# טופס הצעת פרויקט- י"ג הנדסת תוכנה:

**סמל מוסד:** 659284

**שם מכללה:** בארי WAN TEC

**שם הסטודנט:** גפן נגה

**ת.ז הסטודנט:** 326670692

**שם הפרויקט:** Zombie Pursuit

**שפת תכנות :** C++

**סביבת עבודה:** Visual Studio 2022

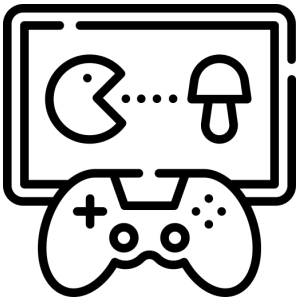
**חתימת הסטודנט:**

**חתימת מנחה המגמה:**

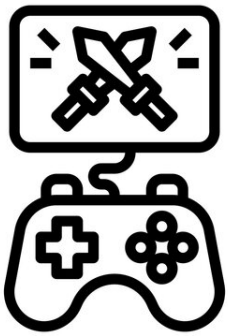
**אישור משרד החינוך:**

## תוכן עניינים:

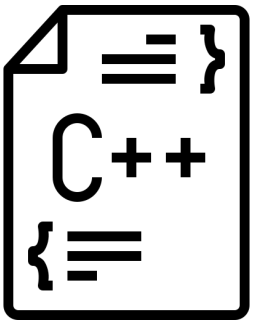
# רקע תיאורטי בתחום הפרוייקט:



עם התפתחות המחשבים, נכנס מונח חדש לתודעה העולמית- משחקי מחשב. משחקי המחשב הבסיסיים הראשונים היו משחק "איקס עיגול" ומשחק "טניס לשניים". בשנת 1972, יצא המשחק הראשון שיפעל על מסך הטלוויזיה- PONG של חברת אטארי. לראשונה הוא גם כלל צלילים ואפקטים קוליים. את "פונג" מחשיבים רבים כחלוץ משחקי הווידאו האמיתי, וודאי משחק הטלוויזיה הראשון בהיסטוריה. בשנים הבאות, משחקי וידאו ביתיים הפכו יותר ויותר פופולריים והחליפו את מכונות הארקייד המיושנות, בעוד שיצאו משחקים מתקדמים יותר, קונסולות עם יכולות גרפיות, תצוגת תלת מימד, סאונד מתקדם, יכולות תכנות ועוד. מאז ועד היום, משחקי מחשב הפכו למוצר פופולרי ומוכר בקרב כל הגילאים ונוער בפרט.



ז'אנר אחד של משחקי המחשב הוא משחק אקשן. משחק פעולה/אקשן הוא שם כולל לסוגת משחקי וידאו המתמקדת ביכולות ביצוע גבוהות ואתגר פיזי. בדרך כלל, הדרישות מהשחקן הן תזמון, מהירות תגובה וקואורדינציה. משחקי אקשן לרוב מכילים יריות, מכות ולחימה. משחקים אלה מופרדים לגוף ראשון (השחקן מביט על העולם מעיניו של הדמות) או גוף שלישי (השחקן מביט על הדמות מאחוריה למעלה). היתרון הוא במשחקי גוף שלישי בהם לשחקן יש טווח ראייה רחב יותר ויכולת להגיב לאירועים מהר יותר.




על מנת, בין היתר, ליצור ולממש משחקים אלה על המחשב, פותחו שפות תכנות וממשקים אשר תומכים ביכולות גרפיות, השמעת אודיו ועוד. אחת השפות היא ++C- שפת תכנות שהומצאה בשנת 1979 ומטרתה הייתה ליצור שפת C עם מחלקות- שילוב של השפה simula (שפה מונחית עצמים אך איטית) ביחד עם שפת C (שפה מהירה ויעילה יותר). שפת ++C הינה שפה המשלבת כמה מודלים תכנותיים- תכנות פרוצדורלי (חלוקה לפעולות), תכנות מונחה עצמים (חלוקה למחלקות) ובתכנות גנרי (יצירת templates כלליים של מחלקות).

