שם הפרויקט: רמזורים חכמים

Project Name: Smart Lights

דוח מסכם

|  |  |
| --- | --- |
| שם הסטודנט: | פז לוי זלמנוביץ' |
| מספר תעודת זהות: |  |
| שם המנחה: | ויקטור טאובקין |
| חתימת המנחה: |  |
| תאריך ההגשה: | 8.6.2021 |

תוכן עניינים

[**4. תודות**](#_oasgjj40yz3b) **4**

[**5. רשימות**](#_nhc2j87bkuip) **5**

[**6. תקציר**](#_qpkd68n7tp) **8**

[**7. תקציר (אנגלית)**](#_r2phl54pykfd) **9**

[**8. מילון מונחים**](#_2iqwcjhrx95p) **10**

[**9. מבוא**](#_7wi17tw65hyc) **11**

[**10. מטרות ויעדים**](#_36cpfmxo2ii) **12**

[**11. סקירת ספרות ביקורתית מורחבת / סקר שוק**](#_48cqsa5wbsc9) **13**

[**12. ניתוח חלופות מערכתי**](#_4mf3sqewzo6x) **18**

[**13. דרישות תוכנה (Software Requirements)**](#_4regti6n59q3) **19**

[**14. אפיון המערכת**](#_lmtrc13o2qm9) **20**

[**15. ניתוח חלופות טכנולוגיות**](#_iuf5qzh0tjdm) **22**

[**16. תכן המערכת (Software Design)**](#_c8jmpmg0br52) **26**

[**17. התוצר**](#_1wioj4efcoik) **32**

[**18. תכנון ותכנית ביצוע הפרויקט (Project Planning)**](#_jtvm19by9tu0) **36**

[**19. בדיקות והערכה (Software Testing and Evaluation)**](#_uvi5b9jsg2dh) **37**

[**21. הצעה לעבודת המשך**](#_hqom00sik0xv) **42**

[**22. רשימת מקורות**](#_jdr8083p9icq) **43**

[**23. נספחים**](#_8ckprbcdfg0o) **44**

[**24.1. מסמך SRD**](#_49sdx0y2mdg0) **58**

[**24.2. מסמך SDD**](#_x99y03khe9xd) **69**

[**24.3. מסמך STD**](#_gjdgxs) **74**

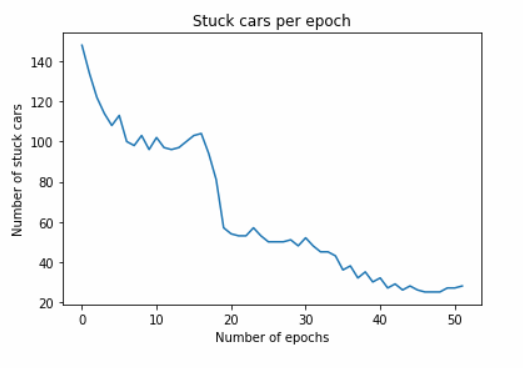
# **4. תודות**

* תודה למנחה מר ויקטור טאובקין על שליווה, הכווין, עזר והקשיב ועל תמיכה יוצאת דופן בכל הקשור לעיצוב ותכנון הפרויקט.
* תודה לגב' אפרת וויינברג על העזרה וההכוונה בנושא הבדיקות.
* תודה לדר' דינה בר-גורן שהעבירה את הקורס "ליווי פרויקט גמר" בצורה מקצועית ויעילה שבמסגרתו למדנו כתיבה טכנית וחשיבה יצירתית.
* תודה למר רועי ג'אן על שנתן עצות מכווינות ויצירתיות בתכנון האלגוריתם.
* תודה למכללת אפקה על התמיכה והעזרה האדמיניסטרטיבית.

# **5. רשימות**

הסבר הגרף:

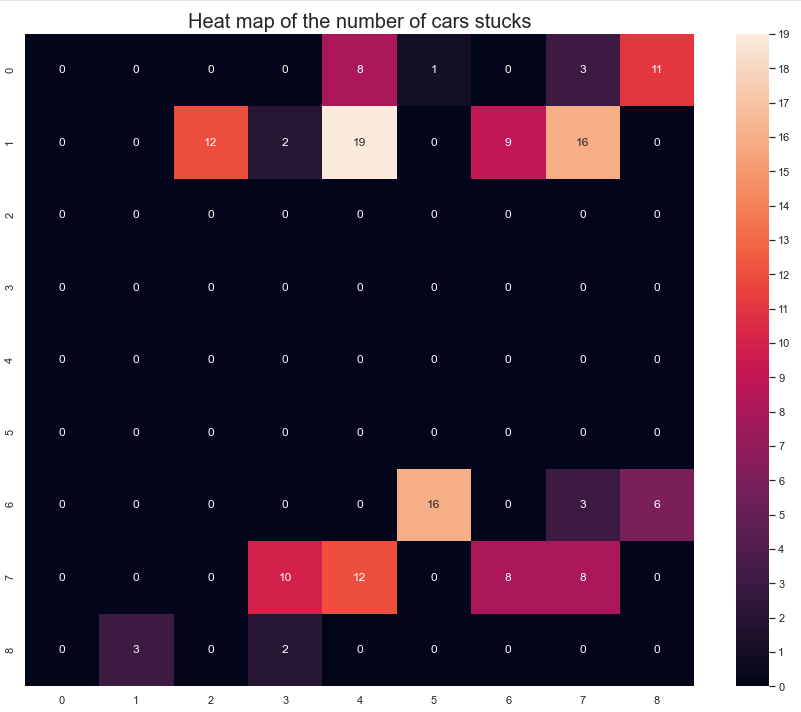
הגרף שלפנינו מציג את מספר התקיעות הכללי שהאלגוריתם מצליח להגיע אליו, ניתן לראות כי במצב ההתחלתי (נתינת 7 פריימים לכל הרמזורים בעיר) אנו מקבלים 148 תקיעות של רכבים בעיר, ואילו באיטרציה ה-50 ניתן לראות שהורדנו משמעותית את כמות התקיעות (פחות מ-40 תקיעות).

הגרף מציג ירידה מתונה במספר התקיעות ולא אירוע מקרי של ירידה נקודתית אלא מגמתית, וזה מראה את יכולת האלגוריתם להתכנסות.  


הסבר הגרף:

מפת החום מציגה את פיזור התקיעות בעיר שלנו, כך נוכל לנתח את צפיפות העומסים בעיר שלנו מכיוון שאנו פותרים זאת ע"י האלגוריתם בהתחשבות איזורית.

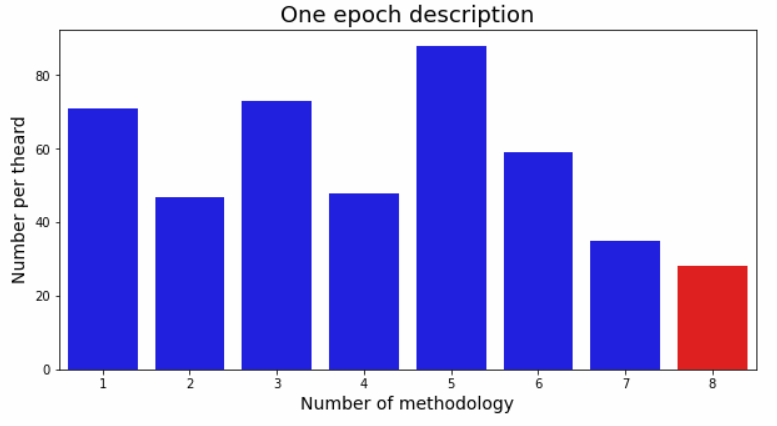
כל ריבוע במפת החום מייצג צומת (סה"כ 81 צמתים) והמספר מייצג את כמות הרכבים התקועים בצומת.



הסבר הגרף:

הגרף מציג את מספר התקיעות הכללי שמניבה כל שיטה (מתודולוגיה), ניתן לראות כי יש פערים גדולים בין השיטות כך שאנו בוחרים בכל הרצה את המצב הטוב ביותר (הצבוע באדום).

הגרף מתייחס לפירוט של איטרציה בודדת.



# **6. תקציר**

הפרויקט שלנו עוסק בהצגת עיר על ידי מספר צמתים המחוברים לזה לזה, שבה מוזרמים מספר נבחר של רכבים כאשר לכל רכב היעד שלו והמסלול שלו אשר נתנים לו על ידי אלגוריתם למציאת מסלול.

מטרת הפרוייקט הינה להוריד עומס בכבישים ולחסוך פקקים על ידי כך שרשת הרמזורים המוצגת בפרוייקט מתחשבת בעומסים. הפרוייקט כולל תכנון ובנייה של רשת רמזורים שמווסתת את התנועה על ידי שימוש באלגוריתם שבנינו בעצמנו בהכוונת עולם הבינה המלאכותית לפתרון הבעיה, ובניית סימולציה שמדמה את פעילות הרשת והצגת השיפור.

בפרוייקט שלנו, שמכוון להורדת עומס בכבישים וחסכון בזמן, אנו מרוויחים גם בתחומים נוספים:

* הנהגים מגיעים ליעדם מהר יותר ומתוסכלים פחות (רווח חברתי)
* הנהגים חוסכים בדלק ובבלאי של הרכב (רווח כלכלי)
* חסכון בזמן על הכביש מוביל לצמצום בזיהום האוויר (רווח סביבתי)

# **7. תקציר (אנגלית)**

Our project is a representation of a city by a large number of interconnected junctions, in which a selected number of vehicles are streamed. Each vehicle has its own target and its route which is given to it by a path-finding algorithm.

The aim of the project is to reduce congestion on the roads and traffic jams by taking traffic into account. The project includes the design and construction of a traffic light network that regulates traffic by using an algorithm we built ourselves with the guidance of artificial intelligence to solve the problem, and a simulation that simulates the activity of the network and shows the improvement.

In our project, which aims to reduce congestion on the roads and save time, we also benefit in other areas:

* Drivers get to their destination faster and less frustrated (social gain)
* Drivers save fuel and vehicle wear and tear (economic gain)
* Saving time on the road leads to a reduction in air pollution (environmental gain)

# **8. מילון מונחים**

|  |  |
| --- | --- |
| מונח | משמעות המונח |
| סימולציה | תכנית ב-Open GL אשר מראה את יכולות המערכת וזרימת התנועה, מציגה את העיר והרכבים הנעים בה. |
| אלגוריתם לשיפור המערכת | אלגוריתם לשינוי הזמנים ושיפור העומסים שמבוסס על שיטות מיצוע על בסיס מונטה קרלו. נכתב על ידינו בשפת Python. |
| מספר התקיעות | הפלט של הסימולציה. מספר המייצג את כמות הפעמים שרכבים נתקעו במהלך הסימולציה. |
| רכב תקוע | רכב שעומד ברמזור ירוק ולא יכול לנוע לצומת הבאה במסלול משום שהצומת אליה הרכב צריך להתווסף מלאה. |
| מספר פריימים | הקלט של הסימולציה. מספר המייצג את הזמן (בפריימים) שניתן אור ירוק לכל רמזור. |
| עיר | מספר נבחר של צמתים. |
| צומת | נקודת מפגש של ארבעה כבישים. |
| כביש | דרך בעלת שני נתיבים בשני כיוונים מנוגדים. |
| מכונית | מיוצגת על ידי נקודה שחורה. לכל מכונית יש צומת התחלה, צומת יעד ומסלול. |
| רכב תקוע | רכב אשר לא יכול לעבור לצומת הבאה במסלול שלו. נגרם עקב כך שהכביש שאליו הרכב צריך לעבור מלא, נצבע בצבע אדום בסימולציה. |
| מתודולוגיה | שיטה עיבוד תמונה של זמני הרמזורים. |
| שיפור | הקטנת מספר הרכבים שנתקעו בעיר. |
| סיום ריצת סימולציה | כאשר כל הרכבים הגיעו ליעדם, או מצב נעילה. |
| איטרציה (Epoch) | בדיקת כל המתודולוגיות ובחירה של השיפור הטוב ביותר. |
| מקביליות | חישוב זמנים חדשים על ידי מספר שיטות בצורה מקבילית והרצת הסימולציה שלהם. |
| נעילה (deadlock) | מצב שבו כל הרכבים בעיר לא יכולים לזוז (נובע מכך שרכבים אחרים תקועים) |

# **9. מבוא**

כיום, עומסי התנועה מהווים בעיה יומיומית עבור נהגים פרטיים וכלי תחבורה ציבורית המשתמשים בכבישים הציבוריים. בעשור האחרון, מספר הרכבים על הכביש גדל ב-69 אחוזים, לעומת שטח הכבישים שגדל ב-40 בלבד באותו פרק זמן. חלק גדול מהפקקים נגרם בשל שימוש בנאלי ולא מושכל ברמזורים, מה שגורם לחלוקה לא נכונה ולא מתחשבת של עומסים.

מערכת Smart Lights: רשת של רמזורים אשר עושה שימוש באלגוריתם שבנינו ותוך התחשבות בפרמטרים נוספים מנווטת את התנועה ומאזנת עומסים על מנת להקל על בעיית הפקקים.

מטרות: [מטרות צד]

* הפחתת העומס
* [רווח כלכלי: פחות זמן על הכביש = בלאי נמוך יותר לרכב וחסכון בדלק]
* [רווח חברתי: אנשים יגיעו למחוז חפצם מהר יותר ומתוסכלים פחות]
* [רווח סביבתי: צמצום בזיהום אוויר]

לקוחות אפשריים:

* נהגים פרטיים
* חברות תחבורה ציבורית
* משרד התחבורה\עיריות

אתגרים הנדסיים:

* פיתוח **רשת** של רמזורים שמתקשרים בניהם.
* פיתוח אלגוריתם לשיפור הרשת.
* תכנון ובנייה של סימולטור להמחשת הרשת.
* הערכה ושיפור של זמנים, עומסים וכו'.

התוצר הסופי:

* קובץ שמציג זמנים משופרים שנוצר לאחר הרצת האלגוריתם על זמנים התחלתיים. הרצת הקובץ על הסימולציה, או לחילופין השמת הזמנים על מערכת באופן מעשי, תוביל לירידה בכמות התקיעות.

חלקו של סטודנט 1 (פז לוי): אלגוריתם לתפעול המערכת.

חלקו של סטודנט 2 (אביחי אליהו): סימולציה להצגת יכולות המערכת.

בסופו של דבר, המערכת תציג על ידי הסימולציה את ההישגים של האלגוריתם.

# **10. מטרות ויעדים**

המטרה העיקרית של הפרויקט הינה הפחתת עומסים בכבישים על ידי שינוי זמני הרמזורים באופן חכם. על ידי כך, נוכל לקצר את זמן הנסיעה של הנהגים.

יעד פונקציונלי: עבודת המערכת בצורה יעילה שתאפשר את קיצור זמנים ומספר מינימאלי של תקיעות.

מדד: נרצה להגיע לשיפור של לפחות 10 אחוזים במספר התקיעות במערכת בזמן ההרצה. ניתן להסיק אם עמדנו ביעד או לא על ידי השוואה: האם בעזרת המערכת שלנו מספר התקיעות ירד לעומת מספר התקיעות ההתחלתי?

יעדים פונקציונליים צדדיים: הפחתה בזיהום האוויר, צריכת דלק ובבלאי של הרכב.

מדד: מתוך כמות התקיעות שירדה נוכל להסיק על הפחתה בזיהום האוויר ובצריכת הדלק, והבלאי שנחסך.

# **11. סקירת ספרות ביקורתית מורחבת / סקר שוק**

פתרון אפשרי לבעיה הוא על ידי שימוש ב-Image Filtering כמוצג במאמר (מספר 6 ברשימת המקורות) - זו שיטה להבהרה, טשטוש, הבלטה, זיהוי קצוות בתמונה ועוד. בעיבוד תמונה הפילטר הוא הגרעין, מטריצת קונבולציה או מסכה (מטריצה ​​קטנה). נעשה על ידי שילוב בין הגרעין (Kernel) לבין התמונה. במאמר מוצגות מספר שיטות לסינון תמונה (image filtering).

למה המאמר רלוונטי לפרויקט שלנו?

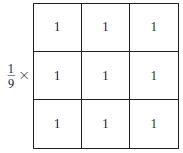
הפעולה של סינון תמונה רלוונטית לפרויקט שלנו מכיוון שראינו בה פתרון יצירתי לחישוב זמנים בצורה אזורית (מתחשבת בצמתים שכנים). מתוך המאמר למדנו ובחרנו מספר שיטות שיניבו תוצאות טובות לפתרון הבעיה שלנו. בפרויקט שלנו אנו מסתכלים על מספר תקיעות גדול כעל "רעש" בתמונה, שאותו אנו מנסים "לנקות". האלגוריתם שלנו משתמש במספר פילטרים נבחרים על מנת להגיע ל-"תמונה נקיה מרעשים", אשר במקרה שלנו מייצגת מערכת עם פחות תקיעות.

בסקירת הספרות נתעמק בפילטרים השונים בהם השתמשנו באלגוריתם.

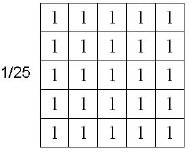
Mean Filter Module:

שיטה פשוטה, אינטואיטיבית וקלה ליישום להחלקת תמונות, כלומר הקטנת גיוון העוצמות בין פיקסל אחד למשנהו. היא משמשת לעתים קרובות להפחתת רעש בתמונות. הרעיון של סינון ממוצע הוא להחליף כל ערך פיקסל בתמונה לערך הממוצע (mean) של שכניו, כולל עצמו. זה משפיע על ביטול ערכי פיקסל שאינם מייצגים את סביבתם.

גרעין (kernel) בגודל 3X3:

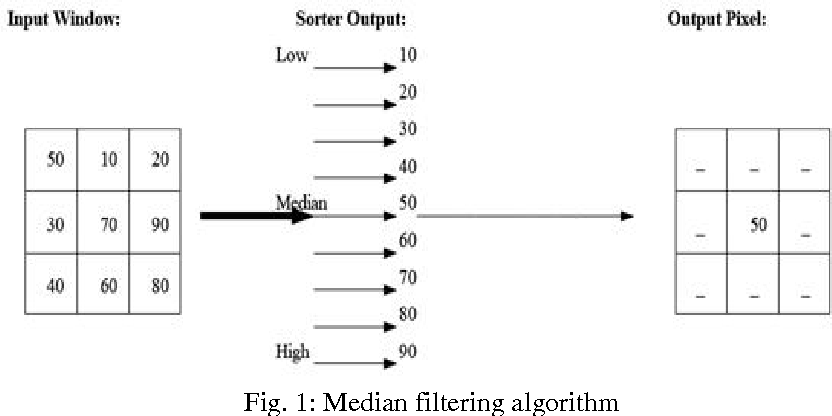


גרעין (kernel) בגודל 5X5:

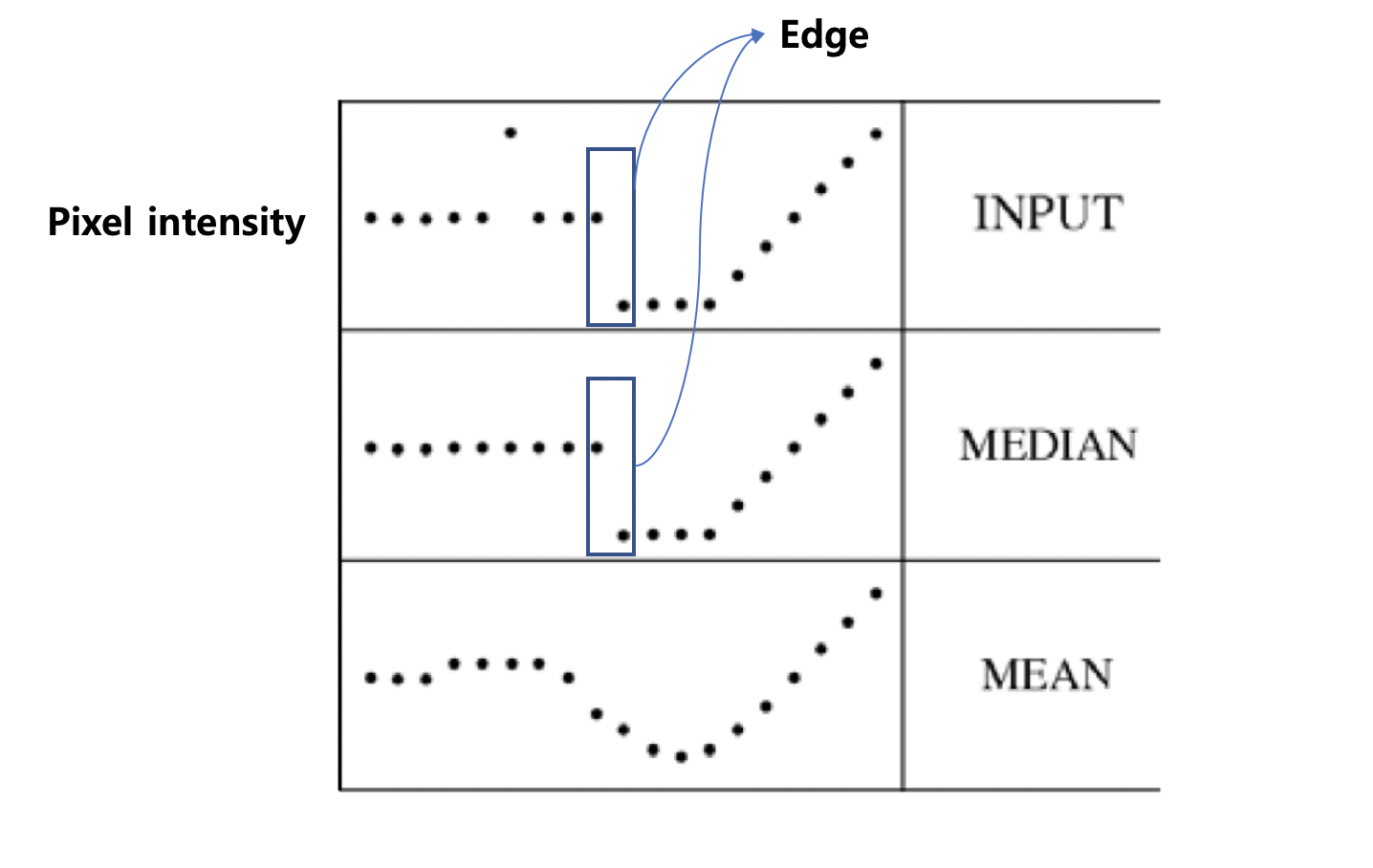


Median Filter Module:

המסנן החציוני משמש להסרת רעש מתמונה על ידי החלפת פיקסלים בערך הפיקסל האמצעי שנבחר מגודל חלון מסוים. המסנן החציוני יעיל מאוד להסרת רעש. הרעיון הכללי שעומד מאחורי הפילטר הוא למצע פיקסל באמצעות ערכי פיקסלים שכנים, אך בו זמנית לדאוג למבני תמונה חשובים. הדאגה העיקרית של המסנן היא להבחין בין וריאציות מקומיות בגלל רעש ובשל מבנה התמונה. על מנת להשיג זאת, עבור כל פיקסל אנו מפיקים ערך המבטא את מידת הנגזרת בכיוון מסוים קטן (gradient\שיפוע). ניתן לראות בתמונה כי הערכים שבמטריצת הקלט עוברים מיון מקטן לגדול ונלקח הערך החציוני.



