Link-a-Pix

פרויקט גמר בקורס מבוא לבינה מלאכותית

2020

מגישים:

רז לוי

גפן ברנמן

עמית דוד

איימי אילנה צביקה

תוכן עניינים

הגדרת המשחק:

בדומה למשחק "שחור ופתור" המטרה היא לצבוע לוח משבצות עד לקבלת תמונה. כל לוח משחק מכיל מספרים במקומות שונים. כל מספר מהווה את אורך המסלול שיוצא ממנו. הערך במשבצת המקור שווה לערך של משבצת היעד, וזהו אורך המסלול ביניהן (כולל המשבצות הממוספרות). הצביעה היא לאורך ולרוחב, ללא אלכסונים. משבצת המכילה את הספרה 1 מחוברת לעצמה. קיימת למשחק גירסא נוספת עם צבעים, בה בנוסף למספרים יש להתאים בין הצבעים של הספרות.

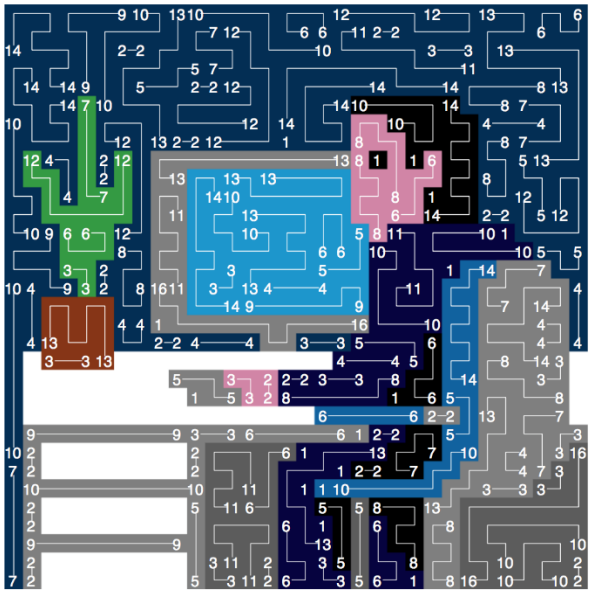
אילוצי המשחק:

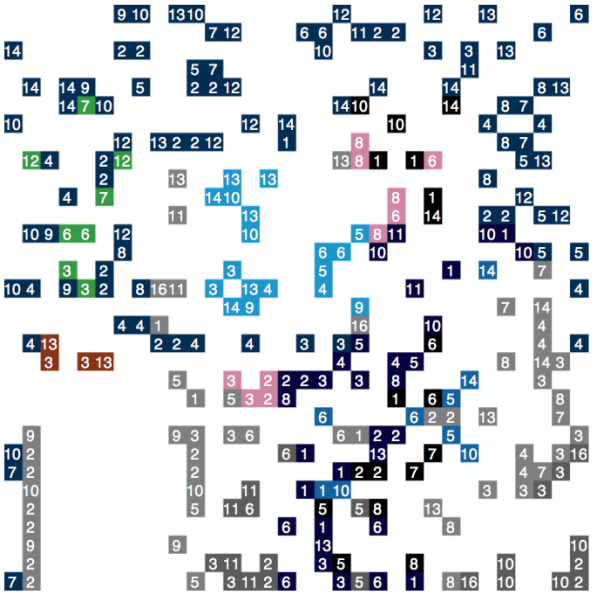
- מסלול מתחיל ונגמר במשבצות בעלות אותו מספר וצבע ובאורך המספר.
- דרך כל משבצת עובר בדיוק מסלול אחד, כלומר מסלולים לא חותכים אחד את השני.
- יכולות להשאר משבצות שנותרו לבנות ולא נצבעו באף צבע ושלא עובר בהן מסלול.

מצורף קישור למשחק:

https://www.conceptispuzzles.com/index.aspx?uri=puzzle/link-a-pix

משמאל זהו הלוח ההתחלתי עם המשבצות הממוספרות והצבועות ומימין הלוח הפתור והסופי.





ייצוג הבעיה כבעיית חיפוש:

בפרויקט זה האתגר הוא לממש אלגוריתם יעיל לפתירת לוחות של המשחק בגדלים שונים. הפתרון הוא מציאת סדרת מסלולים חוקיים תחת האילוצים על מנת לקבל את התמונה הסופית (לכל לוח יש פתרון יחיד). ניתן לראות מחוקי המשחק כי קיים דמיון בינו לבין בעיות חיפוש שראינו בקורס ולכן החלטנו להגדיר את הבעיה שלנו כבעיית חיפוש כאשר עלינו למצוא את הפתרון החוקי עבור כל לוח.

פתרון ראשון כבעיית CSP:

ניתן להסתכל על הבעיה כאוסף של תתי בעיות שכל תת בעיה היא מציאת מסלול בין שתי משבצות שמכילות את אותו צבע ואותו מספר כך שהמסלול באורך הנדרש. כלומר פתרון אוסף תתי הבעיות יהיה מציאת השמה חוקית של המסלולים והצבעים למשבצות בלוח כך שכל אילוצי המשחק מסתפקים.

Constraint Satisfaction Problems

- A Constraint Satisfaction Problems (CSP) consists of
  - a set of variables $\{X_1, X_2, \ldots, X_n\}$ to which
  - values $\{d_1, d_2, \ldots, d_k\}$ can be assigned
  - such that a set of constraints over the variables is respected
- A CSP is solved by a variable assignment that satisfies all given constraints.

נגדיר את הבעיה כבעיית CSP:

נגדיר השמה חוקית ללוח $n \times m$ באופן הבא:

- נסמן את מספר המשבצות הממוספרות שאינן 1 ב-k ואת מספר המשבצות הממוספרות ב-1 ב-r. אזי כמות המסלולים בלוח תהיה $\frac{k}{2} + r$.
- מסלול בין שתי משבצות יהיה באורך המספר שכתוב על שתי המשבצות (כל מסלול יתחיל וייגמר במשבצת עם אותו מספר וצבע).
- דרך כל משבצת עובר בדיוק מסלול אחד, כלומר מסלולים לא חותכים אחד את השני.

מצב הוא השמה חלקית של המסלולים על הלוח.
מצב התחלתי הוא לוח ריק בו יש משבצות ממוספרות ובעלות צבע ללא מסלולים צבועים.
מצב המטרה הוא לוח בו מלאים כל המסלולים באופן חוקי המציגים את התמונה הסופית.
פונקציית המעברים בין מצבים היא צביעה או מחיקה של מסלול יחיד על הלוח.

מימוש באמצעות backtracking:

מאחר ועבור כל משבצת ממוספרת יכולים להיות מספר מסלולים מתאימים אך רק אחד הוא חלק מהפתרון הנכון החלטנו להשתמש תחילה בשיטת ה-backtrack שכשאר מגיעה למבוי סתום חוזרת אחורה ומשנה את בחירתה ובכך לבסוף מגיעה לפתרון הנכון.

