

האוניברסיטה הפתוחה

המחלקה למתמטיקה ומדעי המחשב

תוכנה לחיפוש יעיל של אובייקטים בתחומים גאומטריים

פרויקט מתקדם במדעי המחשב (22997)

מגיש: איגור דורפמן

ת"ז: 306329863

מנחה: ד"ר ג'ק וינשטין

סמסטר 2016 ב

תוכן עניינים

4	1. תקציר.....
5	2. מבוא.....
5	2.1 עצי תחומים (Range Trees).....
7	2.2 עצי חיפוש עם עדיפויות.....
8	2.2.1 עץ חיפוש עם עדיפויות מסוג RADIX (RPST).....
9	2.3 עץ תחומים עם עדיפויות (Range Priority Tree).....
11	3. דרישות המערכת.....
12	3.1 דרישות פונקציונליות.....
13	3.2 דרישות לא פונקציונליות.....
13	3.3 מגבלות והנחות המערכת.....
14	4. עיצוב מערכת התוכנה.....
14	4.1 דיאגרמת בלוקים של המערכת.....
14	4.1.1 תיאור רכיבים עיקריים.....
15	4.2 סביבת פיתוח.....
16	4.3 עץ חיפוש דו-מימדי.....
16	4.3.1 Radix Priority Search Tree עץ.....
19	4.3.2 עץ תחומים.....
22	4.4 אלגוריתם חיפוש נקודות בתחום שאילתה מלבני.....
25	5. ממשק משתמש.....
25	5.1 סימולציית מסלול טיסה.....
27	5.2 מסך ראשי.....
27	5.2.1 התפריטים במסך הראשי.....
29	5.2.2 פונקציות של המסך הראשי.....
30	5.3 מסך תצורה (קונפיגורציה).....
31	5.4 מסך מרחקים.....
35	6. בדיקות המערכת.....
35	6.1 בדיקות יחידה.....
40	6.2 בדיקות אינטגרציה.....

41	7. מקרי בוחן
44	8. סיכום והצעות לפיתוחים עתידיים
45	9. ביבליוגרפיה
46	10. נספחים
46	10.1 נספח א : רשימת איורים
46	10.2 נספח ב : קוד פרויקט
46	10.3 נספח ג : התקנת המערכת
47	10.4 נספח ד : הצעת פרויקט

1. תקציר

בשנים האחרונות חיפושים גאוגרפיים הפכו לנפוצים עד מאוד בחיי היומיום. דוגמה בסיסית לכך היא שהרבה מכוניות מצוידות היום בצגים המציגים את הסביבה הגאוגרפית של מיקום המכונית כגון תחנות דלק קרובות, מצלמות מהירות, ערים בסביבה ועוד. דוגמאות נוספות לכך הן תוכנות כמו סימולטורי טיסה אשר צריכות להציג לטייס את האובייקטים שנמצאים בטווח הראיה שלו ועוד.

כמות המידע הגאוגרפי הכולל היא עצומה, מה שמעניין את המשתמש הוא הצגה של חלק קטן מתוך כל המידע הגאוגרפי, אותו חלק הנמצא בסביבת המכונית, המטוס וכו'. כאשר עוברים מחלון אחד למשנהו צריכה התצוגה להתחלף באובייקטים הנמצאים בחלון החדש. בעיה זו נקראת בעיית windowing [1] (מכיוון שיש להציג רק את האובייקטים הנמצאים בתוך ה"חלון" סביב הגוף הנע).

מכיוון שכמויות המידע הגאוגרפי הכולל הן גדולות, יש צורך בחיפוש יעיל של האובייקטים אותם מעוניינים למצוא (האובייקטים הנמצאים בחלון הנוכחי בלבד).

במסגרת פרויקט מתקדם במדעי המחשב פותחה חבילת תוכנה המתמודדת עם בעיית ה-windowing. לצורך חיפוש יעיל של נקודות במישור של האובייקטים הנמצאים בתוך החלון הרלוונטי נעשה שימוש במבני נתונים "דו-מימדיים", עץ תחומים (range tree) ([1],[3],[7]) כעץ חיפוש ראשי ועץ חיפוש עם עדיפויות (priority search tree) ([1],[2],[4],[5],[6]) כעץ משני. כמו-כן נעשה שימוש באלגוריתמי החיפוש המתאימים למבנים אלה ([1],[2],[7]).

התוכנה מומשה בעולם התעופה כאשר ההנחה היא שהגוף הנע הינו מטוס והמידע הגאוגרפי אותו מעוניינים להציג הוא שדות התעופה (המוצגים כנקודות במישור) הנמצאים בסביבת המטוס. התוכנה מציגה מפה דו-מימדית של איזור הטיסה אשר מחולקת לחלונות מלבניים ומציגה תוך שימוש באלגוריתמים ומבני הנתונים היעילים שתוארו מעלה את שדות התעופה הנמצאים בחלון (או בחלונות) שבהן נמצא המטוס באותו רגע. כאשר המטוס עובר מחלון אחד למשנהו מתחלפת התצוגה בשדות התעופה הנמצאים בחלון החדש.

לתוכנה זו פוטנציאל רב לשימוש הן בכלי טיס אמיתיים והן במקומות אחרים כגון מכוניות שעבורן יכול להיות עניין בתחנות דלק קרובות (בעיקר באיזורים שוממים) ועוד. בכלי טיס מערכת תוכנה כזו יכולה להיות קריטית במקרי חירום של אובדן קשר וצורך בנחיתה מיידית. על מנת לשלב את התוכנה בפלטפורמות אמיתיות יש צורך להוסיף ממשק עם מערכת איכון מסוימת (כגון GPS) אשר יספק את המיקום האמיתי של הפלטפורמה ושל המטרות (שדות תעופה, תחנות דלק וכו') בכל רגע נתון.

2. מבוא

עקב הגידול המתמיד במערכות העושות שימוש בחיפוש גאוגרפיים שונים ועקב הגידול המתמיד בכמות המידע הגאוגרפי, הוצעו בעשורים האחרונים מבני נתונים המאפשרים למצוא את האובייקטים המעניינים ברגע נתון מתוך כל מסד הנתונים בצורה יעילה.

מבני הנתונים העיקריים שבהם משתמשים הם עצי חיפוש מאוזנים המחזיקים את כלל המידע הגאוגרפי ([1],[3],[5],[6]). במבנים אלה ניתן לבצע חיפוש בצורה יעילה.

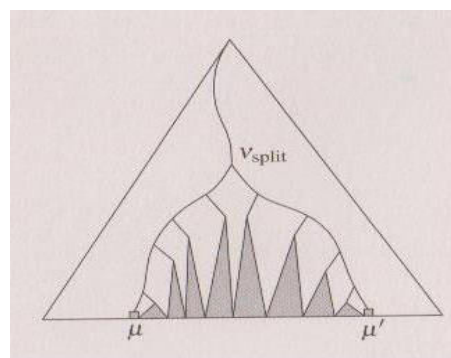
בפרויקט זה עושים שימוש בשני מבני נתונים כאלה – עצי תחומים (Range Trees) ועצי חיפוש עם עדיפויות (Priority Search Trees, ובקצרה PST), כאשר האחרונים הינם בעלי תכונות של ערימה ושל עץ חיפוש ([1],[2],[5],[6],[7]).

2.1 עצי תחומים (Range Trees)

עץ תחומים הוא מבנה נתונים גאומטרי אשר מטרתו היא לענות על שאילתות גאומטריות בתחומים אשר הצלעות שלהם מקבילות לצירים. במקרים רבים משתמשים בעץ תחומים לצורך מענה על שאילתות כגון: אילו נקודות נמצאות בתחום $[X_0, X_1]$ במקרה החד-מימדי או אילו נקודות נמצאות בתוך המלבן $[X_0, X_1] \times [Y_0, Y_1]$ במקרה הדו-מימדי. ניתן להשתמש בעצי תחומים גם במימדים גבוהים.

נתבונן תחילה במקרה החיפוש החד-מימדי – נניח כי רוצים למצוא את כל הנקודות מתוך אוסף הנקודות $P = \{P_1, \dots, P_n\}$ הנמצאות בתחום $[X, X']$. נניח כי T הוא עץ חיפוש בינרי מאוזן כך שכל הנקודות מ- P שמורות בעלים של T והצמתים הפנימיים ב- T מאפשרים לנווט במהלך החיפוש בעץ (הערכים בכל צומת פנימי v מחזיקים את החציון של כל הנקודות הנמצאות בעלים של העץ המושרש בצומת v).

האלגוריתם למציאת כל הנקודות בתחום $[X, X']$ הוא: מתחילים משורש העץ T ומחפשים את הצומת V_{split} שהוא הצומת המפצל בין מסלולי החיפוש של X ושל X' (כלומר מסלול החיפוש של X ממשיך שמאלה מ- V_{split} ומסלול החיפוש של X' ממשיך ימינה מ- V_{split}). אם ערכו של המפתח בצומת V_{split} הוא X'' אזי מתקיים כי $X \leq X''$ וכן $X'' > X'$. לאחר שנמצא V_{split} , במסלול החיפוש של X , אם פונים שמאלה לצומת Y מדווחים על כל העלים בתת-עץ הימני של Y (הם נמצאים בתת-עץ שמאלי של V_{split} ולכן הם קטנים מ- X' ומכיוון שהם בתת-עץ ימני של X הם גדולים מ- X). באופן דומה במסלול החיפוש של X' אם פונים ימינה לצומת Y' מדווחים על כל העלים בתת-עץ השמאלי של Y' .



איור 1 - עץ חיפוש חד מימדי, מסלולי החיפוש מודגשים באפור. מסלולי החיפוש של X ושל X' מסתיימים בעלים μ ו- μ' בהתאמה

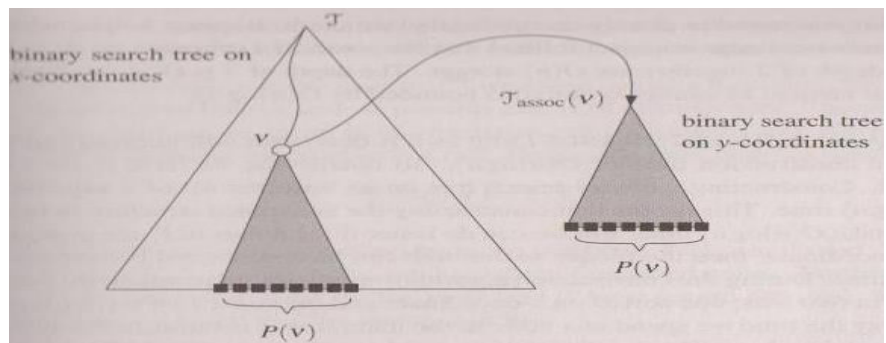
העץ שתואר מעלה הוא עץ תחומים חד-מימדי. עלות השאילתה היא $O(\log n + k)$ כאשר k הוא מספר נקודות הפלט. מכיוון שעוברים על שני מסלולים (מ- V_{split} לעלים) בעץ המאוזן (גובה העץ הוא $O(\log n)$) ומדווחים על $O(k)$ הנקודות בתחום כמתואר מעלה.

במקרה הדו-מימדי רוצים לענות על השאילתה במישור – יש למצוא את כל הנקודות הנמצאות בתחום $[X, X'] \times [Y, Y']$. נסמן ב- $P(v)$ את הקבוצה הקנונית של הצומת v (אוסף כל הנקודות הנמצאות בתת-עץ המורש בצומת v).

תיארנו קודם את האלגוריתם למציאת הערכים לפי שאילתה חד-מימדית. נזכיר אותו בקצרה למען שלמות. מתחילים משורש העץ T ומחפשים את הצומת V_{split} שהוא הצומת המפצל בין מסלולי החיפוש של X ושל X' (כלומר מסלול החיפוש של X ממשיך שמאלה מ- V_{split} ומסלול החיפוש של X' ממשיך ימינה מ- V_{split}). אם ערכו של המפתח בצומת V_{split} הוא X'' אזי מתקיים כי $X \leq X''$ וכן $X' > X''$. לאחר שנמצא V_{split} , במסלול החיפוש של X אם פונים שמאלה לצומת v מדווחים על כל העלים בתת-עץ הימני של v (הם נמצאים בתת-עץ שמאלי של V_{split} ולכן הם קטנים מ- X' ומכיוון שהם בתת-עץ ימני של X הם גדולים מ- X). באופן דומה במסלול החיפוש של X' אם פונים ימינה בצומת v' מדווחים את כל העלים בתת-עץ השמאלי של v' . באופן כזה קיבלנו איחוד זר של $O(\log n)$ קבוצות קנוניות.

כאשר רוצים לפתור את השאילתה בתחום דו-מימדי, במקום להחזיר את כל העלים הנמצאים בתת-עצים כמו במקרה החד-מימדי, לכל צומת בעץ (מלבד העלים) תהיה הצבעה לעץ חיפוש בינרי מאוזן משני (associated tree) לפי קואורדינטות ה- y כך שהוא מחזיק את הנקודות הדו-מימדיות בעלים שלו. כלומר לכל צומת v בעץ הראשי T בעל הקבוצה הקנונית $P(v)$ תהיה הצבעה לעץ $T_{assoc}(v)$ אשר יהיה עץ חיפוש בינרי מאוזן של הקבוצה $P(v)$ לפי קואורדינטת ה- y .

לכן האלגוריתם במקרה הדו-מימדי הוא כזה: מתחילים משורש העץ T ומחפשים את הצומת V_{split} בעץ הראשי שהוא הצומת המפצל בין מסלולי החיפוש של X ושל X' (כלומר מסלול החיפוש של X ממשיך שמאלה מ- V_{split} ומסלול החיפוש של X' ממשיך ימינה מ- V_{split}), אם ערכו של המפתח בצומת V_{split} הוא X'' אזי מתקיים כי $X \leq X''$ וכן $X' > X''$. לאחר שנמצא V_{split} , במסלול החיפוש של X אם פונים שמאלה, בודקים בעצים המשניים (T_{assoc}) המוצבעים מהצמתים בתת-עץ הימני לפי קואורדינטת ה- y ומדווחים בעץ המשני על הנקודות שערכי ה- y שלהם הם בתחום $[Y, Y']$. באופן דומה במסלול החיפוש של X' אם פונים ימינה בודקים בעצים המשניים המוצבעים מהצמתים בתת-עץ השמאלי לפי קואורדינטת ה- y ומדווחים בעץ המשני על הנקודות שערכי ה- y שלהם הם בתחום $[Y, Y']$.



איור 2 - תיאור עץ תחומים דו-מימדי עם עץ ראשי T ועצים משניים T_{assoc} המוצבעים מכל צומת בעץ הראשי

עלות השאילתה בעץ הדו-מימדי היא $O(\log^2 n + k)$ כאשר k הוא מספר הנקודות המדווחות (גודל הפלט). מכיוון שיש לבצע שני חיפושים (חיפוש בעץ המאוזן המשני לכל צומת במסלול מהשורש לעלים בעץ המאוזן הראשי).

מכיוון שהעץ הוא מאוזן ויש בו $O(\log n)$ רמות, עלות המקום היא $O(n \cdot \log n)$ (כל נקודה מופיעה $O(\log n)$ פעמים בזיכרון).

סיבוכיות זמן הבניה: $O(n \cdot \log n)$. נמייך את הנקודות ב- P לפי קואורדינטת ה- x שלהן ונשמור אותן ברשימה אחת, כמו-כן נמייך אותן לפי קואורדינטת ה- y שלהן ונשמור אותן ברשימה אחרת. כעת נבנה את העץ הראשי T מלמטה למעלה, מכיוון שהנקודות ממוינות לפי קואורדינטת ה- y , בכל צומת בעץ הראשי כאשר בונים את העץ המשני נמצאים זמן שהוא לינארי בגודל הקבוצה הקנונית ולכן זמן הבניה הוא כמו סדר הגודל של סיבוכיות המקום – כלומר $O(n \cdot \log n)$. מכיוון שעלות המיונים המוקדמים לפי קואורדינטת ה- x ולפי קואורדינטת ה- y הוא $O(n \cdot \log n)$, זו גם סיבוכיות זמן הבניה.

2.2 עצי חיפוש עם עדיפויות

עצי חיפוש עם עדיפות (PST) הם מבני נתונים המשלבים תכונות של ערימה בינרית ושל עץ חיפוש בינרי מאוזן.

עצים אלה שימושיים בפתרון שאילתות מישוריות שאינן חסומות בצד אחד, לדוגמה אם נתונה קבוצת הנקודות $P = \{P_1, \dots, P_n\}$ רוצים למצוא את כל הנקודות הנמצאות בתחום $X(-\infty, Y] \times [X, X']$ כך שצלעות התחום מקבילות לצירים $\{[1],[2],[3],[4],[6],[7],[8]\}$.

כדי לבנות את ה-PST נשתמש בתכונות של ערימת מינימום (minimum heap) ושל עץ חיפוש בינרי מאוזן. תהליך הבניה של העץ הוא כדלהלן:

- בהינתן קבוצת נקודות $P = \{P_1, \dots, P_n\}$, ממיינים את כל הנקודות ע"פ קואורדינטת ה- y – הנקודה p בעלת קואורדינטת ה- y הנמוכה ביותר מוכנסת לשורש ה-PST.
- עבור הנקודות $P - \{p\}$ מוצאים את החציון לפי x שנשמר אף הוא בצומת של p ואז הנקודות שקואורדינטת ה- x שלהם גדולה מהחציון מוכנסות לתת-עץ הימני והאחרות מוכנסות לתת-עץ השמאלי.
- ממשיכים ברקורסיה על התת-עץ הימני ועל התת-עץ השמאלי.

לאחר שנבנה עץ PST, האלגוריתם לחיפוש הנקודות בתחום $X(-\infty, Y] \times [X, X']$ הוא: סורקים את העץ החל משורש העץ עד שמגיעים לצומת V_{split} המפצל את המסלולים ל- X ול- X' בודקים אם ערך ה- y של הנקודה קטן מ- Y ובודקים שערך ה- x נמצא בתחום $[X, X']$. לאחר שמוצאים את V_{split} מחפשים את X בתת-עץ השמאלי, אם החיפוש אחר X פונה שמאלה בצומת v , הנקודות בתת-עץ הימני של v נמצאות בתחום $[X, X']$, יש לבדוק לנקודות אלה שערכי ה- y שלהן קטנים מ- Y , כאשר מגיעים לצומת ראשון שערך ה- y של הנקודה שלו גדול מ- Y מפסיקים לחפש בתת-עץ המורשש באותו צומת בגלל תכונת ערימת המינימום. באופן דומה מחפשים את X' בתת-עץ הימני, אם החיפוש פונה ימינה בצומת v' , הנקודות בתת-עץ השמאלי של v' נמצאות בתחום $[X, X']$ וגם להן יש לבדוק את ערכי ה- y באותו אופן המתואר מעלה.

באופן דומה ניתן לענות גם על השאילתות בתחום $X[X', X'] \times [Y, +\infty)$ תוך שימוש בערימת מקסימום לפי ערכי y .

