TP C#14 : Évolution génétique

Consignes de rendu

A la fin de ce TP, vous devrez rendre une archive respectant l'architecture suivante :

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

N'oubliez pas de vérifier les points suivants avant de rendre :

— Remplacez `prenom.nom` par votre propre login et n'oubliez pas le fichier `AUTHORS`.
— Les fichiers `AUTHORS` et `README` sont obligatoires.
— Pas de dossiers `bin` ou `obj` dans le projet.
— N'oubliez pas de supprimer le dossier caché `.idea` du projet.
— Respectez scrupuleusement les prototypes demandés.
— Retirez tous les tests de votre code.
— **Le code doit compiler !**

AUTHORS

Ce fichier doit contenir une ligne formatée comme il suit : une étoile (*), un espace, votre login et un retour à la ligne. Voici un exemple (où `$` est un retour à la ligne et ␣ un espace) :

```
*␣prenom.nom$
```

Notez que le nom du fichier est `AUTHORS` sans extension. Pour créer simplement un fichier `AUTHORS` valide, vous pouvez taper la commande suivante dans un terminal :

```
echo "* prenom.nom" > AUTHORS
```

README

Vous devez écrire dans ce fichier tout commentaire sur le TP, votre travail, ou plus généralement vos forces / faiblesses, vous devez lister et expliquer tous les boni que vous aurez implémentés. Un `README` vide sera considéré comme une archive invalide (malus).

Table des matières

1 Introduction

1.1 Objectifs

Ce TP a pour but de vous montrer le principe de fonctionnement d'un algorithme génétique, et de vous en faire créer et utiliser un.

Algorithme Génétique

Algorithme dont le but est d'obtenir une solution approchée à un problème d'optimisation, lorsqu'il n'existe pas de méthode exacte (ou que la solution est inconnue) pour le résoudre en un temps raisonnable. Les algorithmes génétiques utilisent la notion de sélection naturelle et l'appliquent à une population de solutions potentielles au problème donné. La solution est approchée par « bonds » successifs.

(Cf : https://fr.wikipedia.org/wiki/Algorithme_génétique)

Comme énoncé dans la définition ci dessus, le but d'un algorithme génétique est de donner une solution approchée à un problème trop "complexe". Dans notre cas, vous aller résoudre un jeu de plates-formes en 2D que nous avons créé.

Votre but est de créer une IA tout terrain.

2 Cours partie 1

Le principe de fonctionnement d'un algorithme génétique est très simple, c'est la sélection naturelle ! Vous aller créer des populations de joueurs, tester les joueurs un par un et leur attribuer un score. Puis vous ne garderez que les meilleurs et recommencerez.

2.1 Étape par Étape

2.1.1 Génération 0

Vous aller créer une première génération de joueurs :

Joueur	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 Attribution d'un score

Puis l'évaluer sur un niveau de votre choix, pour attribuer un score à chacun des joueurs.

Joueur	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 Tri des joueurs

Ensuite trier ce tableau, et enlever les éléments les plus faibles.

Joueur	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 Évolution !

Il y a deux possibilités pour faire évoluer cette population :
- Générer aléatoirement des nouveaux joueurs. (Cas 1)
- Faire une copie de la meilleure moitié des joueurs et leur appliquer des modifications. (Cas 2)

Ces nouveaux joueurs générés remplaceront la moitié des joueurs actuels ayant le moins bon score.

Cas 1

Joueur	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

Cas 2

Joueur	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 Évolution sans fin ?

Ensuite vous aller répéter les étapes précédentes depuis l'attribution d'un score.
Ce sera ensuite à vous d'estimer comment entraîner vos populations de joueurs.
En effet, selon le nombre de générations et selon les niveaux que vous aller utiliser pour l'entraînement, de gros changements peuvent advenir.
N'oubliez pas, votre IA devra être capable de s'adapter à différents terrains.