פסודוקוד לריצת ה-backtrack:

קלט: לוח, איטרטור על המשבצות הממוספרות (variable_selection) והיוריסטיקה (עבור הלוח – heuristic)

- נקבל מהאיטרטור את המשבצת הראשונה בסידור – (x, y)
- נקבל את כל ה-possible_moves מהמשבצת (x, y) נסמן – paths
- נמיין את paths על פי ההיוריסטיקה שקיבלנו (כתלות בלוח הנוכחי)
- עבור כל מסלול ב-paths בצע :
  - נצבע את המסלול הנוכחי על הלוח.
  - אם המסלול מוביל ללוח חוקי, נעבור למשבצת הבאה על ידי קריאה רקורסיבית
  - אם המסלול לא מוביל ללוח חוקי נחזיר את הלוח הנוכחי (לאחר מחיקה של המסלול הנוכחי) ואת המסלול שמחקנו.
- החזר None

Variable Selection:

במהלך ריצת ה-backtrack קיים מעבר על המשבצות הממוספרות בסדר מסויים. שמנו לב כי יש חשיבות גדולה מאוד לסדר המעבר על המשבצות, ולכן החלטנו ליצור מספר שיטות למעבר על המשבצות.

Top to bottom:

היוריסטיקה דיפולטיבית. מסדרת את רשימת המשבצות הממוספרות כך שהלוח יתחיל להתמלא מלמעלה ועד למטה.

Small to big:

ההיוריסטיקה מסדרת את רשימת המשבצות הממוספרות לפי המספרים על המשבצות. כלומר הלוח יתחיל להתמלא ממשבצות שעליהן הספרה 1, לאחר מכן יעבור למשבצות שעליהן הספרה 2 וכן הלאה עד שהלוח יקבל השמה מלאה וחוקית. השתמשנו בהיוריסטיקה זאת מכיוון שלמשבצות הממוספרות בספרה 1 יש תמיד מסלול יחיד ובכך אנחנו מונעים מחיקה עתידית של מסלולים אלו במהלך ריצת הbacktrack.
בנוסף כאשר עוברים על המשבצות לפי המספר בסדר עולה אנחנו יודעים שלרוב למשבצות עם ספרות קטנות יהיו מספר מסלולים קטן יותר מאשר משבצות עם מספרים גדולים ולכן כשנגיע למשבת עם ספרה יתכן שחלק מהמסלולים האפשריים כבר לא יהיו חוקיים (כיוון שהמצשבצות שלהן כבר נצבעו) ובכך נחסוך בדיקות.

LCV - Least Constrained Values:

ההיוריסטיקה מסדרת את רשימת המשבצות הממוספרות ע"פ כמות המסלולים האפשריים החוקיים עבור כל משבצת (בהתאם ללוח הנוכחי, כלומר אנחנו מעדכנים את סדר הרשימה לאחר כל פעולה על הלוח) – מהקטן לגדול. כפי שלמדנו בכיתה נבחר עבור כל משתנה ערך שהכי פחות מאלץ את השכנים של המשתנה הנוכחי. לכן אנחנו מסדרים את רשימת המשבצות הממוספרות שלנו לפי סדר עולה כך שהמשבצות שיש להן מסלול חוקי אחד בגרף ראשונות ובכך כשאנחנו צובעים את המסלולים הללו אנחנו יודעים שאנחנו לא חוסמים או משפיעים על מסלולים אחרים עתידיים בגרף.

MRV - Most Remaining Values:

ההיוריסטיקה מסדרת את רשימת המשבצות הממוספרות ע"פ כמות המסלולים האפשריים החוקיים עבור כל משבצת (בהתאם ללוח הנוכחי, כלומר אנחנו מעדכנים את סדר הרשימה לאחר כל פעולה על הלוח) – מהגדול לקטן.

כפי שלמדנו בכיתה נבחר את המשתנה שלו יש את הכי הרבה אופציות לצעד הבא. לכן אנחנו מסדרים את רשימת המשבצות הממוספרות שלנו לפי סדר יורד כך שהמשבצות שיש להן הכי הרבה מסלולים חוקיים בגרף יהיו ראשונות.

By Bullet – חלוקה של הלוח:

נחלק את הלוח ל-9 משבצות. סדר צביעת המסלולים (המעבר על המשבצות הממוספרות) יהיה לפי מעבר על המשבצות מהחלוקה של הלוח.
לדוגמא פה נעבור על המשבצות לפי סדר המספרים, ובכל משבצת נעבור על כל התאים הממוספרים בה:

1	4	7
2	5	8
3	6	9

Random Selection:

ההיוריסטיקה מסדרת את רשימת המשבצות הממוספרות ע"פ סדר רנדומלי. הוספנו את ההיוריסטיקה הזאת על מנת לראות את ההבדלים בין ההיוריסטיקות השונות וכמה הן משפיעות על זמני הריצה ומהירות מציאת הפתרון (יפורט בהמשך)

פתרון שני כבעיית חיפוש רגילה:

בבעיית החיפוש הרגילה מימשנו **ארבעה** אלגוריתמי חיפוש: DFS, BFS, UCS – uniform cost search, A*.
מצב הוא לוח ומסלול נוכחי אותו נרצה לצבוע בשלב הנוכחי. במצב ההתחלתי ניצור successors עבור כל המסלולים האפשריים, לאחר מכן עבור כל לוח ניצור עבורו את ה-successors שלו וכן הלאה.
החיפושים משתמשים במבני הנתונים – Stack, Queue, Priority Queue, על מנת לשמור את המצבים שלנו בחלק מהם לפי סדר מסוים.

מימוש באמצעות DFS:

מרחיב כל פעם את הקדקוד העמוק ביותר ב-fringe. הוא ממשיך במעמקיי העץ, עד שהוא מגיע לקדקוד שאין לו בנים, ולאחר מכן ממשיך את החיפוש בקדקוד הכי עמוק הבא, שעוד יש לו בנים שלא הורחבו.
בפתרון שלנו, האלגוריתם תחילה ינסה למלא את כל הלוח, עד שמגיע למבוי סתום, ולאחר מכן ינסה בדרך אחרת, עד שיעבור על כל הדרכים. אם איפשהו בדרך מצא פתרון – יעצור.

מימוש באמצעות BFS:

אסטרטגיית חיפוש פשוטה, בה קדקוד השורש הוא הראשון שמורחב, לאחר מכן כל הבנים שלו, ואז כל הבנים שלהם וכן הלאה. כלומר - לא נרחיב קדקודים מרמה מסוימת לפני שכל הקדקודים ברמה הקודמת לה הורחבו.
המימוש באמצעות תור.
בפתרון שלנו, האלגוריתם יעבור תחילה על כל הלוחות שבהם יש מסלול יחיד צבוע, לאחר מכן על כל הלוחות עם 2 מסלולים צבועים וכך הלאה.
זהו אלגוריתם "נאיבי" **ומאוד** לא יעיל, לכן הנחנו שלא יגיע לפתרון חוקי בזמן ריצה סביר.