עלות זמן השאילתה ב-PST היא $O(\log n + k)$ כאשר גודל הפלט הוא k נקודות, מכיוון שיש לעבור על עץ חיפוש בינרי מאוזן מהשורש עד העלים (תכונת עץ החיפוש) ולדווח על כל נקודות הקטנות מ- Y (תכונת הערימה).

סיבוכיות המקום ב-PST היא לינארית $O(n)$ מכיוון שכל הנקודות נשמרות ב-PST. ניתן לבנות את העץ בזמן $O(n \cdot \log n)$ מכיוון שניתן למיין קודם את הנקודות לפי קואורדינטות ה- x ואז לבנות את העץ באופן שבו בונים ערימה.

2.2.1 עץ חיפוש עם עדיפויות מסוג (RPST) RADIX

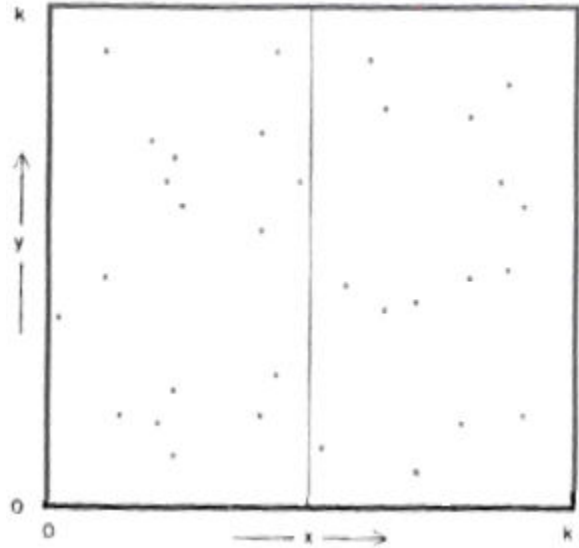
עץ RPST $([2],[8])$ הוא גרסה של עץ חיפוש עם עדיפויות (PST). העץ הזה תומך גם בפעולות חיפוש "במימד 1.5" עבור תחומי שאילתה מהסוג $[X,X'] \times (-\infty, Y]$ או $[X,X'] \times [Y, +\infty)$ כך שצלעות התחום מקבילות לצירים.

בעץ ה-RPST מניחים כי ערכי ה- x הינם שלמים וחסומים בין הערך 0 לבין $k-1$ (כאשר k הוא פרמטר שלם שנבחר מראש). בכל צומת v בעץ מלבד הנתונים הנשמרים בעץ PST כללי, נשמרים כאן גם שני ערכים $UpX, LowX$ המגדירים את תחום הערכים של קואורדינטת ה- x של כל הנקודות הנמצאות בעץ המושרש בצומת v . לשורש העץ הערך $LowX=0$, הערך $UpX=k$.

החלוקה לפי x ב-RPST איננה לפי החציון של x כמו ב-PST רגיל, אלא החלוקה היא "גאוגרפית" לפי x , כלומר בכל שלב לוקחים את האמצע בין $LowX$ ל- UpX , כך שערכי x בין $LowX$ לנקודת האמצע נשמרות בתת-עץ השמאלי וכל ערכי ה- x בין נקודת האמצע ל- UpX נשמרות בתת-עץ הימני. לאחר $O(\log k)$ חלוקות תיווצרנה k רצועות ברוחב 1 (ובהן נקודה אחת אם מניחים שאין נקודות בעלי אותו ערך של x). כמו-כן החסם על גובה העץ הוא $O(\lg k)$, ולכן עלות כל פעולת עדכון של העץ היא $O(\lg k)$ $([2],[8])$ והיא לא דורשת פעולות איזון על העץ.

בצורה פורמלית עץ RPST נבנה באופן הבא :

- ממיינים בצורה יעילה את כל הנקודות ע"פ קואורדינטת ה- y : הנקודה p בעלת קואורדינטת ה- y הנמוכה ביותר מוכנסת לשורש ה-RPST, בנוסף מוכנס התחום $(LowX, UpX)$ שבו נמצאות כל הנקודות בעץ המושרש בצומת זה. עבור השורש $LowX=0, UpX=k$.
- לנקודות $P-\{p\}$ מוצאים את ערך האמצע לפי x : $MidX = \lfloor (UpX + LowX) / 2 \rfloor$. הנקודות שקואורדינטות ה- x שלהן קטנות מ- $MidX$ מוכנסות לתת-עץ השמאלי והאחרות מוכנסות לתת-עץ הימני.
- השורש של התת-עץ הימני הוא בעל התחום $(MidX, UpX)$ והשורש של התת-עץ השמאלי הוא בעל התחום $(LowX, MidX)$.
- ממשיכים ברקורסיה על התת-עץ הימני ועל התת-עץ השמאלי.



איור 3 - החלוקה בכל שלב לפי x נעשית ע"פ נקודת האמצע של התחום $[LowX, UpX]$ של השלב הקודם

מכיוון שנקודת האמצע נבחרת בצורה "גאוגרפית" במקום בחירת החציון לא מובטח שעץ RPST הוא מאוזן (מכיוון שיתכן שרוב הנקודות נמצאות בצד אחד של האמצע). ברוב היישומים המעשיים עץ ה-RPST הוא קרוב למאוזן (אם פיזור הנקודות הוא אחיד) והביצועים שלו במקרה הממוצע הם כמו של עץ מאוזן [2].

2.3 עץ תחומים עם עדיפויות (Range Priority Tree)

כמוצג ב-[7], לצורך חיפוש בתחום מישורי $[X, X'] \times [Y, Y']$ ניתן להשתמש בעץ תחומים (ראה סעיף 2.1) במימד אחד כאשר לצורך חיפוש במימד השני משתמשים במבנה המסונף מכל צומת v $\mathcal{T}_{assoc}(v)$ במונחים של סעיף 2.1 שהוא עץ חיפוש עם עדיפויות.

העץ נבנה באופן הבא: בונים את עץ התחומים הראשי לפי קואורדינטת y כך שהנקודות עצמן נמצאות בעלים, לכל בן ימני יוצרים הצבעה לעץ PST המכיל את כל העלים הנמצאים בעץ המושרש בצומת זה, ובעל תכונת ערימת מינימום לפי קואורדינטת y . לכל בן שמאלי יוצרים הצבעה לעץ PST המכיל את כל העלים הנמצאים בעץ המושרש בצומת זה, ובעל תכונת ערימת מקסימום לפי קואורדינטת y . לשורש ולעלים אין הצבעה.

אם רוצים למצוא את כל הנקודות מתוך $P = \{P_1, \dots, P_n\}$ הנמצאות בתחום השאילתה $[B_y, E_y] \times [B_x, E_x]$ אזי יש למצוא את הצומת V_{split} אשר מפריד בין מסלולי החיפוש של B_y ו- E_y בעץ הראשי (ראה סעיף 2.1). לאחר מציאת צומת זה ברור שכל הנקודות הנמצאות בעלים של העץ המושרש ב- $left(V_{split})$ הן בעלות קואורדינטות y קטנות מ- E_y , נותר למצוא את אלה שגדולות מ- B_y ונמצאות בתחום $[B_x, E_x]$ – ניתן לעשות זאת ע"י חיפוש ב-PST המוצבע ע"י $left(V_{split})$ בעל תכונת ערימת מקסימום. באותו אופן כל הנקודות הנמצאות בעלים של העץ המושרש ב- $right(V_{split})$ הן בעלות קואורדינטות y גדולות מ- B_y , נותר למצוא את אלה שקטנות מ- E_y ונמצאות בתחום $[B_x, E_x]$ – ניתן לעשות זאת ע"י חיפוש ב-PST המוצבע ע"י $right(V_{split})$ בעל תכונת ערימת מינימום.

עלות החיפוש תוך שימוש במבנה הנתונים הנ"ל היא $O(\log n + k)$, כאשר יש k נקודות פלט. עוברים מהשורש עד V_{split} בעץ הראשי (גובה העץ הוא $O(\log n)$) ואז עוברים פעמיים על מבני ה-PST בתת-עץ הימני והשמאלי. עלות השאילתה ב-PST היא $O(\log n + k)$ (ראה סעיף 2.2).

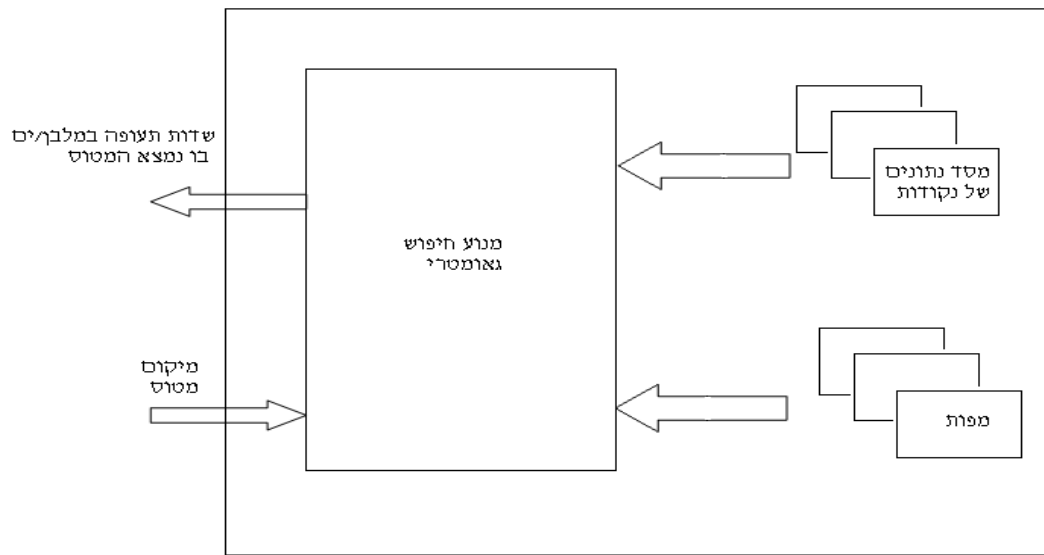
סיבוכיות המקום וסיבוכיות זמן הבניה היא $O(n \cdot \log n)$ (בהתאם למתואר בסעיף 2.1 על עצי תחומים דו-מימדיים).

בפריקט זה לצורך חיפוש שדות התעופה הנמצאים בתחום המלבני שבו נמצא המטוס נשתמש בעץ תחומים עם עדיפויות כמתואר בסעיף 2.3 כך שהעץ הראשי יהיה עץ תחומים והעצים המשניים יהיו RPST (Radix Priority Search Trees) כמתואר בסעיף 2.2.1. עץ ה-RPST המוצבע מכל בן שמאלי בעץ הראשי יהיה מבוסס על ערימת מקסימום לפי קואורדינטה y ועץ ה-RPST המוצבע מכל בן ימני בעץ הראשי יהיה מבוסס על ערימת מינימום לפי קואורדינטה y .

מבנה נתונים זה מספק יעילות חיפוש טובה יותר לעומת עץ תחומים דו-מימדי (יעילות אופטימלית) ועלות זמן הבניה וסיבוכיות המקום שלו זהים לאלו של עץ התחומים הדו-מימדי. יש לציין כי בעץ התחומים הדו-מימדי ניתן להשיג יעילות אופטימלית גם כן תוך שימוש בטכניקת fractional cascading ([11],[12]) אולם המימוש של טכניקה זו מסורבל בהרבה מהמימוש של עץ התחומים עם עדיפויות.

3. דרישות המערכת

המטרה העיקרית של פרויקט זה היא למצוא, באמצעות שימוש במבני נתונים ובאלגוריתמים של גאומטריה חישובית (ראה פרק 2), ולהציג את כל הנקודות (המייצגות שדות תעופה) הנמצאות במלבן (או במלבנים, במקרה שיש איזורי חפיפה למלבנים) שבו נמצא המטוס ברגע נתון על פני מפה דו-מימדית המייצגת את איזור הטיסה.



איור 4 - תיאור המערכת

המערכת מקבלת בתור קלט קובץ עם מיקומי כל שדות התעופה לאורך כל מסלול הטיסה. כמו-כן היא מקבלת כקלט את המפה של מסלול הטיסה. באמצעות הממשק הגרפי המשתמש יכול לשנות את גודל התחומים המלבניים וכן לייצר חפיפה בין המלבנים.

במהלך הטיסה מתקבל מיקום המטוס, נמצא המלבן (או המלבנים, במקרה שנבחרה חפיפה) בו נמצא המטוס באותו רגע ומנוע החיפוש הגאומטרי מוצא את כל שדות התעופה באותו מלבן.

הפלט הוא התצוגה של כל שדות התעופה במלבן שבו נמצא המטוס על פני מפת איזור הטיסה. המשתמש יכול לבחור אופציות שונות דרך הממשק הגרפי כגון הצגת מרחקים ממוינים משדות התעופה הנמצאים במלבן בו נמצא המטוס ועוד.

3.1 דרישות פונקציונליות

קלטי המערכת :

- קובץ עם מיקומי שדות התעופה ושמותיהם, מפות עם איזורי הטיסה, חלוקה למלבנים (כפי שנבחרה ע"י המשתמש או חלוקת ברירת מחדל).
- מיקום המטוס בכל רגע נתון.

פלט המערכת :

- מפת מסלול הטיסה עם שדות התעופה הנמצאים בתחום המלבני שבו נמצא המטוס.
- (אופציונלי) אם המשתמש מאפשר זאת – יוצגו המרחקים בין המטוס לבין כל שדות התעופה הנמצאים במלבנים בהם נמצא המטוס בק"מ (בסדר ממוין מהקרוב לרחוק).

קלט/פלט של תת-מערכות :

קלט מנוע החיפוש הגאומטרי :

- מצביע לעץ חיפוש (עץ תחומים עם עדיפויות).
- תחום/תחומי שאילתה מלבניים.

פלט מנוע החיפוש הגאומטרי :

- מיקומי הנקודות במלבן/מלבני השאילתה.

דרישות תפעוליות ומידע :

- יכולת טעינת קבצי טקסט עם מסד נתונים של שדות התעופה
- יכולת טעינת ותצוגת מפת מסלול טיסה
- בניית עץ תחומים ראשי עם נקודות ממסד הנתונים
- בניית עץ RPST (ראה סעיף 2.2.1) משני וקישורו לצמתי העץ הראשי
- יכולת הגדרת מלבנים ע"י המשתמש ע"י מסך ייעודי
- יכולת סימולציה של מסלול טיסה
- מציאת המלבן (או המלבנים) בהם נמצא המטוס
- חיפוש יעיל של שדות התעופה הנמצאים במלבן שבו נמצא המטוס מבין אוסף כל שדות התעופה
- תצוגה גרפית על גבי מפה של שדות התעופה במלבן המטוס
- ממשק גרפי שיאפשר למשתמש לבצע פעולות גרפיות שונות לצורך נוחות תצוגה
- ממשק גרפי ייעודי לתצוגת המרחקים בסדר עולה של המטוס משדות התעופה במלבן בו הוא נמצא
- יכולת הדפסת המסך ע"פ בחירת משתמש (אם קיימת מדפסת מוגדרת)
- יכולת שמירת מסך לקובץ תמונה (בפורמט BMP או JPEG)

3.2 דרישות לא פונקציונליות

דרישות חומרה מינימליות:

- מעבד 2.7GHz לפחות.
- דיסק קשיח עם קיבולת 200 GHz לפחות (תלוי בגודל ובכמות המפות וקבצי הטקסט עם מיקומי שדות התעופה).
- זיכרון RAM של 2 GHz.
- מסך עם רזולוציה של 1024x768 (רזולוציה סטנדרטית למחשב שולחני) או 1366x768 (רזולוציה סטנדרטית למחשב נייד).

אילוצי מימוש:

- המפות יכולות להיות בפורמט BMP או JPEG.
- מערכת ההפעלה הנדרשת – Windows.
- כאשר מסומלצת טיסה, המסך מתעדכן פעם בשניה, איכות התצוגה תלויה בכרטיס הגרפי של המחשב (ככל שהכרטיס הגרפי "חזק" יותר, כן המסך מתעדכן בצורה חלקה יותר ורואים פחות "קפיצתיות").

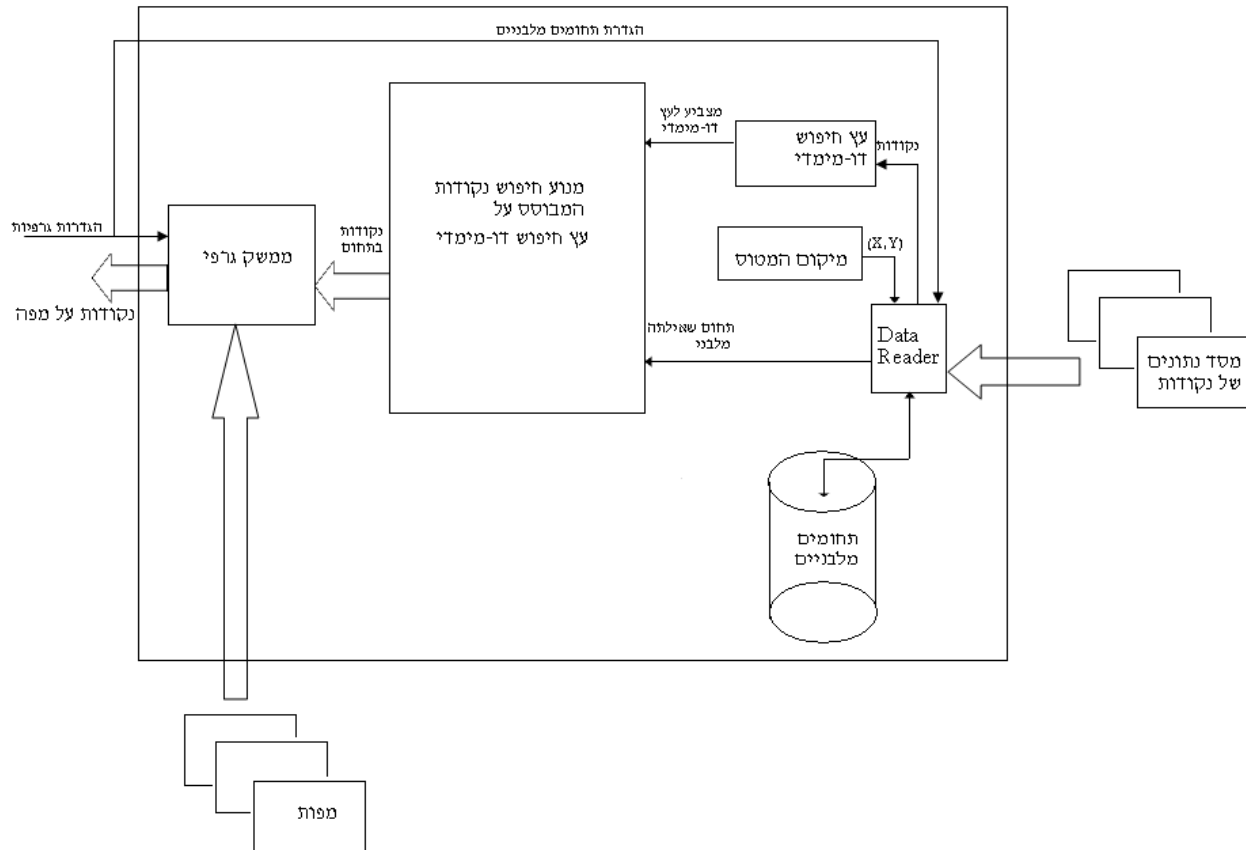
3.3 מגבלות והנחות המערכת

- המפה של מסלול הטיסה וקובץ שדות התעופה צריכים להתאים, הם נטענים בנפרד ע"י המשתמש.
- במקרה שלא נטען קובץ של שדות תעופה, קיים קובץ ברירת מחדל שאינו מתאים לאף מפה.
- קובץ שדות התעופה צריך להיות קובץ טקסט בפורמט מסוים (ראה פירוט בהמשך).

