**Final project: Neuro-Evolutional "Don't Touch the Spikes" with Erlang.**

**Creators: Tamir Cohen and Nadav Hadad.**

**Course: Functional Programming in Concurrent and Distributed Systems, Ben Gurion University.**

**Professor: Yehuda Ben-Shimol.**

**Teaching Assistant: David Leon.**  
  
**Graphical user interface

Description automatically generated**

### התכנון שכתבנו בהתחלה

המאסטר:

מחלק עבודה בין 4 המחשבים. למשל אם בכל איטרציה של אוכלוסיה יש 1000 ציפורים, אז השרת שולח לכל אחד מהם 250 ציפורים (ויש 1000 רשתות נוירונים שונות סך הכל, כלומר משקולות שונות). כל מחשב נותן לציפור "לרוץ" ומגדיר לה פונקציית רווח (fitness function) שתהיה שווה לכמות הפריימים שהציפור בחיים.  
המחשבים אומרים לשרת מי הם 10% הציפורים הכי טובות שלהם והמאסטר משלב את המידע ומוליד את הדור הבא, שהוא השארה של הציפורים הכי טובות מהדור הקודם וגם מוטציות שלהן (ע"י רינדום קל במשקולות הרשת שלהם). וכך ממשיכים עוד איטרציות.

* נשלח הודעה im\_alive כל שניה ובמקרה והיא לא הגיעה נכריז על המחשב כנעדר ונחלק את העבודה בין שאר המחשבים שעדיין בחיים.

**מכונת מצבים** של כל **ציפור** (לכל פרוסס):

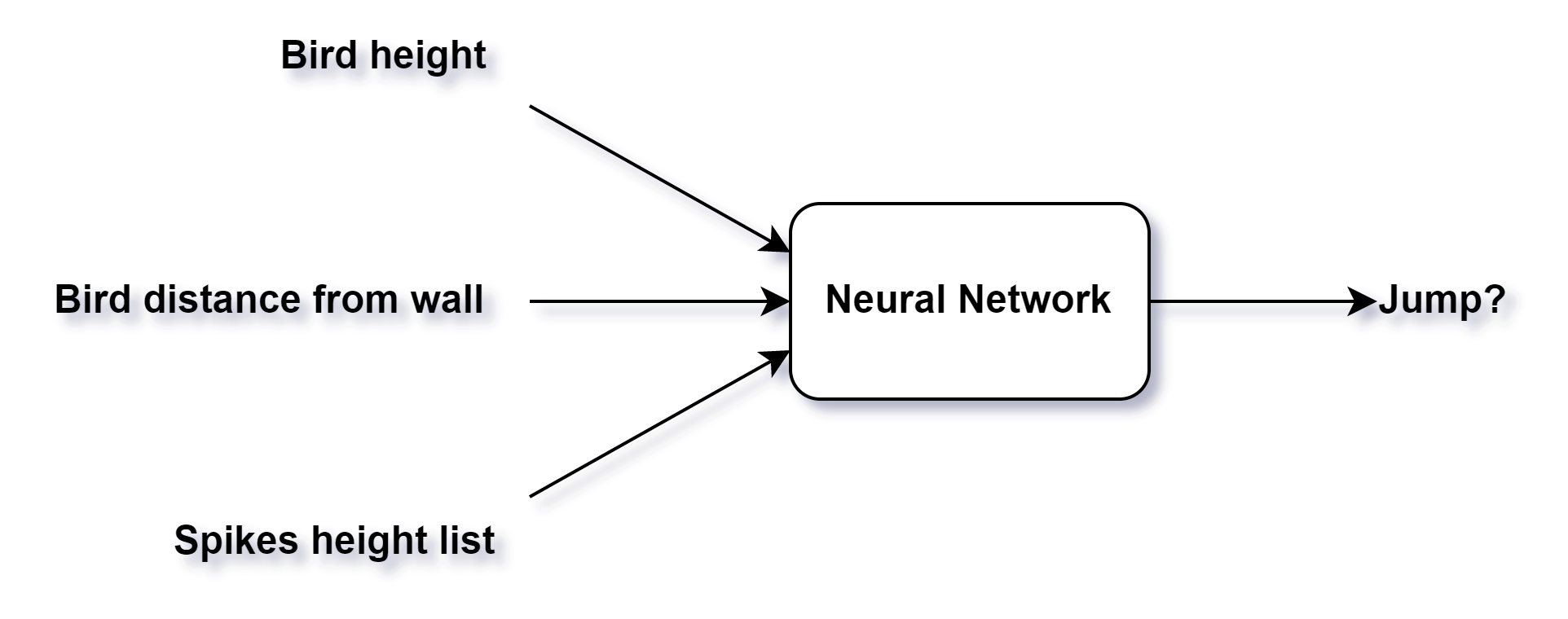
1. המתנה ללחיצה על הכפתור של המשחק או לקבלת רשת נוירונים לתפעול של הציפור. **idle**))
2. התחלת ריצה של המשחק (בעזרת הרשת שקיבל) עד פסילה (נגיעה בקוץ). (**simulation**)

