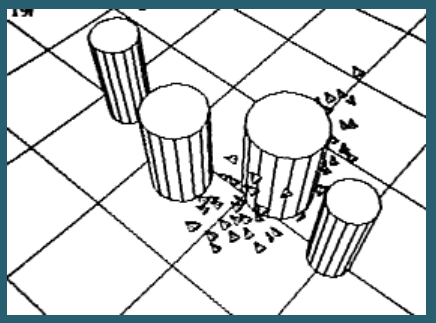
PROGRAMMATION
Projet Boids

MAY 8

FLOCH **Basile**

ROBACH **Kayané**



Programmation

Dans cette partie nous nous intéresserons plus particulièrement à la programmation de notre projet. Nous allons donc expliciter les différents algorithmes utilisés dans le code de programmation mais également illustrer les fonctions définies dans la partie d'analyse à l'aide de séries de jeux de tests. Nous ferons part également des modifications que notre projet a subi suite aux hypothèses émises lors de la partie d'analyse et de conception.

Graphisme

Plateau

Nous avons donc créé un plateau de simulation carré de taille (700x700), dont le fond porte un motif de ciel nuageux. Lors de la partie d'analyse et de conception nous avions eu pour idée de réaliser des obstacles face auxquels les oiseaux modifieront leur comportement.
Deux types d'obstacles ont donc été mis en place :

Des arbres, de forme circulaire, représentés en vert :

```
'on va creer des cerises aleatoirement sur chaque arbre
Sub creerarbres(ByVal nbr As Integer)
        nbarbres = nbr
        If nbr > 10 Then
            MsgBox("le nombre d obstacles doit être inférieur ou égal a 10")
        Else
            Dim i As Integer
            ReDim arbres(nbr - 1)
            Dim rdm As Random
            rdm = New Random()
            For i = 0 To nbarbres - 1
                'on fait - 40 pour éviter qu'un arbre se dessine au bord
                arbres(i).x = (plateauX - 40) * rdm.NextDouble()
                arbres(i).y = (plateauY - 40) * rdm.NextDouble()
                arbres(i).rayon = 20
                Dim j As Integer
                Randomize()
                'nbcerises est le nombre de cerises par arbres
                arbres(i).nbcerises = Int(8 * Rnd()) + 2 'définition d un nombre aléatoire
entier entre 2 et 8
                ReDim arbres(i).cerises(arbres(i).nbcerises - 1)
                For j = 0 To arbres(i).nbcerises - 1
                    'position initiale définie aléatoirement sur chaque arbre
                    Randomize()
                    arbres(i).cerises(j).x = arbres(i).x - 10 + 10 * rdm.NextDouble()
                    arbres(i).cerises(j).y = arbres(i).y + 10 - 10 * rdm.NextDouble()
                    arbres(i).cerises(j).rayon = 5
                Next
            Next
        End If
End Sub
```

La création des obstacles sera détaillée sur le fichier Annexe à travers des algorithmes commentés.

Nous avons ajouté un système d'alimentation, à travers la création de cerises disponibles sur les arbres. Les boids peuvent en consommer autant que leur masse ; si la valeur de leur masse est dépassée en consommation de cerises : ils disparaissent.

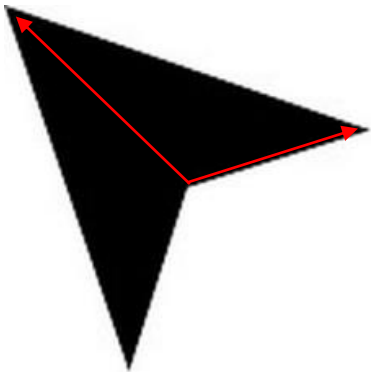
Des maisons, de forme carré, représentées en rouge :

```
Sub creerobstacles(ByVal nbr As Integer)
        nbobstacles = nbr
        If nbr > 10 Then
            MsgBox("le nbr d obstacles doit etre inferieur ou egal a 10")
        Else
            Dim i As Integer
            ReDim obstacles(nbr - 1)
            Dim rdm As Random
            rdm = New Random()
            For i = 0 To nbobstacles - 1
                'on fait - 40 pour eviter qu un obstacle se dessine au bord
                obstacles(i).x = (plateauX - 40) * rdm.NextDouble()
                obstacles(i).y = (plateauY - 40) * rdm.NextDouble()
            Next
        End If
    End Sub
```

Boids

Dans la première partie du dossier nous représentons les Boids avec un triangle, nous permettant de visualiser leur position et leur direction. Ici nous avons choisi de le représenter par une flèche ce qui nous permet de mieux visualiser leur position exacte. De plus les Boids sont plus ou moins volumineux en fonction de leur masse prédéfinie aléatoirement par un entier entre 3 et 5.

Pour modéliser la forme des Boids nous utilisons une formule dite de « rotation », qui assure la rotation d'un vecteur selon un angle (donné en radian). Le point central de la flèche est la position du Boid, quant à la pointe de la flèche, elle est définie par la position du Boid à laquelle on ajoute le vecteur de direction du Boid multiplié par 10 (pour le rendre plus visible car les vecteurs sont normalisés).
Pour les 2 points de la base de la flèche on utilise la formule rotation pour faire la rotation du vecteur direction du Boid selon 2 angles : $2\pi/3$ et $-2\pi/3$, puis on multiplie ces vecteurs par la masse du Boid afin d'obtenir une représentation des Boids plus volumineuse et qui nous permettra donc de mieux distinguer leurs mouvements sur le plateau.