4. עיצוב מערכת התוכנה

מימוש ותכנון המערכת בוצע תחת פרדיגמת התכנות הפונקציונלי והשתמש בשפת C לצורך בניית המערכת.

4.1 דיאגרמת בלוקים של המערכת



איור 5 - דיאגרמת בלוקים של המערכת

המערכת מורכבת משלושה חלקים עיקריים :

1. עץ החיפוש הדו-מימדי אשר שומר את הנקודות במישור שנקראו מתוך הקובץ החיצוני בסדר המתאים לפי הקואורדינטות שלהן.
2. מנוע החיפוש של הנקודות המבוסס על אלגוריתם החיפוש בעץ תחומים עם עדיפויות (ראה סעיף 3.2). המנוע מקבל את תחומי השאילתה שבהם נמצא המטוס ומוצא את כל הנקודות בעץ הדו-מימדי הנמצאות בתחום זה.
3. ממשק גרפי אשר מציג למשתמש את הנקודות שנמצאו בתחומים המלבניים בהם נמצא המטוס על פני מפה המתארת את איזור הטיסה הנטענת מקובץ חיצוני. הממשק הגרפי מאפשר למשתמש לבצע התאמות שונות כגון קביעת גודל התחומים המלבניים וכן התאמות גרפיות ותפעוליות שונות.

4.1.1 תיאור רכיבים עיקריים

- Data Reader : מודול אשר תפקידו הוא להוות ממשק לקריאת נתונים חיצוניים למערכת. המודול קורא ומבצע parsing למידע מתוך קבצי הטקסט החיצוניים המחזיקים את מסד הנתונים של

הנקודות (שדות התעופה), נקודות אלה מוכנסות אח"כ לתוך עץ החיפוש הדו-מימדי. המודול קורא את הגדרות המלבנים הנקבעות ע"י המשתמש (גדלי המלבנים והגדרת חפיפה) ומאחסן את המלבנים בתוך רשימת המלבנים הכללית (אם המשתמש לא בחר לשנות את הגדרות המלבנים, נבנים מלבנים בעלי גודל default מסוים). בנוסף מודול זה מקבל את מיקום המטוס בכל רגע נתון, ניגש אל רשימת המלבנים ומוצא את המלבן (או המלבנים) בהם נמצא המטוס, אותם מלבנים מועברים כתחומי השאילתה למנוע החיפוש של הנקודות.

- עץ חיפוש דו-מימדי: במודול זה נבנה ונשמר עץ חיפוש דו-מימדי המחזיק את מסד הנתונים של כל נקודות הקלט. העץ הוא בעל המבנה שתואר בסעיף 2.3 (עץ תחומים ראשי בציר ה-y כאשר כל צומת פנימי מחזיק מצביע לעץ RPST המחזיק את הנקודות ע"פ שני המימדים). תיאור מפורט של העצים מופיע בהמשך פרק זה.
- מנוע חיפוש גאומטרי: במודול זה ממומש אלגוריתם החיפוש של הנקודות בתחומי שאילתה מסוימים. המודול מקבל מצביע לעץ החיפוש הדו-מימדי וכן הוא מקבל את תחומי השאילתה ממודול ה-data reader (המלבנים בהם נמצא המטוס). האלגוריתם מוצא את הנקודות הנמצאות בתחומים אלה (בדומה למתואר בסעיף 2.3) ומעביר אותן אל הממשק הגרפי לצורך תצוגה. תיאור מפורט של האלגוריתם מופיע בהמשך פרק זה.
- מיקום המטוס: מודול זה איננו מודול נפרד כי אם אוסף של כמה פונקציות המהוות חלק מתוך מודול הממשק הגרפי. במודול זה מבצעים סימולציה אקראית של תנועת המטוס. מתבצעת הגרלה של המיקום ההתחלתי של המטוס ובכל עדכון של הממשק הגרפי (כל שניה) מתבצעת הגרלה של המיקום הבא של המטוס. הסימולציה נפסקת כאשר המטוס יוצא מגבולות המסך. מיקום המטוס בכל רגע נתון מועבר למודול ה-data reader לצורך מציאת המלבן (או המלבנים) בו נמצא המטוס.
- ממשק גרפי: מודול זה ממשש את התצוגה למשתמש. המודול מתפרש על פני שני מודולים עיקריים: מודול עזר (Interface Auxiliary) ומודול תצוגה ראשי. במודול העזר ממומשות פונקציות עזר המאפשרות לצייר את מיקום המטוס, לצייר את המלבנים ואת הנקודות הרלוונטיות. כמו-כן מודול זה מכיל את פונקציות סימולציית מיקום המטוס. למודול העזר תפקיד נוסף – הוא מחשב את המרחקים בין המטוס לבין שדות התעופה הנמצאים במלבן שבו הוא נמצא וכן הוא ממיין אותם מהקרוב לרחוק, אם אופציה זו נבחרה ע"י המשתמש. מודול הממשק הראשי הוא מודול אשר מקבל את הפקודות מהמשתמש וקורא לפונקציות הרלוונטיות בהתאם. מודול זה מעדכן בפועל את התצוגה ע"פ שעון פנימי וכן מודול זה הוא המציג את מפת איזור הטיסה אשר נטענת ע"י המשתמש.

4.2 סביבת פיתוח

הפרויקט מומש תוך שימוש בסביבת הפיתוח והכלים המתוארים בחלק זה.

קוד הפרויקט נכתב בשפת C. סביבת הפיתוח שנעשה בה שימוש לצורך הקידוד ולצורך המימוש של הממשק הגרפי היא סביבת Borland C++ Builder 6 [10].

חבילת Borland C++ Builder היא סביבת פיתוח תוכנה המאפשרת יצירת אפליקציות שנועדו לרוץ על מחשבים עם מערכת הפעלה ממשפחת מערכות ההפעלה של Microsoft Windows. על מנת לאפשר זאת, בסביבה מוטמעת ספריית Visual Component Library (VCL) המאפשרת ליצור ממשקים גרפיים בסביבת חלונות בצורה נוחה למשתמש כך שמבחינת המפתח בניית המסך הגרפי מסתכמת בפעולות Drag & Drop בסיסיות ללא צורך בהגדרת הרכיבים הגרפיים עצמם.

4.3 עץ חיפוש דו-מימדי

מודול עצי החיפוש מכיל עץ חיפוש דו-מימדי המורכב מעץ חיפוש ראשי, עץ תחומים בציר ה-y, וכן מעצי חיפוש משניים, עצי חיפוש עם עדיפויות מסוג Radix, המוצבעים מכל צומת פנימי בעץ הראשי. כל המבנה נקרא עץ תחומים עם עדיפויות.

תפקידו של מודול זה הוא לקבל את אוסף כל הנקודות (שדות התעופה) הקיימות בקובץ מסד הנתונים ולבנות מהן את עץ החיפוש הראשי, לבנות את עצי החיפוש המשניים ולייצר הצבעה ביניהם.

כמ-כן מודול זה מממש את כל הפונקציות הרלוונטיות בעץ התחומים, את כל הפונקציות הרלוונטיות בעצי ה-RPST וכן מממש את פונקציות החיפוש בעצים אלה.

4.3.1 עץ Radix Priority Search Tree

תחילה נתאר כיצד נראה העץ המשני מסוג RPST המוצבע מכל צומת פנימי בעץ הראשי.

```
struct _RPST_TreeNode
```

```
{
```

```
    point p;
```

```
    struct _RPST_TreeNode *right;
```

```
    struct _RPST_TreeNode *left;
```

```
};
```

```
typedef struct _RPST_TreeNode RPST_TreeNode;
```

כל צומת בעץ ה-RPST (RPST_TreeNode) מכיל נקודה (ראה הגדרת מבנה נקודה בהמשך) וכן מצביע לתת-עץ הימני ולתת-עץ השמאלי.

```
struct _point
```

```
{
```

```
    int x;
```

```
    int y;
```

```
};
```

```
typedef struct _point point;
```

מבנה הנקודה (point) מכיל שתי קואורדינטות שלמות של נקודה (x,y).

פונקציות של RPST:

שם פונקציה: InsertPointMinHeapRPST

- חותמת הפונקציה: `void InsertPointMinHeapRPST (RPST_TreeNode **t, point newP, int lowerX, int upperX)`
- קלט: מצביע לעץ RPST, נקודה שיש להכניס לעץ newP, גבול תחתון של קואורדינטת x וגבול עליון של קואורדינטת x (lowerX ו- upperX בהתאמה).
- פלט: עץ RPST עם הנקודה newP.
- תיאור הפונקציה: הפונקציה מבצעת הכנסה של נקודה חדשה לתוך עץ RPST קיים, עץ זה הוא בעל תכונת ערימת מינימום לפי קואורדינטת y. תחילה הגבול העליון והגבול התחתון של קואורדינטת x הם המקסימליים (למשל, רזולוציית המסך 0-1024) וככל שיוורדים בעץ טווח קואורדינטת ה-x קטן פי 2. אם ערך קואורדינטת ה-y של newP גדול מערך קואורדינטת ה-y של השורש, ממשיכים ברקורסיה ע"י כך שמחשבים $midX = \lfloor (lowerX + upperX) / 2 \rfloor$, אם ערך קואורדינטת ה-x של newP קטן מ- $midX$ ממשיכים ברקורסיה לתת-עץ השמאלי עם תחום ערכי ה-x: $[lowerX, midX)$, אחרת ממשיכים לתת-עץ הימני עם תחום ערכי ה-x: $[midX, upperX)$. אם ערך קואורדינטת ה-y של newP קטן מזו של השורש, מציבים את newP בשורש וממשיכים ברקורסיה עם הנקודה שהיתה בשורש כמתואר מעלה. לבסוף הנקודה newP ממוקמת במקומה המתאים ומקיימת את תכונת ערימת המינימום לפי קואורדינטת ה-y ואת תכונת עץ החיפוש לפי קואורדינטת ה-x (ראה סעיף 2.2).

שם פונקציה: InsertPointMaxHeapRPST

- חותמת הפונקציה: `void InsertPointMaxHeapRPST (RPST_TreeNode **t, point newP, int lowerX, int upperX)`
- קלט: מצביע לעץ RPST, נקודה שיש להכניס לעץ newP, גבול תחתון של קואורדינטת x וגבול עליון של קואורדינטת x (lowerX ו- upperX בהתאמה).
- פלט: עץ RPST עם הנקודה newP.
- תיאור הפונקציה: הפונקציה מבצעת הכנסה של נקודה חדשה לתוך עץ RPST קיים, עץ זה הוא בעל תכונת ערימת מקסימום לפי קואורדינטת y. תחילה הגבול העליון והגבול התחתון של קואורדינטת x הם המקסימליים (למשל, רזולוציית המסך 0-1024) וככל שיוורדים בעץ טווח קואורדינטת ה-x קטן פי 2. אם ערך קואורדינטת ה-y של newP קטן מערך קואורדינטת ה-y של השורש, ממשיכים ברקורסיה ע"י כך שמחשבים $midX = \lfloor (lowerX + upperX) / 2 \rfloor$, אם ערך קואורדינטת ה-x של newP קטן מ- $midX$ ממשיכים ברקורסיה לתת-עץ השמאלי עם תחום ערכי ה-x: $[lowerX, midX)$, אחרת ממשיכים לתת-עץ הימני עם תחום ערכי ה-x: $[midX, upperX)$. אם ערך קואורדינטת ה-y של newP גדול מזו של השורש, מציבים את newP בשורש וממשיכים ברקורסיה עם הנקודה שהיתה בשורש כמתואר מעלה. לבסוף הנקודה newP ממוקמת במקומה המתאים ומקיימת את תכונת ערימת המקסימום לפי קואורדינטת ה-y ואת תכונת עץ החיפוש לפי קואורדינטת ה-x (ראה סעיף 2.2).

שם פונקציה: DeletePointMinHeapRPST

- חותמת הפונקציה: `void DeletePointMinHeapRPST (RPST_TreeNode **t, point oldP,int lowerX, int upperX)`
- קלט: מצביע לעץ RPST, נקודה שיש למחוק מהעץ oldP, גבול תחתון של קואורדינטת x וגבול עליון של קואורדינטת x (lowerX ו- upperX בהתאמה).
- פלט: עץ RPST ללא הנקודה oldP.
- תיאור הפונקציה: הפונקציה מבצעת מחיקה של נקודה מתוך עץ RPST בעל תכונת ערימת מינימום לפי קואורדינטת y. הפונקציה מחפשת את הנקודה בעץ, כל עוד הנקודה לא נמצאה מחשבים $midX = \lfloor (lowerX + upperX) / 2 \rfloor$, אם קואורדינטת ה-x של oldP קטנה מ- $midX$ ממשיכים ברקורסיה לתת-העץ השמאלי עם תחום הערכים $(lowerX, midX)$, אחרת ממשיכים ברקורסיה לתת-העץ הימני עם תחום הערכים $(midX, upperX)$. כאשר הנקודה נמצאת בעלה, העלה נמחק מהעץ, כאשר הנקודה איננה בעלה, הצומת המכיל את הנקודה מוחלף עם הבן בעל ערך ה-y הנמוך יותר, ואז ממשיכים במחיקה ברקורסיה עם הבן שערכו החליף את את הנקודה oldP.

שם פונקציה: DeletePointMaxHeapRPST

- חותמת הפונקציה: `void DeletePointMaxHeapRPST (RPST_TreeNode **t, point oldP,int lowerX, int upperX)`
- קלט: מצביע לעץ RPST, נקודה שיש למחוק מהעץ oldP, גבול תחתון של קואורדינטת x וגבול עליון של קואורדינטת x (lowerX ו- upperX בהתאמה).
- פלט: עץ RPST ללא הנקודה oldP.
- תיאור הפונקציה: הפונקציה מבצעת מחיקה של נקודה מתוך עץ RPST בעל תכונת ערימת מקסימום לפי קואורדינטת y. הפונקציה מחפשת את הנקודה בעץ, כל עוד הנקודה לא נמצאה מחשבים $midX = \lfloor (lowerX + upperX) / 2 \rfloor$, אם קואורדינטת ה-x של oldP קטנה מ- $midX$ ממשיכים ברקורסיה לתת-העץ השמאלי עם תחום הערכים $(lowerX, midX)$, אחרת ממשיכים ברקורסיה לתת-העץ הימני עם תחום הערכים $(midX, upperX)$. כאשר הנקודה נמצאת בעלה, העלה נמחק מהעץ, כאשר הנקודה איננה בעלה, הצומת המכיל את הנקודה מוחלף עם הבן בעל ערך ה-y הגבוה יותר, ואז ממשיכים במחיקה ברקורסיה עם הבן שערכו החליף את את הנקודה oldP.

שם פונקציה: EnumerateRectYUpperBound

- חותמת הפונקציה: `void EnumerateRectYUpperBound (RPST_TreeNode **t,int X0,int X1,int Y1,int lowerX,int upperX)`
- קלט: עץ RPST, תחום ערכי קואורדינטת x: $[X0, X1]$, ערך קואורדינטת y עליון - $Y1$ וכן גבול עליון ותחתון של קואורדינטת x בצומת הנתון $(lowerX, upperX)$.
- פלט: כל הנקודות בעץ ה-RPST שקואורדינטות ה-x שלהן הן בתחום $[X0, X1]$ וקואורדינטות ה-y שלהן בין 0 ל- $Y1$.

- תיאור הפונקציה : הפונקציה מבצעת חיפוש על עץ ה-RPST, בעל תכונת ערימת המינימום לפי קואורדינטת y , של כל הנקודות בטווח המבוקש. עוברים על העץ ובודקים שאם ערך ה- y של נקודה בצומת מסוים קטן מ- $Y1$ האם ערך ה- x של נקודה זו הוא בין $X0$ ל- $X1$, אם כן מחזירים את הנקודה וממשיכים ברקורסיה : מחשבים ערך x אמצעי $Midx = \lfloor (LowerX + UpperX) / 2 \rfloor$ ואז אם $X0 < MidX$ ממשיכים ברקורסיה לתת-עץ שמאלי (עם גבולות x בין $LowerX$ ל- $MidX$). אם $X1 \geq MidX$ אזי ממשיכים ברקורסיה לתת-עץ ימני (עם גבולות x בין $MidX$ ל- $UpperX$). אם שני התנאים הנ"ל מתקיימים אזי עוברים בענפים של שני התת-עצים. אם ערך קואורדינטת ה- y גדול מ- $Y1$, אותו ענף לא נבדק יותר בגלל תכונת ערימת המינימום.

שם פונקציה : EnumerateRectYLowerBound

- חותמת הפונקציה : `void EnumerateRectYLowerBound (RPST_TreeNode **t, int X0, int X1, int Y0, int lowerX, int upperX)`
- קלט : עץ RPST, תחום ערכי קואורדינטה x : $[X0, X1]$, ערך קואורדינטת y תחתון- $Y0$ וכן גבול עליון ותחתון של קואורדינטת x בצומת הנתון $(lowerX, upperX)$.
- פלט : כל הנקודות בעץ ה-RPST שקואורדינטות ה- x שלהן הן בתחום $[X0, X1]$ וקואורדינטות ה- y שלהן גדולות מ- $Y0$.
- תיאור הפונקציה : הפונקציה מבצעת חיפוש על עץ ה-RPST, בעל תכונת ערימת המקסימום לפי קואורדינטת y , של כל הנקודות בטווח המבוקש. עוברים על העץ ובודקים שאם ערך ה- y של נקודה בצומת מסוים גדול מ- $Y0$ האם ערך ה- x של נקודה זו הוא בין $X0$ ל- $X1$, אם כן מחזירים את הנקודה וממשיכים ברקורסיה : מחשבים ערך x אמצעי $Midx = \lfloor (LowerX + UpperX) / 2 \rfloor$ ואז אם $X0 < MidX$ ממשיכים ברקורסיה לתת-עץ שמאלי (עם גבולות x בין $LowerX$ ל- $MidX$). אם $X1 \geq MidX$ אזי ממשיכים ברקורסיה לתת-עץ ימני (עם גבולות x בין $MidX$ ל- $UpperX$). אם שני התנאים הנ"ל מתקיימים אזי עוברים בענפים של שני התת-עצים. אם ערך קואורדינטת ה- y קטן מ- $Y0$, אותו ענף לא נבדק יותר בגלל תכונת ערימת המקסימום.