השוואה בין Mean Filter לבין Median Filter:

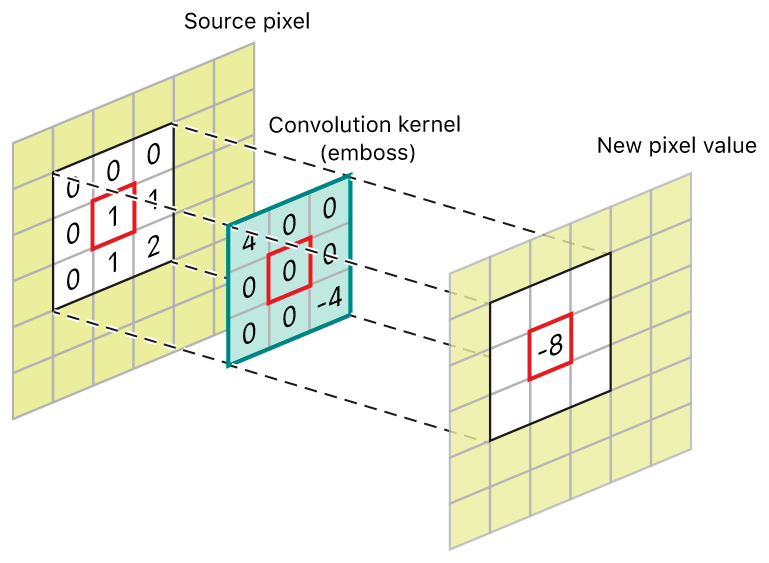


Gaussian Blur:

השיטה עושה שימוש בפונקציית גאוסיאן בעלת שני מימדים (x,y):

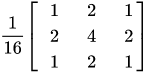


הערכים מפונקציה זו ייצרו את מטריצת הגרעין (kernel) שנחיל על כל פיקסל בתמונה המקורית. הגרעין בדרך כלל די קטן - ככל שהוא גדול יותר כך עלינו לעשות יותר חישובים בכל פיקסל.

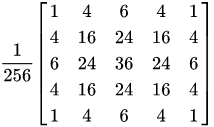


בתמונה זו ניתן לראות סכימה של מכפלת כל הערכים במקומות מתאימים של פיקסל כלשהו (source pixel) במטריצת הגרעין (kernel), והשמת התוצאה במרכז הפיקסל החדש (new pixel value).

גרעין (kernel בגודל 3X3):

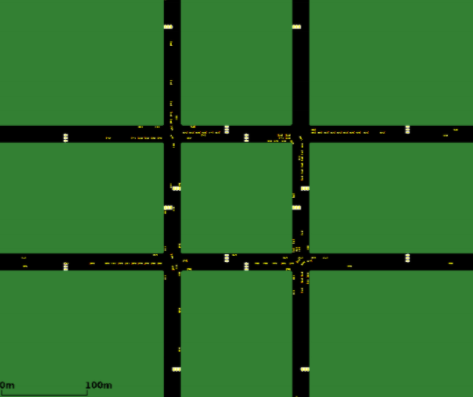


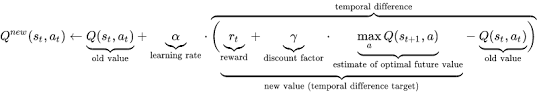
גרעין (kernel בגודל 5X5):



סקירת ספרות בנושא Q-Learning:

המאמר: "Reinforcement Learning for Traffic Optimization" (מאמר מספר 3 ברשימת המקורות)

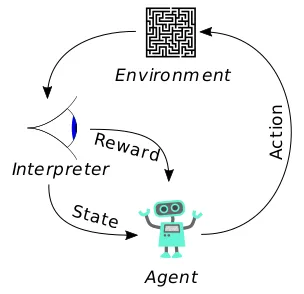
במאמר זה מיישמים טכניקות שונות של למידה על מערכת הרמזורים במטרה להגדיל את זרימת התנועה בצמתים. משווים בין סוגי מדיניות שונות כולל מחזורים קבועים, תחילה טכניקת "התור הארוך ביותר" (LQF) וב-reinforcement learning בטכניקת Q-learning.   




הנוסחה שלפנינו מציגה את עדכון טבלת המצבים (Q-Table) המצב החדש על ידי החישוב שלפנינו.

Q-learning הוא אלגוריתם RL. המטרה של Q-learning היא ללמוד מדיניות, שתגדיר לסוכן איזו פעולה לנקוט בהתאם לנסיבות.

האלגוריתם אינו מצריך מודל של הסביבה, והוא יכול לטפל בבעיות עם מעברים אקראיים ותגמול (או ניקוד לפעולה), ללא צורך בהתאמות.



עבור כל תהליך המודל יבחר את הצעד עם התגמול הטוב ביותר, Q-learning מוצא מדיניות אופטימלית במובן של מקסום הערך הצפוי של התגמול הכולל על כל השלבים הבאים,

החל מהמצב הנוכחי.

Q-learning יכול לזהות מדיניות בחירת פעולה אופטימלית עבור תהליך החלטה, בהינתן זמן חיפוש אינסופי ומדיניות אקראית חלקית.

"Q" מציין את הפונקציה המחזירה את התגמול, כשהתגמול משמש לחיזוק ומציין את האיכות (quality) של ביצוע הפעולה במצב הנתון.

**לפי המחקר שערכנו כחלק מסקירת הספרות, למידת מכונה בשיטת Q-Learning מתאימה למקרה של צומת אחת או מספרים צמתים קטן בעלי תלות נמוכה בניהם. בעת נסיון להשתמש באלגוריתם Q-Learning בפרויקט שלנו, נוכחנו לדעת שאנו לא מתכנסים לפתרון הבעיה בשל התלויות הרבות שיש בעיר שיצרנו.**

סקר שוק וניתוח מתחרים:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| קריטריונים | Smart Lights | רמזורים רגילים | Greenlights | Intelligent Traffic Control |
| קיים בפועל | X | V | X | X |
| מבוסס למידת מכונה | V | X | V | V |
| רשת רמזורים | V | X | X | V |
| סימולטור | V | X | X | X |
| פתירת הבעיה עבור מספר צמתים עם תלות בניהן | V | X | X | X |

# **12. ניתוח חלופות מערכתי**

|  |  |  |  |
| --- | --- | --- | --- |
| שם החלופה | יתרונות | חסרונות | ציון |
| שינוי הרמזורים באופן ידני | * אין צורך בכוח חישוב | * שינוי רמזורים בעזרת אדם שאחראי על הפעלת רמזורים * שיטה פרמיטיבית -   מימוש לא סביר, שכן מדובר בהרבה כוח אדם, תשתיות   * פתרון נקודתי שלא פותר את בעיה העומסים | 0/5 |
| שינוי רמזור ללא התחשבות ברשת | * אין צורך בהתחשבות ברמזורים אחרים | * לא פותר את הבעיה עבור עיר * לא מתחשב בתנועה * לא מתחשב בזרימה * עלול ליצור עומסים נוספים | 1/5 |
| מתן זמן קצוב לכל הרמזורים | * לא דורש התערבות נוספת * מימוש פשוט | * מימוש לא מתחשב * מימוש לא גמיש * לא פותר את הבעיה | 2/5 |
| שימוש באלגוריתם Q-Learning | * אלגוריתם חכם * יעיל עבור צומת | * יקר מאוד חישובית * מורכב למימוש * לא מתאים עבור מספר צמתים עם תלות: * לא מבטיח התכנסות (תשובה סופית) * בנוי לפתירת הבעיה עבור צומת אחת * טיפול נקודתי | 2.5/5 |

**שינויים לעומת דוח הביניים: נוספה חלופה של שימוש באלגוריתם Q-Learning.**

# **13. דרישות תוכנה (Software Requirements)**

**דרישות פונקציונאליות:**

* התממשקות עם אלגוריתם מבוסס למידת מכונה.
* רכיב במערכת שאחראי על עיבוד תוצאות והסקת מסקנות.
* בניית סימולציה להצגת יכולות המערכת.
* מצב התחלתי שלא מוביל לנעילה.
* קישור בין הסימולציה לאלגוריתם.
* מערכת שעובדת באופן מקבילי.

**דרישות לא פונקציונאליות:**

* ממשק לקליטת נתוני קצה.
* ממשק לשמירת הנתונים לקובץ טקסט.
* מחשב להרצת הסימולציה, מחשב שתומך ב-Visual Studio 15 ומעלה.
* מחשב להרצת האלגוריתם בתוכנת Python (עם יכולות מתאימות לעבודה מקבילית).
* הפקת דוחות - לפני ואחרי הפעולה.
* שליפה מהירה מבסיס הנתונים.
* מערכת אמינה ונוחה לתפעול.
* תוכנת jupyter lab (בגרסת 3.0.12) להפעלת קובץ הפייתון.
* דפדפן לצורך הצגת הקוד והגרפים (ללא יכולת הרצה) בקובץ HTML (רשות).

**דרישות שנוספו:**

* הוספת מקביליות למערכת, נוספו עקב הוספת תמיכה במקביליות באלגוריתם.
* דרישה להרצת המערכת דרך תוכנת Jupyter-Lab, הדרישה נוספה לשם נוחות בהצגת גרפים.
* דפדפן לצורך הצגת הקוד בקובץ HTML, נוסף על מנת להקל על קורא הקוד.

# **14. אפיון המערכת**

**מבנה מערכת כללי:**

סימולציה - להצגת פעילות המערכת והשיפורים שנעשים.

אלגוריתם - אלגוריתם מבוסס למידת מכונה אשר אחראי לשינוי הזמנים לצורך הורדת מספר תקיעות הרכבים.

קישור בין שתי המערכות - שני קבצי טקסט שהסימולציה כותבת אליהם את הזמנים ומספר תקיעות הרכבים הכללי והמפורט, והאלגוריתם אחראי על שיפורו של מספר הרכבים שנתקעו. האלגוריתם כותב את הזמנים החדשים לקובץ והסימולציה רצה לפיהם.

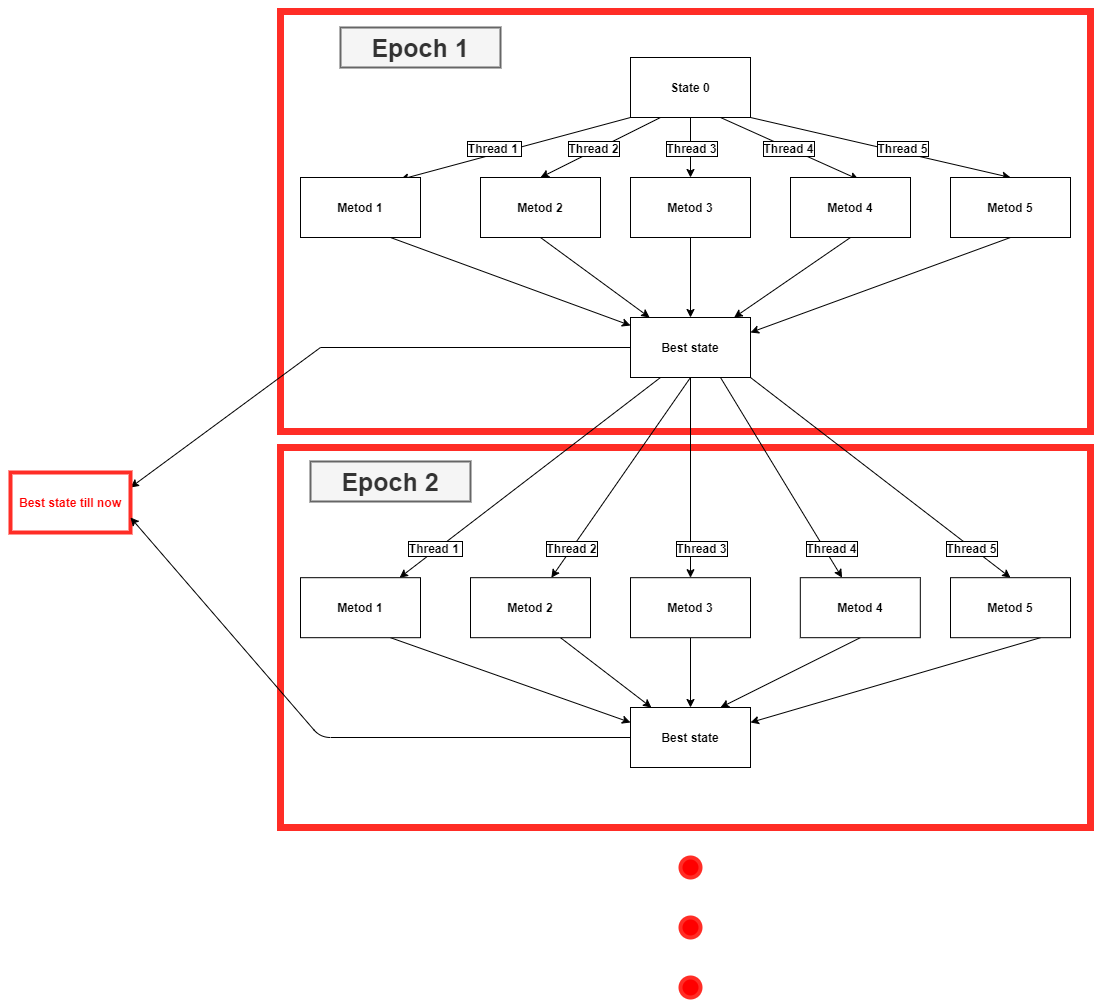
**מודולי וארכיטקטורת המערכת:**

המערכת שלנו בנויה כך שהסימולציה והאלגוריתם מתקשרים בניהם על ידי קבצי טקסט.

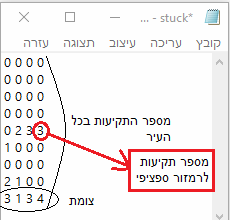
סימולציה - להצגת הנתונים, ספירת מספר תקיעות הרכבים ולהוכחת נכונות תוצאות האלגוריתם.

אלגוריתם - לשינוי הזמנים למטרת שיפור מספר תקיעות הרכבים.

בתמונה: תרשים שמתאר איטרציה אחת (מתוך ה-SRD)



בתמונה: דוגמה לקובץ תקיעות (מתוך ה-SRD)



**ביצועים עיקריים:**

על מנת להביא למקסימום את הביצועים:

* כפי שמתואר במסמך ה-SRD, האלגוריתם פועל לחישוב מספר של שיטות , ואנו בוחרים את השיטה הטובה ביותר.
* האלגוריתם פועל באופן מקבילי (כפי שמתואר במסמך ה-SRD): מספר תהליכונים כמספר השיטות שאנחנו משתמשים בהן. כפי שמתואר בפרק העוסק בבדיקות, השיטה המקבילית מהירה פי 7.2 לעומת הרצה סדרתית.
* הסימולציה נבנתה ב-OpenGL כך שהיא קלה להרצה.
* המודולים מתקשרים ביניהם על ידי קבצים מקומיים, דבר שמאפשר גישה מהירה.

**אילוצים:**

* נדרוש שהקובץ ההתחלתי לא יוביל למצב של נעילה (deadlock)- במצב זה איננו מקבלים מצב תקיעות לשיפור מכיוון שבעת מצב נעילה מספר התקיעות הוא 1-.
* נדרוש שהקבצים יהיו במבנה הנכון לקריאה - על מנת שהאלגוריתם יוכל לקרוא את הנתונים ולהכניסם למשתנים הנכונים.
* נדרוש שהזמנים בקובץ יהיו מספרים שלמים וחיוביים.

# 

# **15. ניתוח חלופות טכנולוגיות**

סימולציה:

|  |  |  |  |
| --- | --- | --- | --- |
| קריטריון | Open GL | Unity | Java FX |
| התאמה לפתרון הבעיה | V | X | X |
| קל להרצה | V | X | V |
| נותן כלים לפתרון הבעיה | V | V | X |
| תכנות מונחה עצמים | X | V | V |

פירוט עם משקלים וציונים:

|  |  |  |
| --- | --- | --- |
| חלופה | יתרונות | חסרונות |
| הצגה מספרית (במקום סימולציה) | * קל לשימוש | * לא ויזואלי * קשה למצוא שגיאות * קשה לראות בעיות בתנועה |
| Unity | * סימולציה יותר "יפה" * יותר קרוב למציאות | * קשה למימוש * כבד לעיבוד * אין צורך לפרוייקט מסוג זה |
| Java FX | * מבוסס על תכנות מונחה עצמים | * מימוש קשה * יש צורך במימוש גראפיקה בסיסית (צורות בסיסיות וכו') |
| OpenGL | * מתאים לפתרון הבעיה * ויזואליזציה טובה * נותן כלים למימוש פשוט | * לא מבוסס תכנות מונחה עצמים * מימוש ברמת קושי בינונית |

טבלת קריטריונים הצגה מספרית:

|  |  |  |
| --- | --- | --- |
| קריטריון | משקל | ציון |
| קלות מימוש | 10 | 10 |
| קל להבנה | 10 | 10 |
| ויזואלי | 30 | 0 |
| מתאים לפתרון הבעיה | 50 | 3 |

טבלת קריטריונים Unity:

|  |  |  |
| --- | --- | --- |
| קריטריון | משקל | ציון |
| קלות מימוש | 10 | 2 |
| קל להבנה | 10 | 7 |
| ויזואלי | 30 | 10 |
| מתאים לפתרון הבעיה | 50 | 4 |

טבלת קריטריונים Java FX:

|  |  |  |
| --- | --- | --- |
| קריטריון | משקל | ציון |
| קלות מימוש | 10 | 1 |
| קל להבנה | 10 | 1 |
| ויזואלי | 30 | 8 |
| מתאים לפתרון הבעיה | 50 | 3 |

טבלת קריטריונים OpenGL:

|  |  |  |
| --- | --- | --- |
| קריטריון | משקל | ציון |
| קלות מימוש | 10 | 8 |
| קל להבנה | 10 | 8 |
| ויזואלי | 30 | 10 |
| מתאים לפתרון הבעיה | 50 | 10 |

**ניתן לראות ש-OpenGL עונה על הקריטריונים הרלוונטיים לנו ותומכת בפתרון הבעיה.**

אלגוריתם:

|  |  |  |  |
| --- | --- | --- | --- |
| קריטריון | האלגוריתם שלנו | Q-Learning | MCTS  (Monte Carlo Tree Search) |
| התאמה לפתרון הבעיה (עבור עיר) | V | X | V |
| התכנסות לפתרון מספק | V | X | V |
| זמן למידה מהיר | V | X | X |
| צריכת זכרון קטנה | V | X | X |
| קל חישובית | V | X | X |
| תומך במקביליות | V | X | V |
| תמיכה בשיטות שונות | V | X | V |

פירוט עם משקלים וציונים:

|  |  |  |
| --- | --- | --- |
| חלופה | יתרונות | חסרונות |
| Q-Learning | * מתאים לפתרון בעיה של צומת אחת | * לא מתאים לפתרון הבעיה של עיר * לא מבטיח התכנסות |
| MCTS  (Monte Carlo Tree Search) | * מתאים לפתרון בעיה של עיר * מבטיח התכנסות לפתרון | * קשה למימוש * כבד לעיבוד * צורך זכרון |
| האלגוריתם שלנו | * מתאים לפתרון בעיה של עיר עם תלות בין צמתים * מתכנס לפתרון * מהיר חישובית * תומך במקביליות | * מבטיח פתרון אך לאו דווקא הטוב ביותר. |

טבלת קריטריונים Q-Learning:

|  |  |  |
| --- | --- | --- |
| קריטריון | משקל | ציון |
| קלות מימוש | 15 | 2 |
| מתכנס לפתרון | 70 | 1 |
| יקר (חישובית וזכרון) | 15 | 1 |

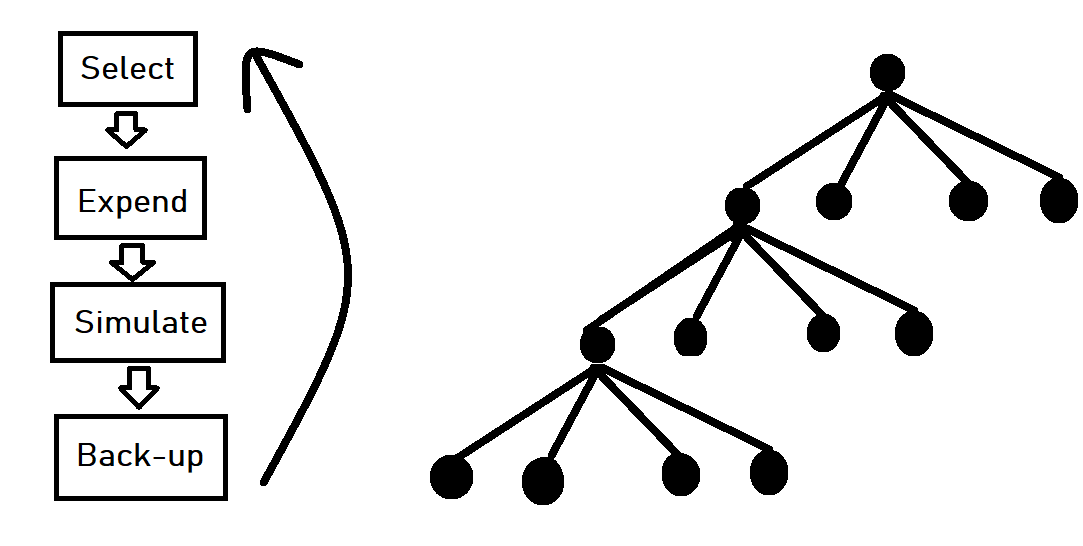
טבלת קריטריונים MCTS (מצורף הסבר למטה):

|  |  |  |
| --- | --- | --- |
| קריטריון | משקל | ציון |
| קלות מימוש | 15 | 1 |
| מתכנס לפתרון | 70 | 10 |
| יקר (חישובית וזכרון) | 15 | 1 |

טבלת קריטריונים האלגוריתם שלנו:

|  |  |  |
| --- | --- | --- |
| קריטריון | משקל | ציון |
| קלות מימוש | 15 | 10 |
| מתכנס לפתרון | 70 | 7 |
| יקר (חישובית וזכרון) | 15 | 10 |

ניתן לראות שאלגוריתם Q-Learning לא מתאים לפתרון הבעיה שלנו מכיוון שהוא אינו מתאים לרשת של צמתים עם תלות בניהם. אלגוריתם MCTS מתאים לפתרון הבעיה אך יקר מאוד מבחינה חישובית ומבחינת זכרון. **האלגוריתם שלנו נבנה במיוחד לפתרון הבעיה ולכן עונה על הדרישות בצורה הטובה ביותר.**  
הסבר קצר על MCTS: (מספר 4 ו-5 ברשימת המקורות)



במדעי המחשב, חיפוש העץ של מונטה קרלו (MCTS) הוא אלגוריתם חיפוש היוריסטי לכמה תהליכי החלטה, בעיקר אלו המועסקים בתוכנה שמשחקת משחקי לוח. בהקשר זה MCTS משמש לפתרון עץ המשחק.

