TP C#14: Genetic evolution v1.7

Archive

You must submit a zip with the following architecture:

```
rendu-tp-prenom.nom.zip
|-- firstname.lastname/
    |-- AUTHORS
    |-- README
    |-- Genetics/
        |-- Genetics.sln
        |-- packages
        |-- Genetics/
            |-- Program.cs
            |-- Matrix.cs
            |-- Factory.cs
            |-- ACDC/
            |-- Tests/
            |-- map/
            |-- img/
            |-- Properties/
            |-- Genetics.csproj
            |-- save/
                |-- bot.save
```

- You shall obviously replace *firstname.lastname* with your login (the format of which usually is *firstname.lastname*).
- The code **MUST** compile.
- In this practical, you are allowed to implement methods other than those specified. They will be considered bonuses. However, you must keep the archive's size as small as possible.

AUTHORS

This file must contain the following : a star (*), a space, your login and a newline.
Here is an example (where `$` represents the newline and ␣ a blank space):

```
*␣firstname.lastname$
```

Please note that the filename is AUTHORS with **NO** extension. To create your `AUTHORS` file simply, you can type the following command in a terminal:

```
echo "* firstname.lastname" > AUTHORS
```

README

In this file, you can write any and all comments you might have about the practical, your work, or more generally about your strengths and weaknesses. You must list and explain all the bonuses you have implemented. An empty `README` file will be considered as an invalid archive (malus).

Contents

1 Introduction

1.1 Objectives

This TP will show you how a genetic algorithm works, then you will make one and use it.

Genetic Algorithm

In computer science and operations research, a genetic algorithm (GA) is a metaheuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA). Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on bio-inspired operators such as mutation, crossover and selection.

(https://en.wikipedia.org/wiki/Genetic_algorithm)

As stated in the previous definition, the goal of a genetic algorithm is to give an approximate solution to a problem which seems too complex. In this case, you'll have to solve a 2D Platform Game which we created.

2 Course part 1

Genetic algorithms are based on natural selection, you'll have to create populations of players, test the players one at a time and to assign each one of them a score. Then you will only keep the best players and start again.

2.1 Step by step

2.1.1 Generation 0

To begin with, you'll create the first generation of players:

Player	score
Smlep	0
Bjorn	0
Heldhy	0
PetitPrince	0
Tetra	0
Jeanne	0
Paul	0
Tom	0
Bot1	0
Bot2	0
Aelita	0
Yumi	0
Ulrich	0
Jeremie	0

2.1.2 Score assignment

Then evaluate the players on a level, to assign a score to each one of them.

Player	score
Smlep	290
Bjorn	270
Heldhy	730
PetitPrince	20
Tetra	0
Jeanne	50
Paul	70
Tom	500
Bot1	7000
Bot2	300
Aelita	30
Yumi	40
Ulrich	70
Jeremie	900

2.1.3 Players sorting

Then sort this list, and remove the weakest elements.

Player	score
Bot1	7000
Jeremie	900
Heldhy	730
Tom	500
Bot2	300
Smlep	290
Bjorn	270
Paul	70
Jeanne	50
Ulrich	70
Yumi	40
Aelita	30
PetitPrince	20
Tetra	0

2.1.4 Evolution !

There are two possibilities to make this population evolve:
- Randomly generate new players. (First case)
- Make a copy of the best half and apply modification to these players. (Second Case)

These newly generated players will replace the previously deleted half.

First case

Player	score
Bot1	7000
Jeremie	900
Heldhy	730
Tom	500
Bot2	300
Smlep	290
Bjorn	270
new Player 1	0
new Player 2	0
new Player 3	0
Alain Connu	0
Sonic	0
Mario	0
Luigi	0

Second Case

Player	score
Bot1	7000
Jeremie	900
Heldhy	730
Tom	500
Bot2	300
Smlep	290
Bjorn	270
Bot1 v2	0
Jeremie v2	0
Heldhy v2	0
Tom v2	0
Bot2 v2	0
Smlep v2	0
Bjorn v2	0

2.1.5 Endless evolution?

Then, just repeat the previous steps since scores assignment.
It will then be your job to estimate how to train your players.
Indeed, depending on how many generations you have and on the maps, you will use to train your player, huge changes can happen.
Don't forget, your AI will have to be able to adapt to different environments!