מימוש באמצעות UCS:

אסטרטגיית חיפוש שמבוססת על BFS, רק שכאן יורחבו קודם הקדקודים שה-"מחיר" להגיע אליהם היה הנמוך ביותר. המימוש יהיה באמצעות תור קדימויות על פי $g(board)$ - שתחזיר מהו "המחיר" של הלוח.
בפתרון שלנו $g(board)$ יהיה מספר המשבצות הריקות על הלוח. לכן, לוח עם פחות משבצות ריקות, ויותר משבצות צבועות יקבל $g(board)$ יותר נמוך ולכן האלגוריתם יגיע אליו קודם.

מימוש באמצעות A* search:

הצורה נוספת וידועה של best-first-search הינה A*. האלגוריתם מעריך את הקודקודים על פי שילוב של המחיר עד ההגעה לקדקוד n ומסומן ב-$g(n)$ ו-$h(n)$ – המחיר המוערך להגיע מ-n אל קודקוד המטרה.
רעיון האלגוריתם הוא לא להרחיב קודקודים שהם יקרים מלכתחילה. בבעיה שלנו החלטנו שכל קודקוד יהיה לוח ועבור כל לוח אנחנו מחשבים את $g(board), h(board)$ והסכום של שניהם יהיה המחיר של הלוח. כך האלגוריתם יחליט כיצד לסדר את הלוחות בתור הקדימויות.

היוריסטיקות (בשימוש ע"י *A ו-CSP):

Null Heuristic:

היוריסטיקת האפס. ההיוריסטה מחזירה עבור כל לוח את התוצאה 0 ולכן הופכת את החיפוש לחיפוש BFS רגיל מאחר ואנו מרחיבים את כל המצבים העתידיים בעומק עולה. לרוב משתמשים בהיוריסטיקה זאת על מנת להשוות את ריצת הבעיה עם היוריסטיקות שונות.

Stick To Walls:

ההיוריסטיקה מעדיפה מסלולים שנצמדים לקירות על פני מסלולים שמתרחקים מהם. עבור כל משבצת במסלול אנו בודקים כמה היא קרובה לאחד הקירות וככל שיש יותר משבצות שקרובות לקיר כך המסלול מקבל מחיר גבוה יותר וטוב יותר. כלומר המחיר המוחזר הוא כמות המשבצות במסלול שצמודות לקירות של הלוח.
לאחר מספר הרצות של המשחק שמנו לב שלרוב המסלולים שלא חוסמים מסלולים אחרים הם מסלולים הנצמדים לקירות ולכן בחרנו להשתמש בהיוריסטיקה זאת.

Stick To Other Paths:

ההיוריסטיקה מעדיפה מסלולים שנצמדים למסלולים אחרים על פני מסלולים שמתרחקים ממסלולים קיימים עבור כל משבצת במסלול אנו בודקים האם היא מוקפת באחד מהשכנים שלה בשכן שכבר צבוע וככל שיש יותר משבצות שקרובות למסלול קיים כך המסלול מקבל מחיר גבוה יותר וטוב יותר. כלומר המחיר המוחזר הוא כמות המשבצות במסלול שמוקפות במשבצת צבועה באחד מצדדיה.

Count Possible Paths:

ההיוריסטיקה סוכמת את מספר המסלולים האפשריים מנקודת ההתחלה, ומנקודת הסוף. ומחזירה את הערך הגדול יותר מבין ההופכיים של המספרים האלה.

Linear Combination:

קומבינציה לינארית של כל ההיוריסטקיות שהזכרנו לעיל.

$$10 \cdot stick\ to\ walls\ +\ 5 \cdot stick\ to\ other\ paths + count\ possible\ paths\ + 15 \cdot stick\ to\ paths\ or\ walls$$

Neural Network:

מוסבר בפירוט בעמוד הבא.

Invalid State:

ההיוריסטיקה מחזירה מינוס אינסוף במידה והלוח שהתקבל הוא לא חוקי כדי שנדע לא לבחור במסלול הנוכחי כיוון שהוא גורר לוח לא חוקי, כלומר בהכרח צריך לבחור במסלול אחר שיוביל ללוח אחר.
אנחנו משתמשים בהיוריסטיקה הזו בכל ריצה.

פתרון שלישי כבעיית Machine Learning – בעזרת Neural Network:

ניתן להסתכל על הבעיה כבעיה שניתן לפתור באמצעות Machine Learning, החלטנו לפעול באופן הבא:

יצרנו את הפיצ'רים הבאים:

- אורך ורוחב הלוח
- מספר הצבעים על הלוח
- אחוז התאים הצבועים על הלוח כאשר הוא פתור (ניתן להגיע לערך זה גם שהלוח ריק ע"י סכימה חכמה של הספרות על הלוח)
- נקודת התחלה במסלול – (x, y)
- נקודת סיום במסלול – (end_x, end_y)
- מסלול מנורמל (תמיד מתחיל ב-(0,0)) – יצרנו לכל צורת מסלול שמצאנו dummy_variable (עמודה ייחודית בשבילו).
- Label – ציון למסלול –
  - 100 למסלול נכון
  - 0 למסלול לא נכון

<u>הערה:</u> למרות שאנו נותנים ערכים "בולייאנים" למסלולים, הרשת תחזיר מספר בין 0 ל-100 שמייצג את כמה הרשת "בטוחה" במסלול שמצאנו, כאשר 0 אומר שהרשת בטוחה שמסלול זה לא נכון ו-100 כאשר הרשת בטוחה שהמסלול נכון.

הסיבה שנירמלנו את המסלולים היא שמתוך ההנחה שהרשת תמצא צורת מסלול הנפוצה ביותר (ללוחות בגדלים שונים) ותבחר את צורה זו קודם (דוגמא, ייתכן שעבור לוח קטן ומסלולים באורך 6 שהרשת מעדיפה מסלולים מאוד ישרים, אבל מעדיפה מסלולים מפותלים אם אורך המסלול הוא 7 וגודל הלוח הוא בינוני).

עברנו על מאגר של לוחות פתורים וידועים מראש, ממנו הוצאות מאגר מסלולים נכונים ולא נכונים (40,000 מסלולים סה"כ), ושמרנו את train_set על הדיסק בקובץ csv.

