

This project must be done individually or in groups of 2 students using Python. The students must deliver a **zip** file with the project source code. The name of the zip file must be composed of the names and student numbers of the group, for example: "123456_JoaoSilva_654321_MariaSantos.zip".

The delivery of the project must be done **via Moodle until 23:59 of 27/05/2025**. Only **one** of the elements of the group needs to submit the project.

Evolving a Feed-Forward Neural Network to Control the Fruit Catcher game.

This project aims to explore the application of artificial neural networks in controlling complex systems by evolving a **feed-forward neural network** to play the *Fruit Catcher* game. The goal of the game is to catch the good fruits while avoiding the bombs (which are disguised as fruits). The objective of your work is to optimize the performance of the neural network using **a genetic algorithm**. Additionally, you should train a **decision tree classifier** to identify which fruits are safe to catch.

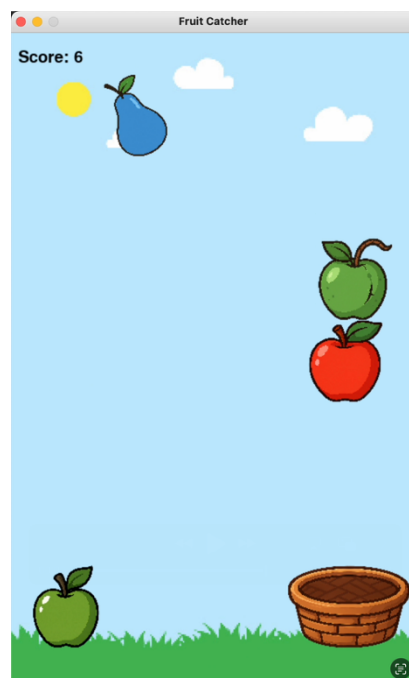


Figure 1 - Fruit Catcher game
















To accomplish this, the project will involve the following steps:

1. Implement and train a decision tree classifier to distinguish between fruits that should be caught and those that should be avoided, based on their features.
2. Design and develop a suitable feed-forward neural network architecture, featuring an input layer to receive the game state and an output layer to generate the player's actions.
3. Optimize the network's weights using a genetic algorithm, implementing key evolutionary mechanisms such as selection, mutation, and crossover
4. Generate and evaluate a population of neural networks by running them in the game and scoring their performance using the fitness function.
5. Apply the evolutionary process over multiple generations to improve the network's gameplay.

This project offers an opportunity to develop practical skills in machine learning and optimization techniques while exploring the potential of artificial neural networks for controlling complex systems.

Decision Tree Classifier for Fruit vs. Non-Fruit Classification

Build a decision tree classifier using a provided dataset containing 15 labeled examples (items.csv). Each example represents either a fruit or a non-fruit (the bombs), characterized by three attributes: **name**, **color**, and **format**. The final objective is to classify an item as being either a fruit (**is_fruit = 1**) or not a fruit (**is_fruit = -1**), based on these features.

ID	name	color	format	is_fruit	
1	apple	red	circle	1	
2	orange	orange	circle	1	
3	blueberry	blue	circle	1	
4	banana	yellow	curved	1	
5	pear	green	oval	-1	
6	orange	blue	circle	-1	
7	apple	blue	circle	-1	
8	banana	red	curved	-1	
9	apple	green	circle	1	
10	banana	blue	curved	-1	
11	pear	blue	oval	1	
12	pear	yellow	oval	1	
13	apple	green	circle	-1	
14	banana	orange	curved	1	
15	apple	yellow	circle	1	

- Attributes:
 - **name**: Name of the object (e.g., "apple", "banana")
 - **color**: Visual appearance (e.g., "red", "blue")
 - **format**: Shape or structure (e.g., "circle", "curved", "oval")
- Target Variable: **is_fruit** — Binary label indicating whether the object is a fruit (1) or not (-1)

Objectives:

1. Train a decision tree classifier using the first 12 examples (train.csv).
2. Build the model using information gain to choose attribute splits.
3. Test the accuracy of the model using the last 3 examples (test.csv).
4. The classifier will be used by the AI player to identify the fruits and the bombs.

Neural Network evolution with a Genetic Algorithm

As stated above, you should implement a genetic algorithm to **optimize the network's weights** and to do so, you should base yourself on the content learnt during this semester. Feel free to be creative and explore ways to improve your default genetic algorithm, test different approaches and compare their results. However, for your own time's sake, please have this in mind:

You should **only calculate the fitness of each individual once** and only once.

Game Mechanics Summary

- A basket moves horizontally at the bottom of the screen.
- Fruits (and sometimes bombs) fall from the top of the screen.
- The player controls the basket by moving it **left (-1)** or **right (+1)** each time step.
- The game ends if a bomb hits the basket or after a fixed number of time steps.