We strongly recommend that you create your own maps to train your players. (See Section 3.3 level creation)

2.2 A few videos to help you understand

These videos are a good illustration of genetic algorithm's principles (in four parts):
https://www.youtube.com/watch?v=GOFws_hhZs8
https://www.youtube.com/watch?v=31dsH2Fs1IQ
https://www.youtube.com/watch?v=IVcvvqxtNwE
https://www.youtube.com/watch?v=KrTbJUJsDSw

3 General aspects

3.1 For Windows

If the version you have does not work (on the graphical part) on Windows, you'll have to add a NuGet package.
Tools > NuGet > Manage NuGet Package for ...
You will need the NuGet package "MonoGame.Framework.WindowsDX version 3.4.0.459"
(https://github.com/MonoGame/MonoGame/issues/5531#issuecomment-326928360)

3.2 The viewer

In this TP, the graphical interface is optional, since you can see the scores without it.
To turn it on, you have to use one of the functions below.

```
1 static void Showbest() // Show the best player
2 static void ShowNth(int nth) /* Show the nth player (on 200 players,
3 sorted by scores in ascending order) */
4 static void PlayAsHuman() // Allow you to play instead of the AI
```

3.3 Level creation

To create your own level (map), go into the "map" directory, and create a file "*.map" (the star "*" means any kind of string).
You have to write on the first line the maximum number of frames which the player will have to try, to complete the level. Then, you will use the following characters:
"W": a wall
"S": starting point
" ": empty cell
to build your own map.

The only constraint is to create a rectangle with only these characters.
Our parser reads all files in the "map" directory and throws an exception if one of the previous rules is not respected.

We strongly recommend that you use the already created maps as examples.
You can access the maps with:

```
1 RessourceLoad.SetCurrentMap("FileNameWithoutExtension");
```

3.4 Evaluation of your best player's performance

To evaluate the performance of your best player, you will have to submit a file "bot.save" which will be located in the "save" directory (see "Archive").
You will use the function:

```
1 static void SaveBest()
```

3.5 Saving scores

You are allowed to change this line in Program.cs:

```
1 private const string PathForTest = "test.save";
```

If you want to have multiple saving files.

However, we chose to leave the .save training files in the bin folder so they won't be here in your submission.
You'll have a penalty if you leave the .save training files in your submission!
The current saving directory is the bin directory, which means that if you don't try to go up (using ".."), there should not be any problem.
Here are valid examples:

```
private const string PathForTest = "test1.save";
private const string PathForTest = "bestTeam.save";
private const string PathForTest = "toutReprogrammer.save";
private const string PathForTest = "pourUnMondeSansDanger.save";
```

You must not change the submit path!

```
private const string PathBotToSubmit = "../../save/bot.save";
```

You will be evaluated on the presence and the quality of this file.

3.6 The player

The player is based on multiple methods and attributes.

3.6.1 His current position

The abscissa of this position will be used to calculate the score.

3.6.2 3 matrices

The player has three matrices which represent his brain, they can be accessed with the method:

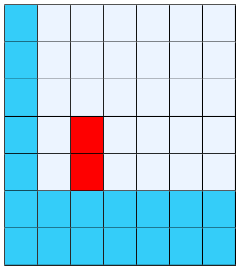
```
public List<Matrix> Getbrains()
```

You have to use these matrices to determine the player's moves in `Matrix.cs`
When the player will receive an order with the method :

```
public void ReceiveOrder(bool left, bool right, bool up, bool reset)
```

he will look at the surrounding cells and decide what to do.