Vous êtes fortement invités à créer vos niveaux. (Voir Section 3.3 création de niveau)

2.2 Illustration Vidéo

Ces vidéos sont une bonne illustration du principe des algorithmes génétiques (4 parties) :
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 Aspect général

3.1 Pour Windows

Si la version que vous avez ne fonctionne pas au point du vue graphique sous Windows, vous devez ajouter un NuGet package.
Tools > NuGet > Manage NuGet Package for ...
Vous aurez besoin du NuGet package "MonoGame.Framework.WindowsDX version 3.4.0.459"
(Source : https://github.com/MonoGame/MonoGame/issues/5531#issuecomment-326928360)

3.2 L'interface graphique

Dans ce TP l'interface graphique est facultative, car vous pouvez voir le score obtenu par vos joueurs.
Cependant pour l'activer il suffit d'utiliser une des fonctions ci-dessous, dans la fonction `Main`.

```
1  static void Showbest() // Montre le meilleur joueur
2  static void ShowNth(int nth) /* Montre le joueur à la position
3  nth de la liste des joueurs (comportant 200 joueurs, triés par
4  ordre croissant de score) */
5  static void PlayAsHuman() // Permet de jouer à la place de l'ordinateur
```

3.3 Création de niveau

Pour créer un niveau c'est très simple, rendez vous dans le dossier map, et créez un fichier "*.map" (l'étoile "*" signifie n'importe quelle chaîne de caractères).
Il vous suffit ensuite d'y écrire en première ligne, un nombre maximum de "frame" (le nombre d'action que l'on va demander à un joueur). Puis vous pourrez utiliser les caractères suivants :
"W" : un mur
"S" : point de départ
" " : une case vide
pour former votre niveau.

La seule contrainte est de former un rectangle avec ces seuls caractères.
Notre parser lit tous les fichiers présents dans le dossier map et renvoie une exception en cas de non respect d'une de ces consignes.

Nous vous recommandons de vous inspirer des maps que nous avons créé.
Vous pouvez accéder aux maps grâce à :

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

3.4 Évaluation de la performance de votre meilleur joueur

Pour évaluer la performance de votre meilleur joueur vous devrez nous rendre un fichier "bot.save" qui sera situé dans le dossier "save" (Voir architecture du projet)
Vous utiliserez la fonction :

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

3.5 Sauvegarde de score

Vous êtes autorisés à changer cette ligne dans Program.cs :

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

Si vous souhaitez avoir plusieurs fichiers de sauvegarde.
Cependant nous avons choisi de laisser les fichiers `.save` d'entraînement dans le dossier "bin" pour qu'il n'apparaissent pas dans votre rendu.
Vous aurez un malus si vous laissez des fichiers `.save` d'entraînement dans votre rendu.
Le dossier courant de sauvegarde est le dossier "bin", donc si vous ne remontez pas dans la hiérarchie, tout devrait bien se passer.
Voici des exemples valides :

```
1  private const string PathForTest = "test1.save";
2  private const string PathForTest = "bestTeam.save";
3  private const string PathForTest = "toutReprogrammer.save";
4  private const string PathForTest = "pourUnMondeSansDanger.save";
```

Vous ne devez pas toucher le chemin :

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

Vous serez notés sur la présence et la qualité de ce fichier.

3.6 Le joueur

Le joueur est composé de différentes méthodes et attributs.

3.6.1 Sa position actuelle

L'abscisse de cette position servira à calculer le score.

3.6.2 3 matrices

Le joueur contient 3 matrices qui représentent son cerveau, elles sont accessible sous forme de liste via la méthode

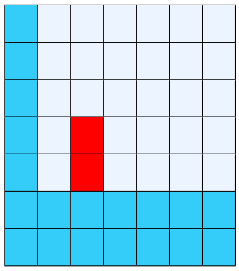
```
1  public List<Matrix> Getbrains()
```

Vous devrez utiliser ces matrices pour déterminer le mouvement du joueur dans `Matrix.cs`
Lorsque le joueur va effectuer une action via la méthode :

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

Il va regarder les cases autour de lui pour décider de ce qu'il fait.

Exemple : (Rouge : le joueur, Bleu : les murs, Gris Clair : l'air)



Il va donc recevoir une matrice de float F telle que :

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

Notons B_1 B_2 B_3 respectivement de dimensions 49x16, 16x16 et 16x4 ; les 3 matrices contenues dans le joueur.

Le joueur a besoin de la matrice S de dimension 4x1 pour savoir quelles décisions prendre. En considérant S comme une liste que l'on note S, on peut alors utiliser pour définir les actions du joueur tel que :

```
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();
```

Pour calculer S vous appliquerez la formule :
$S = F * B_1 * B_2 * B_3$
Avec l'opérateur "*" légèrement différent de la multiplication de matrice. (Il sera détaillé dans la partie matrice).