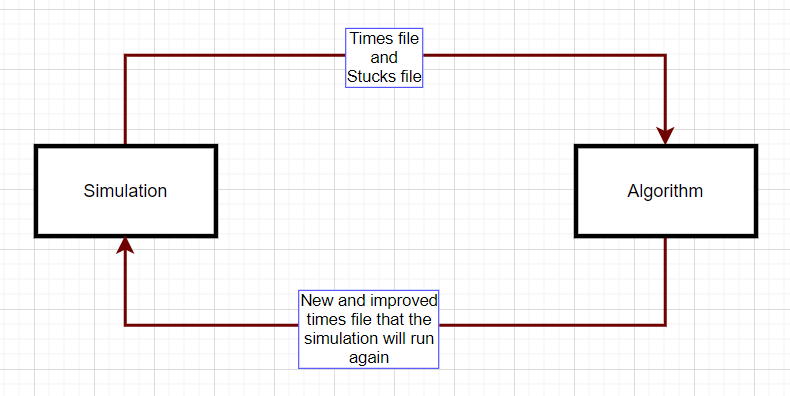
המוקד של MCTS הוא בניתוח המהלכים המבטיחים ביותר, הרחבת עץ החיפוש בהתבסס על דגימה אקראית של מרחב החיפוש. היישום של חיפוש עצים במונטה קרלו במשחקים מבוסס על הפעלות רבות. בכל הפעלה, המשחק (האלגוריתם) מתבצע עד הסוף על ידי בחירת מהלכים באופן אקראי. לאחר מכן, תוצאת המשחק הסופית של כל הפעלה משמשת לשקלול הצמתים בעץ המשחק, כך שיש יותר סיכוי שייבחרו בצמתים טובים יותר בהפעלות בעתיד.

# **16. תכן המערכת (Software Design)**

ארכיטקטורת המערכת:

החלטנו לבנות סימולציה שתציג את הבעיה ותשמש לנו כמדד איכותי לשיפורה, הסימולציה מציגה הדמיה של מצב בשטח אשר אותו אנו מנסים לפתור.

הארכיטקטורה בנויה בצורה הבאה:

  
תיכון: (כולל הסבר)

ניתן לראות כי דרשנו שהאלגוריתם והסימולציה אשר תלויים אחד בשני יעבדו בצורה סינכרונית ומדוייקת על מנת להבטיח את אמינות המערכת בצורה הטובה יותר, זמן הסימולציה פועל כחלק מהלמידה ולכן יצרנו מערכת יעילה ומהירה ככל שניתן ולכן שליפה של קבצים אשר נשמרים לנו במחשב המפעיל תחסוך זמן מיותר בשליפתם על ידי הסימולציה והאלגוריתם.  
  
כמו שניתן לראות, סביבות העבודה של הסימולטור והאלגוריתם שונות ועל ידי קבצים משותפים גרמנו לסביבות זרות לעבוד יחדיו, לקו מחשבה זה ישנו יתרון נוסף והוא ששמירתם של הקבצים בסוף ההרצה תאפשר המשך למידה (שמירת משקולות) מהנקודה הטובה ביותר כך שאם נרצה בעתיד להוסיף פיצ'ר שונה לחישוב הרעש והמיצוע נוכל לעשות זאת בעזרת המצב השמור בקובץ.  
  
בחרנו לייצר את הסימולטור בספריית OpenGL, בשפת ++C, משום שרצינו להבטיח סימולציה קלה להרצה.  
את האלגוריתם תכננו ומימשנו בשפת Python, כיוון שזו השפה המתאימה והנוחה ביותר לביצוע האלגוריתם והדרישות.

הארכיטקטורה פועלת כך שהאלגוריתם מפעיל את הסימולטור ומקבל ממנו את המידע המבוקש (פידבק) על מנת לחשב את הצעד הבא של האלגוריתם כדי להבטיח שיפור גדול ככל הניתן.

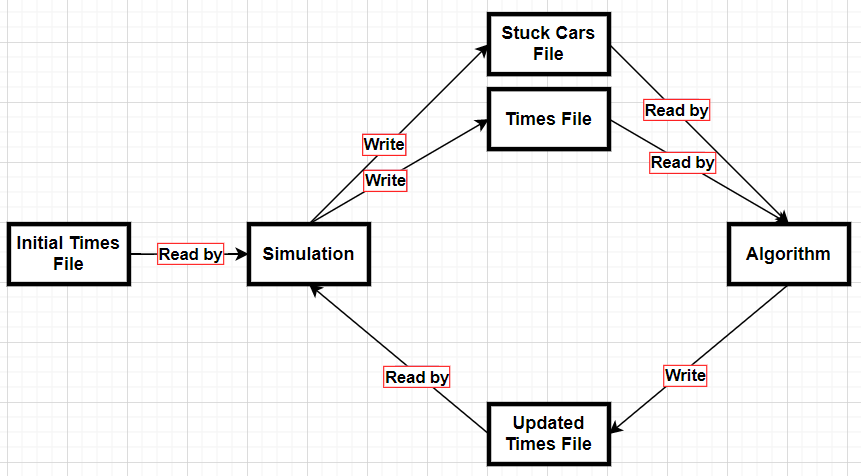
חלופות עיצוב (Design):

|  |  |  |
| --- | --- | --- |
| עיצוב | יתרונות | חסרונות |
| הפעלת הסימולציה דרך האלגוריתם | * האלגוריתם אחראי על הפעלת כל התכנית. * האלגוריתם תלוי בסימולציה (מחכה לתוצאות שלה) | * לא ניתנת גישה נוחה למשתמש |
| הפעלת האלגוריתם דרך הסימולציה | * החלק הראשון של התכנית הוא הרצת הסימולציה | * הסימולציה לא תלויה באלגוריתם * הסימולציה לא יכולה ולא צריכה להפעיל את האלגוריתם |
| תוכנית מקומית שאחראית על הפעלת המערכת | * תכנית שאחראית על כל החלקים * ניתן להוסיף ממשק משתמש | * חלק לא נחוץ בפרוייקט שלנו * מקשה על התקשורת בין האלגוריתם לסימולציה |
| תוכנית בענן שאחראית על הפעלת המערכת | * פעולה בענן ושמירת הנתונים למקרה של תקלה | * הופך את התכנית לפגיעה מבחינת אבטחת מידע * מצריך גישה לרשת |

ניתן לראות כי העיצוב "הפעלת הסימולציה דרך האלגוריתם" הוא העיצוב המשתלם, המהיר והפשוט ביותר עבור הפרויקט שלנו.

ניתן לקחת את האופציה של "תוכנית מקומית שאחראית על הפעלת המערכת" כפיתוח עתידי.

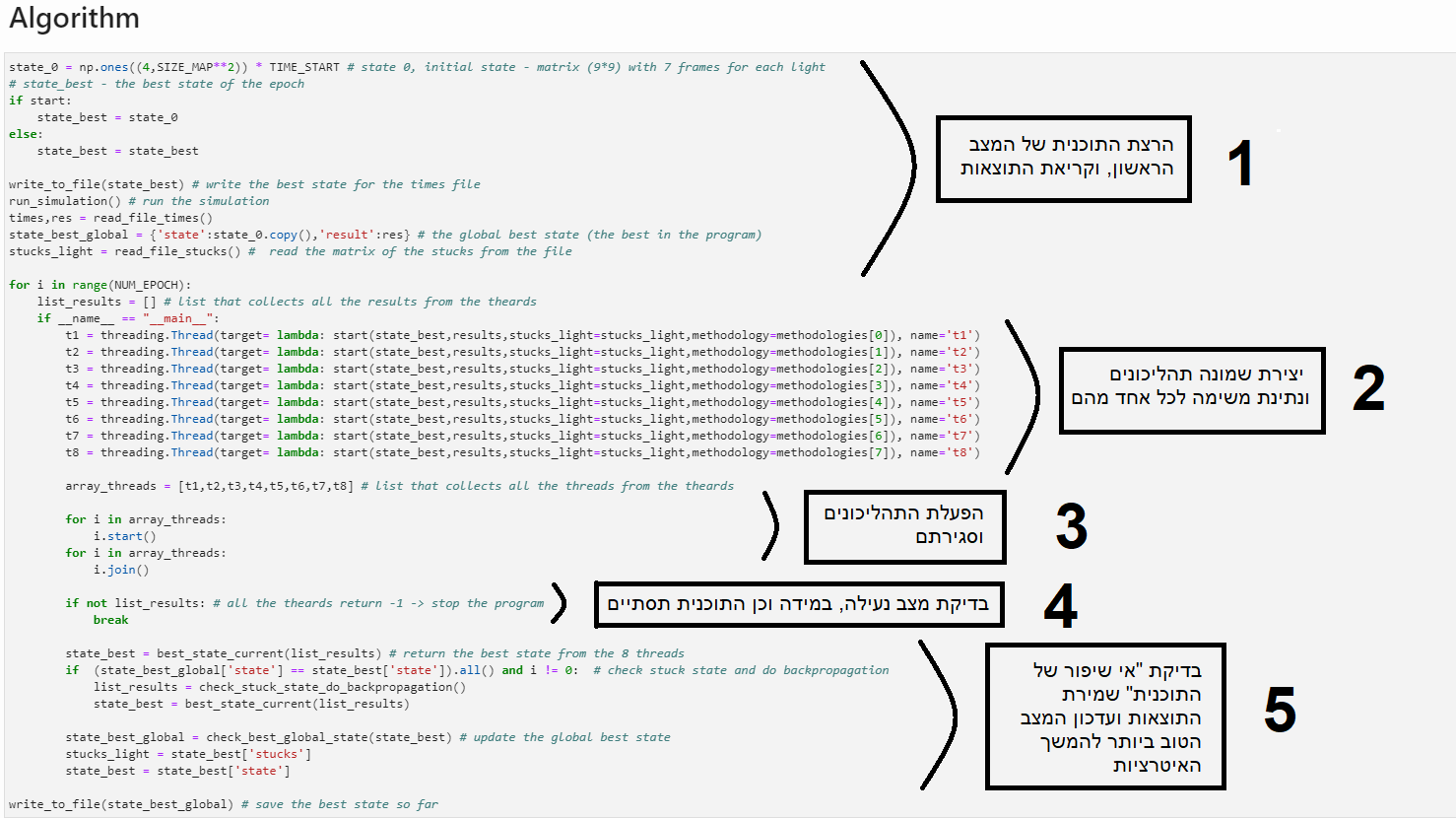
תרשים בלוקים כללי של המערכת:



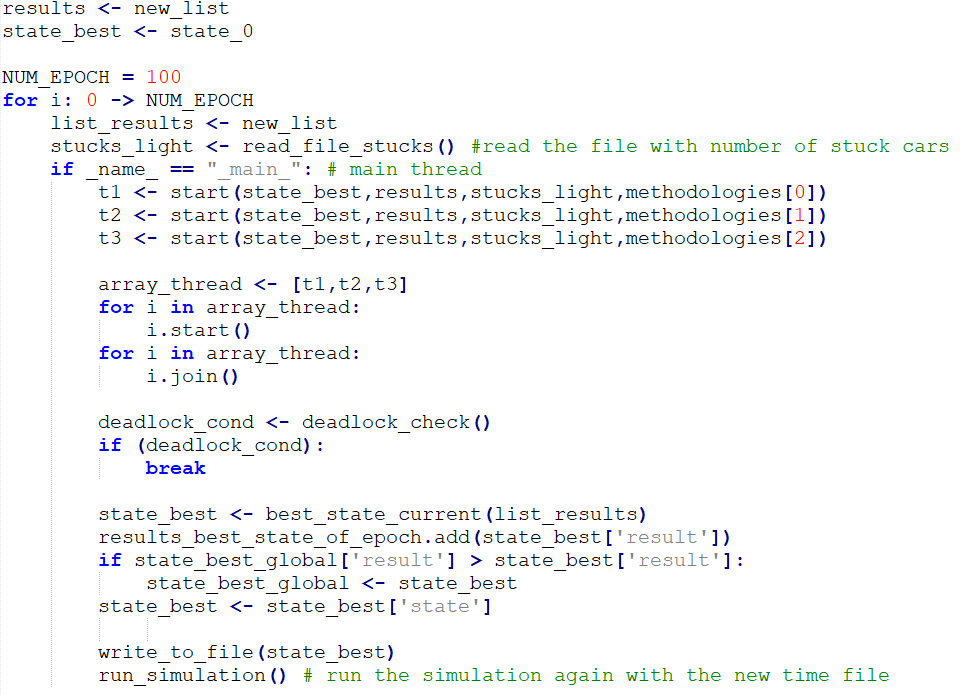
אלגוריתמים:

**אלגוריתם מבוסס למידת מכונה:**

קוד:



פסאודו-קוד:

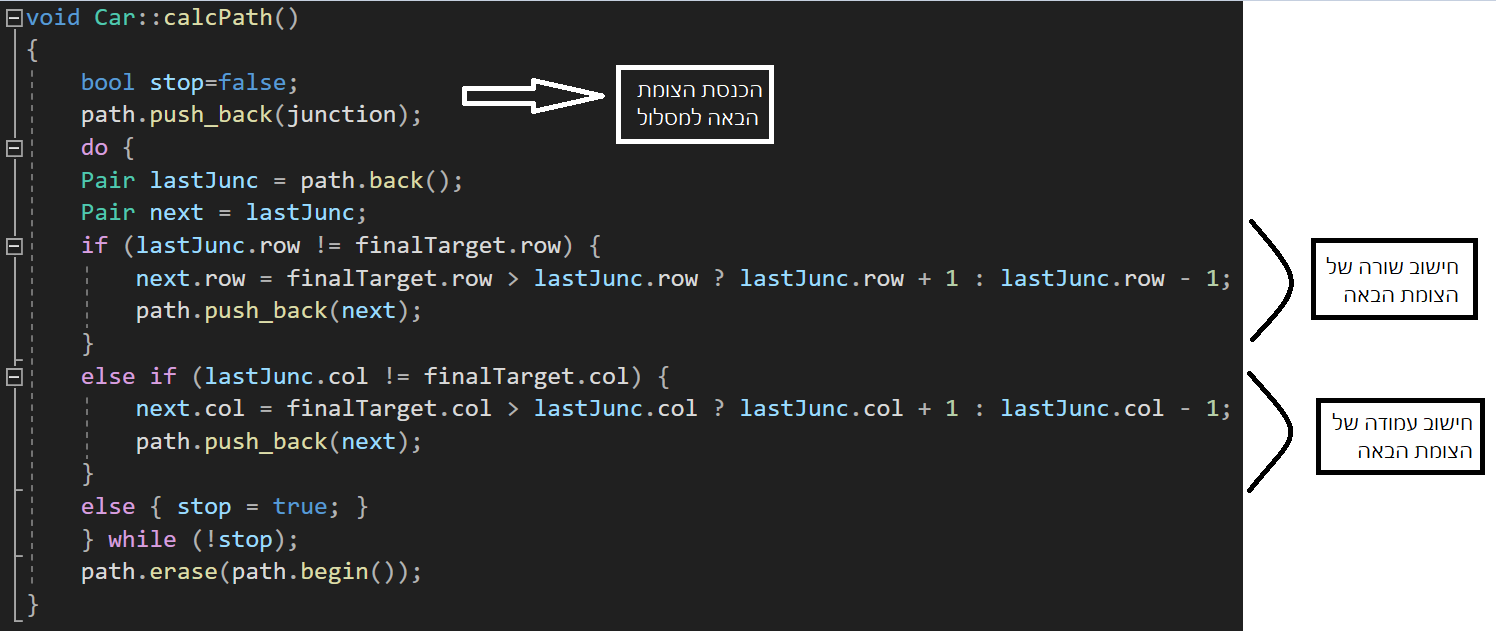


הוכחת נכונות:

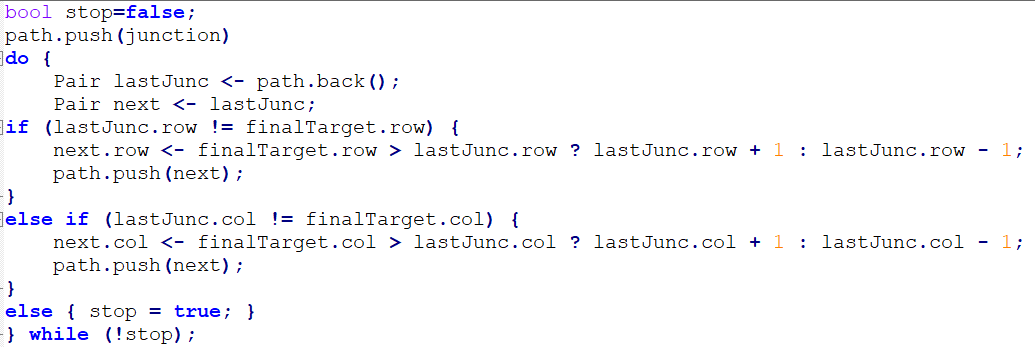
* המערכת שלנו מבטיחה מינימום מקומי כך שבכל איטרציה (epoch) אנו בוחרים את התוצאה הטובה ביותר. במידה ואין שיפור באף אחת מהשיטות, נבחר את השיטה שהניבה המצב הטוב ביותר.
* התוצאה הסופית המוצגת בעזרת הסימולציה שמראה שיפור, מהווה הוכחת נכונות. כל מצב שמחושב על ידי האלגוריתם ניתן לבדיקה מבחינה ויזואלית על ידי הסימולציה וספירה באופן ידני לצורך בדיקה.

**אלגוריתם למציאת מסלול:**

קוד:

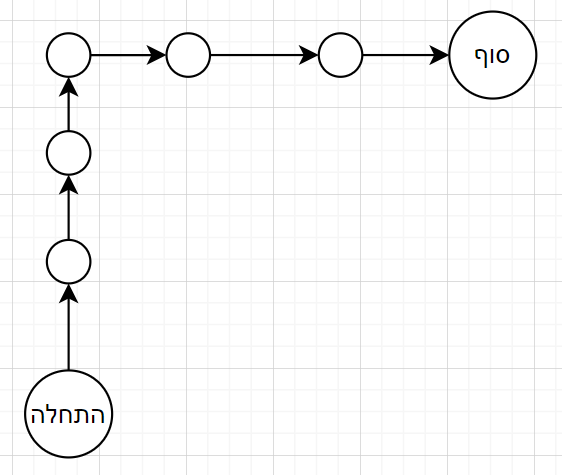


פסאודו-קוד:



הסבר:

האלגוריתם מוסיף צמתים קודם על פי שורה (כלומר כל עוד לא הגענו לשורה של צומת היעד) ואחרי זה על פי עמודה (כלומר כל עוד לא הגענו לעמודה של צומת היעד).



על פי התמונה ניתן לראות כי כל עוד לא הגענו לשורת היעד אנחנו ממשיכים לעלות בשורות. רק כאשר אנחנו באותה שורה עם צומת היעד, אנו מתחילים לנוע בעמודות.

# 

# **17. התוצר**

התוצרים הסופיים של המערכת:

קובץ זמנים אשר בעת הרצתו על הסימולציה נקבל מספר תקיעות מופחת בשיעור של עד 85%. שיפור הזמנים שבקובץ הוא תהליך שנעשה על ידי האלגוריתם שבנינו.

מכלולי המערכת: אלגוריתם מבוסס למידת מכונה וסימולציה להדגמת יכולות המערכת.

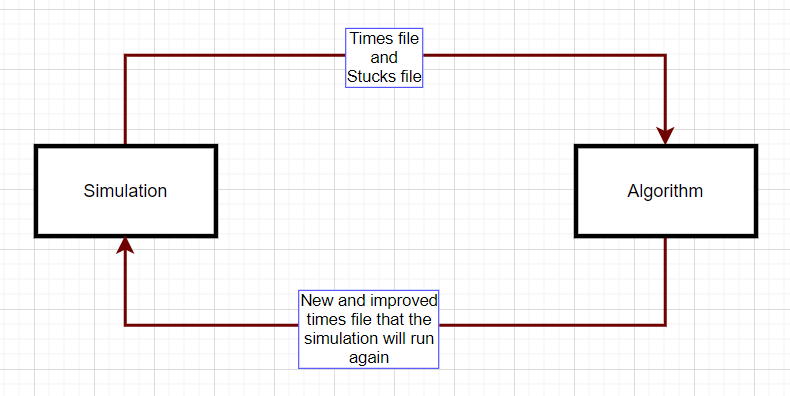
חבילות התוכנה שפותחו:

* סימולציה בתוכנת Visual Studio בשפת ++C ובטכנולוגיית OpenGL.
* אלגוריתם מבוסס למידת מכונה בשפת Python.

תקציר האלגוריתם: (שמופיע בפירוט במסמך SRD)

סימולציה אשר כותבת לקבצים את מספר התקיעות שהיו בעת הרצתה (מספר התקיעות הכללי ומספר התקיעות בכל רמזור). האלגוריתם מקבל את מספר התקיעות ופועל לשיפורו (כמפורט באלגוריתם המלא) ומחזיר קובץ זמנים חדש לסימולציה. התכנית נגמרת כאשר נסיים את האיטרציה האחרונה או כאשר נגיע למצב שבו כל אחת מהשיטות תקועה (במצב deadlock) ולא יכולה להתקדם. במקרה כזה יילקח הזמן הטוב ביותר עד לרגע התקיעה.

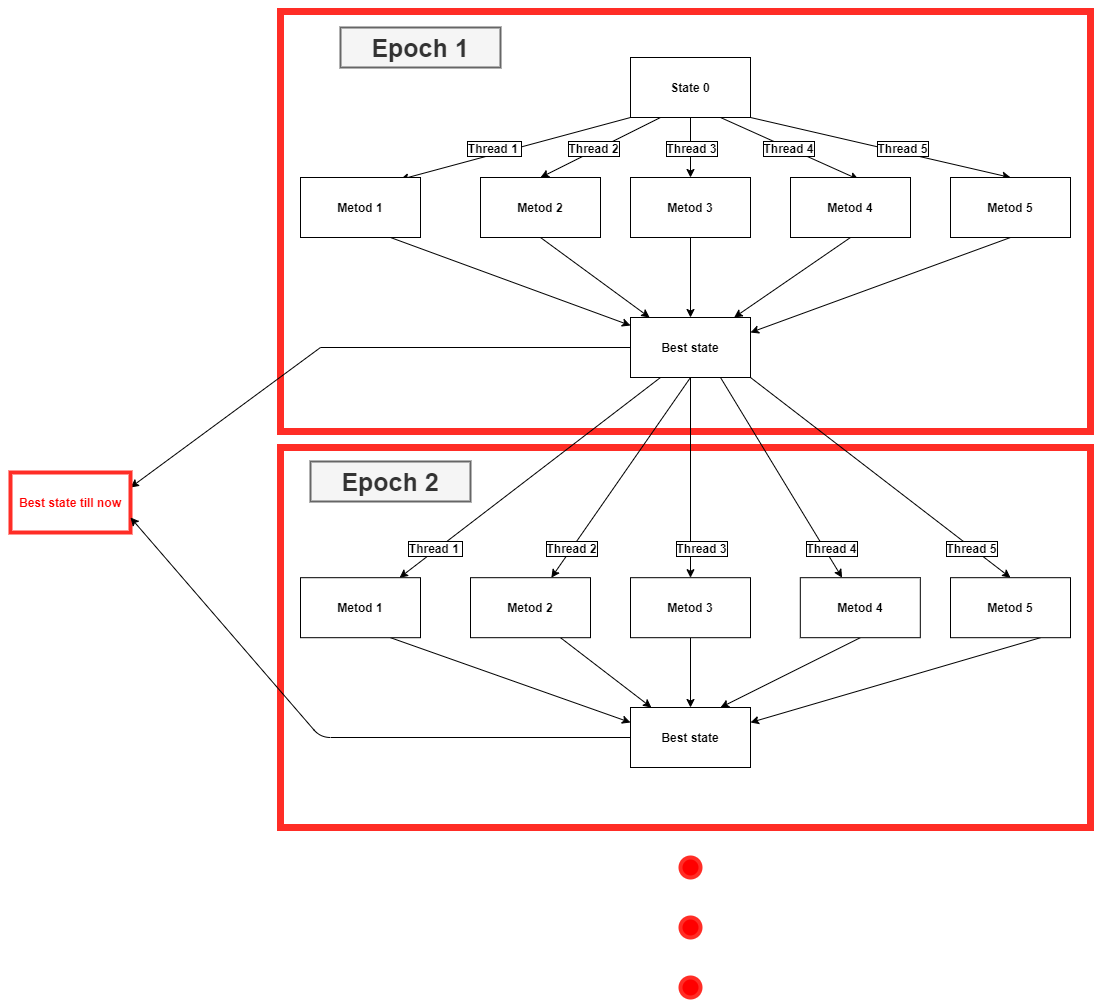
תקציר האלגוריתם של התכנית: (מפורט בפרק אפיון התוכנית)



בתמונה: דוגמא של הרצת סימולציה

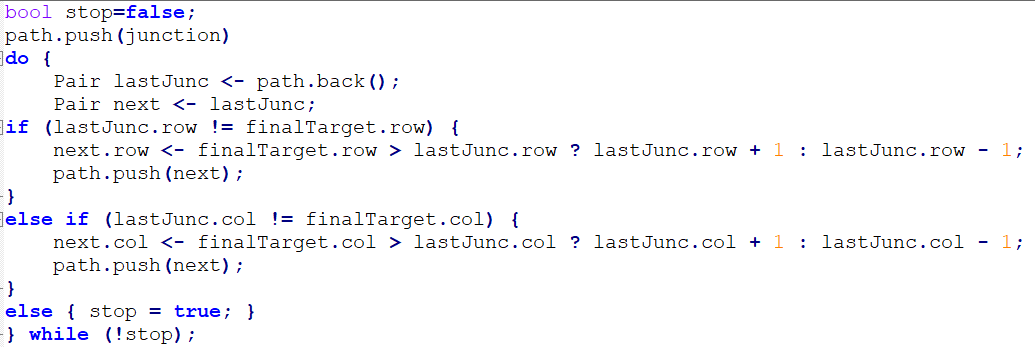
תיאור התמונה: סימולציה שמציגה עיר שמורכבת מצמתים: לכל צומת ארבע כבישים ולכל כביש רמזור אחד. ניתן לראות שבכל רגע נתון בכל צומת יש רק רמזור אחד ירוק. הרכבים בתמונה מיוצגים על ידי ריבוע שחור. רכבים שנתקעו (שאינם יכולים להגיע לצומת הבאה במסלול) ייצבעו באדום.