Example: (Red: the player, Blue : the walls, Light Grey : the air)



He will receive a float matrix F such as:

0.5	0	0	0	0	0	0
0.5	0	0	0	0	0	0
0.5	0	0	0	0	0	0
0.5	0	0	0	0	0	0
0.5	0	0	0	0	0	0
0.5	0.5	0.5	0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5	0.5	0.5	0.5

B_1 B_2 B_3 respectively of dimensions 49x16, 16x16 and 16x4; the three matrices held by the player.

The player will need the matrix S of dimension 4x1 to know what decision he has to make. Saying that S is a list named S, we can then use it to define the player's actions such as:

```
1  if(S[0] > 0.5)
2      go("left");
3  if(S[1] > 0.5)
4      go("right");
5  if(S[2] > 0.5)
6      go("up");
7  if(S[3] > 0.5)
8      reset();
```

To calculate S apply the formula:
$S = F * B_1 * B_2 * B_3$
with the operator "*" slightly different from the classic matrices multiplication. (It will be explained in the matrices part).

Note
Let x be the element from on of the previous matrices "M". $\forall x \in M, x \in [0, 1] \cap \mathbb{R}$

3.6.3 His speed

When the player jumps, his speed is half his normal speed. This avoids useless jumps during training (and adds realism).

3.6.4 His score

The score tells how far the player went in a level, and allows you to compare this player with the others.
It is calculated from the player's abcissa, once the number of actions done by the player is equal to the maximum number of frames allowed by the map.

3.6.5 Physical forces

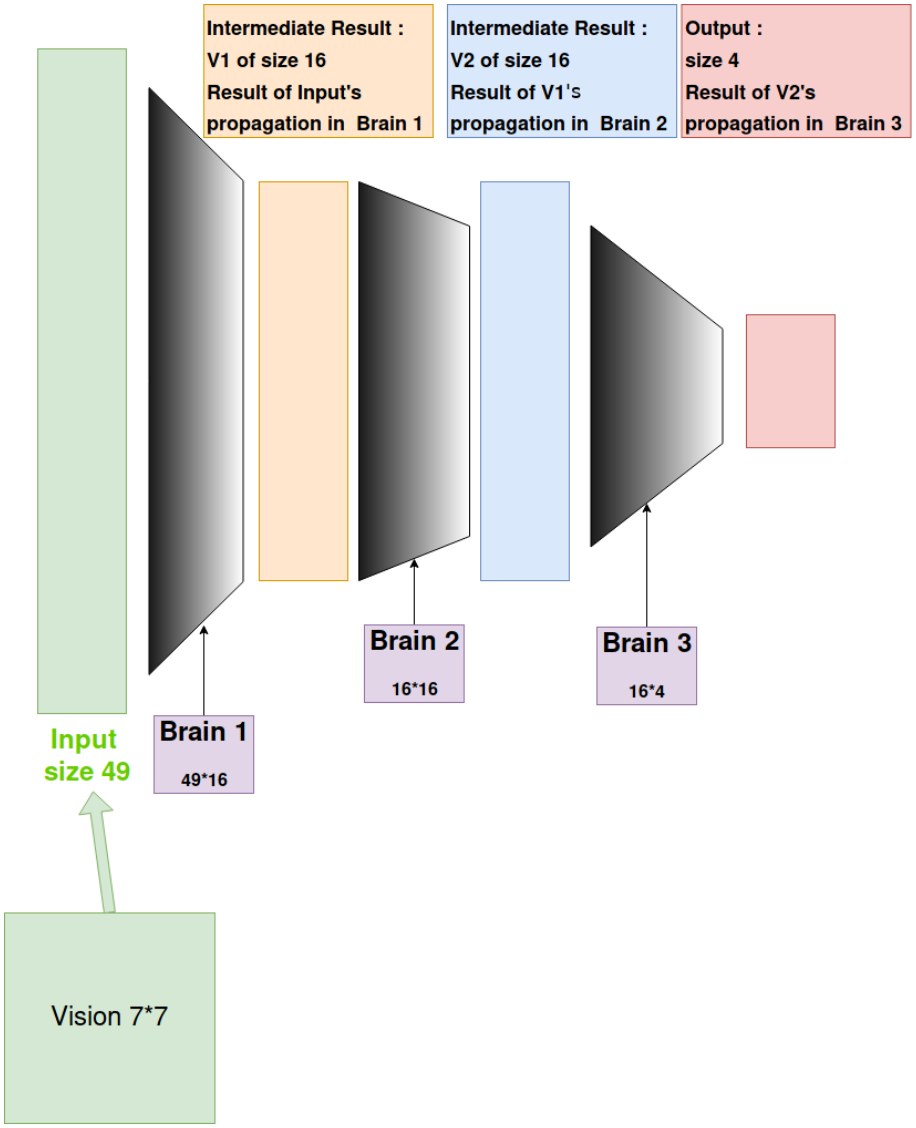
Nothing will be asked from you in this TP about them, but theses forces are the reasons for the gravity and the reason the player is falling after a jump.