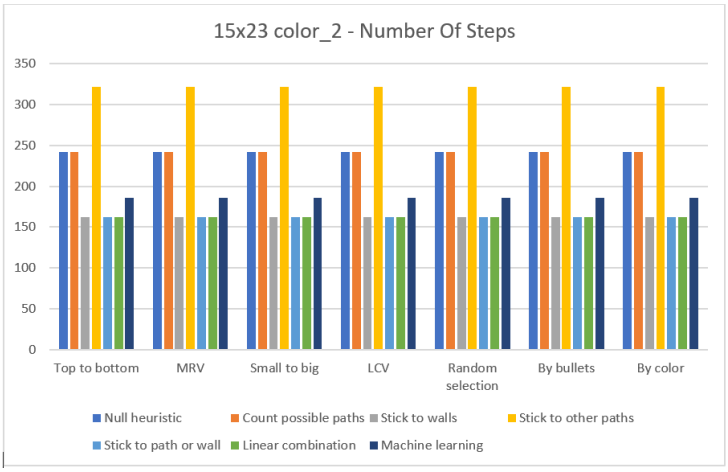
את ה-**Neural Network** אימנו בעזרת ה-train_set שבנינו אם הפרמטרים הבאים:

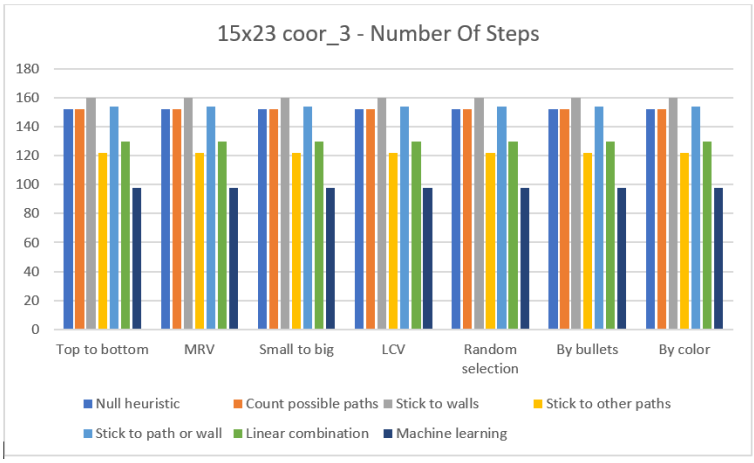
- בעזרת רמה מוסתרת אחת בעלת 10 ניורונים
- שימוש באקטיבציה לוגיסטית (**Logistic Activation**: $f(x) = \frac{1}{1+e^{-x}}$)
- שימוש ב-adam solver, שזוהי אופטימיזיה של **Stochiastic Gradient Descent**.

גם את הרשת שיצרנו שמרנו על הדיסק, וזאת כדי לא לאמן אותה שוב מחשב בכל פעם שמריצים את הקוד.

במהלך ריצת ה-GUI ניתן לבחור בהיוריסטיקה Machine Learning, והתוכנית כבר יודעת לבנות את הרשת מהקובץ ששמור בתיקייה learner באופן אוטומטי.

תוצאות:





מסקנות:

- מאילוצי חומרה, השתמשנו רשת פשוטה מאוד ובמאגר מצומצם של מסלולים (במקור המאגר היה בגודל של 70,000 מסלולים וגם מאגר זה לא השתמש בכל התמונות שהיו ברשותינו). אילו היה לנו יותר זמן ומחשב חזק יותר, התוצאות שאליהן הייתה הרשת מגיעה היו משתפרות אפילו יותר.
- זמן הריצה של הרשת ארוך יותר מזמן הריצה של ההיוריסטיקות האחרות (ותופעה זו הגיונית ביוון שהרשת צריכה לבצע הרבה יותר חישובים לעומת ההיוריסטיקות האחרות). אולם בלוחות גדולים, הרשת תבצע הרבה פחות מהלכים, ולכן זמן הריצה יתתקזז ולעיתים יהיה אפילו מהיר יותר.
- הרשת כמו שהיא כרגע מגיע לתוצאות טובות ובאיכות דומה לזו של ההיוריסיטיקה **Linear Combination.** במקרה הגרוע ביותר קיבלנו תוצאה פחות טובה, אך עדיין יותר טוב מההיוריסטיקה **Null Heuristic** (ז"א שהרשת איננה בוחרת מסלולים "גרועים" שמאריכים את הריצה שלא כצורך).

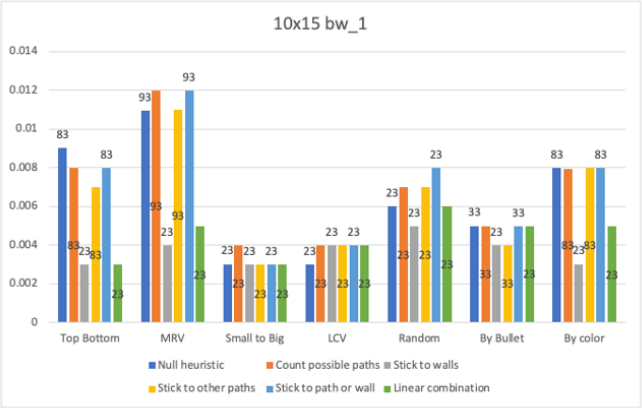
השוואות של זמני ריצה ויעילות :

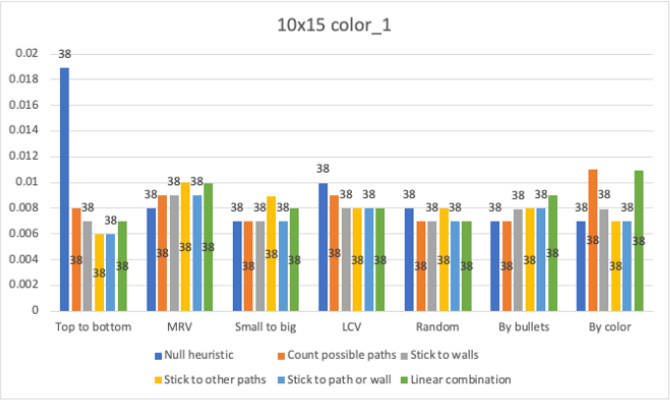
הערות על אופן קריאת הגרפים:

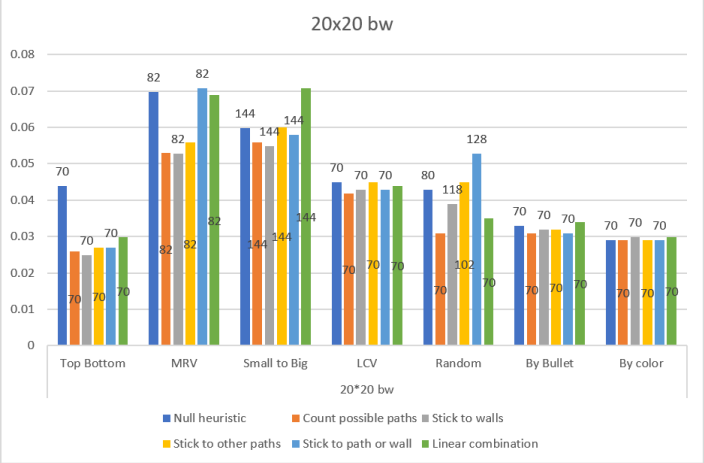
- כל הגרפים מציגים את זמני הריצה על לוחות שונים כתלות בהיוריסטיקה ובסדר של ה- variable selection (בהתאם לסידור שבחרנו).
- על כל עמודה בגרף יש מספר, או את הסימן inf. המספר מייצג את מספר הצעדים שנדרשו כדי להגיע לפיתרון. במידה ולא מצאנו פתרון חוקי אחרי 3 דקות של ריצה, יופיע הסימן inf. (במקרה זה, זמן הריצה יהיה 999)