State Representation

At each time step, the environment provides a state vector composed of 10 numerical values, representing:

1. **basket_y**: The y-coordinate (vertical position) of the basket.
2. For each of the 3 nearest fruits/bombs :
 - **fruit_x**: x-coordinate (horizontal position)
 - **fruit_y**: y-coordinate (vertical position)
 - **is_fruit**: binary flag indicating whether this object is a bomb (-1) or a fruit (1)

This results in a total of:

- 1 value for basket position
- $3 \times 3 = 9$ values for the 3 nearest fruits

Total state size: 10 numbers

Action Space

The agent has only two possible actions at each time step:

- Move left: **-1**
- Move right: **+1**

This defines a discrete action space with two options.

Start by implementing a perceptron model (only one neuron) that maps the 10-dimensional input state to a single output (action). Then use a genetic algorithm to evolve its weights.

After successfully evolving a single perceptron, evolve a more complex controller using a feedforward neural network with one hidden layer.

Project structure

You are given a modular Python project that allows you to:

- Train a decision tree classifier;
- Implement and evolve neural network controllers;
- Use them to play a simple fruit-catching game.

Note: The project requires the *pygame* and *numpy* libraries.

The codebase consists of the following files:

- **main.py**: Main entry point; runs training or plays the game;
- **game.py**: Game engine implementation;
- **dt.py**: Decision Tree learning module;
- **nn.py**: Neural Network architecture;
- **genetic.py**: Genetic Algorithm to evolve neural networks.

main.py

This file serves as the main interface between the user and the rest of the system. **You do not need to modify it.** It handles command-line arguments, starts training, loads AI players, and launches the game.

- Trains a neural network using a genetic algorithm (**train_ai_player**);
- Loads pre-trained weights from a file (**load_ai_player**);
- Trains a decision tree classifier based on a dataset in CSV format (**train_fruit_classifier**);
- Starts the game loop with either AI or human player.

dt.py

This file contains a basic skeleton of a decision tree classifier. You'll use this to classify fruits vs bombs based on the features provided in **train.csv**.

- Implement the **DecisionTree** constructor (**__init__** method) to build the tree recursively using information gain index;
- Implement **DecisionTree.predict()** to classify a new item;
- Make sure it works with categorical data (e.g., names, colors);
- Modify the **train_decision_tree()** function according to your selected parameterization.

genetic.py

This file implements a genetic algorithm to evolve the weights of a neural network.

- Complete the **genetic_algorithm()** function:
 - Evaluate the fitness of each individual using the fitness function provided as argument; **Note:** In addition to the individual, the fitness function may receive an additional integer argument that sets the seed of the random generator. This can be used to ensure that the individuals play the same game.
 - Use elitism to select and preserve top performers;
 - Perform crossover and mutation;
 - Replace worst individuals;
 - Stop when target fitness is reached or max generations exceeded;
 - Return a pair (best individual, fitness).

nn.py

This file defines a flexible neural network class that can represent both a perceptron and a multi-layer feedforward network.

- Implement **compute_num_weights()** to calculate how many weights are needed;
- Implement **forward()** to perform a forward pass through the network;
- Modify **create_network_architecture()** to define:
 - A simple perceptron first (just one neuron);
 - Then try a feedforward network with one hidden layer.

game.py

This file contains the full implementation of the game environment. **You do not need to modify it** nor to access its functions directly as that is handled by the main program.

How to run the game

To run the game in graphical mode:

```
python main.py [--file FILE]
```

To train the AI player:

```
python main.py --train [--population POP --generations GEN --file FILE]
```

To play the game with the AI player without graphics:

```
python main.py --headless [--file FILE]
```

The default values of the parameters are:

- **--file:** best_individual.txt
- **--population:** 100
- **--generations:** 100

Deliverables

At the end of the project, you should submit:

1. All Modified Code Files

Ensure your submission includes:

- **dt.py** – Your decision tree implementation;
- **nn.py** – Your neural network controller;
- **genetic.py** – Your genetic algorithm;
- Any additional helper functions or classes you created.

2. Best Neural Network Configuration

Save the best evolved neural network configuration in a file named: **best_individual.txt**

This file should contain a comma-separated list of weights that represent the best-performing individual found during evolution. (Use the functionality provided by the **main** program)

Example: 0.56,-0.34,1.23,... # etc.

Evaluation criteria

- | | |
|--|-----|
| • Correct implementation of Decision Tree | 20% |
| • Correct implementation of Neural Network | 25% |
| • Correct implementation of Genetic Algorithm | 30% |
| • Performance of Evolved AI Player (Score in Game) | 15% |
| • Creativity | 10% |