4.3.2 עץ תחומים

בחלק זה נתאר כיצד נבנה עץ החיפוש הראשי וכיצד הצמתים הפנימיים שלו מקושרים לעצי RPST.

מבנה של צומת בעץ התחומים נראה באופן הבא :

```
struct _RangeTreeNode
{
    point p;
    int val;
    struct _RangeTreeNode *right;
    struct _RangeTreeNode *left;
    RPST_TreeNode *Assoc;
```

};

typedef struct _RangeTreeNode RangeTreeNode;

בכל צומת של עץ התחומים (מטיפוס RangeTreeNode) ישנו שדה של נקודה (בכל צומת פנימי הערך של הנקודה הוא ערך של (-1,-1), הערכים של הנקודות שמורים בעלים של העץ בלבד). שדה val הוא ערך החציון ע"פ קואורדינטת ה-y של הנקודות הנמצאות בעלים של העץ המושרש בצומת זה. לכל צומת יש מצביע לתת-עץ ימני ומצביע לתת-עץ שמאלי. בנוסף לכל צומת פנימי (צמתים מלבד השורש והעלים) יש מצביע לעץ משני (עץ Assoc) (עץ RPST).

על מנת לבנות את עץ התחומים, נמיין תחילה את הנקודות ע"פ קואורדינטת ה-y שלהם. על מנת לדעת את הנקודות עלינו לקרוא תחילה את קובץ הקלט ולבצע עליו עיבוד מתאים.

מבנה קובץ הקלט: רשומה (שורה) בקובץ היא בעלת המבנה $X : \{X, Y, Name\}$ מציין את קואורדינטת ה-x של הנקודה, Y מציין את קואורדינטת ה-y של הנקודה, Name מציין את שם הנקודה (שם שדה התעופה).

דוגמה :

60 85 Madison

80 155 Rockford

178 195 Chicago

47 383 Springfield

7 487 St. Louis

נתאר את הפונקציה הראשית המשמשת לקריאת קובץ הקלט :

שם פונקציה: ReadTargetsPosition

- חותמת הפונקציה: `point *ReadTargetsPosition(char *FileName, TargetsDistance *TgtDist)`
- קלט: שם קובץ הקלט (מסד הנתונים) וכן מצביע למבנה נתונים השומר את הנקודות, שמותיהם והמרחקים שלהן מהמטוס.
- פלט: מצביע לרשימת הנקודות בקובץ.
- תיאור הפונקציה: הפונקציה פותחת את קובץ הקלט, אם היא מצליחה היא עוברת על כל הרשומות בתוכו וממלאת שני מבנים בו-זמנית. היא בונה רשימה של הנקודות (קואורדינטות x ו-y בלבד) כך שלבסוף זו הרשימה המוחזרת מהפונקציה. כמו-כן היא ממלאת את הרשימה TgtDist שניתן אליה מצביע – ברשימה זו ממלאים את הקואורדינטות של הנקודות וכן את שם שדה התעופה ומרחקו מהמטוס (תחילה שדה זה מאופס).

נעבור כעת לבניית עץ התחומים: לאחר שהוחזרה רשימת הנקודות, מבצעים להן מיון לפי קואורדינטת y באמצעות אלגוריתם מיון יעיל סטנדרטי – Quick Sort (בעל תוחלת זמן ממוצעת של $O(n \cdot \log n)$).

פונקציות לבניית עץ התחומים:

שם פונקציה: BuildRangeTree

- חותמת הפונקציה: `void BuildRangeTree (RangeTreeNode **t,point *p,int len)`
- קלט: מצביע לעץ תחומים. מצביע לרשימת נקודות (ממוינת ע"פ קואורדינטות ה-y שלהן). אורך רשימת הנקודות.
- פלט: עץ תחומים חד-מימדי המבוסס על קואורדינטת ה-y כך שכל הנקודות נמצאות בעלים.
- תיאור הפונקציה: הפונקציה בונה את עץ התחומים בצורה רקורסיבית. כאשר גודל הרשימה גדול מ-1, הפונקציה מחשבת את החציון לפי קואורדינטת ה-y של הנקודות ומפצלת את הרשימה כך שהנקודות שערכן גדול מהחציון מופנות לתת-עץ הימני והאחרות לתת-עץ השמאלי, בצומת הפנימי נרשם החציון וערך הנקודה הוא (-1,-1), כאשר אורך הרשימה הוא 1 (נשאר איבר אחרון) הוא מוכנס לתוך העלה של העץ (מוכנס ערך הנקודה האמיתי (x,y)), ערך החציון בעלה הוא ערך קואורדינטת ה-y של הנקודה המאוחסנת בעלה זה.

שם פונקציה: GetAllLeavesOfNode

- חותמת הפונקציה: `void GetAllLeavesOfNode (RangeTreeNode **t)`
- קלט: מצביע לעץ תחומים
- פלט: רשימה גלובלית עם הנקודות בעלים של העץ.
- תיאור הפונקציה: כאשר פונקציה זו מקבלת מצביע לצומת בעץ תחומים, היא עוברת על כל העלים הנמצאים בעץ המושרש בצומת זה ומכניסה אותם לרשימה גלובלית.

שם פונקציה: CreateLinkForNode

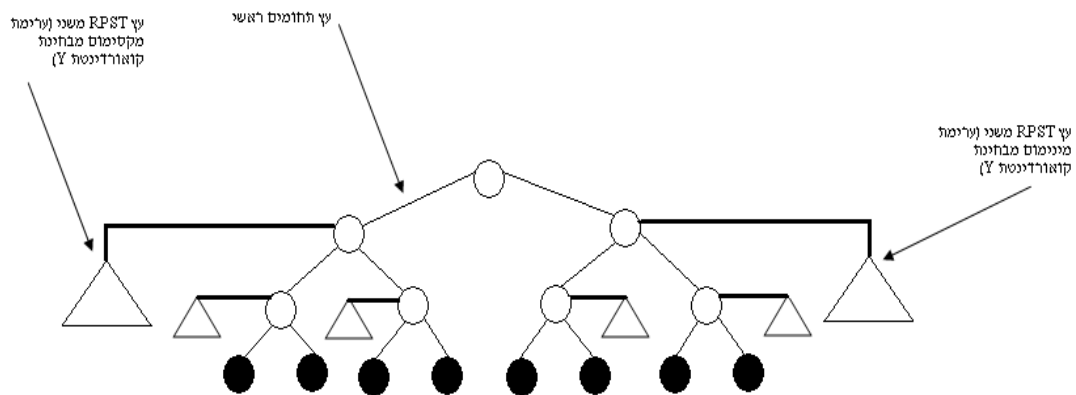
- חותמת הפונקציה: `void CreateLinkForNode (RangeTreeNode **t,int RPST_Type,int lowerX,int upperX)`
- קלט: t - מצביע לצומת של עץ תחומים. RPST_Type מציין האם עץ ה-RPST שיוצב מצומת זה הוא בעל תכונת ערימת מינימום או מקסימום ע"פ קואורדינטת y. LowerX,UpperX הם תחום קואורדינטות ה-x לעץ ה-RPST שנבנה.
- פלט: עץ תחומים כך שהצומת המתקבל (t) מצביע לעץ משני (עץ Radix Priority Search Tree) בעל תכונת ערימת מינימום או מקסימום לפי קואורדינטת y. העץ המשני מכיל את כל הנקודות הנמצאות בעלים של עץ התחומים המושרש בצומת t.

שם פונקציה: CreateLinksToRPST

- חותמת הפונקציה: `void CreateLinksToRPST (RangeTreeNode **t,int RPST_Type,int lowerX,int upperX,bool Init)`
- קלט: שורש של עץ תחומים t. RPST_Type מציין האם עץ ה-RPST שיוצב מצומת זה הוא בעל תכונת ערימת מינימום או מקסימום ע"פ קואורדינטת y. LowerX,UpperX הם תחום קואורדינטות ה-x לעץ ה-RPST שנבנה. Init - מציין האם זו קריאה ראשונה לפונקציה (קריאה עם השורש).
- פלט: עץ תחומים כך שלכל צומת פנימי ישנו מצביע לעץ משני מסוג RPST.

- תיאור הפונקציה : עבור השורש המצביע Assoc מאופס, בצורה רקורסיבית כאשר פונים לתת-עץ השמאלי – אם הצומת איננו עלה מוציאים את כל העלים מהתת-עץ שהוא שורשו, בונים עץ RPST שהוא בעל תכונת ערימת מקסימום לפי קואורדינטת y ומייצרים מצביע אליו מהצומת t בעץ התחומים. כאשר פונים לתת-עץ הימני – אם הצומת איננו עלה מוציאים את כל העלים מהתת-עץ שהוא שורשו, בונים עץ RPST שהוא בעל תכונת ערימת מינימום לפי קואורדינטת y ומייצרים מצביע אליו מהצומת t בעץ התחומים. כאשר מגיעים לעלים, המצביע שלהם ל- Assoc מאופס.

לאחר קריאת הנקודות מקובץ מסד הנתונים, ולאחר בניית עץ התחומים הראשי ועצי ה-RPST המשניים המוצבעים מכל אחד מהצמתים בעץ הראשי, כל הנקודות (שדות התעופה בכל איזור הטיסה) מאוחסנות בתוך מבנה נתונים גאומטרי בעל סיבוכיות מקום אופטימלית (ראה סעיף 2.3) המאפשר חיפוש יעיל של נקודות בתחומי שאילתה נבחרים (נקודות הנמצאות בתוך המלבן של המטוס) מתוך אוסף כל הנקודות.



איור 6 – עץ תחומים ראשי, כל צומת פנימי מצביע לעץ עדיפויות משני. הנקודות בעץ הראשי שמורות בעלים המסומנים בשחור

4.4 אלגוריתם חיפוש נקודות בתחום שאילתה מלבני

אלגוריתם החיפוש בתחום שאילתה נתון מהווה, למעשה, את לב מודול מנוע החיפוש הגאומטרי ואת לב הפרויקט כולו.

הגדרת האלגוריתם:

קלט: $P = \{P_1, \dots, P_n\}$ נקודות קלט. תחום מלבני $R = [B_x, E_x] \times [B_y, E_y]$ במישור.

פלט: תת-קבוצה של נקודות $Q \subseteq P$ הנמצאות בתחום $R = \{Q_1, \dots, Q_k\}$.

האלגוריתם פועל באופן הבא:

1. בעץ התחומים הראשי מצא צומת V_{split} שהוא אב קדמון משותף עמוק ביותר של B_y ו- E_y . צומת זה מפצל את מסלולי החיפוש של B_y ו- E_y .
2. כל הנקודות בתת-עץ השמאלי של V_{split} הן בעלות ערך קטן מ- E_y , יש למצוא רק את אלה $B_y \leq$. מצא נקודות אלה ע"י סריקה של עץ ה-RPST המבוסס על ערימת מקסימום לפי קואורדינטות y המוצבע מהבן השמאלי של V_{split} ומצא את הנקודות בתחום $[B_x, E_x] \times [B_y, +\infty)$.
3. כל הנקודות בתת-עץ הימני של V_{split} הן בעלות ערך גדול מ- B_y , יש למצוא רק את אלה $E_y \geq$.

מצא נקודות אלה ע"י סריקה של עץ ה-RPST המבוסס על ערימת מינימום לפי קואורדינטות y המוצבע מהבן הימני של V_{split} ומצא את הנקודות בתחום $[B_x, E_x] \times [-\infty, E_y]$.

עצם העובדה שכל הנקודות ב-P נשמרו בעץ תחומים עם עדיפויות מאפשרות לאלגוריתם החיפוש להגיע לזמני חיפוש קרובים לאופטימליים.

האלגוריתם צריך לעבור על עץ התחומים הראשי עד מציאתו את V_{split} , מכיוון שהעץ מאוזן – עלות זמן החיפוש של האב הקדמון המשותף העמוק ביותר של B_y ו- E_y היא $O(\log n)$. לאחר שנמצא הצומת הנ"ל מבצעים שני מעברים על עצי RPST (אחד שמוצבע מבן ימני של V_{split} ואחד שמוצבע מבן שמאלי של V_{split}), עלות כל מעבר היא $O(\log n + k)$ (בהנחה שעץ ה-RPST קרוב למאוזן) ולכן סה"כ זמן הביצוע של האלגוריתם הוא $O(\log n + k)$.

פונקציות של אלגוריתם החיפוש:

שם פונקציה: FindSplitNode

- חותמת הפונקציה: `void FindSplitNode (RangeTreeNode **t,int Y_Low,int Y_High)`
- קלט: עץ חיפוש דו-מימדי ששורשו t . תחום החיפוש בקואורדינטה y : $[Y_Low, Y_High]$.
- פלט: מצביע גלובלי שיצביע ל- V_{split} בעץ הדו-מימדי ששורשו t אשר מפצל את מסלולי החיפוש ל- Y_Low ו- Y_High .
- תיאור הפונקציה: אם גם Y_Low וגם Y_High גדולים מערך קואורדינטת ה- y בצומת מסוים אזי ממשיכים ברקורסיה לתת-עץ הימני של עץ התחומים. אם גם Y_Low וגם Y_High קטנים מערך קואורדינטת ה- y בצומת מסוים אזי ממשיכים ברקורסיה לתת-עץ השמאלי של עץ התחומים. אם הערך $Y_Low \leq$ מערך קואורדינטת y של צומת v ו- $Y_High \leq$ מערך קואורדינטת ה- y של צומת v אזי מוחזר הצומת v . אם החיפוש הגיע לעלה אזי העלה מוחזר בתור הצומת המפצל.

שם פונקציה: GetSplitNode

- חותמת הפונקציה: `RangeTreeNode *GetSplitNode (RangeTreeNode **t,int Y_Low,int Y_High)`
- קלט: מצביע לשורש עץ חיפוש דו-מימדי t . תחום חיפוש לפי קואורדינטת y : $[Y_Low, Y_High]$.
- פלט: מצביע לצומת בעץ החיפוש הדו-מימדי ל- V_{split} אשר מפצל את מסלולי החיפוש ל- Y_Low ו- Y_High .
- פונקציה זו היא פונקציית מעטפת לפונקציה `FindSplitNode` ומחזירה את המצביע הגלובלי ל- V_{split} .

שם פונקציה: FindPointsInRectangle

- חותמת הפונקציה: `void FindPointsInRectangle (RangeTreeNode **t,int X_Low,int X_High, int Y_Low,int Y_High,int lowerX,int upperX)`
- קלט: מצביע לצומת המפצל בין Y_Low ל- Y_High בעץ התחומים הראשי t . תחום שאילתה מלבני שצלעותיו מקבילות לצירים $[X_Low, X_High] \times [Y_Low, Y_High]$. תחום קואורדינטות ה- X המקסימלי $[lowerX, upperX]$.

- פלט : רשימה עם אוסף כל הנקודות בתוך המלבן $[X_Low, X_High] \times [Y_Low, Y_High]$.
- תיאור הפונקציה : בהינתן צומת מפצל t בין Y_Low ל- Y_High , הפונקציה פונה לבן השמאלי של t ובודקת בעץ ה-RPST המוצבע ממנו אילו נקודות נמצאות בתחום $[X_Low, X_High] \times [Y_Low, +\infty)$.
- הפונקציה פונה לבן הימני של t ובודקת בעץ ה-RPST המוצבע ממנו אילו נקודות נמצאות בתחום $[X_Low, X_High] \times (-\infty, Y_High]$. אם הבן הוא עלה שאין לו הצבעה ל-RPST נבדק האם הנקודה שבעלה נמצאת בתחום המבוקש. הפונקציה ממשיכה ברקורסיה בחיפוש הנקודות בתת-עצים השמאליים ע"י הפונקציה `EnumerateRectYLowerBound` ובתת-עצים הימניים ע"י הפונקציה `EnumerateRectYUpperBound`.

שם פונקציה : `GetRectanglePoints`

- חותמת הפונקציה : `point *GetRectanglePoints(void)`
- קלט : אין
- פלט : מצביע לרשימת נקודות הנמצאות בתחום שאילתה מלבני.
- תיאור הפונקציה : הפונקציה הזו היא פונקציית מעטפת לפונקציה `FindPointsInRectangle` ומחזירה מצביע לנקודה הראשונה ברשימה הגלובלית של הנקודות הנמצאות בתחום השאילתה המלבני.

שם פונקציה : `TargetsLocationAlgorithm`

- חותמת הפונקציה : `void TargetsLocationAlgorithm (rectangle *rects, point *RectPoints, RangeTreeNode *root_range_tree, int lowX, int upX)`
- קלט : `rects` – מצביע לרשימת מלבנים. `RectPoints` – מצביע לרשימת נקודות הנמצאות בתוך המלבנים. `root_range_tree` – עץ חיפוש דו-מימדי (עץ תחומים עם עדיפויות). `lowX, upX` – תחום קואורדינטות x .
- פלט : הרשימה `RectPoints` מחזיקה את אוסף כל הנקודות הנמצאות ברשימת המלבנים `rects`.
- תיאור הפונקציה : פונקציה זו היא פונקציית מעטפת שקוראת לפונקציות שתוארו קודם על מנת למצוא את רשימת הנקודות בכל מלבני הקלט. נקודה יכולה להימצא בארבעה מלבנים בו זמנית לכל היותר (באיזור חפיפה אופקי ואנכי של ארבעה מלבנים). הפונקציה עוברת על אחד מהמלבנים, קוראת לפונקציה `GetSplitNode` כדי למצוא את הצומת המפצל בעץ התחומים הראשי בין Y_High ל- Y_Low של אותו מלבן ואז היא קוראת לפונקציה `FindPointsInRectangle` וממלאת את הרשימה המוצבעת ע"י `RectPoints` בנקודות הנמצאות באותו מלבן.

5. ממשק משתמש

בממשק המשתמש קיימים 3 מסכים עיקריים:

1. מסך ראשי – במסך זה המשתמש טוען את הקובץ עם אוסף כל הנקודות וכן טוען את המפה המתאימה. במסך זה מוצג מסלול הטיסה וכן התוצאות של אלגוריתם החיפוש. במסך זה ישנן אופציות גרפיות שונות שיכולות לספק תצוגה נוחה יותר ע"פ בחירתו של המשתמש. מהתפריט של מסך זה ניתן גם להגיע למסכים האחרים הקיימים בפרויקט.
2. מסך תצורה (קונפיגורציה) – מסך זה פעיל כאשר אין הפעלה של הטיסה. מסך זה מאפשר לקבוע את תצורת התחומים המלבניים. תחת התצורה במסך הראשי, המשתמש יכול לבחור האם המלבנים יהיו בעלי איזורי חפיפה (אופקיים ואנכיים) בקצותיהם או לאו. במסך התצורה, המשתמש יכול לבחור את גודל הצלעות של התחומים המלבניים (הן בציר ה-x והן בציר ה-y). כאשר נפתח מסך זה מופיעים ערכי ברירת המחדל לגודל המלבנים.
3. מסך מרחקים – מסך אשר פעיל רק במהלך טיסה. כאשר מוצגת טיסה, המשתמש יכול לבחור להציג מרחקים מהמסך הראשי. במקרה כזה נפתח מסך שמופיע מעל גבי המפה ומציג את המרחקים בין המטוס לבין שדות התעופה הנמצאים במלבן (או המלבנים) שבו הוא נמצא. המרחקים מוצגים בסדר ממזרח לרחוק ומתעדכנים בכל פעם שהטיסה מתקדמת (פעם בשניה).