# 

בתמונה: דוגמא של האלגוריתם על שתי הרצות לאחר שקיבל את הקבצים הרלוונטיים מהסימולציה

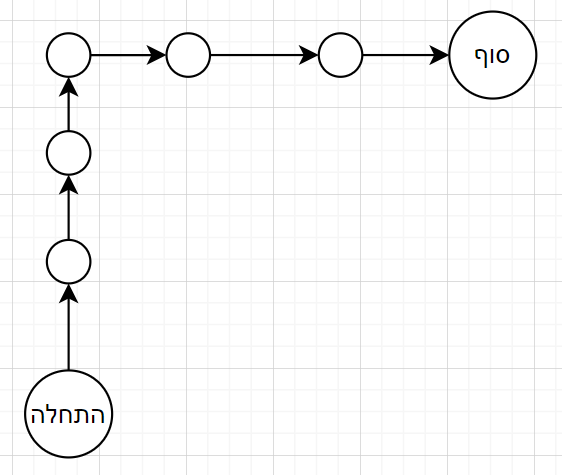
אלגוריתמים:

**אלגוריתם לחישוב מסלול:** (פסאודו קוד)



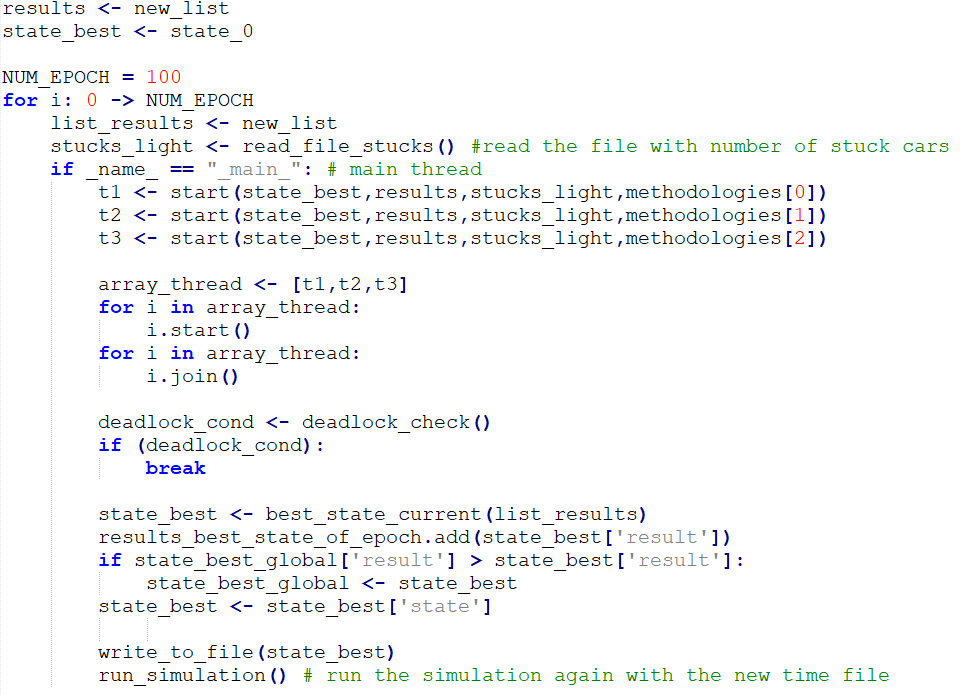
הסבר:

האלגוריתם מוסיף צמתים קודם על פי שורה (כלומר כל עוד לא הגענו לשורה של צומת היעד) ואחרי זה על פי עמודה (כלומר כל עוד לא הגענו לעמודה של צומת היעד).



על פי התמונה ניתן לראות כי כל עוד לא הגענו לשורת היעד אנחנו ממשיכים לעלות בשורות. רק כאשר אנחנו באותה שורה עם צומת היעד, אנו מתחילים לנוע בעמודות.

**אלגוריתם מבוסס למידת מכונה:** (פסאודו קוד עם דוגמא של שלוש מתודולוגיות. באלגוריתם המקורי יש יותר מתולוגיות ויותר תהליכונים (threads))

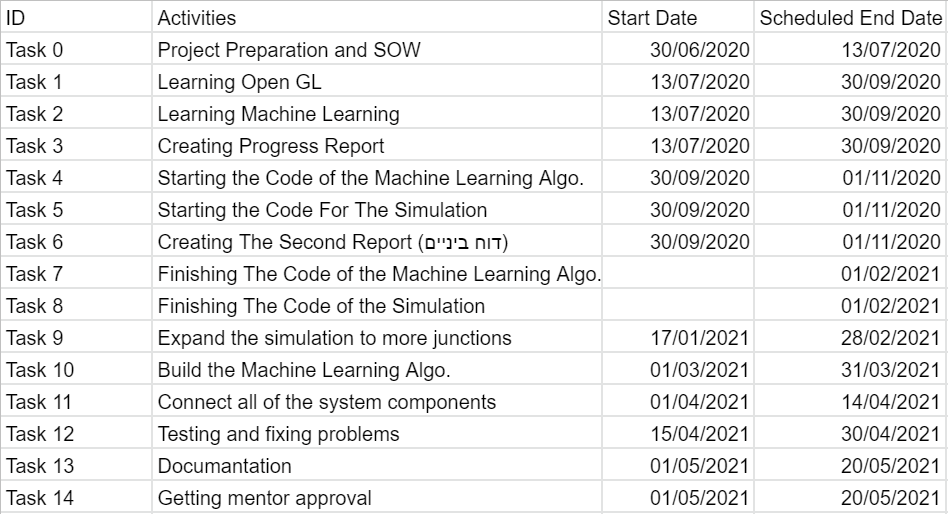


האלגוריתם מחלק את המשימות למספר התהליכונים (threads הנדרש). כל תהליכון מחשב על פי השיטה שנתנה לו ומחזיר את התוצאה. האלגוריתם עובר על התוצאות ובוחר את התוצאה הטובה יותר, שתהווה המצב ההתחלתי לאיטרציה הבאה. האלגוריתם בודק אם התוצאה הטובה של האיטרציה היא התוצאה הטובה ביותר בתכנית (global) ומעדכן בהתאם.

# 

# **18. תכנון ותכנית ביצוע הפרויקט (Project Planning)**

תוכנית עבודה סופית:



ריכוז שינויים:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| השינוי | היכן בוצע | מתי | ביוזמת מי השינוי | משמעות השינוי |
| מעבר ממדידת זמנים למדידת מספר תקיעות | בפרוייקט | בעת הבנה שלזמנים אין משמעות | הסטודנטים | הוספת פונקציה לבדיקת תקיעות וספרנו את כמות התקיעות. |
| הוספת קוד לזיהוי מצב נעילה (deadlock) | סימולציה | בעת הבנה שיש צורך בזיהוי מצב זה | הסטודנטים | יצירת פונקציה לזיהוי מצב נעילה ופעולה בהתאם (עצירת הסימולציה והחזרת ערך 1-) |
| פתירת הבעיה באמצעות אלגוריתם שלנו ולא Q-Learning | אלגוריתם | חקירה מעמיקה על אלגוריתם Q-Learning ונסיון להתאים אותו לבעיה שלנו. | הסטודנטים | יצירת אלגוריתם משלנו לפתירת הבעיה. |

ניהול וגידור סיכונים:

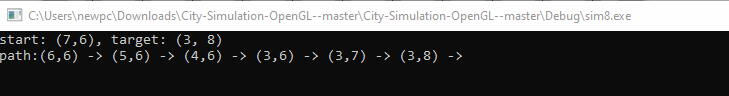
אין.

# **19. בדיקות והערכה (Software Testing and Evaluation)**

מפרט ובדיקות תוכנה:

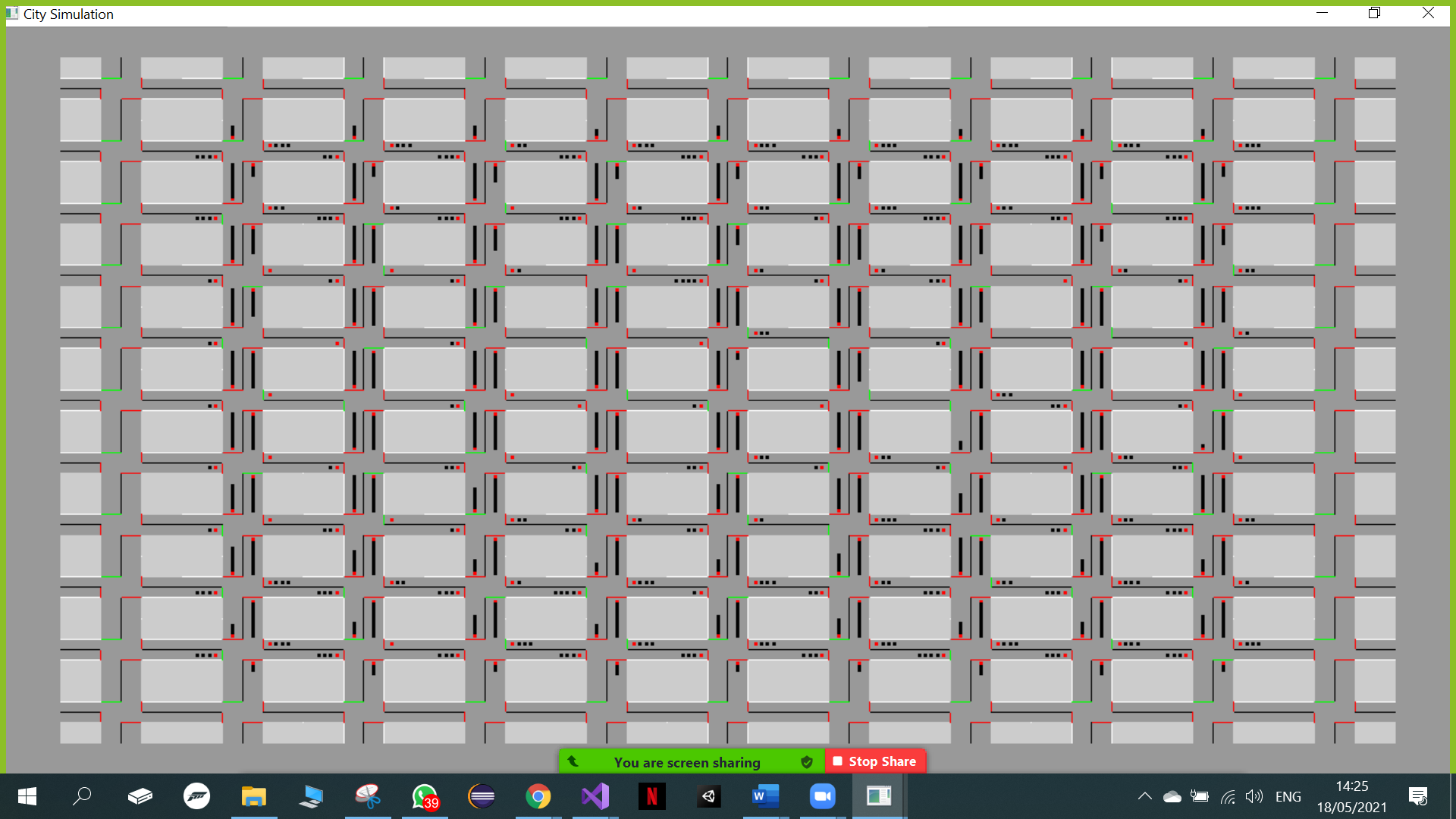
הבדיקות בפרויקט שלנו כוללות בדיקות לסימולציה ולאלגוריתם בנפרד, וכן בדיקת איטגרציה בין המערכות. בנוסף לכך, התבצעה בדיקה של מדדים על מנת לוודא עמידה במדדים שהוגדרו ב-SOW, וכן בדיקת זמנים להרצה מקבילית אל מול הרצה סדרתית.

דוגמאות הפעלה:

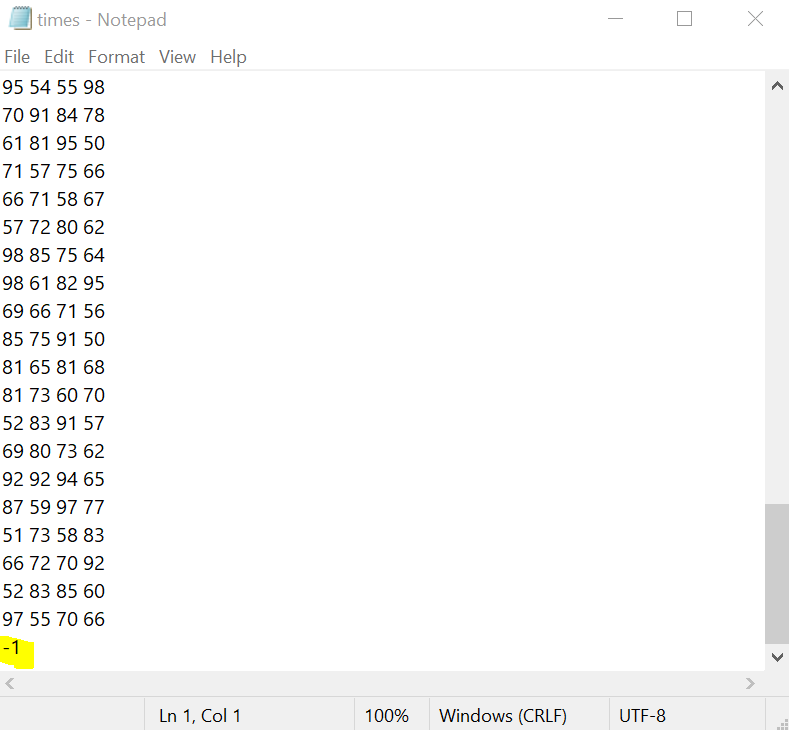
**דוגמא 1 - אלגוריתם למציאת מסלול:**  


על ידי פונקציה שנקראת showPath (במחלקה Car) אנו יכולים לראות את כל הצמתים במסלול של מכונית ספציפית, וכך לבדוק האם ישנם צמתים שלא במסלול. (ניתן לראות את האלגוריתם בפירוט בחלק המתאים בספר).

**דוגמא 2 - מצב deadlock:**

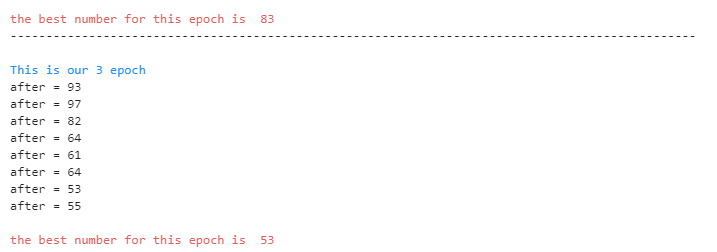


בתמונה מתואר מצב נעילה (deadlock) שבו כל הרכבים לא יכולים לזוז. ניתן לראות כי בקובץ הזמנים המצורף, נכתב המספר 1- (כלומר, המערכת זיהתה שהיא במצב נעילה ופעלה בהתאם).



בתמונה: קובץ הזמנים לאחר שהמערכת נכנסה למצב deadlock. ניתן לראות במקום כמות התקיעות נכתב המספר 1-, המעיד על תקיעה.

**דוגמא 3 - בחירת המצב הכי טוב (בהרצת איטרציה אחת):**



על ידי הדפסות מתאימות בקוד ניתן לראות בכל איטרציה מה הייתה התוצאה של כל תהליכון ומהי התוצאה הסופית שנבחרה.

בדיקות שמישות:

לאחר בירור ניתן לומר שבדיקות שימוש (usability) לא רלוונטיות בפרויקט שלנו מכיוון שהוא אלגוריתם (ולא מערכת) שאין בו חלק של התממשקות מול המשתמש.

בדיקת מדדים:

שיפור כללי - האלגוריתם שלנו מניב תוצאה עם שיפור של עד 90%, בתלות ב:

* כמות האיטרציות (epochs) שנתנו.
* מצב התחלתי (זמנים התחלתיים).
* מספר השיטות שהאלגוריתם מחשב ושמתוכן הוא יכול לבחור.
* גודל העיר.
* מספר רכבים.

שיפור בזמני הריצה של איטרציה אחת בריצה מקבילית בהשוואה לריצה סדרתית:

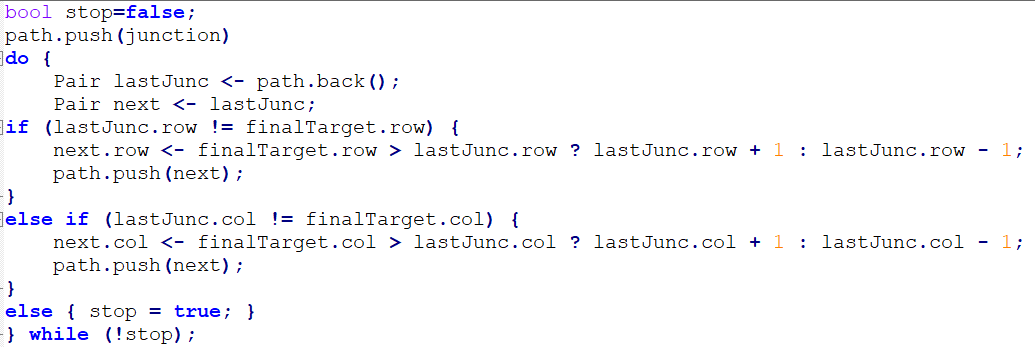
* הרצה של כל 8 השיטות
* זמן הרצה סדרתית: 2:17 דקות (2 דקות ו-17 שניות) = **137 שניות**.
* זמן הרצה מקבילית: 0:19 דקות (19 שניות) = **19 שניות**.
* ניתן לראות שיפור של פי 7.2 עבור 8 שיטות. ניתן להניח שככל שמספר השיטות גדל כך היחס גדל.
* מסקנה: ההחלטה להריץ את התכנית האופן מקבילי מקטינה משמעותית את זמן הלמידה.

אמינות:

* בעת הרצה הפרויקט באופן סדרתי ובאופן מקבילי - התקבל שיפור זהה.
* בכל הרצה של האלגוריתם, עם אותם נתונים התחלתיים, הגענו לאותה תוצאה.

אלגוריתמים:

**אלגוריתם לחישוב מסלול:** (פסאודו קוד)



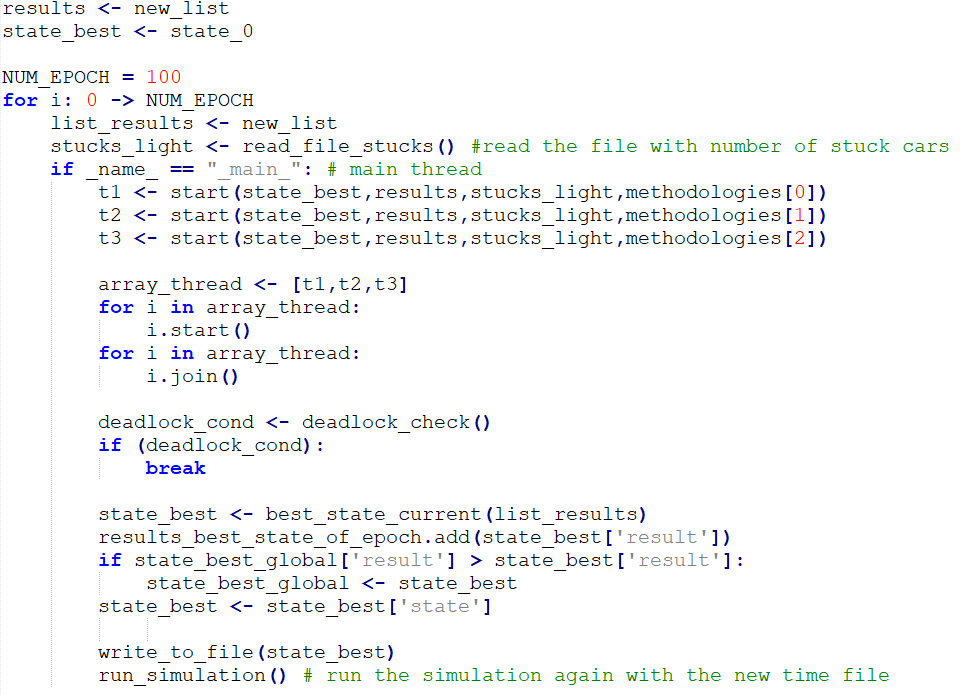
הסבר:

האלגוריתם מוסיף צמתים קודם על פי שורה (כלומר כל עוד לא הגענו לשורה של צומת היעד) ואחרי זה על פי עמודה (כלומר כל עוד לא הגענו לעמודה של צומת היעד).

בדיקות שנעשו עבור אלגוריתם זה:

* בדיקה שנוצר מסלול על ידי הצמתים הרלוונטיים.
* בדיקה שאכן נוצר מסלול על סמך שורות ואחר כך על סמך עמודות.

