עבודת גמר

לקבלת תואר

טכנאי תוכנה

הנושא: סימולטור לימוד נהיגה של מכונית – AlphaCar

המגיש: יותם לויט

מנחים: מיכאל צ'רנובילסקי ואלון חיימוביץ

מאי 2020 תש"ף

AlphaCar

תוכן עניינים

הצעת פרויקט סיום הנדסת תוכנה

ואלון חיימוביץ שמות מנחים: מיכאל צ'רנובלסקי
שם סטודנט: יותם לויט ת"ז 200041119

<u>נושא הפרויקט</u>: סימולטור מכונית ה"לומדת" לנהוג.

רקע תיאורטי

תיאור האפליקציה:

כיום יש תחרויות רובוטיקה ובהן קבוצות ילדים המתחרות והן בונות רובוטים אוטונומיים כמו כן חברות העובדות על תוכנה למכונית אוטונומית. גם קבוצות הרובוטיקה וגם בחברות אלה בונות מודל אמיתי של מכונית או רובוט כדי לבדוק את הפיתוח, ועלותו הכללית של המודל היא גדולה.
הסימולטור נועד למזער את עלות הפיתוח של מודל המכונית האוטומטית בכך שהוא נותן לחברות סביבה ווירטואלית לבדיקת בינת המכונית.
הסימולטור הוא תוכנה המריצה קוד של מכונית אוטונומית בסביבה ממוחשבת, בין אם מדובר בלמידה בין אם מדובר באלגוריתם טכני. התוכנה מאפשרת לחברות לאתר טעויות בקוד או ללמוד במקרה של למידה של המערכת לנהיגה בשלב ראשיתי, קודם לשלב המודלים האמיתיים שאם יתנגשו ויהרסו יאבד הרבה כסף. הודות לסימולטור תוכלנה חברות להריץ את האלגוריתם בסימולטור, לחסוך זמן - אם מדובר בלמידה, וכסף - אם מדובר בטעות באלגוריתם.
בפרויקט זה קוד המכונית יהיה "אלגוריתם מסובך" והמכונית תהיה בעלת חיישנים למדידת מרחק מעצמים.

מהלך הלמידה:

ותיצור מספר רב של מכוניות. לכל אחת יש את Genetic algorithmהלמידה תיעשה בשיטת ה-" ייחודי לכל מכונית. האלגוריתם יבחר את DNAהמערכת הנוירונים שלה עם ערכים שיוצרים " " אחד, וייצור מספר רב DNA" שלהם לתוך "DNAהמכוניות בעלות הביצוע הטוב ביותר, ישלב את ה" של "מוטציות" חדשות מתוך השילוב.

הפעלת אלגוריתם המכונית:

הפעלת האלגוריתם תבוצע במסלולים מתוכננים, אשר אינם מוכרים לאלגוריתם.

במהלך הלמידה המכונית תלמד להגיב לאופציות השונות הקיימות בנהיגה.

הגדרת האלגוריתם:

של המשחק. המשחק יקבל את "APIבמסך ההפעלה יוכל המשתמש לתת קובץ המופעל עם "

יוכל להריצו. APIהקובץ ובעזרת ה

מטרות הפרויקט:

1. מימוש רשת הנוירונים.
2. מימוש אלגוריתם שנותן ציון לכל "DNA" של מכונית.
3. בניית מערכת שבוחרת את המכוניות המצטיינות, משלבת את ה"DNA" שלהם לתוך "DNA" אחד ויוצרת מספר רב של המוטציות שלו.
4. יצירת ממשק משתמש, אתו יוכל המשתמש לתקשר עם התוכנה.
5. יצירת סביבה תלת-ממדית, שבה האלגוריתם יורץ עם מסלולים ללמידת המכוניות.

דרישות מערכת:

1. מימוש רשת נוירונים
   1.1. רשת בעלת 3-4 שכבות.

2. מימוש אלגוריתם שנותן ציון לכל "DNA" של מכונית
   2.1. הציון יקבע לפי אחוזי התנגשות, שרידות ה"DNA" ומרחק נסיעה

3. יצירת ממשק משתמש, אתו יוכל המשתמש לתקשר עם התוכנה.
   3.1. ממשק אינטואיטיבי ונעים לשימוש.

4. יצירת סביבה תלת-ממדית, שבה האלגוריתם יורץ עם מסלולים ללמידת המכוניות.
   4.1. מסלולים מתוחכמים ללמידה יעילה יותר
   4.2. נדרש כי המסלול לא יהיה סלעי

משאבים:

עמדת פיתוח:

פיתוח המערכת נעשה בסביבות העבודה – Microsoft .NET Framework, תוך כדי שימוש בMicrosoft Visual Studio 2019. כמו כן נעשה שימוש במנוע המשחק Unity 2019.2.8f1 ובעורך שלו, תוך כדי שימוש במנוע משחק זה ישנו גם שימוש בDirect x 11 של חברת מיקרוסופט.

עמדת משתמש:

- האפליקציה מתאימה לכל מחשב עם מערכת ההפעלה Windows שמותקן בו Unity.

לוח זמנים:

היעד	תאריך היעד
- הגשת טיוטה של הצעת לפרויקט	5/11/2019
- הגשה הצעת פרויקט סופית	20/11/2019
- תכנון המערכת ופירוק למודולים. - חשיבה על כל מודול וחשיבה על ארכיטקטורת המערכת.	7/12/2019
- תכנון סכמתי של ממשק המשתמש	10/12/2019
- פיתוח אב טיפוס ובנייתו. בשלב זה יסודות ייבוצרו	10/1/2020
- מימוש ממשק המשתמש והרחבת המערכת	10/2/2020
- בדיקת תקינות התוכנה וגם Debugging. שיפורים לממשק המשתמש.	1/3/2020
- כתיבת תיק פרויקט ושינויים קלים בממשק ובמערכת	1/4/2020
הגשת הפרויקט	1/5/2020

1. מבוא

1.1 מטרה

פיתוח סימולטור נהיגה ב-*C#* עם מנוע משחק *Unity* אשר מדמה את העולם האמיתי ותכונותיו. נוסף על כך, בניית מודל של אינטליגנציה מלאכותית תוך כדי שימוש ברשת נוירונים בכדי לאפשר למחשב ללמוד תוך כדי ניסיון הכולל הצלחה וכישלון לנהוג במכונית.

1.2 תיאור המערכת:

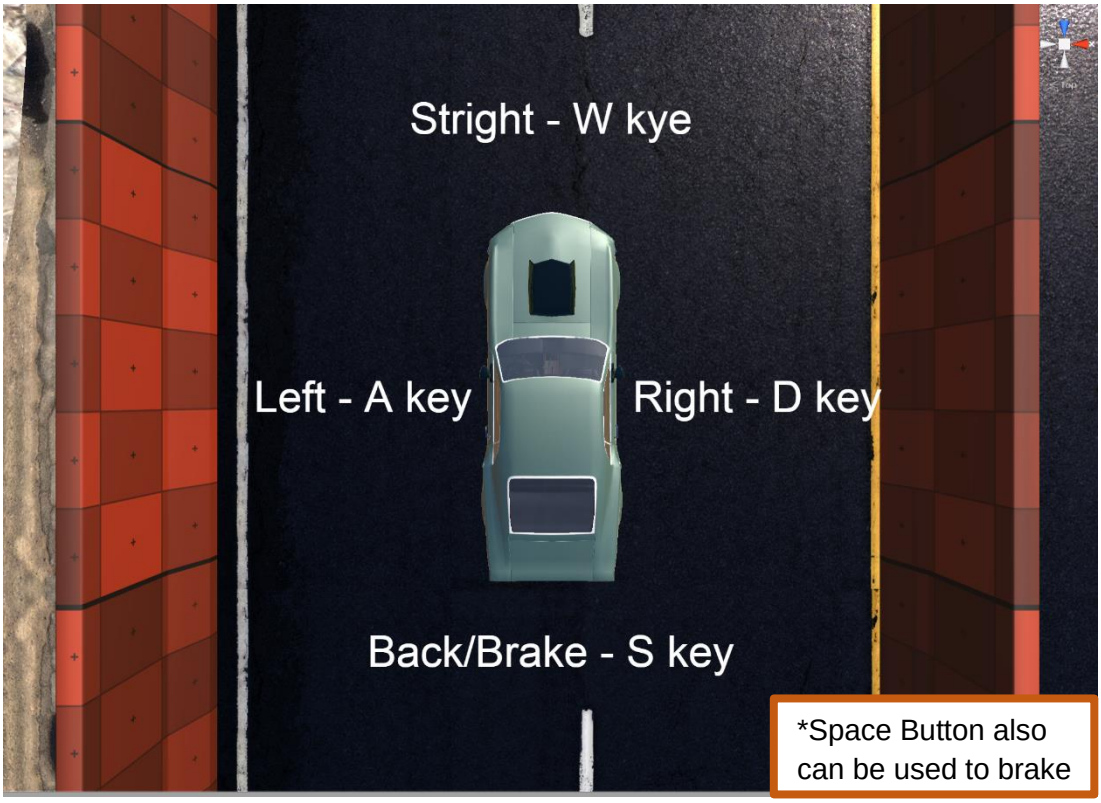
1.2.1 תיאור הסימולטור:

הסימולטור הינו סימולטור אשר מדמה נהיגה של מכונית בעולם הפיזי בתוך תוכנת מחשב.
הסימולטור כולל את חוקי התנועה של ניוטון, אווירודינמיקה וגיאומטריית ההיגוי ברכב אשר כוללת זווית שפיעה – **Camber**, קדם אופן – **Caster**, נטיית ציר היגוי – **Steering Axis Inclination**, כינוס גלגלים והתבדרותם – **Toe-in,Tow-out**, וזווית אקרמון – **Ackerman Steering Principle**.

למטרת הפרויקט נבנה מסלול מיוחד בו המשתמש יפעיל את המכונית.

1.2.2 אפשרויות המשתמש:

למשתמש ארבעה כיווני תנועה בו יוכל לשלוט על שתי צירי **X** ו **Y** – ימינה, שמאלה, ישר ואחורה להלן תמונה במבט על המתארת את כיווני התנועה בו המשתמש יוכל לשלוט:



1.2.3 מטרת המשתמש:

מטרת השחקן היא להגיע כמה שיותר רחוק במסלול ובדרך המהירה ביותר שאפשר תוך כדי להימנע מהריסת המכונית ומהתנגשותה בקירות המסלול.

1.3 שפת התכנות ופירוט סביבת העבודה והכלים:

- שפת תכנות: בחרתי ב-#C לאורך כל הקוד שלי משום ששפה זו מחולקת בשיטת OOP כמו כן למנוע המשחק Unity יש תמיכה בשפה. דברים זה מאפשר גמישות בשיטות הפעולה ומתן אפשרויות רבות להרחבת הפרויקט.
- חלוקה למחלקות בשיטת OOP: שיטה זו יוצרת מדרגיות ושינויים בה קלים יותר. בנוסף OOP מספקת פשטות וארגון במהלך התכנות.

1.3.1 דיון בנושא העיצוב הנבחר:

- שפת תכנות: בחרתי ב-#C לאורך כל הקוד שלי משום ששפב זו מחולקת בשיטת ה OOP כמו כן למנוע המשחק Unity יש תמיכה בשפה זו. דבר זה מאפשר גמישות בשיטות הפעולה ומתן אפשרויות רבות להרחבת הפרויקט.
- חלוקה למחלקות בשיטת OOP: שיטה זו יוצרת מדרגיות ושיוניים בה קלים יותר. בנוסף OOP מספרת פשטות וארגון במהלך התכנותץ

1.3.2 סביבת פיתוח:

פיתוח המערכת נעשה בסביבות העבודה – **Microsoft .NET Framework**, תוך כדי שימוש ב**Microsoft Visual Studio 2019**. כמו כן נעשה שימוש במנוע המשחק **Unity 2019.2.8f1** ובעורך שלו, תוך כדי שימוש במנוע משחק זה ישנו גם שימוש ב**Direct x 11** של חברת מיקרוסופט.

2. מפרטי תוכנה

2.1 תיאור כללי:

- בתוך הסימולטור נמצא מסלול בו המשתמש ישלוט המכונית, מטרת המשתמש היא להגיע כמה יותר רחוק וכמה שיותר מהר בתוך המסלול.

להלן המסלול במפת הסימולטור:

- בכל רגע נתון (כמו במציאות) המשתמש יוכל לשלוט במכונית ולשנות את מסלולה לדוגמה המשתמש נהג במכונית ישר והחליט שרצונו לעצור או לפנות ימינה. להלן תמונה בה המכונית שנשלטת על ידי המשתמש משנה את מסלולה מישר לימינה:

- פסילה נקראת כאשר המכונית מתנגשת בקיר וכאשר זה קורה המכונית עוצרת והמשחק מתחיל מהתחלה. להלן פסילה:

2.2 ניסוח וניתוח הבעיה האלגוריתמית:

בפרויקט פיתחתי אלגוריתם אשר משתמש/משחק בסימולטור. כלומר אלגוריתם של מחשב שלומד לנהוג במכונית.

בכדי לנהוג במכונית יש צורך בהבנה של כמה דברים – כאשר נוהגים המפעיל צריך לדעת כיצד לשלוט במכונית כמו כן המפעיל צריך לנהוג בתוך סביבה משתנה ולכן יש את הצורך שידע כיצד להבין את הסביבה לעבד אותה ולפעול על פי השנויים שקרו בה. בנוסף היה צורך ליצור סביבה ווירטואלית בה הסימולטור יתקיים. מכאן נבעו מספר בעיות אלגוריתמיות:

הבעיה האלגוריתמית:

1. <u>ייצוג סימולטור ווירטואלי:</u>
כאמור המערכת היא הסביבה ווירטואלית (סימולטור) בו המשתמש (אדם או אלגוריתם) יפעלו בסביבה. כמו כן הסביבה צריכה לתמוך בחוקי הפיזיקה של העולם האמיתי ולדמות במחשב את מה שהיה קורה אילו הסביבה הייתה במציאות.

2. <u>סיבוכיות, מורכבות ונוחות הסביבה:</u>
הסביבה שתוצג בסימולטור צריכה להיות מסובכת ומורכבת בכדי להוכיח קושי בנסיעה במכונית בסימולטור ובנוסף היא צריכה להיות נוחה לשימוש ומובנת.

3. <u>ייצוג הסימולטור תוך שמירה על יעלות וריצה חלקה של הקוד:</u>
הסימולטור בו המכונית תהיה חלק ממנו יצרוך הרבה המשאבים ולכן ישנה הבעיה שבפיתוח לא נכון הסימולטור יתקע את המערכת.

4. <u>החלטת צעד על פי סביבה משתנה (איזה אלגוריתם):</u>
אלגוריתם הAI צריך לקבל מידע משתנה מהסביבה ולהגיב לכל מקרה בפני עצמו. כלומר הסביבה המשתנה במצב מסוים והאלגוריתם יצטרך לחשב מתוך מצב הסביבה את הצעד הכי טוב לעשות.

2.3 פיתוח הפתרון ויישומו:

פתרון הבעיות:

1. <u>ייצוג סימולטור ווירטואלי:</u>

למטרת יישום הדרישות של הסימולטור בחרתי לפתח את הסביבה במנוע משחק מוכן בשם Unity. למנוע יש אפשרות לחישובים פיזיקאליים בסיסיים וכמו כן אפשרות קלה לאינטגרציה של חישובים פיזיקאליים ספציפיים שפותחו מחוץ למנוע וברצוני להוסיף.

2. <u>סיבוכיות, מורכבות ונוחות הסביבה:</u>

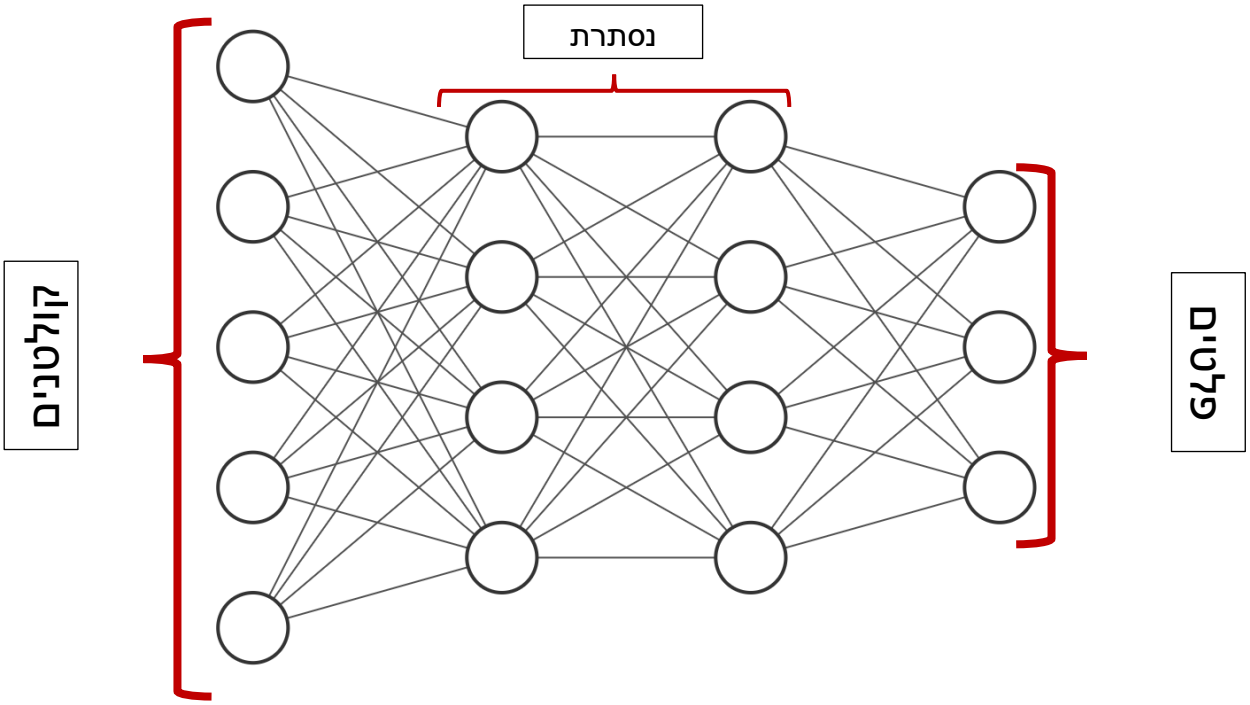
בכדי לסבך את הסימולטור, המנוע Unity מאפשר פיתוח סביבה בתלת ממד ובכך להוסיף עוד ממד בו המשתמש צריך להתרכז בו בנוסף לכך Unity מאפשר הפעלה קלה וממשק ידידותי.

3. <u>ייצוג הסימולטור תוך שמירה על יעילות וריצה חלקה של הקוד:</u>

בכך שחולטת על שימוש במנוע משחק מוכן (Unity) הוא דואג לפיצול הצריכה ובשימוש הDirectX 11 API וכך שומר לבד על פיצול המשאבים ביניהם.

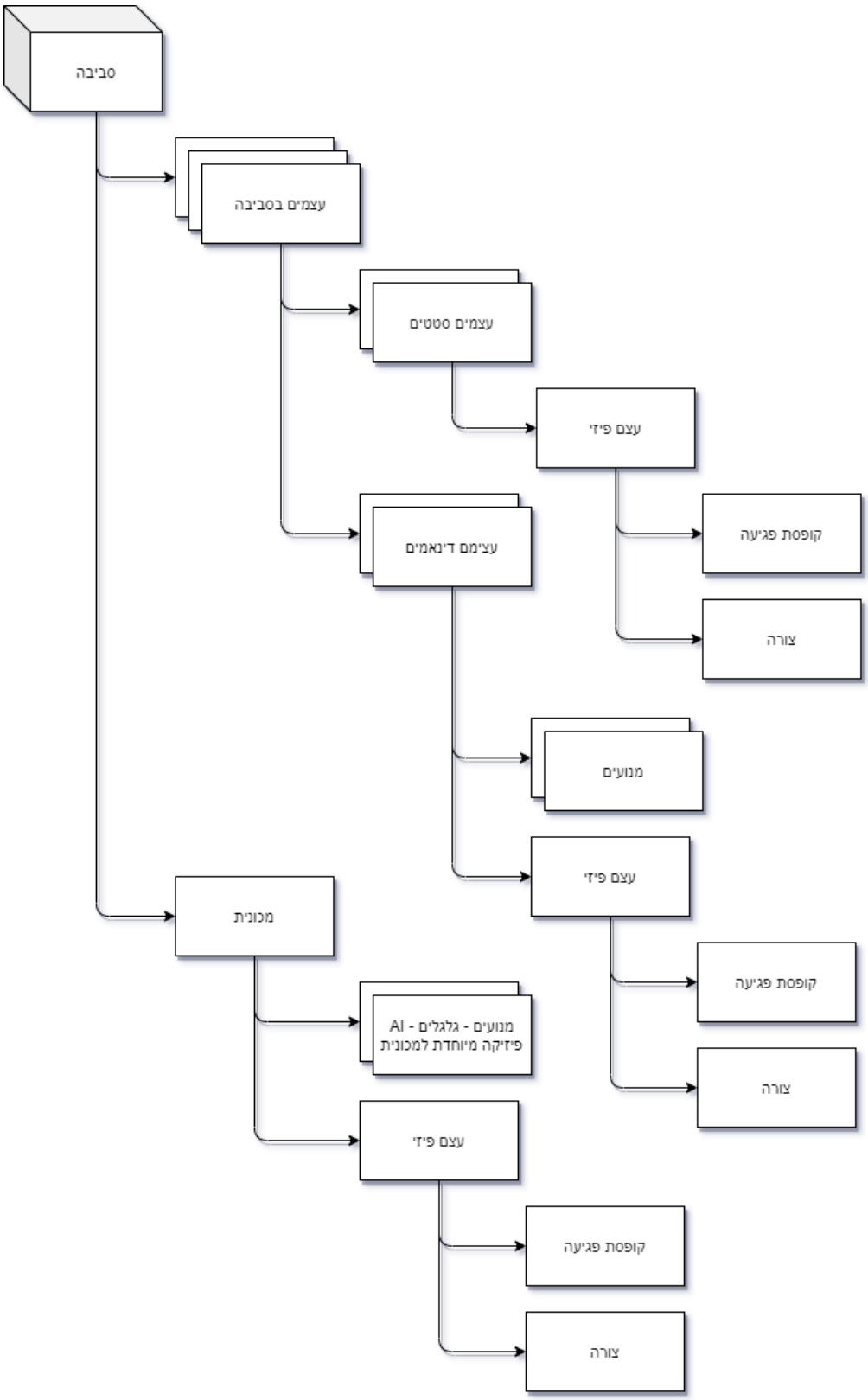
4. <u>החלטת צעד על פי סביבה משתנה:</u>

הדרך הטובה ביותר לפתרון בעיה זו היא AI מסוג ML – Machine Learning שמשתמש ברשת נוירונים בכדי לקבל קלט מהסביבה, מחשב בעזרת שכבותיו את התוצאה ומוציא את הפלט שהוא הצעד הבא הנכון ביותר. לשם כך תכננתי רשת נוירונים המכילה 5 קולטנים – שיהיו מיוצגים על ידי 5 חיישני מרחק, 2 שכבות נסתרות ו-3 פלטים שמייצגים את אפשרויות השליטה הניתנות לאלגוריתם – ישר, ימינה ושמאלה. להלן הרשת:

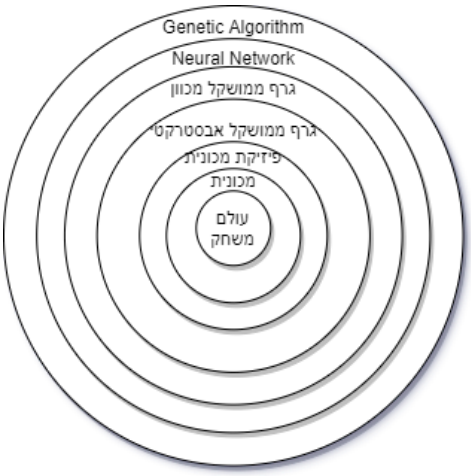


פיתוח ויישום הסימולטור:

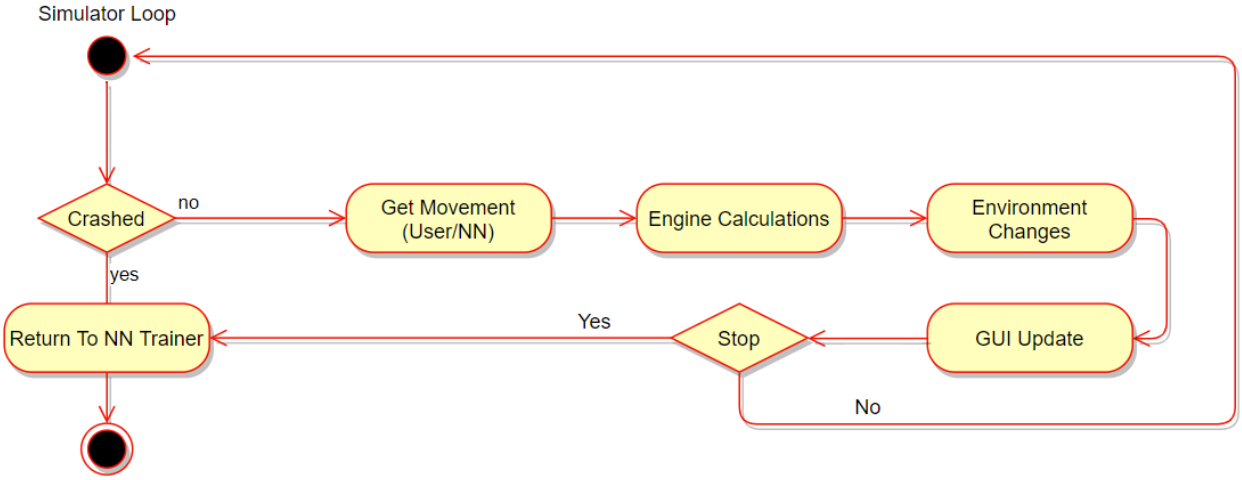
מבנה כללי של הסימולטור –



מעגלי פיתוח:



<u>מבט על:</u>



הסימולטור הוא המרכז את הGUI ומנוע הסימולציה. הפונקציה מריצה את המנוע, בודקת התנגשויות ומשנה את הGUI כך שיהיה תצורה גרפית בזמן ההרצה

<u>הרכב של צורה:</u>

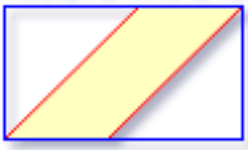
כל עצם מורכב מקווים המתוארים בפונקציה מתמטית וכמו כן כווקטור המגביל את גודל הקו כל קו מתחיל בנקודת הסיום של הקו לפניו חוץ מהקו הראשון שהוא מתקבל ביצירת הצורה. כך נוצר פוליגון שמהווה גוף לעצם. דוגמה לפוליגון שיכול להיות:

<u>קופסת פגיעה – Hit Box</u>:

כל עצם בסביבה יש מלבן או "קופסה" סביבו המאונך לצירי x z ו-y של המסך. לקופסה קוראים "Hit Box".

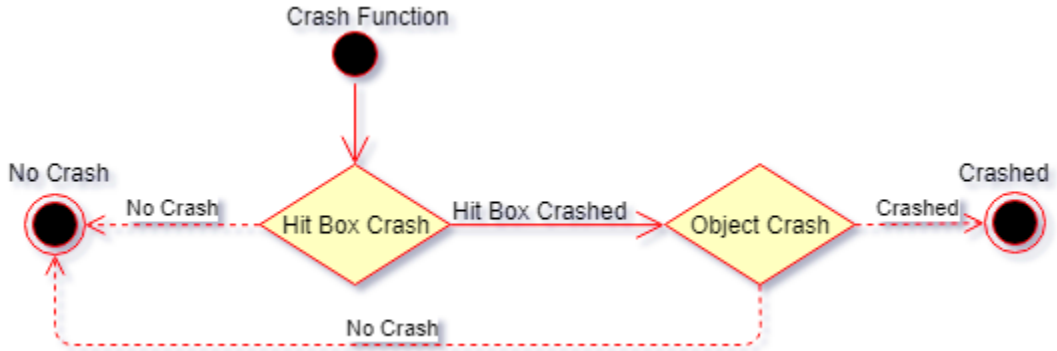
קצוות הקופסה נמצאים בערכים הכי קיצוניים של העצם שעליו הם נמצאים. מטרת הקופסה היא להקל על החישובים של התנגשות בכך שהיא מורכבת רק מארבעה ישרים וכך החישוב קצר יותר כמו כן חישובים של פונקציות שמאונכות לצירים יותר פשוטות .דוגמה לקופסה:

צורה בקווים כחולים וקופסה בכחול



<u>בדיקת התנגשות בין עצמים</u>:

כאשר הסימולטור בודק אם יש התנגשות בין שתי עצמים הוא קודם בודק אם קורת התנגשות בין Hit Box יש עצם אחד עם Hit Box של עצם אחר. המחשה וויזואלית של הבדיקה:



<u>איך בודקים אם שתי צורות מתנגשות:</u>

על צורה של עצם יש קווים ממנה היא מורכבת וכך גם עצם Hit Box מורכב מצוהר שבנויה מישרים.
כל ישר מתואר בעזרת פונקציה מתמטית על המישור x, y ,z ווקטור, כדי לבדוק התנגשות צריך

- לבדוק אם לקוו מצורה של עצם אחד יש נקודה משותפת עם קוו מצורה של עצם אחר. כלומר צריך להשוות בין משוואות הישרים
- אם יוצא נקודה משותפת משמע הצורות מתנגשות. לכן:
  - בשלב ראשון בודקים את ה-Hit Box משום שמשוואות הישרים שלו מקבילים לצירים
  - אם יש התנגשות בודקים את הצורה עצמה שיכולה להיות מורכבת ממספר רב של ישרים בזוויות שונות.

להלן תמונה של התנגשות:

2.4 תיאור האלגוריתם:

כפי שתיארתי מקודם האלגוריתם שבחרתי להשתמש בו הוא ML – Machin Learning. ML משתמש ברשת נוירונים מלאכותית האמורה לדמות המחשב את אותו תהליך שקורה בתוך מוח של יצור חיי ובכך לקבל תוצאות דומות ולפתור בעיות. בפרויקט זה בחרתי לבנות ML שלומד מ-0 כלומר, תחילה האלגוריתם לא יודע מה לעשות והוא לא מודע לכלום ובמשך הזמן לומד.

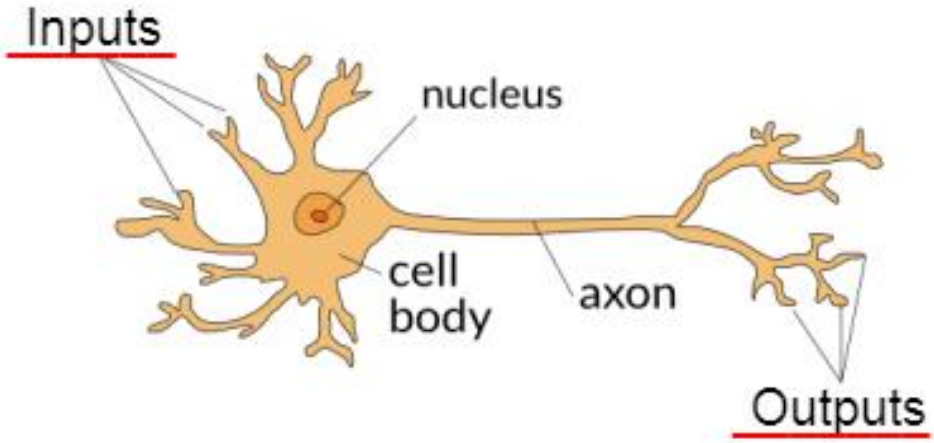
<u>מה זה רשת נוירונים:</u>

רשת נוירונים היא מודל מתמטי חישובי המדמה את התהליכים המוחיים הקוגניטיביים שמתחרשים במערכת עצבים צבעית אשר נמצאת בבעלי חיים.

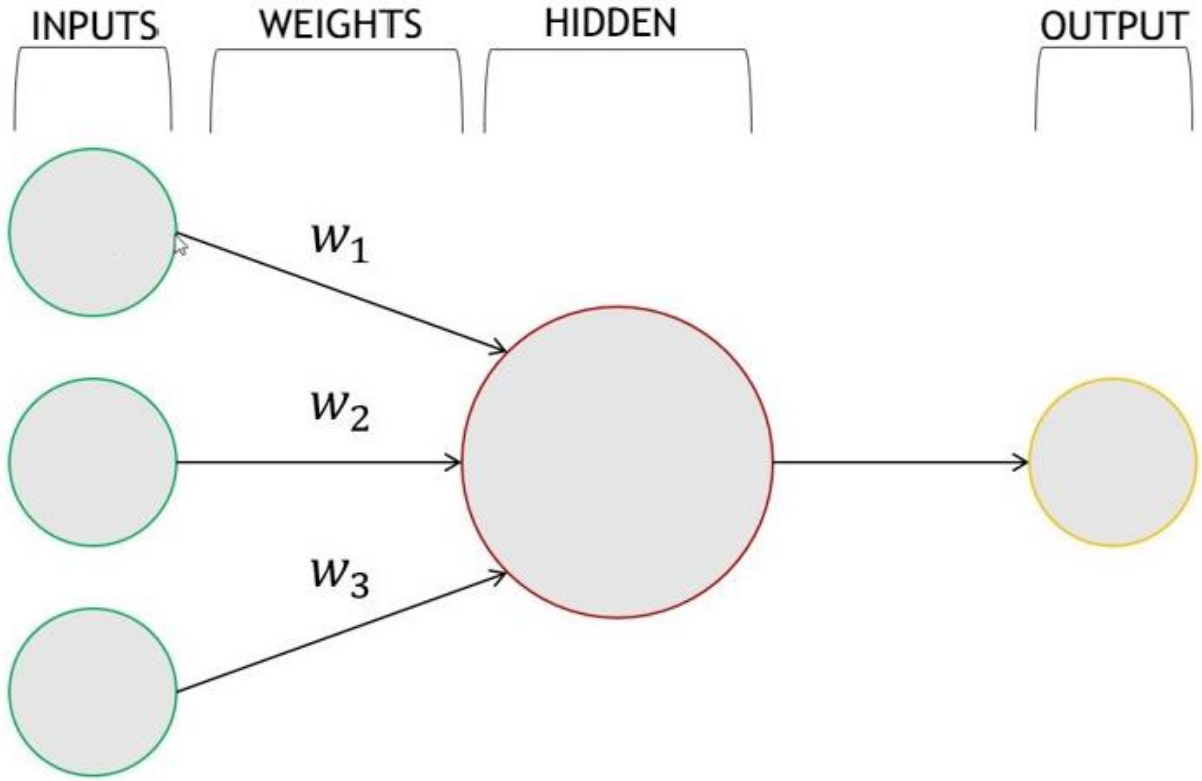
רשת נוירונים מלאכותית בנויה ממספר רב של יחידות מידע המתארות נוירונים של מוח שביניהן יש מספרים שנקראים "משקלים", ככל שמשקל גדול יותר כך אותו נוירון שהוא מחובר אליו משפיע יותר, בנוסף לכל נוירון שך Bias או ההטיה שגורמת לחישובים מדויקים יותר. סט של משקלים בתוך רשת נקרא "מחשבה". בכדי להשיג תוצאה מרשת מעבירים ברשת את הקלטים, מעבירים אותן בין יחידות במידע שמרכיבות שכבות וביניהן עושים פעולות מתמטיות עם שמקלים לבסוף לוקחים את התוצאה משכבת הפלט של הרשת. ישנן 3 סוגי יחידות מידע ברשת: יחידות קלט, יחידות פלט ויחידות חבויות אשר מקשרו בין יחידות הקלט והפלט ויוצרות ממד חישוב נוסף ביוצא דיוק רב יותר, לכל חיבור בין יחידות מידע יש "משקל" בו משתמשים בחישובים המתמטיים בכדי לקבל את הפלט. להלן שרטוט מופשט של רשת נוירונים אמיתית ומלאכותית:

אמיתית –

רשת



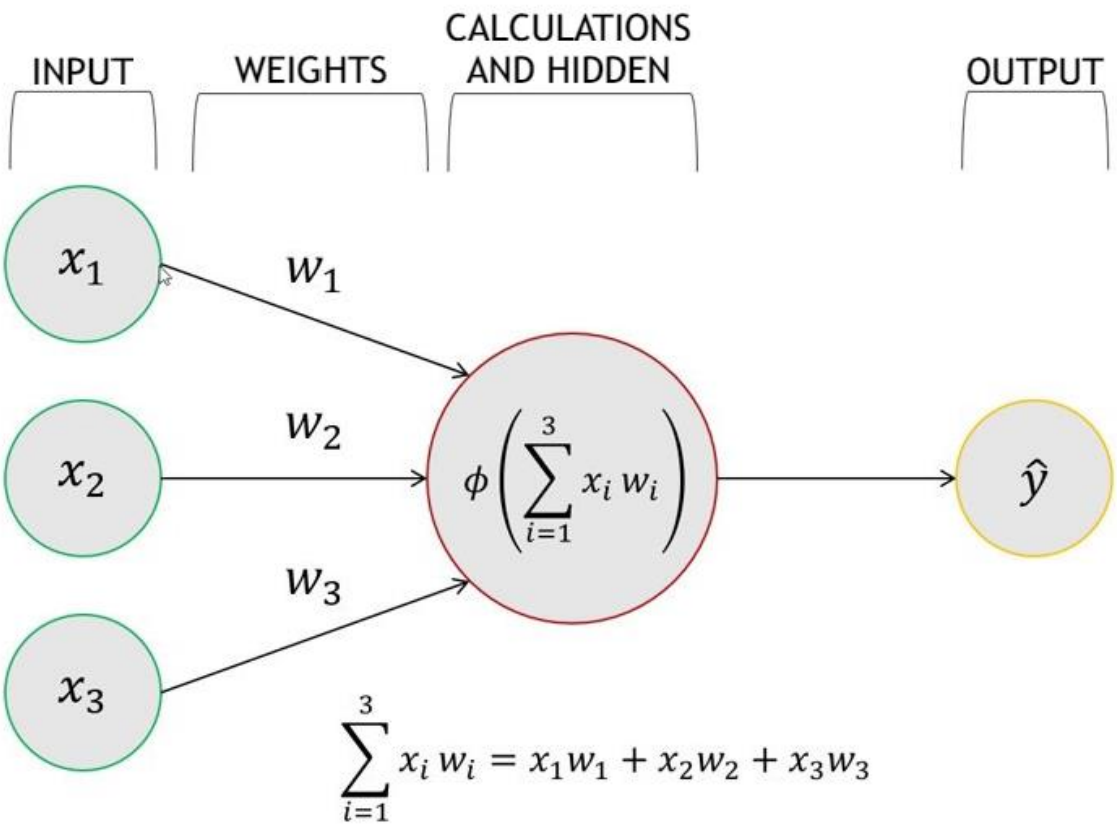
נוירונים מלאכותית -



<u>איך רשת נוירונים עובדת, FeedForward</u>:

כפי שתואר מקודם רשת נוירונים מורכבת ממספר שכבות (קלט, נסתרת ופלט), בין שכבת הקלט לשכבת הפלט ישנם חישובים מתמטיים הנעשים על הקלט וכך עד שמגיע לפלט. דרך טובה לתאר רשת נוירונים היא בעזרת גרף מכוון ממושקל מתורת הגרפים כאשר כל יחידת מידע היא צומת והן מחולקות לפי השכבות (קלט, פלט, נסתרת) וכל הקישורים בין יחידות המידע שעליהן יש משקלם הם קשתות ממושקלות.

כל צומת בשכבה אחת מחוברת לכל הצמתים בשכבה הבאה עם קשתות בין כל הצמתים. תהליך החישוב והתקדמות הקלט עד לפלט נקרא FeedForward, בתהליך קורים הדברים הבאים : 1) צומת מקבלת ערכים, 2) נפעיל את פונקציית Sigmoid מותאמת המכניסה כל במספר בין 0 ו-1, 3) מכפילים לכל צומת השכבה הבאה את הערך בתוך הצומת הנוכחית במשקל שעל הקשת שמחברת בין הצומת הנוכחית לצומת בשכבה הבאה, 4) משום שלכל צומת בשגבה הבאה יש כמה צמתים בשכבה הנוכחית אשר מעבירים אליה מידע אזי נחבר את כל התוצאות ונאכסן בצומת החדשה, 5) חוזרים על 1-4 לכל צומת עד הגעה לסיום בו אין יותר קשתות להמשיך אליהן. להלן תיאור גפרי והסבר עימו :



ניתן לראות שהערך שיש בתוך צומת הבאה שווה לסכום של כל הצמתים שמחוברים אליה כאשר כל צומת מוכפלת במשקל של החיבור ולבסוף מפעילים על הסכום את פונקציית Sigmoid.

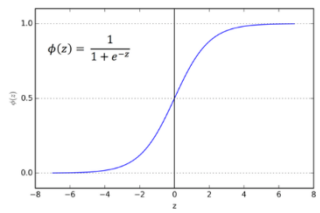
<u>פונקציית האקטיבציה - Sigmoid הרחבה:</u>

פונקציית אקטיבציה נועדה כדי להשאיר את ערכי הצמתים בין 0 ל-1 כך שמספרים גבוהים יהיו קרובים יותר לאחד ומספרים נמוכים יותר יהיו קרובים יותר ל-0. בנוסף לזאת תוך כדי שמירה על ערכים בין 1 ל-0 ניתן לערוך את ערך אקטיבציה של צומת גבוהה. כלומר, הפונקציה מוטת כך שנקודת הפיתול שלה תוטה לכיוון הציר החיובי וכך ערכים גבוהים יותר יתחילו להיות קרובים ל1 דבר זה משמש כBias ונותן תוצאות מדויקות יותר. בML בפרויקט ישנן 2 פונקציות אקטיבציה אחת לקלט ואחת לשימוש בתוך הרשת

- פונקציית האקטיבציה לקלט נראית כך: $F(x) = \frac{1}{1+e^{-1*(x-25)}}$

- פונקציית האקטיבציה לשאר הרשת נראית כך: $F(x) = \frac{1}{1+e^{-1*(x\mp 2)}}$

שרטוט של הפונקציה ללא הזזה:



אימון רשת הנוירונים:

כבדי שרשת הנוירונים תעבוד ותפעל בשביל המשימה שהוצבה בשבילה צריך לאמן אותה כך שתתאים למשימה. ישנם שלושה סוגי אימון: Supervised, Unsupervised, Reinforcement בפרויקט רציתי לגרום למחשב ללמוד לבד ללא ידע קודם וללא עזרה של מתכנת ולכן בחרתי באימון מסוג Reinforcement ובתוך סגנון זה בחרתי בGenetic Algorithm או GA. GA מנסה לקחת את רעיון הברירה הטבעית של צ'ארלס דארווין, רעיון זה אומר ש"החזק שורד" כלומר, מתוך אוכלוסיית יצורים (או במקרה שלנו רשתות נוירונים) יש מגוון של תכונת – כאלה שיותר מתאימות לסביבה ולמשימה וכאל שפחות. ברעיון מתורגם לתוכנה בכך שיוצרים בהתחלה אוכלוסייה של רשתות נוירונים או "מחשבות" שמבנן דומה אך מורכבות מסטים שונים של משקלים שאיתן יעשו את הפעולות המתמטיות. לאחר מכך יתנו להן לעבוד בסביבה ותוך כדי ייתנן לכל מחשבה ציון בהמשך ישתמשו בציון הזה כדי לתת הסתברות לכל מחשבה מה הסיכויים שהיא תמשיך לדור הבא ותרבה את ה"גנים" שלה לצאצאים. בהמשך יבחרו לפי ההסתברות של כל מחשבה את המחשבות החזקות וכמו בטבע ירבו אותן ויצאו ביניהן צאצאים.

ML בפרויקט:

A. בעיות אלגוריתמיות:

1. מבנה רשת נוירונים ביחד לזמן הרצה:

רשת הנוירונים בנויה ממפר שכבות וכל שכבה עושה מספר רב של חישובים. ככל שיש מספר רב יותר של חישובים מספר שתוצאות שיצאו החלטות דומות לצעד הבא מבחינה סטטיסטית תהיה קטנה יתר וכך ה"מחשבה" שה AI יוציא תהיה יותר נכונה ולא רנדומלית. לעומת זאת ככל שיש יותר חישובים כך למערכת ייקח יותר זמן לחשב את הצעד הבא וכך אפשר להסתכן שחישוב איטי מידי וריצה לא חלקה של הסימולטור.

2. שיפור רשת הנוירונים ואבולוציה:

בריצות הראשונות של האלגוריתם. הוא רץ על מחשבות רנדומליות והמחשבות ששרדו הכי הרבה זמן ומרחק יהיה להם ציון ואלה עם הציון הכי גבוהה יעלו לשלב הבא באבולוציה.

בעיה היא כאשר צריך מספיק גיוון "גנטי" אך לא לגוון יותר מידי וכך לפגוע בביצועים ובכושר ההישרדות.

3. <u>אבחון ונתינת ציון לכל מחשבה:</u>

בכדי לדעת אילו "מחשבות" יעלו לשלב הבא באבולוציה צריך לתת ציון הישרדות. ציון הישרדות לכל מחשבה יהי בנוי מפקטורים שיתנו את האבחון הכי טוב לאותה מחשבה. הבעיה היא שצריך למצוא פקטורים שגם יתנו משקל לזמן בו המכונית שרדה עם אותה מחשבה וגם משקל ליעילות של המכונית באותו הזמן, כלומר כמה קרוב היא הייתה להתנגשות.

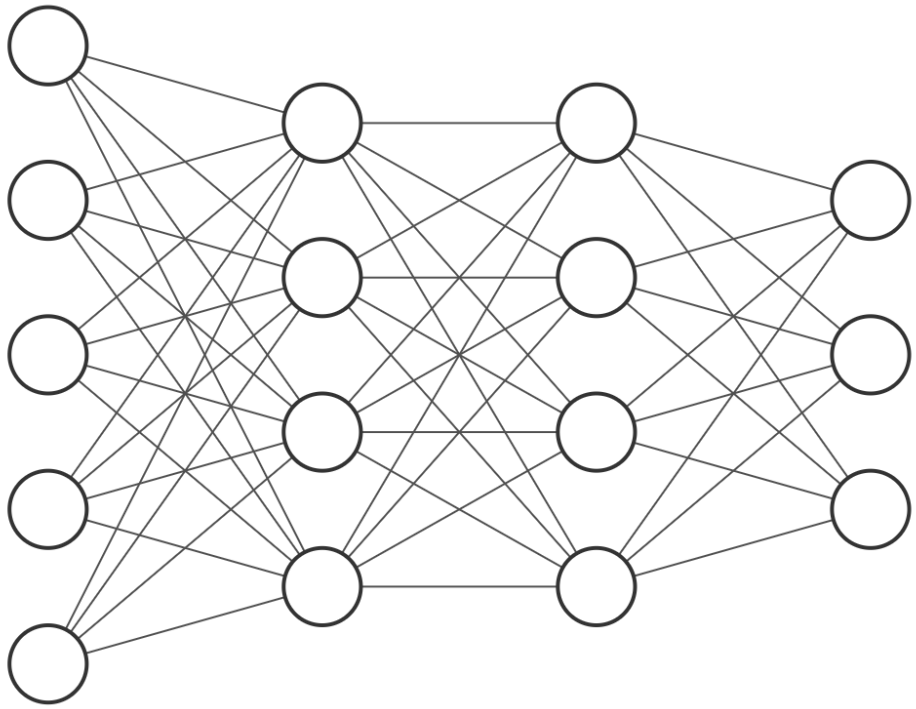
4. <u>זמן ריצה:</u>

האלגוריתם צריך להיות יעיל בחישוביו בכדי לא לגזול זמן מיותר וכך "לתקוע" את המערכת.

B. פתרון הבעיות:

1 <u>מבנה רשת הנוירונים ביחס לזמן ההרצה:</u>

לפתרון בעיה זו החלטתי לבנות את רשת הנוירונים עם 2 שכבות פנימיות בעלות גודל של 4 זאת במקום לעשות שכבה אחת גדולה פנימית. דבר זה נותן גיוון גדול יותר של פעולות מתמטיות ויותר מפעולות מתמטית לכל קלט שנכנס למערכת. כלומר במקום שבמסלול מסוים של הלקט יהיו 2 סבבים של פעולות מתמטיות אז עם 2 שכבות נסתרות יהיו 3 שכבות ובנוסף בגלל שאין שימוש בשכבה אחת מסיבית ישנו חילוק של מגוון של פעולות בסבבים שונים מה שנותן סיכוי טוב יותר שהתשובה תהיה נכונה יותר ושקולה יותר. להלן שרטוט הרשת בה אני משתמש בפרויקט:



כפי שניתן לראות בשרטוט למעלה – ברשת יש 5 צמתי קלט, 2 שכבות נסתרות ובכל אחת 4 צמתים ו3 צמתי פלט. בסך הכל 48 משקלים בתוך הרשת שיוצאים 3 סביבי חישובים. בעצם אם נתאר קבוצה של צמתים שמחוברים לצומת בשכבה הבאה כמספור של X מ-0 עד N (XN...3X,2X,1X), את הצומת בשכבה הבאה כY ואת המשקל שמתאים לX כמספור של W מ-0 עד N (WN...3W,2W,1W) החישוב יהיה מתואר לכל צומת הבאה ככה : $Y = \sigma(\sum_{i=0}^{n} X_i * W_i)$

2 <u>שיפור רשת הנוירונים ואבולוציה :</u>

בכדי להימנע בפגיעה בביצועים תוך כדי שיפור ההישרדות של הAI, האלגוריתם מחשב לכל מחשבה את ההסתברות שהיא תמשיך לדור הבא לפי ציון ההישרדות שלה. לאחר מכן שאלגוריתם עושה "רולטה" בין כל המחשבות ומוציא את המחשבה שיצאה, פעולה זו חוזרת לפי מספר ההורים שצריך. דבר זה מדמה את ההסתברות בטבע של שרידה. לכל ההורים האלגוריתם לוקח אותם ומשלב אותם שילוב גנטי כמו בטבע ויותר מוטציות שלהם להרצה ולבדיקה. בנוסף האלגוריתם לוקח את ה"מחשבה" הכי חזקה ומעביר אותה לדור הבא בכדי שציון בהישרדות של הדור החדש יקטן ככל שאפשר. כמו כן כדי לשמור על הגיוון הגנטי בכל סבב 3 מחשבות נוצרות רנדומלית ונבדקות ביחד עם המוטציות החזקות. על תהליך האבולוציה אסביר בהמשך.

להלן דוגמא של שיפור של מדור אחד לשני :

Thought (set of weights)	Fitness Score	Next Gen Probability
Set Weights 1	1000	0.00147500
Set Weights 2	1200	0.00240000
Set Weights 3	900	0.00004822
Set Weights 4	800	0.00000983
……	……	……..
Set Weights 49 (Best)	2100 (Best)	0.10043200
Set Weights 50	200	0.00000001

=>

New Gen Thought (set of weights)
Set Weights 49 (Best From Before)
Random Weights 1
Random Weights 2
Random Weights 3
Evolution Weights 1
……..
Evolution Weights 46

3 אבחון ונתינת ציון לכל מחשבה:

כפי שנאמר בבעיה הציון שצריך להתייחס גם לזמן שבו המכונית שרדה וגם ליעילות שלה בזמן שהיא הרדה ולכן ניתן ציון רגעי וציון סופי שכולל סכום של כל הציונים הרגעיים שניתנים בכל רגע נתון. הציון הרגעי יהיה מורכב מממוצע המרחקים של המכונית מהקירות חלקי 10 (בשביל לשמור על המספר קטן) ובכך נדע את הציון של היעילות בכל רגע נתון. ואת ציון זה נחבר כל פעם שיש ציון חדש וככל שהמכונית תשרוד יותר זמן יהיה יותר חיבורים וכך ציון גדול יותר ובדרך זאת נתן יחס גם לזמן השרדה. בהמשך יהיה הסבר מפורט יותר.

4 זמן ריצה:

בכדי להקל את עומס החישובים על המערכת. לאחר שרשת הנוירונים תקבל את עצמה כגרף היא תמיר את הגרף למטריצות ולווקטורים וכך ניתן לפשט את הקוד ואת החישובים. כלומר לאחר ההמרה יהיה מערך תלת ממדי בגודל מספר השכבות פחות אחת, שכל תא בו הוא מטריצה דו-ממדית בגדלים שונים (לפי מספר הצמתים בשכבה) שמכילות את המשקלים שעל הקשתות שבין הצמתים של שכבה אחת לשכבה השנייה בגרף. כמו כן יהיה מערך דו ממדי בגודל של מספר השכבות המכיל בכל תא מערך שכל מערך בגודל שונה (לפי מספר הצמתים בשכבה) המכילים את ערכי הצמתים של הגרף.

לאחר ההמרה אם נייצג את מספר השכבות ב-N ואת מטריצת המשקלים לשכבה ספציפית כ-W, את גודלה כM*M ואת מערך הצמתים לאותה שכבה כ-A ואת הגודל כ-M אז החישוב לשכבה אחת יהיה: $A_{next} = \sigma(W * A)$

כלומר זה יהיה כפל מטריצות שסיבוכיות היא $O(M^2)$

ומשום שיש N שכבות אז הסיבוכיות לכל החישוב של רשת הנוירונים היא:

$$O(N * M^2)$$

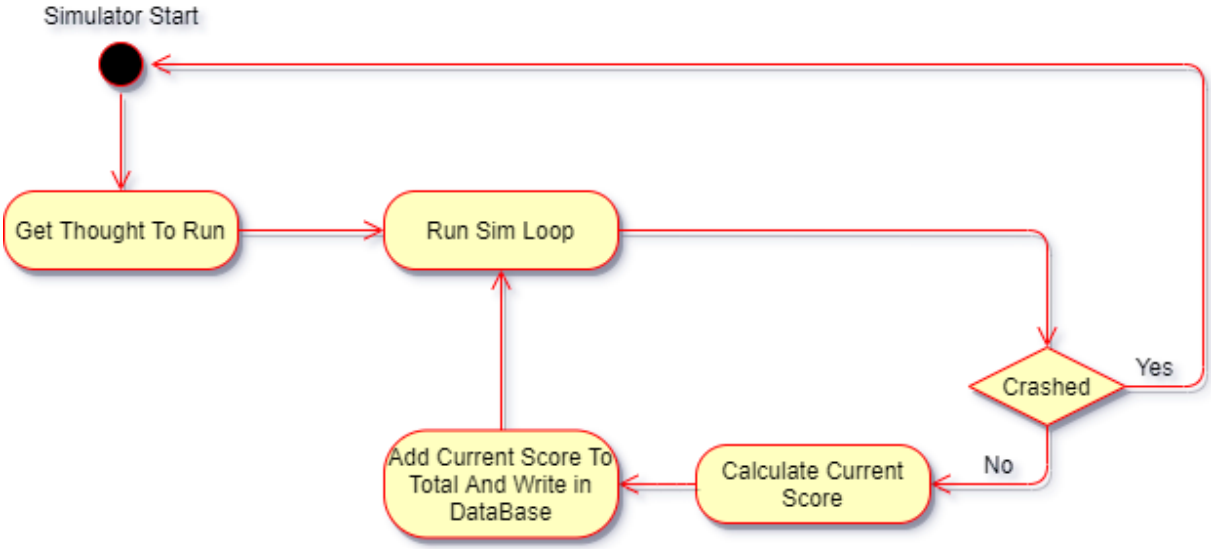
C. שימוש GA בפרויקט:

<u>פסאודו קוד של GA:</u>

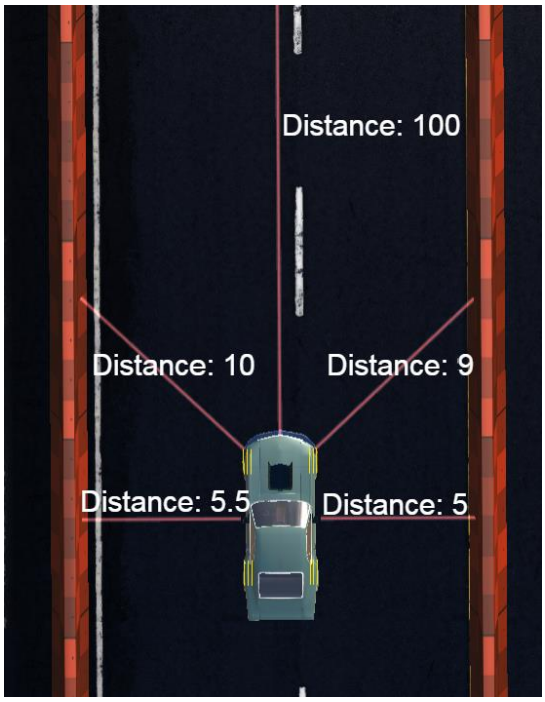
1. צור 50 מוחות רנדומליים
2. לכל מחשבה תעשה:
   2.1. תריץ אותה בסימולטור ותעשה:
      2.1.1. תחשב לה ציון רגעי
      2.1.2. תוסיף אותו לציון הנצבר ותשמור
      2.1.3. אם התנגשות תצא מההרצה אחרת תמשיך
3. תעתיק את המחשבה הכי חזקה לדור הבר
4. תיצור 3 מחשבות רנדומליות לדור הבא
5. לכל מחשבה שנבדקה תחשב את ההסתברות שהיא תעבור לדור הבא
6. תעשה 46 פעמים רולטה ותאחסן את המחשבה שיצאה בתוך מערך בו ישמרו כל 46 התוצאות
7. לכל זוג מחשבות בתוך המערך של התוצאות:
   7.1. תעשה חיבור רנדומלי בין המשקלים שיש
   7.2. תעשה מוטציה לשתי הבנים שיצאו
   7.3. הכנס את שתי הצאצאים שיצאו לדור הבא
8. תחזור לשלב 2

<u>נתינת ציון:</u>

כפי שנאמר מקודם הציון הרגעי יהיה מורכב מממוצע המרחקים של המכונית מהקירות חלקי 10 (בשביל לשמור על המספר קטן) ובכך נדע את הציון של היעילות בכל רגע נתון. ואת ציון זה נחבר כל פעם שיש ציון חדש וככל שהמכונית תשרוד יותר זמן יהיה יותר חיבורים וכך ציון גדול יותר ובדרך זאת נתן יחס גם לזמן השרדה. להלן תרשים שמתאר את רצף החישוב של הציון:



תמונה המתארת נתינת ציון רגעי :



מרחק קדמי: 100
מרחק ימין קדימה: 9
מרחק ימין: 5
מרחק שמאל: 5.5
מרחק קדימה שמאל:10

$$Score = \left(\frac{(100 + 9 + 5 + 5.5 + 10)}{5}\right)/10$$

$$Score = \frac{25.9}{10} = 2.59$$

<u>בחירת המחשבות שעולות לשלב הבא :</u>

ישנן כמה אפשרויות :

1) לקיחת המחשבות הכי חזקות ומהן לעשות אבולוציה
2) שיטה המכילה הסתברות ואפשרות לכל מחשבה להיכנס לדור הבא

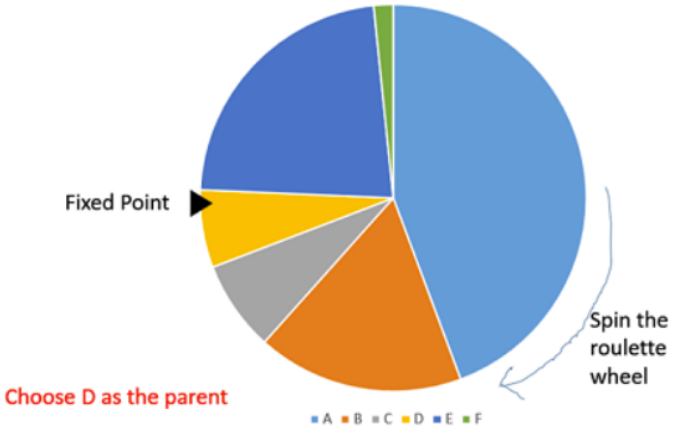
בפרויקט בחרתי בשיטה 2 ויותר ספציפית בשיטת ה-Rolette Wheel Selection .

לבחירת ההורים לדור הבא נשתמש בשיטת Rolette Wheel Selection. בשביל שיטה זו נחשב את ההסתברות של כל מחשבה :

לאחר שכל 50 הריצות נעשו וישנם ציונים לכל המחשבות. נחשב את ההסצברות שיש לכל מחשבה, ככל שלמחשבה יש ציון גדול יותר כך יהיה לה הסברות גדולה יותר, כלומר, יותר סיכוי לעבור לדור הבא. חישוב ההסתברות נעשה בנוסח הבא : $Probebility_i = \frac{FitScore_i}{TotalFitScore}$

לאחר שחישבנו את ההסתברות נוכל לדמות גלגל רולטה אשר יש בו 50 תאים (כמספר המחשבות) כל תא שניתן לצאת בגלגול הגלגל הוא מחשבה והסיכוי שהגלגל יצא על מחשבה היא ההסתברות שלה.

להלן שרטוט המדמה את הגלגל:



Chromosome	Fitness Value
A	8.2
B	3.2
C	1.4
D	1.2
E	4.2
F	0.3

בכדי להשיג 46 הורים בשביל לעשות להשיג צאצאים נסובב את הגלגל 46 פעמים

להלן פסאודו קוד הגלגל:

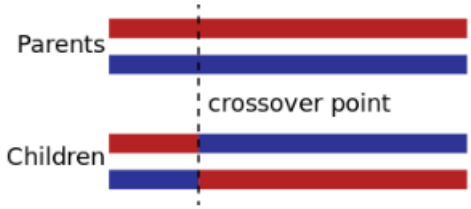
1. חשב את ההסתברות של כל מחשבה
2. תעשה מספר רנדומלי ממשי בין אחד לאפס
3. SumOfProb = 0
4. לכל הסתבברות של כל מחשבה:
   4.1. אם המספר הרנדומלי גדול שווה מ-המחשבה עם ההסתברות האחרונה + SumOfProb:
      4.1.1.תחזיר את המחשבה האחרונה
   4.2. אם המספר הרנדומלי גדול שווה מ-SumOfProb +ההסתברות הנוכחית וקטן מההסתברות ההבאה:
      4.2.1.תחזיר את המחשבה הנוכחית

אבולוציה והתרבות:

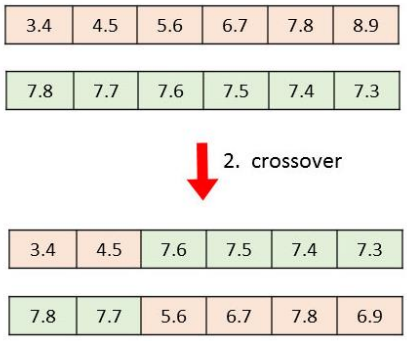
שיטות שילוב "DNA" של רשת:

1) שילוב חציה – genetic crossover:

שילוב זה פועל בדומה לשחלוף כרומוזומים (Choromosomal Crossover) בביולוגיה, חותך רצף של כרומוזומים מDNA אחד במקרה שלנו מחשבה שמורכבת ממשלקים אז יחתך רצף משלקים וכמו כן יחתך רצף משלקים ממחשבה שנייה וייבצר איחוד בין השנים. להלן תיאור של האיחוד בביולוגיה:



ולהלן השילוב עם מחשבות שמורכבות ממשקלים:

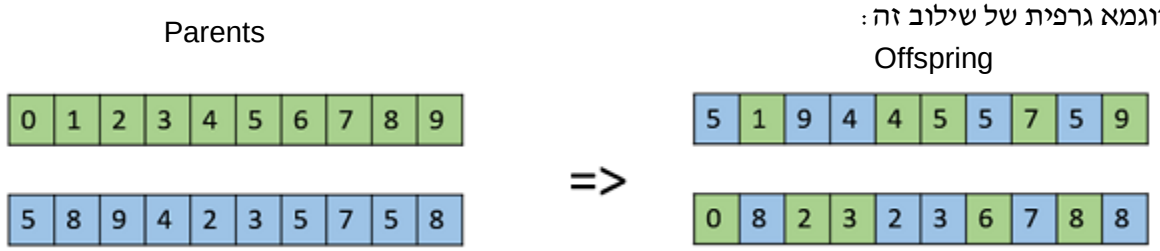


2) שילוב רנדומלי:

בשילוב רנדומלי מחליטים רנדומלית בין כל זוג תואם של משקלים בין שתי מחשבות האם להחליף במשלקים בין המחשבות או לא. פסאודו קוד של שילוב זה הוא כזה:

1. קבל את זוג המחשבות (ההורים)
2. לכל משקל בתוך המחשבה 1 ומחשבה 2 בהתאם:
   2.1. החליט הנדומלית אם להחליף בין המשקלים
   2.2. אם הוחלט להחליף החלף
   2.3. אחרת תמשיך
3. החזר את זוג המחשבות המחודשות

להלן דוגמא גרפית של שילוב זה:



<u>מוטציה:</u>

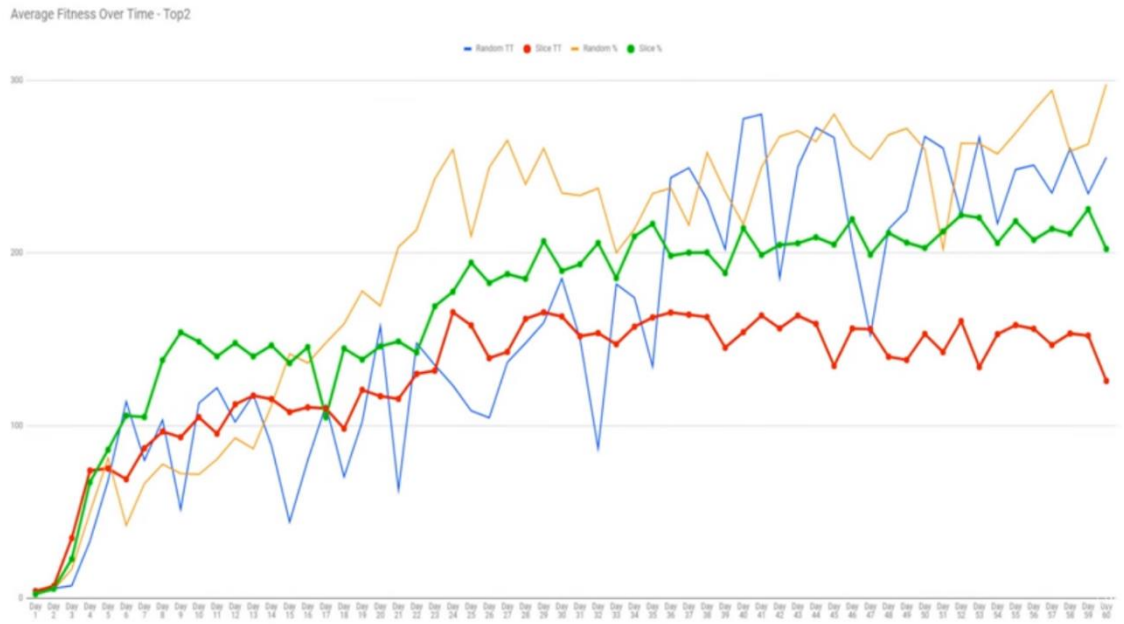
מוטציה נועדה בשביל להשאיר מגוון שונה של מחשבות וריענון של האוכלוסייה לאפשרויות חדשים. תהליך המוטציה הוא לשנות ערכים רנדומלית בכ10 אחוז מעצמם להוסיף או להפחית

<u>ניסוי – (מוכן אז צורך שכתוב - להוספה):</u>

לאחר הסבר על כל האפשרויות של בחירת הורים ואבולוציה. ערכתי ניסוי בכדי להראות שהבחירות היו נכונות. בניסוי ניסיתי לשלב את כל האופציות. כלומר לקחת את ההכי חזקים ולשלב אותם ברנדומליות, לקחת את ההכי חזקים ולהרבות אותם בשילוב חציה, לקחת את ההורים לפי הסתברות ולשלב רנדומלית ולקחת את ההורים לפי הסתברות ולשלב בחציה. כל האפשרויות באופן וויזואלי:

אפשרויות שילוב / בחירת ההורים	שילוב רנדומלי(R)	שילוב חציה(H)
בחירת הורים חזקים(S)	RS בכחול	HS באדום
בחירה סטטיסטית(M)	RM בצהוב	HM בכחול

להלן שרטוט של גרף המראה יעילות של קוד לפונרציה של זמן: כפי שניתן לראות האפשרות שהגיע הכי מהר ליעילות הכי גבוהה היא הפגשות של שילוב רנדומלי ובחירה סטטסטית ולכן מבחר בה



2.5 מבנה מתונים:

בפרויקט השתמשתי במבני הנותנים הבאים:

גרף:

בשביל קוד העברה נוחה של נתונים לספריית רשת הנוירונים שבניתי השתמשתי בספריית הגרף בכתבתי כדי לייצג את הרשת. דרך טובה לתאר רשת נוירונים היא בעזרת גרף מכוון ממושקל מתורת הגרפים כאשר כל יחידת מידע היא צומת והן מחולקות לפי השכבות (קלט, פלט, נסתרת) וכל הקישורים בין יחידות המידע שעליהן יש משקלם הם קשתות ממושקלות.

מערכים תלת ממדיים בעלי מטריצות שונות בגודלן:

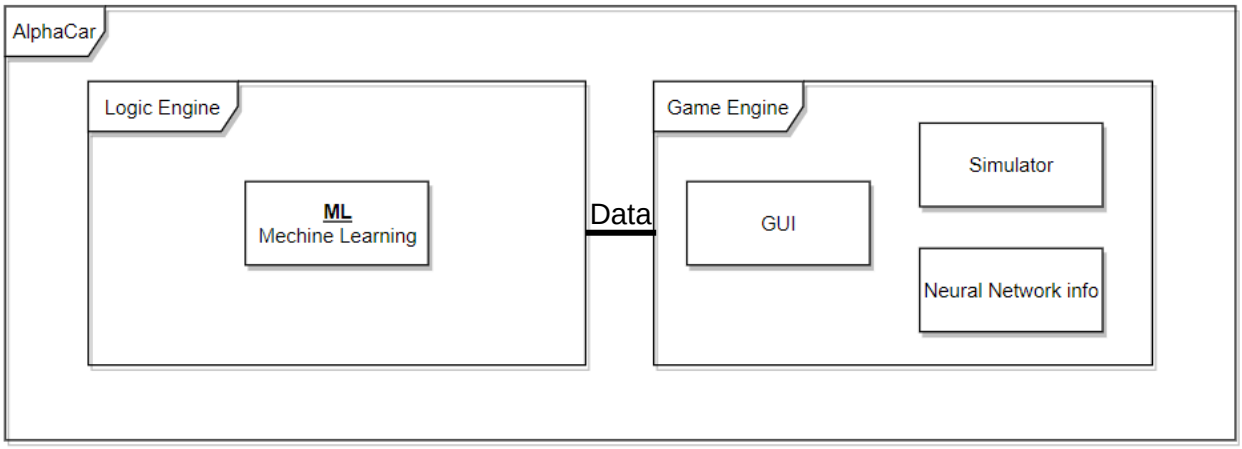
על מנת לעשות חישובים נוחים ופשוטים בחרתי שלאחר קבלת הגרף נמיר את הקשתות למטריצות ומערכים. המערכים שיהיו הם: מערך תלת ממדי בגודל מספר השכבות ברשת פחות אחת, שכל תא בו הוא מטריצה דו-ממדית שמכילה את המשקלים שעל הקשתות שבין הצמתים של שכבה אחת לשכבה השנייה בגרף. גודל המטריצה יהיה מספר הצמתים בשכבה אחת כפול מספר הצמתים בשכבה הבאה (כמספר המשקלים או הקשתות בין השכבות). כלומר, לשכבה אחת בעלת 4 צמתים ושכבה שנייה בעלת 5 צמתים יהיה מטריצה בגודל של 4*5. כמו כן יהיה מערך דו ממדי בגודל של מספר השכבות המכיל בכל תא מערך המכיל את ערכי הצמתים של שכבה מסוימת וגודלו של המערך יהיה כמספר הצמתים באותה שכבה.

3 חלוקה למודלים:

3.1 חלוקה למודלים:

UML (להוספה):

חלק זה כולל את תיאור מבנה המערכת ופירוט המודולים השונים בה



ישנם 13 קבצי-קוד מרכזיים המרכיבים ומפעילים את הפרויקט

- **7 קבצים אחראים על ממשק הגרפי: מסך סימולציה.**
  אלה שמות הקבצים:
  - ArcadeCar.cs
  - CameraCar.cs
  - CameraTrigger.cs
  - CarCollision.cs
  - LaserSensor.cs
  - MovingBlock.cs
  - SuspensionTest.cs

- **2 קבצים אחראים על היחסים בין הסימולטור והרשת הנוירונים:**
  - ArcadeCar.cs
  - CarCollision.cs

- **7 קבצים אחראים על הגיאומטריה האנליטית, התנגשות בין אובייקטים בסביבה ועל צורה של האובייקטים בפרויקט:**

  אלה שמות הקבצים:
  - ArcadeCar.cs
  - CameraCar.cs
  - CameraTrigger.cs
  - CarCollision.cs
  - LaserSensor.cs
  - MovingBlock.cs
  - SuspensionTest.cs

- **שלושה קבצים ושתי ספריות האחראים על מנן התזוזה של אובייקטים זזים כמו** Car:

  שלושת הקבצים:
  - MovingBlock.cs
  - ArcadeCar.cs
  - CarCollision
  - SuspensionTest

  שתי הספריות:
  - **NeuralNetworkLibary**
  - **GraphLibary**

ArcadeCar:

```
ArcadeCar.cs
  ArcadeCar
    WeightSetIndex : int
    NextMove : string
    CountT : int
    WHEEL_LEFT_INDEX : int
    WHEEL_RIGHT_INDEX : int
    wheelWidth : float
    nn : NeuralNetwork
    ls1 : LaserSensor
    ls2 : LaserSensor
    ls3 : LaserSensor
    ls4 : LaserSensor
    ls5 : LaserSensor
    WheelData
    Axle
    centerOfMass : Vector3
    accelerationCurve : AnimationCurve
    accelerationCurveReverse : AnimationCurve
    steerAngleLimit : AnimationCurve
    steeringResetSpeed : AnimationCurve
    steeringSpeed : AnimationCurve
    debugDraw : bool
    flightStabilizationForce : float
    flightStabilizationDamping : float
    handBrakeSlipperyTime : float
    controllable : bool
    downForceCurve : AnimationCurve
    downForce : float
    axles : Axle[]
    afterFlightSlipperyTiresTime : float
    brakeSlipperyTiresTime : float
    handBrakeSlipperyTiresTime : float
    isBrake : bool
    isHandBrake : bool
    isAcceleration : bool
    isReverseAcceleration : bool
    accelerationForceMagnitude : float
    rb : Rigidbody
    style : GUIStyle
    wheelRay : Ray
    wheelRayHits : RaycastHit[]
    Reset(Vector3) : void
    AddEdgesForNN(DirectedGraph<string, double>, double[], int[]) : void
    AddEdgesForNN4D(DirectedGraph<string, double>, double[], int[]) : void
    GetCountMove() : int
    SetCountMove(int) : void
    SetNextMove(string) : void
    SetWeightSetIndex(int) : void
    GetWeightSetIndex() : int
    GetWeights() : double[]
    Start() : void
    OnValidate() : void
    GetHandBrakeK() : float
    GetSteeringHandBrakeK() : float
    GetAccelerationForceMagnitude(AnimationCurve, float, float) : float
    GetSpeed() : float
    CalcAccelerationForceMagnitude() : float
    GetSteerAngleLimitInDeg(float) : float
    CalculateFitScore(double[]) : double
    WriteThoughtFitnessScore(double) : void
    UpdateInput() : void
    Update() : void
    FixedUpdate() : void
    OnGUI() : void
    AddForceAtPosition(Vector3, Vector3) : void
    OnDrawGizmosAxle(Vector3, Axle) : void
    OnDrawGizmos() : void
    RayCast(Ray, float, ref RaycastHit) : bool
    CalculateWheelForces(Axle, Vector3, WheelData, Vector3, int, int, int) : void
    CalculateAxleForces(Axle, int, int) : void
    CalculateAckermannSteering() : void
    CalculateWheelVisualTransform(Vector3, Vector3, Axle, WheelData, int, float, out Vector3, out Quaternion) : void
    CalculateWheelRotationFromSpeed(Axle, WheelData, Vector3) : void
    ApplyVisual() : void
```

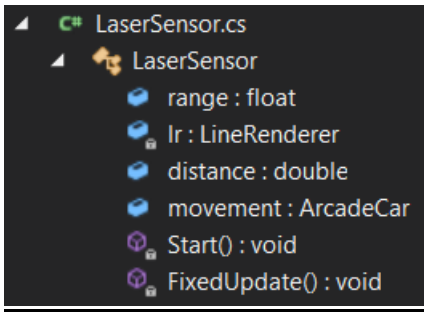
ArcadeCar - **מחלקה יורשת מ**MonoBehaviour **של המנוע**
תכונות:
• **Wheel_Left_Index** – המיקום של הגלגל הקדמי השמאלי
• **Wheel_Right_Index** - המיקום של הגלגל הקדמי השמאלי
• **Nn** – רשת הנוירונים
• **Ls1-ls5** – חיישנים 1-5 מסוג **Laser Sensor**
• **(Class)WheelData** – מחלקה שמכילה את כל המידע על הגלגל
• **(Class)Axle** – מחלקה שמכילה מידע על ציר
• **centerOfMass** – ווקטור שמייצג מרכז של מסה

- **accelerationCurveReverse\accelerationCurve** - גרף שמייצג את התקדמות האצה
- **steerAngleLimit** – גבול זווית פנייה
- **downForce\downForceCurve** – כוח שמושך למטה
- **axles** – מערך שמכיל את הצירים
- **isBrake\ isHandBrake\ isAcceleration\ isReverseAcceleration** – משתנים כללים לפעולת המכונית
- **WeightsSetIndex** – מספר ה"מחשבה" שעכשיו נבדקת

פעולות:

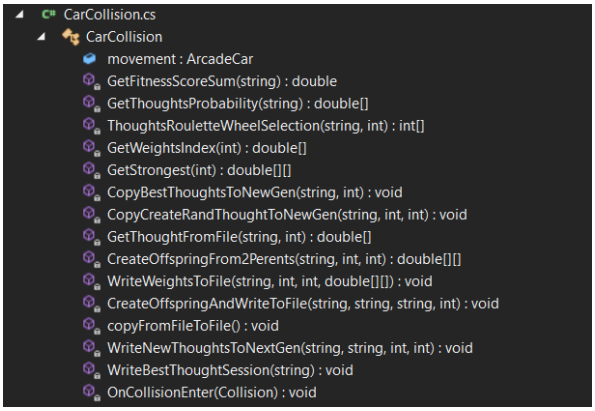
- **Reset** – מחזיר את המכונית למצב ההתחלתי שלה עם **DNA** חדש
- **AddEdgesForNN4D** – מוסיף **Edges** לגרף של ה**NN**
- **Start** – נקראת פעם אחת בתחילת ריצת הקוד – יוצר את ה**NN**, מרכז המסה ואת הגוף הפח
- **GetHandBrakeK** – מחזיר את הכוח של הבלם ים
- **GetSteeringHandBrakeK** – מחזיר את הכוח עצירה בפנייה
- **GetAccelerationForceMagnitude** – מחזיר כוח האצה
- **GetSpeed** – מחזיר את המהירות
- **GetSteerAngleLimitInDeg** – מקבלת את המהירות במטר לשנייה ומחזיקה את גבול הפנייה במעלות
- **UpdateInput** – עושה את קליטת הצעד הבא
- **FixedUpdate** – נקראת כל סיבוב של הסימולטור – עושה את כל הקריאות של הפעולות שאיתן מפעילים את המכונית ואת כל הפיזיקה
- **AddForceAtPosition** – מקבלת ווקטור כוח וווקטור מיקום ומסיפה את הכוח באותו מיקום
- **WriteThoughtFitnessScore** – כותב ל**DataBase** את הציון הכולל החדש של המכונית
- **CalculateFitScore** – מחשב את הציון הנוכחי את המכונית

LaserSensor:



LaserSensor – מחלקה יורשת מ MonoBehaviour **של המנוע**
תכונות: • **Range** – המרחק המקסימלי של החיישן • **Lr** – דרך להראות את הלייזר עם **LineRenderer** של **Unity** • **Distance** – המרחק כרגע שהחיישן מחזיר • **Movement** – המכונית שהחיישן עליה
• **Start** – נקראת פעם אחת בתחילת ריצת הקוד – יוצרת את ה**LineRenderer** • **FixedUpdate** – נקראת כל סיבוב של הסימולטור – עושה את כל חישובי המרחק ואת שינויי הפיזיקה והלייזר. כמו כן מעדכן את המרחק שהחיישן מחזיר (**Distance**)

CarCollision:

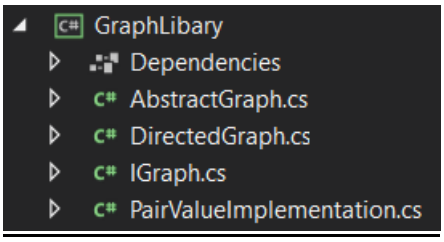


CarCollision - מחלקה יורשת מ MonoBehaviour **של המנוע**
תכונות: • **Movement** – אובייקט מסוג **ArcadeCar** שמכיל את המכונית שתתנגש במחסומים כשתטעה
• פעולות:

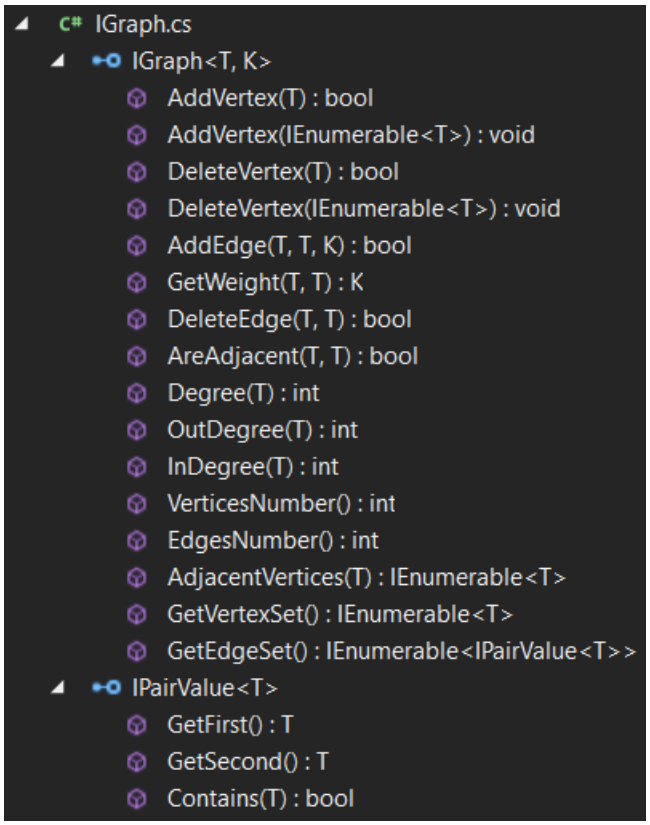
- **OnCollisionEnter** – פעולה שקוראת רק כאשר ישנו שילוב במשוואות הגופים במרחב של המכונית ומחסום – הפעולה בודקת אם מדובר במחסום ואיזה מחסום אם כן יש התנגשות בקיר אזי נפונקציה תעביר את הריצה למחשבה הבאה אם אין הפונקציה תקרא לכל הפונקציות כדי שתהיה אבולוציה וייבצר דור חדש.
- **GetFitnessScoreSum** – הפונקציה מקבלת שם של קובץ בינארי בו שמורים הציונים של המחשבות ומחזירה את הסכום של כל הציונים הסופיים של המחשבות ומחזיר אותו.
- **GetThoughtsProbability** - הפונקציה מקבלת שם של קובץ בינארי בו שמורים הציונים של המחשבות, הפונקציה מחשבת את ההסתדברות שכל אחד מהמחשבות יעבור לדור הבא ומחזירה מערך עם הסיכויים.
- **ThoughtsRouletteWheelSelection** – הפונקציה מקבלת שם של קובץ בינארי בו שמורים הציונים של המחשבות, ומספר מחשבות להשיג מהרולטה. הרולטה תפעל ויוחזר מערך של ה**ID** של המחשבות שנבחרו.
- **GetWeightsIndex** - הפונקציה מקבלת שם של קובץ בינארי בו שמורים הציונים של המחשבות, ומספר מחשבות להשיג. הפונקציה תחזיר את ה**ID** של המחשבות הכי חזקות.
- **GetStrongest** – הפוקנציה מחזירה מערך דו-ממדי שכל תא בו הוא מערך המכיל מחשבה החזקה ביותר
- **CopyBestThoughtsToNewGen** – מעתיקה את המחשבה החזקה ביותר לקובץ שמכיל את הדור הבא.
- **CopyCreateRandThoughtToNewGen** – יוצר מחשבות רנדומליות חדשות לדור הבא
- **GetThoughtFromFile** – משיג לפי **ID** מחשבה מקובץ המחשבות
- **CreateOffspringFrom2Perents** – יותר 2 צאצעים מ2 הורים לדור הבא.
- **WriteWeightsToFile** – מקבל מחשבות וכותב אותם לקובץ המכיל את הדור הבא
- **copyFromFileToFile** – מעתיק את המחשבות מקובץ אחד לשני
- **WriteNewThoughtsToNextGen** – פונקציה הקוראת לכל הפונקציות שיוצרות את הדור הבא
- **WriteBestThoughtSession** – כותה את המחשבה החזקה ביותר לקובץ המכיל את הדוק הבא

הספריות:

GraphLibary:



IGraph ממשקים:



1) ממשק **IGrapg**:

IGraph<T,K> – ממשק לגרף
• **T** – סוג משתנה של הצומת בגרף • **K** – סוג משתנה של הקשתות בגרף – לאחר מכן **K** יהיה **PairValue**
• **AddVertex(T vertex)** – מוסיף צומת אחת הבגרף • **AddVertex(IEnumerable<T> vertexSet)** – מוסיף לגרף מספר צמתים שנמצאים בסט, אם צומת קיימת מדגל עליה

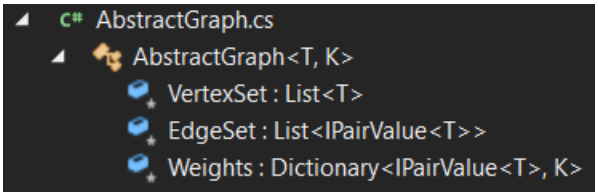
- **DeleteVertex(T vertex)** – מסיר צומת מהגרף
- **DeleteVertex(IEnumerable<T> vertexSet)** – מסיר את כל הצמתים שיש בסט מתוך הגרף
- **AddEdge(T v1, T v2, K weight)** – מוסיף קשת ממושקלת בין שתי צמתים
- **GetWeight(T v1, T v2)**- מחזיר משקל בין שתי צמתים
- **DeleteEdge(T v1, T v2)** – מוחק קשת
- **AreAdjacent(T v1, T v2)** – מחזיק אמת אם צומת אחת שקילת סמוכה לצומת השנייה
- **Degree(T vertex)**- מחזיר את מספר הצמתים (קשתות) שמחוברות לצומת שקיבל
- **OutDegree(T vertex)** – מחזיר את מספר הקשתות במחוברת אל צומת שקיבל ומכוונות אליה
- **InDegree(T vertex)** - מחזיר את מספר הקשתות במחוברת אל צומת שקיבל ומכוונות ממנה
- **VerticesNumber()** – מחזיר את מספר הצמתים שיש בגרף
- **EdgesNumber()** – מחזיר את מספר הקשתות שיש בגרף
- **AdjacentVertices(T vertex)** – מחזיר סט של צמתים הסמוכים לצומת שקיבל
- **GetVertexSet()** – מחזיר סט של כל הצמתים
- **GetEdgeSet()** – מחזיר סט של כל הקשתות

2) **IPairValue**:

IPairValue<T> – ממשק לקשת בגרף
תכונות: • **T** – סוג משתנה של הצומת בגרף
פעולות: • **GetFirst** – מחזיר את הצומת הראשונה בקשת (האחת שהצומת מכוונת ממנה) • **GetSecond** – מחזיר את הצומת השנייה בקשת(האחת שהקשת מכוונת אליה) • **Contains** – מקבל צומת טמחזיר אמת אן היא קשורה לקשת ושקר אם לא

קבצים בספריה:

AbstractGraph:



AbstractGraph <T,K> – מימוש אבסטרקטי לגרף – משתמש בממשק **IGrpah**

תכונות:

- **T** – סוג משתנה של הצומת בגרף
- **K** – סוג משתנה של הקשתות בגרף – לאחר מכן **K** יהיה **PairValue**
- **VertexSet** – רשימה של צמתים
- **EdgeSet** – רשימה של **PairValue** כלומר, רשימה של קשתות
- **Weights** – מילון שמחבר כל קשת למשקל שלה

פעולות – כמו ב-IGrapg:

- **AddVertex(T vertex)** – ממומש במחלקה הנוכחית
- **AddVertex(IEnumerable<T> vertexSet)** – ממומש במחלקה הנוכחית
- **DeleteVertex(T vertex)** – ממומש במחלקה הנוכחית
- **DeleteVertex(IEnumerable<T> vertexSet)** – ממומש במחלקה הנוכחית
- **AddEdge(T v1, T v2, K weight)** – אבסטרקטי
- **GetWeight(T v1, T v2)**- אבסטרקטי
- **DeleteEdge(T v1, T v2)** – אבסטרקטי
- **AreAdjacent(T v1, T v2)** – אבסטרקטי
- **Degree(T vertex)**- אבסטרקטי
- **OutDegree(T vertex)** – אבסטרקטי
- **InDegree(T vertex)** - אבסטרקטי
- **VerticesNumber()** – ממומש במחלקה הנוכחית
- **EdgesNumber()** – ממומש במחלקה הנוכחית
- **AdjacentVertices(T vertex)** – אבסטרקטי
- **GetVertexSet()** – ממומש במחלקה הנוכחית
- **GetEdgeSet()** – ממומש במחלקה הנוכחית

DirectedGraph:

DirectedGraph<T,K> – מחלקה יורשת מ-AbstractGraph ומשתמש בממשק **IGraph**
תכונות מ-AbstractGraph **:** • **T** – סוג משתנה של הצומת בגרף • **K** – סוג משתנה של הקשתות בגרף – לאחר מכן **K** יהיה **PairValue** • **VertexSet** – רשימה של צמתים • **EdgeSet** – רשימה של **PairValue** כלומר, רשימה של קשתות • **Weights** – מילון שמחבר כל קשת למשקל שלה
פעולות – כמו ב-IGrapg**:** • **AddVertex(T vertex)** – ממומש ב**AbstractGaprh** • **AddVertex(IEnumerable<T> vertexSet)** – ממומש ב**AbstractGaprh** • **DeleteVertex(T vertex)** – ממומש ב**AbstractGaprh** • **DeleteVertex(IEnumerable<T> vertexSet)** – ממומש ב**AbstractGaprh** • **AddEdge(T v1, T v2, K weight)** – ממומש במחלקה הנוכחית • **GetWeight(T v1, T v2)**- ממומש במחלקה הנוכחית • **DeleteEdge(T v1, T v2)** – ממומש במחלקה הנוכחית • **AreAdjacent(T v1, T v2)** – ממומש במחלקה הנוכחית • **Degree(T vertex)**- ממומש במחלקה הנוכחית • **OutDegree(T vertex)** – ממומש במחלקה הנוכחית • **InDegree(T vertex)** - ממומש במחלקה הנוכחית • **VerticesNumber()** – ממומש ב**AbstractGaprh** • **EdgesNumber()** – ממומש ב**AbstractGaprh** • **AdjacentVertices(T vertex)** – ממומש במחלקה הנוכחית • **GetVertexSet()** – ממומש ב**AbstractGaprh** • **GetEdgeSet()** – ממומש ב**AbstractGaprh**

ספריית NeuralNetWork:

NeutalNetwork:

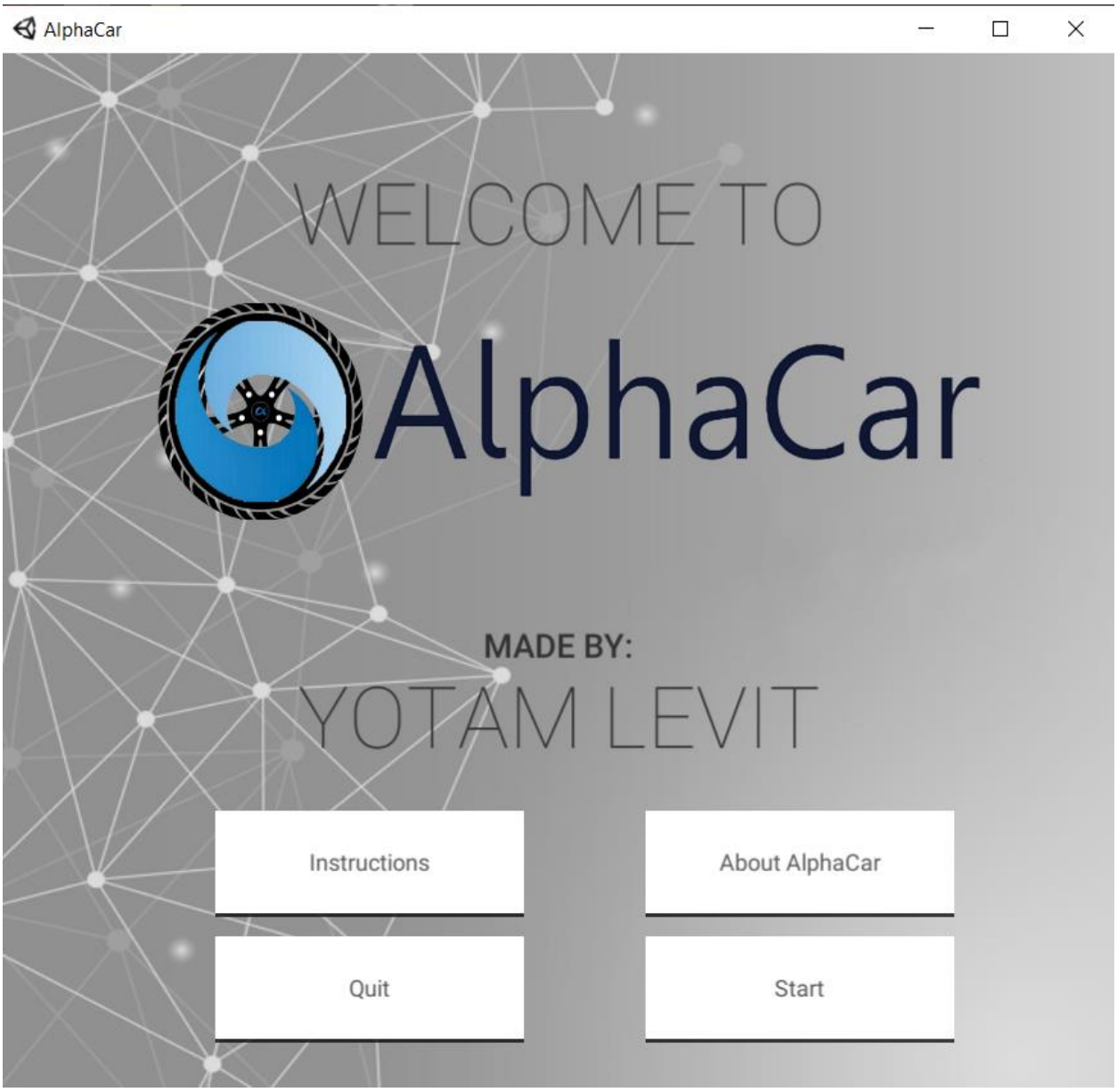
```
NeuralNetwork.cs
  Constants
    SigmoidAPara : double
  NeuralNetwork
    DGraph : DirectedGraph<string, double>
    weights : double[][][]
    layes : double[][]
    NeuralNetwork(DirectedGraph<string, double>, int[])
    UpdateNeuralNetwork(DirectedGraph<string, double>, int[]) : void
    initMatrixes(int[]) : void
    initActivationsMetrix(int[]) : void
    initWeightsMetrix() : void
    NextMove(double[]) : int
    Input(double[]) : void
    Output() : int
    SigmoidFunction(double) : double
    SigmoidInputFunction(double) : double
    WeightsMulActivation(double[][], double[], double[]) : void
    FeedForward() : void
    PrintMetrix() : void
```

NeuralNetwork
• **DGraph** – גרף מכוון ממושקל שמייצג את הרשת • **Weights** – מערך תלת ממדי שמכיל מטריצות בגדלים שונים שכל מטריצה מכילה את המשקלים שנצרכים לחישוב בין שכבה לשכבה • **Layes** – מערך שמכיל מערכים בגדלים שונים במייצגים שכבות כל תא במערכים מכיל את יחידות המידע של הרשת באותה שכבה
• **NeuralNetwork** – פעולה בונה שמקבלת גרף ובונה את המטריצות • **UpdateNeuralNetworl** – מעדכן את רשת הנוירונים עם גרף חדש • **initMatrix** – לוקח את הגרך וממיר אותו לכל המטריצות מהמערכים • **initActivationsMetix** – מאתחל את המערך התלת ממדי שמכיל את המטריצות שמכילות את המשקלים לכל שכבה • **NextMove** – מקבל את הקלט ומחזיר איזה מהפלטים הוא התוצאה שיצאה • **Input** – מכניס את הנתונים של הקלט לתוך **Layes** ומעביר אותם דרך פונקציית סיגמויד • **OutPut**- מחזיר את היזה מהפלטים הוא הפלט של הצעד הבא שהולחט • **SigmoidFunction** – עושה את פונקציית האקטיבציה (סיגמויד) על ערך

- **SigmoidInputFunction** – עושה את פונקציית האקטיבציה (סיגמויד) על ערך שנכנס לתוך הקלטים של הרשת
- **WeightsMulActivation** – עושה את הפעולות המתמטיות של שכבה אחת – מקבל את מטריצת המשקלים, מערך של יחידות מידע המקור (שכבה שממנה יוצאים) ומערך של יחידות מידע היעד (השכבה שממנה מגיעיים ויכנס הערך החדש)
- **FeedForword** – עושה את פעולת **FeedForword** של רשת נוירונים. עושה את כל הפעולות בחישוביות בשביל כל בשכבות – מעביר את הקלט בכל החישובים לפלט
- **PrintMatrix** – מדפיס את המערכים ונמטריצות בשביל בדיקות

4 מדריך למשתמש:

1) הורד את התקייה AlphaCar.zip
2) פתח אותה
3) הרץ את הקובץ AlphaCar.exe



5 ביבליוגרפיה

1) https://towardsdatascience.com/everything-you-need-to-know-about-neural-networks-and-backpropagation-machine-learning-made-easy-e5285bc2be3a
2) https://visualstudiomagazine.com/articles/2013/06/01/neural-network-activation-functions.aspx
3) http://www.ai-junkie.com/ga/intro/gat2.html
4) https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_parent_selection.htm
5) Choosing Mutation and Crossover Ratios for Genetic Algorithms—A Review with a New Dynamic Approach Ahmad Hassanat 1,2,* ,†,‡ , Khalid Almohammadi 1 , Esra'a Alkafaween 2,‡ , Eman Abunawas 2 , Awni Hammouri 2 and V. B. Surya Prasath 3,4,5,6
6) Genetic algorithms for modelling and optimization by john McCall

6 קטעי קוד חשובים:

IGraph:

```
using System.Collections.Generic;

namespace GraphLibary
{
    public interface IGraph<T,K>
    {
        //add a Vertex
        bool AddVertex(T vertex);

        //add to teh graph the vertices, if vertex already exists then it is skipped
        void AddVertex(IEnumerable<T> vertexSet);

        //Deletes the vertex specified in the graph, if existing - return true
        bool DeleteVertex(T vertex);

        //Deletes every vertex of the set
        void DeleteVertex(IEnumerable<T> vertexSet);

        //add Edge - add a new edge to the graph, starting from v1 and ending in v2,
having weight w,
        //return true on success
        bool AddEdge(T v1, T v2, K weight);

        //Returns the weight of the edge starting from v1 and ending in v2
        K GetWeight(T v1, T v2);

        //Deletes the edge starting from v1 and ending in v2, if existing Returns true on
success
        bool DeleteEdge(T v1, T v2);

        //Returns true if v1 is adjacent to v2
        bool AreAdjacent(T v1, T v2);

        //Computes the degree of the specified vertex
        int Degree(T vertex);

        //Computes the outgoing degree of the specified vertex
        int OutDegree(T vertex);

        //Computes the ingoing degree of the specified vertex
        int InDegree(T vertex);

        //Retirns the number of vertices in the graph
        int VerticesNumber();

        //Returns the number of edges in the graph
        int EdgesNumber();

        //Returns a set of adjacent vertices to the one specified as argument
        IEnumerable<T> AdjacentVertices(T vertex);

        //Returns an IEnumerable containing the vertex set of the graph
        IEnumerable<T> GetVertexSet();

        //Returns an IEnumerable containing the edge set of the graph,
```

```
        // represented by couples of vertices
        IEnumerable<IPairValue<T>> GetEdgeSet();
    }

    // interface to simple data struckture - contain a pair of vertices
    public interface IPairValue<T>
    {
        //Returns the first vertex
        T GetFirst();

        //Returns the second vertex
        T GetSecond();

        //Returns true if givin value is storedin the data structure
        bool Contains(T value);
    }
}
```

קובץ – AbstractGraph:

```
using System;
using System.Collections.Generic;

namespace GraphLibary
{
    public abstract class AbstractGraph<T, K> : IGraph<T, K>
    {
        protected readonly List<T> VertexSet = new List<T>();// list of the vertices
        protected readonly List<IPairValue<T>> EdgeSet = new List<IPairValue<T>>();//
list of the edges
        //dictionary in order to associate for each edge a weight.
        //- keyset : PairValues ; Value - weight
        protected readonly Dictionary<IPairValue<T>, K> Weights = new
Dictionary<IPairValue<T>, K>();

        //add Edge - add a new edge to the graph, starting from v1 and ending in v2,
having weight w,
        //return true on success
        public abstract bool AddEdge(T v1, T v2, K weight);

        //add a Vertex
        public bool AddVertex(T vertex)
        {
            if (vertex == null)
                throw new ArgumentNullException();
            if (VertexSet.Contains(vertex))
                return false;
            VertexSet.Add(vertex);
            return true;
        }

        //add to teh graph the vertices, if vertex already exists then it is skipped
        public void AddVertex(IEnumerable<T> vertexSet)
        {
            if (vertexSet == null)
                throw new ArgumentNullException();
            using (var it = vertexSet.GetEnumerator())
            {
                while (it.MoveNext())
```

```
                {
                    if (it.Current != null && !VertexSet.Contains(it.Current))
                        VertexSet.Add(it.Current);
                }
            }
        }

        //Returns a set of adjacent vertices to the one specified as argument
        public abstract IEnumerable<T> AdjacentVertices(T vertex);

        //Returns true if v1 is adjacent to v2
        public abstract bool AreAdjacent(T v1, T v2);

        //Computes the degree of the specified vertex
        public abstract int Degree(T vertex);

        //Deletes the edge starting from v1 and ending in v2, if existing Returns true on
success
        public abstract bool DeleteEdge(T v1, T v2);

        //Deletes the vertex specified in the graph, if existing - return true
        public bool DeleteVertex(T vertex)
        {
            if (vertex == null)
                throw new ArgumentNullException();
            if (!VertexSet.Contains(vertex))
                return false;
            VertexSet.Remove(vertex);
            return true;
        }

        //Deletes every vertex of the set
        public void DeleteVertex(IEnumerable<T> vertexSet)
        {
            if (vertexSet == null)
                throw new ArgumentNullException();
            using (var it = vertexSet.GetEnumerator())
            {
                while (it.MoveNext())
                    if (it.Current != null)
                        VertexSet.Remove(it.Current);
            }
        }

        //Returns the number of edges in the graph
        public int EdgesNumber()
        {
            return EdgeSet.Count;
        }

        //Returns an IEnumerable containing the edge set of the graph,
        // represented by couples of vertices
        public IEnumerable<IPairValue<T>> GetEdgeSet()
        {
            return EdgeSet;
        }

        //Returns an IEnumerable containing the vertex set of the graph
```

```
        public IEnumerable<T> GetVertexSet()
        {
            return VertexSet;
        }

        //Returns the weight of the edge starting from v1 and ending in v2
        public abstract K GetWeight(T v1, T v2);

        //Computes the ingoing degree of the specified vertex
        public abstract int InDegree(T vertex);

        //Computes the outgoing degree of the specified vertex
        public abstract int OutDegree(T vertex);

        //Retirns the number of vertices in the graph
        public int VerticesNumber()
        {
            return VertexSet.Count;
        }
    }
}
```

קובץ – DiractedGraph:

```
using System;
using System.Collections.Generic;

namespace GraphLibary
{
    public class DirectedGraph<T, K> : AbstractGraph<T, K>
    {
        //add Edge - add a new edge to the graph, starting from v1 and ending in v2,
having weight w,
        //return true on success
        public override bool AddEdge(T v1, T v2, K weight)
        {
            if (v1 == null || v2 == null || weight == null)
                throw new ArgumentNullException();
            if (!VertexSet.Contains(v1) || !VertexSet.Contains(v2))
                return false;
            IPairValue<T> pair = new PairValueImplementation<T>(v1, v2);
            if (EdgeSet.Contains(pair))
                return false;
            EdgeSet.Add(pair);
            Weights[pair] = weight;
            return true;
        }

        //Returns a set of adjacent vertices to the one specified as argument
        public override IEnumerable<T> AdjacentVertices(T vertex)
        {
            foreach (IPairValue<T> p in EdgeSet)
                if (p.GetFirst().Equals(vertex))
                    yield return p.GetSecond();
        }

        //Returns true if v1 is adjacent to v2
        public override bool AreAdjacent(T v1, T v2)
        {
```

```
            if (v1 == null || v2 == null)
                throw new ArgumentNullException();
            if (!VertexSet.Contains(v1) || !VertexSet.Contains(v2))
                throw new ArgumentException();
            return EdgeSet.Contains(new PairValueImplementation<T>(v1, v2));
        }

        //Computes the degree of the specified vertex
        public override int Degree(T vertex)
        {
            if (vertex == null)
                throw new ArgumentNullException();
            if (!VertexSet.Contains(vertex))
                throw new ArgumentException();
            return InDegree(vertex) + OutDegree(vertex);
        }

        //Deletes the edge starting from v1 and ending in v2, if existing Returns true on
success
        public override bool DeleteEdge(T v1, T v2)
        {
            if (v1 == null || v2 == null)
                throw new ArgumentNullException();
            IPairValue<T> pair = new PairValueImplementation<T>(v1, v2);
            if(EdgeSet.Contains(pair))
            {
                EdgeSet.Remove(pair);
                Weights.Remove(pair);
                return true;
            }
            return false;
        }

        //Returns the weight of the edge starting from v1 and ending in v2
        public override K GetWeight(T v1, T v2)
        {
            if (v1 == null || v2 == null)
                throw new ArgumentNullException();
            IPairValue<T> pair = new PairValueImplementation<T>(v1, v2);
            if (!Weights.ContainsKey(pair))
                throw new ArgumentException();
            return Weights[pair];
        }

        //Computes the ingoing degree of the specified vertex
        public override int InDegree(T vertex)
        {
            if (vertex == null)
                throw new ArgumentNullException();
            if (!VertexSet.Contains(vertex))
                throw new ArgumentException();
            int counter = 0;
            foreach (var pair in EdgeSet)
                if (pair.GetSecond().Equals(vertex))
                    counter++;
            return counter;
        }
```

```
        //Computes the outgoing degree of the specified vertex
        public override int OutDegree(T vertex)
        {
            if (vertex == null)
                throw new ArgumentNullException();
            if (!VertexSet.Contains(vertex))
                throw new ArgumentException();
            int counter = 0;
            foreach (var pair in EdgeSet)
                if (pair.GetFirst().Equals(vertex))
                    counter++;
            return counter;
        }
    }
}
```

קובץ – Pair:

```
using System;

namespace GraphLibary
{
    // interface to simple data struckture - contain a pair of vertices
    public class PairValueImplementation<T> : IPairValue<T>
    {
        private readonly T _t1;//first vertex
        private readonly T _t2;//second vetex

        //init
        public PairValueImplementation(T t1, T t2)
        {
            if (t1 == null || t2 == null)
                throw new ArgumentNullException();
            if (t1.GetType() != t2.GetType())
                throw new AccessViolationException();
            _t1 = t1;
            _t2 = t2;
        }

        //Returns true if givin value is storedin the data structure
        public bool Contains(T value)
        {
            return value.Equals(_t1) || value.Equals(_t2);
        }

        //Returns the first vertex
        public T GetFirst()
        {
            return _t1;
        }

        //Returns the second vertex
        public T GetSecond()
        {
            return _t2;
        }

        public override bool Equals(object obj)
        {
```

```
            if (obj == null || obj.GetType() != typeof(PairValueImplementation<T>))
                return false;
            PairValueImplementation<T> casted = (PairValueImplementation<T>)obj;
            return casted._t1.Equals(_t1) && casted._t2.Equals(_t2);
        }

        //returns the hash of t1 +t2
        public override int GetHashCode()
        {
            return _t1.GetHashCode() + _t2.GetHashCode();
        }
    }
}


using System;
using System.Collections.Generic;
using System.Text;
using GraphLibary;
using System.Linq;
static class Constants
{
    public const double SigmoidAPara = 1;
}

namespace NeuralNetworkLibary
{
    public class NeuralNetwork : INeuralNetwork
    {
        private DirectedGraph<string, double> DGraph;
        private double[][][] weights;
        private double[][] layes;

        //Constractor
        //1) dGraph - a Directed Graph that will represent the NN
        //2) layerCount - an array in length of the number of layers in the NN/ the graph
        //                each sell will contain the number of activations
        public NeuralNetwork(DirectedGraph<string, double> dGraph, int[] layerCount)
        {
            this.DGraph = dGraph;
            this.initMatrixes(layerCount);
        }

        //Update the NN same as constactor
        //1) dGraph - a Directed Graph that will represent the NN
        //2) layerCount - an array in length of the number of layers in the NN/ the graph
        //
        public void UpdateNeuralNetwork(DirectedGraph<string, double> dGraph, int[]
layerCount)
        {
            this.DGraph = dGraph;
            this.initMatrixes(layerCount);
        }

        //init the matrixes
        // init the weights metrix and the layes metrix that contain the activations
        //1) layerCount - an array in length of the number of layers in the NN/ the graph
        private void initMatrixes(int[] layerCount)
```

```
        {
            if (layerCount == null)
                throw new ArgumentException();
            this.initActivationsMetrix(layerCount);
            this.initWeightsMetrix();
        }

        //init the layes metrix that contain the activations
        //1) layerCount - an array in length of the number of layers in the NN/ the graph
        private void initActivationsMetrix(int[] layerCount)
        {
            if (layerCount == null)
                throw new ArgumentNullException();
            this.layes = new double[layerCount.Length][];
            for (int i = 0; i < layerCount.Length; i++)
            {
                this.layes[i] = new double[layerCount[i]];
            }
        }

        //init the weights metrix
        private void initWeightsMetrix()
        {
            this.weights = new double[this.layes.Length - 1][][];
            for (int i = 0; i < this.weights.Length; i++)
            {
                this.weights[i] = new double[this.layes[i + 1].Length][];
                for (int j = 0; j < this.weights[i].Length; j++)
                {
                    this.weights[i][j] = new double[this.layes[i].Length];
                }
            }
            List<IPairValue<string>> EdgeSet = this.DGraph.GetEdgeSet().ToList();
            IPairValue<string> edge;
            int indexWeight = 0;
            for (int i = 0; i < this.weights.Length; i++)
            {
                for (int j = 0; j < this.weights[i].Length; j++)
                {
                    for (int k = 0; k < this.weights[i][j].Length; k++)
                    {
                        edge = EdgeSet[indexWeight++];
                        this.weights[i][j][k] = this.DGraph.GetWeight(edge.GetFirst(),
edge.GetSecond());
                    }
                }
            }
        }

        //return the output from the all NN with a given input
        //1)inputs - an array if doubles that contain the givin inputs
        public int NextMove(double[] inputs)
        {
            Input(inputs);
            FeedForward();
            return Output();
        }
```

```
        //Input
        //place the inputs in the first layer of the activations (the inputs ones)
        private void Input(double[] inputs)
        {
            for (int i = 0; i < inputs.Length; i++)
            {
                this.layes[0][i] = SigmoidInputFunction(inputs[i]);
                // this.layes[0][i] = (inputs[i]);
            }
        }

        //Output
        //return the number of the output node
        private int Output()
        {
            int maxIndex = 0;
            for (int i = 0; i < this.layes[this.layes.Length - 1].Length; i++)
            {
                if (this.layes[this.layes.Length - 1][i] > this.layes[this.layes.Length -
1][maxIndex])
                    maxIndex = i;
            }
            return maxIndex;
        }

        //the math function for Sigmoid Funciton
        //the function gaets a nubmer and return its value after the sigmoid funtion
        // f(x) = 1/(1+e^(-a*x)) ; a = Constants.SigmoidAPara
        //1)x - a double number that cintain the x value
        private double SigmoidFunction(double x)
        {
            double ret = 1 / (1 + Math.Pow(Math.E, (-1 * Constants.SigmoidAPara *
(x+2))));
            return ret;
        }

        private double SigmoidInputFunction(double x)
        {
            double ret = 1 / (1 + Math.Pow(Math.E, (-1 * Constants.SigmoidAPara * (x-
25))));
            return ret;
        }

        // mul the weight metrix with the activations
        //saves the answer in the next layer
        private void WeightsMulActivation(double[][] curr_weights, double[] SrcLayer,
double[] DesLayer)
        {
            for (int i = 0; i < curr_weights.Length; i++)
            {
                for (int j = 0; j < curr_weights[i].Length; j++)
                {
                    DesLayer[i] += curr_weights[i][j] * SrcLayer[j];
                }
                DesLayer[i] = SigmoidFunction(DesLayer[i]);
                //DesLayer[i] = (DesLayer[i]);

            }
```

```
        }

        //feed the input forward in the inputs to the output layer
        private void FeedForward()
        {
            for (int i = 0; i < this.weights.Length; i++)
            {
                WeightsMulActivation(this.weights[i], this.layes[i], this.layes[i + 1]);
            }
        }

        //print the metrixes
        public void PrintMetrix()
        {
            for (int i = 0; i < this.layes.Length; i++)
            {
                Console.WriteLine("Layer:" + i);
                for (int j = 0; j < this.layes[i].Length; j++)
                {
                    Console.Write(this.layes[i][j] + ", ");
                }
                Console.WriteLine();
                Console.WriteLine();
            }
            Console.WriteLine();
            for (int i = 0; i < this.weights.Length; i++)
            {
                Console.WriteLine("Layer:" + i);
                for (int j = 0; j < this.weights[i].Length; j++)
                {
                    Console.WriteLine();
                    for (int k = 0; k < this.weights[i][j].Length; k++)
                    {
                        Console.Write(this.weights[i][j][k] + ", ");
                    }
                }
                Console.WriteLine();
                Console.WriteLine();
            }
        }
    }
}
```

:NeuralNetwork

```
using System;
using System.Collections.Generic;
using System.Text;
using GraphLibary;
using System.Linq;
static class Constants
{
    public const double SigmoidAPara = 1;
}

namespace NeuralNetworkLibary
{
    public class NeuralNetwork : INeuralNetwork
    {
        private DirectedGraph<string, double> DGraph;
        private double[][][] weights;
        private double[][] layes;

        //Constractor
        //1) dGraph - a Directed Graph that will represent the NN
        //2) layerCount - an array in length of the number of layers in the NN/ the graph
        //                each sell will contain the number of activations
        public NeuralNetwork(DirectedGraph<string, double> dGraph, int[] layerCount)
        {
            this.DGraph = dGraph;
            this.initMatrixes(layerCount);
        }

        //Update the NN same as constactor
        //1) dGraph - a Directed Graph that will represent the NN
        //2) layerCount - an array in length of the number of layers in the NN/ the graph
        //
        public void UpdateNeuralNetwork(DirectedGraph<string, double> dGraph, int[]
layerCount)
        {
            this.DGraph = dGraph;
            this.initMatrixes(layerCount);
        }

        //init the matrixes
        // init the weights metrix and the layes metrix that contain the activations
        //1) layerCount - an array in length of the number of layers in the NN/ the graph
        private void initMatrixes(int[] layerCount)
        {
            if (layerCount == null)
                throw new ArgumentException();
            this.initActivationsMetrix(layerCount);
            this.initWeightsMetrix();
        }
```

```
        //init the layes metrix that contain the activations
        //1) layerCount - an array in length of the number of layers in the NN/ the graph
        private void initActivationsMetrix(int[] layerCount)
        {
            if (layerCount == null)
                throw new ArgumentNullException();
            this.layes = new double[layerCount.Length][];
            for (int i = 0; i < layerCount.Length; i++)
            {
                this.layes[i] = new double[layerCount[i]];
            }
        }

        //init the weights metrix
        private void initWeightsMetrix()
        {
            this.weights = new double[this.layes.Length - 1][][];
            for (int i = 0; i < this.weights.Length; i++)
            {
                this.weights[i] = new double[this.layes[i + 1].Length][];
                for (int j = 0; j < this.weights[i].Length; j++)
                {
                    this.weights[i][j] = new double[this.layes[i].Length];
                }
            }
            List<IPairValue<string>> EdgeSet = this.DGraph.GetEdgeSet().ToList();
            IPairValue<string> edge;
            int indexWeight = 0;
            for (int i = 0; i < this.weights.Length; i++)
            {
                for (int j = 0; j < this.weights[i].Length; j++)
                {
                    for (int k = 0; k < this.weights[i][j].Length; k++)
                    {
                        edge = EdgeSet[indexWeight++];
                        this.weights[i][j][k] = this.DGraph.GetWeight(edge.GetFirst(),
edge.GetSecond());
                    }
                }
            }
        }

        //return the output from the all NN with a given input
        //1)inputs - an array if doubles that contain the givin inputs
        public int NextMove(double[] inputs)
        {
            Input(inputs);
            FeedForward();
            return Output();
        }

        //Input
        //place the inputs in the first layer of the activations (the inputs ones)
        private void Input(double[] inputs)
        {
            for (int i = 0; i < inputs.Length; i++)
            {
```

```
                this.layes[0][i] = SigmoidInputFunction(inputs[i]);
                // this.layes[0][i] = (inputs[i]);
            }
        }

        //Output
        //return the number of the output node
        private int Output()
        {
            int maxIndex = 0;
            for (int i = 0; i < this.layes[this.layes.Length - 1].Length; i++)
            {
                if (this.layes[this.layes.Length - 1][i] > this.layes[this.layes.Length -
1][maxIndex])
                    maxIndex = i;
            }
            return maxIndex;
        }

        //the math function for Sigmoid Funciton
        //the function gaets a nubmer and return its value after the sigmoid funtion
        // f(x) = 1/(1+e^(-a*x)) ; a = Constants.SigmoidAPara
        //1)x - a double number that cintain the x value
        private double SigmoidFunction(double x)
        {
            double ret = 1 / (1 + Math.Pow(Math.E, (-1 * Constants.SigmoidAPara *
(x+2))));
            return ret;
        }

        private double SigmoidInputFunction(double x)
        {
            double ret = 1 / (1 + Math.Pow(Math.E, (-1 * Constants.SigmoidAPara * (x-
25))));
            return ret;
        }

        // mul the weight metrix with the activations
        //saves the answer in the next layer
        private void WeightsMulActivation(double[][] curr_weights, double[] SrcLayer,
double[] DesLayer)
        {
            for (int i = 0; i < curr_weights.Length; i++)
            {
                for (int j = 0; j < curr_weights[i].Length; j++)
                {
                    DesLayer[i] += curr_weights[i][j] * SrcLayer[j];
                }
                DesLayer[i] = SigmoidFunction(DesLayer[i]);
                //DesLayer[i] = (DesLayer[i]);

            }
        }

        //feed the input forward in the inputs to the output layer
        private void FeedForward()
        {
            for (int i = 0; i < this.weights.Length; i++)
```

```
            {
                WeightsMulActivation(this.weights[i], this.layes[i], this.layes[i + 1]);
            }
        }

        //print the metrixes
        public void PrintMetrix()
        {
            for (int i = 0; i < this.layes.Length; i++)
            {
                Console.WriteLine("Layer:" + i);
                for (int j = 0; j < this.layes[i].Length; j++)
                {
                    Console.Write(this.layes[i][j] + ", ");
                }
                Console.WriteLine();
                Console.WriteLine();
            }
            Console.WriteLine();
            for (int i = 0; i < this.weights.Length; i++)
            {
                Console.WriteLine("Layer:" + i);
                for (int j = 0; j < this.weights[i].Length; j++)
                {
                    Console.WriteLine();
                    for (int k = 0; k < this.weights[i][j].Length; k++)
                    {
                        Console.Write(this.weights[i][j][k] + ", ");
                    }
                }
                Console.WriteLine();
                Console.WriteLine();
            }
        }
    }
}
```

מאמן – CarCollider:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
using System.IO;
using System;

public class CarCollision : MonoBehaviour
{
   public ArcadeCar check; //the main object - the car