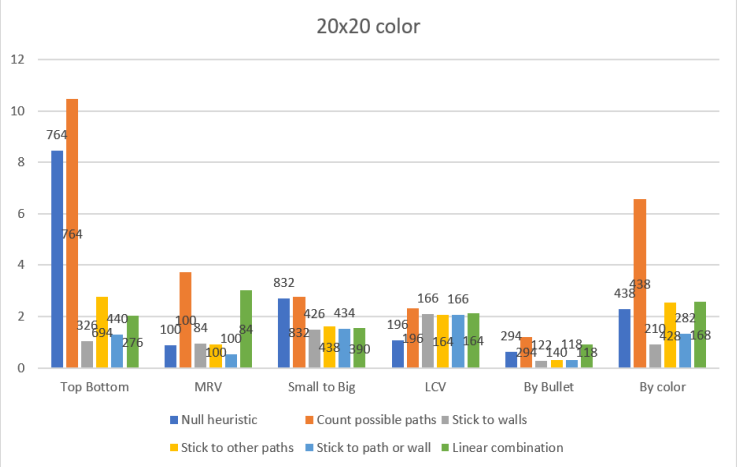
הרצנו את ההיוריסטיקות השונות, ואת ה-variables selections על יותר מ-30 לוחות בגדלים שונים, ובצבעים שונים. קיבלנו תוצאות רבות, וכעת נציג את המשמעותיות ביותר.

זמני ריצה של אלגוריתם הCSP:



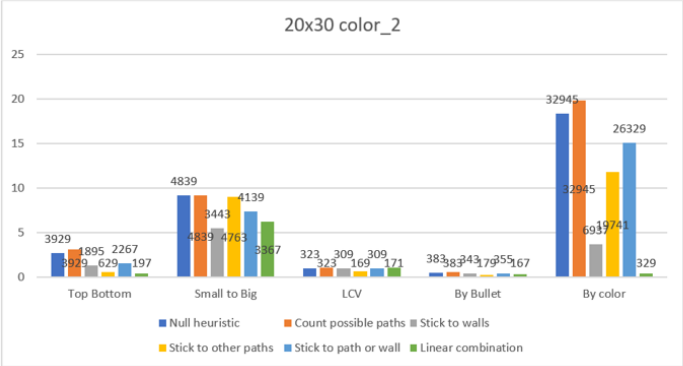


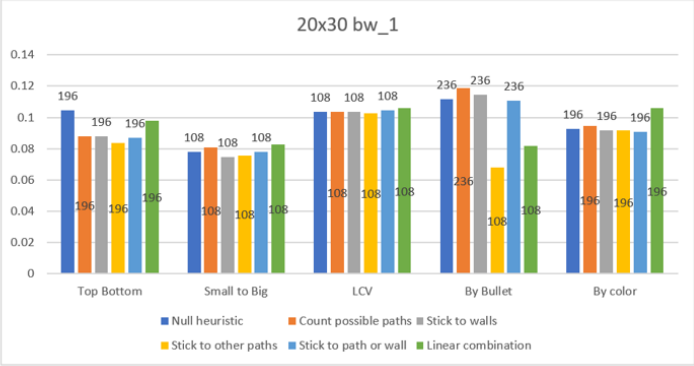




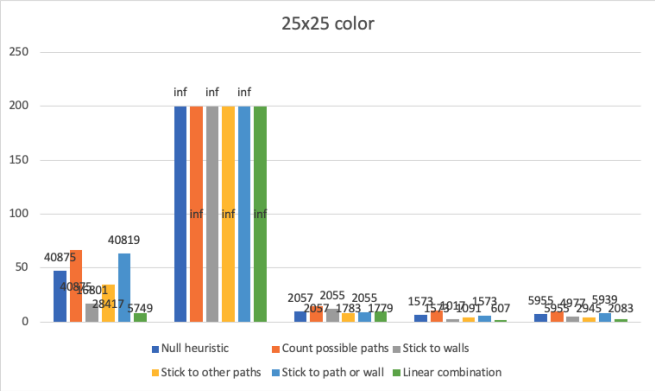
הערה: נשים לב כי בלוחות קטנים (עד 20*20 כולל) זמני הריצה הם קטנים מאוד פרט ללוח 20*20 בצבע אך גם שם זמני הריצה נמוכים יחסית.

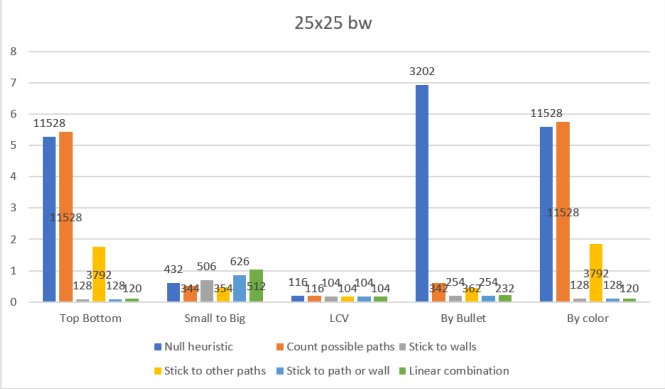
נשים לב כי עבור לוחות שאינם צבעוניים זמן הריצה נשאר זניח עד גדלים של 30*20 ובעבור לוחות צבעוניים זמן הריצה טיפה יותר גבוהה אך עדיין זניח יחסית. כעת נעבור ללוחות גדולים יותר ונוכל לראות את ביצד משפיעות ההיוריסטיקות.
בלוחות הבאים ראינו כי ה-**MRV** מניב תוצאות גרועות בהרבה מהתוצאות הממוצעות (באופן עקבי הוא לא מסיים לרוץ) ולכן החלטנו להוריד את הקטגוריה מהגרפים הבאים.

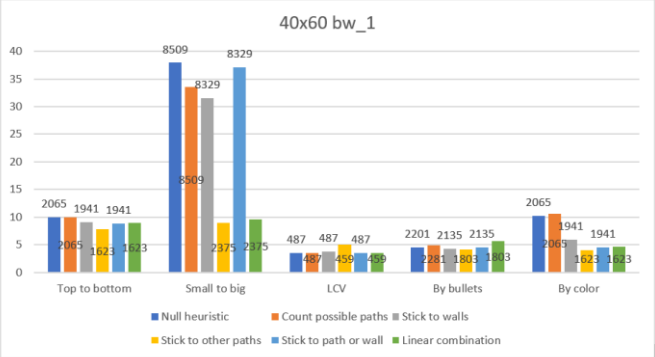




הערה: הלוח הבא (25x25 color) ברמת קושי גבוהה יחסית. במהלך כתיבת הפרויקט זמן הריצה ההתחלתי של לוח זה (עם היורסטיקות בודדות) היה מעל 7 דקות, מה שהוביל אותנו למצוא היוריסטיקות נוספות ובכך לשפר את זמן הריצה.



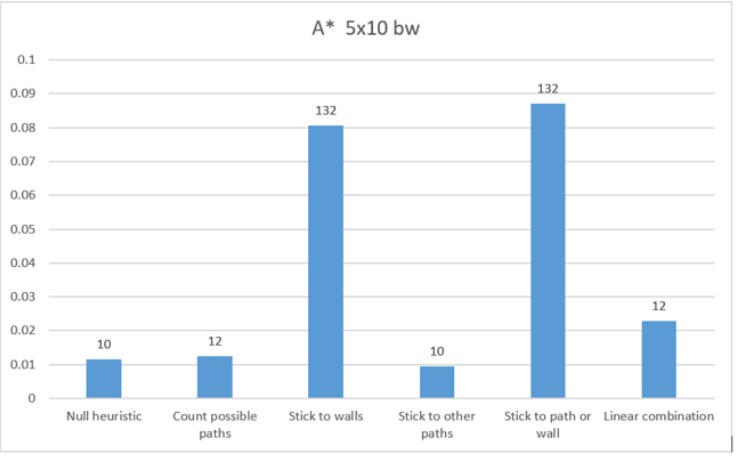


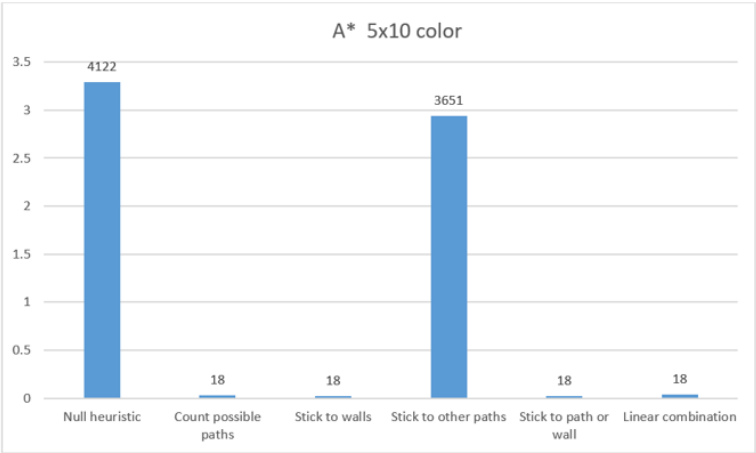


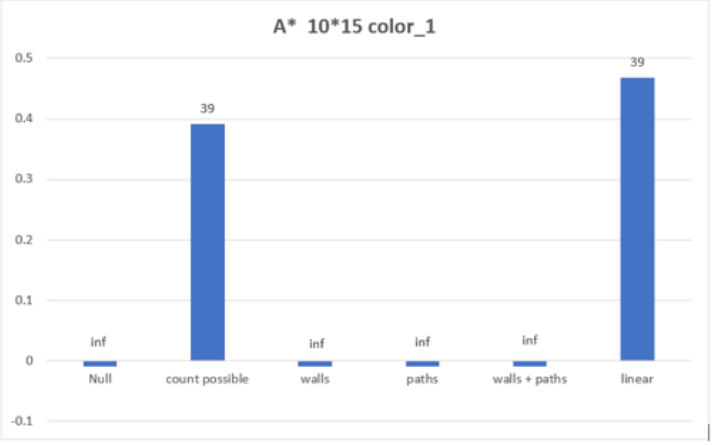
מסקנות:

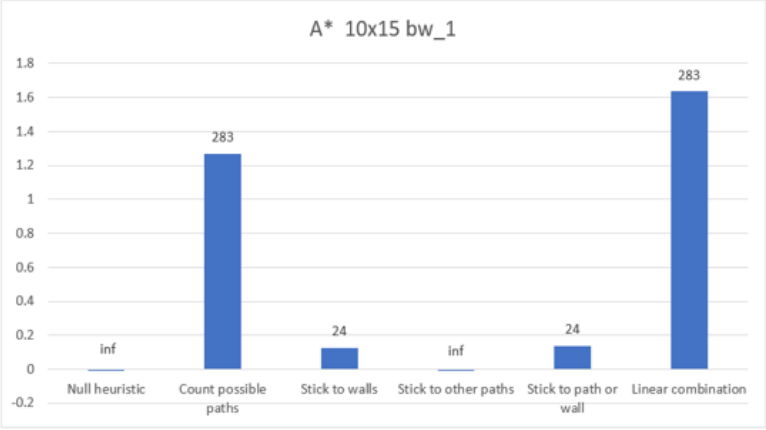
- בלוחות הקטנים (עד 20x20) זמני הריצה של כל בדיקות שעשינו היו פחות או יותר זהות, אנו מאמנים שתופעה נגרמת מכיוון שבלוחות אלו אין כמעט משבצות שלהן יותר ממסלול אחד אפשרי, ולכן כולם רצו באופן אופטימלי או קרוב לאופטימלי.
- על לוחות צבעוניים בגודל 40x60 ועל שאר הלוחות במימדים גדולים יותר רק אלגוריתמים שמשתמשים ב-**LCV** וב-**By bullet** מגיעות לפתרון בזמן ריצה סביר.
- ניתן להסיק מהגרפים הנ"ל כי באופן כללי **LCV** מניב את זמני הריצה ומספר הצעדים הנמוכים ביותר, בייחוד כאשר משתמשים בהיוריסטיקת ה-**Linear Combination** שמשלבת את כלל ההיוריסטקיות.
- נשים לב כי בלוחות שחור לבן **By Color** מתנהג בצורה זהה לזו של **Top To Bottom**.
- שיטת **By Bullet** ברוב המקרים אכן מצליחה להוריד את זמן הריצה של האלגוריתם, לרוב מתרחש במקרים בהם האלגוריתם מתקשה לבחור את המסלולים הנכונים באיזור ספציפי על הלוח. ולכן במקרים אלו הוא "יפתור" את הבעיה באיזור זה ע"פ ה-bullet בו הבעיה נמצאת ורק לאחר מכן ימשיך הלאה לשאר ה-bullets. מכיוון ש-"הבעיה העיקרית" של הלוח נפתרה המשך הפתרון ירוץ בזמן מהיר ותהיה פחות חזרה אחורה.