4 Course part 2

4.1 The propagation in itself

The main idea of propagation is to make a relation between input's information and matrices in order to create new information.
First your input is the vision of the player. It goes through the first brain (that is a matrix) in order to produce a new information that is V_1. V_1 goes through the second brain (that is also a matrix) in order to produce a new information that is V_2. The same process is done for V_2 through the third brain, that gives us the output. This output is the information the player really needs to decide what to do next.

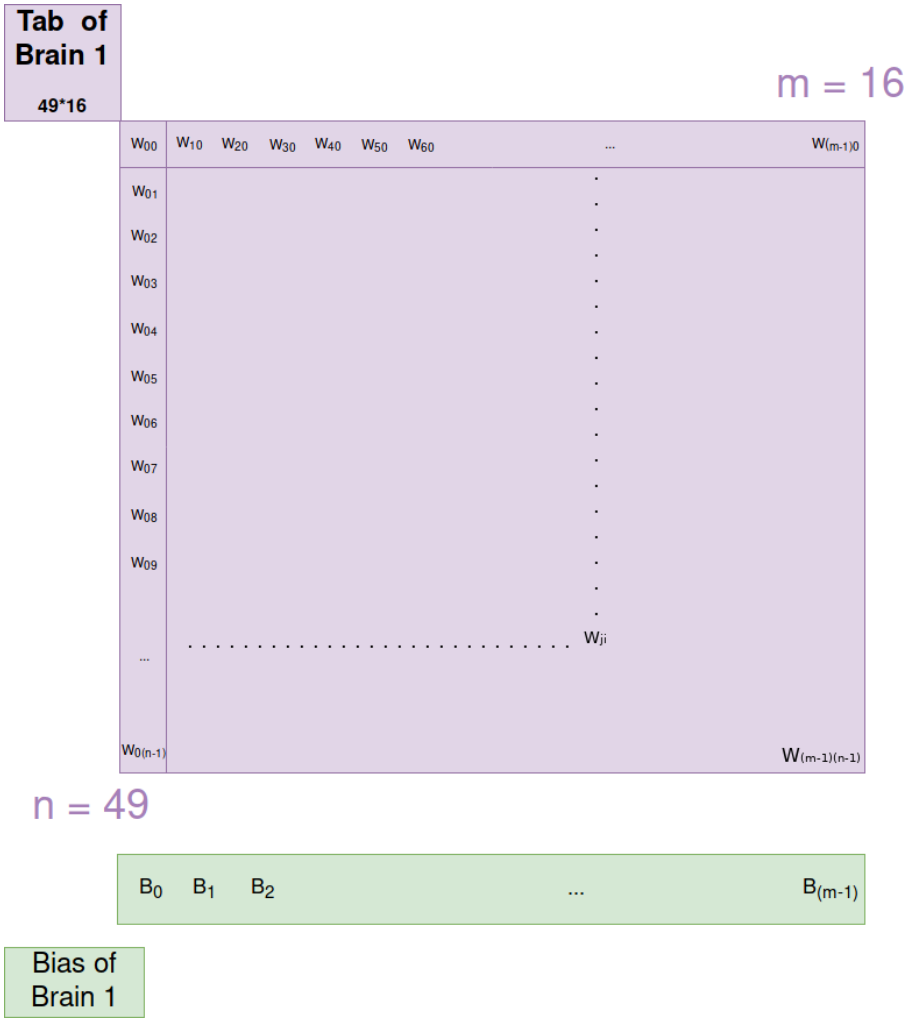
In our case the output is a matrix of size $1 * 4$. For any value of the matrix that is superior to 0.5, the action to this element is activated.
i.e. :
Let's have a matrix M composed of $[0.45; 0.55; 0.56; 0.48]$ that is the result of what we computed previously.
The Output is linked to the following actions [Go left ; Go right ; Go up ; Reset]
So the player will only "Go right" and "Go up".



