**Computational Intelligence 410**

**Mario Nakazawa**

Genetic Algorithm

CATCHING PANCAKES

**\_\_\_**

By David Newswanger and Robert Hosking



# INTRODUCTION

For this project we will create a game for our genetic algorithm to master. Our game is called “Pancakes!”. In Pancakes!, there is a frying pan that the player is able to move either left or right. There are also hundreds of “pancakes” falling from the sky. The goal is to catch as many pancakes in the frying pan as possible.

The overall goal of the project is to have the genetic algorithm beat the human player at this game.

The game has certain restrictions that allow the genetic algorithm (GA) to easily adapt to this game. These restrictions should have minimal effect on the human’s ability to play the game but will be noticeable.

* The pancakes will fall in the same order on each run of the game. This ensures that the GA has a constant target to compare against on each individual.
* The game’s actions will be limited to tick-tock events. This means that there will be a presiding clock such that everything in the game can only move or not move on a clock tick. (i.e. Every tick the user may move the frying pan left, right, or not at all. At the same time, on every tick the pancakes will fall one space closer to the frying pan.)

# Analysis

Upon further analysis, this problem can be represented as a graph and the correct path that the frying pan should follow can be calculated using a directed graph and adjacency matrix.

### Matrix

If we imagine the arena of the game at any particular tick, we can represent the arena as a binary matrix.

[ 0 1 0 ]

[ 1 0 0 ]

[ 0 0 1 ]

[ 1 1 1 ]

In this simple matrix, 1s represent the location of a pancake and 0s represent empty space. The challenge in playing the game effectively is finding the series of columns in the last row of the matrix that can intercept the greatest percentage of 1s in the matrix.

The major calculation is to determine the column position of the last row of the matrix for each tick such that the sum of the values of that position in the matrix is maximized.

### Graph

A graph of possible moves can be created using a weighted digraph using nodes to represent pancakes and weighted edges to represent moves (-1 = move left, 1 = move right, 0 = stay still) to take to reach the next possible pancake.

### Adjacency Matrix

To find the path that the frying pan should move to catch the most pancakes can be found by analyzing an adjacency matrix based on the graph mentioned above.

### Complexity

We’ve determined that the mathematical approach to solving this problem is where is the number of ticks of the game. This calculation was achieved because the size of the adjacency matrix to be analysed is exponentially growing with each new tick. (i.e. For every tick of the game, a new layer of falling pancakes is added. This adds an extra row and an extra column to our adjacency matrix.)

Therefore, a genetic algorithm will provide a more efficient adequate solution. Since the goal is to defeat a human at catching the most pancakes, not to catch the absolute optimal number of pancakes, there is room for error and the solution does not have to be perfect.

# Implementation

Our implementation will consist of four major components:

* Genome Representation
* Corrective Adjustments
* Fitness Calculation
* Crossover Impact

### Genome Representation

Our genome will be a single dimensional array that catalogues the series of moves that the frying pan makes.

-1 = move left

1 = move right

0 = stay still

Since the frying pan can only make one move per tick, this representation is better for ensuring that every move the frying pan makes is “legal”, as opposed to say, a coordinate system.

This leads us to the next major component.

### Corrective Adjustments

Since the arena is a fixed space, we must make for certain that none of the genomes cause the frying pan to escape the arena’s dimensions. This correction can be done effectively in two ways:

#### Destroy invalid individuals

Perhaps the easiest approach involves summing the entire array of moves and determining based on the individual’s starting position if that individual violates the width of the arena.

Let's call the width of the arena

and the starting position of the individual such that .

If we sum the contents one by one of the array that represents the left, right, and null moves of the frying pan and call that sum total ,

then if at any point in the summation process, then the individual violates the parameters of the arena and would be instantly eliminated.

This option is easiest but may lead to the destruction of very otherwise fit individuals. Crossover will be especially effective at causing individuals to be destroyed for this reason.

#### Adjust invalid individuals

Using the same calculation as mentioned above we determine if an individual violates the area parameters. When an individual is found to violate these parameters, the gene that caused this break is modified so as not to break it.

For example, if the frying pan is at the edge of the arena perimeter on the gene in its genome and the gene makes the frying pan go outside the arena space, then the gene is altered to any of the other two possible states: either 0 or -1.

This method introduces another form of mutation to the individuals. A move that perhaps proved beneficial in one individual may be over-written to adjust it to the arena space.

Each of theses methods have their own strengths and weaknesses and will likely both be implemented in development.

### Fitness Calculation

Both the path that the frying pan takes and the game layout can be can be represented by a 2D array.

For example,

The genome [-1, 0, 1, 1 -1, 0 ] look would be formed into the 2D array:

|  |  |
| --- | --- |
| Path:  [ 0 1 0 0 ] <- FINISH  [ 0 1 0 0 ]  [ 0 0 1 0 ]  [ 0 1 0 0 ]  [ 1 0 0 0 ]  [ 1 0 0 0 ]  [ 0 1 0 0 ] <- START | Game Layout:  [ 0 1 0 0 ] <- FINISH  [ 0 1 1 1 ]  [ 1 0 0 1 ]  [ 0 1 1 0 ]  [ 1 0 0 0 ]  [ 0 0 1 1 ]  [ 0 0 0 0 ] <- START |

In this case, this particular genome would have a fitness of 4/7, because the path that the pan takes intersects with 4 pancakes, divided by the total number of moves, 7.

Thinking of the path that the pan takes as a 2D array is useful because it allows for us to visualize exactly how the pan moves and see how many pancakes it catches. In reality we plan to implement this fitness function as a single loop which iterates over each row in the game world. For each row, we can calculate the position that the pan will be at given the history of moves that it has made. If the current location happens to be a pancake, then we add one the number of pancakes caught. After this loop finishes, the number of pancakes caught is divided by the number of potential pancakes caught, which is the same as the number of rows in the game world.

### Crossover Impact

Crossover has the potential to be extremely disruptive. Besides making previous “legal” individual's “illegal” (see “Corrective Adjustments” above), it also displaces patterns of moves that may have been effective at one section of the pancake avalanche and not so effective at another. Although the opposite is also true. Crossover can also cause a section of the genome that was useless and make it highly effective.

Since all of our moves are relative, a single change in an early gene can completely change the outcome of the rest of the game. If the first gene, for instance, is changed from a -1 to a 1, then that shifts all of the positions right by two, creating an entirely new path. Because of this, we need to weight our crossover and mutation so that they predominantly happen higher up in the genome, and are therefore less disruptive to the sequence as a whole.

### Terminating Condition

We don’t know exactly what the maximum fitness is for a given game world. The maximum potential fitness is 1, however it’s not always possible to achieve this. For example, if our game world has one or more rows where there simply aren’t any pancakes, the it is impossible to get a max score of 1. Similarly, if there are two rows where the pancakes on each row are more than one space apart, then it is also impossible to get a maximum score of 1. Because of this, it’s safe for us to stop if an individual reaches a score of 1, however we can’t rely on that for all game worlds.

However, if the algorithm is able to catch one pancake on every tick of the game, then it has reached maximum possible efficiency. This may become the basis of calculating fitness which is subject to change.

Because of this we need to terminate our program when it gets to a point when the average fitness isn’t reliably increasing with each successive generation