UNIVERSITE PROTESTANTE AU CONGO

Faculté des sciences informatiques

B.P 4745

KINSHASA/LINGWALA

TP6
PROGRAMMATION SYSTÈME
L1 FASI

Présenté par :

NGOMA MAKWAMA ALAIN

PROF. DR. Patrick MUKALA

ANNEE – ACADEMIQUE 2020 - 2021

TP 6

« Après traduction dans la langue française »

LES QUESTIONS

Dans cette leçon, nous utiliserons les verrous et sémaphores C # pour synchroniser les threads. Tout d'abord, nous continuons avec l'application de la leçon 3. Dans cette application, il peut être partagé variables qui sont utilisées par plusieurs threads (cela dépend de la façon dont vous programmé l'affectation).

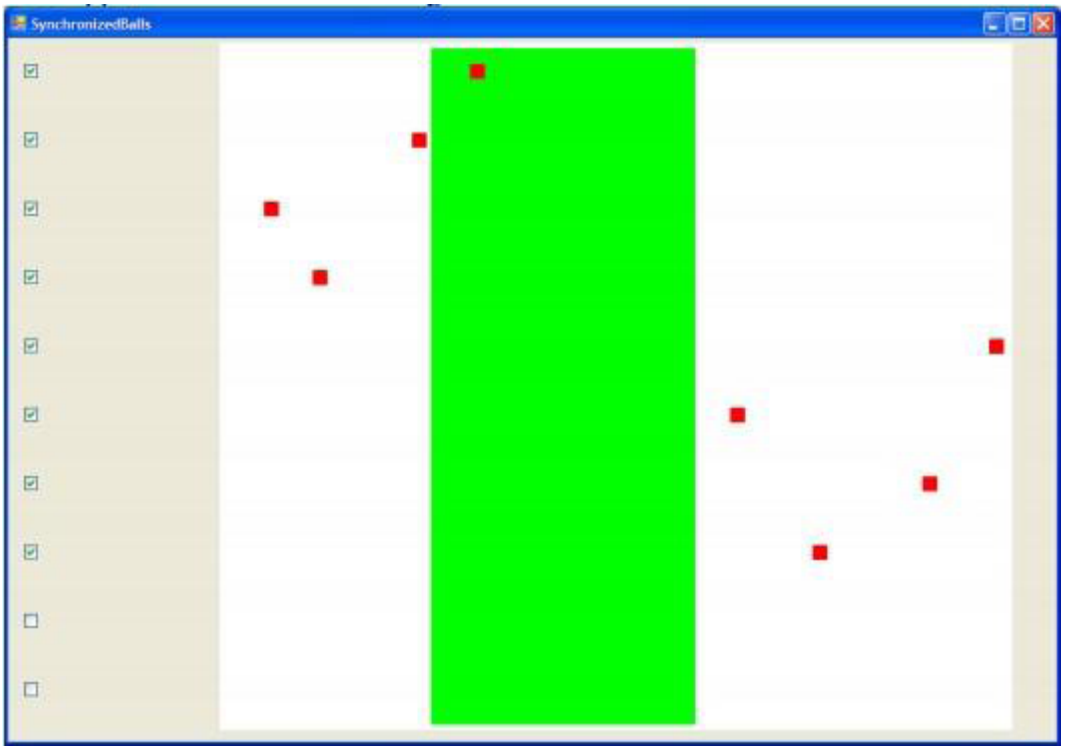
Comme nous l'avons appris cette semaine, cela est intrinsèquement dangereux, nous devons donc résoudre ce problème.

a. Identifiez les variables partagées et assurez-vous qu'elles sont utilisées de manière thread-safe (cela signifie que toujours au plus un thread à la fois est autorisé à utiliser ces variables). Utilisez des verrous pour ce faire.

b. Aussi, changez le programme de telle sorte qu'au plus un thread à la fois puisse mettre des caractères dans la zone de texte. Utilisez des verrous pour cela également.

Ensuite, nous utilisons une nouvelle application SynchronizedBalls dans laquelle les problèmes de concurrence peuvent être affichés de manière plus graphique.

Cette application à l'écran suivant :



C:\Users\HP\AppData\Local\Microsoft\Windows\INetCache\Content.Word\image tp.jpg

Sur la gauche, nous voyons 10 cases à cocher. Chaque fois qu'une case est cochée, un nouveau thread doit être démarré, cela fait bouger une balle de gauche à droite dans la zone blanche. Lorsqu'une case à cocher est décochée, le thread doit être interrompu.

La zone verte représente la section critique, qui joue un rôle spécial par la suite.

L'application a les classes suivantes:

SynchronisationTestForm

Ceci est la forme. Chaque fois qu'une case à cocher change d'état, la méthode checkBox_CheckedChanged est appelée.

BallMover

Cette classe contient la méthode Run, qui doit être exécutée par chaque thread. Dans cette méthode, une balle est déplacée de gauche à droite sur l'écran. Quand la balle est au loin côté droit de la zone blanche, il est replacé sur le côté gauche. Les balles sont boîtes d'image sll.

c. Changez le programme donné de telle sorte que le comportement décrit ci-dessus soit mis en œuvre. Afin de pouvoir interrompre un thread lorsque la case à cocher est en cours décochée, les threads créés doivent être placés dans le tableau ta, qui est déjà défini dans la classe SynchronisationTestForm.
Assurez-vous que tous les threads s'arrêtent automatiquement lorsque la fenêtre principale est fermée.

d. Donnez à chaque balle une vitesse choisie au hasard. Cette vitesse doit être comprise entre environ 100 et 200 pixels par seconde (donc le Thread.Sleep doit être compris entre 5 et 10msec). Vous pouvez utiliser la classe Random pour cela.

e. Identifiez quel morceau de code correspond exactement à la section critique.

f. Changer le programme de telle sorte qu'au plus un thread se trouve dans la section critique des toutes fois.

Utilisez un sémaphore pour cela. Puisque ce sémaphore devra être partagé par tous threads, créez-le dans la classe SynchronisationTestForm et donnez le constructeur de BallMover un argument supplémentaire de type Semaphore (et la classe BallMover un attribut supplémentaire de type Semaphore) pour le faire connaître les fils aussi.
Assurez-vous que cela fonctionne également bien lorsqu'un thread est interrompu dans son Critical Section.

g. Changez le programme de telle sorte qu'au plus trois threads se trouvent dans la section critique à chaque fois.

Enregistrez cette version.
Nous allons maintenant implémenter le problème des lecteurs / écrivains. Pour cela, nous avons besoin de deux types de fils, qui seront représentés par deux couleurs de balle différentes:

• rouge (lecteurs) et

• bleu (écrivains).

h. Changez le programme de sorte que les 5 premières boules soient rouges et les 5 dernières boules soient bleues.

i. Utilisez également 2 méthodes Run dans la classe BallMover au lieu de 1.
Nommez-les RunReader et RunWriter. Laisse les boules rouges s'exécuter RunReader et les boules bleues exécutent RunWriter.

j. Prenez la solution au problème des lecteurs / écrivains qui a été donnée dans les feuilles (en utilisant des sémaphores wrt et mutex, et un entier readcount) et le faire fonctionner dans cette application.
Les sémaphores peuvent être traités de la même manière que dans la question f.

Pour simplifier les choses, vous pouvez supposer qu'un thread ne sera jamais arrêté à l'intérieur de la zone verte, vous n'avez donc pas à gérer cette situation.

LES RESOLUTIONS

Après avoir murement réfléchi et travaillé d'arrache-pied dans un commun accord, en voici les différentes résolutions du présent T.P

Résolution assertion A

Identifiez les variables partagées et assurez-vous qu'elles sont utilisées de manière thread-safe (cela signifie que toujours au plus un thread à la fois est autorisé à utiliser ces variables).
Utilisez des verrous pour ce faire.

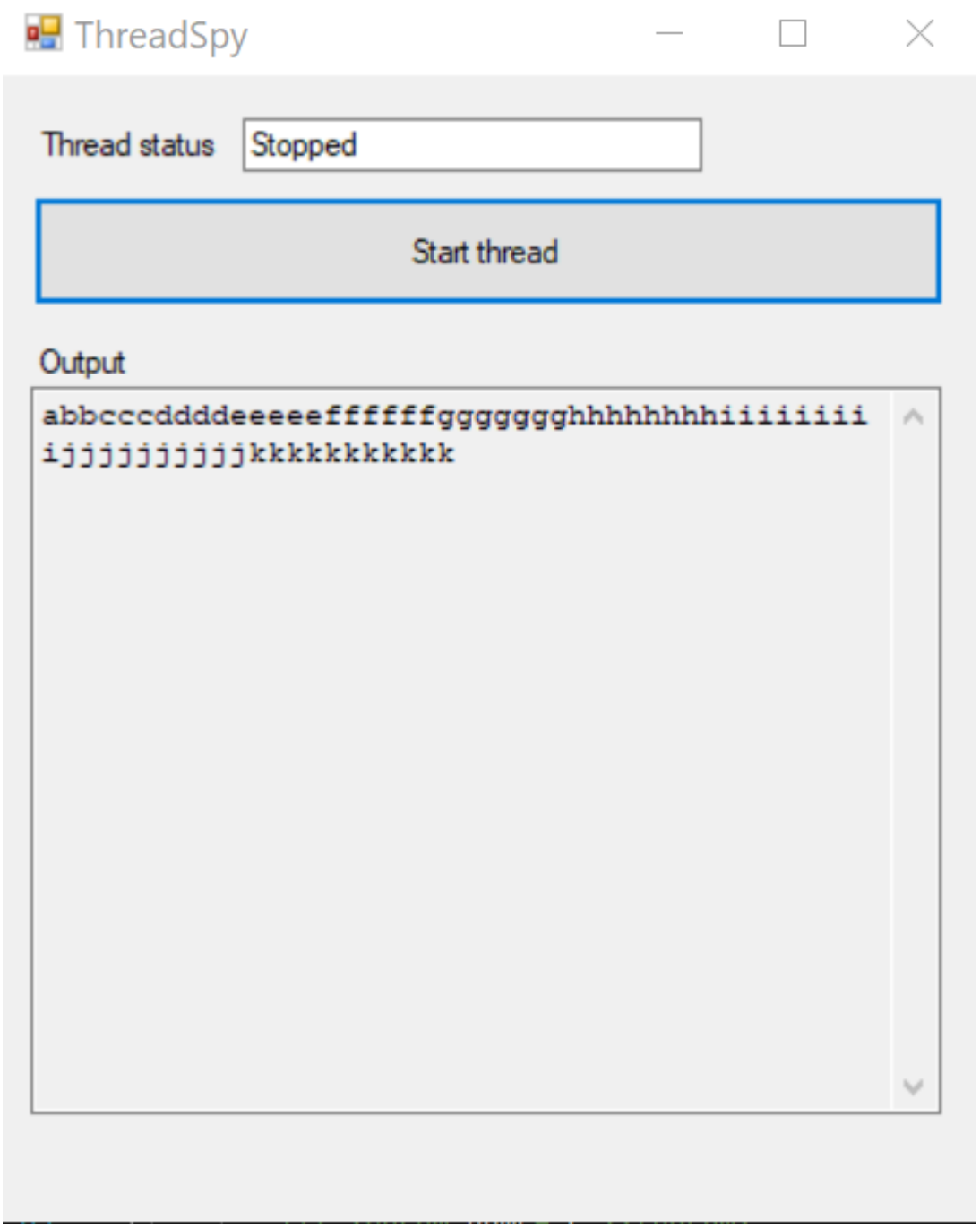
```
private readonly object sharedVariableLock = new object();
private readonly object textBoxLock = new object();
```

Résolution assertion B

a. Aussi, changez le programme de telle sorte qu'au plus un thread à la fois puisse mettre des caractères dans la zone de texte. Utilisez des verrous pour cela également.
Ensuite, nous utilisons une nouvelle application SynchronizedBalls dans laquelle les problèmes de concurrence peuvent être affichés de manière plus graphique.

```
private void Run(Object obj)
{
    if (obj != null)
    {
        lock (textBoxLock)
            for (int i = 0; i < NrOfChars; i++)
            {
                Thread.Sleep(300);
                TextBoxHelper.AddChar(tb, (char) obj);

            }
        NrOfChars++;
    }
}
```

```
private void ButtonStartThread_Click(object sender, EventArgs e)
{
    lock (sharedVariableLock)
        if (threadCount < max)
        {
            threads[threadCount].Start((char)tab[threadCount]);
        }
    threadCount++;
}
```


ThreadSpy
Thread status
Stopped
Start thread
Output
abbcccddddeeeeeffffffgggggggghhhhhhhhiiiiiiiii
ijjjjjjjjjjkkkkkkkkkk

Résolution assertion C

Changez le programme donné de telle sorte que le comportement décrit ci-dessus soit mis en œuvre. Afin de pouvoir interrompre un thread lorsque la case à cocher est en cours décochée, les threads créés doivent être placés dans le tableau ta, qui est déjà défini dans la classe SynchronisationTestForm.

Assurez-vous que tous les threads s'arrêtent automatiquement lorsque la fenêtre principale est fermée.

```
private void checkBox_CheckedChanged(object sender, EventArgs e)
{
    // index is the number of the CheckBox that was clicked
    // This index is derived from the y-position of the CheckBox
    int index = (((CheckBox)sender).Location.Y - 25) / 65;

    // pb is the PictureBox that belongs to this CheckBox
    PictureBox pb = pba[index];


    if (((CheckBox)sender).Checked)
    // The CheckBox was checked, so
    // pb must get a red background color and
    // a new thread, that will move pb, must be created and put into ta[index]
    {
        pb.BackColor = Color.Red;
        BallMover ball = new BallMover(pb);
        ThreadStart ts = new ThreadStart(ball.Run);
        ta[index] = new Thread(ts)
        {
            IsBackground = true
        };
        //ta[index].IsBackground = true;
        ta[index].Start();
        // TODO create thread
    }
      else
      // The CheckBox was unchecked, so
      // the corresponding thread must be interrupted and
      // pb must get transparant background color
      {
          pb.BackColor = Color.Transparent;
          ta[index].Interrupt();

          // TODO interrupt thread
      }
  }
```

Résolution assertion D

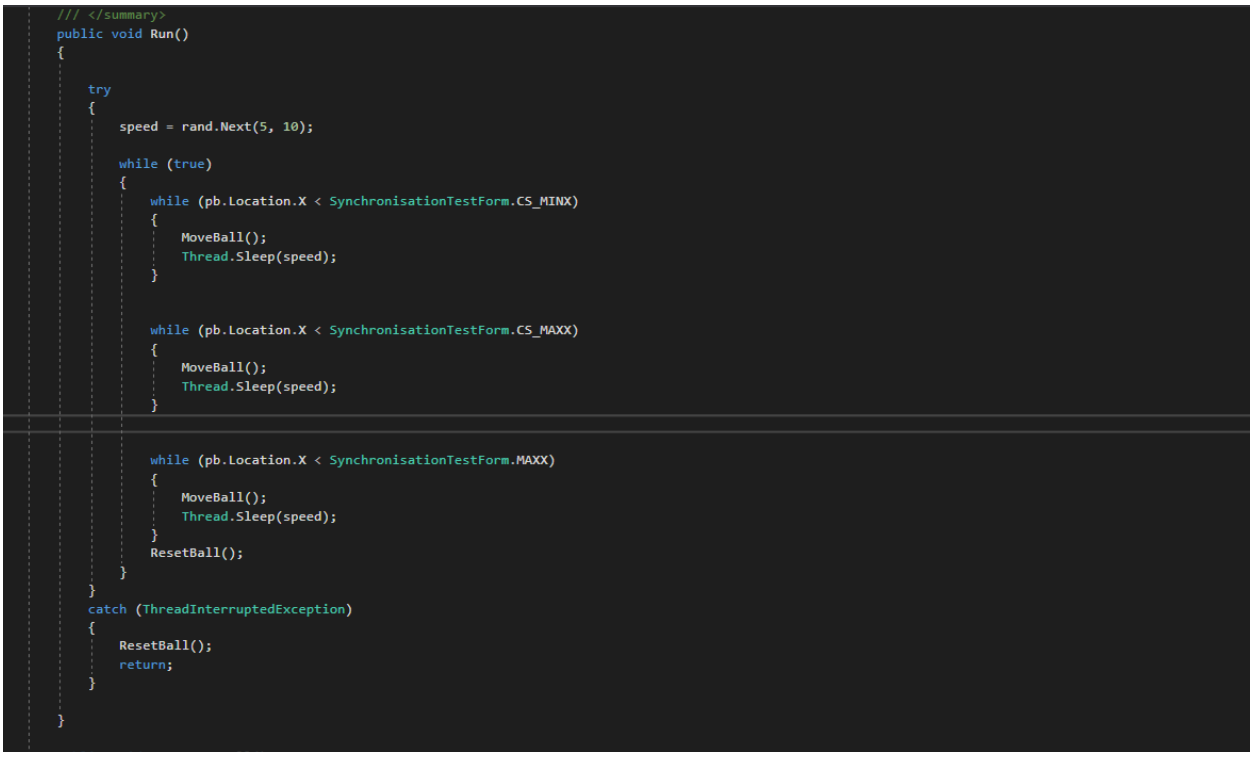
Donnez à chaque balle une vitesse choisie au hasard. Cette vitesse doit être comprise entre environ 100 et 200 pixels par seconde (donc le Thread.Sleep doit être compris entre 5 et 10msec). Vous pouvez utiliser la classe Random pour cela.

```
namespace SynchronizedBalls
{

    /// <summary>
    /// </summary>
    public class BallMover
    {
        private delegate void UpdatePictureBoxCallback(Point p);

        private PictureBox pb;
        private Random rand = new Random();
        private int speed;

        public BallMover(PictureBox pb)
        {
            this.pb = pb;
        }
```

```
    /// </summary>
    public void Run()
    {

        try
        {
            speed = rand.Next(5, 10);

            while (true)
            {
                while (pb.Location.X < SynchronisationTestForm.CS_MINX)
                {
                    MoveBall();
                    Thread.Sleep(speed);
                }


                while (pb.Location.X < SynchronisationTestForm.CS_MAXX)
                {
                    MoveBall();
                    Thread.Sleep(speed);
                }


                while (pb.Location.X < SynchronisationTestForm.MAXX)
                {
                    MoveBall();
                    Thread.Sleep(speed);
                }
                ResetBall();
            }
        }
        catch (ThreadInterruptedException)
        {
            ResetBall();
            return;
        }

    }
```

Résolution assertion E

Identifiez quel morceau de code correspond exactement à la section critique.



```
/// </summary>
public void Run()
{

    try
    {
        speed = rand.Next(5, 10);

        while (true)
        {
            while (pb.Location.X < SynchronisationTestForm.CS_MINX)
            {
                MoveBall();
                Thread.Sleep(speed);
            }


            while (pb.Location.X < SynchronisationTestForm.CS_MAXX)
            {
                MoveBall();
                Thread.Sleep(speed);
            }


            while (pb.Location.X < SynchronisationTestForm.MAXX)
            {
                MoveBall();
                Thread.Sleep(speed);
            }
            ResetBall();
        }
    }
    catch (ThreadInterruptedException)
    {
        ResetBall();
        return;
    }

}
```

Résolution assertion F

Changer le programme de telle sorte qu'au plus un thread se trouve dans la section critique des toutes fois.
Utilisez un sémaphore pour cela. Puisque ce sémaphore devra être partagé par tous threads, créez-le dans la classe SynchronisationTestForm et donnez le constructeur de BallMover un argument supplémentaire de type Semaphore (et la classe BallMover un attribut supplémentaire de type Semaphore) pour le faire connaître les fils aussi.
Assurez-vous que cela fonctionne également bien lorsqu'un thread est interrompu dans son Critical Section.

```
public partial class SynchronisationTestForm : Form
{
    public const int MINX = 0;
    public const int MAXX = 750;

    public const int CS_MINX = 200;
    public const int CS_MAXX = 450;

    private static Semaphore sem = new Semaphore(1, 1, "BallMoverApp");

    private PictureBox[] pba = new PictureBox[10];
    private Thread[] ta = new Thread[10];
```

Résolution assertion G

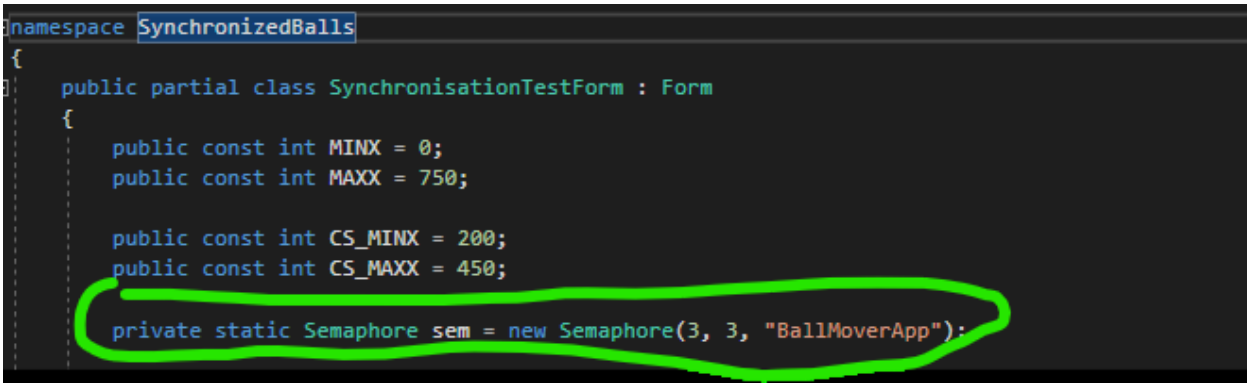
Changez le programme de telle sorte qu'au plus trois threads se trouvent dans la section critique à chaque fois.

Enregistrez cette version.

Nous allons maintenant implémenter le problème des lecteurs / écrivains. Pour cela, nous avons besoin de deux types de fils, qui seront représentés par deux couleurs de balle différentes:

• rouge (lecteurs) et

• bleu (écrivains).



```
namespace SynchronizedBalls
{
    public partial class SynchronisationTestForm : Form
    {
        public const int MINX = 0;
        public const int MAXX = 750;

        public const int CS_MINX = 200;
        public const int CS_MAXX = 450;

        private static Semaphore sem = new Semaphore(3, 3, "BallMoverApp");
```

Ici, il reste à comprendre que la suite de l'assertion F &G sont tous similaires

```
private void checkBox_CheckedChanged(object sender, EventArgs e)
{
    // index is the number of the CheckBox that was clicked
    // This index is derived from the y-position of the CheckBox
    int index = (((CheckBox)sender).Location.Y - 25) / 65;

    // pb is the PictureBox that belongs to this CheckBox
    PictureBox pb = pba[index];

    BallMover ball = new BallMover(pb, sem);

    if (((CheckBox)sender).Checked)
    // The CheckBox was checked, so
    // pb must get a red background color and
    // a new thread, that will move pb, must be created and put into ta[index]
    {

        pb.BackColor = Color.Red;
        ThreadStart ts = new ThreadStart(ball.Run);
        ta[index] = new Thread(ts)
        {
            IsBackground = true
        };
        //ta[index].IsBackground = true;
        ta[index].Start();

        // TODO create thread
    }
```

```
    else
    // The CheckBox was unchecked, so
    // the corresponding thread must be interrupted and
    // pb must get transparant background color
    {
        pb.BackColor = Color.Transparent;
        ball.InterruptBall();
        ta[index].Interrupt();

        // TODO interrupt thread
    }
}
```

```
public class BallMover
{
    private delegate void UpdatePictureBoxCallback(Point p);

    private PictureBox pb;
    private Semaphore sm;
    private Random rand = new Random();
    private int speed;

    public BallMover(PictureBox pb, Semaphore sm)
    {
        this.pb = pb;
        this.sm = sm;
    }

    //public BallMover(PictureBox pb)
    //{
    //    this.pb = pb;

    //}
```

```
/// </summary>
public void Run()
{

    try
    {
        speed = rand.Next(5, 10);

        while (true)
        {
            while (pb.Location.X < SynchronisationTestForm.CS_MINX)
            {
                MoveBall();
                Thread.Sleep(speed);
            }

            sm.WaitOne();
            while (pb.Location.X < SynchronisationTestForm.CS_MAXX)
            {
                MoveBall();
                Thread.Sleep(speed);
            }
            sm.Release();

            while (pb.Location.X < SynchronisationTestForm.MAXX)
            {
                MoveBall();
                Thread.Sleep(speed);
            }
            ResetBall();
        }
    }
    catch (ThreadInterruptedException)
    {
        ResetBall();
        return;
    }
```

```
public void InterruptBall()
{
    if (pb.Location.X < SynchronisationTestForm.CS_MAXX && pb.Location.X > SynchronisationTestForm.CS_MINX)
    {
        sm.Release();
        ResetBall();
    }
    else
    {
        ResetBall();
    }

}
```

Résolution assertion H

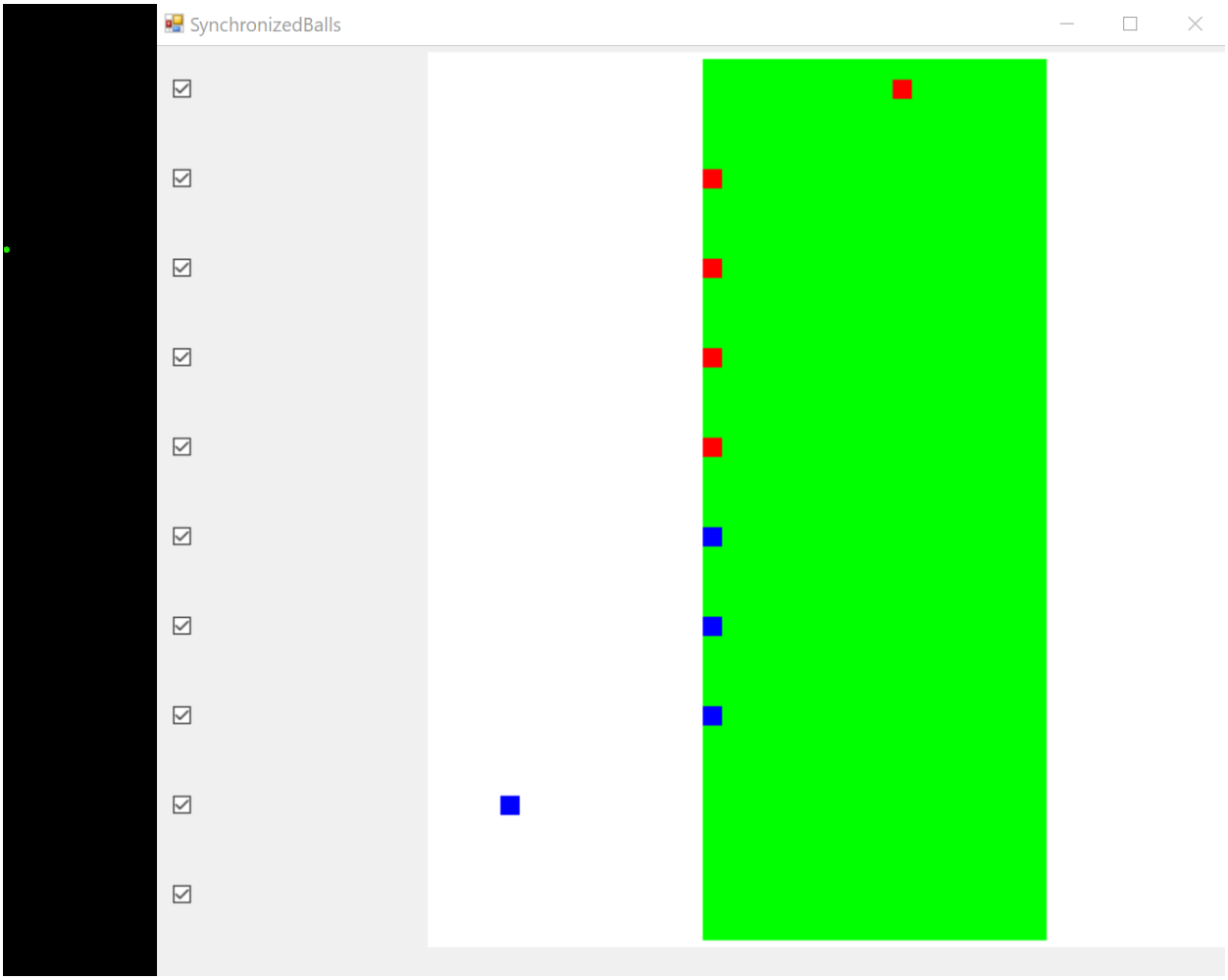
Changez le programme de sorte que les 5 premières boules soient rouges et les 5 dernières boules soient bleues.

```
if (((CheckBox)sender).Checked)
// The CheckBox was checked, so
// pb must get a red background color and
// a new thread, that will move pb, must be created and put into ta[index]
{
    //the first 5 balls are red, and the last 5 balls are blue
    // red balls execute RunReader
    if (index <= 4)
    {
        pb.BackColor = Color.Red;
        ta[index] = new Thread(readerBall.RunReader);

    }
    // blue balls execute RunWriter
    else if (index > 4)
    {
        pb.BackColor = Color.Blue;
        ta[index] = new Thread(writerBall.RunWriter);
    }
    ta[index].IsBackground = true;
    ta[index].Start();

    // TODO create thread
```

Résolution assertion I

Utilisez également 2 méthodes Run dans la classe BallMover au lieu de 1.

Nommez-les RunReader et RunWriter. Laisse les boules rouges s'exécuter RunReader et les boules bleues exécutent RunWriter.



Résolution assertion J

Prenez la solution au problème des lecteurs / écrivains qui a été donnée dans les feuilles (en utilisant des sémaphores wrt et mutex, et un entier readcount) et le faire fonctionner dans cette application.
Les sémaphores peuvent être traités de la même manière que dans la question f.

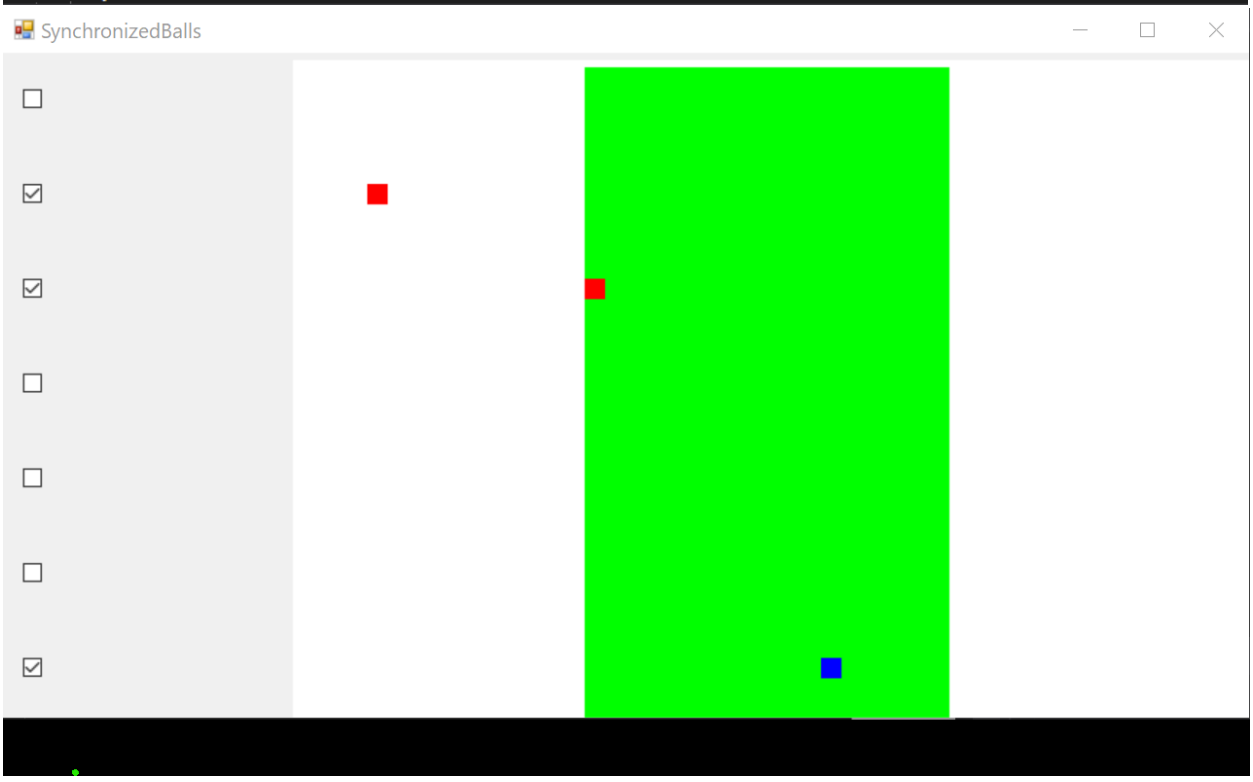
```
public partial class SynchronisationTestForm : Form
{
    public const int MINX = 0;
    public const int MAXX = 750;

    public const int CS_MINX = 200;
    public const int CS_MAXX = 450;

    private static Semaphore wrt_sem = new Semaphore(1, 1);
    private static Semaphore rdr_sem = new Semaphore(5, 5);


    private PictureBox[] pba = new PictureBox[10];
    private Thread[] ta = new Thread[10];
```

```
private void checkBox_CheckedChanged(object sender, EventArgs e)
{
    // index is the number of the CheckBox that was clicked
    // This index is derived from the y-position of the CheckBox
    int index = (((CheckBox)sender).Location.Y - 25) / 65;

    // pb is the PictureBox that belongs to this CheckBox
    PictureBox pb = pba[index];

    BallMover readerBall = new BallMover(pb, wrt_sem, rdr_sem);
    BallMover writerBall = new BallMover(pb, wrt_sem, rdr_sem);

    if (((CheckBox)sender).Checked)
    // The CheckBox was checked, so
    // pb must get a red background color and
    // a new thread, that will move pb, must be created and put into ta[index]
    {
        //the first 5 balls are red, and the last 5 balls are blue
        // red balls execute RunReader
        if (index <= 4)
        {
            pb.BackColor = Color.Red;
            ta[index] = new Thread(readerBall.RunReader);

        }
        // blue balls execute RunWriter
        else if (index > 4)
        {
            pb.BackColor = Color.Blue;
            ta[index] = new Thread(writerBall.RunWriter);
        }
        ta[index].IsBackground = true;
        ta[index].Start();

        // TODO create thread
```

```
public class BallMover
{
    private delegate void UpdatePictureBoxCallback(Point p);

    private PictureBox pb;
    private Semaphore wrt;
    private Semaphore rdr;
    private Mutex mut = new Mutex();
    private Random rand = new Random();
    private int speed;
    private int readcount = 0;

    public BallMover(PictureBox pb, Semaphore wrt,Semaphore rdr)
    {
        this.pb = pb;
        this.wrt = wrt;
        this.rdr = rdr;


    }
```

```
    public void RunReader()
    {

        try
        {
            speed = rand.Next(5, 10);

            while (true)
            {

                while (pb.Location.X < SynchronisationTestForm.CS_MINX)
                {
                    MoveBall();
                    Thread.Sleep(speed);
                }
                mut.WaitOne();
                readcount++;
                if (readcount == 1) wrt.WaitOne();
                mut.ReleaseMutex();
                while (pb.Location.X < SynchronisationTestForm.CS_MAXX)
                {
                    MoveBall();
                    Thread.Sleep(speed);
                }

                mut.WaitOne();
                readcount--;
                if (readcount == 0) wrt.Release();
                mut.ReleaseMutex();

                while (pb.Location.X < SynchronisationTestForm.MAXX)
                {
                    MoveBall();
                    Thread.Sleep(speed);
                }
                ResetBall();
            }
        }
```

```
public void RunWriter()
{

    try
    {
        speed = rand.Next(5, 10);

        while (true)
        {

            while (pb.Location.X < SynchronisationTestForm.CS_MINX)
            {
                MoveBall();
                Thread.Sleep(speed);
            }

            wrt.WaitOne();
            rdr.WaitOne();
            while (pb.Location.X < SynchronisationTestForm.CS_MAXX)
            {
                MoveBall();
                Thread.Sleep(speed);
            }
            wrt.Release();
            rdr.Release();

            while (pb.Location.X < SynchronisationTestForm.MAXX)
            {
                MoveBall();
                Thread.Sleep(speed);
            }
            ResetBall();
        }
    }
    catch (ThreadInterruptedException)
    {
        ResetBall();
        return;
    }
```



Ainsi fait, est le travail demandé !