5.1 סימולציית מסלול טיסה

הכוח המניע את התוכנה הוא התקדמות הטיסה של המטוס. על מנת לדמות מסלול טיסה, משתמשים בסימולציה אקראית של התקדמות המטוס.

הטיסה מתקדמת ע"פ שיעון פנימי פעם בשניה. בפעם הראשונה מוגרל המיקום ההתחלתי של המטוס כאשר הוא נכנס למפת איזור הטיסה. בפעם הראשונה מוגרלים:

- מיקום אופקי התחלתי – הצד של המסך (ימין/שמאל) שממנו מתחילה הטיסה.
- מיקום אנכי התחלתי – מוגרלת קואורדינטת ה-y בצד המסך הנבחר שממנה מתחילה סימולציית הטיסה.

לאחר תחילת הסימולציה, בכל פעימה של השעון (פעם בשניה) מוגרלת הנקודה הבאה שבה יופיע המטוס. הנקודה הבאה תלויה בצד ההתחלתי של הטיסה כך שטיסה לא יכולה לחזור לצד המסך שממנו התחילה. אם נניח כי הטיסה החלה מצד שמאל של המסך אזי הנקודה הבאה של מיקום המטוס מוגרלת מתוך האפשרויות הבאות:

- תנועה ימינה בקו ישר.
- תנועה למעלה בקו ישר.
- תנועה למטה בקו ישר.
- תנועה לימין באלכסון כלפי מעלה.
- תנועה לימין באלכסון כלפי מטה.

יש לציין שאסורה תנועה למעלה ומיד אחריה תנועה כלפי מטה או להיפך, מכיוון שתנועה המבטלת את התנועה הקודמת איננה סבירה ועלולה לגרום להיתקעות ארוכה של הטיסה.

אם נניח כי הטיסה החלה מצד ימין, הנקודה הבאה מוגרלת כמתואר מעלה, אבל במקום תנועה בכיוון ימין, התנועה מתבצעת בכיוון שמאל.

פונקציות המייצרות את סימולציית הטיסה:

שם פונקציה: SelectStartSide

- חותמת הפונקציה: void SelectStartSide (int)
- קלט: אין
- פלט: צד הטיסה ההתחלתי.
- תיאור הפונקציה: הפונקציה מגרילה את המיקום האופקי של תחילת הטיסה (צד ימין או צד שמאל של המסך).

שם פונקציה: SelectStartPoint

- חותמת הפונקציה: void SelectStartPoint(int *x,int *y,int *StartSide)
- קלט: מצביעים לנקודת תחילת הטיסה (x,y) ולצד הטיסה ההתחלתי.
- פלט: נקודת הטיסה ההתחלתית וצד הטיסה ההתחלתי.
- תיאור הפונקציה: הפונקציה משתמשת בפונקציה SelectStartSide כדי להגדיל את צד הטיסה ההתחלתי וכן היא מגרילה את מיקום נקודת תחילת הטיסה בהתאם לצד ההתחלתי שהוגרל ומחזירה אותה.

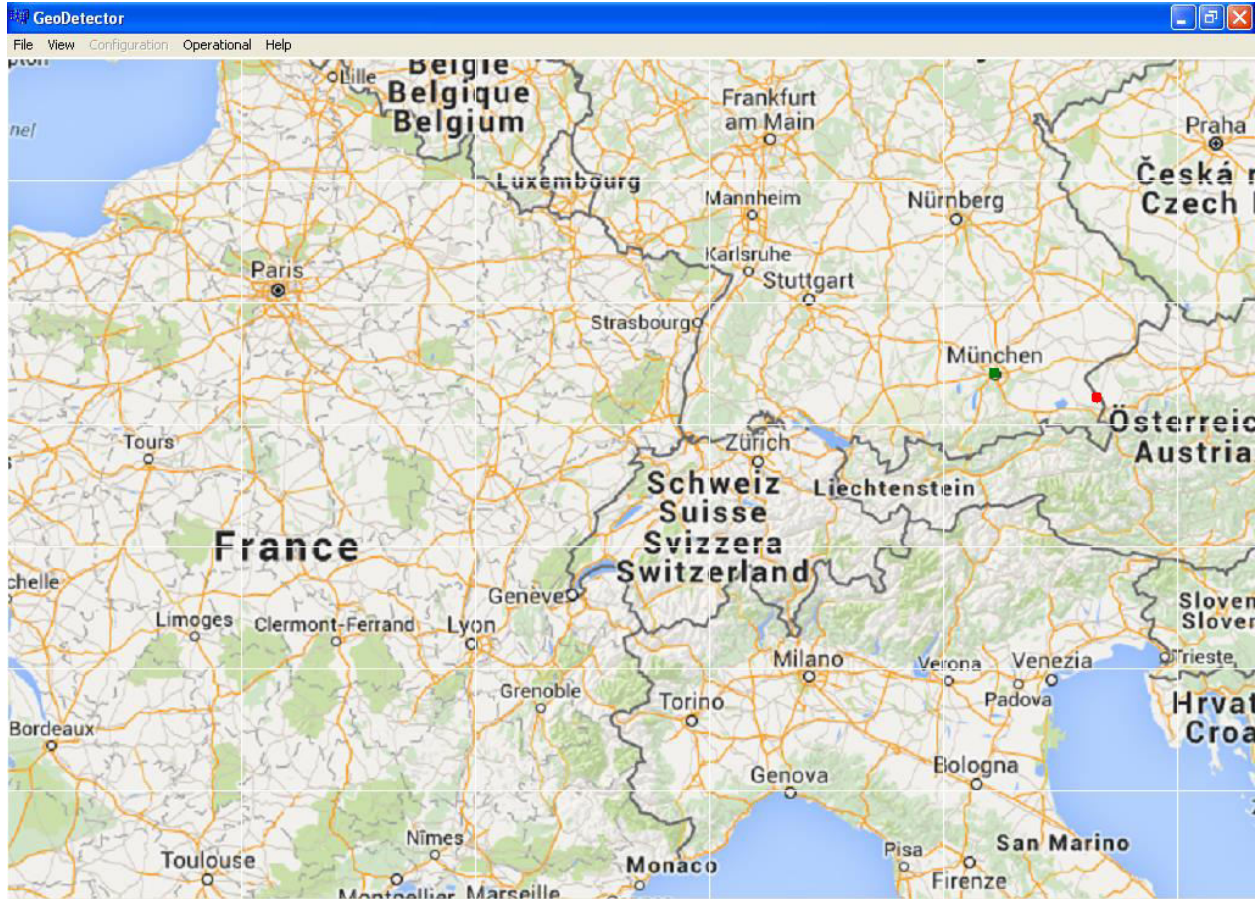
שם הפונקציה: SelectNextPlatformPosition

- חותמת הפונקציה: void SelectNextPlatformPosition(int *x,int *y,int StartSide,bool *Valid_Pos)
- קלט: מצביעים למיקום (x,y) הקודם. StartSide - צד תחילת טיסה.
- פלט: נקודת טיסה הבאה (x,y). Valid_Pos – מציינ האם מיקום המטוס הוא בתוך גבולות המסך או מחוץ להם.
- תיאור הפונקציה: הפונקציה מקבלת את המיקום הקודם של המטוס ואת צד תחילת הטיסה ומחשבת את המיקום הבא של המטוס בצורה אקראית כפי שתואר קודם בפרק זה. לאחר הגרלת המיקום הבא, הפונקציה בודקת האם המיקום החדש נמצא בתוך גבולות המסך (בהתאם לרזולוציית המסך), אם הנקודה נמצאת בתוך גבולות המסך, הסימולציה ממשיכה. אם הנקודה נמצאת מחוץ לגבולות המסך, אזי הטיסה מגיעה לסיומה.

5.2 מסך ראשי

דרך המסך הראשי המשתמש יכול לטעון את המפה של איזור הטיסה ואת שדות התעופה באיזור הטיסה. כמו-כן הוא יכול לשנות פרמטרים גרפיים שונים ואף לגשת למסכים אחרים.

כאשר המשתמש מתחיל את הטיסה – על גבי המפה מוצג המטוס ושדות התעופה שנמצאים במלבן שבו הוא נמצא (אם ישנם כאלה).



איור 7 - המסך הראשי. המלבנים מסומנים בלבן, המטוס מסומן כעיגול אדום, שדה התעופה במלבן שלו מסומן בירוק

5.2.1 התפריטים במסך הראשי

במסך הראשי ישנם תפריטים רבים בסרגל התפריטים המופיע בחלק העליון של המסך. תפריטים אלה יפורטו בחלק זה.

תפריט File:

תפריט זה מאפשר לטעון את מסד הנתונים של שדות התעופה ואת המפות וכן לבצע פעולות תפעוליות שונות של המסך.

- תת-תפריט Load Targets: תפריט זה פותח תיבת דו-שיח (dialog box) אשר באמצעותה המשתמש יכול לנווט בין קבצי טקסט שונים ולטעון את הקובץ אשר מכיל את מסד הנתונים של שדות התעופה הרלוונטיים לאיזור הטיסה המבוקש.

- תת-תפריט Load Map : תפריט זה פותח תיבת דו-שיח (dialog box) אשר באמצעותה המשתמש יכול לנווט בין קבצי תמונות שונים ולטעון את הקובץ אשר מכיל את המפה של איזור הטיסה המבוקש. ברור כי המפה ומסד הנתונים של שדות התעופה צריכים להתאים זה לזה.
- תת-תפריט Save : תפריט זה פותח תיבת דו-שיח המאפשרת לשמור את המסך בכל רגע נתון בצורת תמונה לתוך קובץ (בתור קובץ BMP או בתור קובץ JPEG).
- תת-תפריט Print : תפריט זה פותח תיבת דו-שיח להדפסה. אם קיימת מדפסת מחוברת, תפריט זה שולח את תמונת המסך להדפסה.
- תת-תפריט Exit : תפריט זה סוגר את האפליקציה.

תפריט View:

תפריט זה מאפשר לשנות פרמטרי תצוגה שונים הרלוונטיים בעיקר במהלך הטיסה.

- תת-תפריט Hide/Show Rectangles : תת-תפריט זה מאפשר להסתיר או להראות את התחומים המלבניים ע"פ המפה. ברירת המחדל היא שהמלבנים מוצגים על פני המפה אם לא נבחר אחרת.
- תת-תפריט Hide/Show Platform : תת-תפריט זה פעיל רק במהלך הטיסה, הוא מאפשר להסתיר או להציג את המטוס. ברירת המחדל היא שהמטוס מוצג בכל רגע נתון במהלך הטיסה. אם המשתמש בחר להסתיר את המטוס וכעבור פרק זמן מסוים יבחר להציג אותו, המטוס יופיע במיקום הרלוונטי לרגע הצגתו מחדש.
- תת-תפריט Show/Hide Distances : תת-תפריט זה מאפשר לפתוח מסך חדש אשר בו מוצגים המרחקים לכל שדות התעופה במלבן (או המלבנים) של המטוס מהקרוב ביותר לרחוק ביותר. ברירת המחדל הוא שהמרחקים לא מוצגים.
- תת-תפריט Show/Hide All : תת-תפריט זה מאפשר להציג את כל שדות התעופה הקיימים במסד הנתונים על גבי המפה ללא תלות במלבן שבו נמצא המטוס. אם נבחרה אפשרות זו – כל שדות התעופה במסד הנתונים מוצגים בצבע כחול על פני המפה בעוד שדות התעופה במלבן שבו נמצא המטוס מוצגים בצבע ירוק. ברירת המחדל של תפריט זה היא לא להציג את כל שדות התעופה במסד הנתונים.

תפריט Configuration :

תפריט זה מאפשר לשנות את תצורת המלבנים (בחירת חפיפה וגודל של המלבנים). תפריט זה איננו מאופשר במהלך טיסה, אלא רק בזמנים שבהם הטיסה לא פעילה. הכרח זה נובע מההנחה שהפרמטרים הנבחרים בתפריט זה אינם יכולים להשתנות במהלך טיסה.

- תת-תפריט Overlap/Separate Rectangles : תת-תפריט זה מאפשר לייצר חפיפה בין המלבנים. החפיפה נוצרת בקצוות המלבנים הן בציר האופקי והן בציר האנכי. כאשר מאפשרת חפיפה, המטוס יכול להימצא ביותר ממלבן אחד בו-זמנית (עד ארבעה מלבנים בכל רגע נתון). איזור החפיפה הוא של 10 פיקסלים. ברירת המחדל של תפריט זה היא שהחפיפה לא מאופשרת אלא המלבנים מופרדים.
- תת-תפריט Adjust Rectangle Sizes : תת-תפריט זה פותח מסך חדש שבו ניתן לקבוע את גודל הצלע האופקית ואת גודל הצלע האנכית של המלבנים (כל המלבנים בעלי גודל זהה). אם גודל הצלעות טרם שונה, מופיעים גדלי ברירת מחדל.

תפריט Operational :

בתפריט זה ניתן לבחור אופציות תפעוליות שונות של הטיסה אשר רלוונטיות ברובן במהלך הטיסה.

- תת-תפריט Start : תת-תפריט זה מתחיל את סימולציית הטיסה. אופציה זו לא מאפשרת במהלך הטיסה.
- תת-תפריט Pause/Resume : תת-תפריט זה מאפשר לעצור את הטיסה מהמקום האחרון שבו הטיסה עצרה וגורמת לחידוש העדכון של המסך. תת-תפריט זה מאפשר במהלך הטיסה בלבד.
- תת-תפריט Stop : תת-תפריט זה מאפשר להפסיק את הטיסה. תת-תפריט זה מאפשר במהלך טיסה בלבד.
- תת-תפריט Randomize New Route : תת-תפריט זה מאפשר להתחיל סימולציית טיסה ממקום אקראי חדש מבלי לעצור אותה. לאחר בחירת תפריט זה מוגרל מקום התחלתי חדש לטיסה (בקצה המסך) והטיסה מתחילה משם. תת-תפריט זה מאפשר במהלך טיסה בלבד.

תפריט Help :

בתפריט זה ניתן לבחור בתת-תפריט About המציג קרדיטים ומידע כללי על האפליקציה.

5.2.2 פונקציות של המסך הראשי

שם פונקציה : PointLocate

- חותמת הפונקציה : `rectangle *PointLocate (int x, int y, rectangle *rect, int num_rects)`
- קלט : נקודה (x,y) . `rect` – מצביע לרשימת מלבנים. `num_rects` – מספר מלבנים כולל.
- פלט : מצביע לרשימת מלבנים בהן נמצאת הנקודה (x,y) .
- תיאור הפונקציה : פונקציה זו מוצאת את המלבן (או עד ארבעה מלבנים במקרה של חפיפה) שבה נמצאת הנקודה (x,y) . פונקציה זו מופעלת ע"י השעון הפנימי המקדם את הטיסה ותפקידה הוא למצוא את המלבנים בהם נמצא המטוס בכל רגע נתון. המלבנים אשר נמצאים ע"י פונקציה זו הם תחומי השאילתה המהווים קלט לאלגוריתם החיפוש.

שם פונקציה : DrawRectangles

- חותמת הפונקציה : `void DrawRectangles (TForm1 *Form1, rectangle *Rectangles, int Wd, int Hg)`
- קלט : `Form1` – המסך עליו מציירים. `Rectangles` – מצביע לרשימת מלבנים. `Wd` – רוחב המלבנים. `Hg` – גובה המלבנים.
- פלט : מסך עם מלבנים.
- תיאור הפונקציה : הפונקציה מקבלת את רשימת המלבנים בעלי גובה ורוחב שנקבעו ומציירת אותם על המסך בהתאם לפרמטרים שנבחרו במסך הקונפיגורציה או ברירת המחדל.

שם פונקציה : DrawTargetPoints

- חותמת הפונקציה : `void DrawTargetPoints (point *RectPoints, TForm1* Form1)`
- קלט : `Form1` – המסך עליו מציירים. `RectPoints` – רשימת נקודות (שדות תעופה) שיש לצייר.

- פלט : מסך עם שדות תעופה מצוירים.
- תיאור הפונקציה : פונקציה זו מקבלת את הנקודות הנמצאות במלבן (או מלבנים) של המטוס, כלומר את פלט אלגוריתם החיפוש, ומציירת את הנקודות האלה על גבי התצוגה. אם המשתמש בחר באופציה להציג את כל שדות התעופה במסך (ראה סעיף 5.2.1) אזי פונקציה זו מציירת בצבעים שונים את פלט אלגוריתם החיפוש ואת שדות התעופה האחרים.

5.3 מסך תצורה (קונפיגורציה)

במסך התצורה ניתן לבחור את גודל התחומים המלבניים. יחד עם האפשרות לבחור לבצע חפיפה – מסך זה נותן למשתמש שליטה מלאה על גודל מלבני השאלתה. אם המשתמש צריך חלוקה מעודנת יותר, הוא יכול לבחור מלבנים קטנים יותר ואם הוא מסתפק בחלוקה גסה – יוכל לבחור מלבנים גדולים יותר.

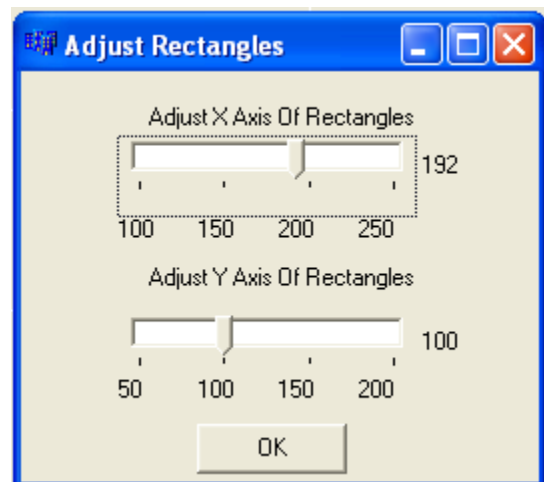
המבנה של מלבן מוגדר בתור 4 פרמטרים : $[X_Low, X_High] \times [Y_Low, y_High]$.

```
struct _Rectangle
```

```
{
    int x_low;
    int x_high;
    int y_low;
    int y_high;
};
```

```
typedef struct _Rectangle rectangle;
```

כפי שניתן לראות מהמימוש במבנה rectangle ישנם 4 פרמטרים המגדירים את גבולות המלבן בשני הצירים.