**אלגוריתם מבוסס למידת מכונה:** (פסאודו קוד עם דוגמא של שלוש מתודולוגיות. באלגוריתם המקורי ייתכנו יותר מתודולוגיות ויותר תהליכונים (threads))



האלגוריתם מחלק את המשימות למספר התהליכונים (threads) הנדרש. כל תהליכון מחשב על פי השיטה שנתנה לו ומחזיר את התוצאה. האלגוריתם עובר על התוצאות ובוחר את התוצאה הטובה יותר, שתהווה המצב ההתחלתי לאיטרציה הבאה. האלגוריתם בודק אם התוצאה הטובה של האיטרציה היא התוצאה הטובה ביותר בתכנית (global) ומעדכן בהתאם.

בדיקות שנעשו עבור אלגוריתם זה:

* בדיקה ליצירת תהליכים.
* בדיקה של חלוקת מטלות לתהליכונים.
* בדיקה של אסיפת התשובות מכל התהליכונים.
* בדיקה של סיום התהליכונים.
* בדיקה להוצאת המצב הטוב ביותר מכל התהליכונים.

**20. סיכום ומסקנות**

לסיכום, ניתן לומר שהגענו ליעד אותו הצבנו.

מערכת - בנינו מערכת שמורכבת מסימולציה ואלגוריתם, אשר מבוססת על הקשר בניהם. כל אחת מהמערכות עובדות ביחד ולחוד. מטרת הסימולציה היא להציג מצב מסויים של עיר על פי קובץ זמנים, ועמדנו במטרה זו. מטרת האלגוריתם היא לקבל את קובץ הזמנים ולשפר אותו, ועמדנו במטרה זו. כמו כן, האלגוריתם והסימולציה מתקשרים ועובדים ביחד כמתוכנן.

שיפור - בעזרת האלגוריתם שבנינו הצלחנו להגיע לשיפור משמעותי של 85% ואף יותר (תלוי בכמות האיטרציות (epochs) ובגודל העיר). בעזרת הסימולציה, היינו יכולים להציג את השיפור.

בעיות שהיו לנו:

* גרסאת האלפא של הסימולציה, שהוצגה בדוח הביניים, נבנתה עבור צומת אחת ולא מתוך מחשבה עתידית להרחיב אותה לעיר של כמה צמתים. לכן, התחלנו ביצירת הסימולציה מההתחלה, הפעם תוך כוונה להרחיב אותה לעיר. עבדנו על פי תכנות מונחה עצמים על מנת לשמור נתונים עבור כל צומת, כביש ורכב, והתבססנו על הקשר בין אובייקטים אלו. עשינו שימוש ביכולות הציור של OpenGL על מנת "לשכפל" צומת אחת לכדי עיר.
* אחת הבעיות העיקריות היו שלאחר חקירה ונסיון של אלגוריתם Q-Learning הבנו שהוא לא מתאים לפתרון הבעיה שלנו כיוון שהוא מניב פתרון שאינו מתכנס.

# **21. הצעה לעבודת המשך**

הצעות להרחבה עתידית של הפרויקט:

|  |  |  |
| --- | --- | --- |
| **הצעה** | **כיצד ההצעה תשפר את המערכת?** | **מדוע לא הוספנו?** |
| הוספת ממשק משתמש | מערכת ידידותית יותר למשתמש, נותן למשתמש בחירה ושליטה. | לא רלוונטי לפתרון הבעיה, מכיוון שהמערכת שלנו מביאה פתרון לבעית עומסים. |
| תמיכה במספר סוגי כבישים (דו סטרי, צמתי T וכו') | התאמה טובה יותר למציאות | תוספת זו דורשת עבודה רבה וזמן נוסף. |
| שיפור האלגוריתם (הוספת שיטות מיצוע, שיפור של אלגוריתם מונטה קרלו) | שיפור התוצאות והתכנסות לפתרון טוב יותר ומהר יותר | תוספת זו דורשת עבודה רבה וזמן נוסף. |
| שיפור הסימולציה (הוספת אנימציות של נסיעת רכבים) | מדמה מצב מציאותי יותר מבחינה ויזואלית | הוספת אנימציות של נסיעת רכבים גרעה מביצועי התכנית ולא תורמת לפתרון הבעיה. |
| שיפור של פונקצית חישוב מסלול | חישוב מסלול יעיל וטוב יותר | המסלול שהרכבים עוברים בו לא קשור לפתרון הבעיה. |
| הוספת בסיס נתונים | גישה ממחשבים מרוחקים לנתונים | על מנת שלא לפגוע במהירות שליפת הנתונים ולא לדרוש חיבור לרשת. |
| טיפול מצבי קיצון (חסימות כבישים, תיקונים בכביש, תאונה וכו') | התאמה טובה יותר למציאות | תוספת זו דורשת עבודה רבה וזמן נוסף. |

# **22. רשימת מקורות**

Image Filtering:

1.

[Kernel - Image Processing (wikipedia)](https://en.wikipedia.org/wiki/Kernel_(image_processing))

2.

[Image Filters: Gaussian Blur](https://aryamansharda.medium.com/image-filters-gaussian-blur-eb36db6781b1)

(Written by: Aryaman Sharda, Publish date: 25.01.21)

Q-Learning:

3.

[Reinforcement Learning for Traffic Optimization](http://cs229.stanford.edu/proj2016spr/report/047.pdf)

(Written by: Matt Stevens & Christopher Yeh, Standford University)

Monte Carlo:

4.

[Monte Carlo Tree Search](https://en.wikipedia.org/wiki/Monte_Carlo_tree_search) - Wikipedia

5.

[Monte Carlo Tree Search - video](https://www.youtube.com/watch?v=onBYsen2_eA)

מאמר שנסקר בשפה האנגלית:

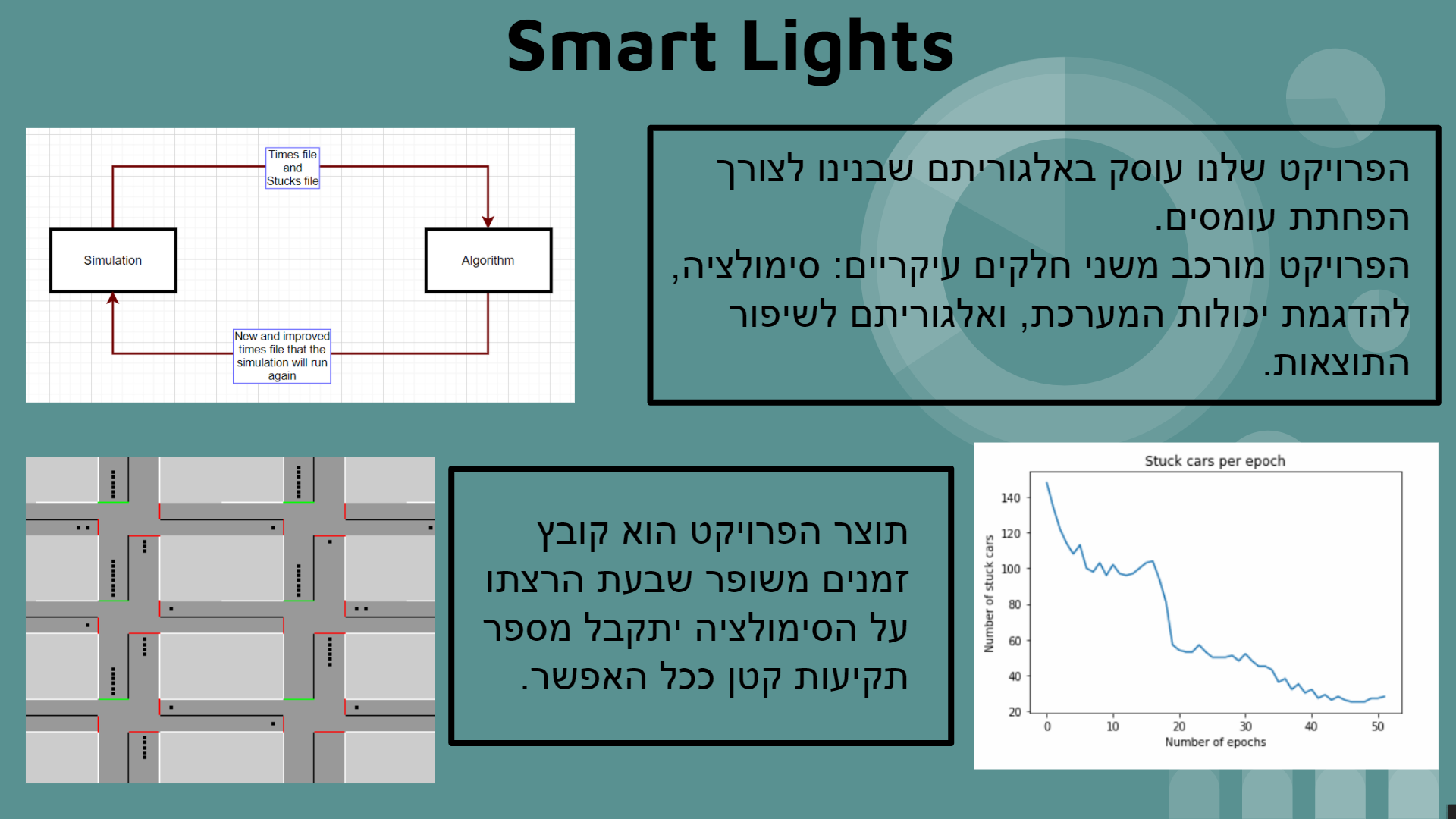
6.

[Image Filtering Algorithms and Techniques: A Review](https://www.researchgate.net/publication/325681876_Image_Filtering_Algorithms_and_Techniques_A_Review)

(Written by: Gaurav Gupta, Shoolini University, Publish date: October 2013)

# **23. נספחים**

**נספח פוסטר הפרויקט:**



**נספח מאמר בשפה האנגלית:** מצורף בנפרד (מספר 6 ברשימת המקורות)

**נספח אלגוריתמים**

בנספח זה נפרט על שלושה אלגוריתמים:

* אלגוריתם למידת מכונה
* אלגוריתם לחישוב מסלול
* סימולציה

**אלגוריתם למידת מכונה**

אלגוריתם - האלגוריתם מקבל את שני הקבצים (קובץ זמנים וקובץ תקיעות), קובץ התקיעות משמש את האלגוריתם בתור מפה עבור האזורים הבעייתים (שבהם מספר גדול של תקיעות). האלגוריתם מבצע חישוב מקדים עבור כל רמזור - - ויוצר מטריצת זמנים חדשה () אשר מייצגת את הרעש לשימוש האלגוריתם. האלגוריתם מקצה שמונה תהליכונים (threads) שונים, שכל אחד מהם אחראי על מיצוע בשיטת חישוב שונה. כל אחד מהתהליכונים יוצר קובץ זמנים חדש על פי המיצוע שלו ומריץ את קובץ הזמנים שלו על הסימולציה על מנת לקבל מספר תקיעות ולהציג את הסימולציה עם הזמנים שלו. כל אחד מהתהליכונים מחזיר מילון אשר בנוי משלושה מפתחות - אחד שומר את מטריצת הזמנים (state) שחושבה על ידו, השני שומר את מספר התקיעות שהתקבל מהרצת הזמנים שלו על הסימולציה שלו והשלישי שומר את קובץ התקיעות. כל תהליכון שומר את המילון שלו בתוך מערך (שנמצא ב-main) שמכיל את כל המילונים מכל התהליכונים ומסיים את תפקידו. נבחר מתוך המערך במטריצת הזמנים עם מספר התקיעות הכללי הקטן ביותר ונשמור אותה בתור קובץ הזמנים הטוב ביותר (\*) **באותה איטרציה**. בנוסף, אנו שומרים ומעדכנים בסוף כל איטרציה (\*\*) את מטריצת הזמנים הטובה ביותר של **התוכנית**. בסוף הרצת כל איטרציה, קובץ הזמנים החדש שמחולק לכל התהליכונים לקראת האיטרציה הבאה הינו הקובץ הטוב ביותר שנבחר מהאיטרציה הקודמת. כך, מובטח שינוי מאיטרציה לאיטרציה.

מקרה חריג שעלול להתרחש (באלגוריתם): הסימולציות של כל התהליכונים האיטרציה ספציפית מגיעות למצב נעילה (deadlock). האלגוריתם מזהה מצב זה (בעת קריאה של הערך 1- מקובץ הזמנים שנכתב על ידי הסימולציה), מסיים את התוכנית ומציג על הסימולציה את קובץ הזמנים הטוב ביותר עד כה.

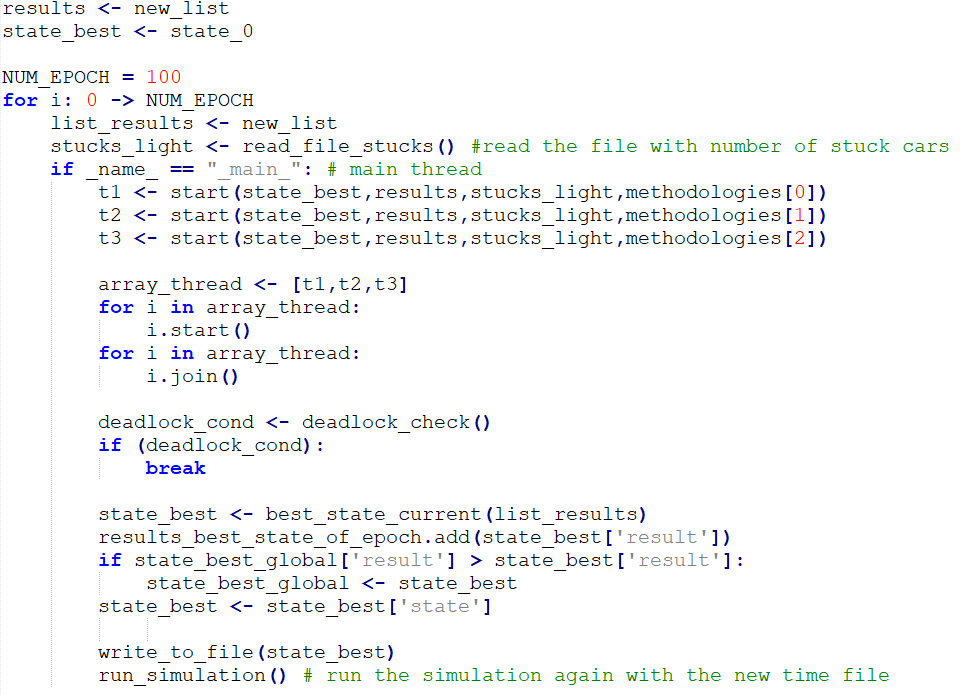
מקרה חריג נוסף שעלול להתרחש (באלגוריתם): כאשר התהליכון הנבחר הוא תהליכון שלא ניתן לשפר אותו ויוצר חזרתיות של מצבים (אין שיפור ואנו נתקעים על אותו המצב), במצב זה המערכת תזהה את הבעיה ותכניס את המצבים הבעייתיים לתוך "רשימה שחורה" של מצבים שמהם אנחנו נמנע ובכך נפתרת הבעיה זו.

סיום התכנית - התכנית תסתיים באחד משני מקרים:

1. סיימנו את האיטרציה האחרונה והתקבל קובץ הזמנים הכי טוב שיכול להתקבל במסגרת האיטרציות שנתנו.
2. כל אחת מהשיטות הניבה קובץ זמנים אשר הכניס את הסימולציה למצב נעילה (deadlock). במקרה כזה ייבחר הקובץ הטוב ביותר האחרון שהצלחנו להגיע אליו עד כה.
3. במידה ונגיע למצב ללא תקיעות התכנית תסתיים ויישמר המצב.

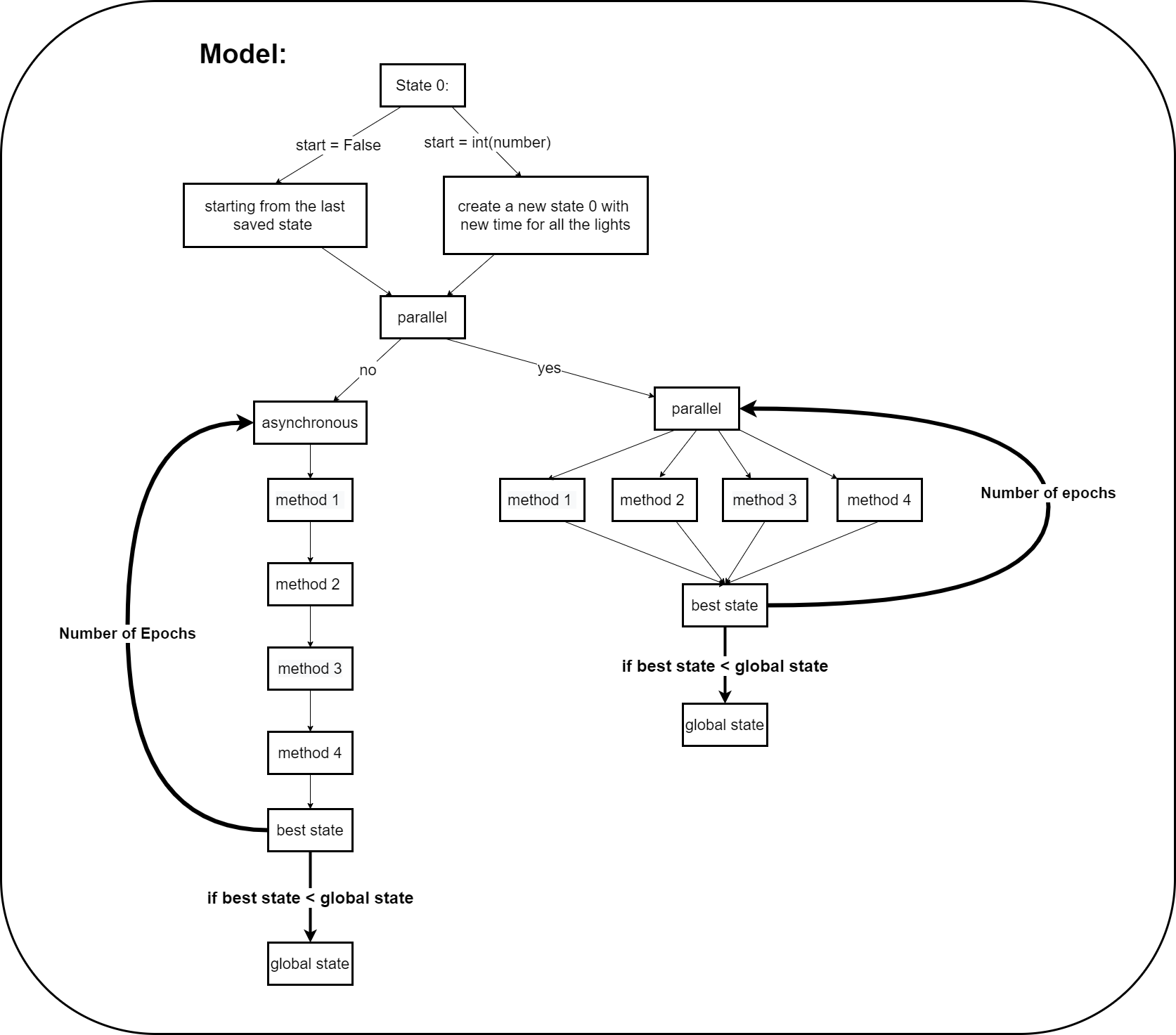
\* קובץ הזמנים הטוב ביותר - קובץ הזמנים שהניב את מספר התקיעות המינימאלי.

\*\* איטרציה = epoch



פסאודו קוד:

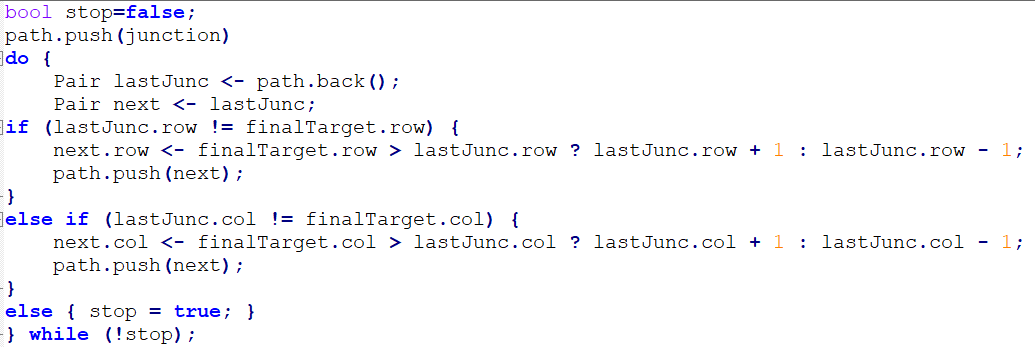
האלגוריתם מחלק את המשימות למספר התהליכונים (threads הנדרש). כל תהליכון מחשב על פי השיטה שנתנה לו ומחזיר את התוצאה. האלגוריתם עובר על התוצאות ובוחר את התוצאה הטובה יותר, שתהווה המצב ההתחלתי לאיטרציה הבאה. האלגוריתם בודק אם התוצאה הטובה של האיטרציה היא התוצאה הטובה ביותר בתכנית (global) ומעדכן בהתאם.



המודל מקבל שלושה פרמטרים לביצוע הלמידה:

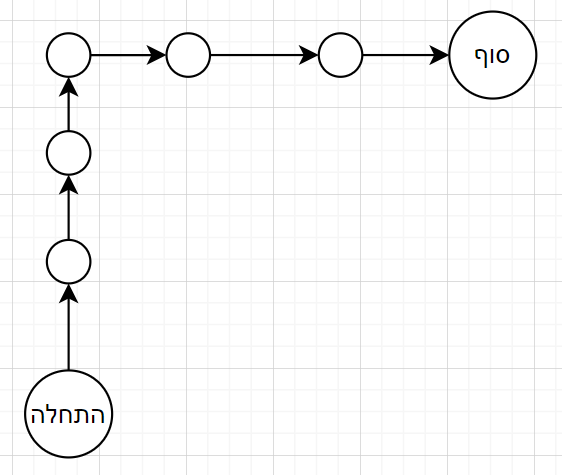
* start - קובע האם להתחיל את הלמידה (state 0) מקובץ זמנים חדש או מקובץ הזמנים שנוצר מההרצה הקודמת.
* NUM\_EPOCHS - קובע את מספר האיטרציות (epoch).
* parallel - קובע האם לבצע למידה בצורה מקבילית או סדרתית.

**אלגוריתם לחישוב מסלול**



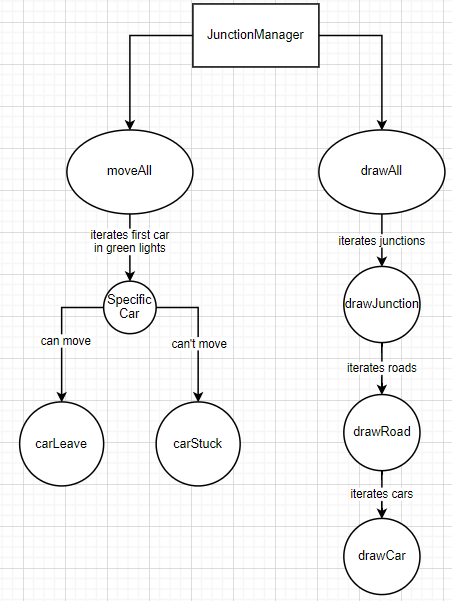
הסבר:

האלגוריתם מוסיף צמתים קודם על פי שורה (כלומר כל עוד לא הגענו לשורה של צומת היעד) ואחר כך על פי עמודה (כלומר כל עוד לא הגענו לעמודה של צומת היעד).