כמו כל שפה מונחית עצמים, גם ++C תומכת בעקרונות הכימוס (חלוקת מידע למחלקות והסתרת מידע), ירושה (הכלת תכונות משותפות בין מחלקות) ופולימורפיזם (כתיבת קוד אבסטרקטי אשר ממומש בכל תת מחלקה בצורה שונה) על מנת למנוע חזרות קוד, להקל על המתכנת ולפשט את מבנה התוכנה. היתרון המרכזי של שפת ++C לעומת השפות מונחות העצמים האחרות, הוא

הקרבה והתאימות לשפת C- גישה ישירה לזכרון המחשב, יעילות ומהירות- קומפילציה ישירה של קוד לשפת מכונה, לעומת שפות אחרות שממירות את הקוד לקוד ביניים עבור ה- Virtual machine.
על מנת לפתח משחקים בפלטורמה מסויימת, אנו זקוקים לממשק- המספק גישה לכרטיס הזיכרון של המחשב, כרטיס הרשת, המקלדת והעכבר. ממשק בו אני אשתמש הוא DirectX. ממשק זה, הפותח בידי Microsoft מיועד לסביבת Windows ומטרתו לתת למתכנת גישה ישירה לחומרת המחשב למטרת פיתוח משחקים. בנוסף, ממשק זה מנגיש למתכנת גרפיקה דו ותלת מימדית בקלות וביעילות. החבילות המרכזיות של DirectX:

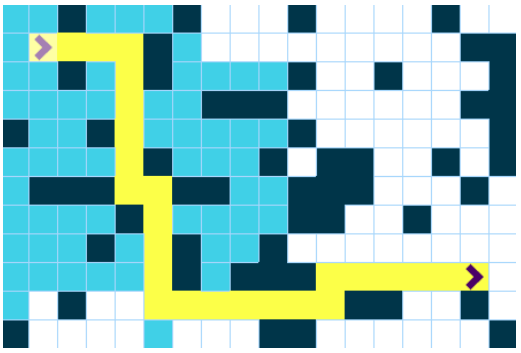
DirectDraw: גישה לכרטיס המסך של המחשב

DirectPlay: גישה לכרטיס הרשת של המחשב

DirectSound: גישה לכרטיס הקול של המחשב

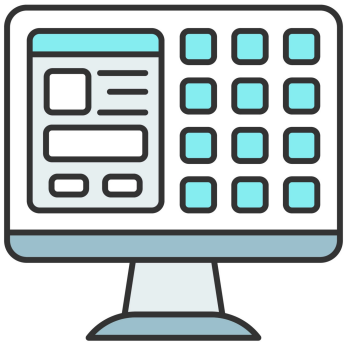
DirectInput: גישה למקלדת שמחוברת למחשב

במשחק האקשן אותו אני אצור נדרש מכל דמות להגיע בזמן הכי קצר לאובייקטים שונים בזירה (הסבר מורחב בהמשך). על מנת שכל דמות תוכל לבצע זאת, יש להשתמש באלגוריתם שימצא לה את המרחק הכי קצר- **האלגוריתם A*.**



האלגוריתם A* הינו אלגוריתם חיפוש פופולרי הפותח בשנת 1968 באוניברסיטת סטנפורד ומטרתו היא למצוא מרחק מינימלי בין נקודת המקור לנקודת היעד. האלגוריתם עובד על גרפים משוקללים, ופועל בצורת סריקה לרוחב (BFS)- מעבר על כל צמתי הגרף בהתאם למרחקם מצומת המקור. בנוסף, A* הינו אלגוריתם הפועל על פי שיטת חיפוש **היוריסטית**- שיטה המבוססת על הערכה וניחוש של הכדאיות לעבור מצומת לצומת.

