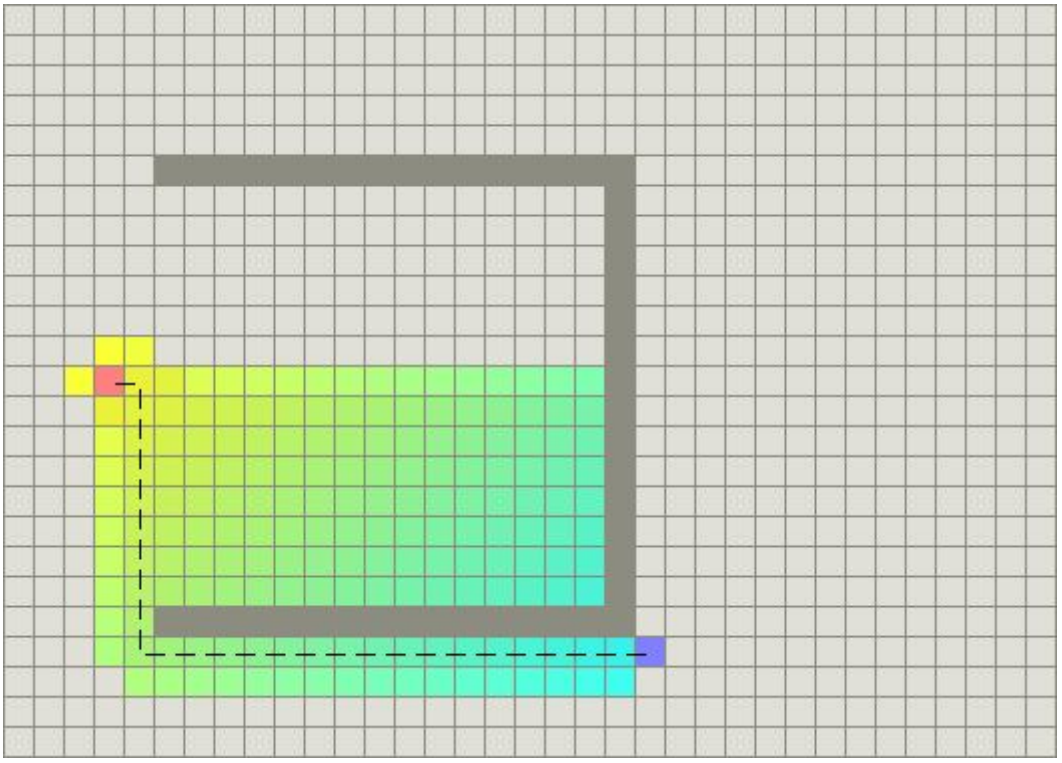
הטכנולוגיה המתקדמת של היום עוזרת לחסוך בהוצאות ענק לחברות גדולות, ולכן משקיעים חברות רבות מאמץ רחב לייעל את התהליכים השונים ולהפוך אותם לאוטומתים,
היום קיימים רובטים בתחומים שונים, בעסקי המזון חברה אמריקאית בשם "מומנטום" מפתחת מכונה שמסוגלת לטגן המבורגרים ולחתוך ירקות במקביל, ולהכין 360 המבורגרים בשעה, כך שהיא תחסוך ליצרניות המזון המהיר כמאות מיליוני דולרים, שמשולמים כיום על משכורות, שטח מטבח גדול ומוצרי מזון שנזרקים בשל שימוש לא יעיל.
ביפן יש כבר כאלף רובוטים שמשמשים כאנשי שירות בחנויות מסיוע ללקוחות למצוא את מבוקשם, דרך מענה לשאלות על המוצרים ועד סיוע בארגון וניקוי החנות.
במחסני אמזון כבר אין צורך במלגזנים. רובוטים מיוחדים מניעים ממקום למקום מדפים עמוסים באלפי פריטים, ומשם רובוטים אחרים ממיינים את הסחורות.

על כן גדלה חשיבות האלוגריתמים היורסטים והבינה המלכותית אשר אנחנו למדים בקורס, אשר מהווים כחלק גדול מהתעשייה ומהמגמה הגדלה של אותם פרויקטים, בנוסף הקרוס תרם לנו הרבה לצורת פיתוח החשיבה.

האלגוריתם A* :

אלגוריתם A* הוא אלגוריתם פופולרי למציאת מסלול בין נקודות, תוך כדי התגברות על מכשולים, הוא משתמש בפונקציה היוריסטית על מנת להעריך את הצומת הזמנית הבא אליו הוא צריך ללכת שתשמש כצומת במסלול הקצר שהוא מצא.

ללא הפונקציה היוריסטית הוא נותן תוצאה למציאת מסלול בדיוק כמו האלגוריתם של Dijkstra ויכול למצוא מסלול באותו זמן עבודה כמו אלגוריתם חמדני (GREEDY BFS לנקודת המטרה).



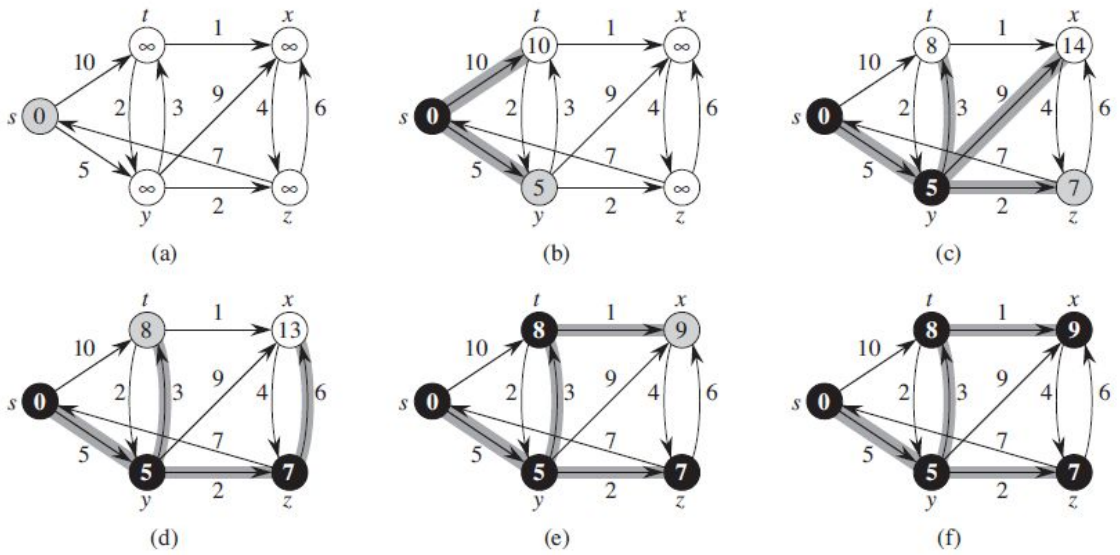
בסרטוט ניתן לראות דוגמה של ריצה של האלגוריתם, כאשר האפור הכהה הוא בעצם מכשול, הנקודה האדומה מהווה את נקודת ההתחלה, הנקודה הכחולה מהווה את נקודת הסיום, הקו המקווקו השחור מהווה את המסלול הקצר שהוא מצא, והנקודות הצבועות מסמנות את הנקודות בהם הוא ביקר או שקל לבקר.

אלגוריתם Dijkstra:

אלגוריתם דיקסטרה הוא אלוגריתם למציאת המסלול הקצר ביותר בין קודקוד לבין קודקודים אחרים בגרף. האלגוריתם המקורי נכתב כדי למצוא את המסלול הקצר ביותר בין שני צמתים בגרף, אחריו פותח גירסא בין קודקוד לשאר הקודקודים בגרף והיא הפופולרית ביותר היום.

מימוש האלגוריתם:

```
function Dijkstra(Graph, source):

    create vertex set Q
    // Initialization
    for each vertex v in Graph:
        // Unknown distance from source to v
        dist[v] ← INFINITY
        // Previous node in optimal path from source
        prev[v] ← UNDEFINED
        // All nodes initially in Q (unvisited nodes)
        add v to Q

    // Distance from source to source
    dist[source] ← 0

    while Q is not empty:
        //// Node with the least distance will be selected first

        u ← vertex in Q with min dist[u]
        remove u from Q

        // where v is still in Q.
        for each neighbor v of u:
            alt ← dist[u] + length(u, v)
            // A shorter path to v has been found
            if alt < dist[v]:
                dist[v] ← alt
                prev[v] ← u

    return dist[], prev[]
```


(a)
(b)
(c)
(d)
(e)
(f)

תכנון הפרויקט

מטרה:

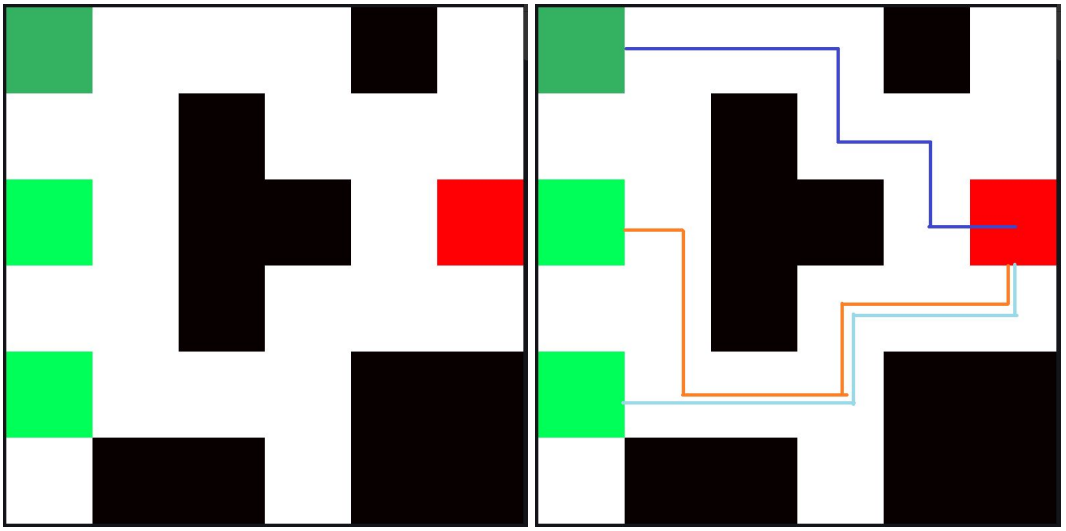
בפרוויקט עסקתי בלנסות לפתור את הבעיה של 3 רובטים אשר מנסים להגיע לאותה נקודת סיום, במסלול קצר בלי להתנגש, כאשר המסלול כולל מכשולים בצורות גאומטריות שונות (מלבן, ריבוע, משולש, תרפז)

פתרונות אפשריים:

1. **התגנשות הרובטים:**
   a. **עצירה של הרובוט במקום -** כאשר הרובוט ירצה לעבור למיקום שרובוט אחר תופס אותו, הרובוט יחכה, עד שהמיקום יהיה פנוי, דבר זה פותר את בעיית ההתנגשויות של הרובטים, הרובטים תמיד יתקדמו לעבר המטרה, ולכן לכל היותר נעצר פעמים. כאשר הרובוט מגיע למטרה, הוא לא רלוונטי יותר, ואפשר להתקדם לעבר המטרה.
   b. **הליכה בדרך אחרת** - כאשר הרובוט מתכוון לעבור למיקום שרובוט אחר תופס אותו, הוא יבחר בדרך ה - N האופטימלית עבורו, הכוונה שהוא יתייחס לרובטים כמוקשים באותה נקודת זמן, ויבחר ללכת בדרך האופטימלית האחרת, וכן יחזיר את האפשרות ללכת לאותו מקום אחרי הצעד (שכן המקום יהיה פנוי)
2. **היכרות עם הסביבה:**
   a. **רובוטים לא מכירים את הרובטים האחרים -** הרובוט לא מודע להחלטות הרובטים האחרים, הוא מודע למפה, הוא מסמן את החשיובים היורסטים עבורו בלבד, כאשר הוא צועד רק אז הוא מודע למיקום של הרובטים על מנת לא להתנגש בהם.