**קלט** לרשת נוירונים:

1. גובה הציפור.
2. מרחק הציפור מהקיר.
3. רשימת הקוצים שבקיר ממול.

**הפלט** מהרשת הוא האם לבצע קפיצה.

כמו באיור:



שלבי עבודה בסיסיים:

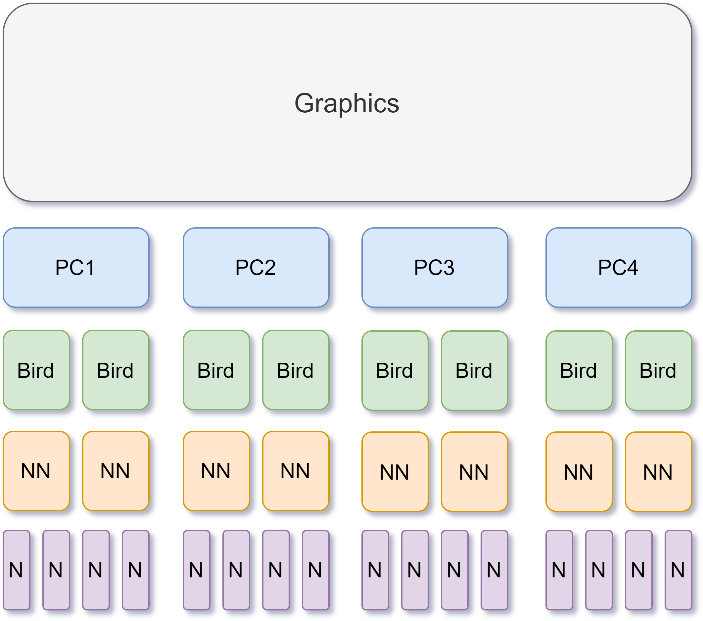
* תחילה נכין את המשחק שיעבוד אינטרקטיבית למשתמש.
* לאחר מכן נשנה אותו כך שאת הבחירה האם לקפוץ תבצע רשת הנוירונים ואז נוסיף המון ציפורים.
* לאחר מכן נפצל את העבודה בין טרמינלים שונים ולבסוף בין מחשבים שונים תוך תמיכה בנפילות.

שאלות ליהודה:

1. האם לעצור האלגוריתם מתישהו או שזה איטרטיבי לתמיד?  
   זה לא אמור להפסיק אלא אם כן אנחנו אומרים לו שהמשחק מפסיק כשמגיעים ל-score מסוים.
2. איך כדאי לייצג את רשימת הקוצים כקלט לרשת?  
   לפי מה שהבנתי ממנו, אפשר לייצג את זה כרשימת מספרים שמייצגת מיקום. בגלל שזה תמיד במקומות קבועים אז אפשר באמת לייצג את זה בצורה יותר נוחה של רשימת בוליאנים/ביטים. בהינתן מיקום של ביט ספציפי אפשר להתייחס אליו כאל מיקום בציר Y ולכן זה שקול.
3. מה קורה אם השרת עצמו נופל ולא אחד מ4 ה"עבדים"?  
   ניתן להניח שהשרת (שמריץ את הגרפיקה) לא נופל. רק 4 העבדים יכולים ליפול.

### הביצוע בפועל

הפרויקט שלנו מורכב **מ-5 שכבות**:



שכבה 1

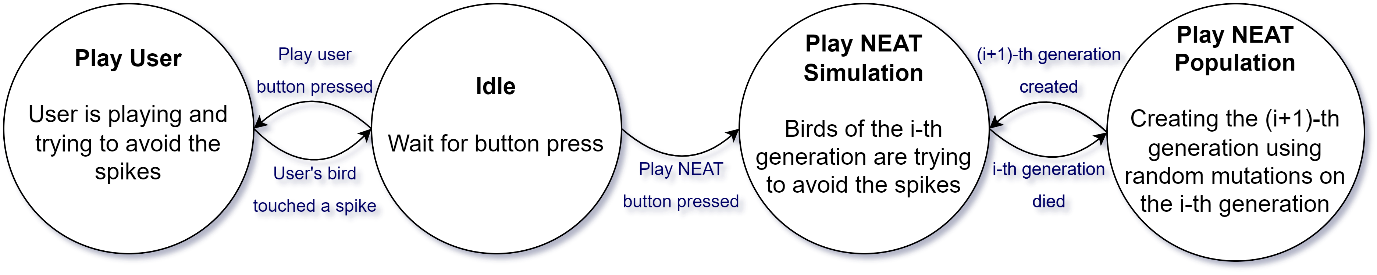
שכבה 2

שכבה 3

שכבה 4

שכבה 5

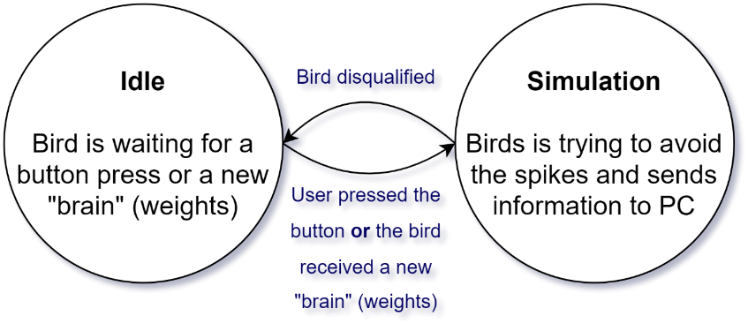
* + - 1. השכבה הראשונה היא **Graphics**. שכבה זו תנהל את הגרפיקה שהמשתמש מתנהל איתה (GUI). קיים רק מופע אחד של המודול הזה, שירוץ על המחשב הראשי, ויהיה אחראי על יצירה ואחזקה של 4 תהליכי PC. הוא ממומש לפי behaviour של **WxWidget** ופועל לפי ה-FSM הבא:



start

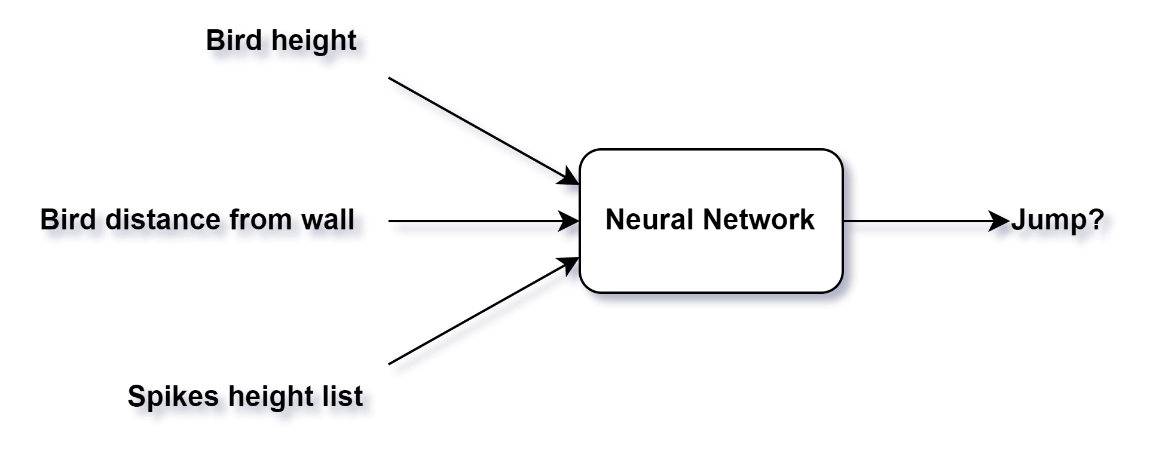
במכונת המצבים הזו קיימים 4 מצבים:

* 1. **Idle** – המתנה ללחיצה על אחד משני הכפתורים:   
       