Note
Soit x l'élément d'une des matrices ci dessus "M". $\forall x \in M, x \in [0,1] \cap \mathbb{R}$

3.6.3 Sa vitesse

Lorsque le joueur saute, sa vitesse est réduite de moitié. Cela permet d'éviter des sauts inutiles lors de l'apprentissage, et ajoute du réalisme.

3.6.4 Son score

Le score d'un joueur sert à déterminer à quel point il a réussi un niveau, et à le comparer aux autres.
Il est calculé à partir de l'abscisse du joueur, une fois le nombre d'actions autorisées sur le niveau est épuisé.

3.6.5 Des forces physiques

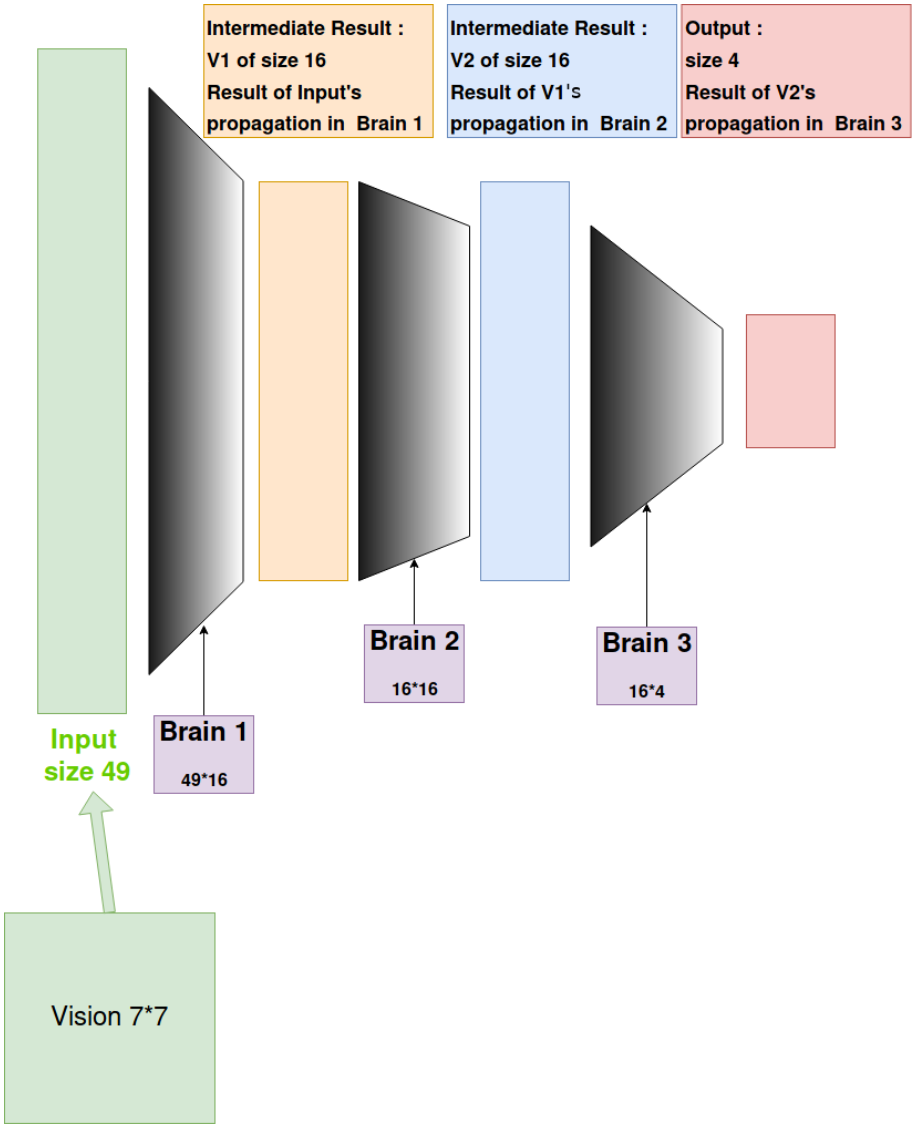
Rien ne vous sera demandé dans ce TP les concernant, mais ce sont ces dernières qui font qu'il y a de la gravité et donc que le joueur retombe après son saut.