זמני ריצה של אלגוריתם ה-*A:





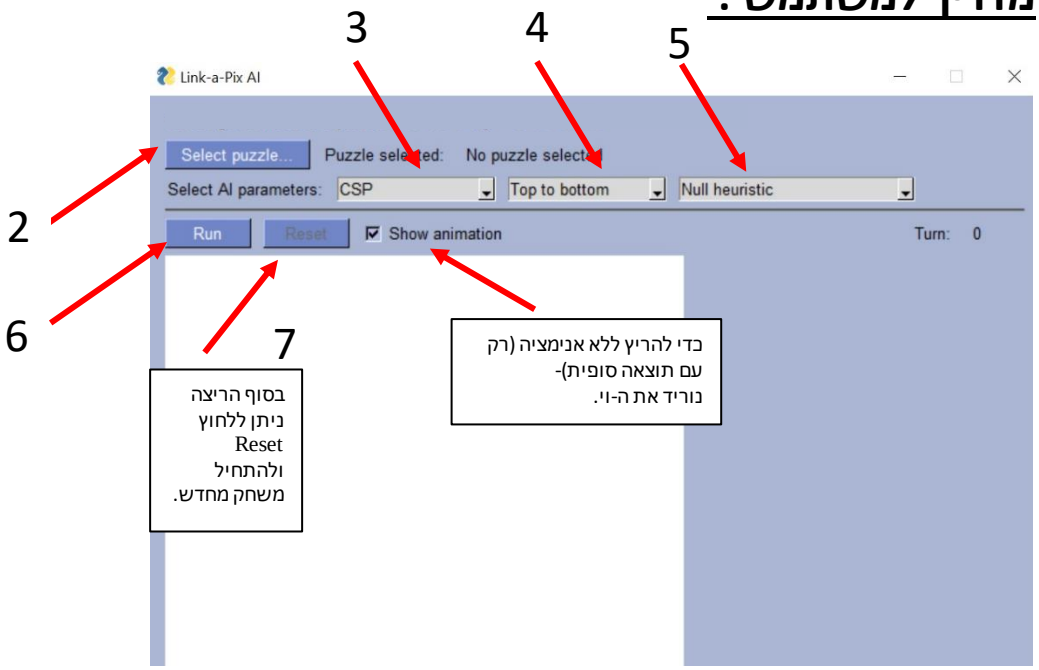




מסקנות:

- כאשר הרצנו את האלגוריתם על לוחות גדולים יותר צבעוניים ולא צבעוניים כאחד האלגוריתם לא סיים את ריצתו בזמן סביר.
- ניתן לראות כי מאחר ולכל לוח ישנו פתרון חוקי יחיד לעיתים אלגוריתם ה-**A*** לא יגיע לפתרון בזמן סביר.
- למשל בלוח בגודל 5x10 צבעוני, האלגוריתם סיים את כל הריצות ואף מהר בחלק מההיוריסטיקות. ואילו כאשר עברנו ללוח גדול יותר בגודל 10x15 הריצה הגיעה לסיום רק עם שתי היוריסטיקות מתוך 6.
- ניסינו להריץ את אלגוריתמי החיפוש: **BFS**, **DFS**, **UCS** על הלוחות הנ"ל, רק ריצה על לוח שחור לבן בגודל 5x10 הסתיימה ומעבר לכך הריצות לא הסתיימו בזמן סביר.

מדריך למשתמש :

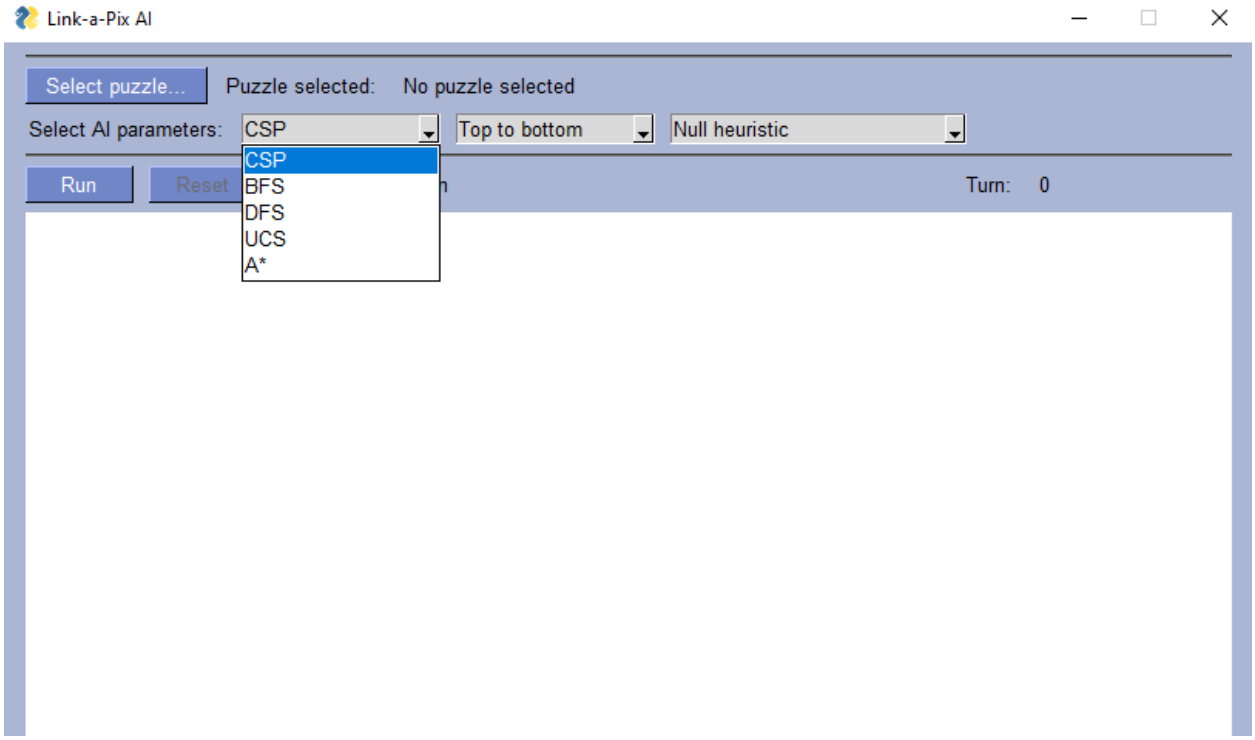


1. הריצו את התוכנית ע"י כתיבה של הפקודה:

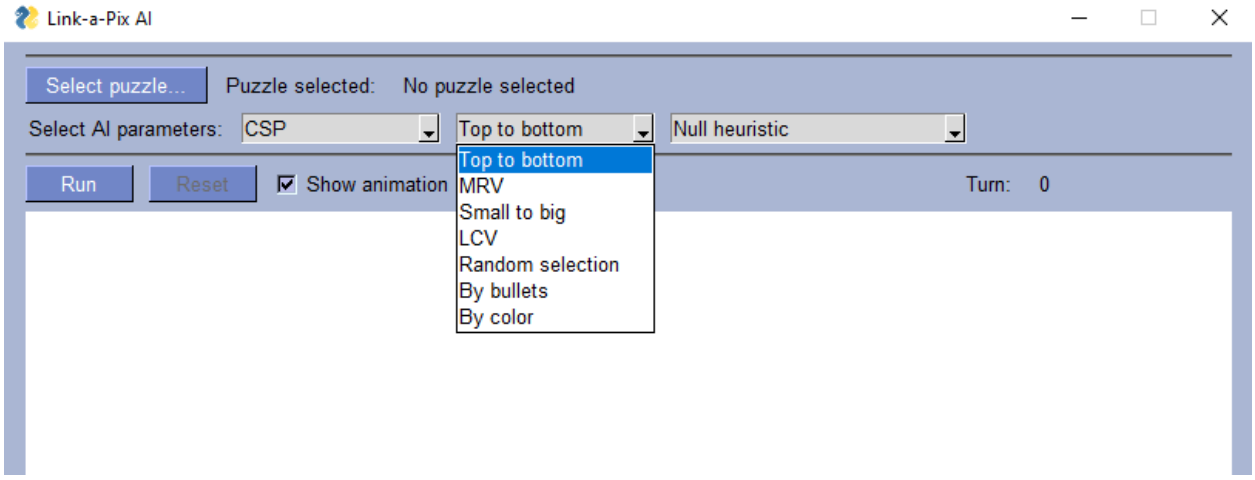
```
python3 gui.py
```

2. תחילה, נלחץ על Select puzzle... כדי לבחור את לוח המשחק אותו נרצה לפתור (קובץ xml).
יש לבחור רק פאזלים מהתיקייה boards. בחירה אחרת **תגרור לשגיאה!**