על פי התמונה ניתן לראות כי כל עוד לא הגענו לשורת היעד אנחנו ממשיכים לעלות בשורות. רק כאשר אנחנו באותה שורה עם צומת היעד, אנו מתחילים לנוע בעמודות.

**אלגוריתם ליצירת\ציור הסימולציה**

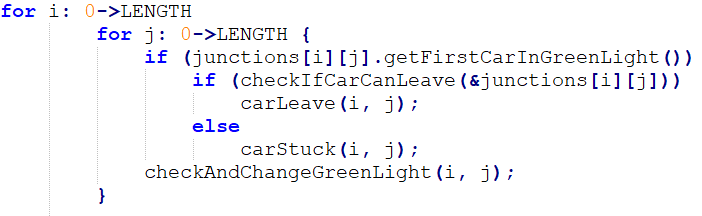


הסבר אלגוריתם לסימולציה:

המחלקה המנהלת (JunctionManager) אחראית גם על הציור (בעזרת הפונקציה drawAll) וגם על ההזזה של הרכבים (בעזרת הפונקציה moveAll).

הפונקציה drawAll עוברת על כל הצמתים ומציירת אותם (על ידי קריאה לפונקציה של ציור כבישים, רכבים וכו').

הפונקציה moveAll (פסאודו קוד):

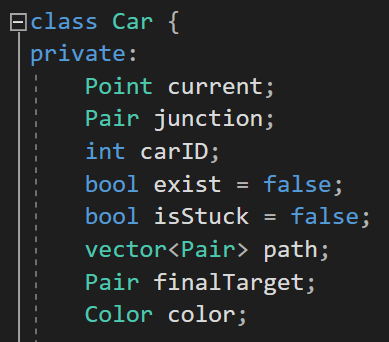


**נספח תיעוד קוד**

בנספח זה נתאר את המחלקות ואת הפונקציות העיקריות בקוד שלנו:

סימולציה:

מחלקה - Car (רכב):



תכונות:

**current** - the current location of the car (for drawing)

**junction** - the current junction of the car

**carID** - the ID of the car

**exist** - whether the car exists or not

**isStuck** - whether the car is stuck or not

**path** - a vector of pairs that represent the path

**finalTarget** - the final junction in the path (the final target for the car)

**color** - the color of the car

פונקציות חשובות:

**void Car::calcPath()**

calculates the path for the car, according to the first and last junctions in its path.

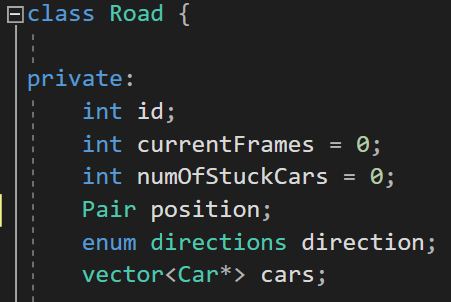
**bool Car::hasArrived()**

check whether or not the car has arrived

**void Car::drawCar()**

draws the car according to its current location

מחלקה - Road (כביש):



תכונות:

**id** - the id of the road

**currentFrames** - frames counter (for green light switching)

**numOfStuckCars** - the number of stuck cars for the road

**position** - the position in the matrix

**direction** - the direction of the road (one of four: UP, RIGHT, DOWN, LEFT)

**cars** - vector of pointers to cars that are currently on the road

פונקציות חשובות:

**bool Road::checkRoadAvailability(Car\* car)**

check whether or not a road can accept more cars (if it is not full)

**Car\* Road::removeFromTop()**

removes the first car in line and returns a pointer to it

**bool Road::addToEnd(Car\* car, bool initial)**

adds a car to the end of the road

**void Road::drawRoad()**

draws the road according to its direction

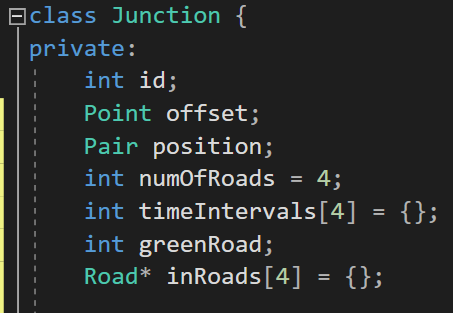
**void Road::drawCars()**

draws all the cars on the road

**Car\* Road::getFirstCar()**

returns the first car in the road

מחלקה - צומת (Junction):



תכונות:

**id** - the id of the junction

**offset** - offset for drawing

**position** - the position in the matrix

**numOfRoads** - number of roads for the junction. in our case, the number is 4.

**timeIntervals** - the green light time that is given to each light in the junction. this is an array of four ints, one for each light.

**greenRoad** - the number of the road that is currently green (0-4)

**inRoads** - an array of pointers to the roads of the junction

פונקציות חשובות:

**Car\* Junction::getFirstCarInGreenLight()**

returns the first car from the road that is currently green

**void Junction::setNextGreenRoad()**

changes the current green light road to the next road in line

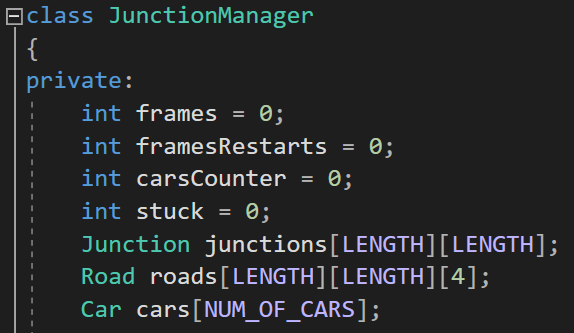
**void Junction::paintLights()**

paints the lights in the junction green or red based on the greenRoad variable.

**void Junction::drawJunction()**

iterates over the roads array and using the drawRoad function on each one of them. drawing all the roads means drawing the junction

מחלקה - JunctionManager (ניהול התוכנית):



תכונות:

**farmes** - frames counter

**framesRestarts** - number of frames counter restarts (used to make sure that the frames number isn’t becoming too high)

**carsCounter** - counts the cars that exist in the simulation

**stuck** - total number of stuck cars

**junctions** - a square matrix of junctions

**roads** - a 3D array that contains roads. Notice that it has the same size as the junctions matrix, but with four roads for each junction.

**cars** - an array of all cars from the simulation.

פונקציות חשובות:

**void JunctionManager::moveAll()**

contains the logic for moving all cars in green lights in each junction (if possible)

**void JunctionManager::drawAll()**

draws all junctions by their position on the matrix

**void JunctionManager::move(Junction\* junction)**

gets a junction and moves the first car in it

**bool JunctionManager::checkIfCarCanLeave(Junction\* current)**

gets a junction and check if the first car in the green light road can leave

**void JunctionManager::carStuck(int i, int j)**

gets the first car in green light from the junction in the i,j index in the junctions matrix and set it as stuck car (set the boolean and paint it red)

**void JunctionManager::carLeave(int i, int j)**

gets the first car in green light from the junction in the i,j index in the junctions matrix and moves it to the next junction in the path.

**bool JunctionManager::deadlockCheck()**

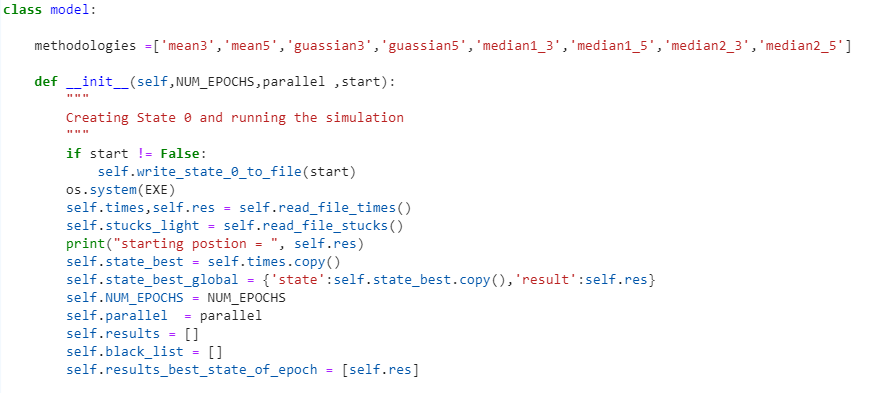
includes the logic for deadlock check

**bool JunctionManager::finish()**

contains the logic for finishing the program

אלגוריתם

בנספח זה נפרט על הפונקציות העיקריות בחלק של האלגוריתם:



The initial parameters of the class:

**times**(list):The time matrix of all the traffic lights in our city

**res**(int):The total number of stuck cars

**stucks\_light**(list)**:** The time matrix of all the stuck cars in our city

**state\_best**(list): The best time matrix of one iteration(epoch)

**state\_best\_global**(list): The best time matrix of all iterations(epochs)

**NUM\_EPOCHS**(int) - Number of iterations(epochs) for learning

**parallel**(boolean) - Boolean parameter whether learning is parallel or not parallel

**results**(list): List of all results of all restraints (for graphic use)

**black\_list**(list): List all the problematic states that were discovered during learning

**results\_best\_state\_of\_epoch**(list): List of results of the best number of stuck vehicles from each iteration

**methodologies**(list): List of names of the types of filters we have chosen to use in the project

פונקציות חשובות:

**fit():**

Fit our model - doing the learning, result:save the best state

**best\_state\_current(array)**

Get list of states and return the best state as a dictionary from the list

**target(state,results,stucks\_light,methodology):**

assign each thread its role

**parallel\_run():**

Run all threads parallely and collects all the results

**stuck\_times(times,stucks\_light,methodology):**

Creates noise for the combination between the state and the stuck file, and calculates this new times by matching filter

**change\_new(state,results ,stucks\_light, methodology):**

Running the simulation for each thread and save its result.

**non\_parallel():**

Run program and collects all the results non-parallely

# 24.1. מסמך SRD

1. הקדמה:

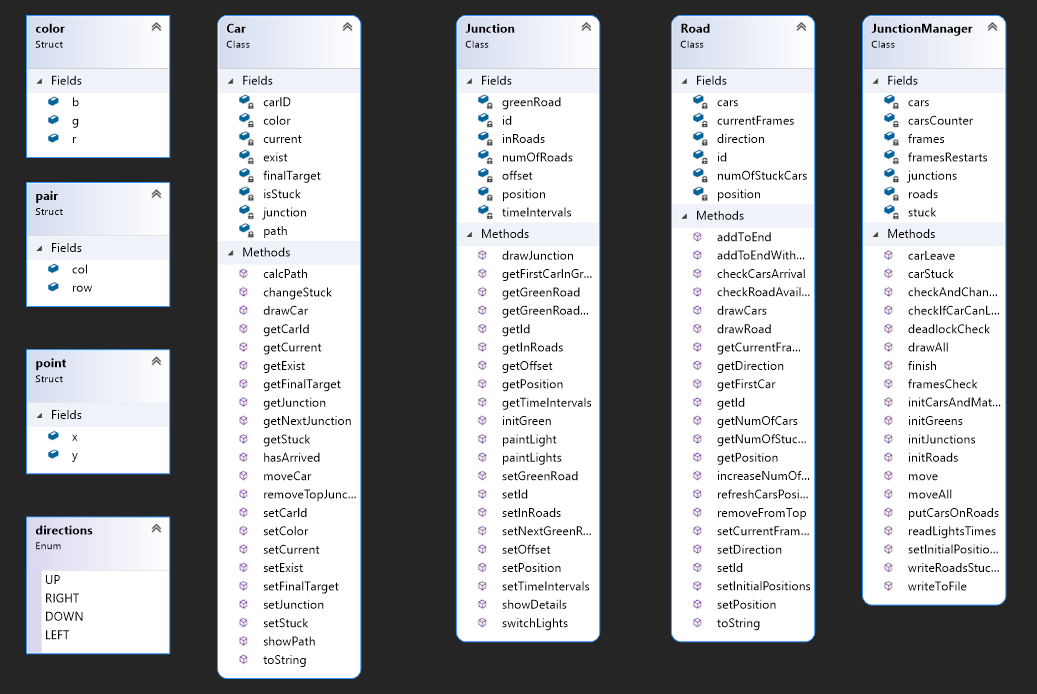
המערכת שלנו הינה מערכת רמזורים חכמים מבוססת למידת מכונה שמטרתה לצמצם את מספר התקיעות ברשת הצמתים.

1. תפיסת המודל:

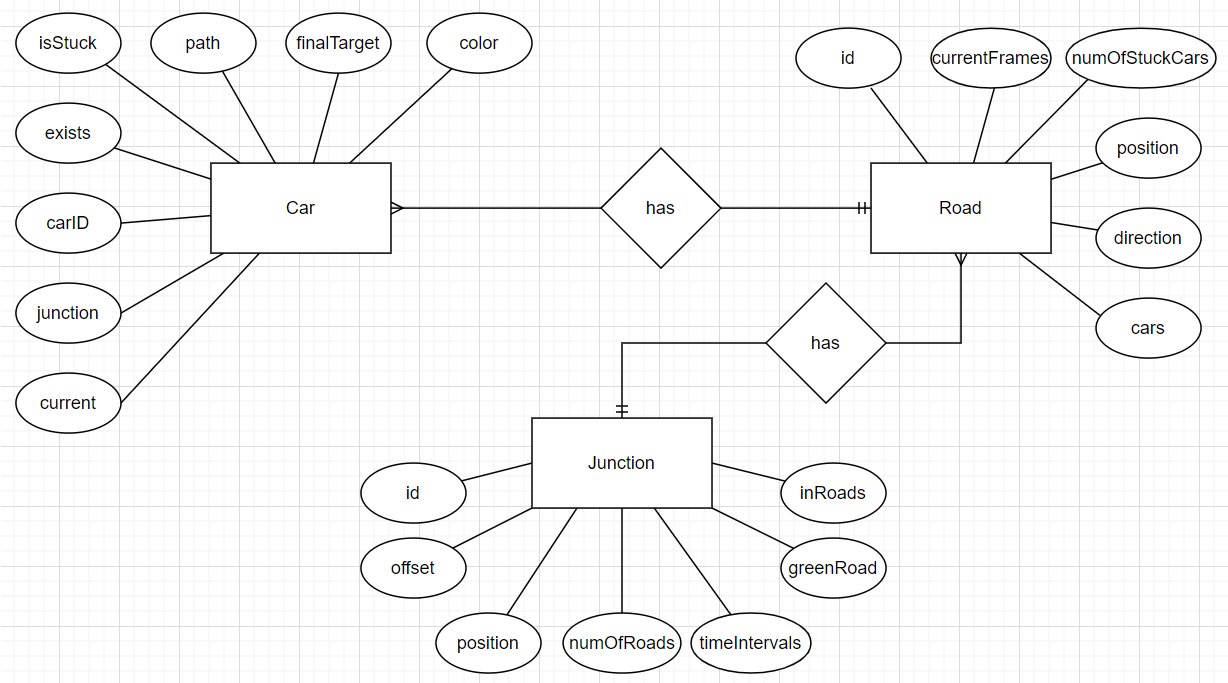
המערכת שלנו בנויה משלושה מודלים עיקריים:

* סימולציה - מאפשרת לראות את המצב בנוכחי של רשת הכבישים והמכוניות
* אלגוריתם מבוסס למידת מכונה - משפר את זמני הרמזורים על מנת למנוע עומסים
* מערכת מקשרת - קובץ טקסט שאליו נכתבים הנתונים לשימוש האלגוריתם.

תרשים UML (סימולציה):



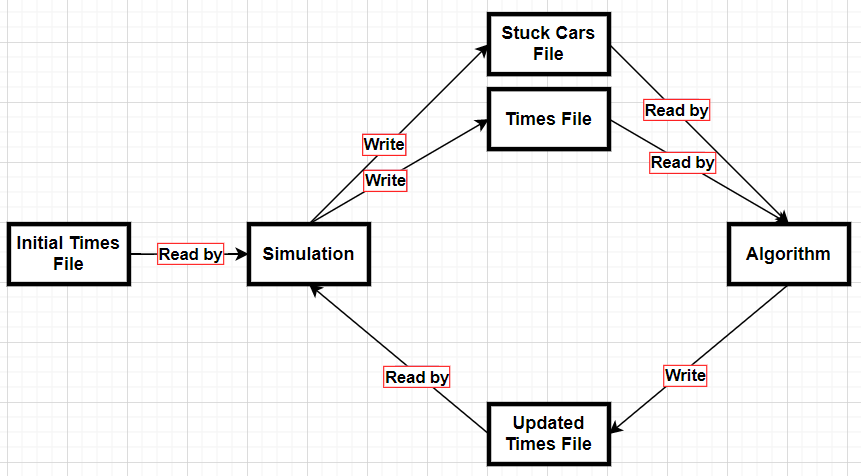
תרשים ERD: (סימולציה)



1. ארכיטקטורה:

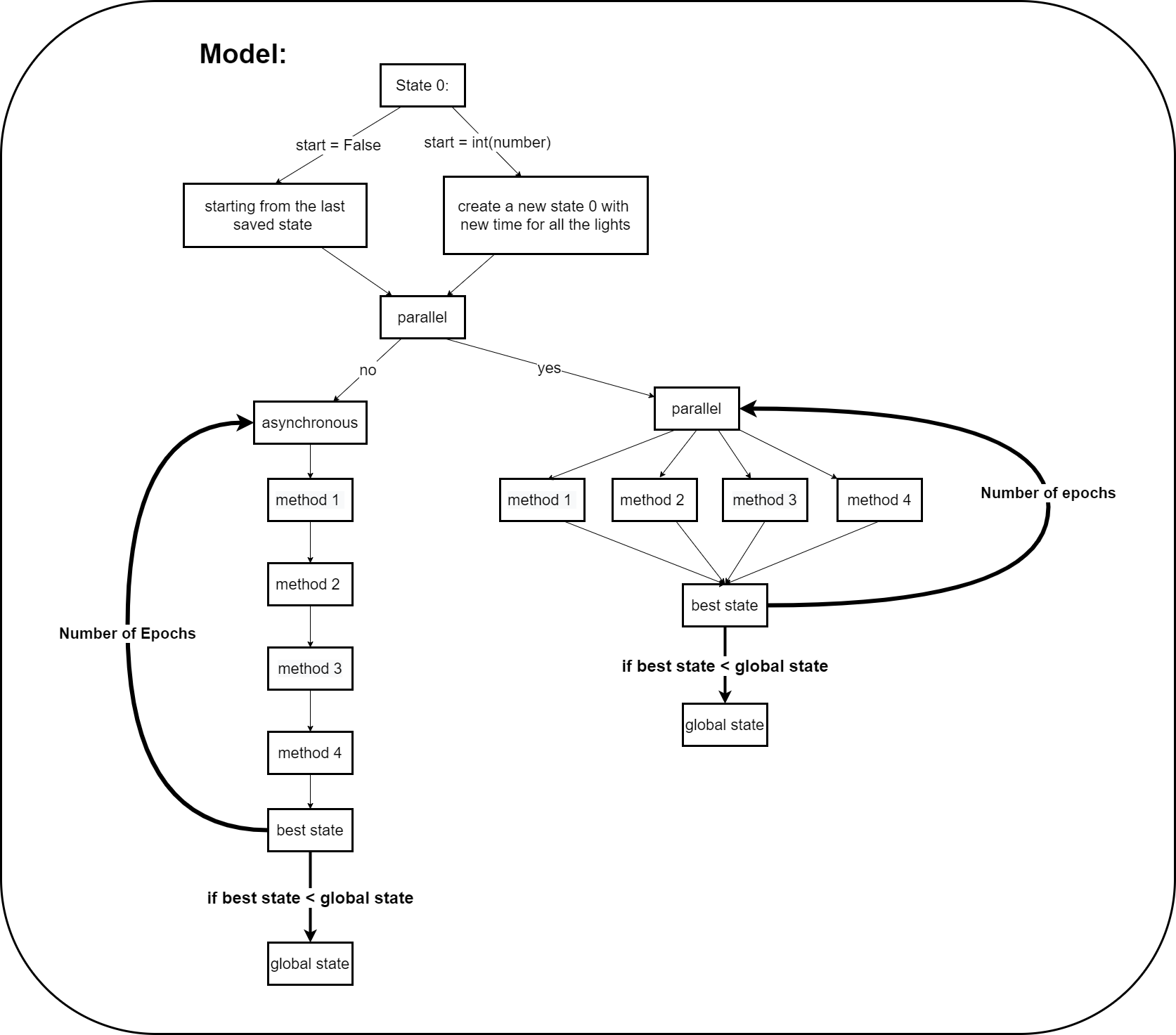
תרשים בלוקים של המערכת:

הסימולציה מקבלת קבצי טקסט עם זמנים לכל הרמזורים ומפיקה קובץ עם מספרי תקיעות (מספר תקיעות כללי בכל ההרצה וגם מספר תקיעות מפורט לכל רמזור). האלגוריתם מקבל את הקבצים הללו, פועל לשיפור הזמנים ומחזיר את קובץ הזמנים המשופר לסימולציה.



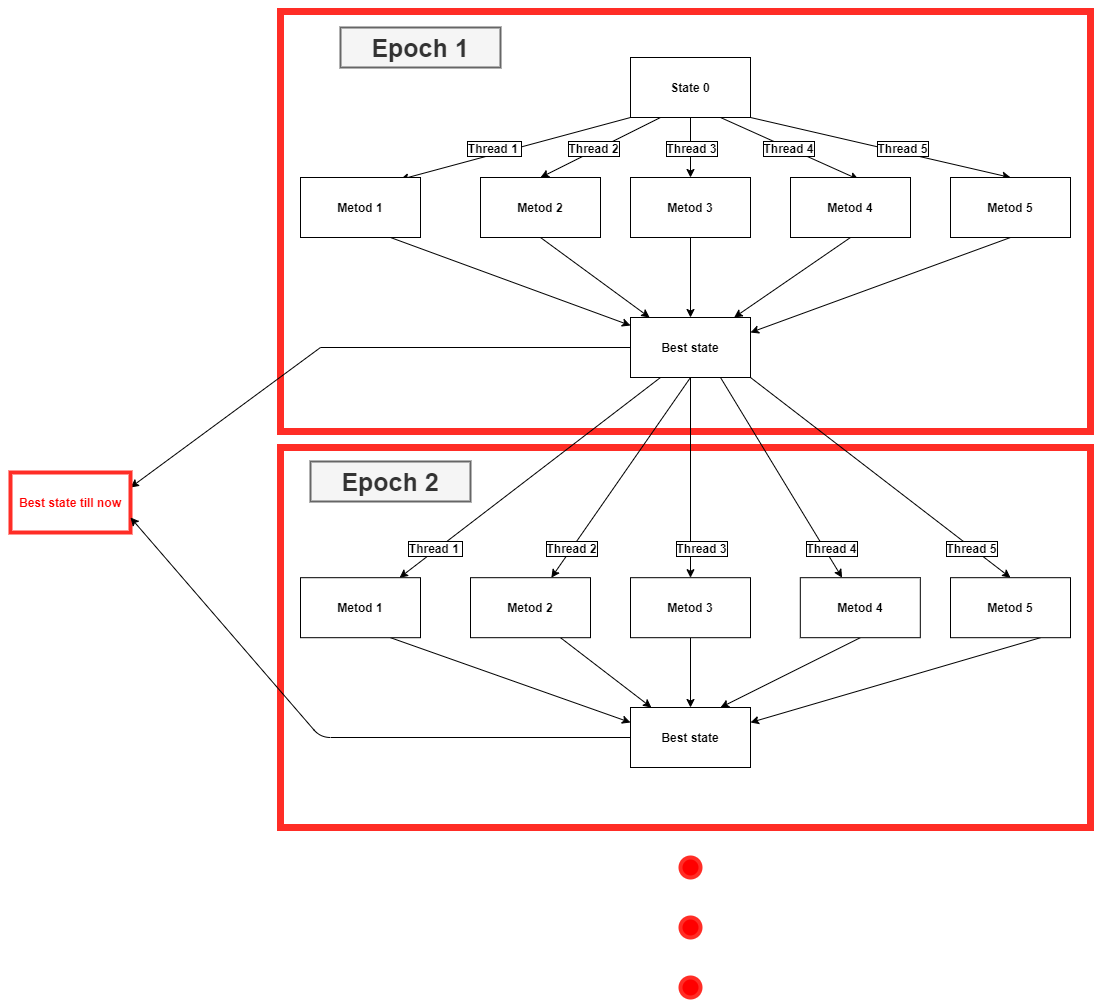
תרשים בלוקים של האלגוריתם:

הלאגוריתים מקבל את הזמנים ואת מספר התקיעות ופועל לשיפור הזמנים על ידי כמה שיטות ובוחר את התוצאת הטובה ביותר. ניתן לבחור בין עבודה מקבילית לעבודה סדרתית.