4 Cours partie 2

4.1 La propagation en elle même

L'idée principale de la propagation est de mettre en relation des informations d'entrée avec des matrices pour produire une nouvelle information.
Premièrement, votre input d'entrée est la vision du joueur. Cette dernière passe à travers le premier cerveau (qui est une matrice), afin de produire l'information V_1. V_1 passe maintenant à travers le second cerveau (qui est aussi une matrice), afin de produire l'information V_2. Le même procédé est effectué pour V_2 à travers le 3ème cerveau. Cela nous donne l'output. Cette information de sortie est nécessaire au joueur pour décider de ce qu'il doit faire.

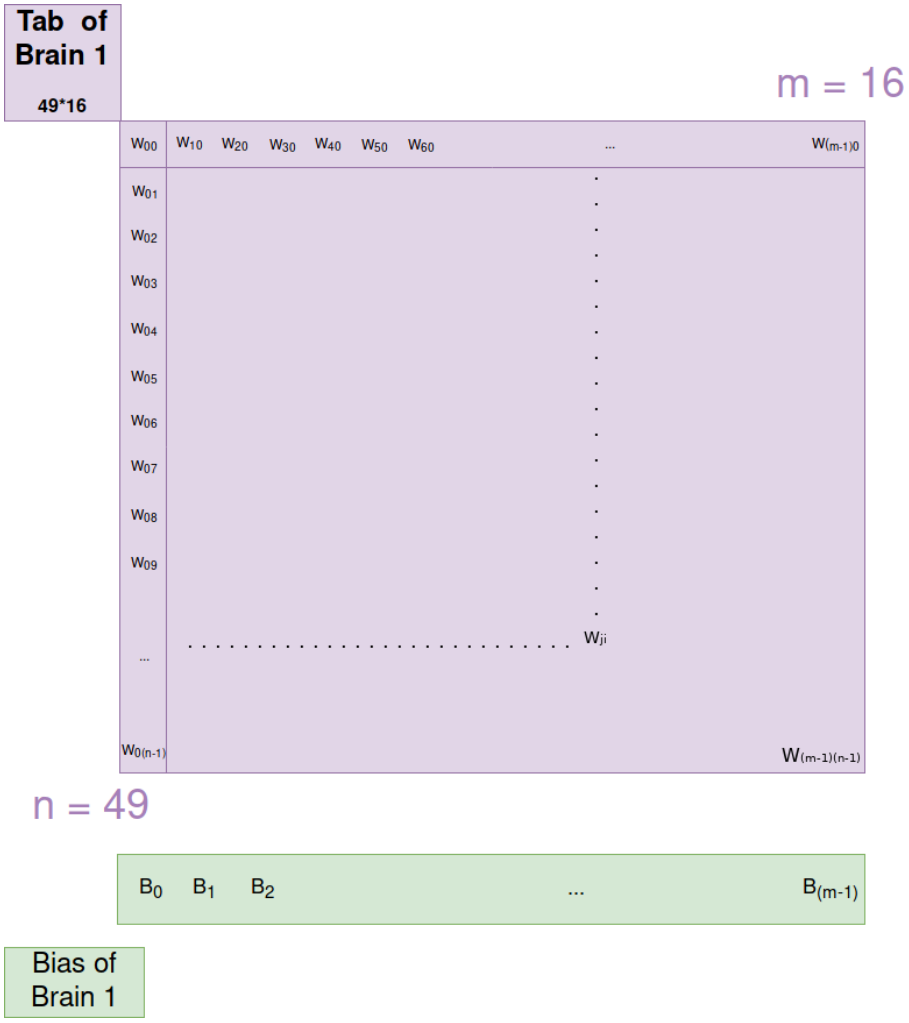
Dans notre cas, l'output est une matrice de taille $1 * 4$. Pour chaque valeur de cette matrice qui est supérieure à 0,5 ; l'action correspondant à cette valeur est activée.
ex :
Prenons une matrice M composée de $[0.45; 0.55; 0.56; 0.48]$ qui est le résultat que nous avons calculé au préalable.
L'output est lié aux actions suivantes : [Go left ; Go right ; Go up ; Reset]
Donc le joueur fera les actions "Go right" et "Go up".