היתרון המרכזי של שיטה זו הוא צמצום אזור החיפוש של פתרון לבעיה- בעזרת הערכת הכדאיות, ובכך קיצור זמן החיפוש שנצרך. בנוסף, השיקולים ההיוריסטיים של האלגוריתם אינם רק מורכבים מחישוב מספרי של הכדאיות של מעבר לצומת הבא, אלא גם על פונקציות היוריסטיות- המתאימות לכל מצב נתון ציון המייצג את השיקול ההיוריסטי, ובכך קובע האם לעבור לצומת הבא או לא.
לעומת זאת, שיטה היוריסטית אינה מבטיחה הצלחה- שיטה זו הינה פרקטית אך לא מבטיחה פתרון אופטימלי, אולם מספקת תנאים עבור מציאת פתרון מיידי. אם פתרון מיידי לא נמצא, השיטה תכוון למצוא פתרון סביר, בתור אלטרנטיבה. אולם, אין ערובה לכך שימצא פתרון. עקב כך, אלגוריתם A* הוא שלם- הרצת האלגוריתם מבטיחה שימצא הפתרון בהנחה שיש פתרון אפשרי. חסרון נוסף של האלגוריתם הוא שחישוב הפונקציות ההיוריסטיות גובה משאבים בזמן ההרצה שמשפיעים על הכדאיות של השימוש באלגוריתם.



ממשק משתמש גרפי (**G**raphical **U**ser **I**nterface) הינו ממשק משתמש עבור תוכנה או אתר אינטרנט המבוסס על עיצוב גרפי של המסך המוצג למשתמש. בעבר, כאשר המחשבים היו דלים באמצעים, נהגו להשתמש בממשק טקסטואלי בלבד (הצגת המידע באמצעות אותיות) בנוסף למקשי המקלדת. עם התפתחות הטכנולוגיה נפוץ הממשק הגרפי- הכולל חלונות, כפתורים, תפריטים ואייקונים. כמובן שהממשק הגרפי עדיף על הטקסטואלי, לאור התצוגה הגמישה יותר בצבעים, גופנים ועיצובים, אינטראציה קלה ושימושית והבהרת כוונת הממשק והשימוש בו. מאוחר יותר נוספה השימוש בעכבר המרחיב את האפשרויות של הצגת הגרפיקה והקלט מהמשתמש, שפותח שנים לאחר מכן כמסך המגע בטלפונים חכמים ועוד. בהנדסת תוכנה, נחוץ שממשק משתמש גרפי יהיה בנוי כראוי, כדי ליצור הפרדה בין רכיבי המשתמש בתוכנה לבין רכיבים אחרים ובכך לנטר את סוג המידע שמופיע למשתמש ולהקל על עריכת שינויים במשק המשתמש.

בפרויקט שלי, ממשק המשתמש הגרפי מורכב מ:

**<u>תפריט ראשי:</u>** התפריט הראשי של המשחק יורכב מכמה כפתורים שהם:

- **יצירת משחק חדש:** המשתמש יוכל להתחיל משחק חדש ולהגדיר את המרכיבים בו לפי רצונו: כמות הבוטים שיוצבו בזירה, תכונות של כל בוט (מהירות, כוח, צבע) והכמות של **עדר הזומבים**- כמות הזומבים שמוכנסים לזירה.
- **טעינת משחק קיים:** המשתמש יוכל להמשיך משחק שהוא התחיל ושמר בעת היציאה ממנו. אם לא קיים משחק שמור, יותחל משחק חדש במקום.
- **קריאה של כללי המשחק:** המשתמש יוכל לבחור לקרוא את כללי המשחק בלחיצת הכפתור "איך משחקים?" ולאחר מכן לשוב לתפריט הראשי.
- **יציאה.**

בעת שהות המשתמש בתפריט הראשי או באחד התפריטים שנגזרים ממנו, תופעל מוזיקת רקע ותהיה תמונת רקע.