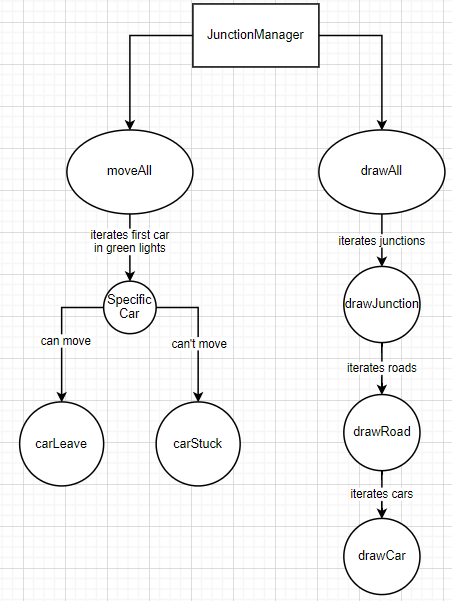
התמקדות בריצה מקבילית:

כל שיטה (method) מניבה מצב (state - קובץ זמנים חדש) המערכת מריצה כל מצב על הסימולציה ובוחרת את המצב הטוב ביותר (כלומר - זה שהניב הכי פחות תקיעות)



**תיאור הארכיטקטורה:**

סימולציה - הסימולציה מבוססת על תכנות מונחה עצמים. כפי שניתן לראות ב-ERD, לכל צומת יש מערך של כבישים ולכל כביש יש מערך של רכבים. בנוסף, לכל אובייקט יש פונקציה שמציירת את עצמו. יש רצה על פי קובץ זמנים כלשהו ומציגה את התנועה. כלומר, הסימולציה קוראת את נתוני הזמנים מקובץ הזמנים ומכניסה אותם למערך של זמנים (\*) של כל צומת. בעת ריצת הסימולציה התכנית סופרת את כמות המכוניות שנתקעו (\*\*) במהלך הסימולציה. בסוף ריצת הסימולציה, המערכת כותבת את התקיעות לקובץ הזמנים. בנוסף, בזמן ההרצה הסימולציה מעדכנת את כמות התקיעות בכל רמזור כך שבסוף ההרצה נוצר קובץ שבו נשמרו כל מספרי התקיעות עבור כל רמזור. מקרה חריג שעלול להתרחש: מצב נעילה (deadlock). הסימולציה תדע לזהות מקרה כזה ולפעול בהתאם: לכתוב את הערך 1- במספר התקיעות ולסיים את התכנית.



אלגוריתם - האלגוריתם מקבל את שני הקבצים (קובץ זמנים וקובץ תקיעות), קובץ התקיעות משמש את האלגוריתם בתור מפה עבור האזורים הבעייתים (שבהם מספר גדול של תקיעות). האלגוריתם מבצע חישוב מקדים עבור כל רמזור - - ויוצר מטריצת זמנים חדשה () אשר מייצגת את הרעש לשימוש האלגוריתם. האלגוריתם מקצה שמונה תהליכונים (threads) שונים, שכל אחד מהם אחראי על מיצוע בשיטת חישוב שונה. כל אחד מהתהליכונים יוצר קובץ זמנים חדש על פי המיצוע שלו ומריץ את קובץ הזמנים שלו על הסימולציה על מנת לקבל מספר תקיעות ולהציג את הסימולציה עם הזמנים שלו. כל אחד מהתהליכונים מחזיר מילון אשר בנוי משלושה מפתחות - אחד שומר את מטריצת הזמנים (state) שחושבה על ידו, השני שומר את מספר התקיעות שהתקבל מהרצת הזמנים שלו על הסימולציה שלו והשלישי שומר את קובץ התקיעות. כל תהליכון שומר את המילון שלו בתוך מערך (שנמצא ב-main) שמכיל את כל המילונים מכל התהליכונים ומסיים את תפקידו. נבחר מתוך המערך במטריצת הזמנים עם מספר התקיעות הכללי הקטן ביותר ונשמור אותה בתור קובץ הזמנים הטוב ביותר (\*\*\*) **באותה איטרציה**. בנוסף, אנו שומרים ומעדכנים בסוף כל איטרציה (\*\*\*\*) את מטריצת הזמנים הטובה ביותר של **התוכנית**. בסוף הרצת כל איטרציה, קובץ הזמנים החדש שמחולק לכל התהליכונים לקראת האיטרציה הבאה הינו הקובץ הטוב ביותר שנבחר מהאיטרציה הקודמת. כך, מובטח שינוי מאיטרציה לאיטרציה.

מקרה חריג שעלול להתרחש (באלגוריתם): הסימולציות של כל התהליכונים באיטרציה ספציפית מגיעות למצב נעילה (deadlock) - מצב בו אף רכב לא יכול להתקדם לצומת הבאה. האלגוריתם מזהה מצב זה (בעת קריאה של הערך 1- מקובץ הזמנים שנכתב על ידי הסימולציה), מסיים את התוכנית ומציג על הסימולציה את קובץ הזמנים הטוב ביותר עד כה.

מקרה חריג שעלול להתרחש (באלגוריתם): כאשר התהליכון הנבחר הוא תהליכון שלא ניתן לשפר אותו ויוצר חזרתיות של מצבים (אין שיפור ואנו נתקעים על אותו המצב), במצב זה המערכת תזהה את הבעיה ותכניס את המצבים הבעייתיים לתוך "רשימה שחורה" של מצבים שמהם אנחנו נמנע ובכך נפתרת הבעיה זו.

סיום התכנית - התכנית תסתיים באחד משני מקרים:

1. סיימנו את האיטרציה האחרונה והתקבל קובץ הזמנים הכי טוב שיכול להתקבל במסגרת האיטרציות שנתנו.
2. כל אחת מהשיטות הניבה קובץ זמנים אשר הכניס את הסימולציה למצב נעילה (deadlock). במקרה כזה ייבחר הקובץ הטוב ביותר האחרון שהצלחנו להגיע אליו עד כה.
3. במידה ונגיע למצב ללא תקיעות התכנית תסתיים ויישמר המצב.

\* מערך של זמנים - מערך בגודל 4 ששמור בכל צומת. במערך מצויינים זמן האור הירוק שניתן לכל רמזור בצומת. (מערך בגודל 4 4 רמזורים בכל צומת).

\*\* נתקעו - בזמן שניתן להן ירוק לא יכלו לעבור לכביש הבא במסלול בגלל שהוא מלא. מכוניות שנתקעו נצבעות בצבע אדום.

\*\*\* קובץ הזמנים הטוב ביותר - קובץ הזמנים שהניב את מספר התקיעות המינימלי.

\*\*\*\* איטרציה = epoch

1. פירוט דרישות

המערכת בנויה כך שבהתחלה הסימולציה רצה ללא שימוש באלגוריתם וכותבת נתונים לקובץ. האלגוריתם משפר את הנתונים שהתקבלו, משנה את הזמנים ויוצר קובץ זמנים חדש.

1. תרשימי זרימה

תרחיש:

כל הרכבים הגיעו ליעדם - כל הרכבים הגיעו ליעדם עם מספר תקיעות כלשהו. הסימולציה תעדכן את האלגוריתם בנתונים והאלגוריתם ינסה לשפר את מספר התקיעות. **מצב תקין**.

תרחיש:

פקק\עומס תנועה - נוצר פקק ברמזור משום שזרימה בכיוון אחד יותר עמוסה מהקיבולת של הרמזור. נחשב זמנים אחרים, תוך התחשבות בשאר הרמזורים ובעיקר בצמתים הסובבים את הצומת הבעייתית, וייתכן ונקבל זרימה טובה יותר ופחות תקיעות.

תרחיש:

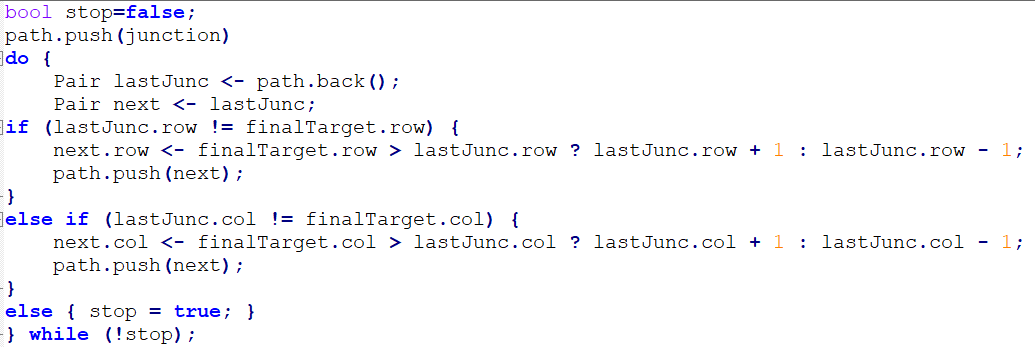
מכונית תקועה - כאשר מכונית רוצה לפנות לצומת הבאה והיא לא יכולה מכיוון שהכביש בקיבולת מלאה ולכן היא תחכה עד שהוא יתפנה.

תרחיש:

נעילה (deadlock) - כאשר כל הרכבים תקועים ולא יכולים לזוז. במצב זה הסימולציה תקועה ולא יכולה להתקדם. ישנה מערכת לזיהוי מצבי נעילה (deadlock) אשר יודעת לזהות את המצב, לסיים את התכנית ולעדכן את האלגוריתם בהתאם. **מצב לא תקין**.

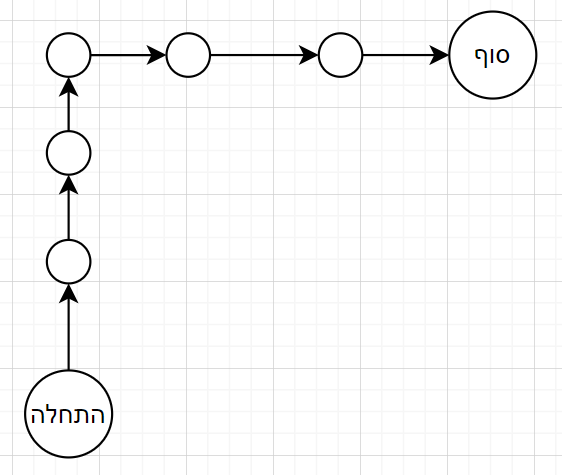
1. אלגוריתמים:

**אלגוריתם לחישוב מסלול:** (פסאודו קוד)



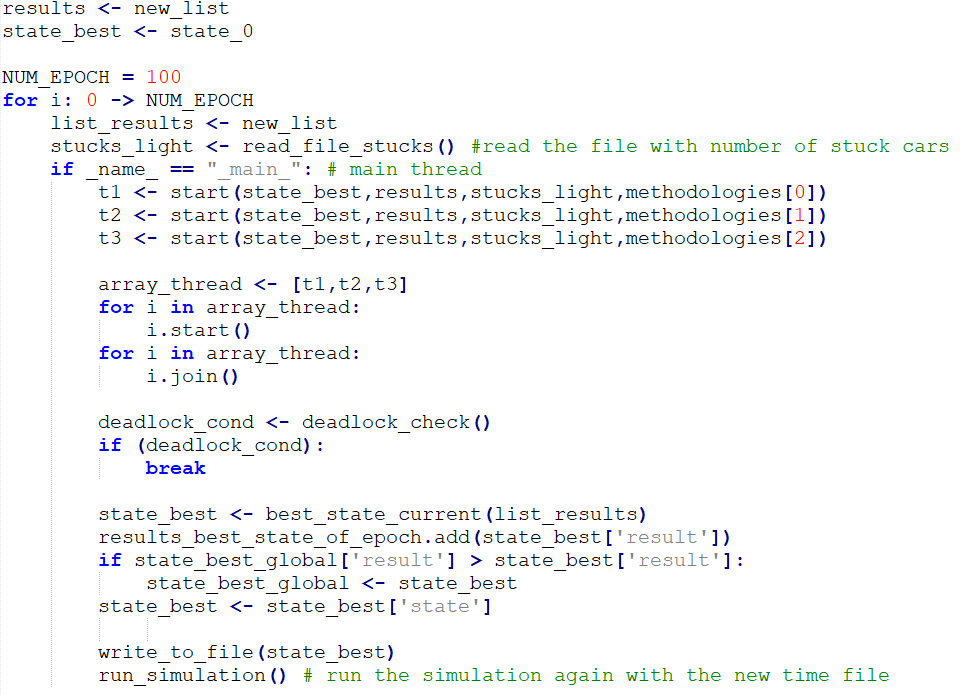
הסבר:

האלגוריתם מוסיף צמתים קודם על פי שורה (כלומר כל עוד לא הגענו לשורה של צומת היעד) ואז על פי עמודה (כלומר כל עוד לא הגענו לעמודה של צומת היעד).



על פי התמונה ניתן לראות כי כל עוד לא הגענו לשורת היעד אנחנו ממשיכים לעלות בשורות. רק כאשר אנחנו באותה שורה עם צומת היעד, אנו מתחילים לנוע בעמודות.

**אלגוריתם מבוסס למידת מכונה:** (פסאודו קוד עם דוגמא של שלוש מתודולוגיות. באלגוריתם המקורי יש יותר מתולוגיות ויותר תהליכונים (threads))



האלגוריתם מחלק את המשימות למספר התהליכונים (threads הנדרש). כל תהליכון מחשב על פי השיטה שנתנה לו ומחזיר את התוצאה. האלגוריתם עובר על התוצאות ובוחר את התוצאה הטובה יותר, שתהווה המצב ההתחלתי לאיטרציה הבאה. האלגוריתם בודק אם התוצאה הטובה של האיטרציה היא התוצאה הטובה ביותר בתכנית (global) ומעדכן בהתאם.

**אלגוריתם לציור הסימולציה:**

לכל אובייקט מסוג צומת ישנם:

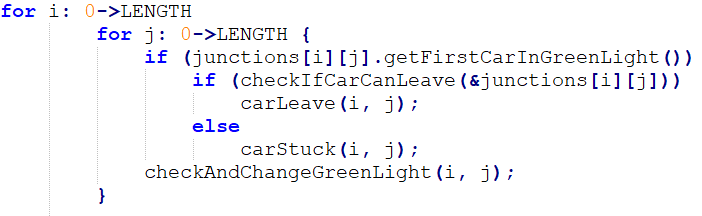
* מיקום על ציר x
* מיקום על ציר y
* ארבעה כבישים שנכנסים אליו
* ארבעה כבישים שיוצאים ממנו

לכל כביש יש פונקציה של ציור שמאפשרת לכביש לצייר את עצמו (drowRoad). בעזרת המיקומים של כל צומת והכבישים שלה, ובעזרת פונקצית glTranslated של Open GL ניתן לעבור על כל הצמתים שבסימולציה ולצייר את כולם תוך שימוש בתכנות מונחה עצמים.

המחלקה המנהלת (JunctionManager) אחראית גם על הציור (בעזרת הפונקציה drawAll) וגם על ההזזה של הרכבים (בעזרת הפונקציה moveAll).

הפונקציה drawAll עוברת על כל הצמתים ומציירת אותם (על ידי קריאה לפונקציה של ציור כבישים, רכבים וכו').

הפונקציה moveAll (פסאודו קוד):



הפונקציה בודקת האם המכונית הראשונה ברמזור הירוק יכולה לעזוב. אם כן, היא מזיזה אותה לצומת הבאה, אם לא היא מסמנת אותה כמכונית תקועה.

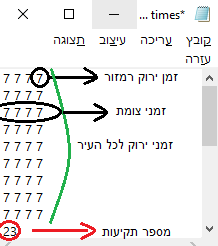
1. אבטחת מידע:

המערכת שלנו, כולל הסימולציה והאלגוריתם, אוטונומית ולא תלויה בהתערבות של גורם חיצוני כלשהו, עם או ללא שימוש ברשת. על כן, אין דרישות אבטחת מידע.

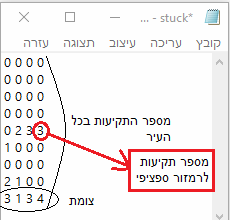
1. הערות:

הסבר על בסיס הנתונים:

קובץ זמנים (times.txt) - כל שורה בקובץ מייצגת צומת. מספר השורות - כמספר הצמתים. בכל שורה ארבע מספרים. כל מספר מייצג את זמן האור הירוק שקיבל כל רמזור בפריימים (לא בשניות). המספר בסוף הקובץ מייצג את כמות התקיעות הכוללת בהרצה.



קובץ תקיעות (stucks.txt) - כל שורה בקובץ מייצגת צומת. מספר השורות - כמספר הצמתים. בכל שורה ארבע מספרים. כל מספר מייצג את כמות התקיעות בכביש ובצומת המתאימים.



# 24.2. מסמך SDD

1. הקדמה:

ניתן לחלק את המערכת שלנו לשני מודולים עיקריים אשר מתקשרים בניהם: סימולציה ואלגוריתם מבוסס למידת מכונה.

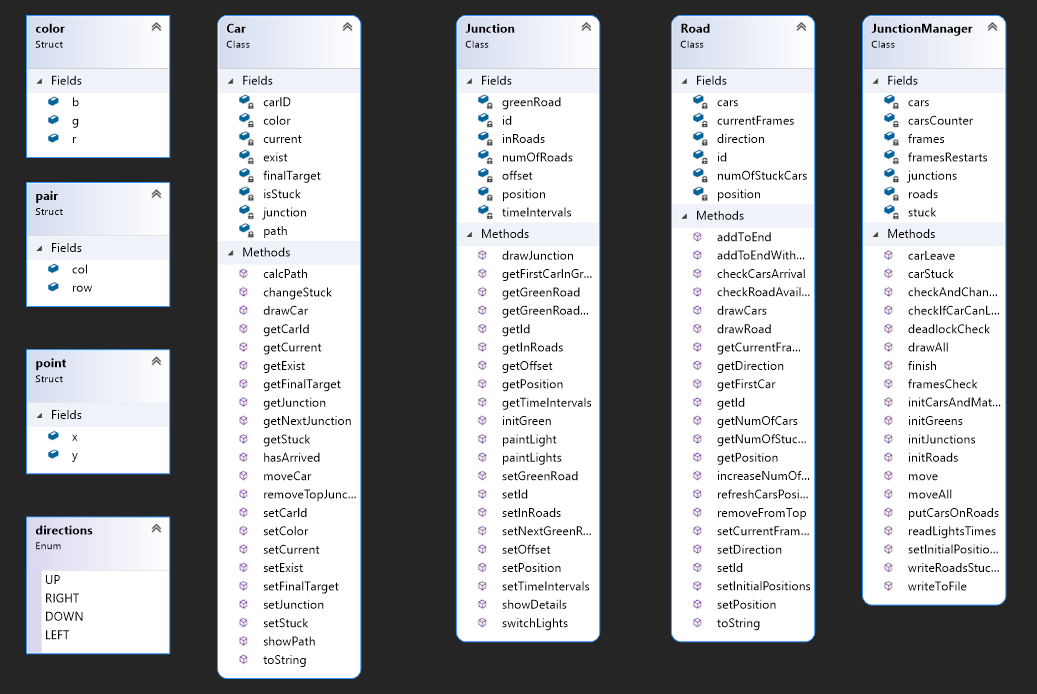
* סימולציה: מציגה את המערכת, הכבישים והמכוניות, אחראית על רישום הנתונים הרלוונטיים לקובץ והצגתם
* אלגוריתם מבוסס למידת מכונה: מקבל את הפרטים מהקובץ שנכתב על ידי הסימולציה ופועל להפחית את התקיעות על ידי שינוי הזמנים.

1. תפיסת המודל:

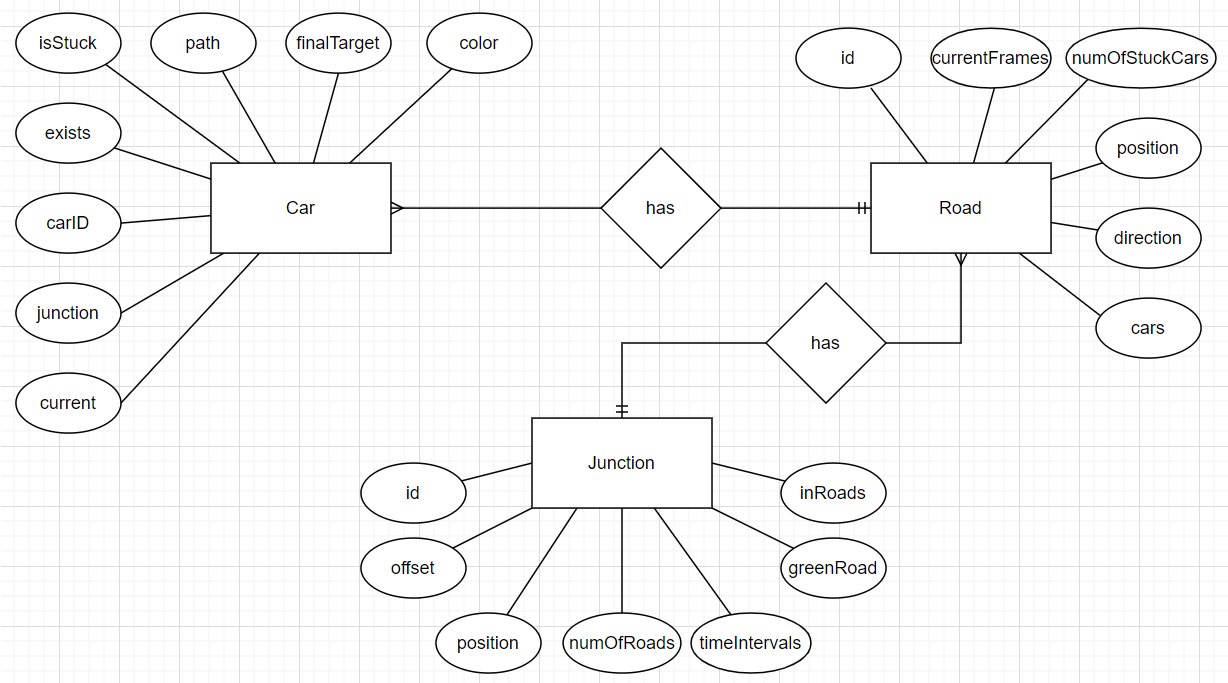
המערכת שלנו בנויה משני מודלים עיקריים:

* סימולציה - מאפשרת לראות את המצב בנוכחי של רשת הכבישים והמכוניות
* אלגוריתם - משנה את זמני זמני הרמזורים על מנת למנוע תקיעות
* מערכת מקשרת - קובץ טקסט שאליו נכתבים הנתונים לשימוש האלגוריתם

תרשים UML (סימולציה):

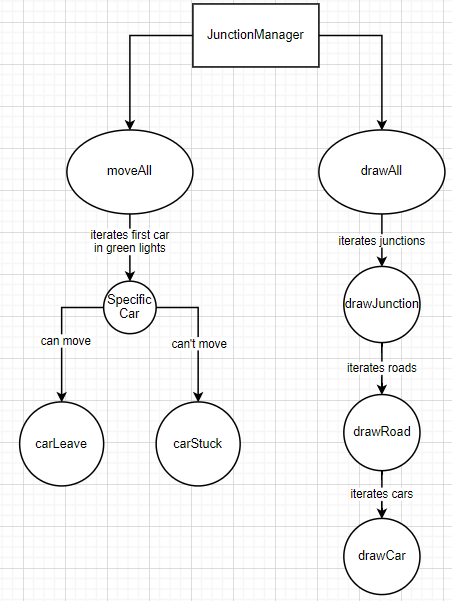


תרשים ERD: (מתייחס לסימולציה בלבד)



1. ארכיטקטורת המערכת:

סימולציה - הסימולציה מבוססת על תכנות מונחה עצמים. כפי שניתן לראות ב-ERD, לכל צומת יש מערך של כבישים ולכל כביש יש מערך של רכבים. בנוסף, לכל אובייקט יש פונקציה שמציירת את עצמו. יש רצה על פי קובץ זמנים כלשהו ומציגה את התנועה. כלומר, הסימולציה קוראת את נתוני הזמנים מקובץ הזמנים ומכניסה אותם למערך של זמנים (\*) של כל צומת. בעת ריצת הסימולציה התכנית סופרת את כמות המכוניות שנתקעו (\*\*) במהלך הסימולציה. בסוף ריצת הסימולציה, המערכת כותבת את התקיעות לקובץ הזמנים. בנוסף, בזמן ההרצה הסימולציה מעדכנת את כמות התקיעות בכל רמזור כך שבסוף ההרצה נוצר קובץ שבו נשמרו כל מספרי התקיעות עבור כל רמזור. מקרה חריג שעלול להתרחש: מצב נעילה (deadlock). הסימולציה תדע לזהות מקרה כזה ולפעול בהתאם: לכתוב את הערך 1- במספר התקיעות ולסיים את התכנית.



