Oriya Yehudai 211544150

Shira Taitelbaum 322207341

בס"ד

מבוא לרובוטיקה – תרגיל 3

תיאור האלגוריתם

פסאודו קוד

הרצת הסימולטור מורכבת מ-2 פונקציות עיקריות בקובץ ה-krembot.ino.cpp, נפרט על הפסאודו קוד בכל אחת מהן.

בפרוייקט שלנו יצרנו מחלקה בשם Node, שמייצגת תאים בגרידים השונים. למחלקה הזו יש מס' שדות, ביניהם: id,x,y,weight, neighbors ועוד.

נסמן: D = גודל הרובוט.

פונק' setup:

1. אתחל גריד ביניים (uniformGrid) של התאים והמשקלים לפי הגריד והמשקלים המקוריים, כך שכל גודל תא בגריד הביניים הוא בגודל D*D.
2. אתחל גריד סופי (coarseGrid) של התאים והמשקלים לפי הגריד והמשקלים מהשלב הקודם (גריד הביניים), כך שכל גודל תא בגריד הסופי הוא 2*2.
3. אתחל מטריצה של אובייקטי Node (nodesMatrixUni) בגודל של גריד הביניים, המייצגת את התאים בגריד הביניים בתור אובייקטים.
4. אתחל מטריצה של אובייקטי Node (nodesMatrixCoarse) בגודל של הגריד הסופי, המייצגת את התאים בגריד הסופי בתור אובייקטים. נסמן את מס' התאים בגריד הסופי ב-L.
5. אתחל מטריצת שכנויות ריבועית (neighborsMatrix) בגודל L*L שתייצג את השכנויות של הקודקודים ב - nodesMatrixCoarse.

   אופן האתחול:

   אם אין קשת בין הקודוקד ה-i לקודקוד ה-j, אז $neighborsMatrix[i][j] = -1$

   אם יש קשת בין הקודוקד ה-i לקודקוד ה-j, אז $neighborsMatrix[i][j] = \max(w(i), w(j))$, כאשר *w(k)* מייצג את המשקל של קודקוד *k*.
6. הרץ את אלגוריתם פרים מקודקוד ההתחלה (של הגריד הסופי) בו נמצא הרובוט, למציאת עץ פורש מינימום, על מטריצת השכנויות neighborsMatrix, ושמור את צלעות העץ במשתנה *mst*.

7. אתחל מטריצה dirMatrix המייצגת את הצלעות הקיימות בכל תא בגריד הסופי, כאשר כל תא במטריצה מורכב מ4 שדות של *bool*, המייצגים את ארבעת השכנים האפשריים של קודקוד, וכאשר יש קשת בין 2 קודקודים, אתחל את הכיוונים שלהם להיות *true* בהתאם לכיוון הקשתות המחברות ביניהם.
8. צור את המסלול להרצת הרובוט במפה –

אתחל משתנה *current* שישמור את ה-*node* הנוכחי עליו עוברים.

אתחל משתנה *path* ריק שיכיל אובייקטים מסוג *Node*.

אתחל משתנה *neighbors* ריק שיכיל את השכנים של *current*.

אתחל משתנה *blackNodes* ריק שיכיל קודקודים שכבר ביקרנו בהם.

דחוף את ה-*node* של תא המיקום ההתחלתי של הרובוט בגריד האמצעי לתוך *neighbors*.

עבור בלולאה:

- אם *neighbors* ריק, צא מהלולאה.
- אחרת – $current = neighbors.front()$
- דחוף את *current* לתוך *path*.
- דחוף את *current* לתוך *blackNodes*.
- שמור בתוך *neighbors* רק את השכנים של *current* שהוא יכול להגיע אליהם לפי ה-*mst*, וגם שלא נמצאים ב-*blackNodes* , כלומר שלא ביקרנו בהם עוד (השמירה דורסת את האובייקטים שהיו מהאיטרציה הקודמת, ולא מוסיפה אליהם).