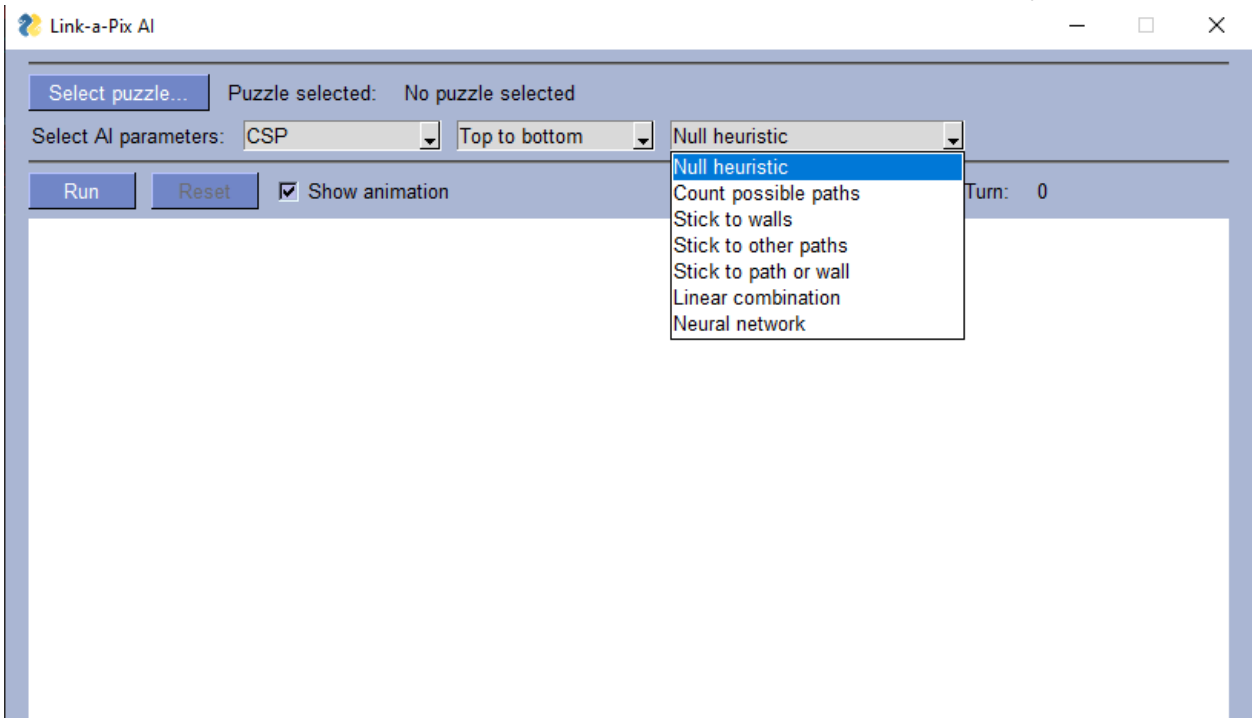
3. כעת, ניתן לבחור אלגוריתם חיפוש מהרשימה:



4. נבחר את סדר המעבר על המשבצות הממוספרות – Variable Selection (בחירה זו משפיעה רק אם אנו משתמשים ב-CSP או ב-A*):



5. נבחר את ההיוריסטיקה בה נרצה להשתמש:



6. כעת נוכל להריץ את ה-AI על הלוח:

a. **כדי להריץ עם אנימציה -**

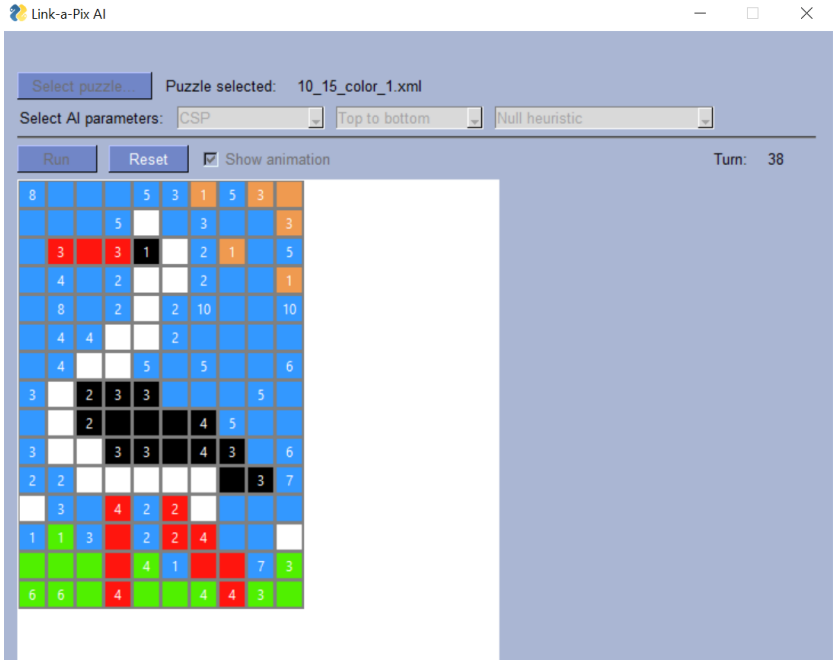
נוודא ש- Show animation **מסומן**. ולאחר מכן נלחץ Run.

b. **כדי להריץ ללא אנימציה (רק עם תוצאה סופית):**

נוודא ש- Show animation **איננו מסומן**. ולאחר מכן נלחץ Run.

הערה – בזמן שהאלגוריתם רץ לא ניתן ללחוץ על אף כפתור ב-GUI, והדרך היחידה לעצור את הריצה היא באמצעות כפתור ה-X.

7. לאחר סיום הריצה ניתן ללחוץ על כפתור ה- Reset כדי לבצע בחירה חדשה או להריץ את אותו הלוח מחדש, עם בחירות חדשות:



8. ליצאה מהתכנית בכל שלב יש ללחוץ על כפתור ה-X.

9. **תיהנו ☺**