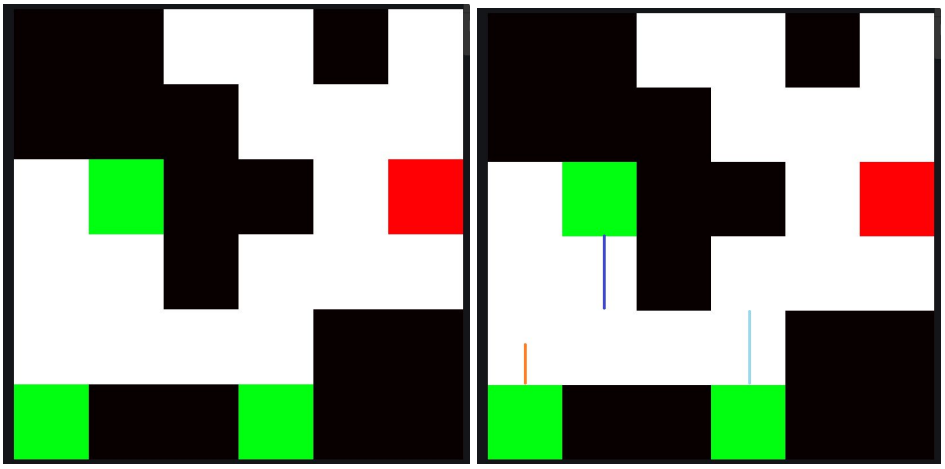
דוגמות מספריות קטנות:

דוגמה 1:

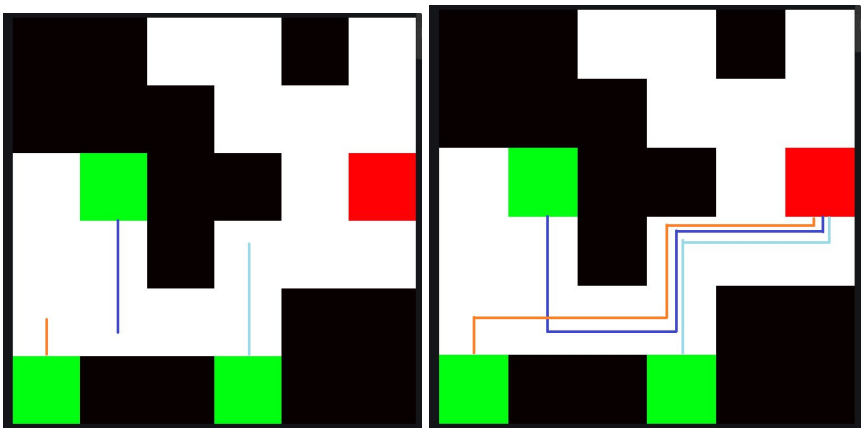


מסלול פשוט, אף רובוט לא מתנגש אחד עם השני שניהם ממשיכים במסלולים שלהם.

דוגמה 2:

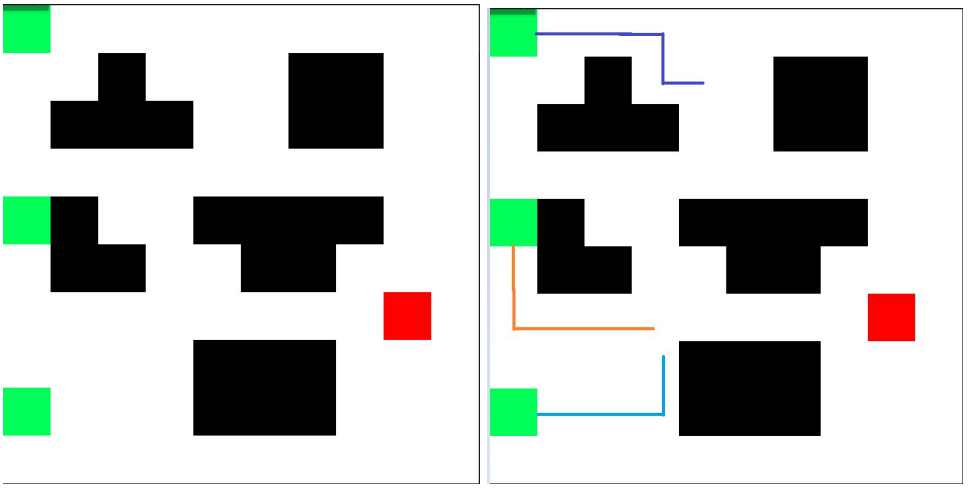


הרובוט הכתום יצתרך לחכות תור , כי בצד הבא של הכחול הוא יזוז צעד אחד ויתפוס את המיקום שהכתום רצה ללכת, ואחרכך נמשיך כרגיל:

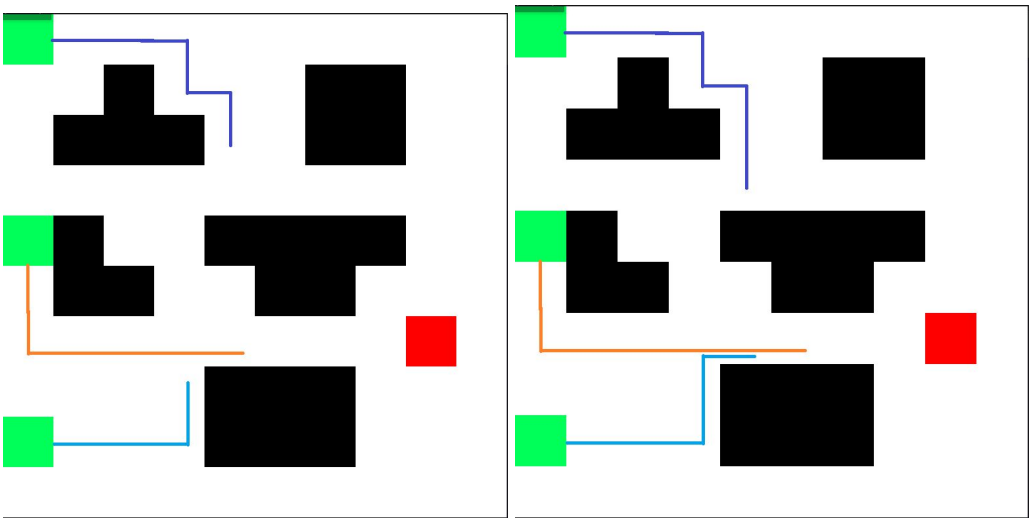


דוגמות מספריות גדולות:

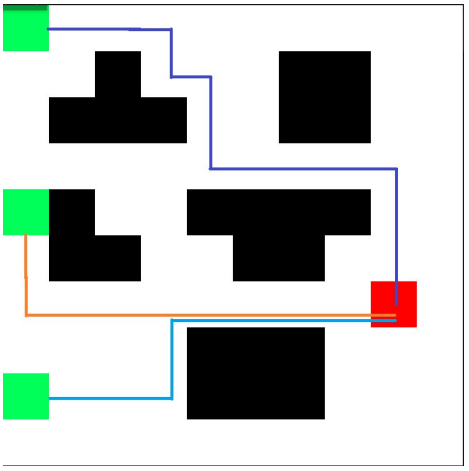
דוגמה מספר 1:



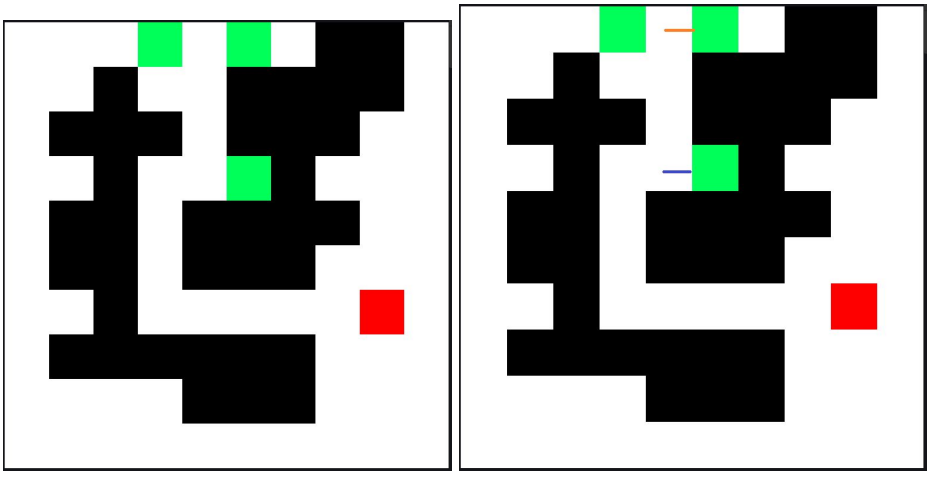
כרגע אנחנו רואים שבצעד של הבא של הרובוט התכלת הוא ינתגש ברובוט הכתום, לכן הרובוט התכלת מפסיק את תזוזתו כרגע, ויתן לרובוט הכתום להתקדם.



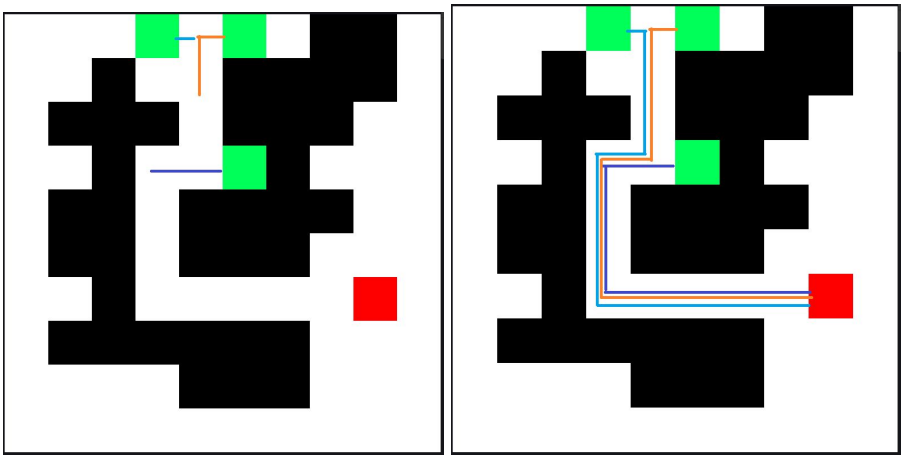
כעת הרובטים ימשיכו כרגיל עד אשר יגיעו ליעדם:



דוגמה מספר 2:

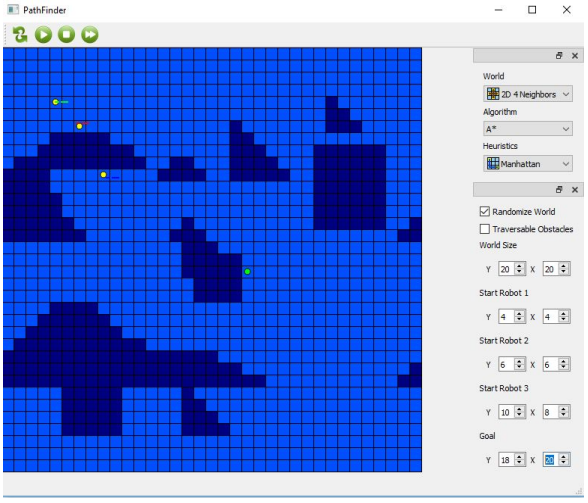


הרובוט השני תקוע כי רובוט מספר 1 לקח את מקומו, לכן רובוט מספר 2 מחכה ,
לתור הבא, במהלכו רובוט מספר 3 התחיל להתקדם לעבר היעד.

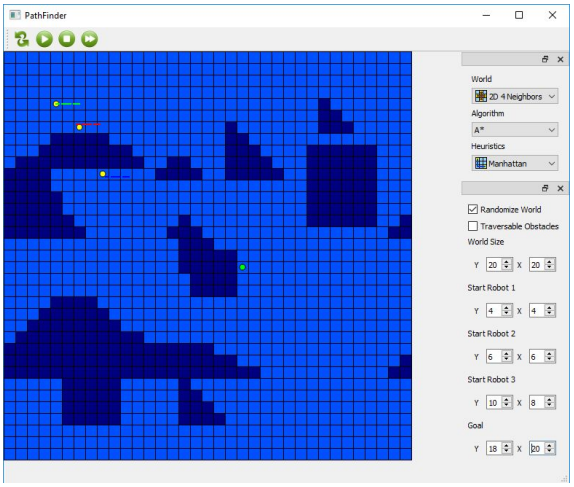


הרובוט התכלת והכתום יבצעו את אותו המסלול בזמנים שונים.