לאחר הלולאה דחוף את ה-*node* של תא המיקום ההתחלתי של הרובוט בגריד האמצעי לתוך *path*, על מנת ליצור מסלול מעגלי.

פונק' loop:

1) אתחל שני משתנים מסוג *Node*: current,next*, ומשתנה גלובלי $loopIndex = 0$.
2) אם לא סיימנו לעבור על כל הקודקודים ב-*path*, שמור את הקודקוד הנוכחי שהרובוט נמצא בו ב-*current*, ואת הקודקוד הבא במסלול ב-*next*.

אחרת, סיימנו לעבור על כל הקודקודים ב-*path* – עבור למצב *Stop*.

3) מצב *move* –

אם לא הגענו לתא הבא במפה לפי התא ש-*next* מסמל, אז המשך בנסיעת הרובוט.

אחרת - הגענו לתא הבא במפה לפי התא ש-*next* ממל, אז עצור את נסיעת הרובוט, והחלף למצב סיבוב.

קדם את אינדקס ה-*loop*: $loopIndex++$.

4) מצב *turn* –

חשב את הזווית שהרובוט צריך לזוז לפיה כדי להגיע מ-*current* ל-*next*.

אם הרובוט לא הגיע לזווית הרצויה, אז חשב את מהירות הסיבוב וסובב את הרובוט.

אחרת – הרובוט כן הגיע לזווית הרצויה, אז עצור את הרובוט ועבור למצב *move*.

5) מצב *stop* – עצור את הרובוט.

אופן בחירת הכיוונים

יצרנו פונקציה (קראנו לה ב-*setup* כאשר אתחלנו את משתנה *path* השומר את המסלול בו הרובוט ילך במפה) שבודקת איפה תא קטן (של הגריד האמצעי) נמצא ב-*MegaCell* (תא גדול בגריד הסופי בגודל 2*2), מתוך 4 האופציות. לאחר מכן, לכל מיקום אפשרי מארבעת המקרים, עברנו בפונק' על כל האופציות לקשתות ב-*megaCell*, כאשר הסתכלנו רק על הקשתות שנשארו בעץ הפורש. בהתאם לכך דחפנו לרשימה את הקודקודים הרלוונטיים שהרובוט יכול לזוז אליהם מהמיקום הנוכחי שלו. בכל דחיפה כזאת, דחפנו קודם את התאים הטריוויאליים (לדוג': אם יש רק צלע ימינה, ואנו נמצאים במקרה של תא תחתון ימני ב-*MegaCell*, אז נדחוף את השכן הימני של התא הנוכחי), ולאחר מכן דחפנו את התאים האחרים של מקרים מיוחדים יותר (בהמשך לדוגמא הקודמת, אם יש רק צלע ימינה, ואנו נמצאים במקרה של תא תחתון ימני ב-*MegaCell*, אז נדחוף את השכן השמאלי של התא הנוכחי, כי אולי נרצה להסתובב סביב הקשת, ולא ללכת בכיוונה).

לבסוף, לאחר שמצאנו את כל הקודקודים שהרובוט יכול לזוז אליהם מהתא הנוכחי, מחקנו את הקודקודים שכבר הוספנו ל-*path* מרשימת השכנים. אם רשימת השכנים לא נשארה ריקה בסופו של דבר, אז שלפנו את הקודקוד הראשון מהרשימה והוספנו אותו למסלול של הרובוט מהתא הנוכחי. כך בעצם הגדרנו את בחירת הכיוונים.

הקפת העץ

הקפת העץ תלויה במיקום ההתחלתי של הרובוט ובתאים של המסלול שנוצר (גם המסלול תלוי בעץ הפורש ובאופן בחירת הכיוונים שהסברנו לעיל), לכן בכל *seed* שונה הרובוט יקיף את העץ מכיוון שונה ולא קבוע.

feedback-control

השתמשנו ב-linear feedback-control, וגם ב-angular feedback-control ב-loop.