Représentation graphique

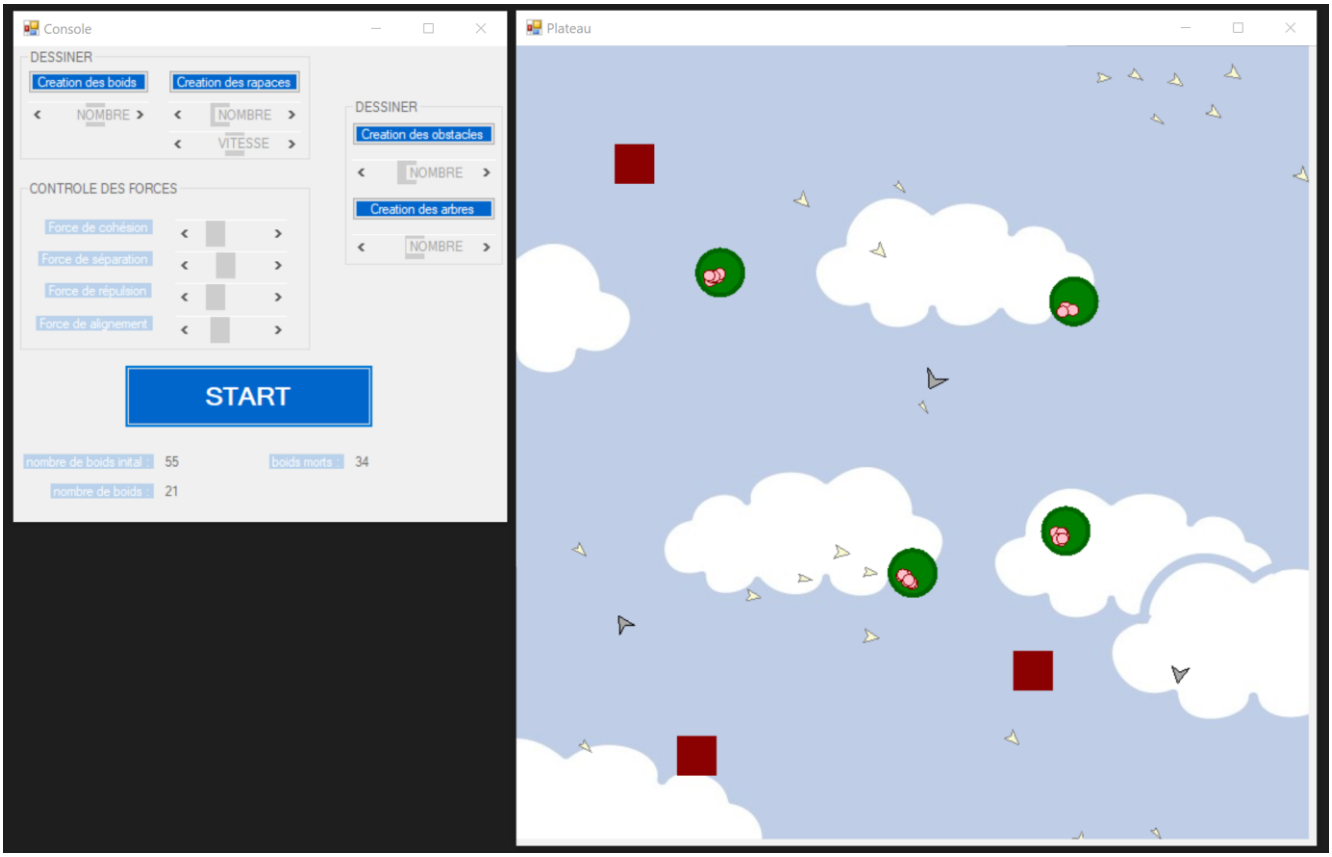
Voici la procédure permettant de réaliser la conception graphique d'un boid :

```
Sub creerboids(ByVal nbr As Integer)
        nbboids = nbr
        If nbr > 100 Then
            MsgBox("le nbr de boids doit etre inferieur ou egal a 100")
        Else
            Dim i As Integer
            ReDim boids(nbr - 1)
            Dim rdm As Random
            rdm = New Random()
            For i = 0 To nbboids - 1
                'position initiale definie aleatoirement sur le plateau
                boids(i).p.x = plateauX * rdm.NextDouble()
                boids(i).p.y = plateauY * rdm.NextDouble()
                boids(i).d.x = rdm.NextDouble()
                boids(i).d.y = rdm.NextDouble()
                boids(i).vitesse.x = rdm.NextDouble()
                boids(i).vitesse.y = rdm.NextDouble()
                boids(i).acc.x = 1
                boids(i).acc.y = 1
                boids(i).repas = 0
                boids(i).id = i
                Randomize()
                boids(i).masse = Int(5 * Rnd()) + 3 'definition d un nombre aléatoire
entier entre 3 et 5
            Next
        End If
    End Sub
```

Et voici la fonction de rotation utilisée pour le dessin :

```
'la fonction rotation fait la rotation d un vecteur defini par le point p selon l angle a (en radian)
    'visual studio utilise les radians comme unite
    Public Function rotation(ByVal p As point, ByVal a As Double) As point
        Dim q As point
        q.x = p.x * Math.Cos(a) - p.y * Math.Sin(a)
        q.y = p.x * Math.Sin(a) + p.y * Math.Cos(a)
        Return q
    End Function
```

Nous avons modifié aussi l'interface graphique vis-à-vis de l'idée de base que nous avions :



Interface graphique définitive

Console

Le plateau de simulation est accompagné de la console sur laquelle l'utilisateur commande la création des obstacles, des arbres, des rapaces et celle des Boids.
Afin de pouvoir observer les différents phénomènes de vol de groupe des Boids tout en restant dans la limite du raisonnable nous avons décidé de mettre en jeu un maximum de 10 arbres, 10 maisons, 10 rapaces et 100 Boids.
Les curseurs de la console de commande nous permettront de modifier a notre guise les coefficients des forces qui s'appliquent sur les oiseaux pour influencer leur comportement.