אלגוריתם - האלגוריתם מקבל את שני הקבצים (קובץ זמנים וקובץ תקיעות), קובץ התקיעות משמש את האלגוריתם בתור מפה עבור האזורים הבעייתים (שבהם מספר גדול של תקיעות). האלגוריתם מבצע חישוב מקדים עבור כל רמזור - - ויוצר מטריצת זמנים חדשה () אשר מייצגת את הרעש לשימוש האלגוריתם. האלגוריתם מקצה שמונה תהליכונים (threads) שונים, שכל אחד מהם אחראי על מיצוע בשיטת חישוב שונה. כל אחד מהתהליכונים יוצר קובץ זמנים חדש על פי המיצוע שלו ומריץ את קובץ הזמנים שלו על הסימולציה על מנת לקבל מספר תקיעות ולהציג את הסימולציה עם הזמנים שלו. כל אחד מהתהליכונים מחזיר מילון אשר בנוי משלושה מפתחות - אחד שומר את מטריצת הזמנים (state) שחושבה על ידו, השני שומר את מספר התקיעות שהתקבל מהרצת הזמנים שלו על הסימולציה שלו והשלישי שומר את קובץ התקיעות. כל תהליכון שומר את המילון שלו בתוך מערך (שנמצא ב-main) שמכיל את כל המילונים מכל התהליכונים ומסיים את תפקידו. נבחר מתוך המערך במטריצת הזמנים עם מספר התקיעות הכללי הקטן ביותר ונשמור אותה בתור קובץ הזמנים הטוב ביותר (\*\*\*) **באותה איטרציה**. בנוסף, אנו שומרים ומעדכנים בסוף כל איטרציה (\*\*\*\*) את מטריצת הזמנים הטובה ביותר של **התוכנית**. בסוף הרצת כל איטרציה, קובץ הזמנים החדש שמחולק לכל התהליכונים לקראת האיטרציה הבאה הינו הקובץ הטוב ביותר שנבחר מהאיטרציה הקודמת. כך, מובטח שינוי מאיטרציה לאיטרציה.

מקרה חריג שעלול להתרחש (באלגוריתם): הסימולציות של כל התהליכונים האיטרציה ספציפית מגיעות למצב נעילה (deadlock). האלגוריתם מזהה מצב זה (בעת קריאה של הערך 1- מקובץ הזמנים שנכתב על ידי הסימולציה), מסיים את התוכנית ומציג על הסימולציה את קובץ הזמנים הטוב ביותר עד כה.

מקרה חריג שעלול להתרחש (באלגוריתם): כאשר התהליכון הנבחר הוא תהליכון שלא ניתן לשפר אותו ויוצר חזרתיות של מצבים (אין שיפור ואנו נתקעים על אותו המצב), במצב זה המערכת תזהה את הבעיה ותכניס את המצבים הבעייתיים לתוך "רשימה שחורה" של מצבים שמהם אנחנו נמנע ובכך נפתרת הבעיה זו.

סיום התכנית - התכנית תסתיים באחד משני מקרים:

1. סיימנו את האיטרציה האחרונה והתקבל קובץ הזמנים הכי טוב שיכול להתקבל במסגרת האיטרציות שנתנו.
2. כל אחת מהשיטות הניבה קובץ זמנים אשר הכניס את הסימולציה למצב נעילה (deadlock). במקרה כזה ייבחר הקובץ הטוב ביותר האחרון שהצלחנו להגיע אליו עד כה.
3. במידה ונגיע למצב ללא תקיעות התכנית תסתיים ויישמר המצב.

\* מערך של זמנים - מערך בגודל 4 ששמור בכל צומת. במערך מצויינים זמן האור הירוק שניתן לכל רמזור בצומת. (מערך בגודל 4 4 רמזורים בכל צומת).

\*\* נתקעו - בזמן שניתן להן ירוק לא יכלו לעבור לכביש הבא במסלול בגלל שהוא מלא. מכוניות שנתקעו נצבעות בצבע אדום.

\*\*\* קובץ הזמנים הטוב ביותר - קובץ הזמנים שהניב את מספר התקיעות המינימאלי.

\*\*\*\* איטרציה = epoch

1. ממשקים:

* עדכון האלגוריתם:

מפרט הממשק - ממשק לקבלת וקריאת נתונים מהקובץ שנכתב על ידי הסימולציה.

מיקום הממשק - ממשק פנימי בין מודול הסימולציה למודול האלגוריתם.

עיתוי הפעלה - בכל סוף פעולת סימולציה ותחילת ה-epoch הבא.

* עדכון הסימולציה:

מפרט הממשק - עדכון קובץ הזמנים שהסימולציה פועלת לפיו. ממשק לקבלת נתונים מהקובץ עם הנתונים המעודכנים שנתכב על ידי האלגוריתם ויישומם על ידי הסימולציה.

מיקום הממשק - ממשק פנימי בין מודול האלגוריתם למודול הסימולציה.

עיתוי הפעלה - בכל סוף פעולה של אלגוריתם.

1. ממשק משתמש:

לא רלוונטי מכיוון שאין צורך להזין ידנית נתונים. מפורט בפרק של פיתוח עתידי בספר הפרויקט.

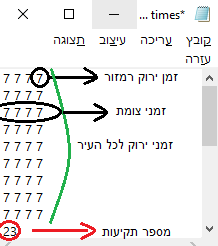
1. אבטחת מידע:

המערכת שלנו, כולל רשת הרמזורים ולמידת המכונה, אוטונומית ולא תלויה בהתערבות של גורם חיצוני כלשהו, עם או ללא שימוש ברשת. על כן, אין דרישות אבטחת מידע.

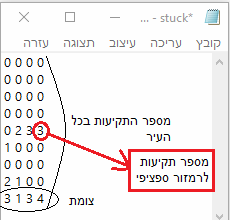
1. הערות:

הסבר על בסיס הנתונים:

קובץ זמנים (times.txt) - כל שורה בקובץ מייצגת צומת. מספר השורות - כמספר הצמתים. בכל שורה ארבע מספרים. כל מספר מייצג את זמן האור הירוק שקיבל כל רמזור בפריימים (לא בשניות). המספר בסוף הקובץ מייצג את כמות התקיעות הכוללת בהרצה.



קובץ תקיעות (stucks.txt) - כל שורה בקובץ מייצגת צומת. מספר השורות - כמספר הצמתים. בכל שורה ארבע מספרים. כל מספר מייצג את כמות התקיעות בכביש ובצומת המתאימים.



# 24.3. מסמך STD

הקדמה: הבדיקות המתוכננות הינן בדיקות לסימולציה, לאלגוריתם ולקישור בניהם. הבדיקות תתבצענה לכל תת מערכת בנפרד, ולאחר מכן גם לשילוב שלהן. בטבלאות הבאות מפורטות הבדיקות לכל תת מערכת וכן בדיקות קלט ופלט שמהוות את הקשר בין המערכות.

בדיקות עבור **הסימולציה**:

|  |  |  |
| --- | --- | --- |
| **בדיקה** | **תקין** | **לא תקין** |
| **קלט ראשוני** | מצב שמניב מספר תקיעות חיובי כלשהו | קלט שיוביל למצב נעילה (deadlock) ונקבל 1-. |
| **קלט** | נקלט בצורה טובה ונכנס למשתנים הנכונים | לא נקלט בצורה טובה. |
| **תפקוד של רכב אחד (בדיקת אמינות)** | תפקוד של רכב אחד כולל קבלת נתונים ומסלול, אמינות של האלגוריתם לחישוב מסלול | אי-תפקוד של רכב או קליטת נתונים לא תקינה. |
| **תפקוד צומת אחת** | תפקוד מלא של הצומת, מבחינת משתנים ומבחינה ויזואלית | הצומת לא מוצגת כראוי או לא מתפקדת כראוי מבחינת תוכנה. |
| **תפקוד מכלול של צמתים והשילוב בניהן** | צומת מתפקדת ומתקשרת עם שכניה | אי-תפקוד של צמתים כמערכת ו\או חוסר תקשורת בין צמתים. |
| **טיפול במצב של נעילה (deadlock)** | המערכת מבינה שיש נעילה ופועלת בהתאם | המערכת לא מזהה נעילה והתכנית נכנסת ללולאה אינסופית, או שהמערכת מזהה נעילה ולא פועלת בהתאם. |
| **פלט (הצגה)** | הצגה ברורה של תוצאות הפלט על פני הסימולציה. | אי הצגה או הצגה לקוייה של הסימולציה. |
| **פלט (העברה לאלגוריתם למידת מכונה)** | העברת הפלת בצורה תקינה לאלגוריתם למידת המכונה | אי העברת קלט בצורה תקינה. |

בדיקות קצה:

|  |  |
| --- | --- |
| **בדיקה** | **תוצאה** |
| הזרמה של 0 רכבים | עיר ריקה |
| הזרמה של 10000000 רכבים | העיר תהיה תקועה מלכתחילה והסימולציה תכנס למצב נעילה |

בדיקות עבור **האלגוריתם**:

|  |  |  |
| --- | --- | --- |
| **בדיקה** | **תקין** | **לא תקין** |
| **קלט מהסימולציה** | נקלט בצורה טובה | לא נקלט בצורה טובה |
| **יצירת תהליכונים (threads)** | נוצר תהליכון (thread) | לא נוצר תהליכון. |
| **חלוקת משימה לכל תהליכון** | כל תהליכון קיבל את המשימה שלו | תהליכונים שלא קבלו משימה או קבלו משימה שלא שלהם. |
| **ביצוע המשימה עבור כל תהליכון** | כל תהליכון מבצע את משימתו כמתבקש. | לא התבצעה המשימה באחד מהתהליכונים לפחות. |
| **הרצת הסימולציה ובמקביל** | הסימולציה מורצת בצורה תקינה (כולל התייחסות למקרים של נעילה [deadlock]) | הסימולציה לא עובדת בלפחות אחד מהתהליכונים |
| **עדכון נתונים תוך כדי פעילות** | עדכון נתונים בצורה תקנית (כולל תהחשבות בתהליכונים שלא אמורים להחזיר תוצאה) | אין עדכון נתונים או עדכון נתונים בצורה לא תקנית |
| **ביצוע האלגוריתם** | האלגוריתם עובד על הקלט באופן תקין | האלגוריתם לא התבצע |
| **פלט** | הפלט נרשם בצורה תקנית לקובץ האחרון | אי רישום או רישום לא תקני לקובץ |

בדיקות קישור בין האלגוריתם לבין הסימולציה:

|  |  |  |
| --- | --- | --- |
| **בדיקה** | **תקין** | **לא תקין** |
| **העברת מידע מהסימולציה לאלגוריתם** | המידע הועבר בצורה טובה ונקלט על ידי האלגוריתם | המידע לא הועבר או לא נקלט על ידי האלגוריתם |
| **העברת מידע מהאלגוריתם לסימולציה** | המידע הועבר בצורה טובה ונקלט על ידי הסימולציה | המידע לא הועבר או לא נקלט על ידי הסימולציה |

פירוט תרחישים:

בדיקות סימולציה:

|  |  |  |  |
| --- | --- | --- | --- |
| פונקציה\תרחיש | תסריט בדיקה | תקין | לא תקין |
| **קלט** | 1. **הרשאות**:   א. קריאה מקובץ שיש לנו הרשאה אליו.  ב. קריאה מקובץ שאין לנו הרשאה אליו.   1. **קובץ קיים**:   א. קריאה מקובץ קיים  ב. קריאה מקובץ לא קיים   1. **מבנה**:   א. קובץ במבנה הנכון  ב. קובץ במבנה הלא נכון   1. **תקינות הקלט (מספרים חיוביים ושלמים בלבד)** 2. **בדיקת משתנים** | א. תבוצע קריאה  ב. תוצג הודעת שגיאה ולא תבוצע קריאה.  א. תבוצע קריאה  ב. תוצג הודעת שגיאה.  א. תבוצע קריאה כראוי  ב. לא יקרא הקובץ ותוצג הודעת שגיאה.   1. הקריאה תמשיך כנדרש 2. הנתונים מהקובץ נשמרו כראוי במשתנים הנכונים | א. לא תבוצע קריאה.  ב. לא תוצג הודעת שגיאה  א. לא תבוצע קריאה  ב. לא תוצג הודעת שגיאה.  א. לא תבוצע קריאה  ב. ייקרא הקובץ במבנה לא תקין.   1. הסימולציה תתקע (הרכבים יתקעו) 2. לפחות אחד המשתנים לא יקבל ערך נכון. |
| **תפקוד של רכב אחד (בדיקת אמינות)** | 1. נתינת יותר מצומת התחלתית אחת לרכב 2. בדיקת צומת התחלה ויעד. 3. בדיקת מסלול (אמינות) 4. בדיקת הצמתים במסלול | 1. הצומת שנתנה אחרונה תקבע. 2. נתנו צמתים **שונים** בגבולות העיר 3. בהנתן אותם צמתי התחלה ויעד, המסלול נשאר זהה בכל הרצה. 4. הצמתים נמצאים על הדרך בין צומת ההתחלה לצומת היעד. | 1. הצומת שתקבע היא לא הצומת האחרונה שנתנה. 2. נתנה אותה צומת להתחלה וליעד או צמתים מחוץ לגבולות. 3. בהנתן אותם צמתי התחלה ויעד, המסלול לא נשאר זהה בכל הרצה. 4. הצמתים לא נמצאים על הדרך בין צומת ההתחלה לצומת היעד. |
| **תפקוד צומת אחת** | 1. הכנסת יותר מרכב אחד לאותו כביש. 2. תזוזת רכבים בזמן. 3. שינוי זמני הרמזורים. | 1. הרכב ייכנס רק אם יש מקום 2. הרכבים זזים כשניתן רמזור ירוק. 3. הרמזורים משתנים בהתאם לזמנים שניתנו. | 1. הרכב יכנס גם אם אין מקום ויגרום לקריסה של הסימולציה.  2. הרכבים זזים כשלא ניתן רמזור ירוק, או לא זזים בכלל.  3. הרמזורים לא משתנים בהתאם לזמנים שנתנו או לא משתנים בכלל. |
| **תפקוד מכלול של צמתים והשילוב בניהן** | 1. העברת רכבים לצומת הבאה. 2. הכרת צמתים שכנים. | 1. הרכבים עוברים לצומת הבא במסלול 2. צומת מכירה את הצמתים השכנים שלה | 1. הרכבים עוברים לצומת שאינה הבאה במסלול, או לא עוברים בכלל. 2. הצומת לא מכירה את שכנותיה. |
| **טיפול במצב של נעילה (deadlock)** | 1. זיהוי מצב deadlock 2. פעולה לאחר זיהוי המצב | 1. המערכת זיהתה שנכנסנו למצב נעילה. 2. המערכת פעלה בצורה המתבקשת בעת זיהוי המצב | 1. לא זוהה מצב deadlock 2. זוהה מצב deadlock אך המערכת לא פעלה בהתאם. |
| **פלט (כתיבה לקובץ)** | 1. **הרשאות**:   א. כתיבה לקובץ שיש לנו הרשאה אליו.  ב. כתיבה לקובץ שאין לנו הרשאה אליו.   1. **קובץ קיים**:   א. כתיבה לקובץ קיים.  ב. כתיבה לקובץ לא קיים. | 1.  א. תבוצע כתיבה.  ב. תוצג הודעת שגיאה ולא תבוצע כתיבה.  2.  א. תבוצע כתיבה.  ב. תוצג הודעת שגיאה. | 1.  א. לא תבוצע כתיבה.  ב. לא תוצג הודעת שגיאה  2.  א. לא תבוצע כתיבה.  ב. לא תוצג הודעת שגיאה. |

בדיקות אלגוריתם למידת מכונה:

|  |  |  |  |
| --- | --- | --- | --- |
| **פונקציה\תרחיש** | **תסריט בדיקה** | **תקין** | **לא תקין** |
| **קלט מהסימולציה** | 1. **הרשאות**:   א. קריאה מקובץ שיש לנו הרשאה אליו.  ב. קריאה מקובץ שאין לנו הרשאה אליו.   1. **קובץ קיים**:   א. קריאה מקובץ קיים  ב. קריאה מקובץ לא קיים   1. **מבנה**:   א. קובץ במבנה הנכון  ב. קובץ במבנה הלא נכון   1. **תקינות הקלט (נתינת מספר תקיעות בקובץ וקבלת קובץ תקיעות מפורט תקין)** 2. **בדיקת משתנים** | א. תבוצע קריאה  ב. תוצג הודעת שגיאה ולא תבוצע קריאה.  א. תבוצע קריאה  ב. תוצג הודעת שגיאה.  א. תבוצע קריאה כראוי  ב. לא יקרא הקובץ ותוצג הודעת שגיאה.   1. הקריאה תתבצע 2. הנתונים מהקובץ נשמרו כראוי במשתנים הנכונים | א. לא תבוצע קריאה.  ב. לא תוצג הודעת שגיאה  א. לא תבוצע קריאה  ב. לא תוצג הודעת שגיאה.  א. לא תבוצע קריאה  ב. ייקרא הקובץ במבנה לא תקין.   1. האלגוריתם יציג הודעת שגיאה. 2. לפחות אחד המשתנים לא יקבל ערך נכון. |
| **תהליכונים** | 1. יצירת תהליכונים בכמות הנדרשת. 2. חלוקת משימה לכל תהליכון. 3. ביצוע המשימה עבור כל תהליכון. 4. ביצוע המשימות במקביל. 5. סיום התהליכונים. | 1. נוצרה כמות תהליכונים כנדרש. 2. כל תהליכון קיבל את המשימה שלו. 3. כל תהליכון מבצע את המשימה שלו. 4. מתקיימת מקביליות (כל המשימות מבוצעות במקביל). 5. כל התהליכונים מסיימים את פעולתם כראוי ורושמים את תוצאתם. | 1. נוצרה כמות תהליכונים שונה מהנדרש או לא נוצרו בכלל. 2. לפחות תהליכון אחד שלא קיבל משימה. 3. לפחות תהליכון אחד לא מבצע את משימתו. 4. לא מתקיימת מקביליות. 5. לפחות אחד מהתהליכונים לא מסיים כראוי או לא רושם את התוצאה. |
| **הרצת הסימולציה ועדכון נתונים תוך כדי פעילות** | 1. הרצת הסימולציה על ידי כל תהליכון 2. עדכון נתונים בסוף איטרציה | 1. הסימולציה רצה כמתבקש. 2. בסוף האיטרציה כל הנתונים הרלוונטיים מתעדכנים. | 1. הסימולציה לא רצה או רצה באופן לא תקין 2. נתון אחד או יותר לא התעדכן. |
| ביצוע האלגוריתם לשיפור התוצאות | 1. הרצת האלגוריתם על פלט שלא ניתן לשיפור 2. הרצת האלגוריתם על פלט שניתן לשיפור | 1. לא יתקבל שיפור 2. יתקבל שיפור | 1. האלגוריתם לא יעבוד. 2. לא יתקבל שיפור. |
| פלט | 1. **הרשאות**:   א. כתיבה לקובץ שיש לנו הרשאה אליו.  ב. כתיבה לקובץ שאין לנו הרשאה אליו.   1. **קובץ קיים**:   א. כתיבה לקובץ קיים.  ב. כתיבה לקובץ לא קיים. | 1.  א. תבוצע כתיבה.  ב. תוצג הודעת שגיאה ולא תבוצע כתיבה.  2.  א. תבוצע כתיבה.  ב. תוצג הודעת שגיאה. | 1.  א. לא תבוצע כתיבה.  ב. לא תוצג הודעת שגיאה  2.  א. לא תבוצע כתיבה.  ב. לא תוצג הודעת שגיאה. |

בדיקות קישור בין האלגוריתם לבין הסימולציה:

|  |  |  |  |
| --- | --- | --- | --- |
| פונקציה\תרחיש | תסריט בדיקה | תקין | לא תקין |
| העברת מידע מהסימולציה לאלגוריתם | 1. כתיבה של הסימולציה לקובץ הזמנים (times) 2. כתיבה של הסימולציה לקובץ התקיעות (stucks) 3. העברת הקבצים | 1. כתיבה תקינה לקובץ הזמנים 2. כתיבה תקינה של קובץ התקיעות 3. העברו הקבצים לאלגוריתם | 1. כתיבה לא תקינה לקובץ הזמנים (לא נכתב או נכתב מידע שגוי) 2. כתיבה לא תקינה לקובץ התקיעות (לא נכתב או נכתב מידע שגוי) 3. הקבצים לא הועברו או לא נקלטו כראוי באלגוריתם |
| העברת מידע מהאלגוריתם לסימולציה | 1. כתיבה של קובץ הזמנים החדש שנוצר על ידי האלגוריתם 2. העברת הקובץ לסימולציה | 1. כתיבה תקינה של הקובץ 2. הועבר בצורה תקינה ונקלט על ידי הסימולציה | 1. כתיבה לא תקינה של הקובץ (לא נכתב או נכתב מידע שגוי) 2. לא הועבר בצורה תקינה או לא נקלט על ידי הסימולציה |

לוח זמנים לבדיקות:

