

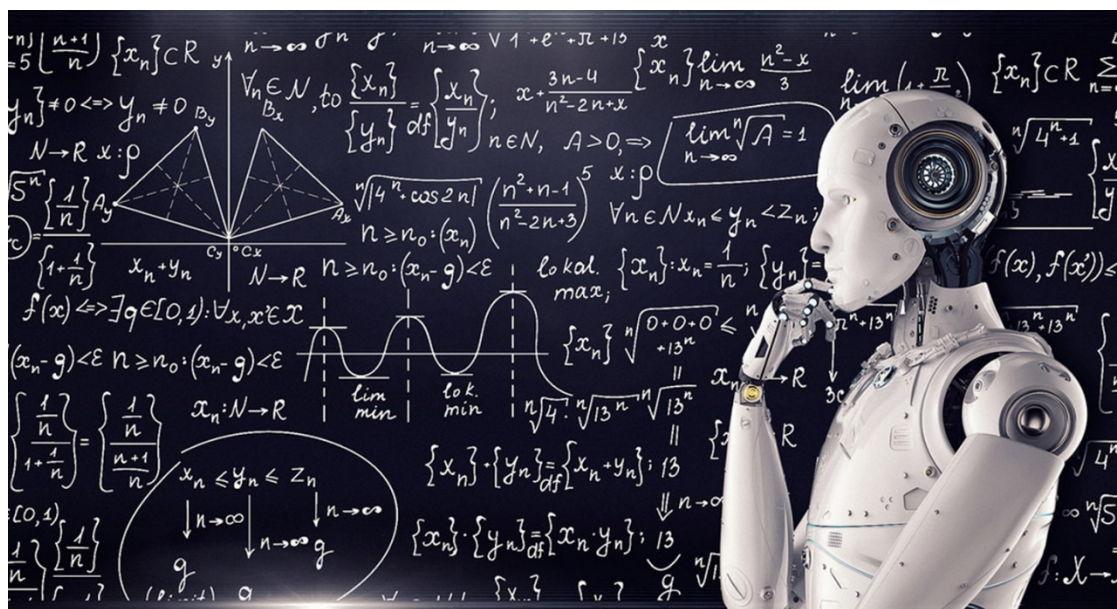
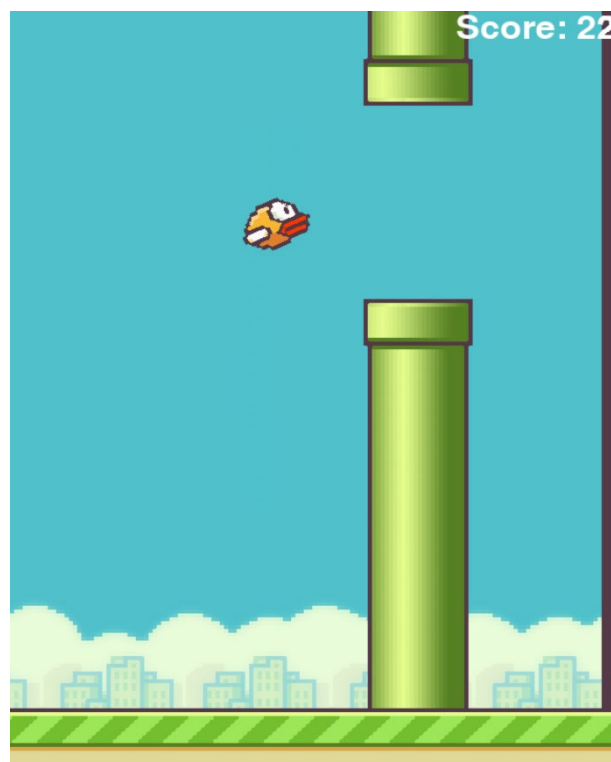
# Flappy bird – AI NEAT

מגיש: שון סירוטה

בית ספר: הכפר הירוק

כיתה: יב"9

מנחה: יודה אור



## תוכן עניינים

3	הקדמה
4	מבוא
5	רפלקציה
6	תוכנת פייתון והספריות שהשתמשתי בפייתון
6	תוכנות שבהן נעזרת
7	OOP
8	המשחק
9-11	הצגת המשחק ותוצאות
12	UML תרשים
13	הקלאסים
14	הפונקציות
14	הקפיצה של הציפור - Jump
15	התזוזה של הציפור
16	תזוזת כנפי הציפור
17	התנגשות בין ציפור ובאובייקט - masks
18-19	עולם ממושך
20	בינה מלאכותית
21	הקדמה
21	למידת מכונה
22	סוגי ML
23	Supervised ML : בפיקוח
24	Unsupervised ML : למידה לא בפיקוח
25	Reinforcement learning : לימוד בעזרת חיזוקים
26	Neural Network
27-29	הסבר על דרך הפעולה של רשת הנורונים והגע לפלט הסופי
30-31	האימון של הרשת
32	Evolutionary Algorithm – אלגוריתמים אבולוציוניים
32-34	Genetic Algorithm
35	Neat – efficient evolution of neural network topologies
36-38	הסבר על הדרך שNeat עובד
39-41	האלגוריתם Neat מופעל בפרויקט שלי
42-45	הקוד של הבינה מלאכותית
46-49	הקוד המעשי
50-56	הקוד
56-59	Config file

# הקדמה

## מבוא

### הפרויקט

הפרויקט שעשיתי הוא משחק Flappy bird, שהמחשב לומד לשחק בעצמו. בנוסף לכך, ניסיתי לעשות את הפרויקט כמה שיותר יעיל. הפרויקט מחולק לשני חלקים, בניית המשחק עצמו תוך תכנות מונחה עצמים - Object-oriented programming, ובניית AI. כאמור, המשחק משחק בעצמו (ML) שמבוסס על NEAT - NeuroEvolution of Augmenting Topologies שזה בעצם Genetic Algorithm וזה הנושא העיקרי של הפרויקט שלי. כדי לבנות את המשחק נעזרת בpygame, שזו מחלקה שמיועדת לבניית משחקים בפייתון. בפרויקט מתואר תהליך בניית הקוד והקוד עצמו. בנוסף אסביר על תהליך קבלת ההחלטות שלי, החל מבחירת העיצוב של המשחק ועד לשימוש בכלים המתקדמים שיש לעולם Machine learning להציע. כמובן שאסביר את הלוגיקה מאחורי genetic Algorithm ואסביר כיצד פועלת רשת נוירונים. נשים לב שהאלגוריתם צריך לקבל החלטות כל פעם בצורה שונה מכיוון שהמכשולים נקבעים בצורה רנדומלית ממקום למקום ולכן עליו לפצח את השיטה במדויק כיצד לעבור במכשול. הגעתי לתוצאה מרשימה במיוחד שבהיתן אוכלוסייה של 50 ציפורים לוקח לתוכנה פעמיים כדי להצליח לפענח את כללי המשחק ולפעול כך שהציפור לא תיפסל. לעיתים התוכנה הייתה גם מצליחה בפעם הראשונה. מצד אחד, המשחק Flappy bird אינו משחק מורכב אלגוריתמית אך מצד שני תוצאות ה machine learning אכן מרשימות ומראות על תוכנה שלומדת בעצמה ברמה גבוהה.

## רפלקציה

במסגרת ה-5 יחידות לימוד הנוספות של מדעי המחשב נדרשנו לבצע פרויקט גמר שמבוסס על אחת מארבעת התחומים הבאים: סייבר, למידת מכונה, אפליקציות או מערכות הפעלה. אני בחרתי את התחום למידת המכונה מכיוון שהתעניינתי בתחום הזה זמן רב. במהלך הפרויקט למדתי הרבה מאוד והעיקר בהם זה שפת פייתון ואת מבני האלגוריתמיקה של הבינה מלאכותית והשימוש בהם. את הפרויקט התחלתי במסגרת ידע של java. בחרתי לעשות את הפרויקט בשפת פייתון מכיוון שידעתי שזו שפה נוחה יותר ומתאימה יותר לסוג פרויקט כזה. את הבינה מלאכותית התחלתי ללמוד מאפס דרך מדריכים באינטרנט, סרטונים ביוטיוב והרבה ניסוי וטעייה של הקוד. חשוב לי להוקיר תודה לאתר <https://stackoverflow.com/> שנעזרתי בו כאשר לא הצלחתי להתקדם בקוד או כאשר לא הצלחתי להבין איפה הבעיה. למדתי במסגרת הפרויקט הרבה מאוד ידע על פייתון ו machine learning ולדעתי למידה שמבוססת על פרויקט היא הדרך היעילה ביותר ללמוד מדעי המחשב מכיוון שהיא משלבת בתוכה למידה בסיסית של החומר והתמודדות אישית עם תקלות ובעיות – שני הכלים העיקריים שנדרשים למתכנתים בימינו.

## תוכנת פייתון והספריות שהשתמשתי בפייתון

### Python

פייתון היא שפת תכנות עילית מהנפוצות בעולם. פייתון ידוע בעיקר בזכות הנוחות שלה והשימוש הידידותי שלה למשתמש. פייתון קלה ללמידה וקלה לתכנות ותומכת בתכנות מתקדם וב Object-oriented programming - OOP שנפוץ מאוד לשימוש בימינו. כיום לפייתון יש שימושים רבים והעיקריים שבהם הם: בינה מלאכותית, פיתוח אתרים ומשחקים, תוכנות מדיה ושמע ומדעי נתונים.

### Pygame

Pygame היא ספרייה בפייתון המשמשת לפיתוח משחקים הכוללים מתמטיקה, לוגיקה, פיזיקה, AI ועוד הרבה. בפייתון, תכנות המשחק נעשה בpygame וזה אחד המודולים הטובים ביותר לעשות זאת. כדי להוריד את pygame צריך לפתוח את חלון cmd ולהקליד את שורת הקוד `pip install pygame`, לאחר מכן יש ללחוץ אנטר והספרייה מוכנה לשימוש. בפרויקט הספרייה שימשה אותי רבות בעיצוב האובייקטים, התזוזה וההתנגשות בין אובייקטים.

### Random

random היא פונקציה מובנית של המודול האקראי ב-Python3. המודול האקראי מעניק גישה לפונקציות שימושיות שונות.

על ספריות של הבינה מלאכותית אסביר בפרק נפרד שעוסק בבינה מלאכותית.

## תוכנות שבהן נעזרתי

### Pycharm

Pycharm היא סביבת פיתוח משולבת לפיתוח תוכנות בעיקר בשפת פייתון, אשר פותחה על ידי תאגיד התוכנה הצ'כי JetBrains. היא כתובה ב-Java ובפייתון, והגרסה היציבה הראשונה שלה יצאה בפברואר 2010. הסביבה מספקת שירותים כגון ניתוח קוד, דיבוג קוד גרפי ותומכת בתכנות בסביבת אינטרנט באמצעות פלטפורמת הפיתוח Django וכן ב Data-Science באמצעות סביבת הפיתוח Anaconda.

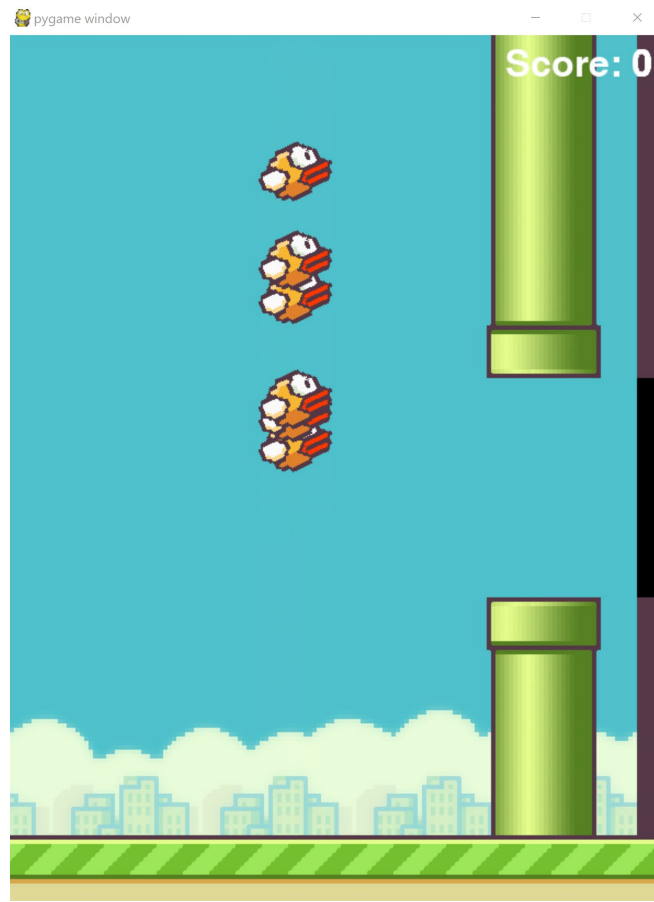
## **OOP–Object-oriented programming**

תכנות מונחה עצמים (OOP) הוא פרדיגמת תכנות בה משתמשים כדי להמיר את העולם האמיתי בקוד. ישנם מספר יתרונות לשימוש בפרדיגמה זו. באמצעות OOP, ניתן בקלות לבנות את העצמים ולבנות להם חוקים לוגים שמדמים את העולם האמיתי. את החלק של המשחק עצמו – רקע ועצמים, כלומר את הקלאסים ביצעתי בעזרת תכנות מונחה עצמים. שפות תכנות high level כגון python ו-C# לרוב בנויות בצורה המאפשרת חלוקה וארגון של מידע בצורה נוחה. OOP במיוחד חשוב כאשר מחלקים את הקוד לקבצים נפרדים על מנת שיהיה רצף לוגי של ה-data.

# המשחק



## המשחק – הצגת המשחק ותוצאות



המשחק מתחיל ב-50 ציפורים שמתחילות את המסלול. מכיוון שהמשחק מבוסס על בינה מלאכותית (ML), אין צורך בשחקן חיצוני שישחק את המשחק. הפרויקט מבוסס על Genetic Algorithm, לכן ככל שכמות האוכלוסייה ההתחלתית בכל דור גדולה יותר, כך המחשב ילמד יותר מהר לשחק.



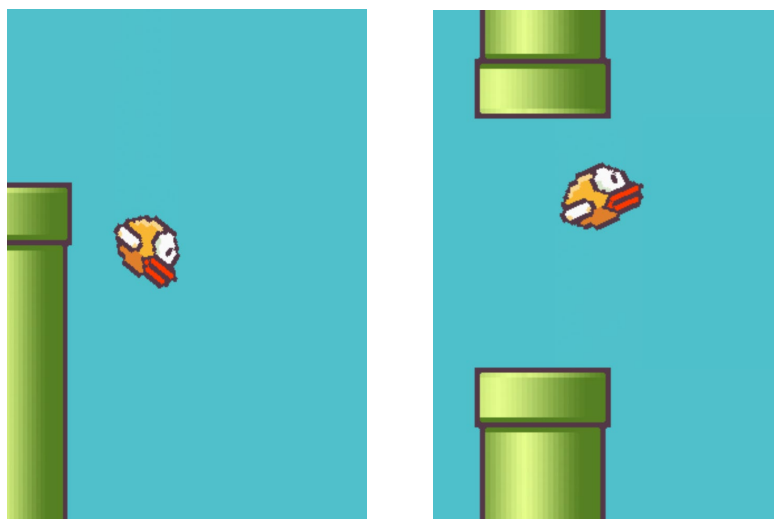
התמונה של הבסיס מוצגת באורך של התמונה המקורית המורדת, שחוזרת על עצמה לפי מיקום ציר X. כך זה מדמה רצפה אינסופית

הצינורות נמתחים לאורך בצורה רנדומלית במסגרת מסוימת. יש צינור למעלה וצינור למטה. הצינורות מהווים מכשול פיזי שפגיעה בו "הורגת את הציפור", כלומר מעלימה אותה מהמסך.

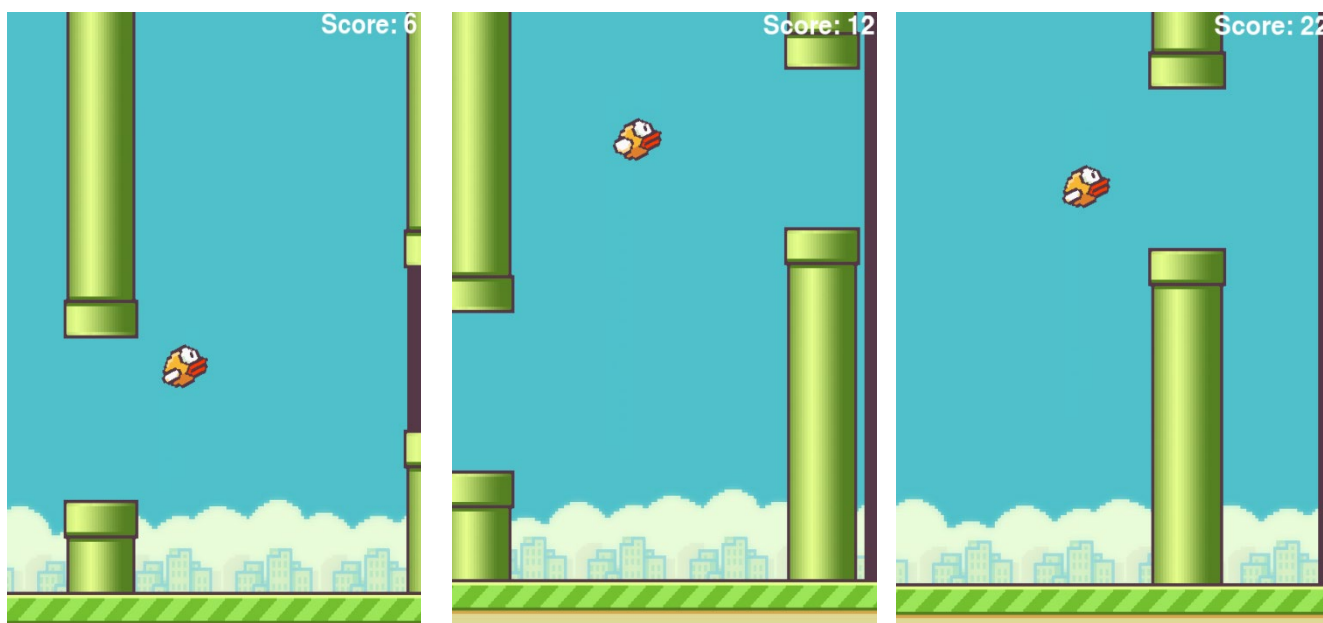




התמונות שימשו לאנימציה התעופה של הציפורים. ניתן לראות שכל ציפור שונה בכנפיים. כל תמונה מיועדת לשהייה במקום, קפיצה למעלה שידמה תעופה למעלה עם הכנפיים או ירידה למטה שיראה שהציפור יורדת למטה. חשוב לציין שבקוד גם הוספתי זווית לציפור כך שיראה ממושכת למטה יראה שהציפור יורדת 90 מעלות מטה ומעלה יש זווית עלייה למעלה.



הציפורים יכולות לזוז למעלה (קפיצה) ולמטה בצורה מדמת מציאות. כלומר כך שיופעל עליה כוח קבוע בציר Y שידמה גרביטציה.



פונקציית עולם מתמשך כך שהמיקום על המסך של הדמות נשאר קבוע והרקע הוא שזז בכיוון ציר X.

```

Population's average fitness: 5.17000 stdev: 4.50401
Best fitness: 13.70000 - size: (1, 3) - species 1 - id 20
Average adjusted fitness: 0.252
Mean genetic distance 1.258, standard deviation 0.444
Population of 20 members in 1 species:
  ID  age  size  fitness  adj fit  stag
  ----  ---  ----  -
  1    0   20   13.7    0.252    0
Total extinctions: 0
Generation time: 6.935 sec

***** Running generation 1 *****

```

```

Population's average fitness: 7.25000 stdev: 5.07272
Best fitness: 13.10000 - size: (1, 2) - species 1 - id 29
Average adjusted fitness: 0.458
Mean genetic distance 1.153, standard deviation 0.367
Population of 20 members in 1 species:
  ID  age  size  fitness  adj fit  stag
  ----  ---  ----  -
  1    1   20   13.1    0.458    1
Total extinctions: 0
Generation time: 4.792 sec (5.863 average)

***** Running generation 2 *****

```

אחרי כל הרצת הציפורים מופיעים נתונים סטטיסטיים כמו זמן ההישרדות וההתקדמות שלהן.

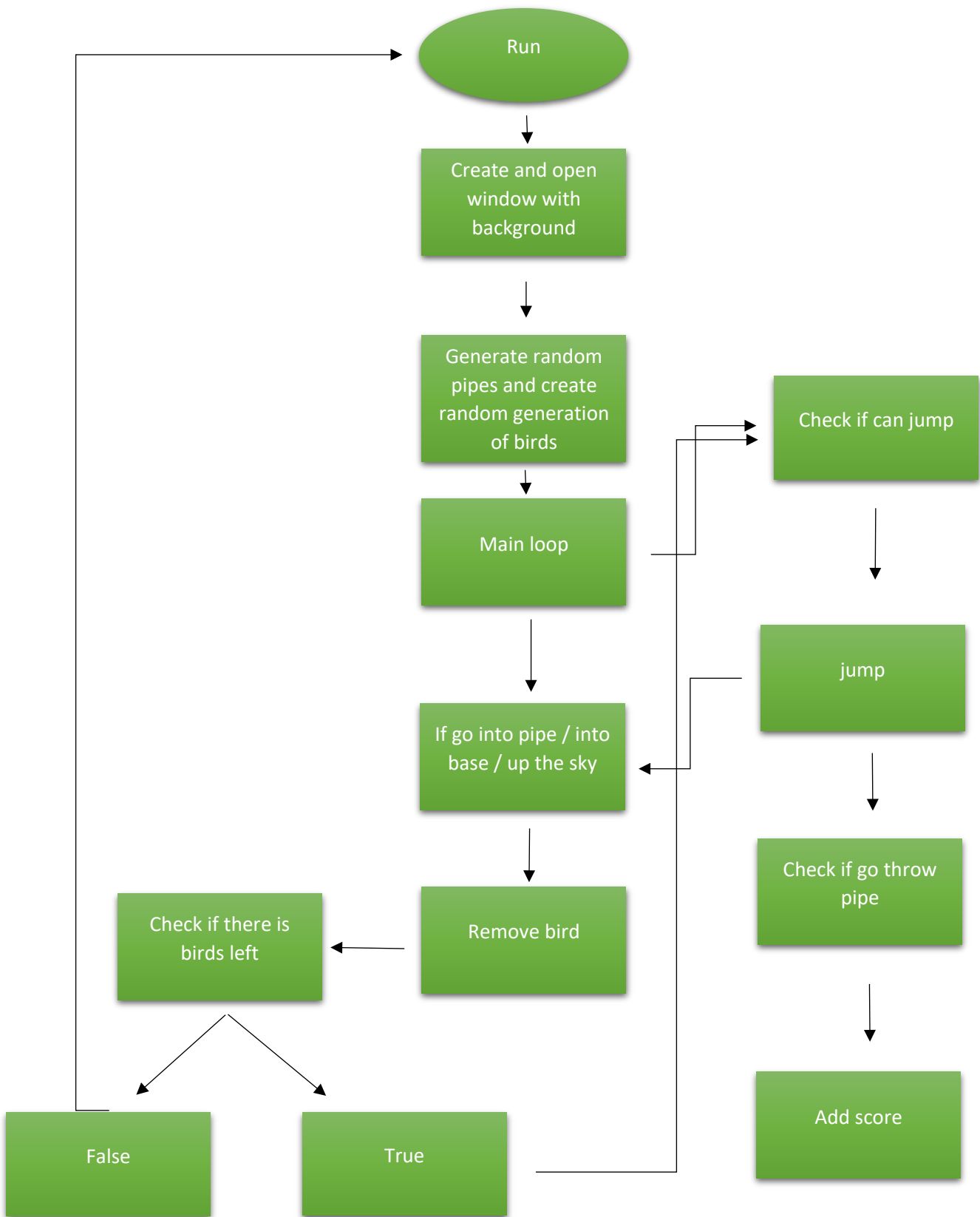
Score: 16

מטרת המשחק היא להגיע לכמה שיותר נקודות.

Gen 2

בכל דור מופיע הדור שכרגע רץ במשחק – הוספתי את זה בסוף לכן בחלק מהתמונות אפשר לראות ובחלקן עוד לא היה את זה.

## תרשים UML של המשחק



## הקלאסים

### **Bird**

בקלאס יש את התכונות של הציפור, כל התמונות שלו לכל הכיוונים. יש פה את הפונקציות `get`, `move` ו `jump`, פונקציית ה `drawn` שמציירת את הציפור בכיוונים השונים ופונקציית `mask`, שמיועדת להתנגשות ויש עליה בפירוט בחלק התיאורטי של הפרויקט.

### **Pipe**

בקלאס יש את התכונות של המכשול. פונקציית `set_height` שקובעת גובה רנדומלי שבו יהיה המכשול. פונקציות `draw` ו `move` נמצאים בקלאס זה. בנוסף יש את הפונקציה `collide` שמשתמשת בפונקציה במחלקה של `Bird – get mask`.

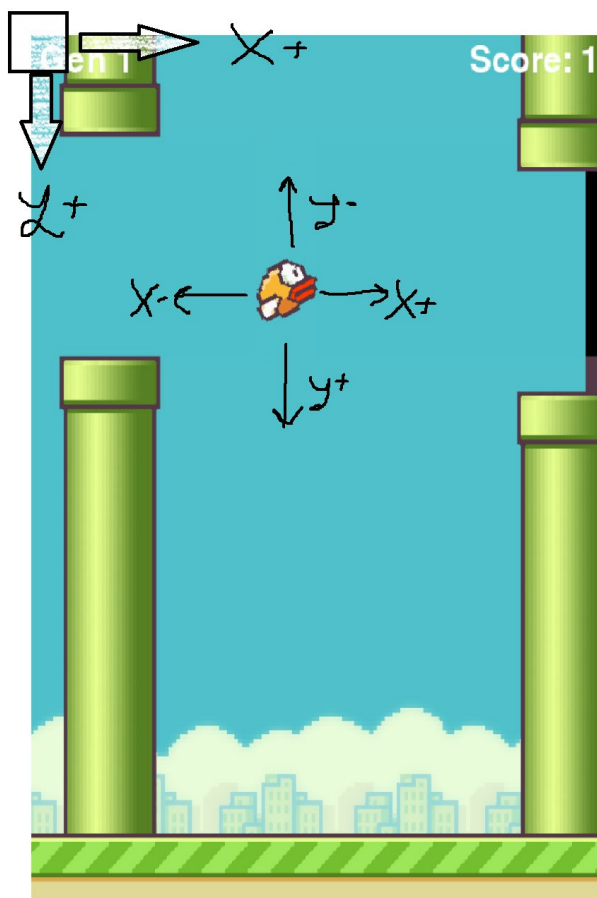
### **Base**

בקלאס יש את התכונות של הבסיס. במחלקה יש גם את הפונקציות `move` שמטרתה להזיז את הבסיס שיראה בתזוזה ותחדש את הבסיס שיראה אינסופי ויש את הפונקציה `draw` שמציירת אותה.

## הפונקציות

הקפיצה של הציפור – jump:

```
def jump(self):  
    self.vel = -10.5 # it is minus because of the xy chart of the program  
    self.tick_count = 0 # keep track of when we last jump  
    self.height = self.y
```



כדי לקפוץ צריך להשתמש במספר שלילי מכיוון שכך זה מוגדר בחלון המשחק בפיתוח. בחרתי במספר 10.5 מכיוון שזה המספר שעבד לי הכי טוב עם שאר המספרים שבחרתי.

## התזוזה של הציפור:

```
def move(self):
    self.tick_count += 1 # keep track of how much we moved

    # how many pixels we moving up or down each frame
    # d = displacement
    d = self.vel * self.tick_count + 1.5 * self.tick_count ** 2

    if d >= 16:
        d = 16


    if d < 0:
        d -= 2

    self.y = self.y + d

    if d < 0 or self.y < self.height + 50:
        if self.tilt < self.MAX_ROTATION: # when we go up we don't want go 90
degree up
            self.tilt = self.MAX_ROTATION
    else:
        if self.tilt > -90: # when we go down we want to go 90 degree down
            self.tilt -= self.ROT_VEL
```

**הסבר לנוסחה**  $d = self.vel * self.tick\_count + 1.5 * self.tick\_count ** 2$

d הוא משתנה העתקה שלנו והוא מציין כמה למעשה נזוז למעלה או למטה. בחרתי בפונקציה של d כמתואר מכיוון שבעזרתה ניתן לדמה את תנועת הציפור לצורה פרבולית. הפונקציה היא פונקציה פיזיקלית – נוסחת מהירות-תנועה פרבולית שמדמה פרבולה שבעזרתה תנועת הציפור נראית כמושפעת מגרביטציה. הנוסחה למעשה **מייעלת** את הקוד מכיוון שאין צורך בהוספת פונקציית גרביטציה. Vel זה בעצם velocity שמתאר את מהירות הציפור. Tick\_count זה למעשה הזמן. כך, כאשר למשל  $Tick\_count = 1$  אנחנו נקבל:

$$d = self.vel * self.tick\_count + 1.5 * self.tick\_count ** 2$$


והתוצאה שתתקבל תהיה -9.

לאחר מכן אני רוצה להגביל העתקה של הציפור כדי שלא תנועה יותר מידי מהר למעלה או למטה. בנוסף אני לא רוצה שהציפור תנועה למעלה בזווית של 90 מעלות אבל כן רוצה שהיא תוכל לרדת למטה ב90 מעלות.

## תזוזת כנפי הציפור:

```
def draw(self, win):
    self.img_count = self.img_count + 1

    if self.img_count < self.ANIMATION_TIME:
        self.img = self.IMGS[0]
    elif self.img_count < self.ANIMATION_TIME * 2:
        self.img = self.IMGS[1]
    elif self.img_count < self.ANIMATION_TIME * 3:
        self.img = self.IMGS[2]
    elif self.img_count < self.ANIMATION_TIME * 4:
        self.img = self.IMGS[1]
    elif self.img_count == self.ANIMATION_TIME * 4 + 1:
        self.img = self.IMGS[0]
        self.img_count = 0

    if self.tilt <= -80:
        self.img = self.IMGS[1]
        self.img_count = self.ANIMATION_TIME * 2

    rotated_image = pygame.transform.rotate(self.img, self.tilt)
    new_rect = rotated_image.get_rect(center=self.img.get_rect(topleft=(self.x,
self.y)).center)
    win.blit(rotated_image, new_rect.topleft)
```

מדובר בפונקציה draw, שמציירת את הציפור ולמעשה בדרך כלל פונקציות כאלה לא מסובכות ואין צורך בהסבר מפורט, אך במשחק שעשיתי אכן יש מספר דברים מעניינים שחשוב לשים אליהם לב.

אנחנו רוצים שכנפי הציפור יראו בתזוזה, כאילו שהציפור מנפנפת בכנפיים שלה כל הזמן. לכן אני משתמש בפרמטר img\_count כפרמטר ספירה משתנה ובפרמטר animation\_time שהוא קבוע ושווה ל-5. כך, בכל כמה רגעים אנו עוברים מתמונה אחת לתמונה אחרת ואז מתאפסים והפונקציה רצה כל עוד המשחק פועל וזה מדמה שכנפי הציפור מתנפנפים כל הזמן.

נשים לב, כאשר הציפור יורדת בקו ישר, שהגדרנו שהיא יכולה בפונקציה move, אנחנו לא רוצים שהיא תנופף בכנפיים ולכן הוספתי גם את זה לפונקציה.

לבסוף, אנחנו רוצים שהציפור תזוז בזווית מסוימת. החלק האחרון של הקוד היה מסובך לעשייה ולמעשה הוא לקוח מתוך **Stack Overflow** והוא עבד טוב כך שהציפור תזוז בזווית.



## התנגשות ציפור ואובייקט: masks

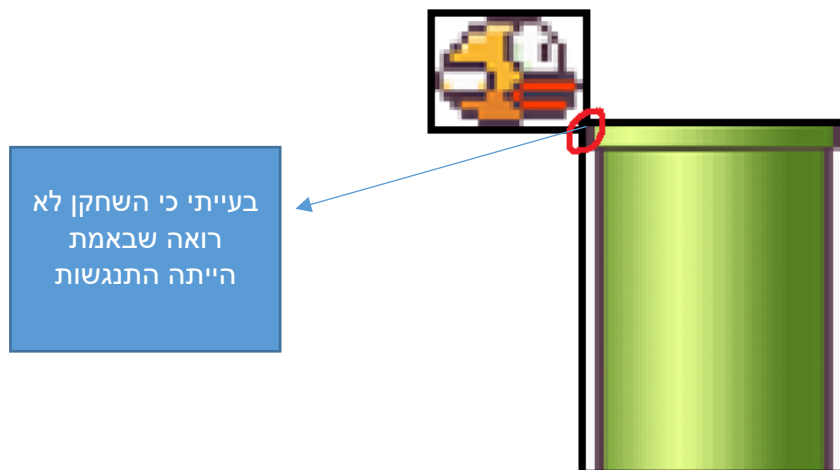
```
def collide(self, bird):
    bird_mask = bird.get_mask()
    top_mask = pygame.mask.from_surface(self.PIPE_TOP)
    bottom_mask = pygame.mask.from_surface(self.PIPE_BOTTOM)

    # check points of collision
    top_offset = (self.x - bird.x, self.top - round(bird.y))
    bottom_offset = (self.x - bird.x, self.bottom - round(bird.y))

    # return us true or false about if birds is touching pipe
    b_point = bird_mask.overlap(bottom_mask, bottom_offset)
    t_point = bird_mask.overlap(top_mask, top_offset)

    if t_point or b_point:
        return True
    return False
```

כדי לבצע התנגשות בין שני אובייקטים השתמשתי בפרויקט הזה בפונקציית mask שכתובה כבר בpygame. כאשר רוצים לבצע התנגשות בין אובייקטים בפייתון, בדרך כלל כל אובייקט מוקף בסוג של מסגרת דמיונית שנקבעת לפי האורך והרוחב של האובייקט והתנגשות נוצרת כאשר יש התנגשות בין המסגרות:



ניתן לראות כאן את הבעייתיות, מכיוון שבמצב כזה הציפור מתנגשת באובייקט גם כאשר היא לא באמת נוגעת בו. לכן, כדי להתמודד עם בעיה זו נעזרתי בפונקציית mask שהיא כבר מובנת בpygame. פונקציה זו למעשה יוצרת רשימה דו ממידית (מטריצה) שמשווה בין מקומי הפיקסלים של האובייקטים במסגרות שלהם וכך מחשבת אם ההתנגשות קרתה במיקומי הפיקסלים. כך ההתנגשות היא אמיתית.

להסבר נוסף על masks ניתן למצוא ב:

<https://stackoverflow.com/questions/67846651/pygame-masks-python>

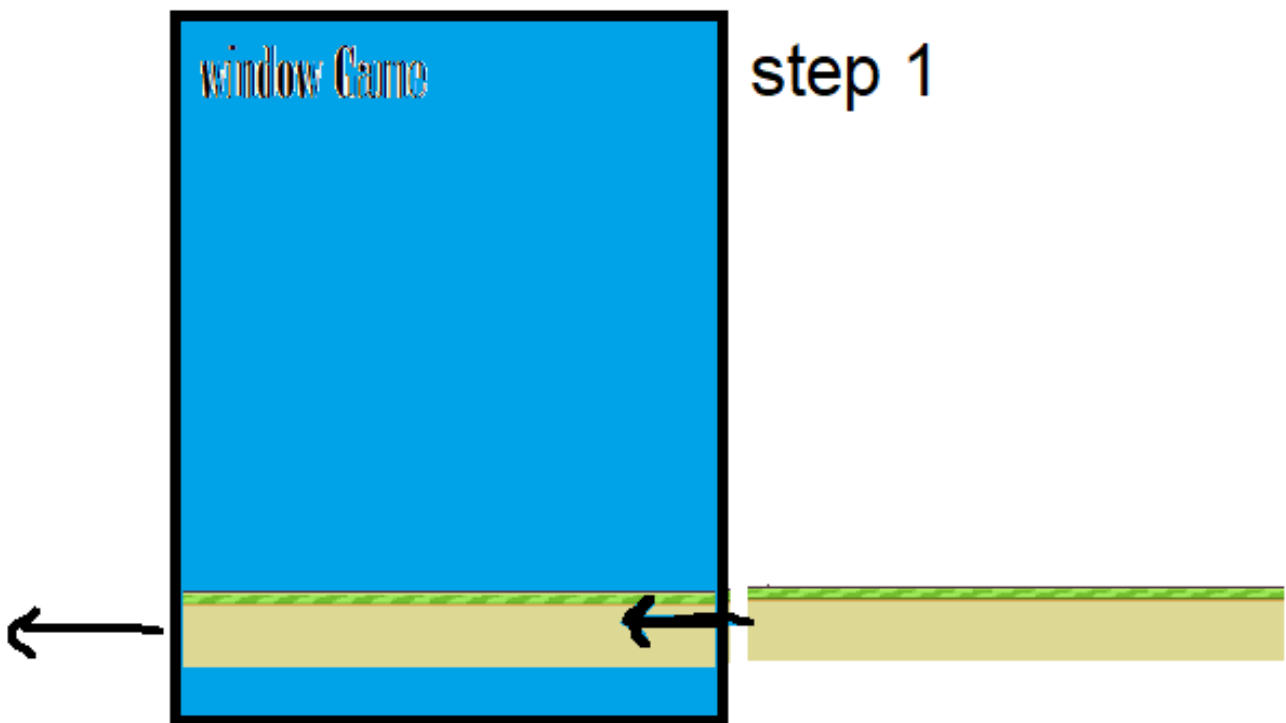
```
def move(self):
    self.x1 -= self.VELOCITY
    self.x2 -= self.VELOCITY

    if self.x1 + self.WIDTH < 0:
        self.x1 = self.x2 + self.WIDTH

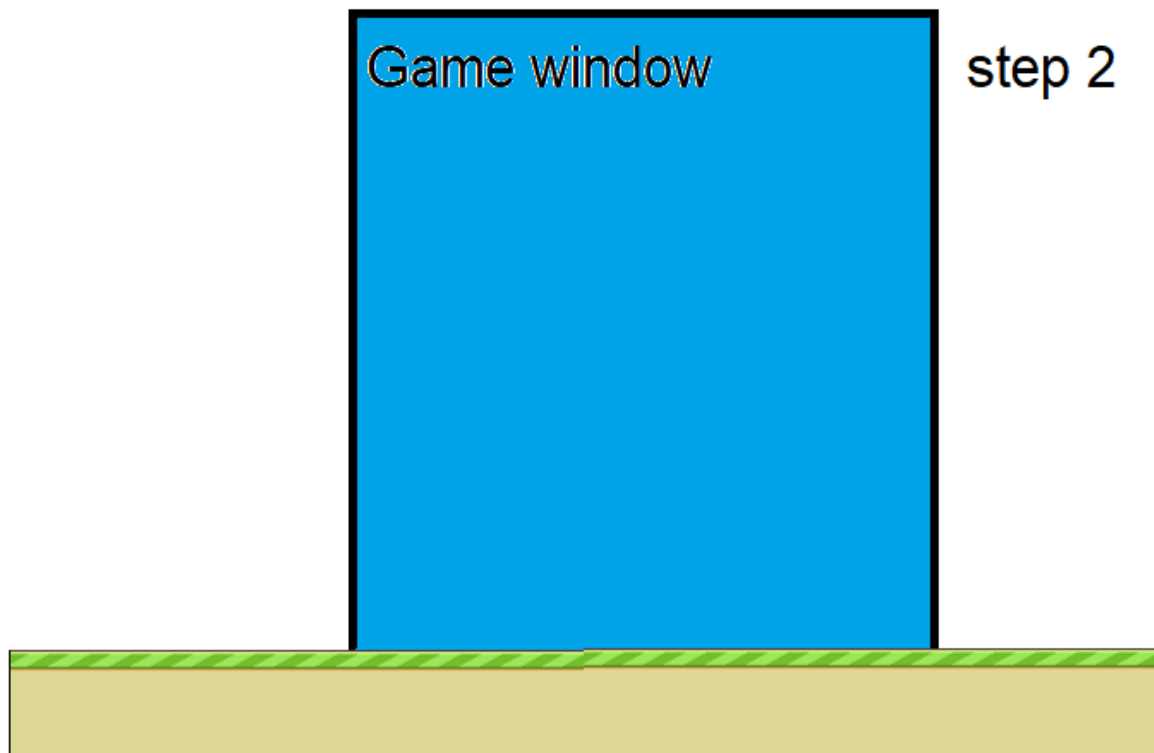
    if self.x2 + self.WIDTH < 0:
        self.x2 = self.x1 + self.WIDTH
```

במשחק הציפור נשארת במקום קבוע על המסך והרקע הוא זה שזז. בשביל ליישם זאת, בנית פונקציה שמחזירה את הבסיס שוב ושוב לפי האורך שלו שיראה כאינסופי וכזז.

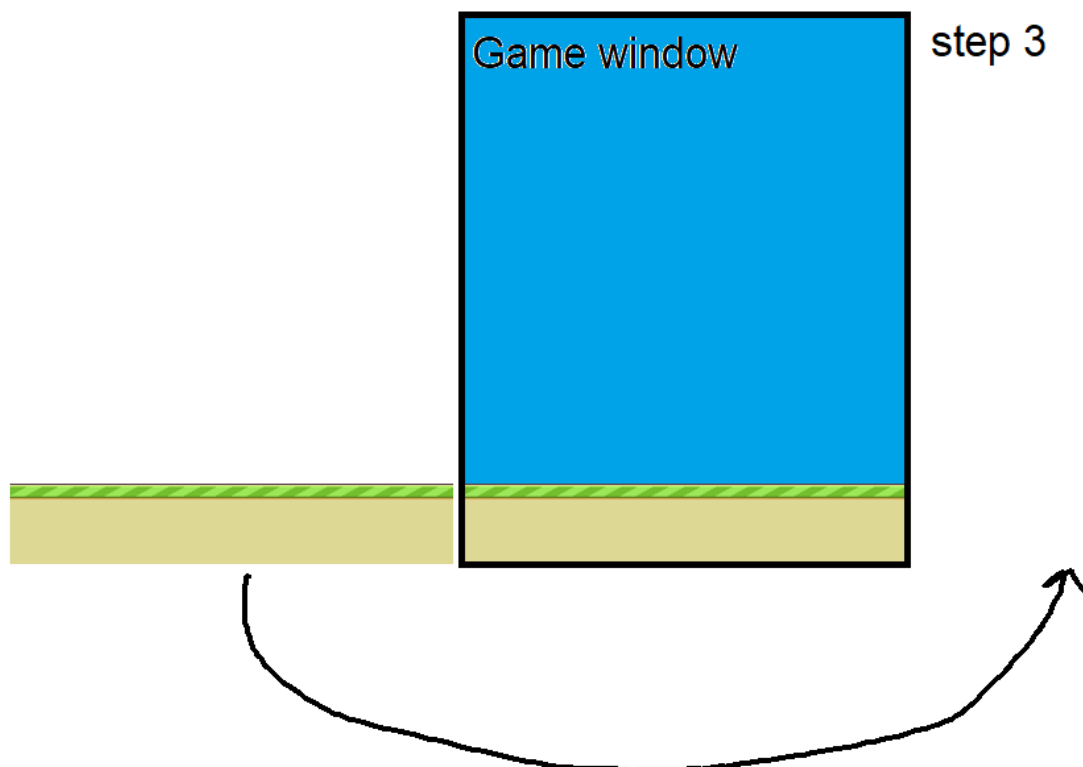
תחילה המיקומים של כל בסיס ממוקמים כך שהלוח הראשון נמצא במסך המשחק והלוח השני מימינו מחוץ למשחק. שני הלוחות נעים שמאלה:



בשלב הבא שני הבסיסים ינועו שמאלה ויהיו כך:



בשלב האחרון בקוד ברגע שהלוח הראשון יוצא מחוץ לשטח החלון של המשחק הוא חוזר אחורה וזה נראה כך:

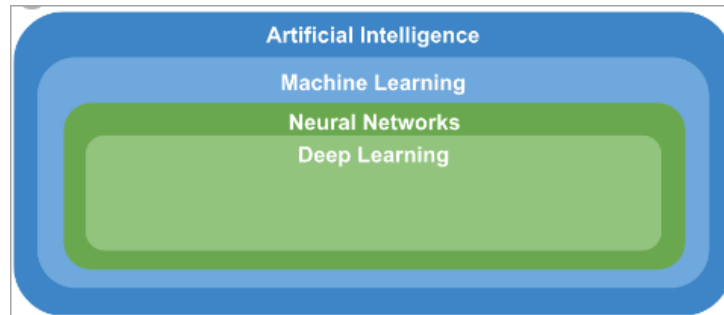


הבסיס עובר להתחלה ושוב חוזרים לשלב הראשון. ככה נוצר לוח אינסופי במשחק.

# בינה מלאכותית

## בינה מלאכותית – הקדמה

בצורתו הפשוטה ביותר, בינה מלאכותית היא תחום המשלב מדעי מחשב ומערכי נתונים חזקים, כדי לאפשר פתרון בעיות. זה כולל גם תחומי משנה של למידת מכונה ולמידה עמוקה, המוזכרים לעיתים קרובות בשילוב עם בינה מלאכותית.



## למידת מכונה – Machine learning

למידת מכונה הוא תחום במדעי המחשב שלומד מניסיון מבלי להיות מתוכנת. למידת מכונה הוא תת נושא של בינה מלאכותית. המדע שעומד מאחורי ML הוא לגרום למחשבים לבצע פעולות לבד. אלגוריתם Machine Learning הוא **תוכנית גנרית** שתבין את הנתונים, ותבנה מודלים עם הנתונים האלה. מודלים אלה זמינים עבור המשתמשים לביצוע משימות.

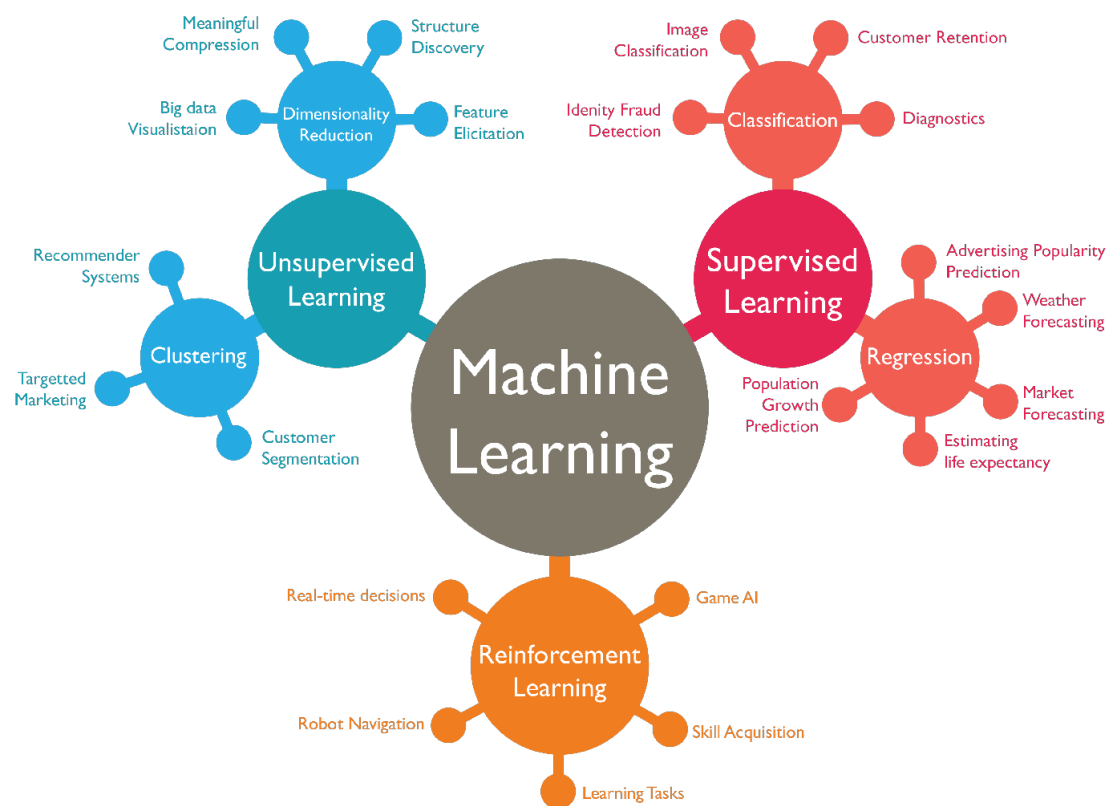
תכנות גנרי (הגדרה) – תכנות גנרי הוא דרך לכתיבת תוכניות שאינן תלויות בטיפוסי המשתנים

למידת מכונה היא קבוצה של אלגוריתמים המבצעים משימה מסוימת עם נתוני הקלט ומשפרים גם את ביצועיהם. אלגוריתמים אלה תואמים את הקלט לפלט, וכך מגיעים לניבוי דפוסים. ככל שמזינים יותר נתונים לאלגוריתמים, כך התחזיות מדויקות יותר.

## סוגים של ML

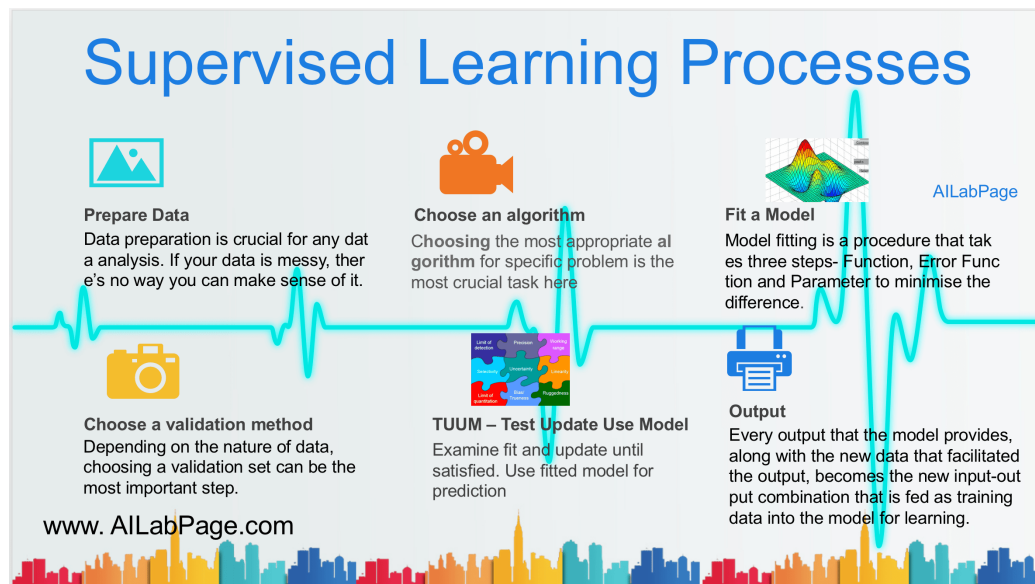
יש שלושה סוגים של machine learning:

- Supervised
- Unsupervised
- Reinforcement learning



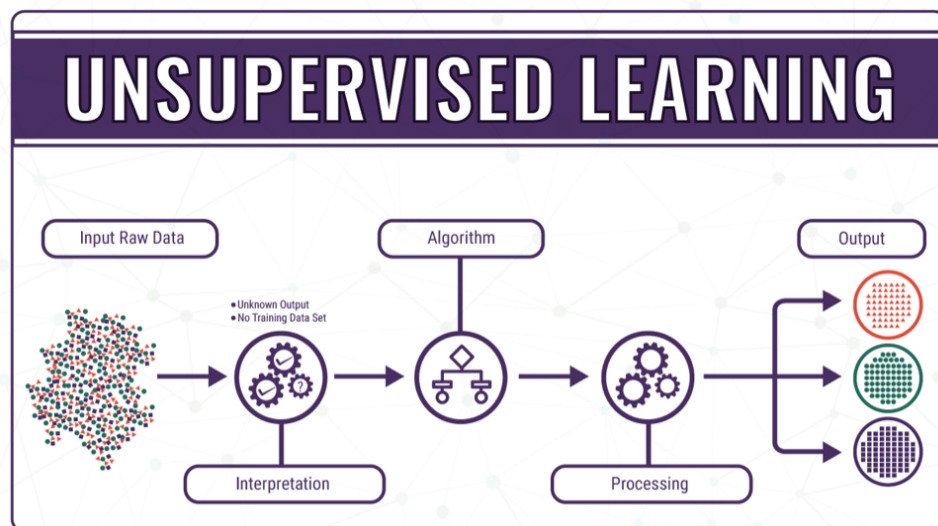
## Supervised ML: בפיקוח

ML בפיקוח דומה לדרך שבה מלמדים ילד קטן. בסוג למידה זו, התוכנה נבדקת על ידי המתכנת והפלטרים שהוא מקבל ולומדת לפיהם. באלגוריתם ה- ML בפיקוח, הפלט כבר ידוע. יש מיפוי של קלט עם הפלט. לפיכך, כדי ליצור מודל, המכונה מוזנת עם הרבה נתוני קלט שמשמשים לאימון (עם קלט ופלט ידועים). נתוני האימון מסייעים להשגת רמת דיוק למודל הנתונים שנוצר. המודל הבנוי מוכן כעת להיות מוזן בנתוני קלט חדשים ולחזות את התוצאות. למידה מפוקחת היא מנגנון למידה מהיר עם דיוק גבוה. בעיות הלמידה המפוקחות כוללות בעיות רגרסיה וסיווג.



## Unsupervised ML: למידה לא בפיקוח

למידה ללא פיקוח מתרחשת ללא עזרת מפקח - זהו תהליך למידה עצמאי. במודל זה, מכיוון שאין פלט הממופה עם הקלט, ערכי היעד אינם ידועים. המערכת צריכה ללמוד מעצמה מקלט הנתונים ולזהות את הדפוסים הנסתרים. מכיוון שאין ערכי פלט ידועים שניתן להשתמש בהם לבניית מודל לוגי בין הקלט לפלט, משתמשים בטכניקות מסוימות בכריית כללי נתונים, דפוסים וקבוצות נתונים עם סוגים דומים. קבוצות אלה עוזרות למשתמשים להבין את הנתונים בצורה טובה יותר וכך למצוא את הפלט הרצוי. כאשר נתונים חדשים מוזנים, המודל ימין אותם לקבוצות והדפוסים של הנתונים הקודמים שהוא יצר. אם הנתונים החדשים לא מתאימים לאף מחלקה של דפוס או קבוצה, הוא ייצור מחלקה חדשה עם הנתונים האלה.





## Reinforcement learning: לימוד בעזרת חיזוקים

במהלך השנים אנחנו לומדים על ידי הפעולות שלנו מה יותר טוב לנו לעשות וכיצד לפעול. אנחנו יודעים את התוצאות שצפויות להגיע מההחלטות שלנו ומכל החלטה חדשה אנחנו מקבלים את התבונות החדשות שלנו. כך גם עובד מנגנון ML הזה. בסוג למידה זה, האלגוריתם לומד על ידי מנגנון משוב. כך שבכל פעם שצריך לבצע את הצעד הבא, הוא מקבל את המשוב מהשלב הקודם, יחד עם הלמידה מהניסיון לחזות מה יכול להיות הצעד הבא הטוב ביותר. למידת חיזוק היא תהליך איטרטיבי לטווח ארוך. ככל שמספר המשובים גדול יותר, כך המערכת הופכת למדויקת יותר. בפרויקט שלי השתמשתי בסוג הזה של ML שמשמש בGenetic Algorithm ששייך לneural network. דרך הפעולה:

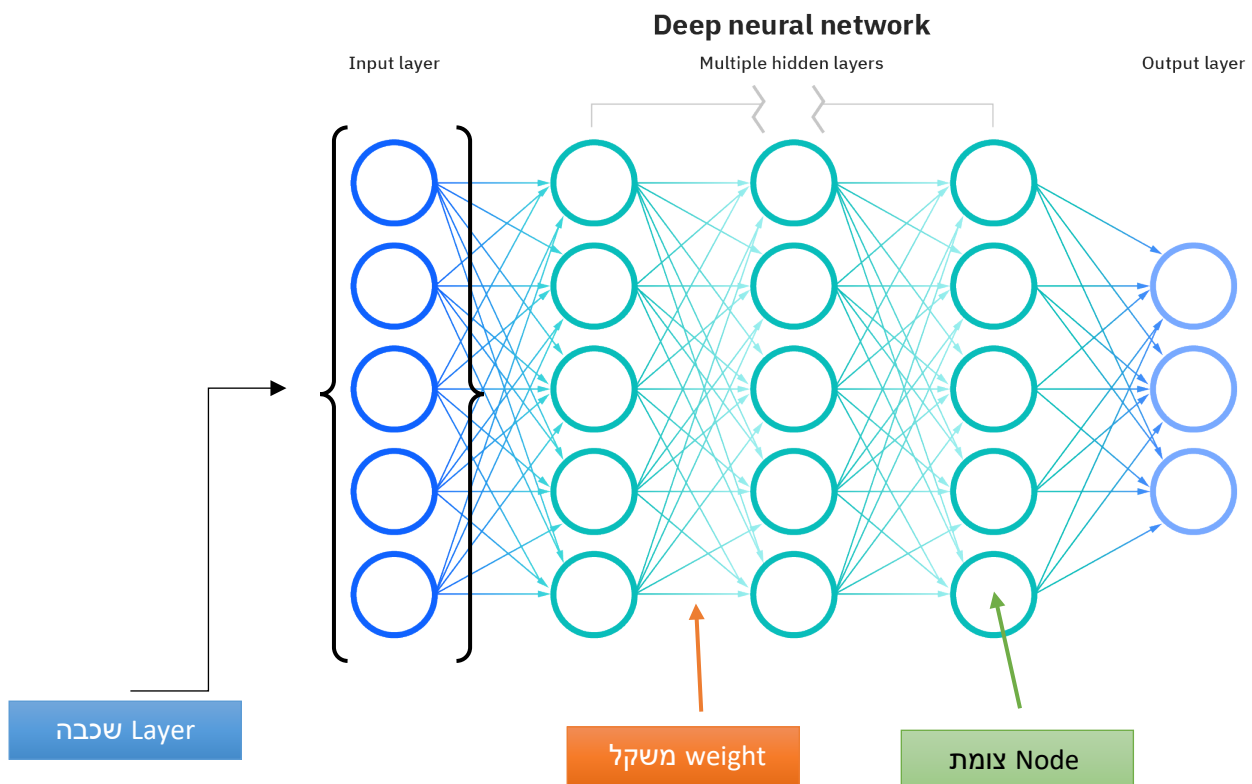
1. הקלט נקלט על ידי המערכת – במקרה שלנו זו הציפור.
2. המערכת (הציפור) עושה פעולה – לקפוץ או לא
3. התגובה לפעולת הציפור נשלחת אליה כמשוב חיובי או שלילי



## Neural Network

Neural network, הן תת-קבוצה של למידת מכונה והן מהוות לב לאלגוריתמי deep learning. שמם ומבניהם נובעים בהשראת המוח האנושי, ומחקים את האופן בו ניירונים ביולוגיים מאותתים זה לזה.

Artificial neural networks מורכבות משכבות המכילות צמתים - שכבת קלט, שכבה אחת או יותר נסתרות ושכבת פלט. כל צומת, או ניירון מלאכותי, מתחבר לאחר ובעל משקל (weight) וסף קשורים. אם הפלט של צומת בודד נמצא מעל ערך הסף שצוין, צומת זה מופעל ושולח נתונים לשכבה הבאה ברשת. אחרת, לא מועברים נתונים לשכבה הבאה ברשת.

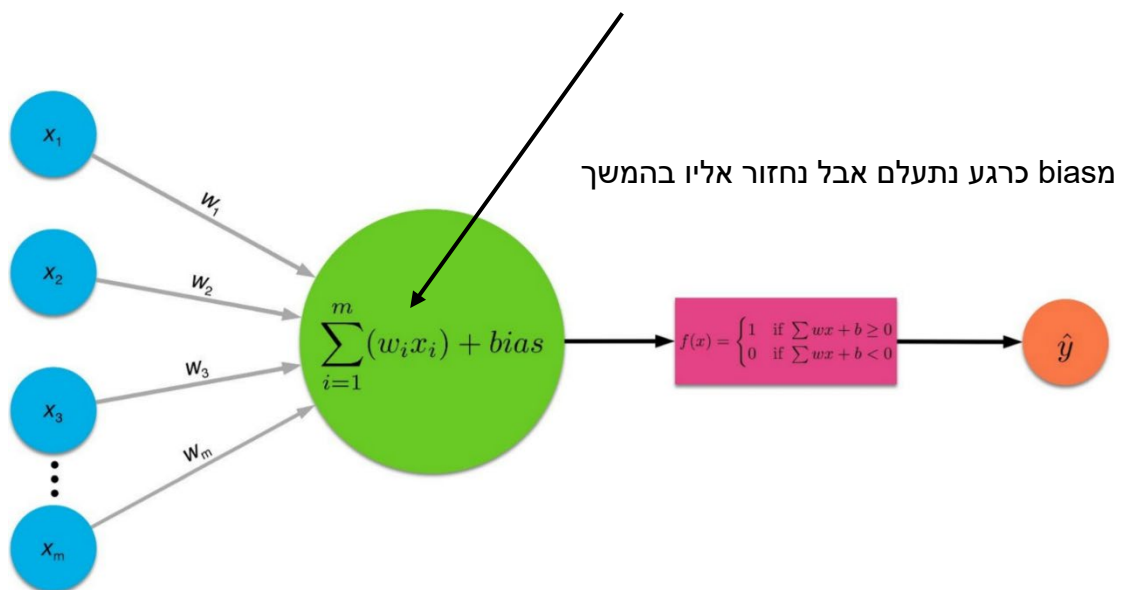


## הסבר על דרך פעולת רשת הנוירונים והגעה לפלט הסופי

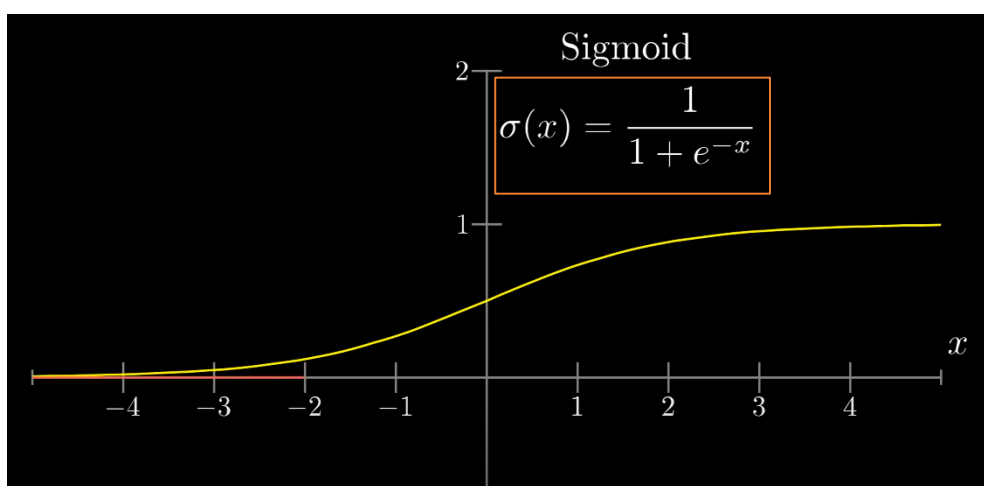
בכל צומת יש מספר, ולכל צומת מחובר משקל שהוא גם מספר. כדי להגיע לצומת הבאה, עלינו לכפול את כל המספרים שבצמתים לכל המשקלים בהתאמה.

כלומר:

$$w_1 a_1 + w_2 a_2 + w_3 a_3 + \dots + w_n a_n$$



כאשר אנחנו מדברים על הפלט, ברצוננו שהוא יהיה פשוט ככל הניתן. לשם כך נרצה לקבל פלט עם ערך מספרי פשוט שאומר דבר פשוט. לדוגמה: פלט שמטרתו להגיד לנו לעשות פעולה או לא לעשות, כאשר אנחנו מתקרבים ל1 וקרובים מספיק הוא יופעל ונעשה את הפעולה ואחרת הוא לא יפעל ולא תהיה פעולה. אך, כאשר אנחנו עושים את החישוב הנ"ל לא מובטח לנו שנגיע לערך בין 0 ל1. לשם כך אנחנו נעזרים בפונקציות שונות שמקבלות כקלט את התוצאה של החישוב שלנו ומחזירות ערך בין 0 ל1. נהוג לקרוא להן Activation Function דוגמה לפונקציה כזאת הינה:



לכן, נוסיף אותו לנוסחה שלנו ונקבל :

$$\text{Sigmoid} \\ \downarrow \\ \sigma(w_1 a_1 + w_2 a_2 + w_3 a_3 + \dots + w_n a_n)$$

כעת, כאשר אנחנו רוצים להעביר את המידע מניריון לניריון (מצומת לצומת), אנחנו נפעיל את החישוב ונקבל ערך מסוים, לפי הערך הזה נדע האם למסור אותו לניריון הבא או להפטר ממנו. אך מה קורה אם אנחנו לא רוצים שהניריון הבא יופעל כאשר הסכום בסוגריים גדול מ0. נגיד שאנחנו רוצים שהוא יופעל רק כאשר הסכום בסוגריים גדול מ10. למטרת יעילות נרצה להפטר מכל הנירונים שסכומם יוצא קטן מ10. לשם כך אנחנו קובעים bias. bias נועד לסלק את כל הסכומי ניירונים הלא רלוונטיים, כלומר את כל אלה הקטנים מ10. לכן במקרה שלנו נקבע את ה bias להיות 10- ונחבר אותו לסכום הניירונים (מה שבתוך הסוגריים). נקבל:

$$\sigma(w_1 a_1 + w_2 a_2 + w_3 a_3 + \dots + w_n a_n - 10)$$

“bias”

למעשה, bias מורה לנו כמה הסכום צריך להיות גבוה, לפני שנגדיר את הניריון לפעול. לבסוף, כדי לחשב את הניירונים מהשכבה הראשונה לשכבה הבאה שלה נצרך לעשות:

$$\text{Sigmoid} \\ a_0^{(1)} = \sigma(w_{0,0} a_0^{(0)} + w_{0,1} a_1^{(0)} + \dots + w_{0,n} a_n^{(0)} + b_0)$$

Bias

מספר ניירון

מספר שכבה

יוצא, שכדי לחשב רק את כל הניירונים של השכבה הראשונה צריך להפעיל חישוב זה מ פעמים, ואחרי זה נצטרך לעשות זה לכל שכבה בנפרד!

לכן, על מנת להקל על החישוב, נחשב את הפלטים של כל שכבה ביחד ונבחר מה מהפלטים האלה יהוו קלטים לשכבה הבאה. כדי לחשב את הפלטים האלה נעזר בכפל מטריצות:

The diagram illustrates the calculation of the next layer's input vector  $a^{(0)}$  by multiplying the weight matrix  $W$  by the current layer's output vector  $a^{(0)}$ . The weight matrix  $W$  is a matrix of size  $(k+1) \times (n+1)$  with elements  $w_{i,j}$ . The input vector  $a^{(0)}$  is a column vector of size  $(n+1) \times 1$  with elements  $a_j^{(0)}$ . The resulting output vector is a column vector of size  $(k+1) \times 1$  with elements  $a_i^{(0)}$ . The diagram uses blue boxes to highlight the weight matrix, the input vector, and the output vector. Arrows point from the labels to the corresponding parts of the equation.

משקלים לנוירון הראשון

משקלים לנוירון השני

משקלים לנוירון האחרון בשכבה הראשונה

ערכי הנוירונים

כעת, לאחר שהבנו את המטריצה נבסס את החישוב הסופי לשכבה, נוסיף את bias ונכפול בactivation function:

The diagram shows the final calculation of the output vector  $a^{(0)}$  by adding the bias vector  $b$  to the weighted input vector and then applying the activation function  $\sigma$ . The weight matrix  $W$  is a matrix of size  $(k+1) \times (n+1)$  with elements  $w_{i,j}$ . The input vector  $a^{(0)}$  is a column vector of size  $(n+1) \times 1$  with elements  $a_j^{(0)}$ . The bias vector  $b$  is a column vector of size  $(k+1) \times 1$  with elements  $b_i$ . The resulting output vector  $a^{(0)}$  is a column vector of size  $(k+1) \times 1$  with elements  $a_i^{(0)}$ . The diagram uses yellow boxes to highlight the weight matrix, the input vector, the bias vector, and the output vector. Arrows point from the labels to the corresponding parts of the equation.

$\sigma$

$w_{0,0} \ w_{0,1} \ \dots \ w_{0,n}$

$w_{1,0} \ w_{1,1} \ \dots \ w_{1,n}$

$\vdots \ \vdots \ \ddots \ \vdots$

$w_{k,0} \ w_{k,1} \ \dots \ w_{k,n}$

$a_0^{(0)}$

$a_1^{(0)}$

$\vdots$

$a_n^{(0)}$

$b_0$

$b_1$

$\vdots$

$b_n$

באמצעות חישוב של כפל מטריצה זה, המחשב יכול לפעול במהירות רבה יותר וקל יותר לתכנת את החישוב הזה במחשב.

## האימון של הרשת

לאחר בניית המודל של neural network, נרצה לאמן את הרשת. נוכל לעשות זאת על ידי supervised learning. נביא למכונה קלטים עם פלטים ידועים. כאשר אני אומר לאמן את המכונה, אני מתכוון לכך שנמצא את המשקלים הנכונים. בהרצה הראשונה נבחר משקלים רנדומליים, לכן נוכל לשער שהרצה הראשונה המכונה לא תפעל טוב. כעת אנחנו נרצה לבדוק את הדיוק של ההרצה כדי שנדע איזה משקלים צריך לשנות. לבדיקת הדיוק אנחנו משתמשים ב-cost function:

$$\text{Cost Function} = \text{MSE} = \frac{1}{2m} \sum_{i=1}^m (\hat{y} - y)^2$$

כאשר:

m – מספר הדגימות

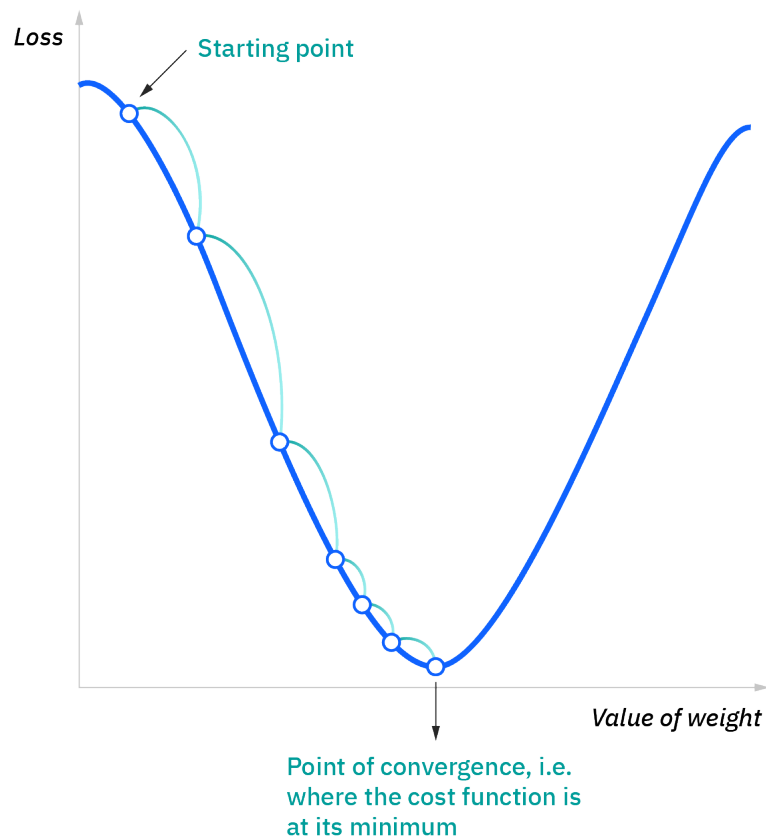
i – מספר דגימה

y – הפלט הרצוי

y קובע – הפלט שהתקבל

בסופו של דבר, המטרה היא למזער את **cost function** שלנו כדי להבטיח נכונות התאמה לכל תצפית נתונה. כאשר המודל מתאים את משקולותיו והטיותו, הוא משתמש ב**cost function** ו **reinforcement learning** כדי להגיע למינימום המקומי. התהליך בו אלגוריתם מכוון את משקליו הוא דרך ירידת שיפוע, המאפשר למודל לקבוע את הכיוון שאליו יש לקחת כדי להפחית שגיאות (או למזער את **cost function**). בכל דוגמה לאימון, הפרמטרים של המודל מתאימים להתכנס בהדרגה למינימום.

**Cost function** – פונקציה שמוגדרת כסכום של ריבוע ההפרש בין הערכים של הנירונים המצויים לרצויים בהתאמה. ככל הערך הפונקציה גדול יותר, כך הרשת הניורנית רחוקה יותר לפלט הרצוי. לכן מטרתנו היא למצוא את המינימום של הפונקציה כדי להבטיח נכונות בפלט.



## Evolutionary Algorithms – אלגוריתמים אבולוציוניים

אלגוריתמים אבולוציוניים מבוססים על מושגים של אבולוציה ביולוגית. תחילה נוצרת 'אוכלוסייה' של פתרונות אפשריים לבעיה כאשר כל פתרון נבדק באמצעות 'פונקציית כושר' (Fitness Function) המציינת עד כמה הם טובים. האוכלוסייה מתפתחת עם הזמן ומזהה פתרונות טובים יותר. מבין הסוגים השונים של אלגוריתמים אבולוציוניים, האלגוריתם הגנטי הוא הידוע ביותר והוא זה שאתאר אותו.

Fitness Function פונקציית כושר (הגדרה) – כאשר אנחנו רוצים לבדוק פתרון של רשת נוירונים אנחנו צריכים לקבוע פונקציית כושר. פונקציה כושר יכולה להיות לדוגמה כמה רחוק שחקן הגיע במשחק שתקבע שהפתרון הטוב ביותר הוא האחד שבאמצעותו

## Genetic Algorithm – אלגוריתם גנטי

לרבים ידועה התאוריה של צ'ארלס דרווין, לפיה רק החזקים שורדים (האנשים המתאימים ביותר נבחרים להתרבות כדי לייצר צאצאים לדור הבא) שמבוססת על ההתפתחות על ידי ברירה טבעית. בהשראת התיאוריה של דרווין, Genetic Algorithm הוא חלק Evolutionary Algorithms, שנועד במיוחד כדי לייצר פתרונות איכותיים לבעיות אופטימיזציה וחיפוש על ידי הסתמכות על אופרטורים בהשראה ביולוגית כמו מוטציה, הצלבה ובחירה.

Genetic Algorithm יש שתי מטרות עיקריות:

1. חיפוש

2. אופטימיזציה

Genetic Algorithms משתמשים בתהליך **איטרטיבי** כדי להגיע לפתרון הטוב ביותר - מציאת הפתרון הטוב ביותר מתוך הפתרונות הטובים ביותר (הטוב ביותר). בהשוואה לבחירה טבעית, הטוב ביותר ישרוד בהשוואה לאחרים.

שיטה איטרטיבית (הגדרה) - במתמטיקה, שיטה איטרטיבית היא שיטה שמשתמשת בניחוש התחלתי כדי ליצור סדרת קירובים טובים יותר ויותר לפתרון בעיה נתונה.



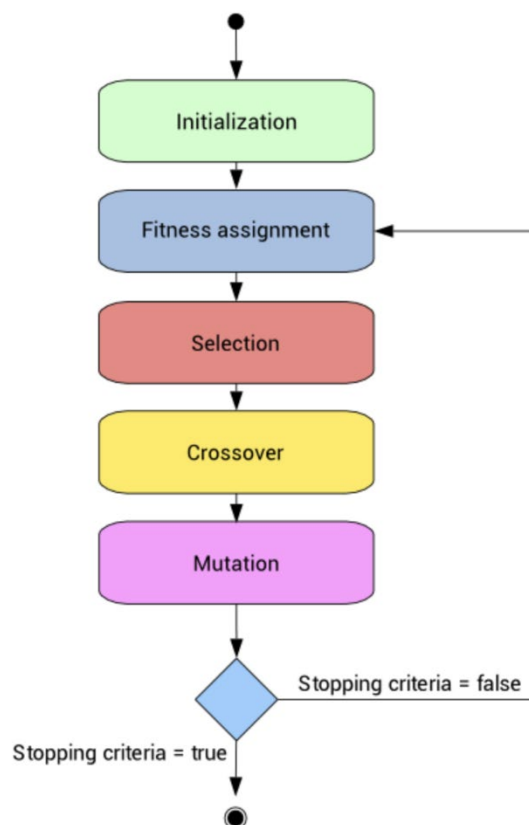
נדגים הלכה למעשה את תהליך הברירה הטבעית של דרווין כ Genetic Algorithm:

- האבולוציה מתחילה בדרך כלל מאוכלוסייה של יצורים שנוצרו באופן אקראי בצורה של איטרציה. (איטרציה תוביל לדור חדש).
- בכל איטרציה או דור, הכושר של כל אדם נקבע לבחור את המתאים ביותר.
- האנשים המתאימים ביותר בדור שנבחרו עוברים מוטציה או שינוי כדי ליצור דור חדש, והתהליך נמשך עד שהפתרון הטוב ביותר הגיע

איטרציה (הגדרה) – איטרציה היא פעולה שחוזרת על עצמה במהלך פתרון של בעיה, בדרך כלל בעיה כמותית. תהליך המורכב מאיטרציות קרוי תהליך איטרטיבי.

התהליך מסתיים באחד משני המצבים הבאים:

1. הגענו למקסימום כמות דורות שרצינו ליצור
2. הגענו לרמת כושר היעילה ביותר של האובייקט.



## בפרויקט שלי:

1. נוצר דור של 20 ציפורים עם קשרים רנדומליים
2. פונקציית הכושר שלי (Fitness Function) מוגדרת לפי מיקום הציפור – ככל שמיקום הציפור גבוה יותר כך פונקציית הכושר שלה גבוהה יותר.
3. לאחר כל הרצה נשמרות הציפורים עם פונקציית הכושר הטובות ביותר ועל ידי שילוב ושינויים קטנים רנדומליים בקשרים הנוירונים שלהם נוצר דור חדש של ציפורים
4. הדור החדש מבצע את המשחק וחוזר לשלב 3.
5. לבסוף אנחנו מגיעים או לכמות דורות מקסימלית אותה הגדרתי 10 או לציפור עם קשרי נוירונים מספיק טובים כדי להצליח את המשחק.

## Neat - Efficient Evolution of Neural Network Topologies

Neuroevolution - האבולוציה המלאכותית של רשתות עצביות המשתמשות באלגוריתמים גנטיים (Genetic Algorithm), הראתה הבטחה רבה במשימות למידה לחיזוק (Reinforcement learning).

שלושת הדברים העיקריים שעליהם עונה האלגוריתם Neat הם:

- לבצע את crossover בצורה לוגית.
- לשמור על הדורות הקודמים – מכיוון שהאלגוריתמים הגנטיים מתפתחים ומתווספים אליהם גם צאצאים רנדומליים, יש צורך לשמור על הדורות הקודמים, במיוחד אלה שתפקדו טוב. זו משימה קשה מכיוון שהרשת הולכת ומתפתחת עוד ועוד ונעשית מורכבת יותר ויותר.
- לשמור על הרשת פשוטה ככל האפשר בלי להשתמש בפונקציית מעקב לשם כך. קודם כל לא נפתח פונקציה כזאת כי זה מאוד קשה לפתח פונקציה שתמדוד את הפשוט ותעמוד בגבולות של המורכבות של הרשת שהיא מתפתחת עוד ועוד. הפתרון היה פה להתחיל מרשת קטנה ולפתח אותה רק כאשר יש צורך בכך.

## הסבר על הדרך שNEAT עובד

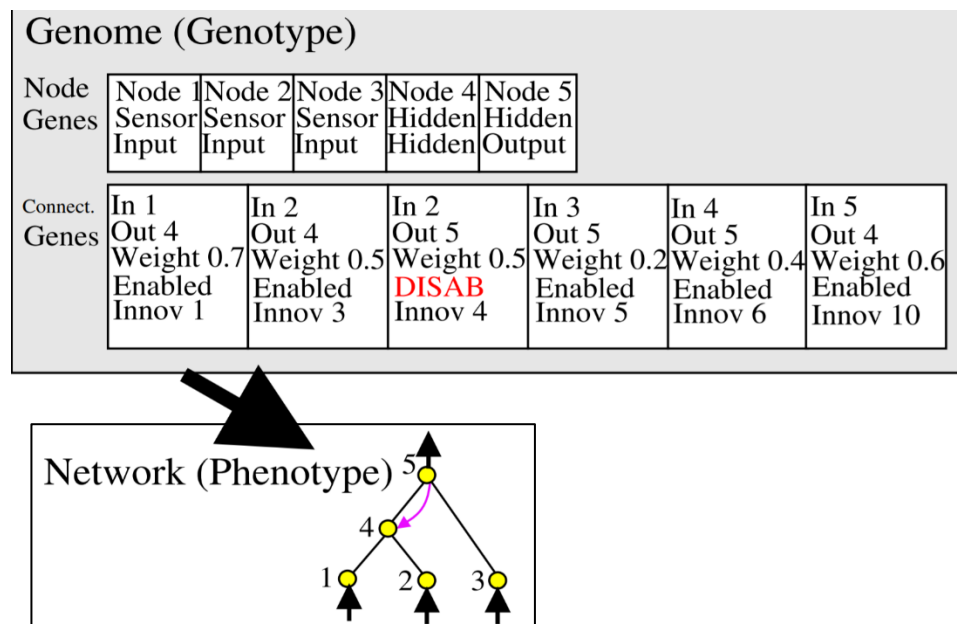
Neat נועד במיוחד לענות שלושת הדברים המרכזיים אותם הצגתי בעמוד הקודם.

### Genetic Encoding – קידוד גנטי

ראשית אנחנו מקבלים את הGenome – Genotype. זה למעשה רשימה של התאים, שכל תא מייצג את הקלט והפלט אליו הוא מחובר, את המשקל המוצמד ומשתנה בוליאני של אם הוא יהיה מחובר לתא אחר או לא.

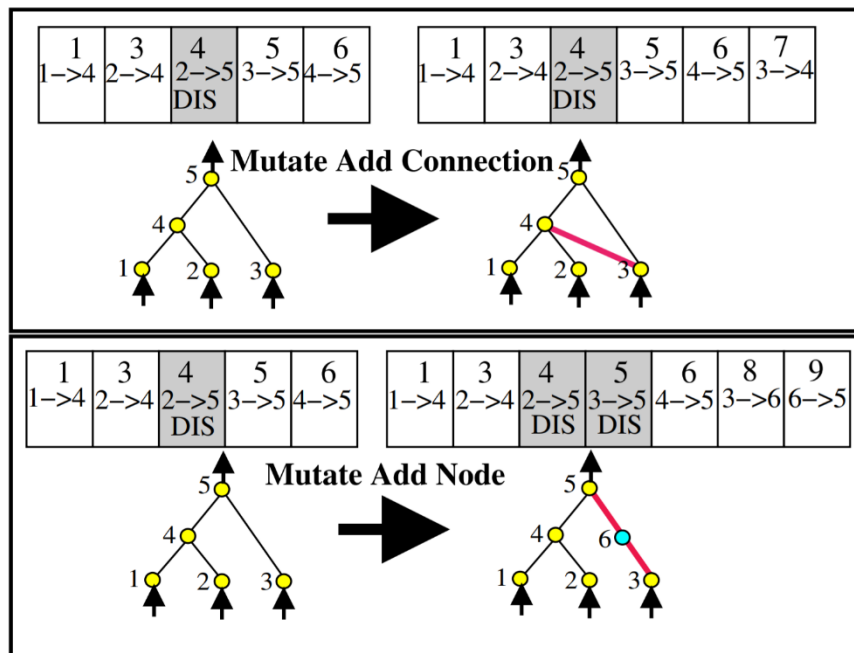
Genome (Genotype)						
Node	Node 1	Node 2	Node 3	Node 4	Node 5	
Genes	Sensor Input	Sensor Input	Sensor Input	Hidden Hidden	Hidden Output	
Connect. Genes	In 1 Out 4 Weight 0.7 Enabled Innov 1	In 2 Out 4 Weight 0.5 Enabled Innov 3	In 2 Out 5 Weight 0.5 <b>DISAB</b> Innov 4	In 3 Out 5 Weight 0.2 Enabled Innov 5	In 4 Out 5 Weight 0.4 Enabled Innov 6	In 5 Out 4 Weight 0.6 Enabled Innov 10

מהם אנחנו מייצרים את המבנה עצמו, שזה הnetwork שלנו או גם בשמו Phenotype. בעזרת רשימה זו אנחנו מרכיבים את הרשת עם החיבורים המתאימים ומקבלים:



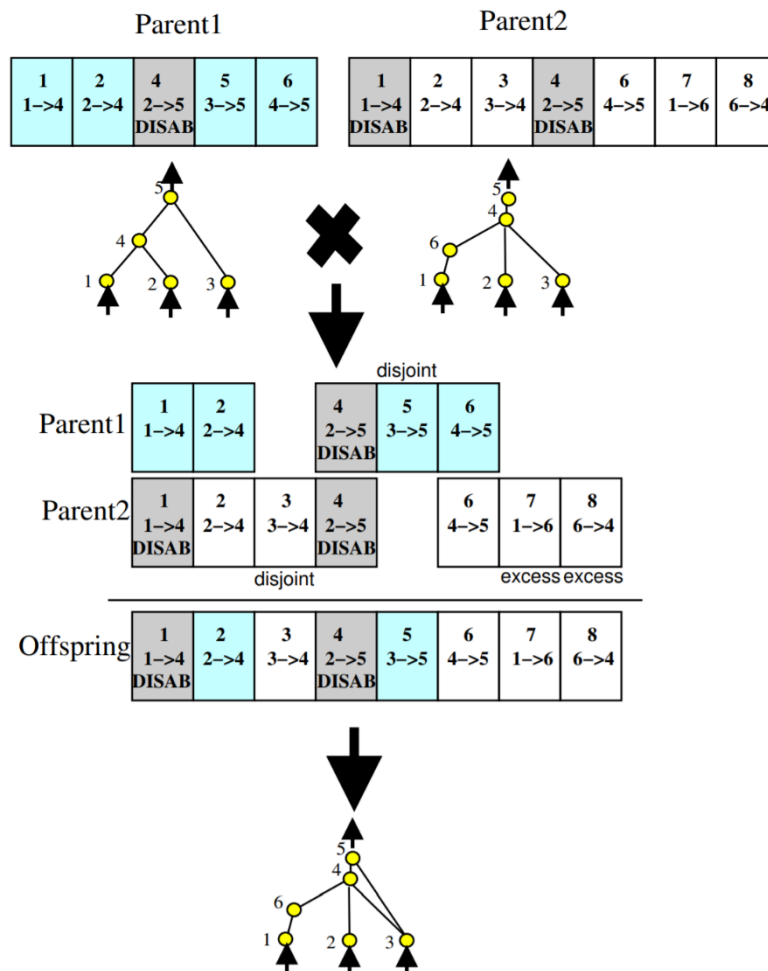
בנוסף ניתן לראות שכל תא בGenotype יש מספר ייצוגי – Innov. לפי המספרים הייצוגיים נדע כיצד לבצע את crossover.

כדי לחבר תא לתא עושים את זה בדרך פשוטה. מסתכלים על הGenotype ולפיו קובעים את מיקום התא. לדוגמה:



## Historical markings

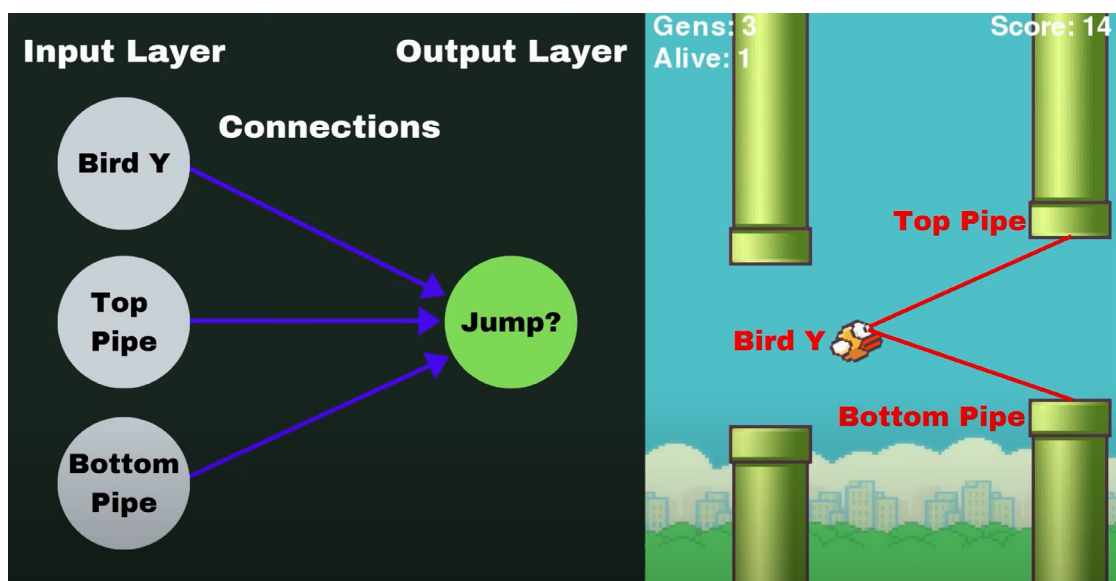
כעת נסביר על החיבורים של Genomes. מטרתנו היא לשמור על הדורות הקדומים גם, לכן אנחנו נבצע crossover רק בין תאים שיש להם מקור משותף. אנחנו נשמור על הדורות הקדומים בעזרת מספרי הייצוג, כך שכל צאצא יירש את המספר המייצג של ההורים שלו. כלומר, אנחנו נחבר את genomes לפי מספרי הייצוג, כך:



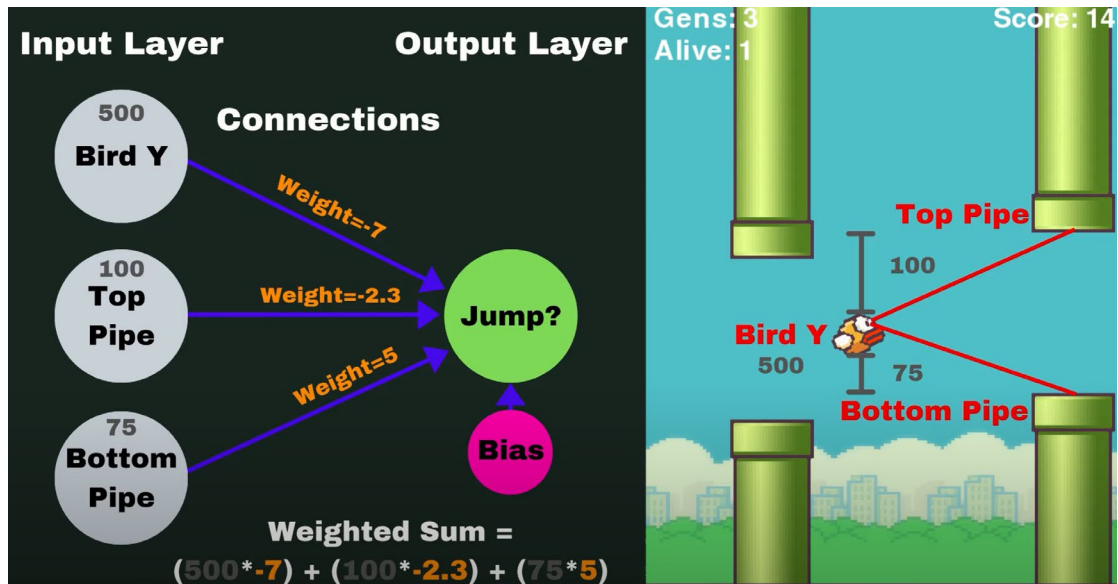
ניתן לראות שאנחנו נותנים עדיפות להורה עם fitness היותר קבוע ואותו אנחנו נשים למעלה. אם להורים מסוימים יש את אותו fitness אנחנו נבצע בהם crossover בצורה רנדומלית. מכיוון שאנחנו יוצרים כללים ברורים אנחנו מפשטים את המערכת ונותנים לה לפתור בקלות את הבעיה. אין צורך באנליזה של המבנה מכיוון שאנחנו מתחילים מקטן ומגדילים אותו לפי הצורך – לדוגמה ניתן לראות שאת ההורים עם הייצוג 1 ו-4 לא הכנסנו.

## האלגוריתם של NEAT מופעל בפרויקט שלי

המשחק מתחיל בדור רנדומלי של ציפורים, שלא יודעות כלום על המשחק וכיצד הוא פועל. כעת אנחנו רוצים להגדיר את הinputs שלנו שנמצאים בinput layer. בחרתי בקלט: מיקום הציפור, מיקום במכשול שלמעלה ומיקום המכשול שלמטה מכיוון שקלטים אלה מספיקים כדי ללמוד כיצד להתקדם במשחק. כפלט בחרתי להיות משתנה בוליאני של לקפוץ או לא לקפוץ. מה שנעשה זה ניתן קלטים למשתני קלט שלנו, אחרי זה נעביר אותם תהליך ה hidden layer שהערך הסופי יקבע אם הציפור תקפוץ או לא.

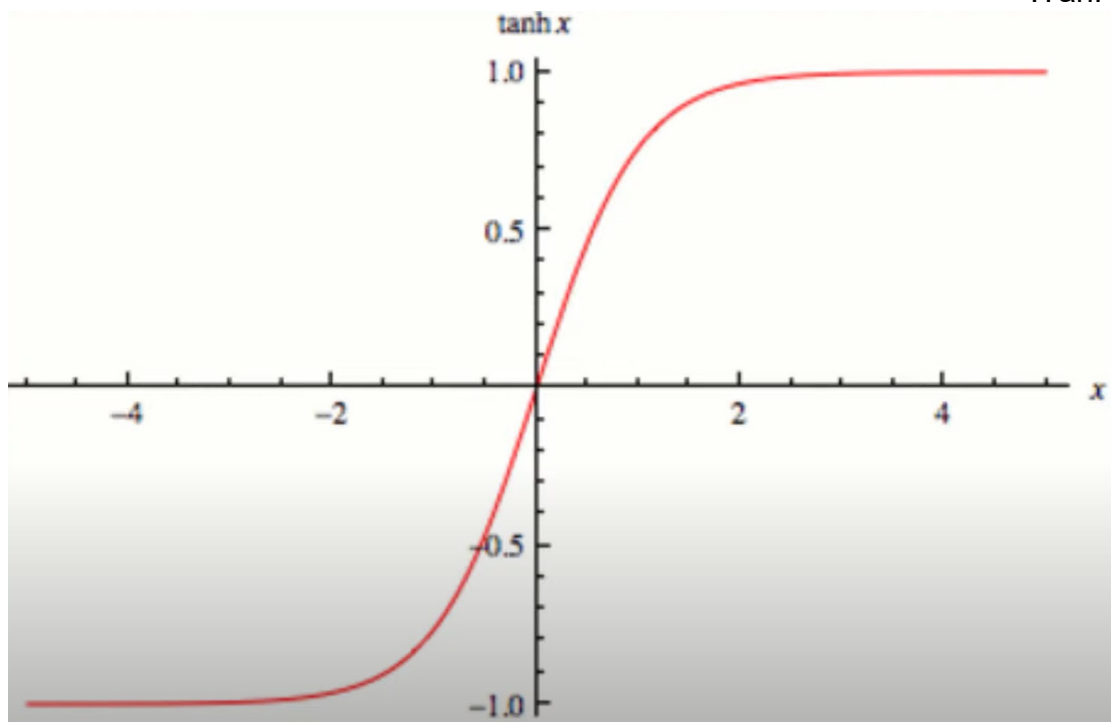


לכל חיבור יש את המשקל שלו שהוא פרמטר לכמה חזק אותו החיבור. על ידי חשיבות החיבורים לקפיצה המשתנים ישתנו ולבסוף נקבל את הסכום שלפיו נחליט לקפוץ או לא. כעת אנחנו רוצים לשים את הרשת שלנו במיקום הנכון ולכן אנחנו נשתמש ב-bias.



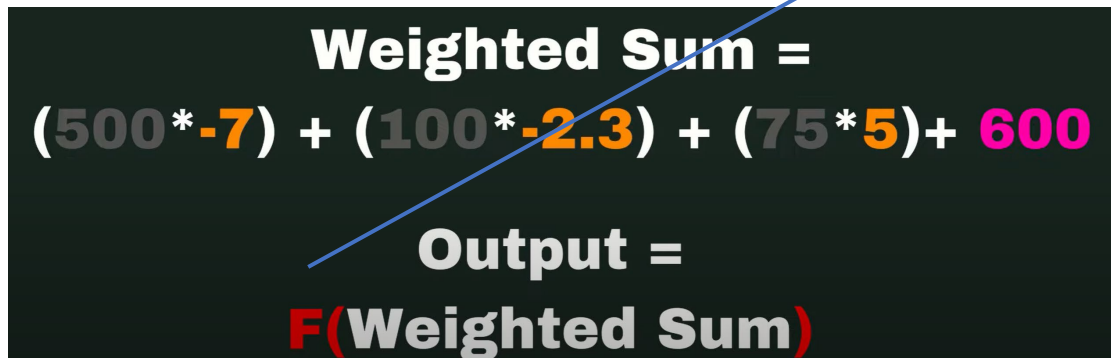
בנוסף, כדי שפונקציית הפלט הסופי תקבל את הערכים המתאימים אני אשתמש בפונקציה שתשים את הערך של הסכום במיקום שבין 1-11.

:TanH





את פונקציה TanH נסמן כActivation Functions ובקיצור F. כלומר ערך הפלט הסופי שלנו יהיה:


$$\text{Weighted Sum} = (500 * -7) + (100 * -2.3) + (75 * 5) + 600$$
$$\text{Output} = F(\text{Weighted Sum})$$

את המשחק אנחנו נתחיל עם אוכלוסייה של 25 ציפורים שלכל אחת רשת נוירונים עם משקלים רנדומליים bias רנדומלי. אנחנו נבדוק את כל רשתות הנוירונים בעזרת fitness function ונבדוק כמה הן טובות. את פונקציה fitness בחרתי להיות כמות המרחק שהציפור הגיעה, כאשר כל פריים זה עוד ניקוד וכאשר היא עוברת מכשול היא מקבלת עוד 5. לאחר הרצה של המשחק ניקח את הציפורים עם הfitness הגבוה ביותר ואיתן ניצור דור ציפורים נוסף שמבוסס על ערכי המשקלים והbias שלהם.

#### סיכום של הפעלת רשת הנוירונים:

Inputs: bird Y, top pipe, bottom pipe

Output: Jump?

Activation Function: TanH

Population size: 20

Fitness Function: distance

Max generations: 30

## הקוד של הבינה מלאכותית

נתחיל בהסבר על Configuration file. מסמך זה הוא מסמך תצורה שבו נקבעים מרבית ההגדרות של הרשת NEAT.

[NEAT]

`fitness_criterion = max`

מקבל ערכים של מינימום ומקסימום. המשתנה קובע האם ניקח לדור הבא את הציפור עם ה-fitness function הכי גבוה או הכי נמוך.

`fitness_threshold = 100`

קובע את הסף של ה-fitness function שכאשר ציפור מגיעה אליו או יותר סימן שהרשת מוכנה ואפשר לעצור את המשך הדורות, כלומר הרשת מספיק טובה.

`pop_size = 50`

קובע את כמות הציפורים בכל דור

`reset_on_extinction = False`

אם זה יקבל ערך True, כאשר כל המינים נכחדים בו זמנית בגלל סטגנציה, תיווצר אוכלוסייה אקראית חדשה. אם יקבל False, יישלח ExceptionExceptionException.

[DefaultGenome] – הציפורים הן למעשה genomes שלנו, נרצה להגדיר את ההגדרות הברירות מחדל של כל ציפור (כפי שראינו בחלק על NEAT הכוונה היא לתא). יש להגדיר הגדרות של התא עצמו וההגדרות של החיבור.

# node activation options – ההגדרות של התאים

`activation_default = tanh`

ה-activation Function שאותה אנחנו רוצים להגדיר כברירת מחדל לכל ציפור.

`activation_mutate_rate = 0.0`

מספק אחוז לשינוי של ה-activation function, אנחנו רוצים להשתמש רק ב-tanh ולכן נגדיר את זה להיות 0.

`activation_options = tanh`

האופציות שלנו ל-activation function, אנחנו רוצים להשתמש רק ב-tanh ולכן נגדיר רק אותה.

# node aggregation options – צבירה של התאים

`aggregation_default = sum`

`aggregation_mutate_rate = 0.0`

`aggregation_options = sum`

# node bias options – הגדרות הברירות מחדל של החיבורים

bias\_init\_mean = 0.0

bias\_init\_stddev = 1.0

bias\_max\_value = 30.0

bias\_min\_value = -30.0

ערכי מינימום ומקסימום של bias. כאשר אנחנו קובעים את הbias הרנדומליים הראשוניים לדור הראשון של הציפור אנחנו רוצים להגביל אותו.

bias\_mutate\_power = 0.5

bias\_mutate\_rate = 0.7

bias\_replace\_rate = 0.1

קובע את הסיכוי של הפרמטרים להשתנות כאשר אנחנו יוצרים אוכלוסייה חדשה של ציפורים.

# genome compatibility options

compatibility\_disjoint\_coefficient = 1.0

compatibility\_weight\_coefficient = 0.5

# connection add/remove rates

conn\_add\_prob = 0.5

conn\_delete\_prob = 0.5

קובע את הסיכוי להוסיף או להוריד חיבורים.

# connection enable options

enabled\_default = True

enabled\_mutate\_rate = 0.01

כפי שראינו בדוגמה בנושא הNEAT, ישנם מקרים כאשר החיבור בין שני תאים לא מופעל. נקבע שערך הברירת מחדל יהיה שכולם מופעלים ויש סיכוי של אחוז אחד שחיבור יוסר.

feed\_forward = True

אנחנו משתמשים ברשת נוירונים שהיא feed forward

initial\_connection = full

אנחנו מתחילים בשכבה שבה כל התאים מחוברים

# node add/remove rates

node\_add\_prob = 0.2

node\_delete\_prob = 0.2

יש סיכוי של 20 אחוזים להוסיף תא או להסיר תא

# network parameters

num\_hidden = 0

```
num_inputs = 3
num_outputs = 1
```

קובע את כמות הנירונים שהם קלטים ואת כמות הנירונים שהם בפלט ואת כמות הנירונים שיש בשכבות שבין לבין. מכיוון שאנחנו משתמשים בNEAT אנחנו מתחילים מרשת נירונים פשוטה ככל האפשר ולכן אין נירונים בשכבות הביניים "הנסתרות".

```
# node response options
response_init_mean = 1.0
response_init_stdev = 0.0
response_max_value = 30.0
response_min_value = -30.0
response_mutate_power = 0.0
response_mutate_rate = 0.0
response_replace_rate = 0.0
```

# connection weight options – הגדרות ברירת המחדל של המשקלים

```
weight_init_mean = 0.0
weight_init_stdev = 1.0
```

```
weight_max_value = 30
weight_min_value = -30
```

המשקלים הראשונים נקבעים רנדומלית ולכן נרצה להגביל אותם.

```
weight_mutate_power = 0.5
weight_mutate_rate = 0.8
weight_replace_rate = 0.1
```

מציין את הסיכוי באחוזים שהמשקלים יתחלפו מדור לדור. אנחנו מחפשים משקלים מתאימים ולכן לא נרצה שהם יתחלפו רנדומלית.

```
[DefaultSpeciesSet]
compatibility_threshold = 3.0
```

```
[DefaultStagnation]
```

```
species_fitness_func = max
```

מקבל ערכים של מינימום ומקסימום. המשתנה קובע האם ניקח לדור הבא את הציפור עם ההfitness function הכי גבוה או הכי נמוך.

```
max_stagnation = 20
```

ציפורים שלא הראו שיפור יוסרו לאחר 20 דורות.

`species_elitism = 2`

קובע את כמות הציפורים שעליהן נשמור, כל דור נשמור 2 ציפורים עם הfitness הכי גבוה. נועד כדי למנוע היכחדות.

[DefaultReproduction]

`elitism = 2`

כמות הציפורים עם הfitness הכי גבוה שישמרו מדור לדור.

`survival_threshold = 0.2`

כמות הציפורים שנבצע ביניהם mutation- כלומר בכל דור נבצע mutation עם 20 אחוז מהציפורים עם הfitness הכי גבוה.

## זימון Configuration file

כדי לזמן את הקובץ השתמשתי בשורות הקוד הבאות:

```
if __name__ == "__main__":
    local_dir = os.path.dirname(__file__)
    config_path = os.path.join(local_dir, "configGame")
    run(config_path)
```

ואז בפונקציית החזר שלנו אנחנו מזמנים את הקובץ עצמו:

```
def run(config_path):
    config = neat.config.Config(neat.DefaultGenome, neat.DefaultReproduction,
                                neat.DefaultSpeciesSet,
                                neat.DefaultStagnation, config_path)
```

כעת נסביר על שורות הקוד המעשי.

כדי ליצור את הצאצאים בדור השתמשתי בשורת הקוד:

```
p = neat.Population(config) # set a population
```

וכדי לזמן את הדור השתמשתי בשורת הקוד:

```
winner = p.run(main, 50) # we call the fitness function 50 times
```

fitness זה ה Main  
function

כמות הפעמים שאנחנו  
מזמנים את הפונקציה

כעת נעבור לקוד ה main שהוא ה fitness function אצלי:

הגדרת המשתנים:

```
global GEN
GEN += 1
nets = []
ge = []
birds = []
```

צריך לשמור על הרשת נירונים מתאימה  
לכל ציפור – לכן אני יוצר רשימה שלהם

כעת נבנה את הרשת נירונים שלנו:

```
for _, g in genomes: # genome is a tuple that has (1, genome object) and we only
    care about the object
    # set up a neural network for genome
    net = neat.nn.FeedForwardNetwork.create(g, config)
    nets.append(net)
    birds.append(Bird(230, 350))
    g.fitness = 0
    ge.append(g)
```

הגדרות נוספות:

```
base = Base(730) # set a base
pipes = [Pipe(700)] # set a pipes
win = pygame.display.set_mode((WIN_WIDTH, WIN_HEIGHT)) # set our window game
clock = pygame.time.Clock() # we set a frame number that will act like time
score = 0
run = True
```

כעת נסתכל על run:

כאשר ציפור נמצאת על המסך, בכל frame מופיעים שני זוגות של מכשולים. אנחנו נרצה שהציפור תסתכל רק על הזוג הראשון. לשם כך השתמשתי בשורות הקוד:

```
# WE HAVE MAX OF TWO PAIRS OF PIPES IN A FRAME AND WE WANT THAT OUR BIRD WILL LOOK IN THE
# FIRST PAIR:
pipe_ind = 0
if len(birds) > 0:
    if len(pipes) > 1 and birds[0].x > pipes[0].x + pipes[0].PIPE_TOP.get_width():
        pipe_ind = 1
else: # if there is no birds left
    run = False
    break
```

### הגדרת fitness

נרצה להגדיר לציפור את החוקים שלפיה היא תשחק, עושים זאת בעזרת ה fitness function. נרצה פרמטר מסוים שלפיו יוצרו הדורות הבאים שישתפרו מהקודמים. כדי לחשוב על פרמטר מתאים קל לנסות לחשוב כיצד אני קובע עד כמה ציפור טובה, וזה לפי המרחק שהיא עברה כמובן. לכן בחרתי במרחק להיות הפרמטר שקובע את הכושר של הציפור. אנחנו רוצים להגדיל את fitness של הציפור בכל frame שהיא מתקדמת. לשם כך כתבתי:

```
for x, bird in enumerate(birds):
    bird.move()
    ge[x].fitness += 0.1
```

נשים לב שההוספה של fitness נראה זניח יחסית, בכל שניה יש 30 frames ולכן 0.1 הוא מספר מתאים לחישוב כזה, שהרי בשנייה אחת שהציפור חייה היא מוסיפה לכושר שלה עוד  $30 \times 0.1$ , כלומר 3.