4.2 Formule détaillée de la propagation

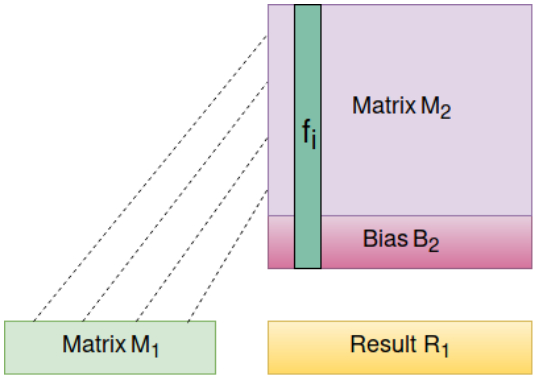
Le cerveau est composé d'un tableau de Float de 2 dimensions et d'un autre tableau de Float de 1 dimension. Appelons V_1 la matrice résultante après la propagation de l'input à travers le cerveau 1.
Pour calculer cette propagation, vous devez appliquer la formule suivante pour chaque élément de V_1 :

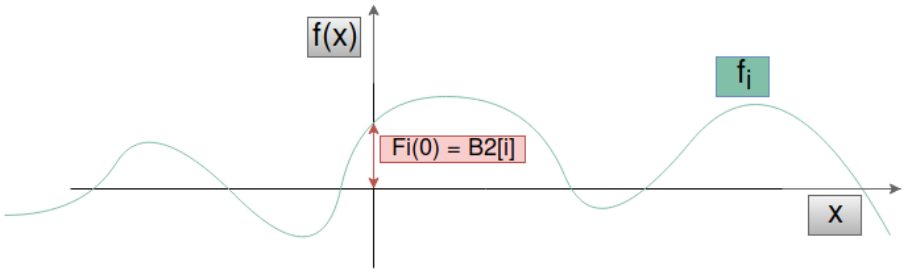
$$V_1[j] = Sigmoid(Bias[j] + \frac{\sum_{i=0}^{n-1} Input[i] * \omega_{ji}}{n})$$



4.3 Principe du Bias

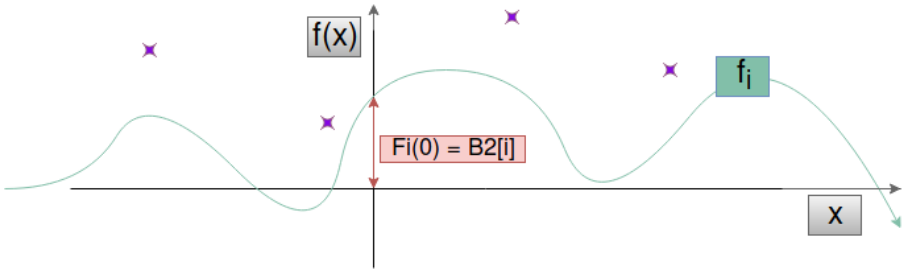
Ci-dessous, vous pouvez voir une représentation simple de la propagation.
Comme dit précédemment, le Bias est une liste des constantes manquantes à l'ensemble des polynômes qui composent la matrice.
Vous allez maintenant voir pourquoi :
Si tous les ω sont à 0, il reste le Bias. Plus formellement : $F_i(0) = B_2[i]$.



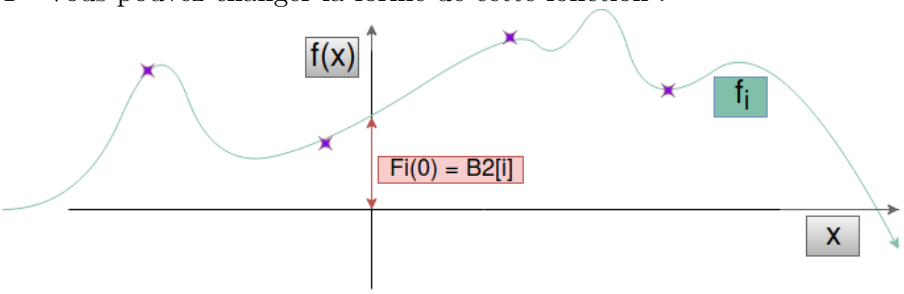


Pourquoi utiliser un Bias ?
Simplement parce que votre fonction peut avoir la forme souhaitée mais avoir une constante incorrecte qui va décaler en ordonnée cette dernière.