**<u>מהלך המשחק:</u>** יפורט בהמשך.

**<u>מקרים מיוחדים:</u>** במהלך המשחק ישנם מספר מקרים בהם ממשק המשתמש משתנה:

- **עצירת המשחק:** מוזיקת המשחק תיעצר, ויופיע למשתמש תפריט של אפשרויות: המשכת המשחק הקיים, יציאה לתפריט הראשי או יציאה מהמשחק.
- **מצבי ניצחון הפסד:** מוזיקת המשחק תיעצר ותופיע על המסך הודעה האם הוא ניצח או הפסיד.

# תיאור הפרויקט:

המשחק מורכב מכמה אלמנטים:

- משחק מחשב שמטרת השחקן היא להרוג כמה שיותר זומבים אשר זזים אקראית ברחבי הזירה. באותה הזירה, מוצבים בוטים אשר מטרתם זהה- עליהם להרוג כמה שיותר זומבים כדי לנצח. בנוסף, עליהם לגבש אסטרטגיה- למצוא את הזומבי הכי קרוב, להגיע ל- powerups ולנסות לעכב את השחקן או בוטים אחרים.
- במשחק קיימים כמה סוגים של זומבים כאשר הריגת כל אחד מהם מזכה בניקוד שונה.
- בתחילת המשחק המשתמש יוכל לבחור את כמות הבוטים ואת התכונות שלהם- צבע, דרגת קושי (מהירות, חוזק).

בנוסף תהיה אפשרות למשתמש לבחור את הכמות של עדר הזומבים, אשר יווצרו במקומות אקראיים בקצב מסויים.

- בזירה יהיו האובייקטים הבאים:
  - זומבים שיוצבו אקראית בזירה.
  - בוטים שונים בהתאם להחלטת המשתמש.
  - מכשולים שונים שמקשים על התנועה של השחקנים.
  - שדרוגים (powerups) שמוצבים אקראית בזירה ומעניקים:
    - פרץ עוצמה לזמן קצר- מעניק למכה של השחקן יותר כוח.
    - פרץ מהירות לזמן קצר.
    - אקסטרה כמות מסויימת של חיים.
- המשחק נגמר כאשר עדר הזומבים כולו הוכנס לזירה ונהרגו כל הזומבים בתוך הזירה.
- המשחק הוא דו-מימדי במבט על.

## מטרות השחקן:

המשחק שאותו אני בונה הוא משחק single player מורכב אשר בכל רגע נתון לשחקן כלשהו (משתמש, בוט) ישנם כמה מטרות אותן הוא צריך לתעדף- המספור מסמן עדיפות :

1. **הישרדות:** להתחמק מהשחקנים האחרים. בסופו של דבר המטרה המרכזית של שחקן היא לשרוד, אחרת הוא מפסיד ויוצא מהמשחק.
2. **הזומבי הכי קרוב:**