4.2 Formula for propagation detailed

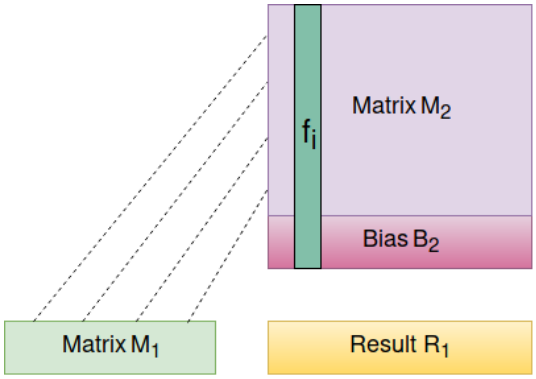
The Brain is composed of a two dimensional Float array and a one dimensional array for the Bias. Let's call V_1 the resulting matrix after the propagation of the input through it.
In order to compute the propagation, you have to apply for each of the element of V_1 the following formula :

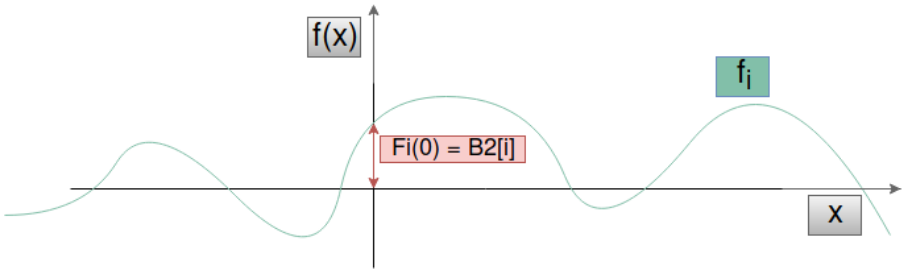
$$V_1[j] = Sigmoid(Bias[j] + \frac{\sum_{i=0}^{n-1} Input[i] * \omega_{ji}}{n})$$



4.3 Purpose of the Bias

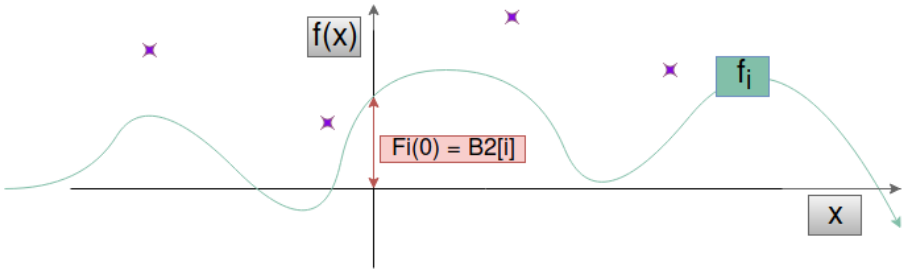
Hereunder, you can see a graphical way to simply propagate.
As said before the Bias, as a list of missing constants for each polynomial that compose the matrix.
Here you can see precisely why :
If all the ω's are 0, there is still the Bias. Formally it is : $F_i(0) = B_2[i]$.