איור 8 - מסך קונפיגורציה לבחירת גדלי מלבנים

אם נבחרה האפשרות של יצירת חפיפה בין התחומים המלבניים, מפת המלבנים נראית כמתואר באיור 9.



איור 9 - מפת מלבנים עם איזורי חפיפה אופקית ואנכית

פונקציות של מסך התצורה:

שם פונקציה: RectanglesBuilder

- חותמת הפונקציה: `rectangle *RectanglesBuilder(int Width, int Height, int OverlapRectangles)`
- קלט: `Width` - רוחב המלבן הנבחר. `Height` - גובה המלבן הנבחר. `OverlapRectangles` - דגל המציין האם יש חפיפה.
- פלט: מצביע לרשימת מלבנים המכסים את המפה.
- תיאור הפונקציה: פונקציה זו בונה בהתאם לרזולוציית המסך את מפת המלבנים בעלי הרוחב והגובה שנבחרו (או ערכי ברירת מחדל) כך שהם יכסו את מלוא המסך. אם המשתמש בוחר אופציה של חפיפה, המלבנים נבנים בהזזה מסוימת (איזור חפיפה של 10 פיקסלים) הן בציר האופקי והן בציר האנכי. רשימת המלבנים המופקת ע"י פונקציה זו מחזיקה את תחומי השאלתה הפוטנציאליים של אלגוריתם החיפוש.

5.4 מסך מרחקים

אם המשתמש בוחר את האפשרות Show Distances במסך הראשי (ראה סעיף 5.2.1), נפתח מסך חדש שנמצא מעל המסך הראשי ומציג את המרחקים לכל שדות התעופה הנמצאים במלבן (או המלבנים) של המטוס.

לכל נקודה במסך הנתונים (לכל שדה תעופה) קיים מבנה המכיל את המיקום (x,y) של הנקודה, את שמה ואת המרחק שלה בכל רגע נתון מהמטוס.


```
struct _TargetsDistance
```

```
{
```

```
    point p;
```

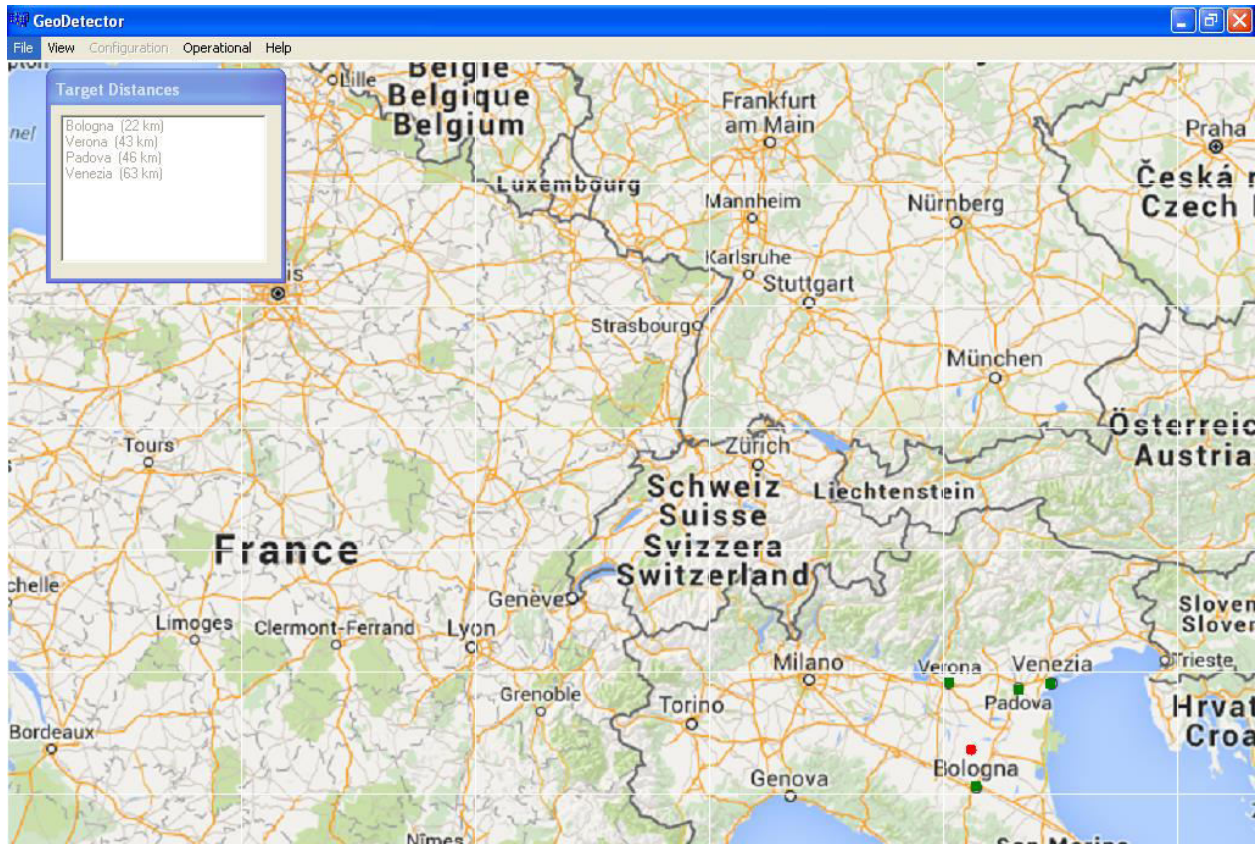
```
    char Name[20];
```

```
    int dist;
```

```
};
```

```
typedef struct _TargetsDistance TargetsDistance;
```

מבנה הנתונים מטיפוס TargetsDistance מכיל את הנקודה p מטיפוס point (ראה סעיף 4.3.1), את שם שדה התעופה (עד 20 תווים) ואת השדה dist המציין את המרחק מהמטוס. מיקום הנקודה והשם נקראים מתוך מסד הנתונים בקובץ החיצוני. שדה המרחק מחושב בזמן ריצה עבור הנקודות הנמצאות במלבן (או המלבנים) של המטוס.



איור 10- מסך מרחקים ממוינים של נקודות במלבן בו נמצא המטוס

פונקציות של מסך המרחקים:

שם פונקציה: CalcEuclideanDistance

- חותמת הפונקציה: `int CalcEuclideanDistance (int x0,int y0,int x1,int y1)`
- קלט: שתי נקודות במישור (x_0,y_0) ו- (x_1,y_1) .
- פלט: מרחק איקלידי בין שתי הנקודות.
- תיאור הפונקציה: הפונקציה משתמשת בקנה מידה של המפה שבה נעשה שימוש כדי לקבוע בקירוב כמה פיקסלים מייצגים ק"מ אחד של מרחק. הפונקציה מחשבת את המרחק האיקלידי בין שתי נקודות על הצג (בפיקסלים) וממירה אותו למרחק בקילומטרים בהתאם לקנה המידה.

שם פונקציה: CalcTargetsDistances

- חותמת הפונקציה: `void CalcTargetsDistances(point *RectPoints,TargetsDistance *TgtDist,int Platform_X,int Platform_Y,int *NumP)`
- קלט: `RectPoints` – מצביע לרשימת נקודות הנמצאות במלבן המטוס. `TgtDistance` – מצביע למבנה המרחקים. `(Platform_X,Platform_Y)` – הנקודה המייצגת את מיקום המטוס. `NumP` – מצביע למספר הנקודות במלבן (מלבנים) של המטוס.
- פלט: `TgtDistance` – מחזיק מצביע לרשימת מבנה המרחקים לכל הנקודות במלבן של המטוס.
- תיאור הפונקציה: הפונקציה מקבלת את רשימת כל הנקודות הנמצאות במלבן (מלבנים) הנוכחי שבו שנמצא המטוס, לכל אחת מהנקודות מחושב שדה המרחק במבנה המרחקים. לאחר סיום החישוב של הפונקציה, לכל נקודה במלבן המטוס מוצב המרחק שלה מהמטוס ברגע נתון.

שם פונקציה: QuickSortDistances

- חותמת הפונקציה: `void QuickSortDistances (TargetsDistance *TgtDist,int start,int end)`
- קלט: רשימת מבני המרחקים `TgtDist`. אינדקס התחלה-`start`. אינדקס סיום-`end`.
- פלט: רשימת מבני המרחקים ממוינת ע"פ שדה המרחק.
- תיאור הפונקציה: הפונקציה ממיינת את רשימת מבני המרחקים של הנקודות במלבן של המטוס ע"פ שדה המרחק בין הנקודה למטוס. נעשה שימוש באלגוריתם מיון יעיל סטנדרטי – Quick Sort (בעל תוחלת סיבוכיות זמן אופטימלית של $O(n \cdot \log n)$).

שם פונקציה: PresentTargets

- חותמת הפונקציה: `void PresentTargets (TargetsDistance *TgtDist,int Num)`
- קלט: רשימת מבני המרחקים של הנקודות במלבן של המטוס ממוינת ע"פ מרחקים – `TgtDist`. מספר הנקודות – `Num`.
- פלט: אין.
- תיאור הפונקציה: הפונקציה מציגה את המרחקים (בק"מ) של שדות התעופה במלבן המטוס מהמטוס ברגע הנתון החל מהקרוב ביותר ועד הרחוק ביותר.

שם פונקציה : CalcAndDisplayDistances

- חותמת הפונקציה : void CalcAndDisplayDistances (void)
- קלט : רשימת מבני מרחקים גלובלית.
- פלט : אין.
- תיאור הפונקציה : הפונקציה מהווה פונקציית מעטפת שעושה שימוש בשלושת הפונקציות הקודמות. פונקציה זו קוראת לפונקציית חישוב המרחקים של הנקודות במלבן המטוס מהמטוס, ממיינת אותם בצורה יעילה (קוראת ל-Quick Sort של המרחקים) ומציגה אותם בסדר ממוין.

6. בדיקות המערכת

הבדיקות של המערכת נחלקות לשני סוגים :

1. בדיקות יחידה – בדיקות אלה מאמתות את הפעולה הנכונה של מודולים נפרדים עד רמת הפונקציות הבודדות, כאשר הם מופרדים משאר המערכת. למשל – האם עץ התחומים נבנה בצורה נכונה עבור נקודות קלט מסוימות וכו'.
2. בדיקות אינטגרציה – בדיקות אלה בודקות את יחסי הגומלין בין המודולים השונים במערכת ואת פעולתם ההדדית התקינה. בבדיקות אלה בונים תת-מערכת מכמה מודולים נפרדים ובודקים את פעולתם התקינה. בהמשך מוסיפים מודולים נוספים עד שמגיעים לבדיקה של כל המערכת.

6.1 בדיקות יחידה

בדיקות היחידה נעשו מרמת הפונקציות עד רמת המודולים הנפרדים. בדיקות היחידה מבוצעות על קלט ידוע מראש כך שניתן לחשב את הפלט בצורה ידנית ולהשוות אותו עם הפלט בפועל. בדיקות היחידה נכתבו בתור תוכנית נפרדת תחת סביבת Borland C++ Builder שבה נקראו הפונקציות הנבדקות עם קלט ידוע מראש ונבדק הפלט של פונקציות אלה.

בדיקות יחידה למודול בניית עץ החיפוש

תחילה הוגדרו נקודות בדיקה ידועות מראש אשר מהן יש לבנות את עץ החיפוש. נקודות הבדיקה הוגדרו באופן הבא :

```
point TestPoints[9];  
TestPoints[0].x = 2;  
TestPoints[0].y = 6;  
TestPoints[1].x = 10;  
TestPoints[1].y = 6;  
TestPoints[2].x = 4;  
TestPoints[2].y = 7;  
TestPoints[3].x = 8;  
TestPoints[3].y = 4;  
TestPoints[4].x = 6;  
TestPoints[4].y = 2;  
TestPoints[5].x = 7;  
TestPoints[5].y = 5;
```

```
TestPoints[6].x = 5;
TestPoints[6].y = 10;
TestPoints[7].x = 4;
TestPoints[7].y = 5
TestPoints[8].x = 4;
TestPoints[8].y = 6;
```

בשלב הראשון בודקים את מיון הנקודות הנ"ל ע"פ קואורדינטת ה-y באמצעות Quick Sort. הפלט הנדרש הוא: (6,2),(8,4),(7,5),(4,5),(10,6),(4,6),(2,6),(4,7),(5,10).
נריץ את הפונקציה:

```
QuickSortPointsYCoord(TestPoints,0,8);
```

ונקבל כי אכן נקודות הקלט ממוינות כנדרש ונשמרות במערך הבדיקה TestPoints.
בשלב הבא נבדוק את בניית עץ התחומים החד-מימדי אשר מקבל כקלט את הנקודות הממוינות ובונה עץ תחומים כך שהנקודות נשמרות בעלים וניתן לנווט בתוך העץ (הבנוי ע"פ קואורדינטות y) ע"י החציונים השמורים בצמתים הפנימיים (ערך הנקודות בצמתים הפנימיים הוא (-1,-1)).
נגדיר אובייקט מסוג עץ תחומים ונאתחל אותו ב-NULL:

```
RangeTreeNode *root_range_tree = NULL;
```

נריץ את הפונקציה

```
BuildRangeTree(&root_range_tree,TestPoints,9);
```

אשר מקבלת את הנקודות הממוינות ע"פ קואורדינטת y ובונה עץ תחומים כך שהנקודות עצמן נשמרות בעלים ודרך הצמתים הפנימיים ניתן לנווט בעץ לפי קואורדינטת y. לא נתאר כאן את העץ שנוצר אבל ע"י בדיקה באמצעות debugger של סביבת העבודה ניתן לבדוק את העץ ואת תהליך בנייתו ואכן נבדק שהוא נבנה כנדרש.

בשלב הבא של בניית העץ רוצים להוסיף לעץ החד-מימדי את המימד הנוסף. על מנת להשיג זאת, כל הצמתים הפנימיים בעץ התחומים (מלבד השורש והעלים) צריכים להצביע לעץ RPST. כדי לבדוק שעץ ה-RPST נבנה בצורה תקינה נקרא לפונקציה:

```
CreateLinksToRPST (&root_range_tree,MIN_HEAP_RPST,lowX,upX,true);
```

אשר בונה עץ RPST (הפונקציה נקראת עם פרמטר המסמן שעץ ה-RPST יהיה בעל תכונת ערימת מינימום לפי קואורדינטה y, אבל הפרמטר הראשון הוא למעשה חסר חשיבות מכיוון שהוא מתייחס לעץ ה-RPST שיוצב מהשורש, אך כפי שכבר צוין, לשורש אין הצבעה לעץ RPST). במקרה הזה התחום של קואורדינטות x: [lowX,upX] נבחר להיות בין 0 ל-12 (בהתאם לנקודות הבדיקה שתוארו קודם).

הפונקציה CreateLinksToRPST קוראת בתורה לפונקציה CreateLinkForNode אשר תפקידה הוא לבנות עץ RPST עבור צומת פנימי כלשהו בעץ התחומים, פונקציה זו קוראת בתורה לפונקציה GetAllLeavesOfNode אשר מטרתה היא למצוא את כל העלים הנמצאים בעץ המושרש בצומת שעבורו בונים את עץ ה-RPST. כמו-כן הפונקציה CreateLinkForNode קוראת לפונקציה InsertPointMinHeapRPST, אשר מכניסה נקודות לעץ RPST בעל תכונה של ערימת מינימום ע"פ קואורדינטה y , עבור הבן הימני של הצומת וכן קוראת לפונקציה InsertPointMaxHeapRPST, אשר מכניסה נקודות לעץ RPST בעל תכונה של ערימת מקסימום ע"פ קואורדינטה y , עבור הבן השמאלי של הצומת.

לבסוף הפונקציה CreateLinksToRPST מייצרת הצבעות לעצי ה-RPST מכל אחד מהצמתים הפנימיים של עץ התחומים הראשי והמצביעים של העלים והשורש מצביעים ל-NULL.

לא נתאר כאן את כל פרטי הבדיקה אך באמצעות ה-debugger של סביבת העבודה, ניתן לבדוק את ההצבעות של כל צומת וצומת בכל אחד מהעצים ואכן לראות שהתוצאות הן כמצופה.

בדיקות יחידה לאלגוריתם החיפוש

לאחר בניית עץ החיפוש הדו-מימדי יש לבדוק שאכן אלגוריתם החיפוש פועל כנדרש ומחזיר את כל הנקודות בתוך תחום שאילתה מלבני נתון. נגדיר את גבולות המלבן:

```
int X_High = 9;
```

```
int X_Low = 3;
```

```
int Y_High = 8;
```

```
int Y_Low = 4;
```

בשלב הראשון נבדוק שאכן נמצא הצומת V_{split} (הצומת המפצל את מסלול החיפוש של Y_{Low} ו- Y_{High} בעץ התחומים כמתואר קודם בעבודה זו) בצורה תקינה ע"י קריאה לפונקציה:

```
root_range_tree = GetSplitNode (&root_range_tree, Y_Low, Y_High);
```

כאשר $root_range_tree$ מצביע לצומת V_{split} . ע"י שינוי הערכים של גבולות המלבן נבדקו כל האפשרויות למיקום של V_{split} (כאשר הצומת המפצל הוא בשורש, בצומת פנימי בעץ או בעלה) ובכולם חישוב הצומת המפצל היה תקין.

לאחר חישוב הצומת המפצל יש לבדוק שאכן האלגוריתם מוצא את כל הנקודות בתחום השאילתה הן בעצי המשנה (RPST) בעלי תכונת ערימת מינימום והן בעצי המשנה בעלי תכונת ערימת מקסימום ע"פ קואורדינטה y .

הבדיקה הזו מתבצעת ע"י הקריאה הבאה:

```
FindPointsInRectangle (&root_range_tree, X_Low, X_High, Y_Low, Y_High, lowX, upX);
```

פונקציה זו עוברת על עץ המשנה בעל תכונת ערימת מקסימום ע"פ קואורדינטה y המוצבע מהבן השמאלי של $root_range_tree$ (הצומת המפצל) וכן עוברת על עץ המשנה בעל תכונת ערימת מינימום ע"פ קואורדינטה y המוצבע מהבן הימני של הצומת המפצל. ע"י קריאה לפונקציות EnumerateRectYUpperBound ו-EnumerateRectYLowerBound

EnumerateRectYLowerBound הפונקציה מוצאת את הנקודות הנמצאות בתחום השאילתה המלבני :
[X_Low,X_High]x[Y_Low,Y_High].

תוכנית הבדיקה הורצה גם עם אוסף הנקודות ותחום השאילתה המלבני המופיע ב-[7] בדומה לתיאור מעלה ולבסוף פלט תוכנית הבדיקה הוא :

First Query:

(4,6)

(4,7)

(4,5)

(7,5)

(8,4)

Second Query:

(35,42)

(27,35)

כאשר התוצאה הראשונה מתייחסת לאוסף הנקודות ותחום השאילתה שתואר מעלה והתוצאה השנייה מתייחסת לאוסף הנקודות ותחום השאילתה מ-[7]. שתי התוצאות הינן תקינות ומתאימות לפלט המצופה.