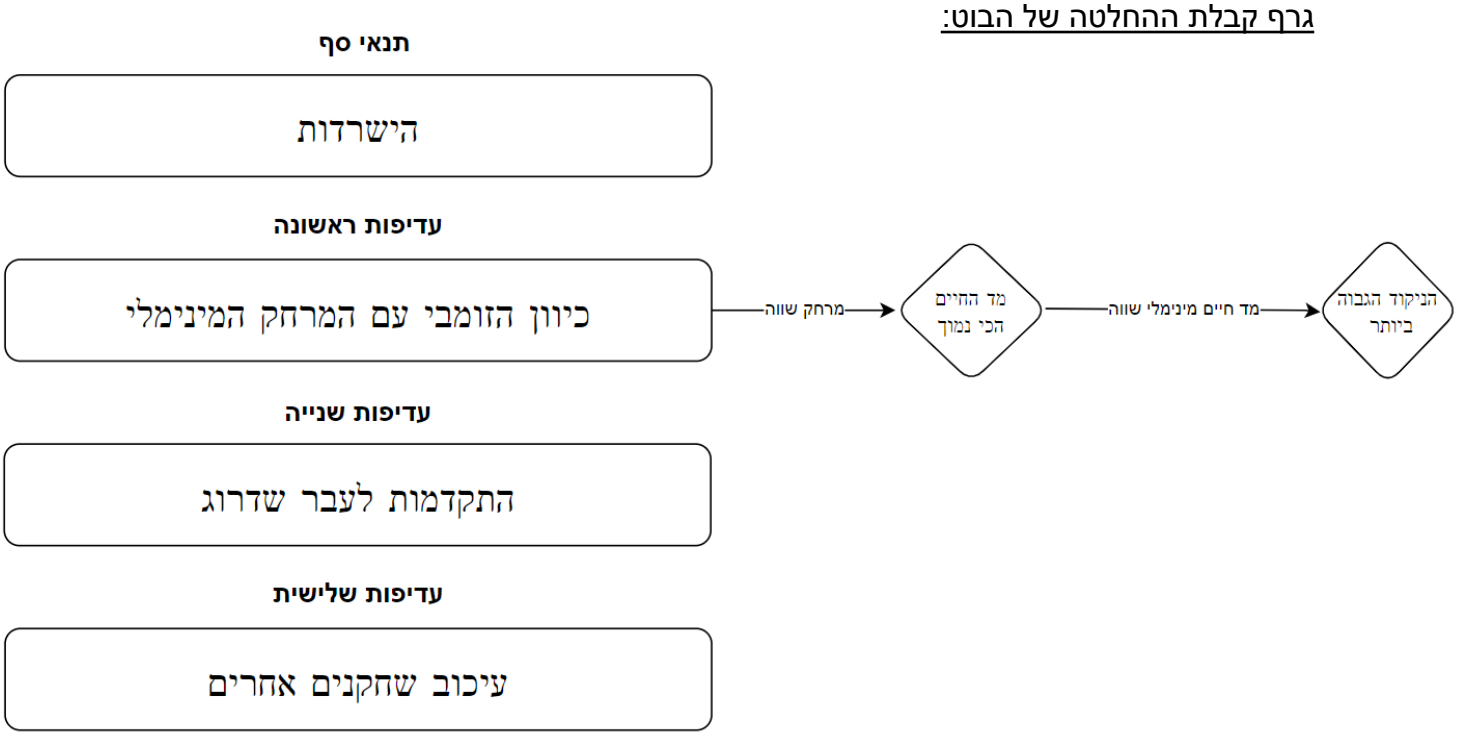
המטרה הראשונית היא ללכת לזומבי הכי קרוב, אך אם ישנם כמה זומבים במרחק דומה (בטווח מסויים):

   2.1. הזומבי עם מד החיים הכי נמוך.

   2.2. הזומבי המזכה בניקוד הגבוה ביותר- מטרת השחקן היא להגיע לסוף המשחק עם הניקוד הגבוה ביותר, לכן זוהי אמורה להיות עדיפות שלו.

3. **התקדמות לעבר powerup:** חלק מהאסטרטגיה של השחקן היא לשפר את עצמו כדי שיהיה במצב טוב יותר לאסוף יותר ניקוד.
4. **עיכוב שחקנים אחרים:** לא הכרחי עבור ניצחון במשחק, אבל אלמנט נוסף באסטרטגיה הוא להכות שחקנים אחרים כך שיצטרכו לברוח מהשחקן או ללכת לכיוון של powerup שיעניק להם עוד חיים.