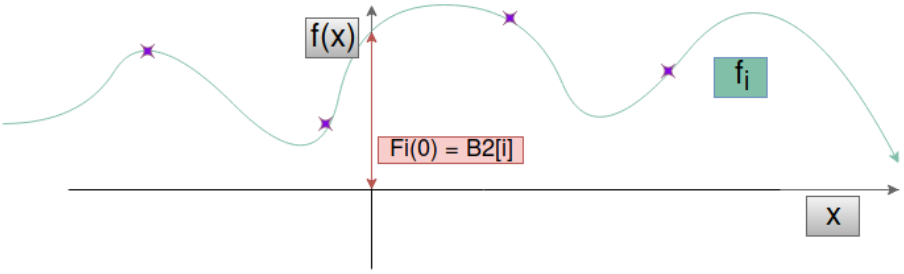
Exemple : Prenons une fonction F_i, nous souhaitons qu'elle soit le plus proche possible des points violets :



Il existe 2 solutions :
1 - Vous pouvez changer la forme de cette fonction :



2 - Vous pouvez changer la constante de cette fonction :



Le Bias étant la constante en question, il est plus simple de changer uniquement le Bias, que de changer complètement la forme en espérant correspondre aux points violets.

5 Exercices

5.1 Matrices

Dans ce TP, les matrices vont être utilisées pour propager des décisions.
Rendez-vous dans le fichier `Matrix.cs` .

5.1.1 Constructeur

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

Vous devez Initialiser une nouvelle matrice en remplissant les attributs nécessaires. (hauteur,largeur, tableau de floats, Bias) Si le booléen "init" est vrai, la matrice doit être initialisée avec des nombres aléatoires compris entre 0 et 1.
Le Bias doit être initialisé avec des nombres compris entre -1 et -0.5 .

Bias
Chaque matrice peut être vue comme un nombre "`Width`" de polynômes de degré "`Height`" .
Le Bias est la constante qui manque à chacun de ces polynômes.

5.1.2 Faire une copie

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

Cette méthode remplace la matrice actuelle avec celle donnée en argument. Vous devez copier tout ce qui est contenu, et pas seulement faire une copie de référence. Sinon votre apprentissage ne marchera pas lorsque vous dupliquerez les matrices.

5.1.3 Muter !

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

Dans cette méthode vous devez appliquer des petits changements sur chacun des éléments du tableau de la matrice et sur son Bias. Nous vous conseillons de faire varier chacun de ces éléments d'une valeur comprise entre [-0.1,0.1] en utilisant des nombres générés aléatoirement.
Vous pouvez faire varier cette valeur pour appliquer des changements entre générations plus ou moins forts.
Nous vous conseillons tout de même d'être centré en 0.
Les nouveaux nombres doivent être compris entre 0 et 1 pour le tableau de la matrice.
Le Bias n'a pas besoin d'être borné. Mais vous pouvez le borner entre [-1 ;-0.5] si vous le souhaitez.

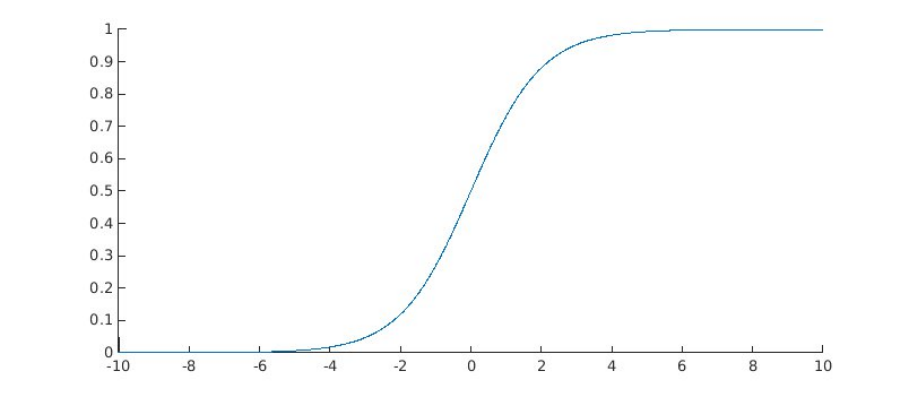
5.1.4 Sigmoïde

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

Dans notre cas nous allons simplement l'utiliser pour contenir éléments entre 0 et 1, sans qu'il n'atteignent 0, ni qu'ils n'atteignent 1.

sigmoïde

Elle représente la fonction de répartition de la loi logistique et est souvent utilisée dans les réseaux de neurones.
Sigmoïde(x) = $\frac{1}{1+e^{-x}}$



5.1.5 Addition de matrice

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

Basique addition de matrice. Vous n'avez rien besoin de faire sur les Bias.