   /// <summary>
   /// This Function Sums all the fitness scors
   /// </summary>
   /// <param name="fileName"> The file name of the file ahtat all the fitness scores are
in</param>
   /// <returns> the sum of the scores</returns>
   double GetFitnessScoreSum(string fileName)
   {
      double SumOfScore = 0;
      using (BinaryReader br = new
BinaryReader(File.Open("C:\\Users\\yotam\\Desktop\\ArcadeCarPhysics-
master\\Assets\\Scripts\\" + fileName, FileMode.Open)))
      {
         for (int j = 0; j < 50; j++)
         {
            br.ReadInt32();
            SumOfScore += br.ReadDouble();
         }
      }
      Debug.Log("Sum Of FitnessScores: " + SumOfScore);
      return SumOfScore;
```

```
    }

    /// <summary>
    /// This function calculates the Probability of each thought to be mutate to the next gen
    /// Thought`s Probability = Probability before + (current Fitness / Sum of Futness)
    /// </summary>
    /// <param name="fileName"></param>
    /// <returns> returns an array with all the probebilities for each thought to get picked for next
gen
    /// the sum of all probebilities is 1 each Probability is between 0-1</returns>
    double[] GetThoughtsProbability(string fileName)
    {
        double SumOfScore = GetFitnessScoreSum(fileName);
        double[] ThoughtsProbability = new double[50];
        double SumOfProbability = 0.0;
        double currFit;
        using (BinaryReader br = new
BinaryReader(File.Open("C:\\Users\\yotam\\Desktop\\ArcadeCarPhysics-
master\\Assets\\Scripts\\" + fileName, FileMode.Open)))
        {
            for (int j = 0; j < 50; j++)
            {
                br.ReadInt32();
                currFit = br.ReadDouble();
                ThoughtsProbability[j] = SumOfProbability + (currFit / SumOfScore);
                SumOfProbability += ThoughtsProbability[j];
            }
        }
        Debug.Log("Thought Probability done");
        for (int i = 0; i < 50; i++)
        {
            Debug.Log("Prob: " + ThoughtsProbability[i]);
        }
        return ThoughtsProbability;
    }
```

```
/// <summary>
/// this funciton doing a Roulette for all the thoughts and get out
/// the thoughts that came out (the Thoughts Probability is based on its fitnessscore)
/// </summary>
/// <param name="fileName"> the file to read the thoughts and calculate the probability with
/// GetThoughtsProbability function</param>
/// <param name="numberOfThoughts"> the number of wanted thoughts to get</param>
/// <returns> an array size of numberOfThoughts the array contains the intexes for the pickes
thoughts to be mutated</returns>
int[] ThoughtsRouletteWheelSelection(string fileName, int numberOfThoughts)
{
    System.Random random = new System.Random();
    double[] ThoughtsProbability = GetThoughtsProbability(fileName);
    int[] ThoughtIndexs = new int[numberOfThoughts];
    for (int i = 0; i < numberOfThoughts; i++)
    {
        double target = random.NextDouble();
        for (int j = 0; j < 50; j++)
        {
            if(j == 49)
            {
                if (target >= ThoughtsProbability[j])
                {
                    ThoughtIndexs[i] = j;
                    break;
                }
            }
            else if(target >= ThoughtsProbability[j] && target < ThoughtsProbability[j+1])
            {
                ThoughtIndexs[i] = j;
                break;
            }
        }
    }
    Debug.Log("Roulette has been done");
    return ThoughtIndexs;
```

```
    }

    /// <summary>
    /// this funciton return an array of the iindexses of the best thoughts
    /// </summary>
    /// <param name="NumberOfThoughts"> the number of best thoughts index to return</param>
    /// <returns>return an array of the iindexses of the best thoughts </returns>
    double[] GetWeightsIndex(int NumberOfThoughts)
    {
        double[] WeightsIndexs = new double[NumberOfThoughts];
        for (int i = 0; i < NumberOfThoughts; i++)
        {
            WeightsIndexs[i] = -1;
        }
        using (BinaryReader br = new
BinaryReader(File.Open("C:\\Users\\yotam\\Desktop\\ArcadeCarPhysics-
master\\Assets\\Scripts\\FitnessScores.bin", FileMode.Open)))
        {
            for (int i = 0; i < NumberOfThoughts; i++)
            {
                br.BaseStream.Seek(0, SeekOrigin.Begin);
                double maxScore = -1;
                int maxIndex = -1;
                int tempIndex;
                double tempScore;
                for (int j = 0; j < 50; j++)
                {
                    tempIndex = br.ReadInt32();
                    tempScore = br.ReadDouble();
                    if (tempScore > maxScore)
                    {
                        if(Array.IndexOf(WeightsIndexs, (double)tempIndex) < 0)
                        {
                            maxIndex = tempIndex;
                            maxScore = tempScore;
                        }
```

```
            }
        }
        WeightsIndexs[i] = (double)maxIndex;
      }
    }
    return WeightsIndexs;
  }

  /// <summary>
  /// this function reads from the weights file the strongers thoughts
  /// </summary>
  /// <param name="NumberOfThoughts">the number of best thoughts index to return</param>
  /// <returns> a pointer(array) to array of pointer ([][]) that each array in the array of arrays
contain an array of wegiths </returns>
  double[][] GetStrongest(int NumberOfThoughts)
  {
    double[][] RetWeights = new double[NumberOfThoughts][];
    for (int i = 0; i < NumberOfThoughts; i++)
    {
      RetWeights[i] = new double[48];
    }
    double[] WeightsIndexes = GetWeightsIndex(NumberOfThoughts);
    using (BinaryReader br = new
BinaryReader(File.Open("C:\\Users\\yotam\\Desktop\\ArcadeCarPhysics-
master\\Assets\\Scripts\\Weights.bin", FileMode.Open)))
    {
      for (int i = 0; i < NumberOfThoughts; i++)
      {
        br.BaseStream.Seek(0, SeekOrigin.Begin);
        // 2.
        // Important variables:
        int length = (int)br.BaseStream.Length;
        int pos = 50000;
        int required = 2000;
        int count = 0;
        int currIndex = 0;
```

```
            // 3.
            // Seek the required index.
            //br.BaseStream.Seek(pos, SeekOrigin.Begin);
            currIndex = br.ReadInt32();
            while (currIndex != WeightsIndexes[i])
            {
                br.BaseStream.Seek(48 * sizeof(double), SeekOrigin.Current);
                currIndex = br.ReadInt32();
            }
            // 4.
            // Slow loop through the bytes.
            string wet = "";
            for (int k = 0; k < 48; k++)
            {
                RetWeights[i][k] = br.ReadDouble();
                wet += RetWeights[i][k].ToString() + ", ";
            }
            Debug.Log(i + " S Weights: " + wet);
        }
    }
    return RetWeights;
}

/// <summary>
/// this function copy the best thought to the new gen
/// </summary>
/// <param name="fileName">the file to write into</param>
/// <param name="numberOfBestThoughts">the number of best thought to copy</param>
void CopyBestThoughtsToNewGen(string fileName, int numberOfBestThoughts)
{
    BinaryWriter bw;
    FileStream fs;
    double[][] weights = GetStrongest(numberOfBestThoughts);
    try
    {
```

```
        fs = new FileStream("C:\\Users\\yotam\\Desktop\\ArcadeCarPhysics-
master\\Assets\\Scripts\\" + fileName, FileMode.Create);
        bw = new BinaryWriter(fs);
      }
      catch (IOException exe)
      {
        Console.WriteLine(exe.Message + "\\Cannot open file");
        return;
      }
      string wet = "";
      for (int j = 0; j < numberOfBestThoughts; j++)
      {
        bw.Write(j);
        for (int i = 0; i < 48; i++)
        {
          wet += weights[j][i].ToString() + ", ";
          bw.Write(weights[j][i]);
        }
        Console.WriteLine("Write to next gen: " + wet);
      }
      bw.Close();
      fs.Close();
    }

    /// <summary>
    /// this function creates and copt new random weights to the new gen
    /// </summary>
    /// <param name="fileName">the file to write into</param>
    /// <param name="startIndex">the start index of the new thoughts</param>
    /// <param name="NewThoughts">the number of new random thoughts wanted</param>
    void CopyCreateRandThoughtToNewGen(string fileName, int startIndex, int NewThoughts)
    {
      BinaryWriter bw;
      FileStream fs;
      try
      {
```

```
        fs = new FileStream("C:\\Users\\yotam\\Desktop\\ArcadeCarPhysics-
master\\Assets\\Scripts\\" + fileName, FileMode.Create);
        bw = new BinaryWriter(fs);
      }
      catch (IOException exe)
      {
        Console.WriteLine(exe.Message + "\\Cannot open file");
        return;
      }
      System.Random rnd = new System.Random();
      double[] weights = new double[48];
      string wet = "";
      for (int j = startIndex; j < startIndex+NewThoughts; j++)
      {
        bw.Write(j);
        for (int i = 0; i < 48; i++)
        {
          weights[i] = rnd.Next(-11, 11);
          wet += weights[i].ToString() + ", ";
          bw.Write(weights[i]);
        }
        Console.WriteLine("Write to next gen: " + wet);
      }
      bw.Close();
      fs.Close();
    }

    /// <summary>
    /// gets a thought out of a file by its index
    /// </summary>
    /// <param name="fileName"> the fiel to search in</param>
    /// <param name="ThoughtIndex"> the index of the </param>
    /// <returns></returns>
    double[] GetThoughtFromFile(string fileName, int ThoughtIndex)
    {
      double[] weights = new double[48];
```

```
    using (BinaryReader br = new
BinaryReader(File.Open("C:\\Users\\yotam\\Desktop\\ArcadeCarPhysics-
master\\Assets\\Scripts\\" + fileName, FileMode.Open)))
    {
      // 2.
      // Important variables:
      int length = (int)br.BaseStream.Length;
      int pos = 50000;
      int required = 2000;
      int count = 0;
      int currIndex = 0;
      // 3.
      // Seek the required index.
      //br.BaseStream.Seek(pos, SeekOrigin.Begin);
      currIndex = br.ReadInt32();
      while (currIndex != ThoughtIndex)
      {
        br.BaseStream.Seek(48 * sizeof(double), SeekOrigin.Current);
        currIndex = br.ReadInt32();
      }
      // 4.
      // Slow loop through the bytes.
      string wet = "";
      for (int i = 0; i < 48; i++)
      {
        weights[i] = br.ReadDouble();
        wet += weights[i].ToString() + ", ";
      }
      Debug.Log("Thought Returnd for CreateOffspringFrom2Perents is: " + wet);
    }
    return weights;
  }

  /// <summary>
  /// this function craetes two offsprings from two perents and put then in aa metrix
  /// </summary>
```

```
/// <param name="fileName">the file that contian all the weights</param>
/// <param name="Perent1Index">index of first perent</param>
/// <param name="Perent2Index">index of second perent</param>
/// <returns> a matrix in a size of 2*48 - 2 offsprings that their size is 48 numbers
(weights)</returns>
double[][] CreateOffspringFrom2Perents(string fileName, int Perent1Index, int Perent2Index)
{
    int iSswitch;
    System.Random rand = new System.Random();
    double[][] Perents = new double[2][];
    Perents[0] = GetThoughtFromFile(fileName, Perent1Index);
    Perents[1] = GetThoughtFromFile(fileName, Perent2Index);
    for (int i = 0; i < 48; i++)
    {
        iSswitch = rand.Next(2);
        if (iSswitch == 1)
        {
            Perents[0][i] += Perents[1][i];
            Perents[1][i] = Perents[0][i] - Perents[1][i];
            Perents[0][i] -= Perents[1][i];
        }
    }
    return Perents;
}

/// <summary>
/// this funciton writes wegihts in to a file
/// </summary>
/// <param name="fileName">the file to write into</param>
/// <param name="exsistThoughts">number of already existing thiughts in the file</param>
/// <param name="ThoughtsToWrite"> nubmer of thoughts givien</param>
/// <param name="Weights">a matrix that cintain the new thoughts</param>
void WriteWeightsToFile(string fileName, int exsistThoughts, int ThoughtsToWrite, double[][]
Weights)
{
    BinaryWriter bw;
```

```
        FileStream fs;
        try
        {
            fs = new FileStream("C:\\Users\\yotam\\Desktop\\ArcadeCarPhysics-
master\\Assets\\Scripts\\" + fileName, FileMode.Create);
            bw = new BinaryWriter(fs);
        }
        catch (IOException exe)
        {
            Console.WriteLine(exe.Message + "\\Cannot open file");
            return;
        }
        System.Random rnd = new System.Random();
        string wet = "";
        for (int j = 0; j < ThoughtsToWrite; j++)
        {
            bw.Write(j+ exsistThoughts);
            for (int i = 0; i < 48; i++)
            {
                wet += Weights[j][i].ToString() + ", ";
                bw.Write(Weights[j][i]);
            }
            Console.WriteLine("index: " + j + " | Write to next gen: " + wet);
        }
        bw.Close();
        fs.Close();
    }

    /// <summary>
    /// this function uses CreateOffspringFrom2Perents, WriteWeightsToFile,
ThoughtsRouletteWheelSelection
    /// to do the selection, muration, and next get part of GA
    /// </summary>
    /// <param name="FitFileName">the file name that contains all the fit scores of the
thouvhts</param>
    /// <param name="weightFile">the file name that contain all the old gen weights</param>
```

```
/// <param name="ToWriteFile">the name of the file that woll ocntain the new gen weights</param>
/// <param name="exsistThoughts">already existing thoughts in the new gen</param>
void CreateOffspringAndWriteToFile(string FitFileName, string weightFile,string ToWriteFile, int exsistThoughts)
{
    int[] PerentsIndex = ThoughtsRouletteWheelSelection(FitFileName, (50 - exsistThoughts));
    for (int i = 0; i < PerentsIndex.Length; i+=2)
    {
        double[][] Offsprings = CreateOffspringFrom2Perents(weightFile, PerentsIndex[i], PerentsIndex[i+1]);
        WriteWeightsToFile(ToWriteFile, exsistThoughts + i, 2, Offsprings);
    }
}

/// <summary>
/// this funciotn creates and writes into file the new gen thoughts
/// </summary>
/// <param name="ToWriteFile">the name of the file to write the new gen</param>
/// <param name="ToReadFile"> the name of the file to read the old gen</param>
/// <param name="numOfBestThoughts">number of nest thoughts to pass to new gen</param>
/// <param name="numOfRandThought">number of random thoughts to pass to new gen</param>
void WriteNewThoughtsToNextGen(string ToWriteFile, string ToReadFile, int numOfBestThoughts, int numOfRandThought)
{
    WriteBestThoughtSession("BestOfEachGen.bin");
    CopyBestThoughtsToNewGen(ToWriteFile, numOfBestThoughts); //1
    CopyCreateRandThoughtToNewGen(ToWriteFile, numOfBestThoughts, numOfRandThought); //3
    CreateOffspringAndWriteToFile("FitnessScores.bin", ToReadFile, ToWriteFile, numOfBestThoughts + numOfRandThought);
}

/// <summary>
```

```
/// this funtion writes the best thought from the old gen to a file that will ocntain all hte best of
each gen
/// </summary>
/// <param name="fileName">ht ename of the file that will ocntain all hte best of each gen
</param>
void WriteBestThoughtSession(string fileName)
{
    int CurrGen;
    using (BinaryReader br = new
BinaryReader(File.Open("C:\\Users\\yotam\\Desktop\\ArcadeCarPhysics-
master\\Assets\\Scripts\\CurrGenNumber.bin", FileMode.Open)))
    {
        // 2.
        // Important variables:
        // 3.
        // Seek the required index.
        //br.BaseStream.Seek(pos, SeekOrigin.Begin);
        CurrGen = br.ReadInt32();
    }
    BinaryWriter bw;
    FileStream fs;
    try
    {
        fs = new FileStream("C:\\Users\\yotam\\Desktop\\ArcadeCarPhysics-
master\\Assets\\Scripts\\" + fileName, FileMode.Create);
        bw = new BinaryWriter(fs);
    }
    catch (IOException exe)
    {
        Console.WriteLine(exe.Message + "\\Cannot open file");
        return;
    }
    double[][] weights = GetStrongest(1);
    string wet = "";
    CurrGen++;
    bw.Write(CurrGen);
```

```
    for (int i = 0; i < 48; i++)
    {
        wet += weights[0][i].ToString() + ", ";
        bw.Write(weights[0][i]);
    }
    Console.WriteLine("Write to next gen: " + wet);
    bw.Close();
    fs.Close();
  }

  //called when ther is a Collision with the car hitbox
  //gets - collisioninfo - the Collision information
  void OnCollisionEnter(Collision collisioninfo)
  {
    //checks if the Collision accured with a Barrier
    if (collisioninfo.collider.tag == "Barrier")
    {
        //print and log the collision basic info
        //Debug.Log("Crash in: " + collisioninfo.collider.name);
        //disable the car
        //movement.enabled = false;
        //restart spot
        //movement.transform.position = new Vector3((float)-178.59, (float)2.6, (float)162.12);
        //movement.transform.eulerAngles = new Vector3((float)0, (float)0, (float)0);
        //restart game
        if (!collisioninfo.collider.name.Equals("Barrier (11)") &&
!collisioninfo.collider.name.Equals("Barrier (27)"))
        {
            check.SetWeightSetIndex(check.GetWeightSetIndex() + 1);
            Debug.Log("index: " + check.GetWeightSetIndex().ToString());
            if (check.GetWeightSetIndex() <= 49)
                SceneManager.LoadScene("SandBox");
            else
            {
                WriteNewThoughtsToNextGen("TempData.bin", "Weights.bin", 1, 3);
                check.enabled = false;
```

```
            }
        }
        else if(collisioninfo.collider.name.Equals("Barrier (158)"))
        {
            check.SetNextMove("left");
        }
        else
        {
            check.SetNextMove("right");
        }
    }
  }
}
```