## גרף קבלת ההחלטה של הבוט:



אופן התנועה של השחקן:

התקדמות השחקן מוחלטת בצורה זו- עבור כל אחד מהכיוונים האפשריים של השחקן, השאלה האם השחקן נמצא בסכנה (טווח מכה של שחקן אחר) הינה תנאי סף עבור התנועה של השחקן. תחת חשיבה זו נגדיר את אופן התנועה של השחקן ונבטא אותו באמצעות **פסאודו קוד**:

1. נבדוק את המסלול בו המרחק בין השחקן לבין זומבי הוא מינימלי.
2. אם יש יותר ממסלול מינימלי אחד, בצע:
    2.1. מצא את המסלול שמוביל לזומבי עם מד החיים הכי נמוך.
        2.1.1. אם יש יותר מזומבי אחד עם מרחק מינימלי ומד חיים שווה, בצע:
            2.1.1.1. מצא את המסלול שמוביל לזומבי עם הניקוד הכי גבוה.
            2.1.1.2. אם המיקום הבא במסלול הנבחר אינו מציב את השחקן בסכנה, תתקדם לשם.
        2.1.2 אם יש רק מסלול מינימלי אחד, תתקדם בכיוונו.
3. אם המיקום הבא במסלול הנבחר אינו מציב את השחקן בסכנה, תתקדם אליו.
4. אם הוא כן, בצע:
    4.1. מצא את המסלול בו המרחק בין השחקן לבין שדרוג הוא מינימלי.
    4.2. אם המיקום הבא במסלול הנבחר אינו מציב את השחקן בסכנה, תתקדם לשם.
    4.3. אם הוא כן, בצע:
        4.3.1. מצא את המסלול בו המרחק בין השחקן לבין שחקן אחר הוא מינימלי.
        4.3.2. התקדם למיקום הבא במסלול הנבחר.
5. הישאר במקום.

## תהליכים עיקריים בפרוייקט:

- יצירת תפריט ראשי שמכיל אפשרות של כניסה למשחק, טעינת משחק קיים, קריאה של חוקי המשחק ויציאה.
- המשחק עצמו- רקע, מוזיקה, מכשולים, אנימציות הדמויות, מימוש התנהגות הזומבים.
- השמה של התכונות של הבוטים בהתאם לבקשות המשתמש.
- מימוש האסטרטגיה וקבלת ההחלטות של הבוטים בהתאם למצב הזירה בכל רגע נתון.
- מקרים מיוחדים- מצבי ניצחון/הפסד, מצב של עצירת המשחק וחזרה לתפריט הראשי, יציאה מהמשחק.

## הגדרת הבעיה האלגוריתמית:

מטרתו של כל שחקן בזירה היא להרוג כמה שיותר זומבים. עבור כך, עליו למצוא את הזומבי שהמרחק בינו לבין השחקן הוא מינימלי, כדי ליצור אסטרטגיה יעילה ולנצח בסופו של דבר את המשחק. עליו לחשב מרחק זה ומרחקים אחרים במקרים מסוימים (כפי שמתואר לעיל) כל פעם שהוא נדרש לקבל החלטה- כלומר לאחר כל תנועה קודמת שלו בהתאם לתנאים בזירה.
האלגוריתם האופטימלי עבור פתירת בעיה זו הינו *A- אלגוריתם למציאת דרך מינימלית בין שני אובייקטים דרך מכשולים וביעילות עדיפה על אלגוריתמים אחרים.

כפי שתואר לעיל, *A הינו אלגוריתם חיפוש היוריסטי המוצא את המסלול הכי קצר (אם קיים) בין צומת מקור לצומת יעד בגרף משוקלל, ומשתמש בתור עדיפויות ממוין (מיישם לוגיקת תור אך מבוסס גם על קוד עדיפות). בכל ביקור בצומת אחת במהלך הרצת האלגוריתם, הוא מנסה להעריך את המסלול הזול ביותר בעזרת שימוש ב**פונקציית עלות**- שילוב של מידע ידוע (המרחק בין צומת המקור לצומת הנוכחית) ופונקציה היוריסטיות (הערכה של המרחק בין הצומת הנוכחית לבין צומת היעד) הקובעת לאיזה צומת עדיף לעבור.