5.1.6 Propagation de matrice

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

Cet opérateur va permettre de propager les choix à faire par rapport à l'environnement autour du joueur.

Vous devez traduire le pseudocode suivant en C# :

```
1  Si largeur(a) != hauteur(b)
2      renvoyer une exception
3  C = matrice de dimension (hauteur(a),largeur(b))
4  Pour i allant de 0 à hauteur(a) - 1:
5      Pour j allant de 0 à largeur(b) - 1:
6          S = 0
7          Pour k allant de 0 à largeur(a) - 1:
8              S = S + a[i,k] * b[k,j]
9          c[i,j] = sigmoïde(S / largeur(b) + b.Bias[j])
10 retourner C
```

5.1.7 Print

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

Cette fonction est facultative et ne sera pas notée, mais elle peut vous être d'une grande aide pour débugger.

5.2 L'usine

Dans le fichier `Factory.cs` vous trouverez tout ce qui concerne l'apprentissage des populations de joueur.

5.2.1 Trier les joueurs

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

Cette fonction sert à trier la liste de joueurs par score, dans l'ordre croissant. Les opérateurs '>' et '<' sont déjà implémentés entre les joueurs.

5.2.2 Getters

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

Retourne la liste de joueurs.

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

Retourne le meilleur joueur.

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

Trie la liste dans l'ordre croissant, et retourne le $n^{ème}$ joueur.

5.2.3 Sauvegarder et restaurer

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

Set l'attribut `_pathLoad`

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

Set l'attribut `_pathSave`

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

Set l'attribut `_pathLoad` et l'attribut `_pathSave`

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

Utilise la classe "SaveAndLoad" pour sauvegarder la liste de joueur. Renvoie une exception si le chemin de sauvegarde n'existe pas.

5.2.4 Initialisation

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

Initialise la liste de joueur et la remplit de "size" joueurs.

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

Utilise la classe "SaveAndLoad" pour charger le fichier de sauvegarde, si le chemin existe, sinon appelle la fonction `InitNew()`.

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

Affiche le score des joueurs tel que :

```
1  Player 0 has a score of 0
2  Player 1 has a score of 0
3  Player 2 has a score of 0
4  ...
5  Player 194 has a score of 2120
6  Player 195 has a score of 4190
7  Player 196 has a score of 4190
8  Player 197 has a score of 4190
9  Player 198 has a score of 7550
10 Player 199 has a score of 9550
```

5.2.5 Train

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

Le niveau actuel s'obtient par "RessourceLoad.GetCurrentMap()"
Récupèrer le nombre maximum de frames avec le "Timeout" du niveau

Répèter sur `n` générations le procédé suivant :

```
1  Pour chaque joueur:
2      remise à 0 du score // Utilisez resetScore() à la place de setScore(0)
3      position du joueur au point de départ du niveau actuel
4      Pour chacune des k frames :
5          on appelle sur le joueur la méthode qui lui fait jouer une frame
6  On appelle la fonction regenerate avec la variable replaceWithMutation
```

Nous vous invitons à parcourir les méthodes déjà présentes dans la classe `Player` pour vous faciliter le travail.
De plus, il est judicieux de faire apparaître un pourcentage d'avancement dans cette fonction.
ex : A la fin de l'entraînement du joueur 1, vous affichez "1%".

5.2.6 Renouveler la population

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

On trie les joueurs, et on remplace la moitié d'entre eux ayant eu le plus mauvais score. "replaceWithMutation" sera à utiliser dans la méthode que vous appelez. Cette variable différencie les 2 types de remplacement, comme expliqué dans la partie cours.