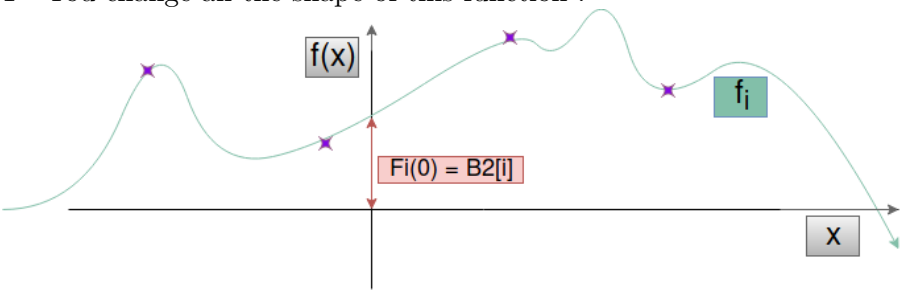


Why Using a Bias?
Simply because your function may have a good shape, but a wrong constant that will translate itself vertically in an unwanted way.

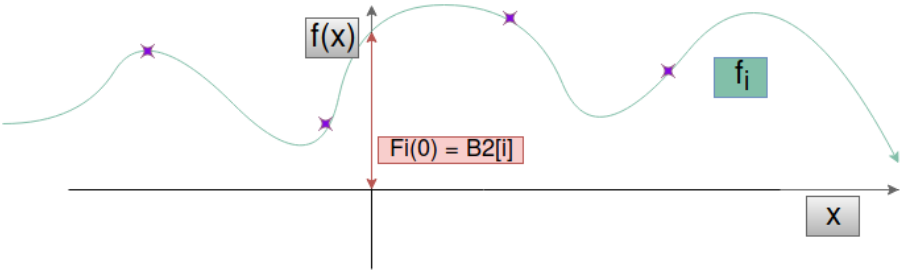
Example : There is a function F_i and you want to it to be as close as possible to the purple dots :



So you have two solutions :
1 - You change all the shape of this function :



2 - You change the Bias of this function :



Since changing the Bias is the constant of this polynomial and is only one that need to be changed to fit the purple dots, you will choose to change the Bias.

5 Exercises

5.1 Matrices

In this TP, the matrices will be used to propagate decision.
Go in `Matrix.cs`.

5.1.1 Constructor

```
1  public Matrix(int height, int width, bool init = false)
```

You have to initialize a new matrix by filling the required attribute(height, width, float table, Bias). If the boolean "init" is true, the matrix has to be initialized with random numbers between 0 and 1.
The Bias has to be initialized with random numbers between -1 and -0.5 .

Bias
Each matrix can be seen as a group of "`Width`" polynomials of "`Height`" degree.
The Bias is the missing constant of each of these polynomials.

5.1.2 Make a copy

```
1  public void MakeCopyFrom(Matrix copy)
```

This method replaces the current matrix with the one given as argument. You have to copy everything stored in the matrix, or your learning won't work when you duplicate the matrices.

5.1.3 Mutation!

```
1   public void ApplyMutation()
```

In this method, you have to apply small changes on each one of the elements in the 2D array of the matrix and on the Bias. We advise that the changes you make to each element be between [-0.1,0.1], using randomly generated numbers. You are free to modify this value in order to change the random range between generations.
It is better to be centered on 0.
The new numbers of the 2D array of the matrix have to be between 0 and 1.
The Bias doesn't need to be bounded. But you are free to do so between [-1;-0.5] if you want.