     זהו בנוסף המצב ההתחלתי של המערכת. בעת לחיצה על "Start (User)", המערכת תיכנס למצב "Play User". בעת לחיצה על "Start (NEAT)", המערכת תיכנס למצב "play NEAT Simulation".
  2. **Play User** – במצב זה המשתמש יכול בעצמו לשחק במשחק עם ציפור אחת על המסך. המשתמש יכול לגרום לציפור לקפוץ על ידי לחיצה על הכפתור "Jump". כאשר יש פסילה (נגיעה באחד מהקוצים), המערכת חוזרת למצב ההתחלתי idle.
  3. **Play NEAT Simulation** – בכניסה הראשונה למצב זה המערכת מיד מתחילה להריץ את דור הציפורים הראשון בו כל המשקולות שברשתות של הציפורים הם רנדומליות לחלוטין. כאשר כולן נפסלות, המערכת עוברת למצב "Play NEAT Population".
  4. **Play NEAT Population** – במצב זה
     + 1. השכבה השנייה היא **PC**. כל מופע מהשכבה הזו מקבל הוראות מהגרפיקה שמעליו על מנת ליצור ולתחזק כמות גדולה של ציפורים. הוראה לדוגמא יכולה להיות simulate\_frame שקוראת כל 80ms, ומשמעותה היא שכל הציפורים שלחו את מיקום ה-Y שלהן וניתן לעבור לבצע את הפריים הבא. מודול זה ממומש לפי behaviour של **gen\_server**.
       2. השכבה השלישית היא **ציפור**. מודול זה ממומש לפי behaviour של **gen\_statem** ופועל לפי ה-FSM הבא:



start

* + - 1. השכבה הרביעית היא **רשת הנוירונים** של הציפור. היא מעין "שרת" שיש לכל ציפור וניתן אף לתאר אותה כ"מוח" של הציפור. בכל פריים חמישי (מדלגים על ארבעה) הציפור שואלת את הרשת האם לקפוץ, וכתלות בקלטים הנוכחיים הרשת תחזיר תשובה אם יש צורך לבצע קפיצה. הרשת מתואר באיור:



גודל השכבות של הרשת שבחרנו לבנות היא [9, 6, 6, 6, 1], כאשר ה-9 הראשון הוא גודל ה-input layer, וה-1 שבסוף הוא ה-output layer. כל רשת נוירונים כזאת יוצרת ומתחזקת נוירונים (מהשכבה הבאה).

* + - 1. השכבה החמישית היא **נוירון** בודד. מודול זה ממומש עם לולאה אינסופית שבה הנוירון מקבל הודעות מהנוירונים שבשכבה שלפניו, מבצע חישובים ולבסוף שולח את התשובה לנוירונים הבאים בשכבה שאחריו.

### הודעות חשובות (מסודרות לפי מבנה המערכת)

|  |  |  |  |
| --- | --- | --- | --- |
| מוצא | יעד | מבנה ההודעה | תיאור |
| גרפיקה | PC | {are\_you\_alive} | הגרפיקה שולחת כל 0.5 שניות ומצפה לתשובה |
| PC | גרפיקה | {im\_alive, PC\_Name} | ה-PC עונה לגרפיקה שהוא בחיים |
| גרפיקה | PC | {start\_bird\_FSM} | הוראה למחשב ליצור תהליכים של ציפורים |
| גרפיקה | PC | {start\_simulation} | התחל סימולציה (העברת הציפורים למצב סימולציה) |
| PC | ציפור | {start\_simulation} | התחל סימולציה (העברת הציפור למצב סימולציה) |
| גרפיקה | PC | {spikes\_list, SpikesList} | עדכון רשימת הקוצים של הציפורים |
| PC | ציפור | {spikes\_list, SpikesList} | עדכון רשימת הקוצים של הציפור |
| גרפיקה | PC | {simulate\_frame} | הודעה שנשלחת כל timer event שאומרת שניתן לקדם את הציפורים בפריים בודד |
| PC | ציפור | {simulate\_frame} | הודעה שנשלחת כל timer event שאומרת שניתן לקדם את הציפור בפריים בודד |
| ציפור | PC | {neat\_bird\_location, Y} | שליחת גובה הציפור אל המחשב |
| PC | גרפיקה | {neat\_bird\_location, Y} | שליחת גובה הציפור אל הגרפיקה |
| ציפור | PC | {bird\_disqualified, BirdPID, FrameCount, WeightsList} | ציפור נפסלה. המחשב מוסיף אותה לרשימה ממוינת לפי כמות הפריימים שהצליחה לשרוד |
| PC | גרפיקה | {pc\_finished\_simulation, CandBirds} | כל הציפורים של המחשב נפסלו והמחשב שולח את הציפורים הכי טובות שלו אל הגרפיקה |
| גרפיקה | PC | {populate\_next\_gen, BestBrains} | שליחה של המוחות (רשימת משקולות) הכי טובים, כדי שכל PC יבצע מוטציות |
| PC | ציפור | {replace\_genes, NewBrain} | המחשב שולח לציפור מוח חדש והיא שולחת אותו אל הרשת נוירונים שלה |
| PC | גרפיקה | {pc\_finished\_population} | המחשב סיים לבצע מוטציות ומוכן להתחיל שוב סימולציה |