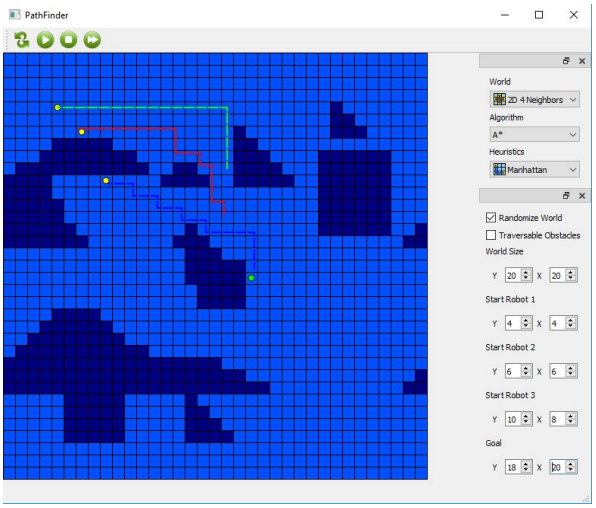
ישום הפרוויקט



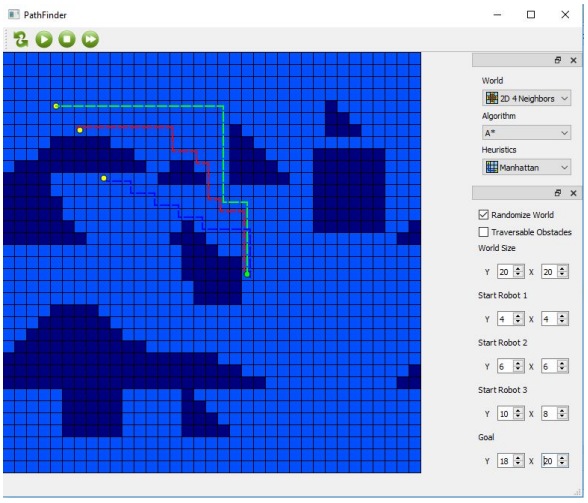
World: 2D 4 Neighbors
Algorithm: A*
Heuristics: Manhattan
☑ Randomize World
☐ Traversable Obstacles
World Size Y 20 X 20
Start Robot 1 Y 4 X 4
Start Robot 2 Y 6 X 6
Start Robot 3 Y 10 X 8
Goal Y 18 X 20



World: 2D 4 Neighbors
Algorithm: A*
Heuristics: Manhattan
☑ Randomize World
☐ Traversable Obstacles
World Size Y 20 X 20
Start Robot 1 Y 4 X 4
Start Robot 2 Y 6 X 6
Start Robot 3 Y 10 X 8
Goal Y 18 X 20



World: 2D 4 Neighbors
Algorithm: A*
Heuristics: Manhattan
☑ Randomize World
☐ Traversable Obstacles
World Size Y 20 X 20
Start Robot 1 Y 4 X 4
Start Robot 2 Y 6 X 6
Start Robot 3 Y 10 X 8
Goal Y 18 X 20



World: 2D 4 Neighbors
Algorithm: A*
Heuristics: Manhattan
☑ Randomize World
☐ Traversable Obstacles
World Size Y 20 X 20
Start Robot 1 Y 4 X 4
Start Robot 2 Y 6 X 6
Start Robot 3 Y 10 X 8
Goal Y 18 X 20

מימשתי את הפרוויקט בשפת Python, כמו שרואים קיימים 3 רובטים שמתקרבים למטרה, ויודעים לסנכרן שהם לא יתנגשו אחד בשני,
האלוגריתם שבתמונה רץ ב A* או Dijkstra

הפרוויקט כולל יצירת מפת מכשולים רנדמולית עם אוסף של צורות שמהוות מכשולים, הדפסת המרחקים שנמצאו, וכמות הקודקודים שהוא חיפש על צמת למצוא את המסלול הקצר ביותר.

סרטונים וקוד:

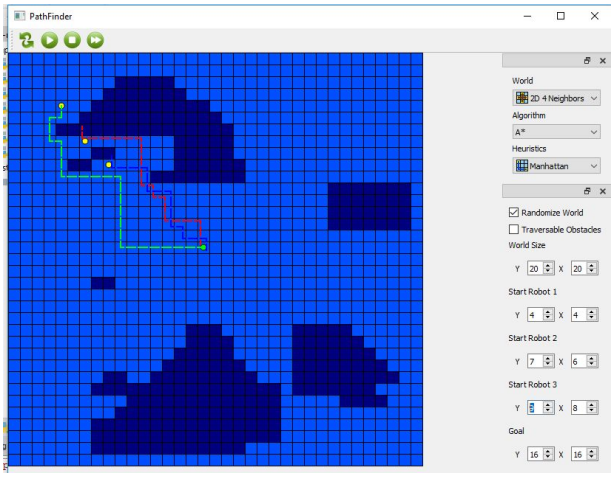
https://github.com/Shaharking/a-star-robots

השוואה בין אלוגריתם Dijkstra ל A*:

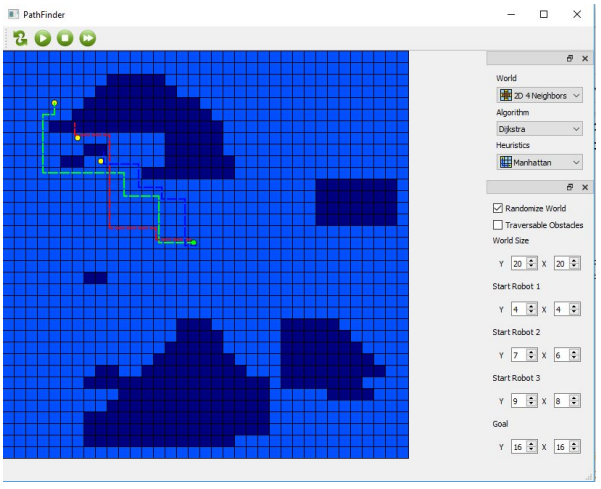
כדי לבחון את ההבדלים בין שני האלגוריתמים אנחנו נריץ את התוכנית מספר פעמים עם דייקסטרה וגם עם A* וננסה לבין את ההבדלים, ההיתרונות והחסרונות של כל אלגוריתם.

הרצה מספר 1:

```
AStar
It searched 81 vertex for the 3 robots
distance for 3 robots
[27, 21, 17]
```



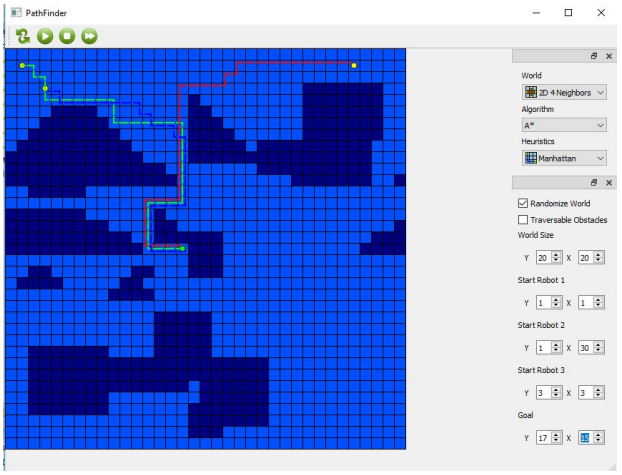

```
Dijkstra
It searched 410 vertex for the 3 robots
distance for 3 robots
[27, 21, 17]
```



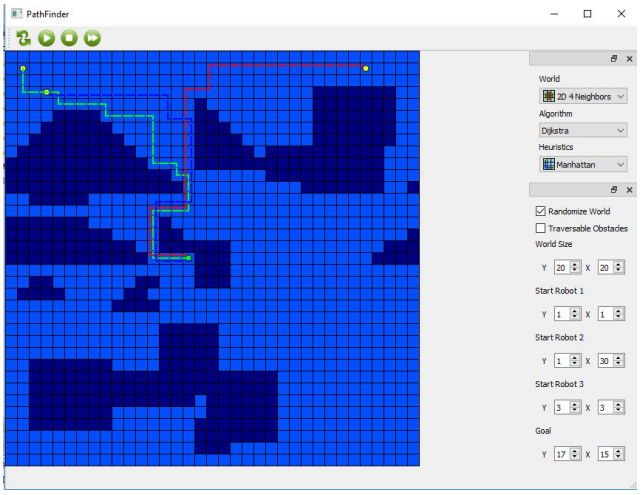
בדוגמה זו ניתן לראות שאלוגריתם דייקסטרה ו A* מצאו מסלולים שונים, אך באורכים זהים ! , לעומת זאת ניתן לראות שאלגוריתם דייקסטרה סרק 410 צמתים, לעומת אלגוריתם A* שסרק רק 81 צמתים.

הרצה מספר 2:

```
AStar
It searched 204 vertex for the 3 robots
distance for 3 robots
[37, 38, 33]
```

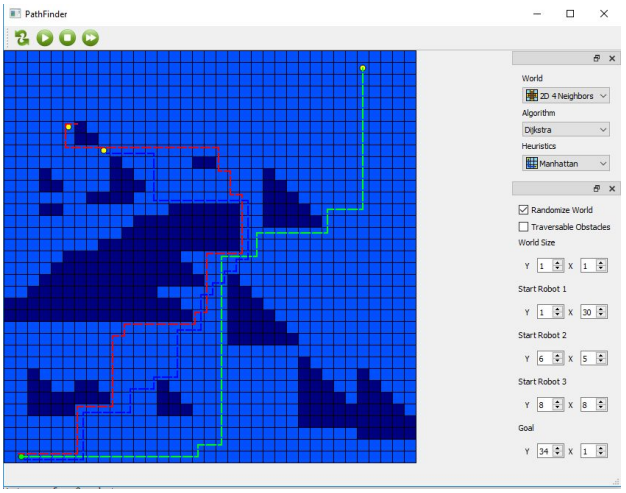



```
It searched 565 vertex for the 3 robots
distance for 3 robots
[37, 38, 33]
```

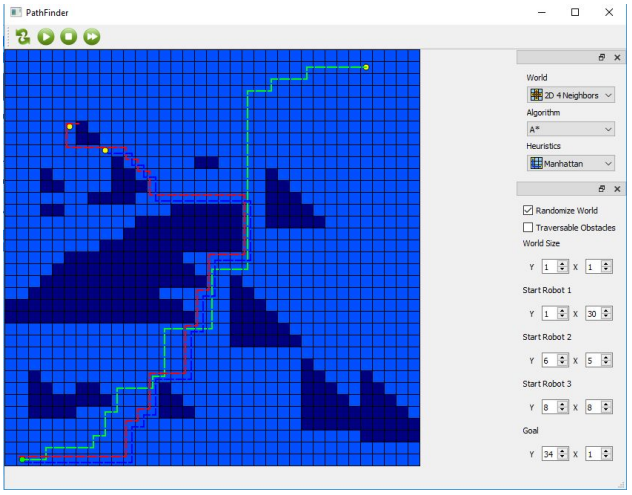


בהרצה זו ניתן לראות שעוד פעם האלגוריתם מצאו מסלולים עם אורכים זהים, אך אלוגריתם A* הצליח לחסוך שוב בכמות הצמתים שאותם סרק על מנת למצוא את המסלול הקצר בין שני נקודות.