במצב move השתמשנו ב-linear feedback-control באמצעות הפונקציה שנלמדה בתרגול got_to_cell,

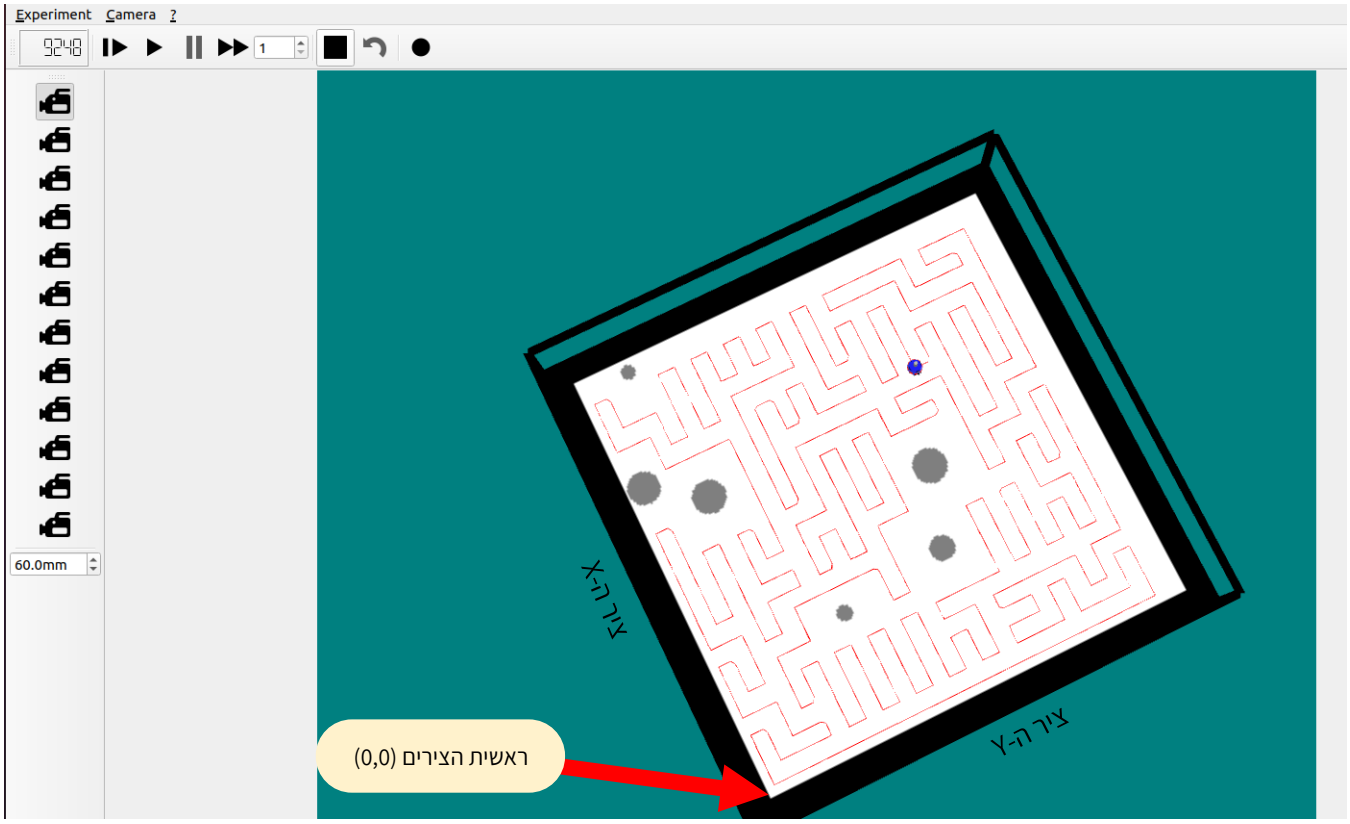
שמשתמשת ב-threshold של 0.0009.