בדיקות יחידה למודל Data Reader

מודול זה קורא את המידע המגיע מהעולם החיצוני ומעבירו לחלקים אחרים במערכת.

אחד התפקידים של Data Reader הוא לקרוא את רשומות הנקודות המגיעות מקובץ מסד הנתונים ולשלוח מהן את החלקים השונים של המידע (הקואורדינטות של הנקודה והשם שלה). הנקודות שנקראו מועברות לאלגוריתם מיון Quick Sort לקראת הכנסתן לעץ החיפוש הדו-מימדי (ראה תיאור מעלה), כמו-כן הנקודות ושמות שדות התעופה נשמרים גם במבנה TgtDistance (ראה פירוט בסעיף 5.4) שתפקידו הוא לשמור את המרחקים מהמטוס לשדות התעופה.

הוגדר קובץ בדיקה המכיל 6 נקודות בדיקה כקלט. תוכן קובץ הבדיקה הוא :

153 240 airport1

38 49 airport2

553 600 airport3

700 468 airport4

990 100 airport5

930 110 airport6

ע"י קריאה לפונקציה ReadTargetsPosition אכן ניתן היה לראות שמהקובץ נשלפו הנקודות:
(930,110),(990,100),(700,468),(553,600),(38,49),(153,240) וכמו-כן הנקודות ושמות שדות התעופה
נשמרו בצורה נכונה במבנה המרחקים.

תפקיד נוסף של המודול Data Reader הוא בניית רשימת התחומים המלבניים. מספר המלבנים וגבולותיהם
נקבעים ע"פ גודל (רזולוציית) המסך, גודל המלבנים וכן ע"פ הגדרה של חפיפה או אי-חפיפה ביניהם. ע"י
קריאה לפונקציה RectanglesBuilder תוך מתן פרמטרים שונים של גודל ורוחב המלבנים וכן קביעת
חפיפה/אי-חפיפה נבדקה רשימת המלבנים המוחזרת ע"י הפונקציה הנ"ל. רשימת המלבנים הייתה כמצופה.
הבדיקה נעשתה על-פני מסכים בעלי רזולוציות שונות (1024x768 פיקסלים וכן 1366x768 פיקסלים שהינן
רזולוציות סטנדרטיות במסכים ניידים ומסכי מחשבים ניידים בהתאמה).

תפקיד נוסף של מודול זה הוא למצוא את רשימת המלבנים בהם נמצאת נקודה מסוימת. הבדיקה של פעולה
זו נבדקה יחד עם בדיקת בניית המלבנים. במהלך הבדיקה הוגרלו נקודות בדיקה אקראיות וע"י קריאה
לפונקציה PointLocate נבדקו אילו מלבנים מתוך רשימת המלבנים הוחזרו בתור המלבנים המכילים את
נקודת הבדיקה (במקרה של חפיפה ושל אי-חפיפה). התוצאות של הפונקציה תאמו את רשימת המלבנים
שנבחרה ואת נקודות הבדיקה האקראיות.

בדיקות יחידה למודול המייצר את מיקום המטוס

תפקידו של מודול סימולציית המסלול הוא להגדיל נקודה התחלתית של מיקום המטוס הנמצאת באחד
הצדדים של המסך (שמאל או ימין), בתוך גבולות התצוגה, וכן להגדיל את הנקודה הבאה של מיקום המטוס
אשר יכול להתקדם בצורה אקראית באחד מ-5 כיוונים (כמתואר בסעיף 5.1). במקרה שהמטוס יצא מגבולות
התצוגה על המודול להודיע על כך.

בבדיקות של מודול זה נקראה תחילה הפונקציה SelectStartPoint אשר בוחרת את מיקום המטוס ההתחלתי
אשר מצידה קוראת לפונקציה SelectStartSide הבוחרת את צד המסך (שמאל/ימין) של תחילת הטיסה.

בבדיקות נראה שהצד ההתחלתי נבחר בהסתברות בערך שווה, כמו-כן נבדק שאכן מיקום המטוס ההתחלתי
נמצא בצד שנבחר ובתוך גבולות המסך כך שלא יתכן שטיסה תתחיל מנקודה שאינה נראית למשתמש.

תפקיד מרכזי של מודול זה הוא להבטיח שהטיסה תתקדם בכיוון הצד המנוגד לצד ההתחלתי וסימולציית
הטיסה תסתיים. על מנת שהטיסה תסתיים נבדק שאכן בכל רגע נתון הנקודה המוגרלת היא נקודה המקרבת
את המטוס בציר האופקי לכיוון המנוגד של המסך. בציר האנכי לא תיתכן תנועה למעלה ואח"כ למטה או
להיפך על מנת למנוע "תקיעה" של המטוס, תנועה בכיוון חדש בציר האנכי חייבת להיות מוגרלת אחרי תנועה
בציר האופקי.

לצורך הבדיקות הנ"ל נקראה הפונקציה SelectNextPlatformPosition עד שהמטוס יצא מגבולות המסך.
בבדיקות נראה שאכן המטוס מתקדם בציר האופקי לכיוון הצד המנוגד לצד התחלת התנועה ובכיוון האנכי
אין "תקיעה" של המטוס. מתוך עשרות טיסות שנבדקו, לא הייתה אף טיסה שלא הסתיימה – לבסוף כל
טיסה יצאה מגבולות המסך מהצד האופקי הנגדי לנקודת ההתחלה או מאחד הגבולות האנכיים של המסך.

6.2 בדיקות אינטגרציה

במהלך בדיקות האינטגרציה בוצעה בדיקה משולבת של כל המודולים במערכת ותפקודם כמקשה אחת המספקת את דרישות המערכת.

בבדיקות האינטגרציה טוענים מפה עם מסד של שדות תעופה המתאימים לאותו איזור טיסה ומפעילים את סימולציית הטיסה. התוצאה הנדרשת היא – תצוגה של שדות התעופה במלבן (או המלבנים) של המטוס בכל רגע נתון, אם יש כאלה.

נתאר את השלבים העיקריים בבדיקות האינטגרציה :

1. בחר תפריט Load Map והעלה מפה של איזור הטיסה לתצוגה.
2. בחר תפריט Load Targets והעלה את קובץ מסד הנתונים המתאים למפה.
3. בתפריט קונפיגורציה קבע גודל רצוי של מלבנים ובחר את אופציית החפיפה.
4. הפעל את סימולציית הטיסה תחת תפריט Operational/Run.
5. בדוק תצוגה של מטוס ואת כיוון ההתקדמות.
6. ודא שבכל מלבן (או מלבנים) בו נמצא המטוס, מוצגים כל שדות התעופה הממוקמים באותו מלבן בצבע ירוק.
7. ודא שכאשר המטוס עובר ממלבן למלבן, שדות התעופה של המלבן הישן לא מוצגים.
8. ודא שכאשר המטוס נמצא באיזורי החפיפה מוצגים שדות התעופה מכל המלבנים החופפים.
9. בדוק שסימולציית הטיסה מסתיימת.

הבדיקות שתוארו בסעיפים 6.1 ו-6.2 לא מכסות את בדיקות כל המערכת, אלא רק את המרכיבים החשובים ביותר שלה. כמובן שמלבד הבדיקות הנ"ל נעשו בדיקות מקיפות לממשק הגרפי ונבדק שכל תפריט אפשרי מבצע את הנדרש ממנו.

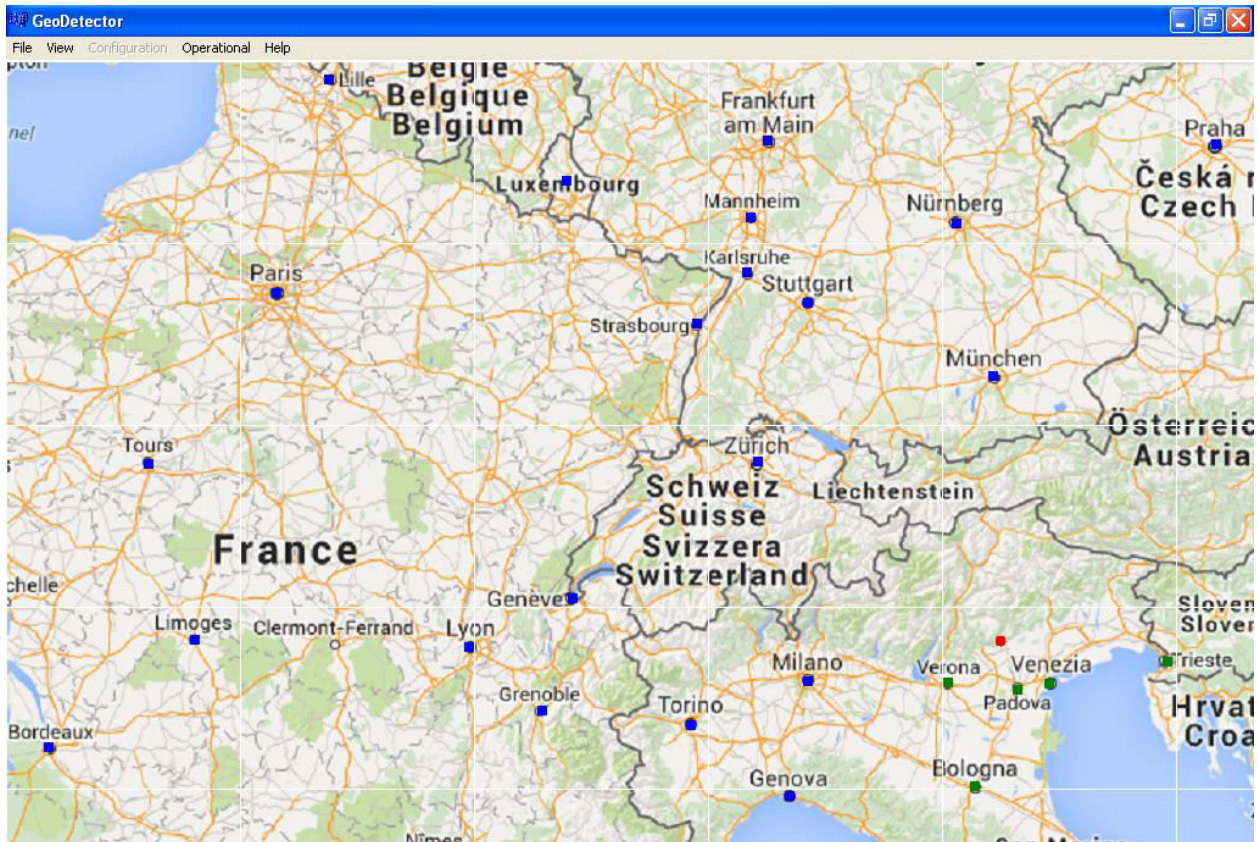
7. מקרי בוחן

נראה כמה מקרי בוחן ודוגמאות ריצה של המערכת. לצורך הדוגמה הראשונה נשתמש במפה של מרכז אירופה שנלקחה מ-[9]. הוגדר מסד נתונים של למעלה מ-30 שדות תעופה המתאימים למפה זו. מסד הנתונים הוגדר בקובץ חיצוני שנקרא ע"י המערכת.

נתבונן בשני מקרים – תחילה נראה דוגמה לריצה של המערכת עם תצורת מלבנים ללא איזורי חפיפה (כאשר המטוס עובר ממלבן למלבן מופיעים רק שדות התעופה של המלבן החדש) ואח"כ נראה דוגמה לתצורה שהוגדרו בה איזורי חפיפה בין מלבנים (כאשר המטוס עובר ממלבן למלבן, באיזור החפיפה, מוצגים שדות התעופה של המלבן החדש והישן כל עוד המטוס לא יוצא מאיזור החפיפה).

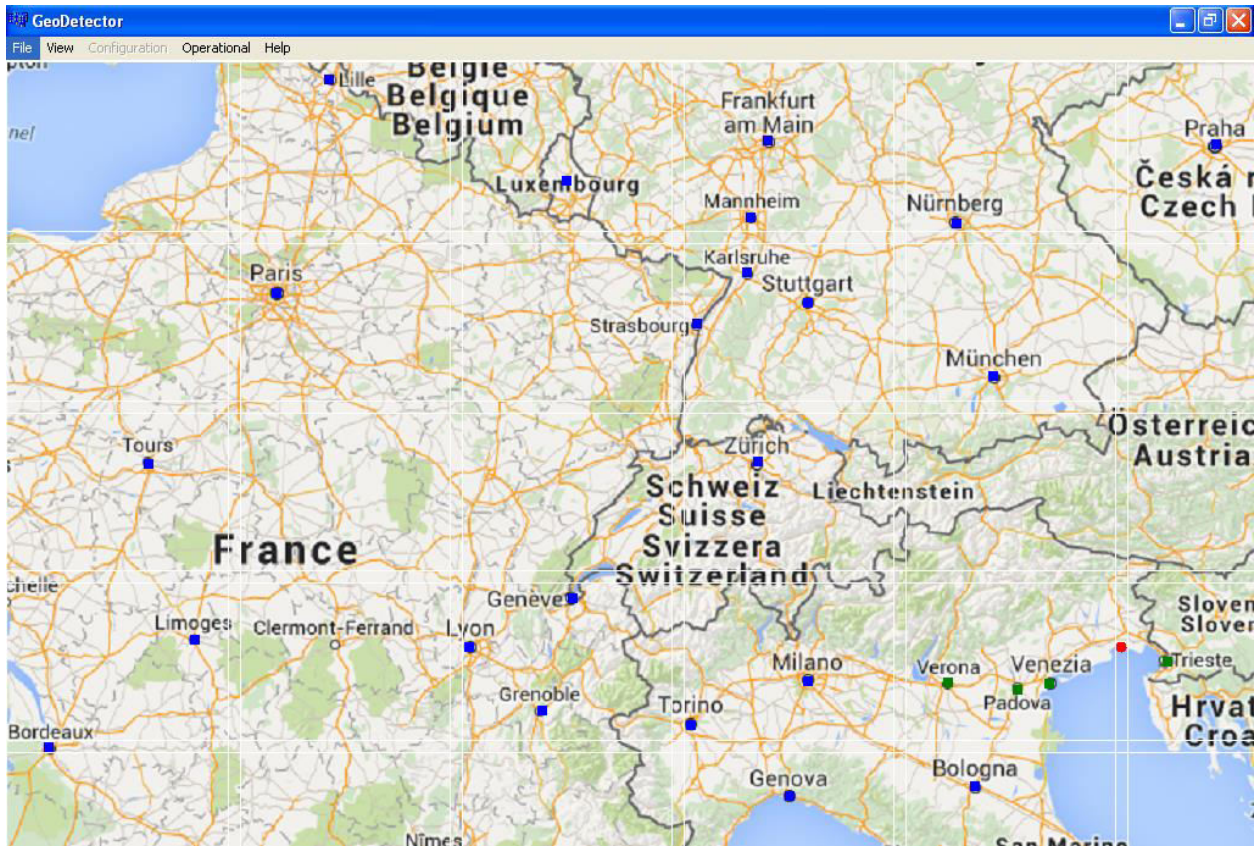
על מנת להציג תמונה מלאה, בדוגמאות הריצה נבחרה האופציה להציג את כל שדות התעופה ממסד הנתונים על המפה (בצבע כחול) לצד שדות התעופה הנמצאים במלבן (או המלבנים) של המטוס (בצבע ירוק).

להלן דוגמת ריצה בקונפיגורציה של מלבנים ללא איזורי חפיפה .



איור 11 - תצורת מלבנים ללא חפיפה. המטוס (באדום) נמצא במלבן שבו חמישה שדות תעופה הצבועים בירוק

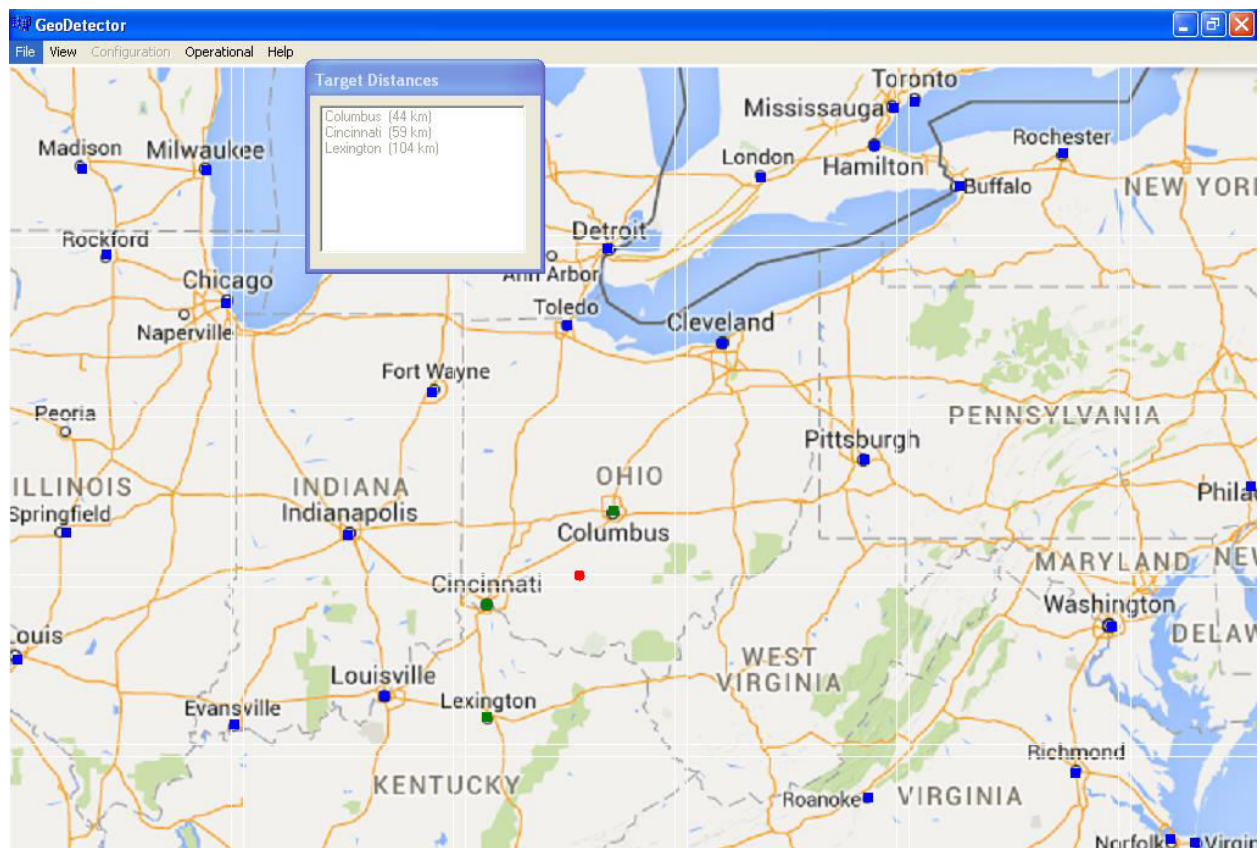
להלן דוגמת ריצה בתצורה של מלבנים עם איזורי חפיפה עם אותה מפת טיסה.