הרצה מספר 3:




```
Dijkstra
It searched 990 vertex for the 3 robots
distance for 3 robots
[63, 64, 58]
```



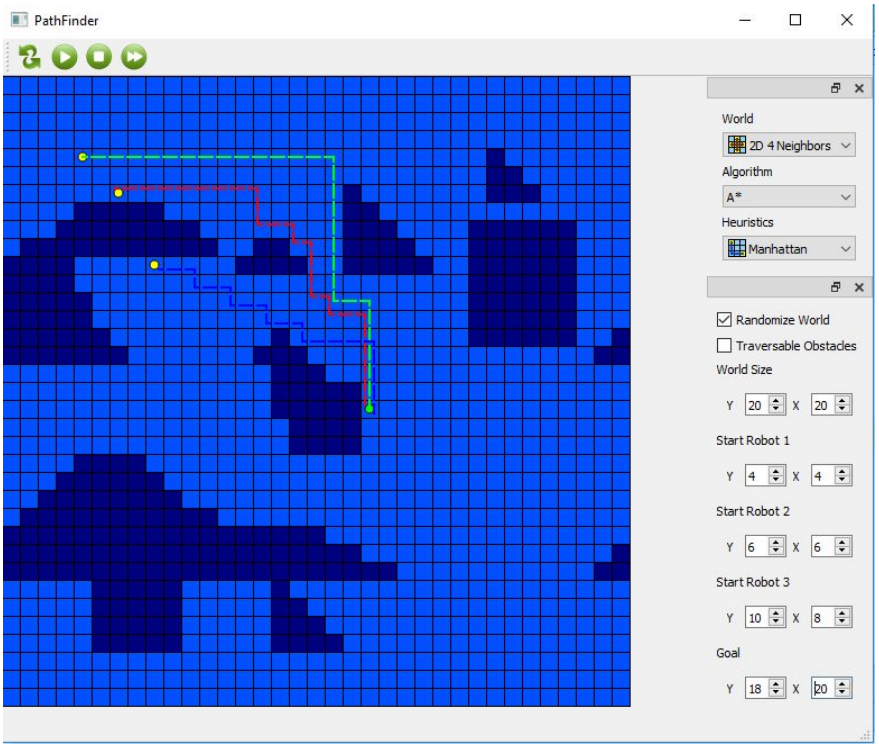

```
AStar
It searched 615 vertex for the 3 robots
distance for 3 robots
[63, 64, 58]
```

בהרצה 3:

ניתן לראות שוב שמספר הצמתים שנסרקו ב A* משמעותית יותר טוב, מאשר דייקסטרה.

לאחר שהרצנו את שני האלגוריתמים, ניתן לראות הבדלים שונים כיצד האלוגרימתמים מתמודדים עם התגברות על מכשולים, והבטחת ביצועים, יעלות וזמן ריצה.

התגברות על מכשולים:
בפרויקט שלי הייתי צריך לגרום ל 3 רובוטים להגיע לנקודת יעד במסלול עם מכשולים, וללא התקלות ברובטים אחרים, והיה ניתן לראות שבשני המקרים האלגוריתמים הצליחו להגיע לנקודת יעד עם מכשולים, עם דייקסטרה שלא מוצא מסלול עם מכשולים, הייתי צריך לחבר בין צמתים רק שאפשר לעבור בינהם, ולכן עם התאמות שני האלוגריתמים יודעים להתגבר על מכשולים.



הבטחת ביצועים:
באלגוריתם דייקסטרה הוא אלגוריתם קבוע, ויעיל בזמן פולינאמלי,
כמו כן A* הוא קבוע ודטרמיניסטי לכן על אותם נתונים נקבל תמיד את אותו תוצאה,
אבל בכל זאת A* לא משיג את המסלול הקצר ביותר בהכרח, ואין לו הבטחת ביצועים.

יעלות וזמן ריצה:
זמני הריצה של A* הם טובים בהרבה מ Dikjstra במקרה הממוצע, מכיוון שהוא לא עובר על כל הצמתים, והוא משתמש בפונקצית היורסטיקה על מנת להעריך לאיפה הוא צריך ללכת על מנת למצוא מסלול קצר יותר.

מסקנות

האלגוריתמים של A* ו דייקסטרה הם שניהם אלגוריתמים שנותנים פתרון טוב לבעיית מציאת המסלול הקצר ביותר עם התגברות על מכשולים (כאשר בשביל DIKJSTRA היינו צריך לעשות טיפה התאמות).

האלגוריתם A* כוכב נותן לנו פתרון היוריסטי לבעיה, אבל ראינו שגם דוגמאות שלנו הוא הביא לנו פתרונות אופטימליים כמו דייקסטרה.

שני האלורתמים מהירים, ושניהם יחסית קלים למימוש, בנוסף A* הוא הרחבה של אלגוריתם דייקסטרה, ואפשר ליישם את אלגוריתם דייקסטרה עם פונקציה יורסטית שתחזיר 0 (וגם כך מיממשתי את דייקסטרה).

האלוגריתם של A* החזיר לנו במהירות גובה יותר את המסלולים הקצרים בצורה משמעותית.

שיעורי בית

משימה בית 1:

1) מה היתרונות והחסרונות של AI בעתיד? (לפי השקפותיהם של הוקינג, מוסק וגייטס)
2) דעתך על העתיד של AI
3) השוואה של PRM ו- Dijkstra

תשובה 1:
על פי סטיבן הוקינג הסכנה הגדולה היא יצירת טכנולוגיות צבאיות היכולות להחליט על הרג של אנשים. האנשים מפתחים על ידי אבולציות ביולוגיות שלוקחות זמן רב, דבר שכלל לא נכון אצל מחשבים והסכנה הנשקפת שהיצורים הממוחשבים האלה יעקפו את היכולות של בני האדם מאד מהר.
אלון מאסק מוסיף ואומר כי ה AI מסוכן יותר מ Nuclear, וצריך לבצע רגולציות ופיקוח על המערכות שמפתחים.
ביל גייטס מחזק בעיקר את אילון מאסק וסטיבן ורואה את ההשלכות העתידיות כאשר אותם רובוטים כבר בעלי אינטליגנציה גדולה מידי כדי להוות איום לאנושות(שימושים צבאים וכו').

תשובה 2:
ה AI הוא תחום בעל פוטנציאל בלתי מוגבל עבורינו , כיום הAI הוא יותר מותאם לפתור בעיות קטנות וספציפיות , והוא בכלל לא ברמה שכמו שהוא מוצג בסרטים, לכן הוא יאפשר לבני האדם להתקדם בתחומים שונים (רפואה, מדע ועוד).
ישנם סכנות רבות לטכנולוגיה, כמו שאקדח מהווה סכנה, גם הטכנולוגיה הזאת, וצריך לדאוג שהכלים האלה יהיו בידים הנכונות של האנשים.

תשובה 3:
Dijkstra -מוצא את הדרך הקצרה ביותר מצומת לצומת אחרת בגרף,
PRM - מוצא דרך להגיע מנקודה לנקודה במפה כאשר ישנם מכשולים, נעזרים ב Dijkstra אלגוריתם על מנת למצוא את המרחקים בין נקודות שונה במפה, בשביל לייצר נקודות במפה, בונים גרף על המפה, כאשר הנקודות הם הצמתים, והקשתות עם הדרכים בין הנקודות השונות במפה (כאשר מעיפים צלעות שמתנגשות עם המכשולים).

משימת בית 2:

א) אלגוריתם דיקסטרה מבטיח מציאת מסלול הקצר ביותר מנקודה ראשית ליעד.
סיבוכיות האלגוריתם תלויה במבנה הנתונים השומר את הקודקודים שטרם ביקרנו בהם, אם מדובר ברשימה או במערך הסיבוכיות היא O(|V|^2) אם משתמש בערימה בינארית סיבוכיות הריצה נהיית O(|E| log |V| + |V| log|v ואם משתמשים בערימת פיבונצי הסיבוכיות משתפרת ל O(|E|+|V|log|V|).

ב) אלגוריתם PRM מחפש את המסלול הקצר הביותר תוך כיסוי של חלקים מן החלל בכל ריצה מכוסה חלק גדול יותר, ולא בודק את החלל כולו כפי שבמצעת אלגוריתם דייקסטרה. מכאן שהוא יעיל יותר מדייקסטרה.

ג) אלגרויתם דייקסטרה מדויק יותר מאלגוריתם PRM וזאת בגלל שהוא בודק את כל הצמתים הבודק חלק מן הצמתים שרלוונטים לגביו למסלול שהוא רוצה לעשות.

ד) אלוגריתם PRM יודע להתמודד עם מכשולים בחלל בגלל שיטת העבודה שלו שהוא בודק אם מנקודה אחת לאחרת יש מסלול פנוי ומוסיף אותך וכך יוצר מסלול מנוקדת התחלה לנקודת יעד.
אלגוריתם דייקסטרה לא יודע להתמודד עם מכשולים הוא נותן מסלול קצר ביותר מנוקדת התחלה לנקודת יעד.

שאלה מספר 3:
השוואה בין PRM דייקסטרה ו A*

PRM

יתרונות:
א)מתגבר על מכשולים שזה בעצם אחד הייתרונות הבולטים של האלגוריתם משהו שדייקסטרה לא יכול לעשות.

ב) מהיר ויעיל לא עובר על כל הצמתים אלא רק מהנקודות שהגרילו עבורו, לכן הוא יתמודד הרבה יותר מהיר בגרפים גדולים.

ג) ימצא את המסלול הרבה יותר מהר מדייקסטרה שכן הוא יוריסטי.

חסרונות:
א) לא מדוויק - לא בכל זמן נצליח להשיג מסלול
ב) אין הבטחת ביצועים - במצבים מסוימים הוא פחות יעיל מדייסקטרה.
ג)בגלל שהאלגוריתם הוא רנדומלי נקבל תוצאות שונות בכל ריצה.

A*

יתרונות:
א) לא עובר בכל הקודקודים כמו דייסקטרה אלא אלגוריתם מוצא את המרחק הקצר ביותר שבין נקודות ליעד באמצעות חישוב המרחק הזול ביותר, בכל ריצה האגלוריתם מוסיף לחישוב את המרחק הקודם שנמצא עד להגעה לנקודת היעד.

ב) יותר מהיר מדייקסטרה בפרקטיקה בדרך כלל יתכן שבמספר ריצות , האלגוריתם ימצא את מהסלול הזול ביותר אל היעד.

ג) דטרמיניסטי מקרוב כלומר הרצת האלגוריתם החיפוש מבטיח את מציאת המסלול בין צומת המקור לצומת היעד עם קיים מסלול כזה.

חסרונות:

א)בשונה מדייקסטרה אין ערבות לכך ש A* ימצא את המסלול הקצר ביותר אלא את המסלול הקצר ביותר הראשון מבין כלל המסלולים שעבר בהם.

ב) במסלולים מסוימים ריצת האלגוריתם יכולה לקחת יותר זמן מאלגוריתם אחר כמו PRM למשל.

דייקסטרה:
א) מדוויק - הוא עובר על כל הצמתים ומעדכן את הערך שלהם אז מחזיר את הקצר ביותר.

ב) יעיל ודטרמינסטי - המסלול הקצר ביותר הוא יתקבל בוודאות מאחר ומבקרים בכל הצמתים.

ג) זמן ריצה קבוע בכל מצב יש התחייבות לזמן ריצה גם במקרה הגרוע.

חסרונות:
א) עובר על כל הצמתים כדי למצוא את המסלול - גם צמתים שלא רלוונטים לתשובה לכן במקרים מסווימים אלגוריתם מקורבים יהיו עדיפים עליו.

ב) לא מתגבר על מכשולים - אין לו שום יכולת להתגבר על בעית המכשולים.

ג) עובד רק עם גרפים - גלוריתם עובד על גרף נתון, מכוון או לא מכוון בעל משקלים אי שלילים של הקשתות.

