Artificial Intelligence 2023/24

Version 1.0

This project must be done in groups of 2 students using Java. The students must deliver a **zip** file with the project extraction, extracted directly from the Eclipse environment (or other IDE), and the report in a PDF format. 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 28/04/2024

Evolving a Feed-Forward Neural Network to Control Two Classic Arcade Games.

This project aims to explore the application of artificial neural networks in controlling complex systems by evolving a feed-forward neural network to play simplified versions of the classic arcade games Breakout and Pacman. The objective is to optimize the network's performance for each game using a genetic algorithm.
Begin by addressing the Breakout game, and only after achieving success with it, proceed to tackle Pacman.

Figura 1- Jogo do Breakout

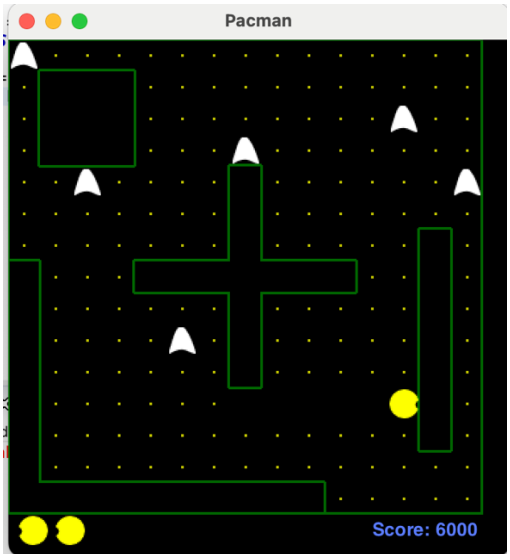

Figura 2 - Jogo do Pacman

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

1. Design and develop a suitable **feed-forward neural network** architecture with an input layer to receive the game state information and an output layer to produce the player's actions.
2. Use a **genetic algorithm to optimize** the network's weights by implementing the evolution process, including selection, mutation, and crossover.
3. Define a proper fitness function that evaluates the network's performance in the game.
4. Generate a population of neural networks and evaluate their performance in the game using the fitness function.
5. Apply the evolutionary process over several generations to improve the network's performance.
6. Summarize the project's results in a **short report** (max three pages), including the design and implementation of the neural network and genetic algorithm, the evolved network's performance, and the network's potential applications for controlling other games and real-world systems.

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.
To interact with both games, an API is provided that includes the necessary methods. Both games use an instance of a GameController to control the game.

```
//The GameController interface defines a method that allows a client to
obtain the next move from the game controller.

public interface GameController {

      /*
       * This method takes in an array of integer values as input and
       * returns an integer value as output. The input array
       * represents the current state of the game, and the output value
       * represents the next move to be made by the player. The exact
       * format and meaning of these values will depend on the specific
       * game being played.
       */
      public int nextMove(int[] currentState);

}
```

In the Breakout game:

The BreakoutBoard class constructor creates an instance of the BreakoutBoard class with a game controller.

breakout.BreakoutBoard.BreakoutBoard(GameController controller, boolean withGui, int seed)

Creates a new BreakoutBoard with a controller. If the boolean parameter withGui is set to false, indicating the absence of a graphical user interface, the program operates in headless mode; otherwise, it initiates with a graphical user interface.

Parameters:

controller controller used to get next action of the player
withGui run with GUI
seed the initial seed

The **setSeed** method sets the seed value for the random number generator. The **getState** method creates a normalized state representation of the game board that can be used as input to the controller, such as the neural network. The **run** method evaluates the fitness of a controller and is the main game loop of the BreakoutBoard class. The **getFitness** method calculates and returns the fitness score of the game.

The controller should be able to receive a state represented by a vector of doubles with size `Commons.BREAKOUT_STATE_SIZE` and return the action to be performed, represented as a vector of doubles with size `Commons.BREAKOUT_NUM_ACTIONS`. If the controller is an artificial neural network, it should have `Commons.BREAKOUT_STATE_SIZE` neurons in the input layer and `Commons.BREAKOUT_NUM_ACTIONS` neurons in the output layer.
To evaluate the behavior of a specific neural network (nn) that implements the GameController interface, the following code could be used:

```
BreakoutBoard b = new BreakoutBoard(nn, false, seed);
b.setSeed(i);
b.runSimulation();
fitness = b.getFitness();
```

To visualize the behavior of a specific neural network using the graphical interface, the following code could be used:

```
new Breakout(network, seed);
```

It is important to note that the runSimulation method runs headless, without opening the graphical interface, allowing a faster evaluation of the controllers.

The game state should indicate the position of the board, the position and direction of the ball and the location of the block. In this game, there are only two possible actions, which are to move the board to the right or to the left.

In the Pacman game:

For the Pacman game, the size of the game state and the number of available actions are also defined within the Commons interface, with the identifiers: `PACMAN_NUM_ACTIONS` and `PACMAN_STATE_SIZE`, respectively. Following similar principles, the approach to implementing the game controller for Pacman should follow the same principles.

In this game, the state is more complex because it needs to represent the positions of the pacman, the monsters, the available dots and the walls. There are five possible actions, which are stay still, move the pacman to the right, left, up or down.