COMPORTEMENT

Lors de la partie d'analyse et de conception du projet nous avions défini un modèle dans lequel les Boids étaient soumis à 4 forces majeures (répulsion, alignement, cohésion, séparation). Nous avons donc modélisé ces différentes forces sur le logiciel afin que celles-ci aboutissent à un résultat le plus réaliste possible.
Les différents algorithmes qui représentent ces forces seront explicités dans le dossier Annexe qui contient le code commenté.

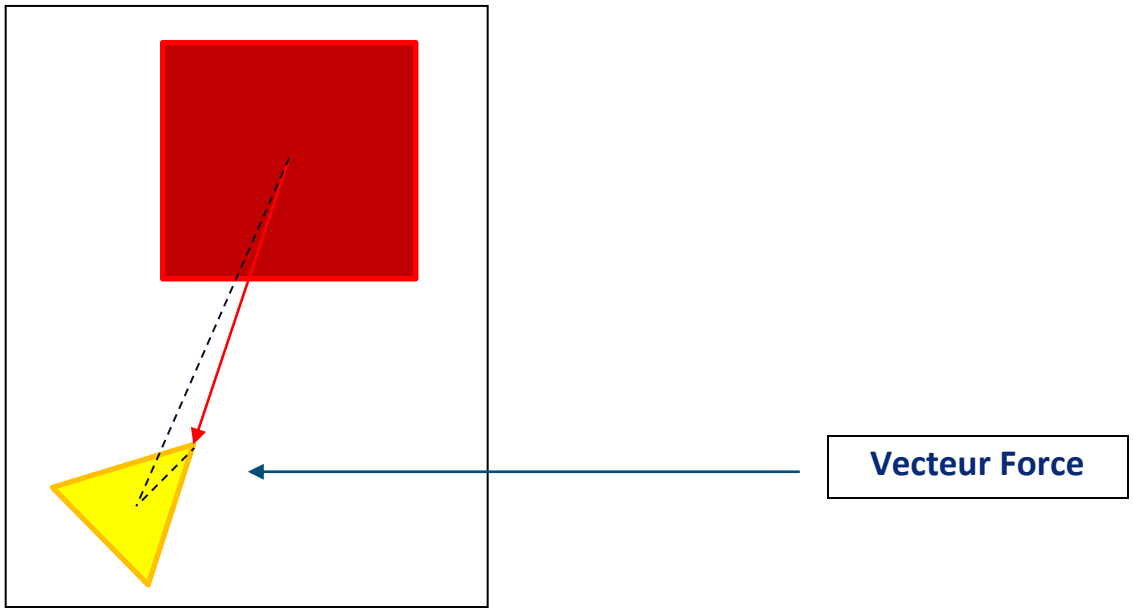
Elle assure le fait qu'il ne doit y avoir aucune collision entre les oiseaux durant le vol, en effet deux Boids ne peuvent être à la même position au même moment.
Pour cela il est nécessaire de définir une distance minimale A entre les oiseaux, si la distance entre deux oiseaux est inférieure à A alors un des deux oiseaux doit s'éloigner.

Nous pouvons à présent expliciter les différentes forces et leurs modules :

Force de Repulsion

La force de répulsion étant la fonction d'évitement par définition, elle assure en effet le fait que les Boids ne doivent pas entrer en collision avec les obstacles présents sur le plateau.

Nous en avons fait une représentation graphique afin de bien comprendre le phénomène voulu et comprendre les mouvements des Boids :



Cette force attire le Boids vers l'arrière pour leur éviter d'entrer en collision avec un obstacle (les obstacles étant les maisons).
Elle est composée de l'inverse du vecteur séparant` l'obstacle et le Boid auquel on somme le vecteur direction du Boid.