מטלה 1 - דירוג רובטים עם BORDA ו TOPSIS

project	עוצמת הפרוויקט	R_i1	הסתברות להצלחת הפרוויקט	R_i2	דירוג בורדה
1	C	5	0.6	3.5	0///5
2	Mo	1	1	1	4///8
3	S	2	0.9	2	3///6
4	Mi	4	0.6	3.5	2///3.5
5	S	3	0.4	5	1///2

השוואה בין הרובטים ב Bonda ו Topsis:

Arduino , Humanoid NAO, Hamster, Lego, Drone Quadcopter

Bonda:

על פי ציונים

	Price	Weight	Learning Rate	Programmability Rate	controlability	reliability	maintenance
Arduino	7.47	6.73	7.88	3.50	7.96	1.14	3.83
Humanoid NAO	1.96	7.66	8.55	3.47	3.77	3.71	4.04
Hamster	4.7	1.8	8.2	9.4	1.2	3.2	5.1
Lego	3.5	6.6	4.4	2.2	1.1	8.5	5.6
Drone Quadcopter	2.2	4.4	1.2	4.3	6.6	7.7	5.3

דירוג על פי ראנק

	Price	Weight	Learning Rate	Programmability Rate	controlability	reliability	maintenance
Arduino	1	2	3	3	1	5	5
Humanoid NAO	5	1	1	4	3	3	4
Hamster	2	5	2	1	4	4	3
Lego	3	3	4	5	5	1	1
Drone Quadcopter	4	4	5	2	2	2	2

דירוג על פי בורדה

	Price	Weight	Learning Rate	Programmability Rate	controlability	reliability	maintenance
Arduino	4	3	2	2	4	0	0
Humanoid NAO	0	4	4	1	2	2	1
Hamster	3	0	3	4	1	1	2
Lego	2	2	1	0	0	4	4
Drone Quadcopter	1	1	0	3	3	3	3

Topsis:

	Price	Weight	Learning Rate	Programmability Rate	controlability	reliability	maintenance

Arduino	7.47	6.73	7.88	3.50	7.96	1.14	3.83
Humanoid NAO	1.96	7.66	8.55	3.47	3.77	3.71	4.04
Hamster	4.7	1.8	8.2	9.4	1.2	3.2	5.1
Lego	3.5	6.6	4.4	2.2	1.1	8.5	5.6
Drone Quadcopter	2.2	4.4	1.2	4.3	6.6	7.7	5.3
	0.2	0.3	0.1	0.2	0.1	0.05	0.05

Step 2:
r1 = sqrt ((7.47 ^ 2) + (1.96^2) + (4.7^2) + (3.5^2) + (2.2^2)) = 9.94
r2 = sqrt ((6.73^2) + (7.66^2) + (1.8^2) + (6.6^2) + (4.4^2)) = 13.04
r3 = sqrt ((7.88 ^ 2) + (8.55^2) + (8.2^2) + (4.4^2) + (1.2^2)) = 14.94
r4 = sqrt((3.50^2) + (3.47^2) + (9.4^2) + (2.2^2) + (4.3^2)) = 11.66
r5 = sqrt ((7.96^2) + (3.77 ^ 2) + (1.2^2) + (1.1^2) + (6.6^2)) = 11.12
r6 = sqrt((1.14^2) + (3.71 ^ 2) + (3.2 ^ 2) + (8.5 ^ 2) + (7.7 ^ 2)) = 12.52
r7 = sqrt ((3.83^2) + (4.04^2) + (5.1^2) + (5.6^2) + (5.3^2)) = 10.79

	Price	Weight	Learning Rate	Programmability Rate	controlability	reliability	maintenance
Arduino	0.75	0.51	0.52	0.3	0.71	0.09	0.35
Humanoid NAO	0.19	0.58	0.57	0.29	0.33	0.29	0.37
Hamster	0.47	0.33	0.54	0.80	0.10	0.25	0.47
Lego	0.35	0.50	0.18	0.18	0.09	0.67	0.51
Drone Quadcopter	0.22	0.33	0.08	0.36	0.59	0.61	0.49

	0.2	0.3	0.1	0.2	0.1	0.05	0.05

Step 3:

	Price	Weight	Learning Rate	Programmability Rate	controlability	reliability	maintenance
Arduino	0.15	0.15	0.05	0.06	0.07	0.004	0.017
Humanoid NAO	0.03	0.17	0.05	0.05	0.03	0.014	0.018
Hamster	0.09	0.09	0.05	0.16	0.01	0.012	0.023
Lego	0.07	0.15	0.01	0.18	0.00	0.033	0.025
Drone Quadcopter	0.04	0.09	0.00	0.03	0.05	0.030	0.024
	0.2	0.3	0.1	0.2	0.1	0.05	0.05

Step 4:
Best: (0.15, 0.17, 0.05, 0.18, 0.07, 0.033, 0.025)
Worst: (0.03, 0.09, 0, 0.03, 0, 0.004, 0.017)

Step 5 (נחשב נקודה לדוגמה)
Humano (0.03 0.17 0.05 0.05 0.03 0.014 0.018)

Diw = sqrt((0.03-0.03)^2 + (0.17-0.09)^2 + (0.05 - 0)^2 + (0.05-0.03)^2 + (0.03-0)^2 + (0.014 - 0.004)^2 + (0.018 - 0.017)^2) = 0.10
Dib = sqrt((0.03-0.15)^2 + (0.17-0.17)^2 + (0.05 - 0.05)^2 + (0.05-0.18)^2 + (0.03-0.07)^2 + (0.014 - 0.33)^2 + (0.018 - 0.025)^2) = 0.36

Step 6: (לאותה נקודה ספציפית)

0.21 = (0.36+ 0.1) / 0.10

לא הכי גרועה , כמעט הכי גרועה אבל , 0 הכי גרוע , 1 הכי טוב.

תרגילון 5:

דרך אלגברית:

PR(A) = 0.5 + 0.5 PR(C)
PR(B) = 0.5 + 0.5 (PR(A) / 2)
PR(C) = 0.5 + 0.5 (PR(A) / 2 + PR(B))

PR(A) = 14/13
PR(B) = 10/13
PR(C) = 15/13

דרך איטרטיבית:

Iteration	PR(A)	PR(B)	PR(C)
0	1	1	1
1	1	0.75	1.125
2	1.062	0.765	1.148
3	1.074	0.768	1.152
4	1.076	0.769	1.153
5	1.076	0.769	1.153

דרך מטריצה:

0	0	1
0.5	0	0
0.5	1	0

עבודת בית 1:

דרך אלגברית:

PR(A) = 0.15 + 0.85(PR(C))
PR(B) = 0.15 + 0.85(PR(A)/2)
PR(C) = 0.15 + 0.85(PR(A)/2 + PR(B) + PR(D))
PR(D) = 0.15

$$\begin{cases} a = 1.49011 \\ b = 0.783296 \\ c = 1.5766 \\ d = 0.15 \end{cases}$$

דרך איטרטיבית

Iteration	PR(A)	PR(B)	PR(C)	PR(D)
0	1	1	1	1
1	1	0.575	2.275	0.15
2	2.08375	0.575	1.19125	0.15
3	1.162563	1.035594	1.651844	0.15
4	1.554067	0.644089	1.651844	0.15
5	1.554067	0.810479	1.485454	0.15
6	1.412636	0.810479	1.626885	0.15
7	1.532853	0.75037	1.566777	0.15
8	1.481761	0.801462	1.566777	0.15
9	1.481761	0.779748	1.588491	0.15
10	1.500218	0.779748	1.570034	0.15
11	1.484529	0.787592	1.577878	0.15
12	1.491197	0.780925	1.577878	0.15

דרך מטריצה:

A =

0.15	0	0	1	0
0.15	0.5	0	0	0
0.15	0.5	1	0	1
0.15	0	0	0	0

עבודה 2:
דרך אלגברית:

PR(A) = 0.5 + 0.5(PR(B)/2 + PR(C)/2 + PR(D))
PR(B) = 0.5 + 0.5(PR(A))
PR(C) = 0.5 + 0.5(PR(B)/2)
PR(D) = 0.5 + 0.5(PR(C)/2)

$$\begin{cases} a = 1.33962 \\ b = 1.16981 \\ c = 0.792453 \\ d = 0.698113 \end{cases}$$

d = 0.7

PR(A) = 0.3 + 0.7(PR(B)/2 + PR(C)/2 + PR(D))
PR(B) = 0.3 + 0.7(PR(A))
PR(C) = 0.3 + 0.7(PR(B)/2)
PR(D) = 0.3 + 0.7(PR(C)/2)

$$\begin{cases} a = 1.40502 \\ b = 1.28352 \\ c = 0.749231 \\ d = 0.562231 \end{cases}$$

d= 0.85

PR(A) = 0.15 + 0.85(PR(B)/2 + PR(C)/2 + PR(D))
PR(B) = 0.15 + 0.85(PR(A))
PR(C) = 0.15 + 0.85(PR(B)/2)
PR(D) = 0.15 + 0.85(PR(C)/2)

$$\begin{cases} a = 1.43582 \\ b = 1.37045 \\ c = 0.732441 \\ d = 0.461287 \end{cases}$$

שיעור בית 2:

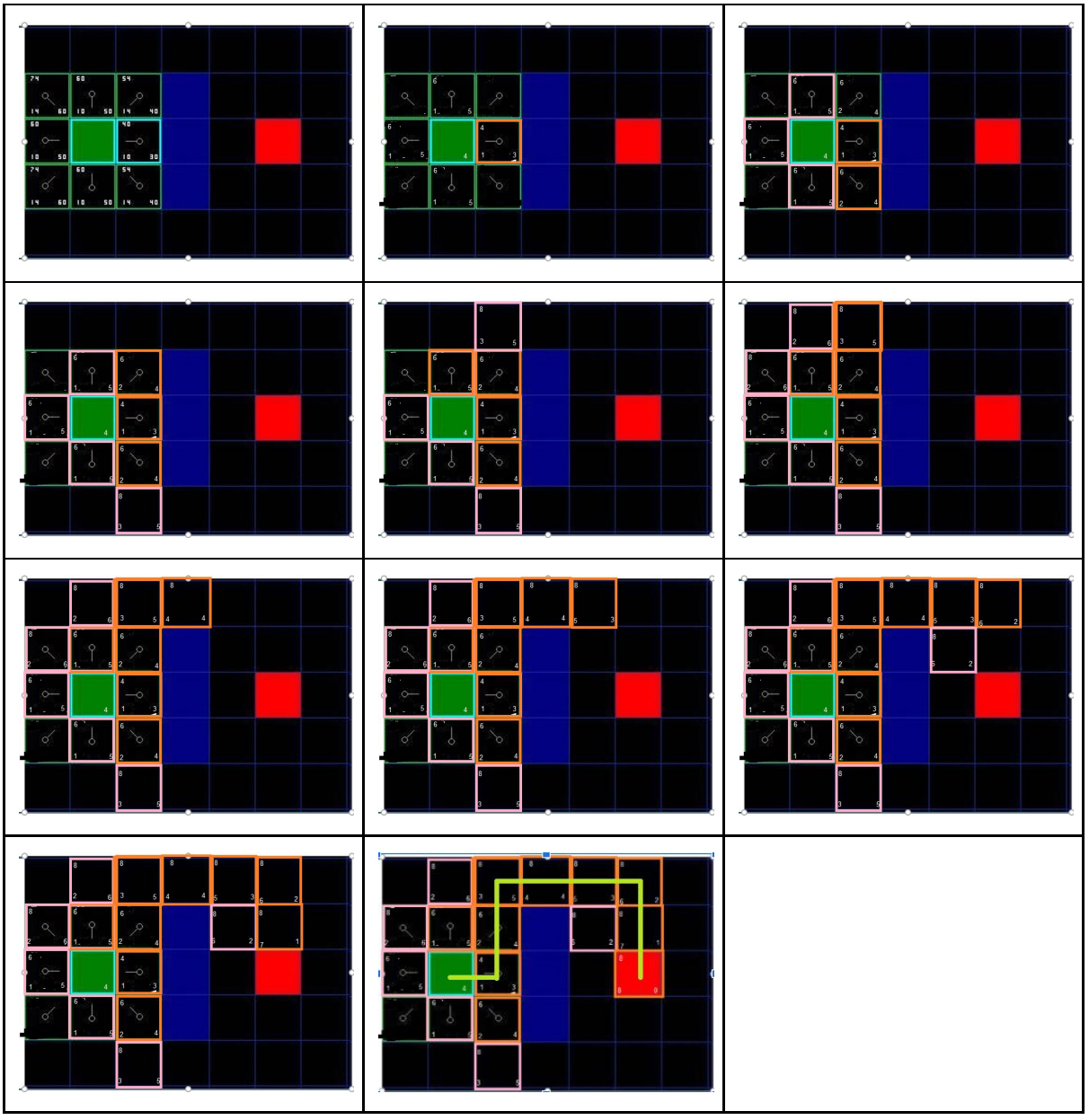
```
def pagerank(graph):
    """
    Graph object as input

    Returns a dictionary where the keys are the node names and the
values are
    the calculated pagerank score for that given node.
    """

    # Initialize values for all nodes s.t. that add up to one
    n = len(graph.nodes)
    init_val = 1.0/n
    ranks = dict(zip(graph.get_nodes(), [init_val] * n))

    new_ranks = ranks

    # Calculate new rank for each node
    for node, prev_rank in ranks.items():
        rank_sum = 0.0

        # Iterate through incoming nodes
        for incoming_node in node.inbound:
            numerator = ranks[incoming_node]
            denominator = len(incoming_node.outbound)
            transfer_amount = numerator / denominator

            # Transfer rank score
            new_ranks[incoming_node] = new_ranks[incoming_node] -
transfer_amount
            rank_sum = rank_sum + transfer_amount

        new_ranks[node] = ranks[node] + rank_sum
```

```
    # Set ranks to the new ranks calculated in this iteration
    ranks = new_ranks

    return ranks
```

