FlapANN

Dawid Grobert
Julia Boczkowska

Silesian University of Technology

dawigro122@student.polsl.pl
juliboc987@student.polsl.pl

May 10, 2022

Overview

What is our project about

FlapANN - Artificial inteligence plays flappy bird game

Flappy bird is a so-called side-scroller in which the player takes the role of a flying bird. The main objective is to avoid the pipes sticking out from the bottom and top of the screen by making the bird jump. In our case, the role of the player is played by artificial intelligence. AI gets an identical interface to the user – may jump or not.

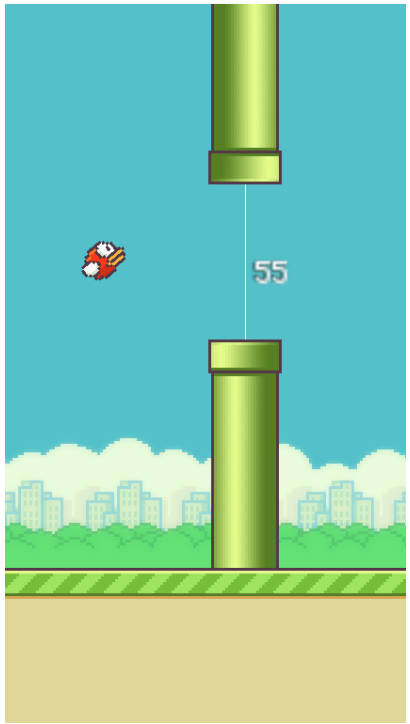

Figure: Screenshot from our game

Mechanics

In each frame, the AI can:

Mechanics

In each frame, the AI can:

- **jump** - which causes the bird to rise

Mechanics

In each frame, the AI can:

- **jump** - which causes the bird to rise
- **do nothing** - gravity will act on the bird and pull it down

Mechanics

In each frame, the AI can:

- **jump** - which causes the bird to rise
- **do nothing** - gravity will act on the bird and pull it down

The bird dies if it touches any part of the pipe. AI in learning is as lazy as students, so in addition to wanting to encourage them to develop properly we started to consider going beyond the top area of the game as death. Touching the ground is also naturally death.

Funfact Early versions of AI notoriously stuck to the top of the screen

Demo of the game

Before getting into specifics, let's see how the AI handles the game in this short video prepared by Dawid.

Figure: Click or scan for a video

Used libraries

The libraries we used in the project are as follows:

- SFML – for creating a window and displaying the game

Used libraries

The libraries we used in the project are as follows:

- SFML – for creating a window and displaying the game
- ImGui – to display "debug" windows inside the game containing any kind of checkboxes or sliders.

Used libraries

The libraries we used in the project are as follows:

- SFML – for creating a window and displaying the game
- ImGui – to display "debug" windows inside the game containing any kind of checkboxes or sliders.
- FANN - Fast Artificial Neural Network Library which implements artificial neural networks in C

Used libraries

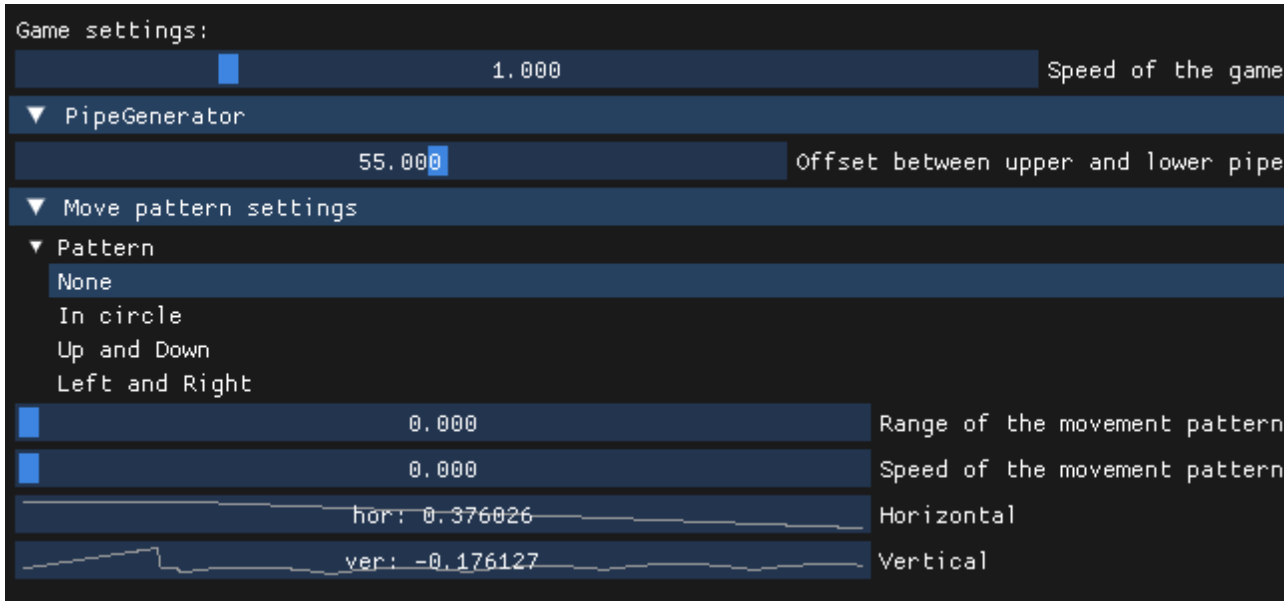

Figure: ImGui debug window

Artificial Neural Network

In this project, we used an evolutionary algorithm, namely genetic algorithm (GA), to train artificial neural networks (ANNs). Each bird has its own neural network used as its AI brain for playing the game. Our ANN consists of an input layer with three neurons:

- horizontal distance of the bird to the closest pipe's gap
- Y position of the bird
- height difference between bird and the closest gap

We also use some hidden layer and an output layer that indicates whether to make a jump or not.

Genetic algorithm

The **FANN** library provided the neural network, but writing the training part of the algorithm was done by us. Our implementation of GA is as follows:

Genetic algorithm

The **FANN** library provided the neural network, but writing the training part of the algorithm was done by us. Our implementation of GA is as follows:

1. We create a population of 100 birds, each with its own neural network filled with random weights and connections.

Genetic algorithm

The **FANN** library provided the neural network, but writing the training part of the algorithm was done by us. Our implementation of GA is as follows:

1. We create a population of 100 birds, each with its own neural network filled with random weights and connections.
2. As long as the birds are alive, we let them play the game. We constantly feed them with inputs that are then used to determine the output - jump or no jump

Genetic algorithm

The **FANN** library provided the neural network, but writing the training part of the algorithm was done by us. Our implementation of GA is as follows:

1. We create a population of 100 birds, each with its own neural network filled with random weights and connections.
2. As long as the birds are alive, we let them play the game. We constantly feed them with inputs that are then used to determine the output - jump or no jump
3. When all birds are dead, we evolve the current population. We do this by selecting the best individuals and mixing their genes. The **crossover** process will be described in more details on the next slide.

Genetic algorithm

The **FANN** library provided the neural network, but writing the training part of the algorithm was done by us. Our implementation of GA is as follows:

1. We create a population of 100 birds, each with its own neural network filled with random weights and connections.
2. As long as the birds are alive, we let them play the game. We constantly feed them with inputs that are then used to determine the output - jump or no jump
3. When all birds are dead, we evolve the current population. We do this by selecting the best individuals and mixing their genes. The **crossover** process will be described in more details on the next slide.
4. Go back to step two and repeat until you reach maximum fitness

Fitness function

Our fitness function is very simple, birds are scored for the distance they have reached. The birds were sorted just based on this value and the best individuals were selected for crossover.

Info: Although this was the initial intention, for the alpha version we improved the fitness by extra points when the bird died closer to the gap between pipes. In this way, we tried to direct and encourage them to pass through.

Crossover

For an algorithm to work, only the genes of the fittest individuals must survive. For this reason, in our code we use a crossover function. The flow of the function is as follows:

Crossover

For an algorithm to work, only the genes of the fittest individuals must survive. For this reason, in our code we use a crossover function. The flow of the function is as follows:

1. Sort the population by fitness

Crossover

For an algorithm to work, only the genes of the fittest individuals must survive. For this reason, in our code we use a crossover function. The flow of the function is as follows:

1. Sort the population by fitness
2. Take X number of top units and preserve their genes to next generation

Crossover

For an algorithm to work, only the genes of the fittest individuals must survive. For this reason, in our code we use a crossover function. The flow of the function is as follows:

1. Sort the population by fitness
2. Take X number of top units and preserve their genes to next generation
3. Replace the remaining population with new offspring created in different ways

Crossover

For an algorithm to work, only the genes of the fittest individuals must survive. For this reason, in our code we use a crossover function. The flow of the function is as follows:

1. Sort the population by fitness
2. Take X number of top units and preserve their genes to next generation
3. Replace the remaining population with new offspring created in different ways
   - As crossover product of two random top units

Crossover

For an algorithm to work, only the genes of the fittest individuals must survive. For this reason, in our code we use a crossover function. The flow of the function is as follows:

1. Sort the population by fitness
2. Take X number of top units and preserve their genes to next generation
3. Replace the remaining population with new offspring created in different ways
   - As crossover product of two random top units
   - As crossover products of two best units

Crossover

For an algorithm to work, only the genes of the fittest individuals must survive. For this reason, in our code we use a crossover function. The flow of the function is as follows:

1. Sort the population by fitness
2. Take X number of top units and preserve their genes to next generation
3. Replace the remaining population with new offspring created in different ways
   - As crossover product of two random top units
   - As crossover products of two best units
   - By taking whole genes of some random top unit

Crossover

For an algorithm to work, only the genes of the fittest individuals must survive. For this reason, in our code we use a crossover function. The flow of the function is as follows:

1. Sort the population by fitness
2. Take X number of top units and preserve their genes to next generation
3. Replace the remaining population with new offspring created in different ways
   - As crossover product of two random top units
   - As crossover products of two best units
   - By taking whole genes of some random top unit
4. At the end, connections and neurons of each of the offspring are randomly selected and mutated to random value.

The end

This is the end