האלגוריתם עובד באמצעות שתי רשימות של צמתים- רשימה פתוחה (צמתים שלא נחקרו באופן מלא) ורשימה סגורה (צמתים שנחקרו באפון מלא).
בכל צעד, האלגוריתם בוחר את הצומת מהרשימה הפתוחה עם פונקציית העלות הקטנה ביותר וחוקר אותו. חקירת הצומת כוללת הוספה שלה לרשימה הסגורה ומעבר לצמתים הבאים בהתאם לחישוב פונקציית העלות מצומת זו. האלגוריתם ממשיך לחקור צמתים עד שנמצא צומת היעד או שנקבע שאין מסלול אפשרי אל צומת היעד. אם נמצא הצומת, *A מסוגל לסגת (backtrack) באמצעות הרשימה הסגורה ולמצוא את המסלול הקצר ביותר.

## אלגוריתם *A- פסאודו קוד:

1. Create an empty list called ***closedSet*** that will store the nodes that have already been evaluated, and an empty list called ***openSet*** that will store the nodes that are discovered but not yet evaluated. Add the start node to ***openSet***.

2. Create a map called ***cameFrom*** that will store the most efficient previous step for each node, a map called ***gScore*** that will store the cost of getting from the start node to that node, and a map called ***fScore*** that will store the total cost of getting from the start node to the goal by passing by that node. Set the value of ***gScore*** for the start node to 0 and the value of ***fScore*** for the start node to the heuristic cost estimate of getting from the start node to the goal.

3. While ***openSet*** is not empty, do:
    a. Set the node with the lowest ***fScore*** value in ***openSet*** as the current node.
    b. If the current node is the goal, return the path to the goal by reconstructing it using the ***cameFrom*** map.
    c. Remove the current node from ***openSet*** and add it to ***closedSet***.
    d. For each neighbor of the current node, do:
        i. If the neighbor is in ***closedSet***, ignore it.
        ii. Calculate a tentative ***gScore*** for the neighbor by adding the cost of getting from the current node to the neighbor to the ***gScore*** of the current node.
        iii. If the tentative ***gScore*** is less than the current ***gScore*** for the neighbor, update the ***gScore*** for the neighbor and set the value of ***cameFrom*** for the neighbor to the current node. Set the value of ***fScore*** for the neighbor to the ***gScore*** for the neighbor plus the heuristic cost estimate of getting from the neighbor to the goal.
        iv. If the neighbor is not in ***openSet***, add it to ***openSet***.
4. If the goal is not reached and ***openSet*** is empty, return "failure".

5. To reconstruct the path to the goal, create a list called ***totalPath*** and add the current node to it. Then, while the current node is in ***cameFrom***, set the current node to the value of ***cameFrom*** for the current node and add it to ***totalPath***. Finally, return ***totalPath***.

## לוחות זמנים:

לוח הזמנים של ביצוע החלקים של הפרויקט, בהערכה כללית (רוב התאריכים אינם מדויקים):

| שלב | תאריך |
|---|---|
| הגשת הצעת פרויקט | 20.12.2022 |
| מחקר על השפה ועל יצירת משחקים בשפה | 4.1.23 |
| תכנון מבנה כללי (מחלקות, תבניות) ולמידת העבודה עם DirectX | 22.1.23 |
| תכנון מחלקות סופיות ו- UML | 10.2.23 |
| בניית המשחק הפשוט- תפריט, התחלת המשחק, אנימציה ויצירת הזומבים והשחקן | 20.3.23 |
| יצירת הבוטים (רק מימוש תכונותיהם המבוקשות) | 5.4.23 |
| מימוש *A עבור הבוטים | 20.4.23 |
| שיפור ועיצוב ממשק המשתמש, תיקון תקלות, התחלת כתיבת ספר פרויקט, הוספת רעיונות שיעלו במהלך הכתיבה, הוספת מורכבויות | 30.4.23 |
| הגשת ספר פרויקט | 1.5.23 |
| בחינה | 15.5.23 |