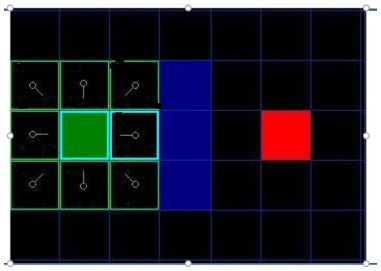
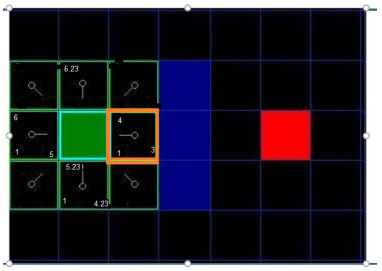
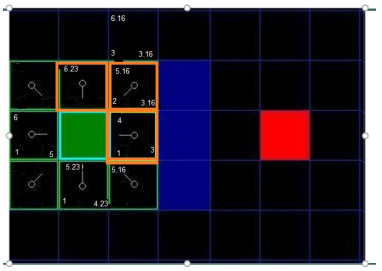
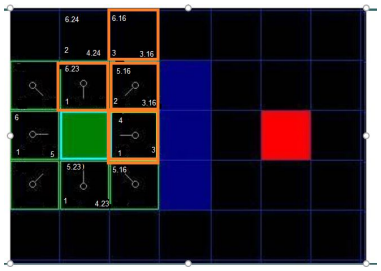
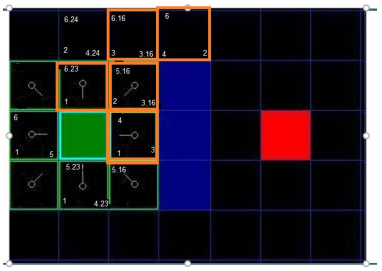
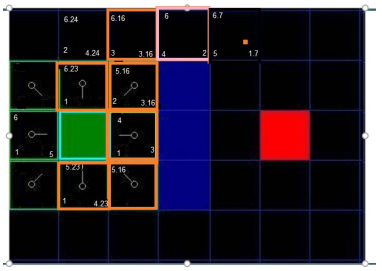
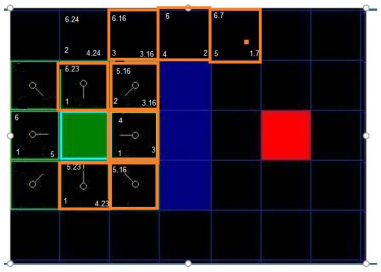
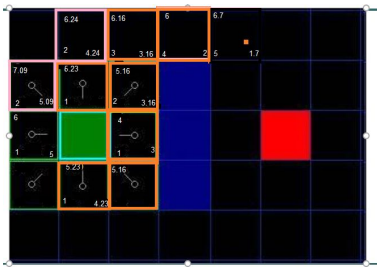
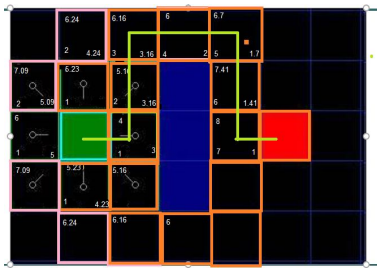
PR(A) = 1*(PR(D)+PR(B)+PR(C))
PR(B) = PR(C)=PR(D) = 0
=> PR(A) = 0

שאלה 3

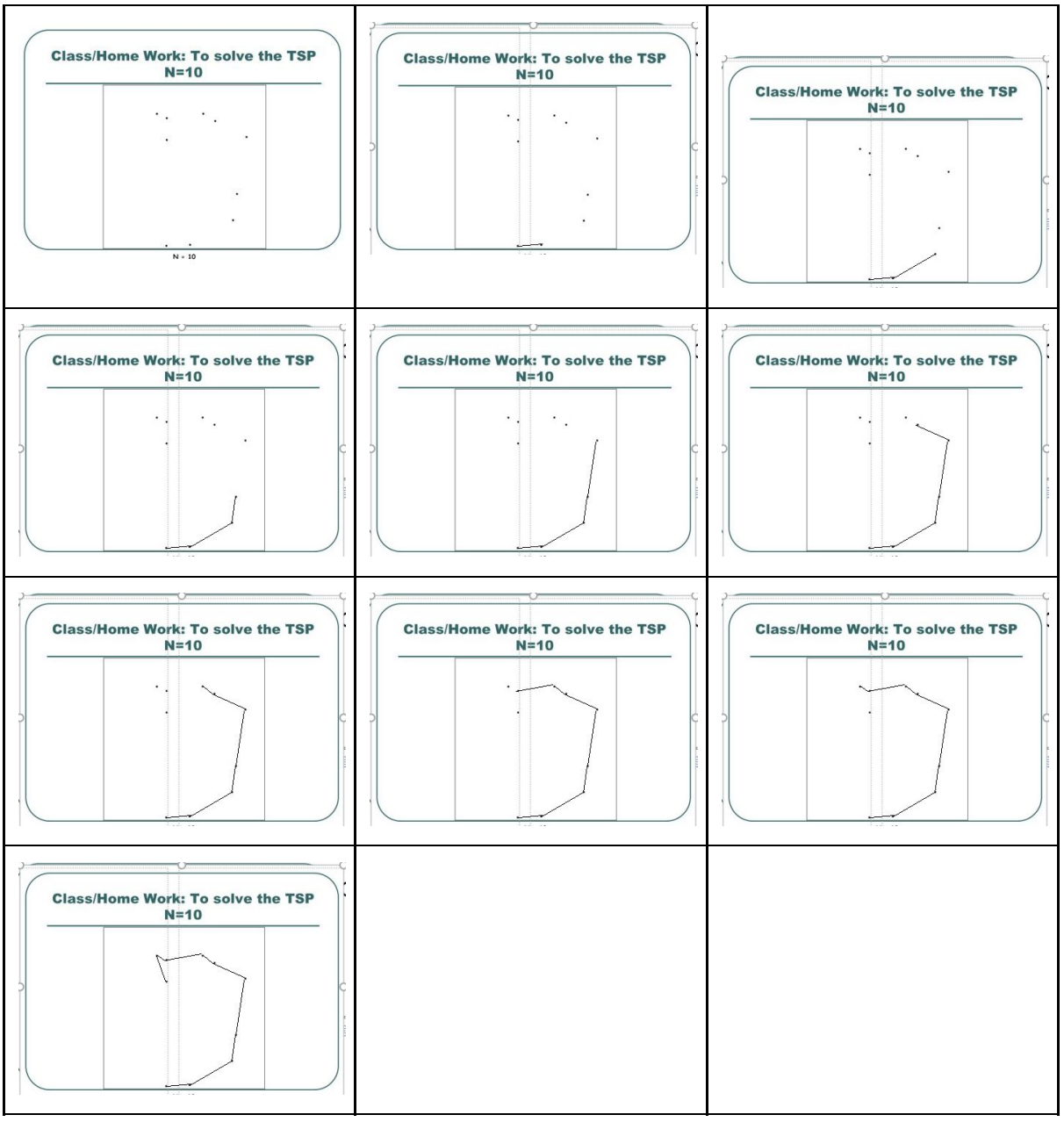
עם מאנטן.



עם אווילר:









…


שאלה 4 - TSP



שאלה 5:

20x1 +18x2 +17x3+15x4 +15x5 +10x6 +5x7 +3x8 +x9 +x10
Subject to
30x1 +25x2 +20x3+18x4 +17x5 +11x6 +5x7 +2x8 +x9 +x10

גריידי:
שלב ראשון נמיין על פי Pi/Si
0.66x1 + 0.72x2 +0.83x4 + 0.85x3+0.88x5+0.90x6+1x7+1x9+1x10 + 1.5x8

B = 30 =>
= 1x8 + 1x10 + 1x9 + 1x7 + 1x6 = 3 + 1 +1 + 5 + 1 + 10 = 21
2 + 1 + 1 + 5 + 11 = 20 < 30

B= 50
= 1x8 + 1x10 + 1x9 + 1x7 + 1x6 + 1x5 = 3 + 1 +1 + 5 + 1 + 10 + 15 = 36
2 + 1 + 1 + 5 + 11 + 17 = 37 < 50

B=70
= 1x8 + 1x10 + 1x9 + 1x7 + 1x6 + 1x5 +1x3 = 3 + 1 +1 + 5 + 1 + 10 + 15 + 17 = 53
2 + 1 + 1 + 5 + 11 + 17 + 20 = 57 < 70

B = 90
= 1x8 + 1x10 + 1x9 + 1x7 + 1x6 + 1x5 +1x3 +1x2 = 3 + 1 +1 + 5 + 1 + 10 + 15 + 17 +18 = 71
2 + 1 + 1 + 5 + 11 + 17 + 20 +25 = 82 < 90

B = 100
= 1x8 + 1x10 + 1x9 + 1x7 + 1x6 + 1x5 +1x3 +1x2 = 3 + 1 +1 + 5 + 1 + 10 + 15 + 17 +18 = 71
2 + 1 + 1 + 5 + 11 + 17 + 20 +25 = 82 < 100

B = 30

28

B= 50

45

B= 70

62

B= 90

75

B= 100

85

```
// Returns the maximum value that can be put in a knapsack of capacity
W
```

```
int knapSack(int W, int wt[], int val[], int n)
{
int i, w;
int K[n+1][W+1];

// Build table K[][] in bottom up manner
for (i = 0; i <= n; i++)
{
      for (w = 0; w <= W; w++)
      {
            if (i==0 || w==0)
                  K[i][w] = 0;
            else if (wt[i-1] <= w)
                        K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]],
K[i-1][w]);
            else
                        K[i][w] = K[i-1][w];
      }
}
```

מעבדות

מעבדה 1

1

```
def basicMathActions(a, b):
    print(a+b)
    print(a-b)
    print(a*b)
    print(a/b)
    print(a**b)
    print(a%b)

basicMathActions(2,4)
```

Result:

```
exercise1 ×
6
-2
8
0.5
16
2
```

2)

```
def hezka(a):
    print a**a

hezka(7)
hezka(2.2)
```

```
823543
5.666695778750081
```

3)

```
def circleAikefSetah(r):
    print( np.pi * 2 * r)
    return np.pi * (r**2)

print (circleAikefSetah(3))
```

```
18.84955592153876
28.274333882308138
```

4)

```python
def minMax(a,b,c):
    print(max(a,b,c))
    print(min(a,b,c))

minMax(1,2,3)
```

```
3
1
```

5)

```python
def isZiunBtvah(num):
    if num <= 100 and num >= 0:
        return 'Success'
    return 'Not in range'

print (isZiunBtvah(50))
print (isZiunBtvah(-20))
```

```
Success
Not in range
```

6)

```python
def printList(lis):
    for i in lis:
        print (i)

printList([0,5,4,1,2,3,4,5,6,7])
```

```
5
4
1
2
3
4
5
6
7
```

7)

```python
def printSorted(lis):
    for i in sorted(lis):
        print (i)

printSorted([0,5,4,1,2,3,4,5,6,7])
```

```
0
1
2
3
4
4
5
5
```

8)

```
printSorted(['a', 'c', 'd', 'f', 'b'])
```

```
a
b
c
d
f
```

9)

```
def sumAvg(lis):
    s = sum(lis)
    print (s)
    return (s)/(len(lis))

print (sumAvg([1,2,3,4]))
```

```
10
2.5
```

11)

```
def testSet():
    s = set([1, 2, 3, 4, 5])
    s.add(6)
    s.add(7)
    s.add(4)
    s.add(5)
```

12)

```
def twoObj():
    arr = [1,2,3]
    string = 'aada'

    print(dir(arr))
    print(dir(string))

twoObj()
```

```
exercise1
C:\Hit\Uristics\venv\Scripts\python.exe C:/Hit/Uristics/exercise1.py
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__',
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__
```

13)

```
def print100():
    l = list(range(1, 100))
    print(l[::3])

print(print100())
```

```
[1, 4, 7, 10, 13, 16, 19, 22, 25, 28, 31, 34, 37, 40, 43, 46, 49, 52, 55, 58, 61, 64, 67, 70, 73, 76, 79, 82, 85, 88, 91, 94, 97]
None
```

```python
import numpy as np

def basicMathActions(a, b):
    print(a+b)
    print(a-b)
    print(a*b)
    print(a/b)
    print(a**b)
    print(a%b)

def hezka(a):
    print (a**a)

def circleAikefSetah(r):
    print( np.pi * 2 * r)
    return np.pi * (r**2)

def minMax(a,b,c):
    print(max(a,b,c))
    print(min(a,b,c))

def isZiunBtvah(num):
    if num <= 100 and num >= 0:
        return 'Success'
    return 'Not in range'

def printList(lis):
    for i in lis:
        print (i)

def printSorted(lis):
    for i in sorted(lis):
        print (i)

def sumAvg(lis):
    s = sum(lis)
    print (s)
    return (s)/(len(lis))

def testSet():
    s = set([1, 2, 3, 4, 5])
    s.add(6)
    s.add(7)
    s.add(4)
    s.add(5)

def twoObj():
    arr = [1,2,3]
    string = 'aada'

    print(dir(arr))
    print(dir(string))

def print100():
    l = list(range(1, 100))
    print(l[::3])


#basicMathActions(2,4)

#hezka(7)
# hezka(2.2)
```

```
#print (circleAikefSetah(3))

#minMax(1,2,3)

#print (isZiunBtvah(50))
#print (isZiunBtvah(-20))

#printList([0,5,4,1,2,3,4,5,6,7])

#printSorted([0,5,4,1,2,3,4,5,6,7])

#printSorted(['a','c','d','f','b'])

#print (sumAvg([1,2,3,4]))

#testSet()

#twoObj()

#print(print100())
```

מעבדה 2

תרגיל 1

```
DijkstraFinder 302 34
+--------------------+
|      #             |
|      #           # |
|  s   #           # |
|  x   #           # |
|  x   #           # |
|  x   # xxx       # |
|  x   #x###x    ### |
|  x   # x # x   #   |
|  x   ###x#  x  # ##|
|  x     #x#   x #   |
| x#######x#    x# ##|
|  xxxxxxx #     x # |
|        ###      x# |
|        #         x |
|        #      ####x|
|        #      #   e|
|        #      #    |
|        #      #    |
|        #   ####    |
|        #   #       |
+--------------------+
```

```
AStarFinder 166 34
+--------------------+
|      #             |
|      #           # |
|  s   #           # |
|  x   #           # |
|  x   #           # |
|  x   # xxx       # |
|  x   #x###x    ### |
|  x   # x # x   #   |
|  x   ###x#  x  # ##|
|  x     #x#   x #   |
| x#######x#    x# ##|
|  xxxxxxx #     x # |
|        ###      x# |
|        #         x |
|        #      ####x|
|        #      #   e|
|        #      #    |
|        #      #    |
|        #   ####    |
|        #   #       |
+--------------------+
```

סעיף ב:

```
DijkstraFinder 310 49
+--------------------+
|      #             |
|      #           # |
|  s   #           # |
|  x   #           # |
|  x   #           # |
|  x   #xxxxxxxxx  # |
|  x   #x###    x### |
|  x   #xxx#    x#   |
|  x   ###x#    x# ##|
| xx     #x#    x#   |
| x#######x#    x# ##|
| xxxxxxxxx#    xxx# |
|        ###      x# |
|        #        xxx|
|        #      ####x|
|        #      #   e|
|        #      #    |
|        #      #    |
|        #   ####    |
|        #   #       |
+--------------------+
Without Diagonal:  0.05063438415527344
With Diagonal:  0.03459334373474121
```

```
AStarFinder 213 49
+--------------------+
|      #             |
|      #           # |
|  s   #           # |
|  x   #           # |
|  x   #           # |
|  x   #xxxxx      # |
|  x   #x###x    ### |
|  x   #xxx#xxxxx#   |
|  x   ###x#    x# ##|
| xx     #x#    x#   |
| x#######x#    x# ##|
| xxxxxxxxx#    xxx# |
|        ###      x# |
|        #        xxx|
|        #      ####x|
|        #      #   e|
|        #      #    |
|        #      #    |
|        #   ####    |
|        #   #       |
+--------------------+
```

תרגיל 2:

סעיף א:

מפה 25 על 25

```
DijkstraFinder 298 17
+-------------------------+
|                         |
| ###                     |
|    #                    |
| s  #                    |
| x  #        #           |
| x  # ###    #           |
| x  ##  #                |
|  x     #    #           |
|   x    #                |
|    x   #    #    #      |
|    x        #    #      |
|    x             ##     |
|     x  #####     #      |
|      x #     #          |
|       x#     #          |
|        xxxxe # ###      |
|                         |
|                         |
|                         |
|                         |
|                         |
|                         |
|                         |
|                         |
|                         |
|                         |
+-------------------------+
```

```
AStarFinder 66 17
+-------------------------+
|                         |
| ###                     |
|    #                    |
| s  #                    |
| x  #        #           |
|  x # ###    #           |
|   x##  #                |
|    x   #    #           |
|     x  #                |
|      x #    #    #      |
|       x     #    #      |
|       x          ##     |
|       x#####     #      |
|       x#     #          |
|       x#     #          |
|        xxxxe # ###      |
|                         |
|                         |
|                         |
|                         |
|                         |
|                         |
|                         |
|                         |
|                         |
|                         |
+-------------------------+
```

```
+-------------------------+
DijkstraFinder 288 24
+-------------------------+
|                         |
| ###                     |
|    #                    |
| sxx#                    |
|   x#        #           |
|   x# ###    #           |
|   x##  #                |
|   xxxxx#    #           |
|       x#                |
|       x#    #    #      |
|       x     #    #      |
|       x          ##     |
|       x#####     #      |
|       x#     #          |
|       x#     #          |
|       xxxxxe # ###      |
|                         |
|                         |
|                         |
|                         |
|                         |
|                         |
|                         |
|                         |
|                         |
+-------------------------+
Without Diagonal:  0.009519815444946289
With Diagonal:  0.013075590133666992
```

```
+-------------------------+
AStarFinder 90 24
+-------------------------+
|                         |
| ###                     |
|    #                    |
| sx #                    |
|  x #        #           |
|  x # ###    #           |
|  x ##  #                |
|  xx    #    #           |
|   xxxx #                |
|      x #    #    #      |
|      x      #    #      |
|      x           ##     |
|      x #####     #      |
|      x #     #          |
|      x #     #          |
|      xxxxxxe # ###      |
|                         |
|                         |
|                         |
|                         |
|                         |
|                         |
|                         |
|                         |
|                         |
+-------------------------+
```

מפה 30 על 30:

```
DijkstraFinder 94 10
+-----------------------------+
|                             |
| #                           |
|  #xxxxxx                    |
| sx# #   xe                  |
|   ###         #             |
|     #        ##             |
|     #       # #             |
|     #      #  #             |
|     #     #   #             |
|     #    #    #             |
|      ####     #             |
|               #             |
|               #             |
|               #             |
|               #             |
|               #             |
|               #             |
|               #             |
|               #             |
|               #             |
|               ##########    |
|                             |
|                       #     |
|                       #     |
|                       #     |
|                       #     |
|                       #     |
|                       #     |
|                       #     |
|                       #     |
+-----------------------------+
```

```
AStarFinder 14 10
+-----------------------------+
|                             |
| #                           |
|  #xxxxxx                    |
| sx# #   xe                  |
|   ###         #             |
|     #        ##             |
|     #       # #             |
|     #      #  #             |
|     #     #   #             |
|     #    #    #             |
|      ####     #             |
|               #             |
|               #             |
|               #             |
|               #             |
|               #             |
|               #             |
|               #             |
|               #             |
|               #             |
|               ##########    |
|                             |
|                       #     |
|                       #     |
|                       #     |
|                       #     |
|                       #     |
|                       #     |
|                       #     |
|                       #     |
+-----------------------------+
```

```
+------------------------------+
|xxxxxxxxxxx                   |
|x#        x                   |
|xx#       x                   |
| s # #    e                   |
|   ###         #              |
|     #        ##              |
|     #       # #              |
|     #      #  #              |
|     #     #   #              |
|     #    #    #              |
|      ####     #              |
|               #              |
|               #              |
|               #              |
|               #              |
|               #              |
|               #              |
|               #              |
|               #              |
|               #              |
|               ##########     |
|                              |
|                      #       |
|                      #       |
|                      #       |
|                      #       |
|                      #       |
|                      #       |
|                      #       |
|                      #       |
+------------------------------+
Without Diagonal:  0.008548021316528
With Diagonal:  0.00701761245727539
```

```
+------------------------------+
AStarFinder 57 18
+------------------------------+
|xxxxxxxx                      |
|x#      x                     |
|xx#     xxxx                  |
| s # #    e                   |
|   ###         #              |
|     #        ##              |
|     #       # #              |
|     #      #  #              |
|     #     #   #              |
|     #    #    #              |
|      ####     #              |
|               #              |
|               #              |
|               #              |
|               #              |
|               #              |
|               #              |
|               #              |
|               #              |
|               ##########     |
|                              |
|                      #       |
|                      #       |
|                      #       |
|                      #       |
|                      #       |
|                      #       |
|                      #       |
|                      #       |
```