6 Votre Score

Vous allez être notés en partie sur le score de votre bot.
Ce score final est la somme des scores de votre bot sur chacune des maps.

6.1 Intra

Vous pouvez consulter le leaderboard sur l'intra pour voir votre progression.

6.2 Conseils

Il vous est vivement conseillé d'entraîner votre bot sur plusieurs maps en même temps et de créer des nouvelles maps. Cela permettra de créer de la diversité dans le type d'entraînement et d'avoir un bot plus polyvalent.
Vous pouvez créer les méthodes que vous voulez, tant que votre code compile et que vous ne changez pas les prototypes des fonctions déjà présentes.

7 Tester ?

Tester son code est un partie très importante de la majorité des projets que vous allez rencontrer. Cela vous permet d'éviter les régressions.
Une régression arrive lorsqu'une ou plusieurs de vos méthodes ne fonctionnent plus de la manière souhaitée. Si vous n'avez pas fait de tests, vous pouvez ne pas le voir.

7.1 Vous avez dit nuggets ?

Il existe des paquets appelés "NuGet packages". Un paquet est un ensemble de bibliothèques qui se place automatiquement (grâce à rider) dans un dossier de votre projet.
Nous sommes actuellement en train d'en utiliser 2, le premier pour nous faciliter la création de tests, le second pour avoir un moteur graphique 2D.

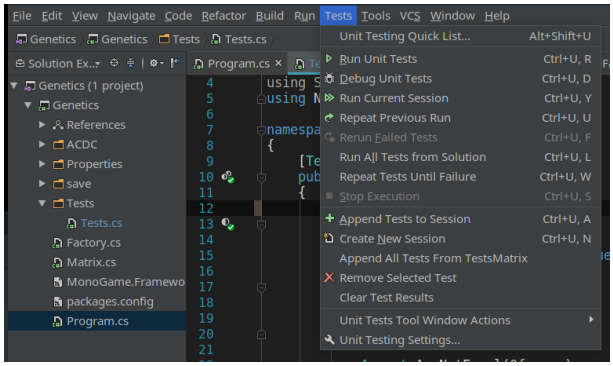
Pourquoi utiliser des paquets ?

Simplement parce qu'ils sont exportables. Cela veut dire que vous pourriez utiliser une clé USB et passer votre projet à un(e) ami(e), et il/elle pourrait lancer ce dernier sans avoir à installer les bibliothèques contenues dans ces paquets. (Les paquets doivent être compatible avec le système d'exploitation)

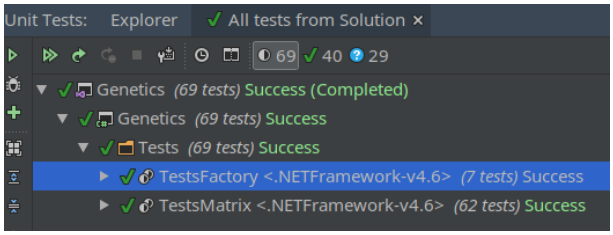
Ceci est un exemple, le partage de code sera toujours considéré comme de la triche.

7.2 Comment tester votre code ?

Nous vous avons préparé une série de tests, vous pourrez les trouver dans l'onglet "Tests" :



Cliquez sur "Run Current Session"
Vous devriez voir apparaître :



Ces tests vous donneront une bonne approche de la création de test-suite en C#.

8 Bonus

Nous allons vous suggérer un certain nombre de bonus, vous indiquerez dans votre README ceux que vous avez fait.

8.1 Plus de niveaux !

Ajoutez des nouvelles maps dans le dossier map.

8.2 One for all

Ajoutez une nouvelle méthode qui ne gardera que le meilleur joueur de chaque génération, qui le dupliquera autant de fois qu'il y avait d'individus dans la population. Et qui appliquera des changements seulement sur les éléments dupliqués.

8.3 Multithreading

Vous pouvez paralléliser l'apprentissage de vos joueurs, pour diviser de temps d'apprentissage par le nombre de thread utilisés.
Plus d'informations sur ce site :
https://www.tutorialspoint.com/csharp/csharp_multithreading.htm

Vous aurez envie d'utiliser des "Task". Vous pourrez trouver des exemples sur :
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 Vos idées sont les bienvenues

Si vous avez des idées de bonus, faites les et écrivez les dans votre README.

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