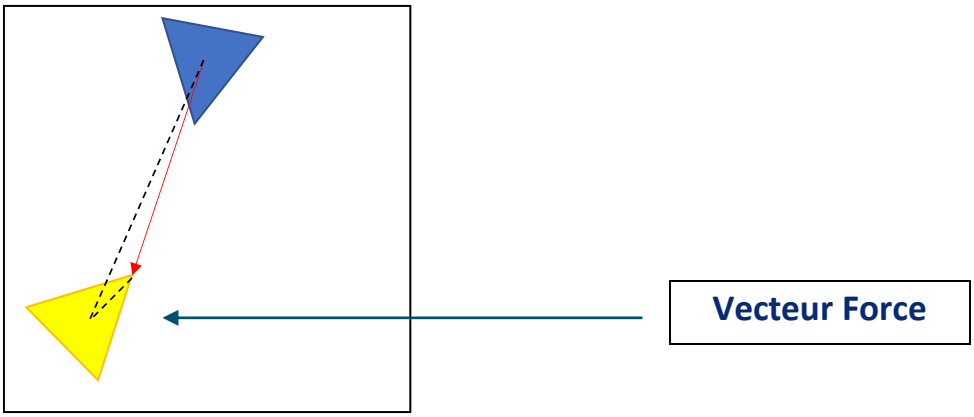
L'algorithme de conception de la force de répulsion est le suivant, il y aura dans le dossier Annexe une analyse plus précise du code :

```
Public Function repulsion(ByVal B As boid) As point
        Dim R As point
        Dim i As Integer
        Dim j As Integer
        Dim min As Integer
        Dim tmp As Double
        tmp = 0
        Dim T(,) As Double
        ReDim T(1, 9)
        For i = 0 To nbobstacles - 1
            T(0, i) = i
            T(1, i) = distance(B.p.x + B.d.x, B.p.y + B.d.y, obstacles(i).x,
obstacles(i).y)
        Next
        For j = 1 To nbobstacles - 1
            If T(1, j) < T(1, tmp) Then
                tmp = j
            End If
        Next
        min = T(0, tmp)
        If T(1, tmp) < 40 Then
            R.x = -(obstacles(min).x - B.p.x) + (B.d.x)
            R.y = -(obstacles(min).y - B.p.y) + (B.d.y)
        Else
            R.x = 0
            R.y = 0
        End If
        Return R
    End Function
```

Force de Separation

La force de séparation se base sur le même principe que la force de répulsion mais assure le fait qu'il ne puisse avoir de collisions entre les boids.

Nous avons également représenté le vecteur force nous permettant de mieux conceptualiser la force de séparation :



Cette force agit comme la force de répulsion et attire le boid vers l'arrière quand il risque de se cogner avec un autre.

L'algorithme de la force de séparation s'inspire donc de celui de la force de répulsion :

```
Public Function separation(ByVal B As boid) As point
        Dim R As point
        Dim i As Integer
        Dim j As Integer
        Dim min As Integer
        Dim tmp As Double
        tmp = 0
        Dim cpt As Double
        cpt = 0
        Dim T(,) As Double
        ReDim T(1, 99)
        For i = 0 To nbboids - 1

            If vision(B, boids(i), 25) = True Then
                T(0, cpt) = i
                T(1, cpt) = distance(B.p.x, B.p.y, boids(i).p.x, boids(i).p.y)
                cpt = cpt + 1
            End If
        Next

        If cpt = 0 Then
            R.x = 0
            R.y = 0
```

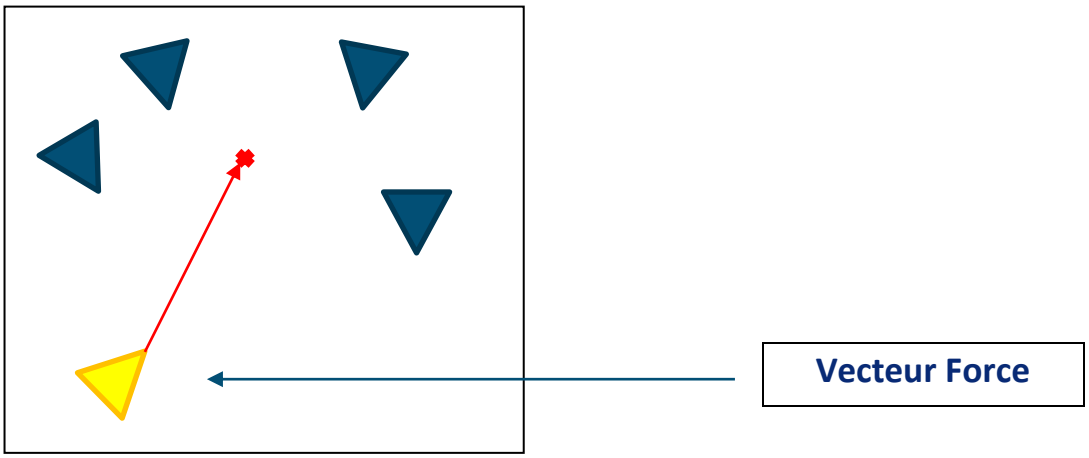
```
        Else
            For j = 1 To cpt - 1
                If T(1, j) < T(1, tmp) Then
                    tmp = j
                End If
            Next

            min = T(0, tmp)
            R.x = -(boids(min).p.x - B.p.x) + (B.d.x)
            R.y = -(boids(min).p.y - B.p.y) + (B.d.y)
        End If

        Return R
    End Function
```

Force de Cohésion

La force de cohésion qui est à l'origine du vol de groupe harmonieux des boids, se base sur le fait que les boids sont attirés les uns avec les autres tout en respectant la force de séparation. Elle s'appuie sur la fonction barycentre que nous avons définis dans la partie d'analyse et de conception de ce dossier.



Les boids convergent vers un point commun (barycentre), cette « convergence » caractérise la force de cohésion. Cette force aura donc pour but d'attirer le boid vers le barycentre du groupe de boids qui sont autour de lui.