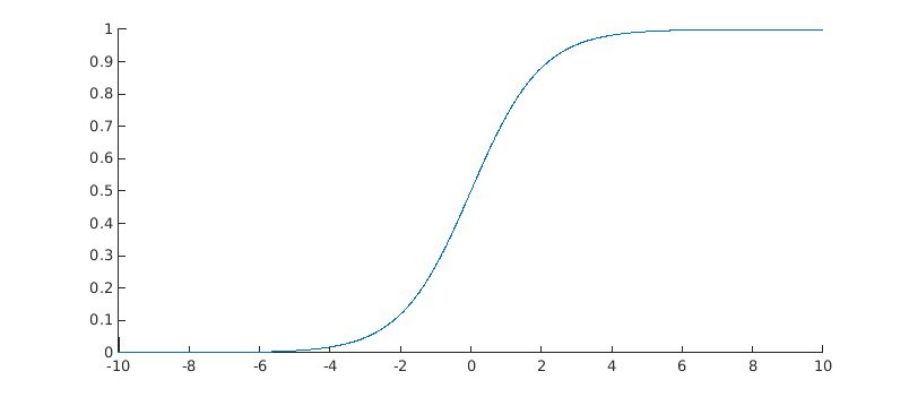
5.1.4 Sigmoid

```
1  private static float Sigmoid(float x)
```

In this case, we will only use it to keep elements between 0 and 1, and to make sure that these elements never reach 0 and never reach 1.

sigmoid

It refers to the special case of the logistic function and is often used in neural networks.
Sigmoid(x) = $\frac{1}{1+e^{-x}}$



5.1.5 Matrix addition

```
1  public static Matrix operator +(Matrix a, Matrix b)
```

In its most classic form! You don't need to do anything on the Bias.

5.1.6 Propagation

```
1   public static Matrix operator *(Matrix a, Matrix b)
```

This operator will allow you to propagate the choice made regarding the environment surrounding the player.

Translate the following pseudo code in C#:

```
1  If width(a) != height(b)
2      throw an exception
3  C = matrix of dimension (height(a), width(b))
4  For i from 0 to height(a) - 1:
5      For j from 0 to width(b) - 1:
6          S = 0
7          For k from 0 to width(a) - 1:
8              S = S + a[i,k] * b[k,j]
9          c[i,j] = Sigmoid(S / width(b) + b.Bias[j])
10 return C
```

5.1.7 Print

```
1   public void Print()
```

This is an optional method which will not be evaluated, but we strongly recommend that you do it, since it will be very useful when debugging.

5.2 The factory

In `Factory.cs`, you'll find everything related to the learning of the player population.

5.2.1 Sorting players

```
1  private static void SimpleSort()
```

This method sorts the player's list based on scores, in ascending order. The operators '>' and '<' between players are already implemented.

5.2.2 Getters

```
1   public static List<Player> GetListPlayer()
```

Returns the player's list.

```
1   public static Player GetBestPlayer()
```

Returns the best player.

```
1   public static Player GetNthPlayer(int n)
```

Sorts the list in ascending order, and returns the n^{th} player.

5.2.3 Save and restore

```
1  public static void SetPathLoad(string path)
```

Sets the `_pathLoad` attribute.

```
1  public static void SetPathSave(string path)
```

Sets the `_pathSave` attribute.

```
1  public static void SetPathLoadAndSave(string path)
```

Set `_pathLoad` and `_pathSave`.

```
1  public static void SaveState()
```

Use the "SaveAndLoad" class to save the player's list. Throws an exception if the saving path does not exist.

5.2.4 Initialization

```
1  public static void InitNew(int size = 200)
```

Initializes the player's list and fills it with "size" players.

```
1   public static void Init()
```

Use the "SaveAndLoad" class to load the new saving file, if the path exists. If it does not, it calls the function `InitNew()`.

```
public static void PrintScore()
```

Displays the scores such as:

```
Player 0 has a score of 0
Player 1 has a score of 0
Player 2 has a score of 0
...
Player 194 has a score of 2120
Player 195 has a score of 4190
Player 196 has a score of 4190
Player 197 has a score of 4190
Player 198 has a score of 7550
Player 199 has a score of 9550
```

5.2.5 Train

```
public static void Train(int generationNumber,
    bool replaceWithMutation = true)
```

The current level can be accessed with "RessourceLoad.GetCurrentMap()".
Get the maximum number of frames with the levels "Timeout".