במצב turn השתמשנו ב-angular feedback-control באמצעות הפונקציה שנלמדה בתרגול

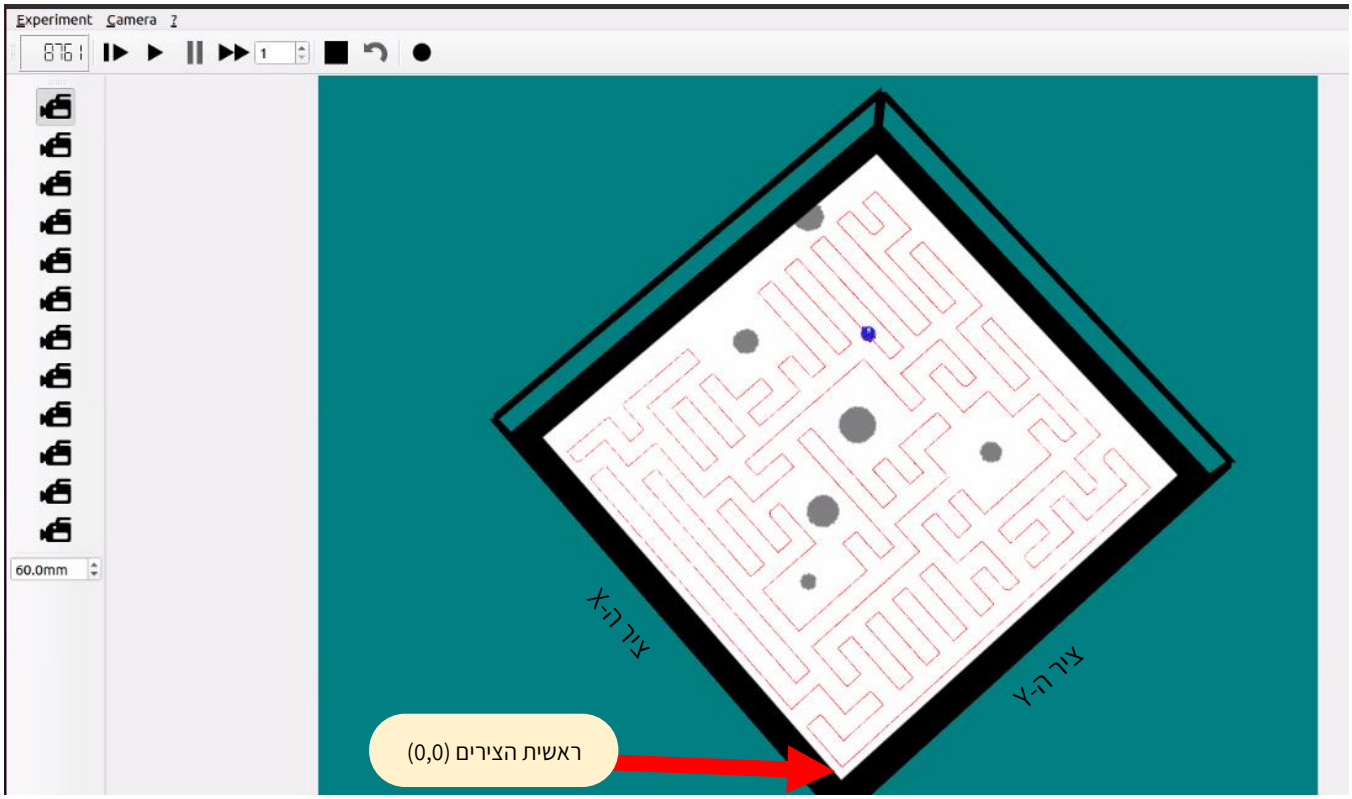
got_to_orientation, שמשתמשת ב-threshold של 0.5 .