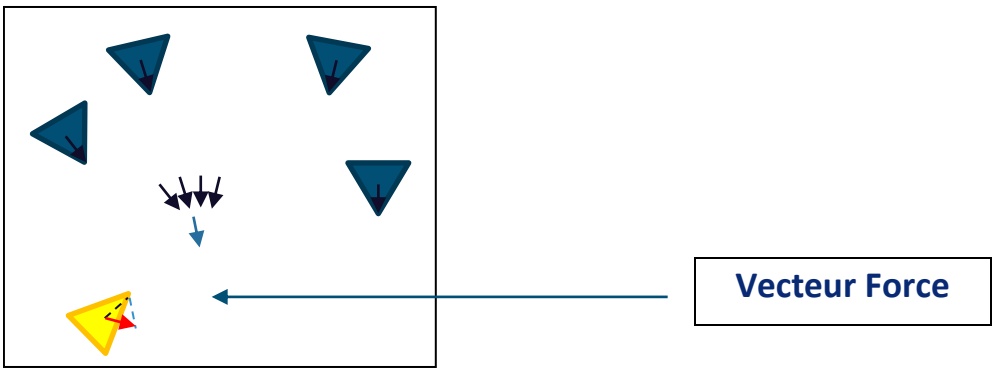
L'algorithme de conception est le suivant :

```
Public Function cohesion(ByRef B As boid) As point
        Dim R As point
        Dim i As Integer
        Dim cpt As Integer = 0
        Dim cptx As Double = 0
        Dim cpty As Double = 0
        For i = 0 To nbboids - 1
            If vision(B, boids(i), 80) = True Then

                cptx = cptx + boids(i).p.x
                cpty = cpty + boids(i).p.y
                cpt = cpt + 1
            End If
            If cpt <> 0 Then
                If distance(B.p.x, B.p.y, boids(i).p.x, boids(i).p.y) > 30 Then
                    Dim g As point
                    g.x = cptx / cpt
                    g.y = cpty / cpt
                    R.x = g.x - B.d.x
                    R.y = g.y - B.d.y
                    Return R
                Else
                    R.x = 0
                    R.y = 0
                    Return R
                End If
            Else
                R.x = 0
                R.y = 0
                Return R
            End If
        Next
    End Function
```

Force d'alignement

La force d'alignement suivie de la force de cohésion sont les deux forces majeures de notre projet. Ce sont en effet les deux forces qui modélisent les différents phénomènes du vol de groupe des boids. La force d'alignement est la force, comme son nom l'indique, qui aboutie à une certaine symétrie dans le vol des boids.
Le principe de cette force est de faire la moyenne des vecteurs directions du groupe de boids présents autour du boid concerné.



L'algorithme de conception est le suivant :

```
Public Function alignement(ByVal B As boid) As point
        Dim R As point
        Dim i As Integer
        Dim cpt As Integer = 1
        Dim cptx As Double = B.d.x
        Dim cpty As Double = B.d.y
        For i = 0 To nbboids - 1

            If vision(B, boids(i), 60) = True Then
                normaliser(boids(i).d)
                cptx = cptx + boids(i).d.x
                cpty = cpty + boids(i).d.y
                cpt = cpt + 1
            End If
        Next

        If cpt <> 0 And cpt <> 1 Then
            R.x = (cptx / cpt) + B.d.x
```

```
            R.y = (cpty / cpt) + B.d.y
            Return R
        Else
            R.x = 0
            R.y = 0
            Return R
        End If
    End Function
```

Nous pouvons à présent à partir de ces formules décrire le comportement du boid au cours du temps :

Son vecteur accélération est défini par la somme des forces qui s'appliquent sur le boid (c'est à dire les forces d'alignement, répulsion, cohésion et séparation), le tout divise par sa masse.

Ainsi à chaque pas du timer on exécute le code suivant :

```
Dim i As Integer
        For i = 0 To nbboids - 1
            boids(i).acc.x = (Val(hssep.Value) * separation(boids(i)).x + Val(hsco.Value)
* cohesion(boids(i)).x + Val(hsali.Value) * alignement(boids(i)).x + Val(hsrep.Value) *
repulsion(boids(i)).x) / boids(i).masse
            boids(i).acc.y = (Val(hssep.Value) * separation(boids(i)).y + Val(hsco.Value)
* cohesion(boids(i)).y + Val(hsali.Value) * alignement(boids(i)).y + Val(hsrep.Value) *
repulsion(boids(i)).y) / boids(i).masse

            Dim B As boid
            B.d.x = boids(i).d.x + boids(i).acc.x
            B.d.y = boids(i).d.y + boids(i).acc.y
            normaliser(B.d)
            boids(i).d = B.d

            boids(i).vitesse = vitesse(boids(i))
            boids(i).p = mouvement(boids(i))
            Dim k As Integer
            For k = 0 To nbarbres - 1
                mange(boids(i), arbres(k))
            Next
            mort_repas(boids(i))

            lblboidsmorts.Text = Str(boidsmorts)
            lblnbboids.Text = Str(hsnbboids.Value)
            lblboids.Text = Str(hsnbboids.Value - boidsmorts)
        Next
```

Son vecteur direction est défini par la formule de récurrence suivante :

Boid.direction = Boid.direction + Boid. Acceleration

Son vecteur position est défini de la même manière :

Boid.position = Boid.position + Boid.vitesse

Son vecteur vitesse étant lui-même défini comme tel :

Boid.vitesse = Boid.vitesse + Boid. Acceleration

La fonction vitesse ressort un point Xv : vecteur vitesse du Boid b.

```
Function vitesse(ByVal b As boid) As point
        Dim Xv As point
        Xv.x = b.vitesse.x + b.acc.x
        Xv.y = b.vitesse.y + b.acc.y
        normaliser(Xv)
        Return Xv
    End Function
```

Pour utiliser ces formules on donne des valeurs initiales à chacune de ces suites récurrentes lors de la création des boids.

```
                boids(i).p.x = plateauX * rdm.NextDouble()
                boids(i).p.y = plateauY * rdm.NextDouble()
                boids(i).d.x = rdm.NextDouble()
                boids(i).d.y = rdm.NextDouble()
                boids(i).vitesse.x = rdm.NextDouble()
                boids(i).vitesse.y = rdm.NextDouble()
                boids(i).acc.x = 1
                boids(i).acc.y = 1
                boids(i).repas = 0
                boids(i).id = i
                Randomize()
                boids(i).masse = Int(5 * Rnd()) + 3
```

Nous devons prendre en compte les coordonnées négatives des boids par conséquent nous discernons les différents cas possibles afin de trouver une alternative à la fonction «modulo» qui n'est pas adaptée ici.

Par ailleurs afin d'obtenir un mouvement torique de la part des boids on ajoute quelques conditions à la méthode qui calcule sa position :

```
Public Function mouvement(ByVal b As boid) As point
        Dim Xp As point
        Xp.x = (b.p.x + vitesse(b).x)
        Xp.y = (b.p.y + vitesse(b).y)
        If Xp.x < 0 Then
            Xp.x = (plateauX + (b.p.x + vitesse(b).x)) Mod plateauX
        Else
            Xp.x = (b.p.x + vitesse(b).x) Mod plateauX
        End If
        If Xp.y < 0 Then
            Xp.y = (plateauY + (b.p.y + vitesse(b).y)) Mod plateauY
        Else
            Xp.y = (b.p.y + vitesse(b).y) Mod plateauY
        End If
        Return Xp
    End Function
```

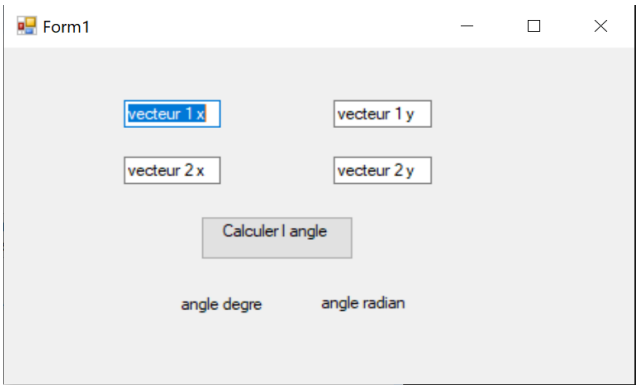
JEUX DE TEST

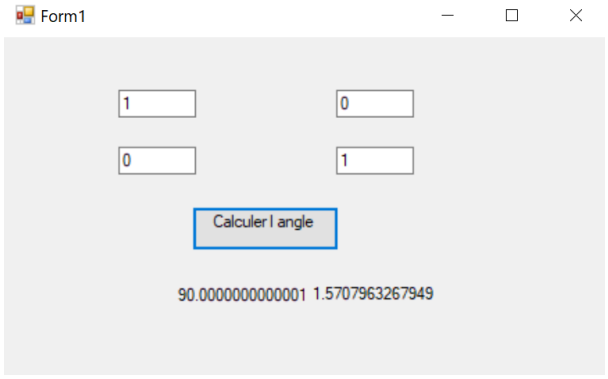
Nous continuons notre dossier avec une étape indispensable, en effet nous faisons part de différents jeux de test des fonctions de base que l'on a énoncé dans la première partie du dossier.

Fonction de calcul d'angle

Cette fonction ressort la valeur de l'angle en degré entre 2 vecteurs définis par 2 points :
p1 et p2.
On la définit avec l'algorithme ci-dessous :

```
Function calculangle(ByVal x1 As Double, ByVal x2 As Double, ByVal y1 As Double, ByVal y2
As Double) As Double
        Return Math.Acos((x1 * x2 + y1 * y2) / (Math.Sqrt(x1 ^ 2 + y1 ^ 2) * Math.Sqrt(x2
^ 2 + y2 ^ 2))) * 180 / PI
    End Function
```





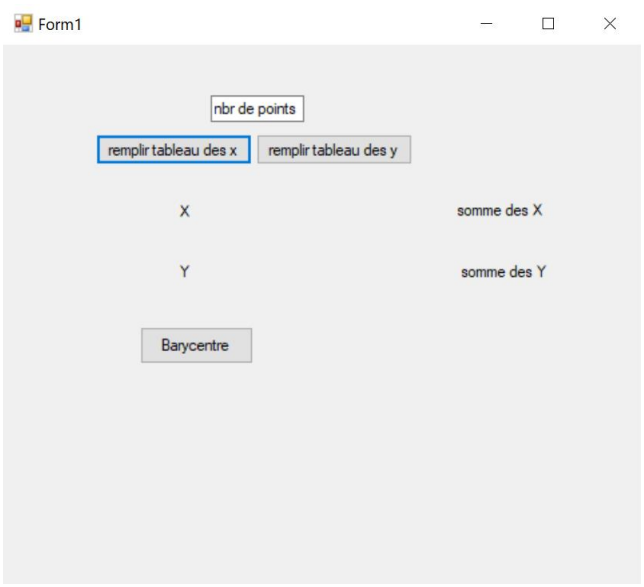
Vérification :

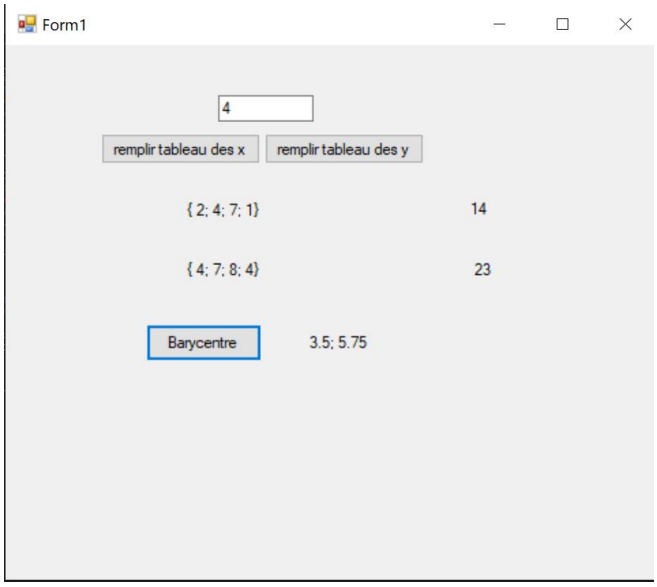
Malgré un faible taux d'erreurs, le résultat semble cohérent puisque : $\pi/2 = 1{,}5707\ldots$

Fonction de calcul du barycentre

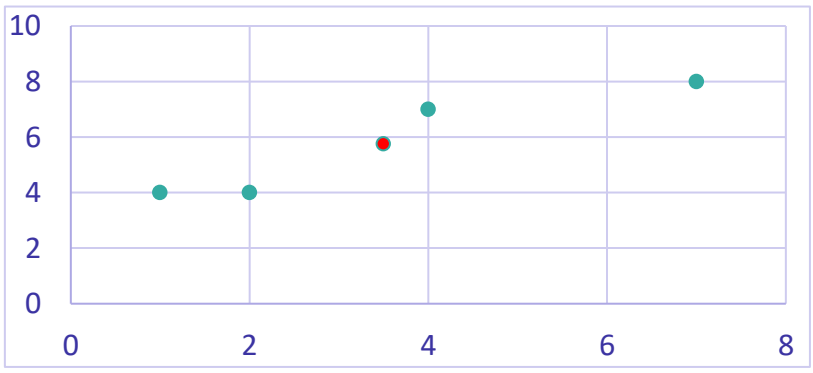
Cette fonction est utilisée à l'intérieur des lignes de codes de la fonction force de cohésion, il correspond en effet au centre de gravité d'un groupe de boid, son algorithme est le suivant :

```
Function barycentre(ByVal tabx() As Integer, ByVal taby() As Integer, ByVal TailleT As
Integer) As point
        Dim sumx As Double
        sumx = somme(tabx, TailleT)
        Dim sumy As Double
        sumy = somme(taby, TailleT)
        Dim G As point
        G.x = sumx / TailleT
        G.y = sumy / TailleT
        Return G
End Function
```





Les résultats sont cohérents et on peut de plus appuyer les calculs par une représentation graphique qui justifie bien le calcul trouvé.

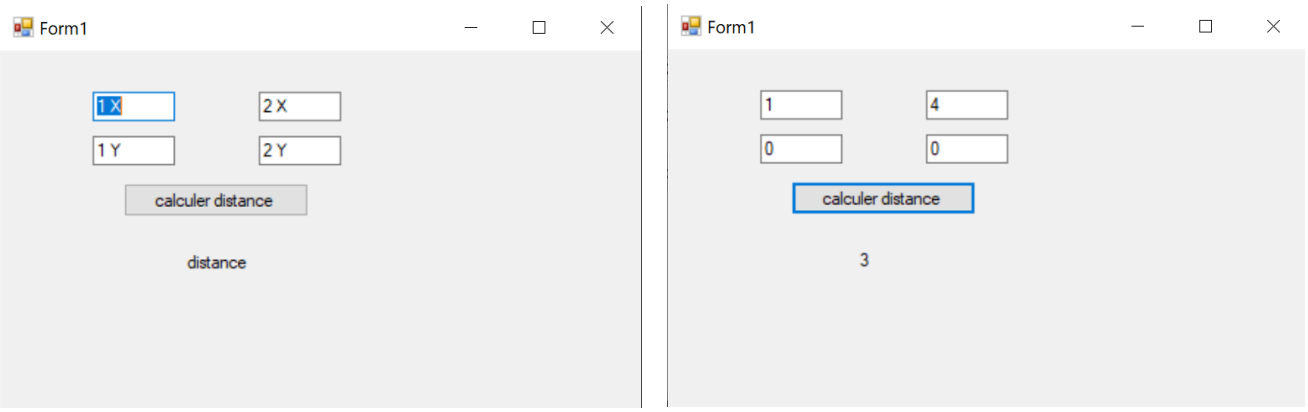


Fonction calcul de distance

La fonction distance qui est primordiale dans le bon fonctionnement du code est définie par l'algorithme ci-dessous :

On la modifiera pour l'intégrer au code afin de réaliser un meilleur mouvement torique.

```
Function distance(ByVal p1x As Double, ByVal p1y As Double, ByVal p2x As Double, ByVal p2y
As Double) As Double
        Return Math.Sqrt((p2x - p1x) ^ 2 + (p2y - p1y) ^ 2)
End Function
```



Le résultat est juste, nous avons conclu que l'algorithme était bien valide et nous l'avons utilisé pour la suite du développement du code.

PERSONNALISATION DU PROGRAMME

Les cerises

Nous avons décidé d'ajouter au comportement de base des Boids quelques fonctionnalités. Comme dit précédemment nous avons créé des cerises positionnées dans les arbres. Quand un Boid passe sur une cerise, il la mange, son compteur repas est incrémenté et la cerise disparait puisqu'elle a été mangée.

Une règle s'applique alors sur le Boid : il ne doit pas manger plus de cerises que la valeur de sa masse (entier entre 3 et 5) car sinon il mourra.

Au niveau de la programmation :

Afin de faire disparaitre les cerises en évitant tout problème dans notre programme nous avons décidé de programmer leur disparition sur le plateau en échangeant dans le tableau des cerises de chaque arbre la cerise mangée et la dernière cerise du tableau. Ensuite on décrémente le nombre de cerises.

```
'en passant sur une cerise un boid la mange
    'la cerise disparait
    'un boid peut manger plusieurs cerises a la fois
    Sub mange(ByRef b As boid, ByRef A As base_arbre)
        Dim d As Integer
        d = A.nbcerises
        Dim j As Integer
        For j = 0 To A.nbcerises - 1
            'si un boid passe sur une cerise alors il la mange et celle ci disparait :
            'on modifie le tableau des cerises de l arbre A en supprimant la case de la cerise
            If b.p.x >= A.cerises(j).x - 3 And b.p.x <= A.cerises(j).x + 3 Then
                If b.p.y <= A.cerises(j).y + 3 And b.p.y >= A.cerises(j).y - 3 Then
                    'a la place de la cerise mangee on met la derniere cerise du tableau de
cerises de l arbre :
                    A.cerises(j) = A.cerises(A.nbcerises - 1)
                    A.nbcerises = A.nbcerises - 1
                End If
            End If
        Next
        Dim c As Integer
        c = d - A.nbcerises
        b.repas = b.repas + c
    End Sub
```

Ainsi dès qu'un Boid a mangé trop de cerises, il meurt. Il disparait de la même manière qu'une cerise le fait : on échange dans le tableau des Boids le Boid en question et le dernier Boid du tableau, puis on décrémente le nombre de Boids.

```
'on ajoute dans la structure boid : Dim repas As Integer 'compte le nbr de cerises mangees
    'un boid ne doit pas avoir mange + de cerises que sa masse sinon il meurt
    'qd un boid meurt : un autre nait
    Sub mort_repas(ByVal b As boid)
        If b.repas >= b.masse Then
            Dim indice As Integer
            indice = b.id
            'le boid meurt :
            'a la place du boid mort on met le dernier boid du tableau :
            boids(indice) = boids(nbboids - 1)
            boidsmorts = boidsmorts + 1
            nbboids = nbboids - 1
        End If
```

```
    End Sub
```

Les rapaces

Les rapaces entre eux sont soumis aux mêmes règles que les Boids sauf celle de cohésion. De plus leur direction est spécifique, ils vont toujours en direction du Boid le plus proche d'eux.

```
'Retourne imin l indice du boid le plus proche
    Public Function BoidLePlusProche(ByVal r As rapace) As Integer
        Dim iMin As Integer
        Dim i As Integer
        Dim q As Integer
        q = r.cible
        'si il existe une cible + proche que celle de r alors la fonction retournera cette
nouvelle cible :
        iMin = q
        For i = 0 To nbboids - 1
            If i <> q Then
                If distance(r.p.x, r.p.y, boids(i).p.x, boids(i).p.y) <= distance(r.p.x, r.p.y,
boids(iMin).p.x, boids(iMin).p.y) Then
                    iMin = i
                End If
            End If
        Next
        Return iMin
    End Function
```

```
'retounre le vecteur direction du boid
    Function comportementcible(ByRef r As rapace) As point
        Dim D As point
        'Aller vers la cible
        D.x = boids(BoidLePlusProche(r)).p.x - r.p.x
        D.y = boids(BoidLePlusProche(r)).p.y - r.p.y
        Return D
    End Function
```

De façon équivalente au système des cerises nous avons ajoute au programme la possibilité pour les Boids de se faire manger par des rapaces. Ces derniers sont représentés de la même manière que les Boids mais de couleur grise (tandis que les Boids sont dessines en jaune) et un peu plus massifs.
Leur mort est programmée de la même façon que quand ils ont trop mangé de cerises.

```
'en passant sur un boid le rapace le mange
    'cette procedure modifie le tableau de boids :
    Sub mangeboid(ByRef r As rapace, ByRef boids() As boid)
        Dim j As Integer
        For j = 0 To nbboids - 1
            'si un rapace passe sur un boid alors il le mange et celui ci disparait :
            If r.p.x >= boids(j).p.x - 3 And r.p.x <= boids(j).p.x + 3 Then
                If r.p.y <= boids(j).p.y + 3 And r.p.y >= boids(j).p.y - 3 Then

                    'a la place du boid mort on met le dernier boid du tableau :
                    boids(j) = boids(nbboids - 1)

                    boidsmorts = boidsmorts + 1
                    nbboids = nbboids - 1
                End If
            End If
        Next
    End Sub
```

Conclusion

Nous avons donc après de nombreuses heures de travail finalisé notre projet qui avait pour but de conceptualiser à travers un logiciel informatique le vol d'oiseaux nommé boids.

Nous avons, afin de se rapprocher le plus près du réel possible, ajouté de nombreuses fonctionnalités supplémentaires à notre projet. En effet nous avons modélisé un système dominant/dominé : les oiseaux sont confrontés à des rapaces (prédateurs) et doivent modifier leur comportement. A cela s'ajoute un système de nutrition, les oiseaux se nourrissent de cerises mais ne doivent pas en consommer trop.