מפה 40 על 40:

```
C:\Users\Shahar\AppData\Local\Programs\Python\Python37-32\python.exe
"C:/Hit/Uristics Mabda/path_find/test/path_test.py"
C:\Hit\Uristics Mabda\path_find\test
AStarFinder 12 10
+----------------------------------------+
|   #   # ####      #                    |
|        #        #          # #        #|
|    #xx# # #   ##    # #        # #    #|
| sxxx##xxxe##  #           #      #     |
|   ## #     # #      #  #      #        |
| #  #      #  #      ##     ###    ##  #|
|    #       #     #  #      #       ##  |
|   # #  #   #      #  #   # #   #       |
|#     ###  #         #      ##   #   # #|
```

```
|## #    ##    #   #       #          #  |
|  #     # ##         #    #      #   #  |
|   #   ###  #      #  ##      # #       |
| #  #          #    #  #  # ##    ##  #|
| #  #  #                     #      ###|
|            #    #     #        ##     |
|  #    #      # # # # #  #    #  #     |
| #       #         #  #        # ##   # |
| #    ## #         #   #   #  #     # # |
|  #        #  #            ##          |
|#  ##        #    #    #  # ##  #  # #  |
|                     # ## #          # |
|    ##   ##               #      ##   #|
|##                # ## #             #|
|     #      # #    #        ###   #    |
|#   #     ###  ###       #      #   #  |
|     ##           #   #     #   #     # |
|   #  ###   #### ##          ## #      |
|              #   #   #            #  |
| #   #   ## #               #         |
| #   #       ##      ##     #   # ##  ##|
|           #    #    #        ##    #  |
|          ###       # ##  ##           |
|           # ##      #    #    #  ##   #|
|    # #    #  # #      ###     # #  #  |
|        #       ##       ## #  # ##   # |
| #       ## ##      #  # #          ## |
| #         # ## #  ## ##   #           |
|# ##      #  ##  #   #  #   # ##    ##  |
|      # ## #       ##  #   #    #    #  |
|     #       #     #  # # #  #      ### |
+----------------------------------------+
DijkstraFinder 85 10
+----------------------------------------+
|   #   # ####      #                   |
|        #        #          # #       #|
|    #xx# # #   ##    # #        # #    #|
| sxxx##xxxe##  #           #      #    |
|   ## #     # #      #  #      #       |
| #  #      #  #      ##     ###    ##  #|
|    #       #     #  #      #       ##  |
|   # #  #   #      #  #   # #   #      |
|#     ###  #         #      ##   #   # #|
|## #    ##    #   #       #          #  |
|  #     # ##         #    #      #   #  |
```

```
|   #   ###  #      #  ##      # #       |
| #  #          #    #  #  # ##    ##  #|
| #  #  #                     #      ###|
|            #    #     #        ##     |
|  #    #      # # # # #  #    #  #     |
| #       #         #  #        # ##   # |
| #    ## #         #   #   #  #     # # |
|  #        #  #            ##          |
|#  ##        #    #    #  # ##  #  # #  |
|                     # ## #          # |
|    ##   ##                #      ##   #|
|##                # ## #             #|
|     #      # #    #        ###   #     |
|#   #     ###  ###       #      #   #   |
|     ##           #   #     #   #     # |
|   #  ###   #### ##          ## #       |
|              #   #   #            #  |
| #   #   ## #                #         |
| #   #       ##      ##     #   # ##  ##|
|           #    #    #        ##    #  |
|          ###       # ##  ##            |
|           # ##      #    #    #  ##   #|
|    # #    #  # #      ###     # #  #   |
|        #       ##       ## #  # ##   # |
| #       ## ##      #  # #          ## |
| #         # ## #  ## ##   #            |
|# ##      #  ##  #   #  #   # ##    ##  |
|      # ## #       ##  #   #    #    #  |
|     #       #     #  # # #  #      ### |
+----------------------------------------+
AStarFinder 109 34
+----------------------------------------+
|   #   # ####      #                    |
|        #        #          # #        #|
|    #  # # #   ##    # #        # #    #|
| sx  ##   e##  #           #      #     |
|  x## #   x # #      #  #      #        |
| #x #     x#  #      ##     ###    ##  #|
|  x #     x #     #  #      #       ##  |
|  x# #  # x #      #  #   # #   #       |
|# xxxx### x#         #      ##   #   # #|
|## # xx ##xxxx#   #       #          #  |
|  #   x # ## x       #    #      #   #  |
|   #  x###  #x     #  ##      # #       |
| #  # xxxxxxxx  #    #  #  # ##    ##  #|
```

```
| #  #  #                      #      ###|
|            #    #     #        ##     |
|  #    #      # # # # #  #    #  #      |
| #       #         #  #        # ##   # |
| #    ## #         #   #   #  #     # # |
|  #        #  #            ##         |
|#  ##        #    #    #  # ##  #  # #  |
|                     # ## #          # |
|    ##   ##                #      ##   #|
|##                # ## #             #|
|     #      # #    #        ###   #     |
|#   #     ###  ###       #      #   #   |
|     ##           #   #     #   #     # |
|   #  ###   #### ##          ## #       |
|              #   #   #            #  |
| #   #   ## #                #         |
| #   #       ##      ##     #   # ##  ##|
|           #    #    #        ##    #  |
|          ###       # ##  ##           |
|           # ##      #    #    #  ##   #|
|    # #    #  # #      ###     # #  #   |
|        #       ##       ## #  # ##   # |
| #       ## ##      #  # #          ## |
| #         # ## #  ## ##   #           |
|# ##      #  ##  #   #  #   # ##    ##  |
|      # ## #       ##  #   #    #    #  |
|     #       #     #  # # #  #      ### |
+----------------------------------------+
DijkstraFinder 371 34
+----------------------------------------+
|   #   # ####      #                    |
|        #        #          # #        #|
|    #  # # #   ##    # #        # #    #|
| sx  ##   e##  #           #      #     |
|  x## #   x # #      #  #      #        |
| #x #     x#  #      ##     ###    ##  #|
|  x #     x #     #  #      #       ##  |
|  x# #  # x #      #  #   # #   #       |
|# xxxx### x#         #      ##   #   # #|
|## # xx ##xxxx#   #       #          #  |
|  #   x # ## x       #    #      #   #  |
|   #  x###  #x     #  ##      # #       |
| #  # xxxxxxxx  #    #  #  # ##    ##  #|
| #  #  #                      #      ###|
|            #    #     #        ##     |
```

```
|  #    #      # # # # #  #    #  #      |
| #       #         #  #        # ##   # |
| #    ## #         #   #   #  #     # # |
|  #        #  #             ##         |
|#  ##        #    #    #  # ##  #  # #  |
|                     # ## #          # |
|    ##   ##                #      ##   #|
|##                # ## #              #|
|     #      # #    #        ###   #     |
|#   #     ###  ###       #      #   #   |
|     ##           #   #     #   #     # |
|   #  ###   #### ##          ## #       |
|               #   #   #             #  |
| #   #   ## #                #          |
| #   #       ##      ##     #   # ##  ##|
|            #    #    #        ##    #  |
|          ###       # ##  ##            |
|           # ##      #    #    #  ##   #|
|    # #    #  # #      ###     # #  #   |
|        #       ##       ## #  # ##   # |
| #       ## ##      #  # #           ## |
| #         # ## #  ## ##   #            |
|# ##      #  ##  #   #  #   # ##    ##  |
|      # ## #       ##  #   #    #    #  |
|     #       #     #  # # #  #      ### |
+----------------------------------------+
Without Diagonal:  0.017546653747558594
With Diagonal:  0.010498046875

Process finished with exit code 0
```

תרגיל 3:

א) דיקסטרה - אלגוריתם שמוצא את המסלול הקצר ביותר בגרף, הוא מחשב את המסלול הקצר ביותר בין צומת לכל שאר הצמתים, והוא מוצא פתרון אופטימלי לבעיה.

אלגוריתם A* - הוא אלגוריתם יוריסטי שמוצא מסלול קצר בין שני נקודות, הוא לא בהכרח ימצא את המסלול הקצר ביותר, אבל הוא יעבור בפחות צמתים על מנת למצוא מסלול קצר, ולכן הוא מהיר יותר מדיקסטרה.

ב)

node.py - מייצג צומת במטריצה, נשמור עבור כל צומת האם ביקרנו בו, קורדינטות, ומרחקים יורסטים ורגילים, והאם הוא בכלל פתוח או סגור.

heuristic.py - מכיל אוסף פונקציות על מנת לחשב מרחק יורסטי בין צמתים שונים , כל פונקציה היא צורה אחרת למדוד מרחק יוריסטי.

grid.py - אחרי לתרגם את המטריצה לצמתים , הוא מכיל או

סף פונקציות עזר כמו השגת השכנים של הצומת, האם צומת בתוך המפה, האם אפשר ללכת לעבור באותו צומת (האם מכשול), והצגת המפה בקונסול.
util.py - מכיל פונקצית עזר על מנת לשחזר את המסלול הקצר שמצאנו.

a_star.py - מכיל את האלוגריתם של A* למציאת המסלול
dijkstra.py - מכיל מימוש של דיקסטרה על פי A* כך שקובעים הפונקציה h תחזיר תמיד 0.

path_test.py - מכיל קובץ שמריץ את האלגוריתם על המפות שיצרנו.

ג)

```
+----------------------------------------+
Without Diagonal:  18.334452867507935
With Diagonal:  11.009337425231934
```

ד)
runs - מסמן את מספר הקודקודים שעברנו בהם על מנת למצוא את המסלול.
path - מסמן את אורך המסלול שמצאנו.

שאלה 4:

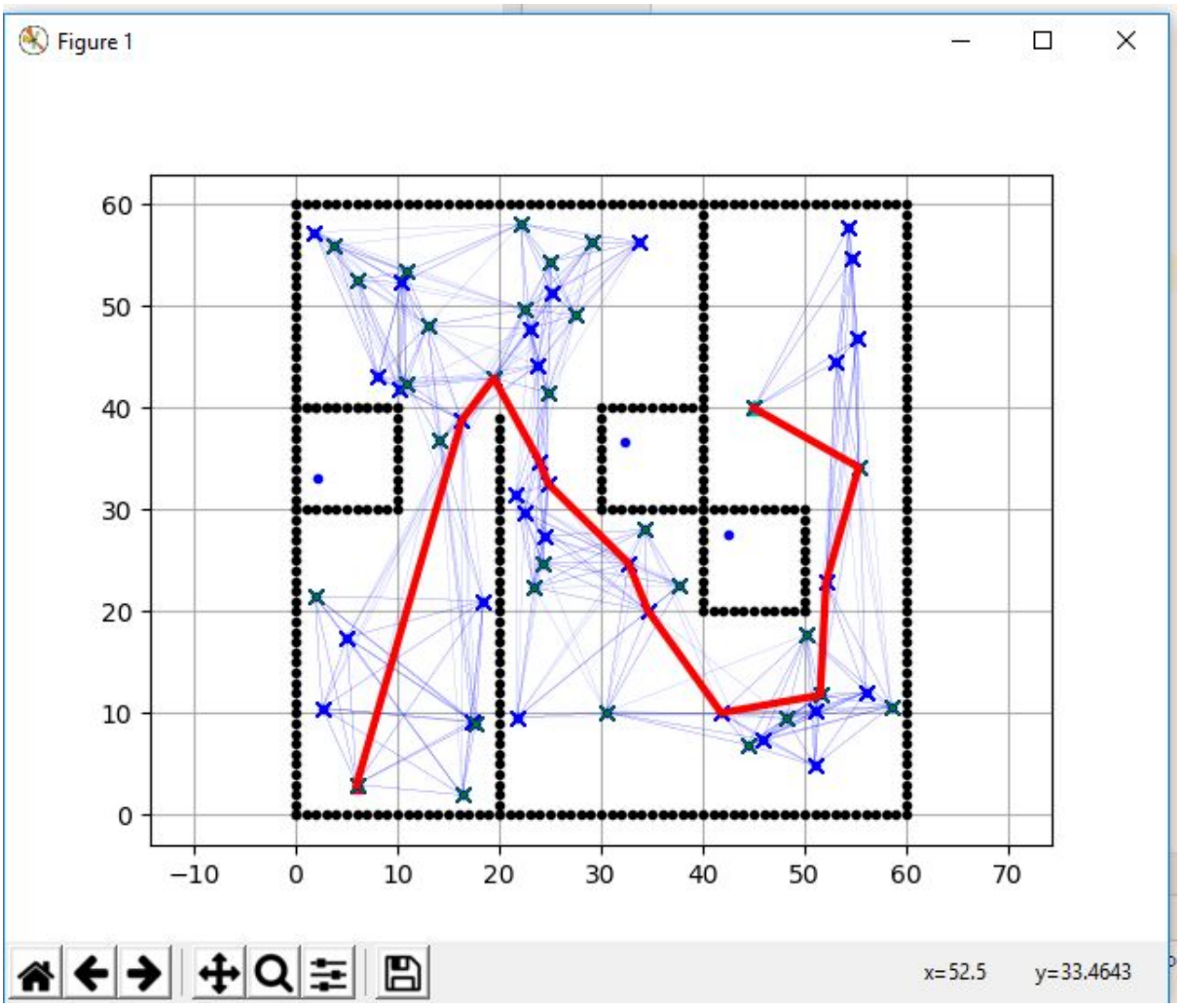
שאלה 5:
בעיה 1: מציאת מסלול המהיר ביותר בין מקום X למקום Y, ואפשר להשתמש בפונקציה היורסטית להתחשב בכמות הפקקים שיש בדרך , על מנת להביא מסלול שנגיע אליו יותר מהר.
בעיה 2: מציאת חניה לאנשים בחניון , לא חייבים להביא את המסלול הקצר ביותר, החשיבות שיותר אנשים יכנסו לחניון במהירות.

מעבדה 3

מטלה מספר 3:

סעיף א:

```
def main():
    print(__file__ + " start!!")

    # start and goal position
    sx = 6.0  # [m]
    sy = 3.0  # [m]
    gx = 45.0  # [m]
    gy = 40.0  # [m]
```



סעיף ב:

זמן מציאת מסלול יצא לי

```
total time %s 4.397373914718628
```

דיוק: האלוגריתם לא הכי מדוויק, ניתן לראות שקיימים מסלולים קצרים יותר, כל ריצה המפה משתנה, וניתן לקבל מסלול טוב יותר.
נפילות: קיימים מספר פעמים שהאלוגריתם רץ, והוא לא מצליח למצוא מסלול הקצר ביותר.
סעיף ג:
ככל שנגדיל יותר את גודל הרובוט יגדל הסיכוי לניפול,
כאשר הרובוט יהיה קטן יותר יגדל הסיכוי שהוא ימצא מסלול.

סעיף ד:
ככל שנעלה את כמות הכדורים נגלה שיש פחות נפילות, אך גם ניראה שזמן מציאת המסלול מתארך, ככל שמעלים את מספר הכדורים שיש.

סעיף ה:
על המרחק המקסימלי בין הצמתות השונות, אם המרחק יהיה גדול מהמותר, לא יווצר קשת מתאימה על הצומת השכן.

סעיף ו

```
# Global parameters
N_SAMPLE = 40    # number of sampled points, default 500
N_KNN = 10  # number of edge from one sampled point, default 10
MAX_EDGE_LEN = 30.0  # [m] Maximum edge length, default 30.0
SHOW_MAP = True # Show map with vertices and edges
ROBOT_SIZE = 0.5 # [m]
show_animation = True
```

הוא נותן מהירות גבוה, הרובוט יכול לעבור דרך המכשולים כי הוא קטן, ויש מספיק נקודות כדי לייצר מסלול מתאים, ואחוזי נפילות מאד נמוכות.