איור 12 - המטוס נמצא באיזור חפיפה של שני מלבנים. מוצג שדה תעופה אחד מהמלבן הימני ושלושה מהשמאלי

ניתן לראות כי בשני המקרים (עם וללא חפיפה) האלגוריתם מוצא את הנקודות הנמצאות במלבן או במלבנים של המטוס.

נראה עוד דוגמת ריצה. נשתמש במפה של אמריקה הצפונית מתוך [9], גם כאן הוגדר מסד נתונים של למעלה מ-30 שדות תעופה המתאימים למפה זו. נשתמש בתצורה עם חפיפת מלבנים. כמו-כן נציג את המרחקים של המטוס מכל שדות התעופה הנמצאים במלבנים בהם הוא נמצא.



איור 13 - המטוס נמצא באזור חפיפה. מוצג שדה תעופה אחד מהמלבן העליון ושניים מהתחתון (מוצגים המרחקים לשדות התעופה)

8. סיכום והצעות לפיתוחים עתידיים

במסגרת פרויקט זה הוצג מימוש של חיפוש יעיל של נקודות בתחומי שאילתה מלבניים במישור. בתחילת הפרויקט נחשף האתגר של חיפוש יעיל של נקודות כלשהן מתוך אוסף גדול של נקודות ביישומים העושים שימוש בנתונים גאוגרפיים. לאחר-מכן הוצג הרקע התיאורטי לעבודה שבו ערכנו היכרות עם מבני הנתונים והאלגוריתמים המהווים את הבסיס להתמודדות עם אתגר החיפוש היעיל.

לאחר ההצגה העיונית הוגדרה המטרה של הפרויקט. הוגדרו דרישות הפרויקט וכן נסקרו סביבת הפיתוח וארכיטקטורת התוכנה שנבחרה לצורך מימוש הפרויקט. נסקרו מבני הנתונים שבהם משתמשים לצורך שמירת הנקודות ממסד הנתונים ומימושם וכן אלגוריתם החיפוש בנקודות אלה ומימושו. כמו-כן נסקר אופן קריאת מסד הנקודות מקובץ חיפוש והמרתו, נסקר אופן הקביעה של תחומי שאילתה המלבניים וכן אופן הסימולציה של הטיסה. בנוסף נסקר הממשק הגרפי ואופן השימוש בו לצורך הצגת שדות התעופה בתחום השאילתה שבו נמצא המטוס על מפה אמיתית.

לבסוף הוצגו הבדיקות של המערכת לצורך הוכחת נכונות דרך הפעולה של המערכת. הבדיקות כללו בדיקות יחידה שנועדו לצורך בדיקת תקינות המודולים הנפרדים עד רמת הפונקציות כאשר הן אינן קשורות לשאר המערכת. בהמשך הוצגו בדיקות אינטגרציה אשר בדקו את כלל פעילות המודולים המשולבים במערכת. כמו-כן הוצגו מקרי בוחן אשר בדקו את אופן פעולת המערכת על מפות אמיתיות, שדות תעופה אמיתיים ומסלול מטוס אקראי שיכול להיות זהה למסלול מציאותי.

המערכת מהווה בסיס לעבודות המשך אשר יכולות להרחיב ולשפר את המערכת. על מנת שלמערכת יהיה שימוש מעשי ניתן להשמיט את מודול סימולציית הטיסה של המטוס ע"י כך שהוא יוחלף בנתוני ניווט אמיתיים בזמן אמת (כגון מערכת GPS). כתוצאה מחיבור למערכת ניווט אמיתית ניתן להשמיט את הדרישה לטעינת מפת איזור הטיסה ע"י המשתמש, אלא להחליף אותה במנגנון טעינות מפה אוטומטי כך שהמפה של האיזור בו נמצא המטוס נטענת ע"פ נתוני הניווט בזמן הריצה. ניתן אף לשנות את אופן קריאת מסד הנקודות כך שלא יקרא מקובץ קבוע אלא שישלף ממסד נתונים שנמצא על שרת אינטרנטי (מה שיאפשר למסד הנתונים להיות דינמי ולהתעדכן ע"י כל המורשה לכך בזמן אמת).

המערכת גם יכולה להוות בסיס למערכות גאוגרפיות מחוץ לעולם התעופה כגון מערכות ניווט לכלי רכב. על מנת להרחיב את המערכת לעולם כלי הרכב יש צורך בהוספת מבני נתונים ואלגוריתמים נוספים אשר מלבד מציאת נקודות בתחומי שאילתה, צריכים להיות מסוגלים למצוא גשרים, כבישים או מסילות ברזל הנמצאים באיזורי שאילתה או החותכים אותם. כמו-כן צלעות תחומי שאילתה אינם חייבים להיות מקבילים לצירים.

מבני הנתונים והאלגוריתמים הנדרשים לצורך מימוש הפעולות הנ"ל יסקרו בהרחבה בעבודתי המסכמת לתואר שני בנושא "מבני נתונים גאומטריים לחיפוש יעיל".

9. ביבליוגרפיה

- [1] M. de Berg, O.Cheong, M. van Kreveld, M.Overmars, "Computational Geometry: Algorithms and Applications", 3rd edition, Springer-Verlag, Berlin 2008.
- [2] E. M. McCreight, "Priority Search Trees", SIAM J. Comput., Vol.14, 1985, pp. 257-276.
- [3] T.D. Lee, "Interval, Segment, Range, and Priority Search Trees", in: Handbook of Data Structures and Applications, edited by D.P.Mehta and S.Sahni, Chapter 18, Chapman and Hall / CRC Press, 2004.
- [4] P. K. Agarwal, "Segment, Interval, Priority-Search Trees", CPS234, Lecture 7, Computational Geometry, Duke University, 2005
- [5] R. Tamassia, "Priority Search Trees", Part I, CS252, Computational Geometry, 1993.
- [6] R. Tamassia, "Priority Search Trees", Part II, CS 252, Computational Geometry, 1994.
- [7] H. Samet, "Range Trees and Priority Search Trees", Institute For Advanced Computer Studies, University of Maryland, 1998.
- [8] P.Gupta, R.Janardan, M.Smid, B. Dasgupta, "The rectangle enclosure and point-dominance problems revisited," In: Proc. 11th Annual Symposium on Computational .Geometry 1995, pp.162-171.
- [9] <https://www.google.com/maps>
- [10] <http://www.yevol.com/bcb>
- [11] B. Chazelle and L.J. Guibas, "Fractional Cascading I: A Data Structuring Technique", Algorithmica1(3), 1986, pp.133-162.
- [12] B. Chazelle and L.J. Guibas, "Fractional Cascading II : A Data Structuring Technique", Algorithmica1(3), 1986, pp.163-191.

10. נספחים

10.1 נספח א: רשימת איורים

- איור 1 - עץ חיפוש חד מימדי, מסלולי החיפוש מודגשים באפור. מסלולי החיפוש של X ושל X' מסתיימים בעלים μ ו- μ' בהתאמה..... 5
- איור 2 - תיאור עץ תחומים דו-מימדי עם עץ ראשי T ועצים משניים T_{assoc} המוצבעים מכל צומת בעץ הראשי 6
- איור 3 - החלוקה בכל שלב לפי x נעשית ע"פ נקודת האמצע של התחום [LowX,UpX] של השלב הקודם..... 9
- איור 4 - תיאור המערכת..... 11
- איור 5 - דיאגרמת בלוקים של המערכת..... 14
- איור 6 – עץ תחומים ראשי, כל צומת פנימי מצביע לעץ עדיפויות משני. הנקודות בעץ הראשי שמורות בעלים המסומנים בשחור..... 22
- איור 7 - המסך הראשי. המלבנים מסומנים בלבן, המטוס מסומן כעיגול אדום, שדה התעופה במלבן שלו מסומן בירוק..... 27
- איור 8 - מסך קונפיגורציה לבחירת גדלי מלבנים..... 30
- איור 9 - מפת מלבנים עם איזורי חפיפה אופקית ואנכית..... 31
- איור 10- מסך מרחקים ממוינים של נקודות במלבן בו נמצא המטוס..... 32
- איור 11 - תצורת מלבנים ללא חפיפה. המטוס (באדום) נמצא במלבן שבו חמישה שדות תעופה הצבועים בירוק..... 41
- איור 12 - המטוס נמצא באיזור חפיפה של שני מלבנים. מוצג שדה תעופה אחד מהמלבן הימני ושלושה מהשמאלי..... 42
- איור 13 - המטוס נמצא באיזור חפיפה. מוצג שדה תעופה אחד מהמלבן העליון ושניים מהתחתון (מוצגים המרחקים לשדות התעופה)..... 43

10.2 נספח ב: קוד פרויקט

דיסק עם קוד המקור המלא של הפרויקט מצורף לספר הפרויקט.

10.3 נספח ג: התקנת המערכת

דרישות המערכת: מערכת הפעלה WindowsXP ומעלה.

התקנת המערכת:

1. העתק מהדיסק את הקובץ GeoDetector.zip.
2. צור ספריה כגון C:\GeoDetector.
3. בצע unzip של הקובץ לתוך הספריה הנ"ל.
4. קובץ ההפעלה של התוכנה הוא GeoDetect.exe והוא נמצא בספריה הנ"ל.

10.4 נספח ד: הצעת פרויקט

שם פרויקט: תוכנה לחיפוש יעיל של אובייקטים בתחומים גאומטריים

שם מנחה: ד"ר ג'ק וינשטין

מטרת הפרויקט

פיתוח חבילת תוכנה אשר תוכל למצוא נקודות בתוך תחומים גאומטריים נתונים בצורה יעילה. התוכנה תחזיק בסיס נתונים (המוגדר בקובץ) של נקודות במישור וכן אוסף של תחומים גאומטריים המחלקים מפה מסוימת. בהינתן גוף נע על פני המפה, התוכנה תמצא בצורה יעילה את אוסף הנקודות מתוך בסיס הנתונים הנמצאות בתוך התחום הגאומטרי שבהן נמצא הגוף הנע בכל רגע נתון. הנקודות שנמצאו בתחום הגאומטרי יוצגו בכל רגע נתון. התוכנה תציג ממשק GUI שבעזרתו יוצג הגוף הנע, התחומים והנקודות שנמצאו.

רקע עם התייחסות למקורות

בשנים האחרונות חיפושים גאוגרפיים הפכו לנפוצים עד מאוד בחיי היומיום. דוגמה בסיסית לכך היא שהרבה מכוניות מצוידות היום בצגים המציגים את הסביבה הגאוגרפית של מיקום המכונית כגון תחנות דלק קרובות, מצלמות מהירות, ערים בסביבה ועוד. דוגמאות נוספות לכך הן תוכנות כמו סימולטורי טיסה אשר צריכות להציג לטייס את האובייקטים שנמצאים בטווח הראיה שלו ועוד [1].

כמות המידע הגאוגרפי הכולל היא עצומה, מה שמעניין את המשתמש הוא הצגה של חלק קטן מתוך כל המידע הגאוגרפי, אותו חלק הנמצא בסביבת המכונית, המטוס וכו'.

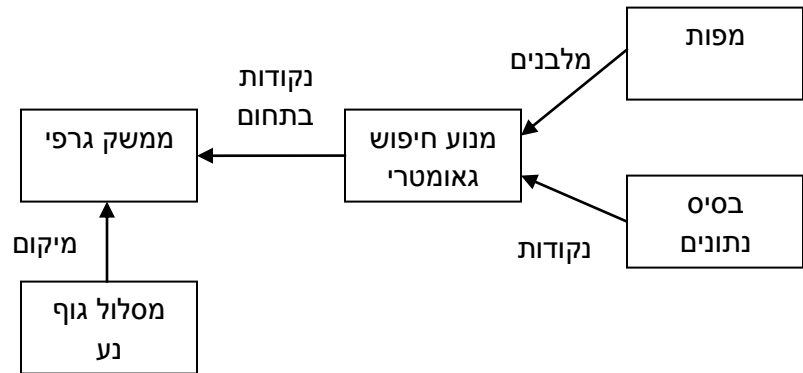
את הסביבה ניתן לייצג באמצעות מלבנים ([1],[2],[3],[8]). כאשר הגוף הנע (למשל, מכונית) נמצא במלבן מסוים, מוצגים האובייקטים שנמצאים באותו מלבן בלבד. כאשר עוברים מאיזור (מלבן) אחד למשנהו מתחלפת התצוגה באובייקטים הנמצאים באיזור החדש. בעיה זו נקראת בעיית windowing (מכיוון שיש להציג רק את האובייקטים הנמצאים בתוך ה"חלון" סביב הגוף הנע). למלבנים יכולים להיות אזורי חפיפה צרים בגבולות כך שהגוף הנע יכול להימצא במעבר מאזור לאזור ביותר ממלבן אחד (בשלב זה יוצגו הנקודות בשני המלבנים).

מכיוון שכמויות המידע הגאוגרפי הן גדולות, יש צורך בחיפוש יעיל של האובייקטים אותם מעוניינים למצוא בתחומים המלבניים ([1],[2]). בעשורים האחרונים הוצעו מבני נתונים חדשים אשר שימושיים לצורך פתרון בעיות גאומטריות מהסוג שתואר מעלה, קיימים אלגוריתמים יעילים אשר מאפשרים, תוך שימוש במבני הנתונים האלה, לבצע חיפוש יעיל הן מבחינת זמן השאילתה והן מבחינת סיבוכיות המקום.

אחד מהמבנים הראשונים שהוצעו לצורך חיפוש נקודות (או מקטעים) בתוך תחומים מלבניים הוא עץ חיפוש עם קדימויות (priority search tree), מבנה נתונים זה מהווה שילוב של עץ חיפוש מאוזן וערימה ([2],[4],[5],[6],[7]). ומאפשר למצוא תוך שימוש בזמן שאילתה של $O(\log n + k)$ וסיבוכיות מקום של $O(n)$ את הנקודות הנמצאות בתחום מלבני מסוים (כאשר n היא כמות נקודות הקלט ו- k היא כמות הנקודות המדווחות כפלט).

תיאור הפרויקט

ארכיטקטורה של הפרויקט :



הפרויקט יפותח בסביבת Borland C++ Builder ויכתב בשפת C.

החלקים העיקריים של הפרויקט הם :

- מפות – המפות מחולקות לתחומים מלבניים.
- בסיס נתונים – בסיס הנתונים יורכב מקובץ אשר מחזיק את כל הנקודות .
- מנוע חיפוש גאומטרי – ימומש מבנה נתונים שהוא סוג של עץ חיפוש עם קדימויות (Radix Priority Search Tree) כמוזכר ב-[2].
- הנקודות מבסיס הנתונים ותחומי השאילתה (המלבנים) יזונו למבנה הנתונים והוא ימצא את הנקודות הנמצאות בתוך המלבן.
- מסלול גוף נע – יגדיר את המיקום של הגוף הנע (למשל מכונית או מטוס) על המפה (ובפרט את המלבן או המלבנים בהם הוא נמצא).
- הממשק הגרפי יציג את הגוף הנע, את המפה עם המלבנים, כמו-כן הנקודות במלבן/מלבנים שבו נמצא הגוף הנע יוצגו על המפה. כאשר הגוף הנע יעבור למלבן אחר, תוצגנה הנקודות שבמלבן החדש (באזורי החפיפה תוצגנה הנקודות של שני המלבנים).

תיאור תהליכים עיקריים :

- בשלב הראשון תיבנה מפה עם חלוקה אקראית למלבנים דומים בגודלם.
- יוגדר בסיס נתונים של נקודות אשר יוחזק בקובץ.
- עבור גוף נע מסוים (למשל גוף המייצג מכונית או מטוס) יוגדר מסלול כלשהו.
- הגוף יוצג בתנועה לאורך המסלול שהוגדר עבורו.
- פעם ביחידת זמן תתבצע דגימה של מיקום הגוף, מנוע החיפוש הגאומטרי ימצא את הנקודות הנמצאות במלבן או המלבנים בהם נמצא הגוף ונקודות אלה תוצגנה על פני המפה.
- עדכון הנקודות יתבצע באופן דינמי בהתאם לתנועת הגוף (כלומר במעבר ממלבן למלבן תוצגנה הנקודות הנמצאות במלבן החדש או בשני מלבנים באזור החפיפה).

חשיבות הפרויקט

בשנים האחרונות הולך וגובר השימוש בנתונים גאוגרפיים בזמן אמיתי. מכיוון שכמות המידע הגאוגרפי הינה גדולה מאוד ומכיוון שבכל רגע נתון רק כמות קטנה של אותו מידע רלוונטית למשתמש חשוב לבצע חיפוש יעיל על המידע אשר יחזיר את האובייקטים הרלוונטיים בתוך זמן אופטימלי.

חבילת התוכנה הנוכחית נותנת מענה לצורך הנ"ל ע"י שימוש באחד ממבני הנתונים היעילים ביותר הקיימים למציאת נקודות בתוך איזורים מלבניים.

לפרויקט פוטנציאל לשימוש בחבילות תוכנה של מכוניות להן חיבור ל-GPS ובסיס נתונים מתאים ע"י כך שלנהג תוחזר תשובה מהירה לשאלתה שהוא יכול לבקש (כגון אוסף כל תחנות הדלק באזור בו הוא נמצא, כל מצלמות הדרכים הנמצאות בקרבתו, כל הערים באזור הגאוגרפי ועוד).

ניתן להשתמש בפרויקט גם בסימולטורי טיסה או במטוסים בכך שהוא יכול לשמש, למשל, למציאת אוסף כל שדות התעופה באזור הגאוגרפי בו נמצא המטוס.

חבילת התוכנה הנוכחית תיכתב בשפת C שתהיה, ללא ספק, בשימוש עוד שנים רבות מה שיקל על תחזוקת ועדכון התוכנה במקרה הצורך וכן על היבילות (portability) שלה.

שיטות לביצוע הפרויקט

1. איסוף חומר רלוונטי על מבני נתונים בגאומטריה חישובית.
2. קריאה על שיטות שונות למציאת נקודות בתוך תחומים מלבניים.
3. ניתוח השיטות השונות למציאת נקודות באפליקציות גאוגרפיות.
4. הגדרת בסיס נתונים של אוסף נקודות בקובץ טקסט.
5. הגדרת המפות וחלוקתן למלבנים.
6. מימוש מנוע החיפוש הגאומטרי המבוסס על Radix Priority Search Tree.
7. עיצוב ומימוש הממשק הגרפי. הממשק הגרפי ישתמש בספריות גרפיות של חבילת התוכנה Borland C++ Builder 6.
8. אינטגרציה של כל המודולים.
9. בדיקות של כל המערכת ע"י הרצות עם מסלולים שונים ומפות שונות.

לוח זמנים של הפרויקט

מטלה	משך ביצוע (חודשים)
איסוף החומר ועיצוב הפרויקט	3
מימוש הפרויקט	3
בדיקות הפרויקט	1
כתיבת עבודה (תיק הפרויקט)	1