### ענישה ומתנה – סוג של Reinforcement

כדי שהציפורים ישתפרו משמעותית מדור לדור, נרצה לתת להם חיזוקים חיוביים על הפעולות שצריך לעשות. לשם כך בחרתי את הפרמטרים הבאים:

1. התנגשות במכשול מורידה את הכושר של הציפור
2. מעבר במכשול מעלה את הכושר של הציפור
3. התנגשות בשמים או באדמה מורידה את הכושר של הציפור

חשוב לציין שכל אחד מהפרמטרים הללו הוא פרמטר מרכזי בפתרון הבעיה, משום שלפיו נקבעים הציפורים הבאות טובות. לדוגמה בהתחלה התעלמתי מהפרמטר השלישי, וכאשר ניסיתי להריץ הקוד כך שבכל דור יהיו 5 ציפורים הרבה פעמים חלקן לא הצליחו ונפלו

לאדמה או לשמיים. לאחר ההוספה של הפרמטר השלישי הצלחתי לצלוח את המשחק לאחר 6 דורות של 5 ציפורים! שיפור מטורף למדי.

כאשר ציפור מתנגשת במכשול, אנחנו רוצים ש:

- א. היא תעלם
- ב. ללמד אותה שזה לא טוב – לשם כך נוריד את הערך של fitness שלה
- ג. להוציא את הציפור הזאת מהרשת ניירונים ולהוציא אות מהדור

לכן כתבתי את השורות:

```
for pipe in pipes:
    for x, bird in enumerate(birds):
        # when bird collide into a pipe
        if pipe.collide(bird):
            ge[x].fitness -= 1 # when a bird touch a pipe is gonna remove from its
fitness score
            birds.pop(x) # remove bird
            nets.pop(x) # remove the neural network associated with that bird
            ge.pop(x) # remove this bird from the generation
```

כך, לציפור של נתקעה במכשול יהיה fitness יותר גבוה בהרבה מה שייתן לציפורים ערך לא להיתקע במכשולים.

באופן דומה, כאשר ציפור נוגעת באדמה או בשמיים אנחנו נרצה להוריד גם לה את fitness.

```
for x, bird in enumerate(birds):
    # we don't want that birds will go down the base or above the sky
    if bird.y + bird.img.get_height() >= 730 or bird.y < 0:
        ge[x].fitness -= 1 # when a bird touch a sky or a base we want to decrease
there fitness
        birds.pop(x) # remove bird
        nets.pop(x) # remove the neural network associated with that bird
        ge.pop(x) # remove this bird from the generation
```

מאידך, כאשר הציפור כן עוברת מכשול, אנחנו רוצים ללמד אותה שזה טוב, לכן כאשר ציפור עוברת מכשול נוסיף לה לfitness:

```
if add_pipe:
    score += 1
    for g in ge:
        g.fitness += 5 # we increase the fitness to encourage the bird go throw
the pipes
    pipes.append((Pipe(700)))
```



כעת נרצה לחשב את הoutput של הרשת נוירונים בעזרת הinputs. נקבע שכשאר הoutput גדול מ0.5 הציפור תקפוץ, אחרת היא לא.

```
# WE ACTIVATE OUR OUTPUT BY OUR INPUTS
output = nets[x].activate((bird.y, abs(bird.y - pipes[pipe_ind].height), abs(bird.y - pipes[pipe_ind].bottom)))
```

נמצא את המרחק בין המכשול שלמטה לבין הציפור

המידע הראשוני שצריך

נמצא את המרחק בין המכשול שלמעלה לבין הציפור

```
if output[0] > 0.5:
    bird.jump()
```

### הסבר על הסטטיסטיקות:

במהלך המשחק מופיעות סטטיסטיקות של כל דור.

הכושר הכי טוב

ממוצע הכושר

```
Population's average fitness: 2.98000 stdev: 2.35746
Best fitness: 7.60000 - size: (1, 3) - species 1 - id 2
Average adjusted fitness: 0.267
Mean genetic distance 0.886, standard deviation 0.776
Population of 5 members in 1 species:
  ID  age  size  fitness  adj fit  stag
  ===  ==  ===  =====  =====  =====
    1   1    5     7.6     0.267     0
Total extinctions: 0
Generation time: 2.947 sec (3.661 average)

***** Running generation 2 *****
```

גודל הדור

זמן הדור

## הקוד

```
import pygame
import neat
import time
import os
import random
pygame.font.init()

WIN_WIDTH = 600
WIN_HEIGHT = 800

GEN = 0

# upload all the images of the game
BIRD_IMGS = [pygame.transform.scale2x(pygame.image.load(os.path.join('imgs',
'bird1.png'))),
              pygame.transform.scale2x(pygame.image.load(os.path.join('imgs',
'bird2.png'))),
              pygame.transform.scale2x(pygame.image.load(os.path.join('imgs',
'bird3.png')))]
PIPE_IMG = pygame.transform.scale2x(pygame.image.load(os.path.join('imgs',
'pipe.png')))
BASE_IMG = pygame.transform.scale2x(pygame.image.load(os.path.join('imgs',
'base.png')))
BG_IMG = pygame.transform.scale2x(pygame.image.load(os.path.join('imgs',
'bg.png')))
STAT_FONT = pygame.font.SysFont("comicsans", 50)

class Bird:

    IMGS = BIRD_IMGS # Name it more short to make it easier to references
    MAX_ROTATION = 25 # how much the bird is going tilt
    ROT_VEL = 20 # how much we going rotate in each frame/move
    ANIMATION_TIME = 5 # effect how fast the bird is going to flappy it wings

    def __init__(self, x, y):
        self.x = x # Represent the started position of the bird
        self.y = y
        self.tilt = 0 # How much the image is actually tilting
        self.tick_count = 0
        self.vel = 0 # rapidity of movement
```

```

        self.height = self.y
        self.img_count = 0 # tracking which image we show
        self.img = self.IMGS[0]

    def jump(self):
        self.vel = -10.5 # it is minus because of the xy chart of the program
        self.tick_count = 0 # keep track of when we last jump
        self.height = self.y

    def move(self):
        self.tick_count += 1 # keep track of how much we moved

        # how many pixels we moving up or down each frame
        # d = displacement
        d = self.vel * self.tick_count + 1.5 * self.tick_count ** 2

        if d >= 16:
            d = 16

        if d < 0:
            d -= 2

        self.y = self.y + d

        if d < 0 or self.y < self.height + 50:
            if self.tilt < self.MAX_ROTATION: # when we go up we don't want go 90
degree up
                self.tilt = self.MAX_ROTATION
            else:
                if self.tilt > -90: # when we go down we want to go 90 degree down
                    self.tilt -= self.ROT_VEL

    def draw(self, win):
        self.img_count = self.img_count + 1

        if self.img_count < self.ANIMATION_TIME:
            self.img = self.IMGS[0]
        elif self.img_count < self.ANIMATION_TIME * 2:
            self.img = self.IMGS[1]
        elif self.img_count < self.ANIMATION_TIME * 3:
            self.img = self.IMGS[2]
        elif self.img_count < self.ANIMATION_TIME * 4:
            self.img = self.IMGS[1]
        elif self.img_count == self.ANIMATION_TIME * 4 + 1:

```

```

        self.img = self.IMGS[0]
        self.img_count = 0

    if self.tilt <= -80:
        self.img = self.IMGS[1]
        self.img_count = self.ANIMATION_TIME * 2

    rotated_image = pygame.transform.rotate(self.img, self.tilt)
    new_rect = rotated_image.get_rect(center=self.img.get_rect(topleft=(self.x,
self.y)).center)
    win.blit(rotated_image, new_rect.topleft)

    def get_mask(self): # use when we get collision between objects
        return pygame.mask.from_surface(self.img)

class Pipe:
    GAP = 200 # the space between the pipes
    VEL = 3 # our bird doesn't move but all the other objects are moving

    def __init__(self, x):
        self.x = x
        self.height = 0
        self.gap = 100
        self.top = 0 # top of our pipe is going to be draw
        self.bottom = 0 # bottom of our pipe is going to be draw
        self.PIPE_TOP = pygame.transform.flip(PIPE_IMG, False, True) # to flip the
image of the pipe
        self.PIPE_BOTTOM = PIPE_IMG

        self.passed = False # check if the bird is already passed the pipe ;
collisions ; AI
        self.set_height()

    def set_height(self):
        self.height = random.randrange(50, 450)
        self.top = self.height - self.PIPE_TOP.get_height()
        self.bottom = self.height + self.GAP

    def move(self):
        self.x -= self.VEL

    def draw(self, win):
        win.blit(self.PIPE_TOP, (self.x, self.top))

```

```

win.blit(self.PIPE_BOTTOM, (self.x, self.bottom))

def collide(self, bird):
    bird_mask = bird.get_mask()
    top_mask = pygame.mask.from_surface(self.PIPE_TOP)
    bottom_mask = pygame.mask.from_surface(self.PIPE_BOTTOM)

    # check points of collision
    top_offset = (self.x - bird.x, self.top - round(bird.y))
    bottom_offset = (self.x - bird.x, self.bottom - round(bird.y))

    # return us true or false about if birds is touching pipe
    b_point = bird_mask.overlap(bottom_mask, bottom_offset)
    t_point = bird_mask.overlap(top_mask, top_offset)

    if t_point or b_point:
        return True
    return False

class Base:
    VEL = 2
    WIDTH = BASE_IMG.get_width()
    IMG = BASE_IMG

    def __init__(self, y):
        self.y = y
        self.x1 = 0
        self.x2 = self.WIDTH # located directly behinde the image

    def move(self):
        self.x1 -= self.VEL
        self.x2 -= self.VEL

        if self.x1 + self.WIDTH < 0:
            self.x1 = self.x2 + self.WIDTH

        if self.x2 + self.WIDTH < 0:
            self.x2 = self.x1 + self.WIDTH

    def draw(self, win):
        win.blit(self.IMG, (self.x1, self.y))
        win.blit(self.IMG, (self.x2, self.y))

```

```

def draw_window(win, birds, pipes, base, score, gen):
    win.blit(BG_IMG, (0, 0))
    for pipe in pipes:
        pipe.draw(win)

    text = STAT_FONT.render("Score: " + str(score), 1, (255, 255, 255))
    win.blit(text, (WIN_WIDTH - 10 - text.get_width(), 10))

    text = STAT_FONT.render("Gen " + str(gen), 1, (255, 255, 255))
    win.blit(text, (10, 10))

    base.draw(win)

    for bird in birds:
        bird.draw(win)

    pygame.display.update()

def main(genomes, config):
    global GEN
    GEN += 1
    nets = []
    ge = []
    birds = []

    for _, g in genomes: # genome is a tuple that has (1, genome object) and we
        # only care about the object
        # set up a neural network for genome
        net = neat.nn.FeedForwardNetwork.create(g, config)
        nets.append(net)
        birds.append(Bird(230, 350))
        g.fitness = 0
        ge.append(g)

    base = Base(730)
    pipes = [Pipe(700)]
    win = pygame.display.set_mode((WIN_WIDTH, WIN_HEIGHT))
    clock = pygame.time.Clock()
    score = 0
    run = True

    while run:

```

```

clock.tick(30)
for event in pygame.event.get():
    if event.type == pygame.QUIT:
        run = False
        pygame.quit()
        quit()

# WE HAVE MAX OF TWO PAIRS OF PIPES IN A FRAME AND WE WANT THAT OUR BIRD
WILL LOOK IN THE
# FIRST PAIR:
pipe_ind = 0
if len(birds) > 0:
    if len(pipes) > 1 and birds[0].x > pipes[0].x +
pipes[0].PIPE_TOP.get_width():
        pipe_ind = 1
else: # if there is no birds left
    run = False
    break

for x, bird in enumerate(birds):
    bird.move()
    ge[x].fitness += 0.1

# WE ACTIVATE OUR OUTPUT BY OUR INPUTS
output = nets[x].activate((bird.y, abs(bird.y -
pipes[pipe_ind].height), abs(bird.y - pipes[pipe_ind].bottom)))

if output[0] > 0.5:
    bird.jump()

add_pipe = False
rem = []
for pipe in pipes:
    for x, bird in enumerate(birds):
        # when bird collide into a pipe
        if pipe.collide(bird):
            ge[x].fitness -= 1 # when a bird touch a pipe is gonna remove
from its fitness score
            birds.pop(x) # remove bird
            nets.pop(x) # remove the neural network associated with
that bird
            ge.pop(x) # remove this bird from the generation

if not pipe.passed and pipe.x < bird.x:

```

```

        pipe.passed = True
        add_pipe = True

    if pipe.x + pipe.PIPE_TOP.get_width() < 0:
        rem.append(pipe)

    pipe.move()

    if add_pipe:
        score += 1
        for g in ge:
            g.fitness += 5 # we increase the fitness to encourage the bird go
throw the pipes
        pipes.append((Pipe(700)))

    for r in rem:
        pipes.remove(r)

    for x, bird in enumerate(birds):
        # we don't want that birds will go down the base or above the sky
        if bird.y + bird.img.get_height() >= 730 or bird.y < 0:
            birds.pop(x) # remove bird
            nets.pop(x)  # remove the neural network associated with that
bird
            ge.pop(x)    # remove this bird from the generation

    base.move()
    draw_window(win, birds, pipes, base, score, GEN)

def run(config_path):
    config = neat.config.Config(neat.DefaultGenome, neat.DefaultReproduction,
neat.DefaultSpeciesSet,
                                neat.DefaultStagnation, config_path)

    p = neat.Population(config) # set a population

    # show us stats
    p.add_reporter(neat.StdOutReporter(True))
    stats = neat.StatisticsReporter()
    p.add_reporter(stats)

    # set the fitness function
    winner = p.run(main, 10) # we call the fitness function 50 times

```



```

if __name__ == "__main__":
    local_dir = os.path.dirname(__file__)
    config_path = os.path.join(local_dir, "configGame")
    run(config_path)

```

## Config File

מהקובץ הזה לקחתי את כל הפרמטרים הדרושים לבניית הרשת ניורונים

```

[NEAT]
fitness_criterion      = max
fitness_threshold      = 100
pop_size               = 20
reset_on_extinction    = False

[DefaultGenome]
# node activation options
activation_default      = tanh
activation_mutate_rate  = 0.0
activation_options      = tanh

# node aggregation options
aggregation_default     = sum
aggregation_mutate_rate = 0.0
aggregation_options     = sum

# node bias options
bias_init_mean          = 0.0
bias_init_stdev         = 1.0
bias_max_value          = 30.0
bias_min_value          = -30.0
bias_mutate_power       = 0.5
bias_mutate_rate        = 0.7
bias_replace_rate       = 0.1

# genome compatibility options
compatibility_disjoint_coefficient = 1.0
compatibility_weight_coefficient   = 0.5

# connection add/remove rates
conn_add_prob           = 0.5
conn_delete_prob        = 0.5

```

```

# connection enable options
enabled_default      = True
enabled_mutate_rate   = 0.01

feed_forward         = True
initial_connection    = full

# node add/remove rates
node_add_prob        = 0.2
node_delete_prob      = 0.2

# network parameters
num_hidden            = 0
num_inputs            = 3
num_outputs           = 1

# node response options
response_init_mean    = 1.0
response_init_stdev    = 0.0
response_max_value     = 30.0
response_min_value     = -30.0
response_mutate_power  = 0.0
response_mutate_rate   = 0.0
response_replace_rate  = 0.0

# connection weight options
weight_init_mean      = 0.0
weight_init_stdev     = 1.0
weight_max_value      = 30
weight_min_value      = -30
weight_mutate_power    = 0.5
weight_mutate_rate     = 0.8
weight_replace_rate    = 0.1

[DefaultSpeciesSet]
compatibility_threshold = 3.0

[DefaultStagnation]
species_fitness_func = max
max_stagnation       = 20
species_elitism       = 2

[DefaultReproduction]

```

```
elitism          = 2  
survival_threshold = 0.2
```