Repeat on `n` generation the following process:

```
For each player:
    reset the score to 0 // Use resetScore() instead of setScore(0)
    set the player's position to the level's starting point
    For each of the k frames :
        we call on the player the method which makes him play one frame
We call the regenerate method with the replaceWithMutation variable
```

We invite you to read the methods already present in the `Player` class to understand them. Moreover, it can be a good idea to print the percentage of progress of this function.
ie : If you have 100 players, at the end of the first player's training you print "1%"

5.2.6 Renew the population

```
private static void Regenerate(bool replace_with_mutation = true)
```

We sort the players, then replace the weakest half of them. "replaceWithMutation" will be used in the method you call. This variable differentiate the two kinds of replacement, as explained in the 'course' section.

6 Your Score

A part of your grade will be on your bot's score.
This final score is the sum of all the scores on each of the maps.

6.1 Intra

You can go and check the leaderboard in order to see your progress. I will be updated regularly.

6.2 Tips

You are encouraged to train your bot on several maps at the same time, and also to create more maps. It will provide diversity in training, and so your bot will be more versatile.
You are free to create any method if your code compiles and if you don't change the current methods' prototypes.

7 Testing

Testing is a mandatory part in all projects. It ensures you to avoid regression.
Regression happens when some methods stop functioning as you intended them to.
If you don't have any tests, you may not see it.

7.1 You said nuggets ?

There are packages that are called NuGet packages in C#. These packages are libraries in a folder in your project. We are currently using a package that facilitate our testing, and an other that is providing us a 2D graphical interface.

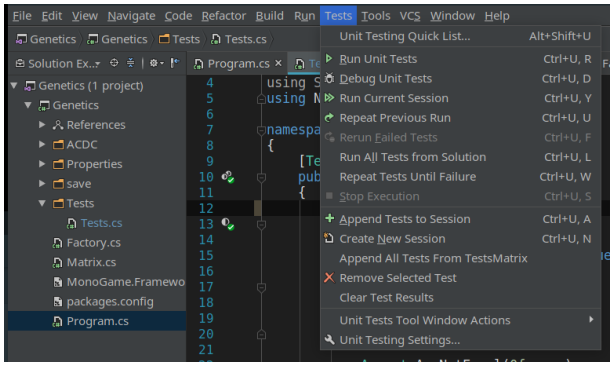
But why using packages ?

Simply because they are exportable. It means that you could use an USB key and give your project to a friend, and he/she would be able to run your code without installing any library contained in these packages. (Your package still have to be compatible with the Operating System)

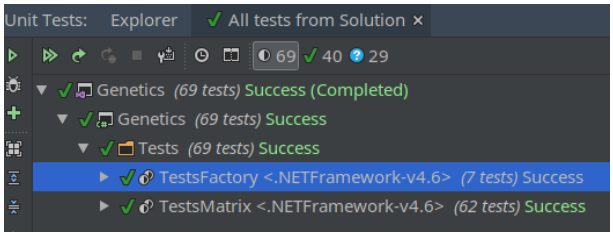
This is an example, sharing your code will still be considered as cheating.

7.2 Testing your code

We made some tests for you, you will be able to find them in the "Tests" tab :



You will then click "Run Current Session"
You should see the following :



They will give you a good approach of testing your code in C#.

8 Bonus

We are going to suggest a few bonuses. You will have to tell us in your README which ones you have done.

8.1 More levels!

Add new maps in the map directory.

8.2 One for all

Add a new method which will keep only the best player from each generation, which will duplicate him as many times as there were individuals in the population, and which will apply changes only on the duplicated elements.

8.3 Multithreading

You can try to parallelize the players' learning, to divide the learning duration by the number of used threads.
More informations here :
https://www.tutorialspoint.com/csharp/csharp_multithreading.htm

You will want to use "Task". You will be able to find examples on :
fr - `https://fdorin.developpez.com/tutoriels/csharp/threadpool/part1/#LV-B`
en - https://docs.microsoft.com/en-us/dotnet/standard/parallel-programming/task-based-asynchronous-programming#creating-and-running-tasks-explicitly

8.4 Show us what you got

If you have any idea, feel free to try them and explain them in your README!

These violent deadlines have violent ends.
The game is end.