ביצועים

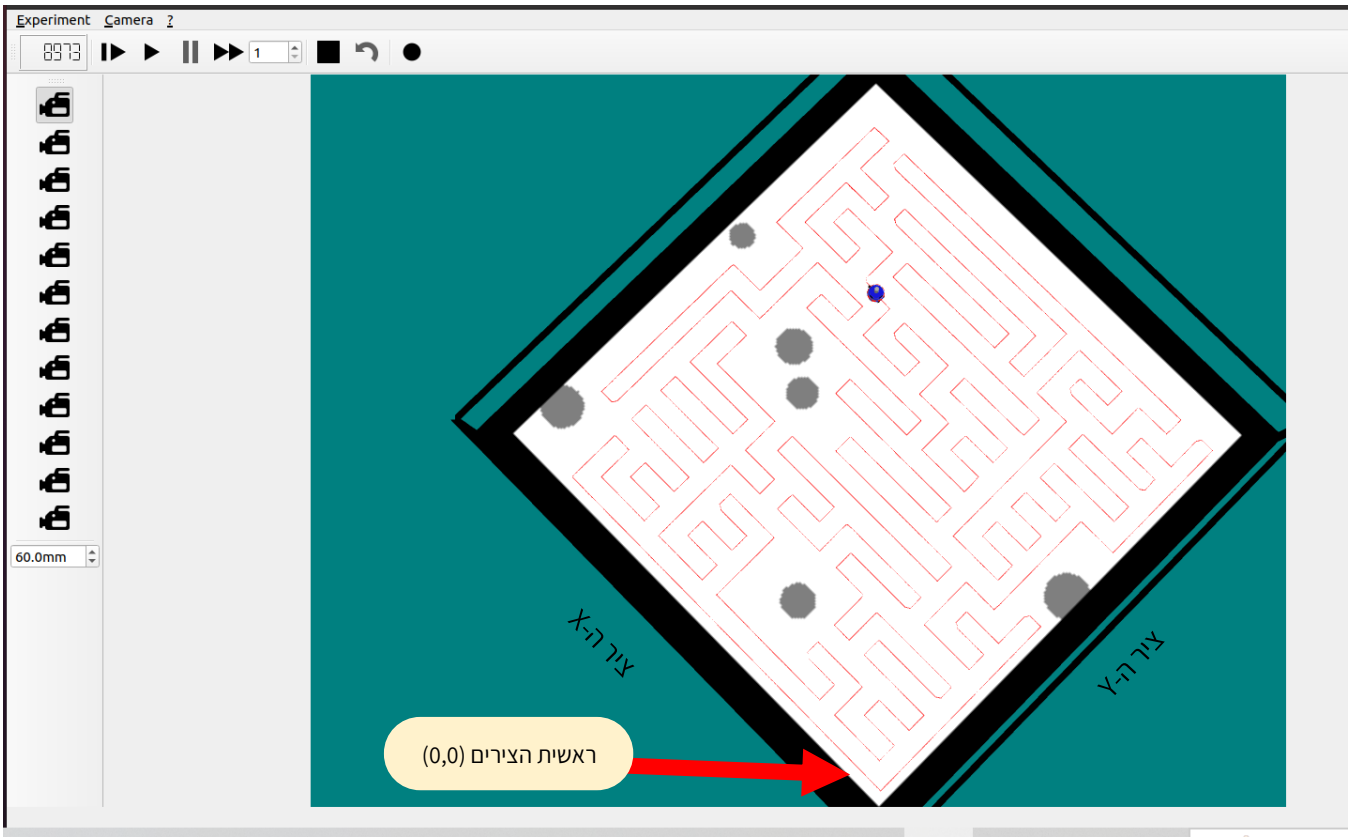
- Seed 1
- כפי שניתן לראות בצילום המסך, הצלחנו ליצור כיסוי מעגלי לעץ.
- צילום מסך:



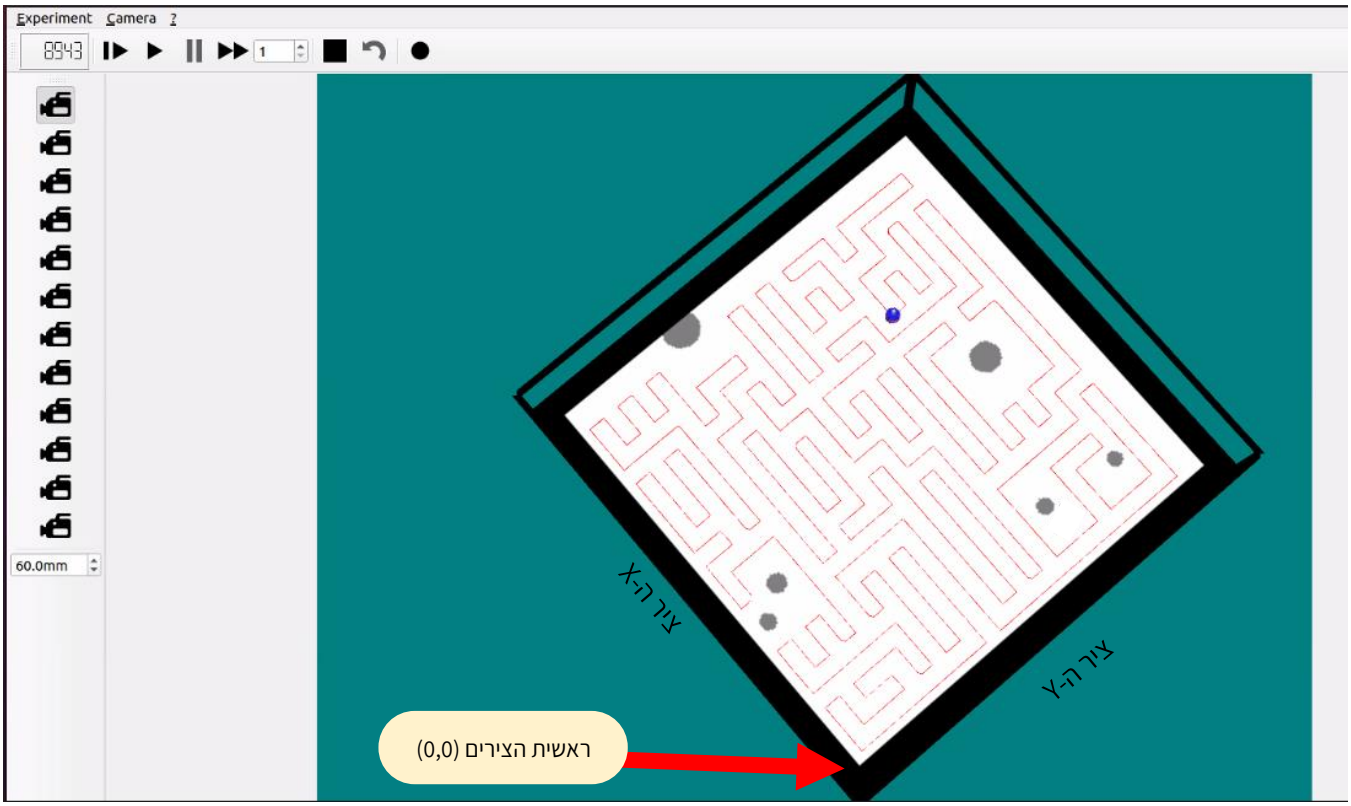
- Seed 2
- כפי שניתן לראות בצילום המסך, הצלחנו ליצור כיסוי מעגלי לעץ.
- צילום מסך:



- Seed 4
- כפי שניתן לראות בצילום המסך, הצלחנו ליצור כיסוי מעגלי לעץ.
- צילום מסך:



- Seed 6
- כפי שניתן לראות בצילום המסך, הצלחנו ליצור כיסוי מעגלי לעץ.
- צילום מסך:



- Seed 7
- כפי שניתן לראות בצילום המסך, הצלחנו ליצור כיסוי מעגלי לעץ.
- צילום מסך: