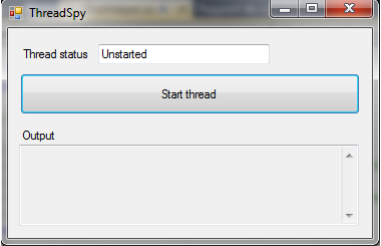
Utilisation des Semaphores pour synchronizer les Thread

On s’appuie sur l’application de la leçon 3. Utilisation des **locks**

**Solution**



**Voici le code**

ImportsSystem**.**Threading

ImportsSystem.Collections.Generic

ImportsSystem.Text

ImportsSystem.Windows.Forms

namespace ThreadSpy

class TextBoxHelper

static private TextBox textbox

public delegate sub UpdateTextCallback(char c)

// /// /// This method will add the char c into the textbox tb

/// /// The TextBox where the char will be added

/// The char to add

static public sub AddChar(TextBox tbox, char c)

textbox = tbox

textbox.Invoke(new UpdateTextCallback(AddCharSave), c)

end sub

static private sub AddCharSave(char c)

textbox.Text += c

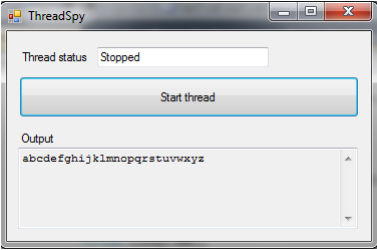
end sub

End sub

Ici c’est le TextBox **textbox** qui est partagé.

b. Changer le programme pour que plus d’une thread puisse placer un caractère dans la textbox toujours en utilisant les **locks**

**Solution**



**Code source**

public sub Run()

while(true)

For K As Integer = 0 to 1

Thread.Sleep(300)

TextBoxHelper.AddChar(tb, a)

Next

For K As Integer = 0 to 1

Thread.Sleep(300)

TextBoxHelper.AddChar(tb, b)

Next

For K As Integer = 0 to 1

Thread.Sleep(300)

TextBoxHelper.AddChar(tb, c)

Next

For K As Integer = 0 to 1

Thread.Sleep(300)

TextBoxHelper.AddChar(tb, d)

Next

For K As Integer = 0 to 1

Thread.Sleep(300)

TextBoxHelper.AddChar(tb, e)

Next

For K As Integer = 0 to 1

Thread.Sleep(300)

TextBoxHelper.AddChar(tb, f)

Next

For K As Integer = 0 to 1

Thread.Sleep(300)

TextBoxHelper.AddChar(tb, g)

Next

For K As Integer = 0 to 1

Thread.Sleep(300)

TextBoxHelper.AddChar(tb, h)

Next

For K As Integer = 0 to 1

Thread.Sleep(300)

TextBoxHelper.AddChar(tb, i)

Next

For K As Integer = 0 to 1

Thread.Sleep(300)

TextBoxHelper.AddChar(tb, l)

Next

For K As Integer = 0 to 1

Thread.Sleep(300)

TextBoxHelper.AddChar(tb, m)

Next

For K As Integer = 0 to 1

Thread.Sleep(300)

TextBoxHelper.AddChar(tb, n)

Next

For K As Integer = 0 to 1

Thread.Sleep(300)

TextBoxHelper.AddChar(tb, o)

Next

For K As Integer = 0 to 1

Thread.Sleep(300)

TextBoxHelper.AddChar(tb, p)

Next

For K As Integer = 0 to 1

Thread.Sleep(300)

TextBoxHelper.AddChar(tb, q)

Next

For K As Integer = 0 to 1

Thread.Sleep(300)

TextBoxHelper.AddChar(tb, r)

Next

For K As Integer = 0 to 1

Thread.Sleep(300)

TextBoxHelper.AddChar(tb, s)

Next

For K As Integer = 0 to 1

Thread.Sleep(300)

TextBoxHelper.AddChar(tb, t)

Next

For K As Integer = 0 to 1

Thread.Sleep(300)

TextBoxHelper.AddChar(tb, u)

Next

For K As Integer = 0 to 1

Thread.Sleep(300)

TextBoxHelper.AddChar(tb, v)

Next

For K As Integer = 0 to 1

Thread.Sleep(300)

TextBoxHelper.AddChar(tb, w)

Next

For K As Integer = 0 to 1

Thread.Sleep(300)

TextBoxHelper.AddChar(tb, x)

Next

For K As Integer = 0 to 1

Thread.Sleep(300)

TextBoxHelper.AddChar(tb, y)

Next

For K As Integer = 0 to 1

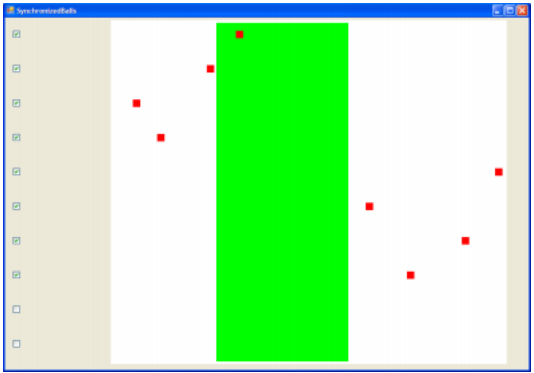
Thread.Sleep(300)

TextBoxHelper.AddChar(tb, z)

Next

End sub

Application SynchronizedBalls



On sait que la zone verte represente la section critique

Voici donc les classes :

Class SynchronisationTestForm

public partial class SynchronisationTestForm : Form

public const MINX = 0 As Int

public const MAXX = 750 As Int

public const CS\_MINX = 200 As Int

public const CS\_MAXX = 450 As Int

public PictureBox[] pictbxa = new PictureBox[10]

public Thread[] ta = new Thread[10]

public SynchronisationTestForm()

pictbxa [0] = pictureBox1

pictbxa [1] = pictureBox2

pictbxa [2] = pictureBox3

pictbxa [3] = pictureBox4

pictbxa [4] = pictureBox5

pictbxa [5] = pictureBox6

pictbxa [6] = pictureBox7

pictbxa [7] = pictureBox8

pictbxa [8] = pictureBox9

pictbxa [9] = pictureBox10

private sub checkBox\_CheckedChanged(object sender, EventArgs e)

index As Integer = (((CheckBox)sender).Location.Y - 25) / 65

PictureBox pb = pictbxa [index]

if (((CheckBox)sender).Checked)

End if

else

// The CheckBox was unchecked, so

// the corresponding thread must be interrupted and

// pb must get transparant

background color

End else

End dub

This is the form. Each time when a checkbox changes state, the method checkBox\_CheckedChanged is called.

Classe BallMover

private delegate sub UpdatePictureBoxCallback(Point p)

private PictureBox pbox

public BallMover(PictureBox pbox)

this.pb = pbox

/// /// Move ball over X axis, bouncing at the right border ///

public sub Run()

try

while (true)

while (pb.Location.X < SynchronisationTestForm.CS\_MINX)

MoveBall()

Thread.Sleep(10)

loop

while (pb.Location.X < SynchronisationTestForm.CS\_MAXX)

MoveBall()

Thread.Sleep(10)

loop

while (pb.Location.X < SynchronisationTestForm.MAXX)

MoveBall()

Thread.Sleep(10)

loop

ResetBall()

Thread.CurrentThread.Interrupt()

catch (ThreadInterruptedException)

ResetBall()

Return

End sub

/// /// This method moves the ball and returns the new location ///

private sub MoveBall()

p As Point = pb.Location

p.X++

pb.Invoke(new UpdatePictureBoxCallback(MovePictureBox), p)

/// /// This method sets the ball back to the left hand side of the white area /// public void ResetBall()

p As Point = pb.Location; p.X = SynchronisationTestForm.MINX

pb.Invoke(new UpdatePictureBoxCallback(MovePictureBox), p)

end sub

private sub MovePictureBox(Point p)

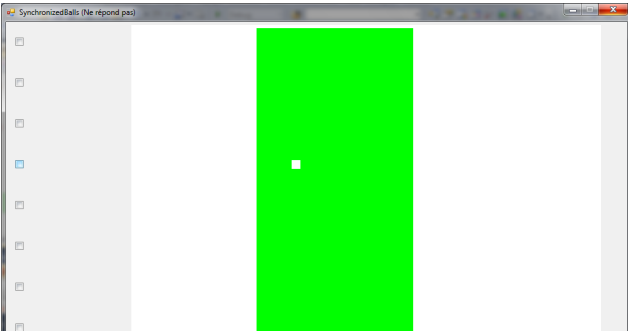
pb.Location = p

End sub

End class

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. Lorsque la balle est à l'extrême droite de la zone blanche, elle est replacée sur le côté gauche. Les balles sont toujours des Picturebox

c. Ici il est question de changer le programme donnée afin de pouvoir interrompre un thread lorsque la case est décochée, les threads créés doivent être placés dans un tableau, qui est déjà défini dans la classe SynchronisationTestForm. tous les threads s'arrêtent automatiquement lorsque la fenêtre principale est fermée.



**Code source**

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

index As Integer = (((CheckBox)sender).Location.Y - 25) / 65

// pbox is the PictureBox that belongs to this CheckBox

PictureBox pbox = 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]

// TODO create thread BallMover

ct = new BallMover(pb)

ct.Run()

End if

else

// The CheckBox was unchecked, so // the corresponding thread must be interrupted and // pb must get transparant background color

// TODO interrupt

thread Thread.CurrentThread.Interrupt()

End else

End sub

d. Give each ball a randomly chosen speed. This speed must be between roughly 100 and 200 pixels per second (so the Thread.Sleep must be between 5 and 10 msec). You can use class Random for that.

e. Identify which piece of code exactly is the Critical Section.

**Solution**

private sub MoveBall()

p As Point = pb.Location;

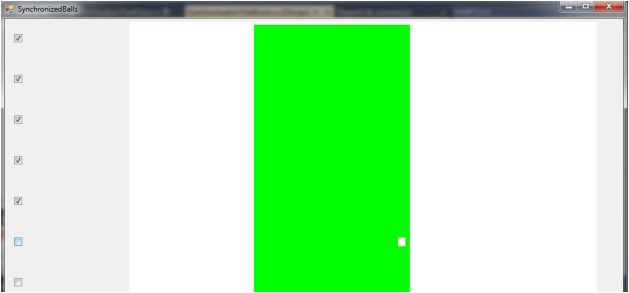
p.X++

pb.Invoke(new UpdatePictureBoxCallback(MovePictureBox), p)

End sub

f. ll est queston de modifiez le programme de telle sorte qu'au plus un thread se trouve à tout moment dans la section critique et d’utilisez un sémaphore partagé par tous les threads. Tous doit bien fonctionner lorsqu'un thread est interrompu dans sa section critique

Solution



**Code source**

public sub Run()

try

while (true)

while (pb.Location.X < SynchronisationTestForm.CS\_MINX)

MoveBall()

Thread.Sleep(10)

Next

while (pb.Location.X < SynchronisationTestForm.CS\_MAXX)

MoveBall()

Thread.Sleep(10)

while (pb.Location.X < SynchronisationTestForm.MAXX)

MoveBall()

Thread.Sleep(10)

Next

ResetBall()

Thread.CurrentThread.Interrupt()

End sub

catch (ThreadInterruptedException)

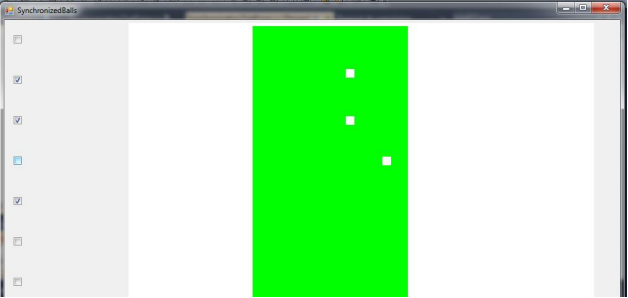
ResetBall()

return

End sub

g. Changer le programme pour qu’au maximum seulement 3 thead s’exécutent dans la section critique.

**Solution**



Code source

ImportsSystem

ImportsSystem.Collections.Generic

ImportsSystem.ComponentModel

ImportsSystem.Data

ImportsSystem.Drawing

ImportsSystem.Text

ImportsSystem.Windows.Forms

ImportsSystem.Threading

public partial class SynchronisationTestForm

public const MINX As Int= 3

public const MAXX As Int = 750

public const CS\_MINX As Int = 100

public const CS\_MAXX As Int = 200

private PictureBox[] pitba = new PictureBox[10]

private Thread[] ta = new Thread[10]

Random rd

public SynchronisationTestForm()

pitba[0] = pictureBox1

pitba [1] = pictureBox2

pitba [2] = pictureBox3

pitba [3] = pictureBox4

pitba [4] = pictureBox5

pitba [5] = pictureBox6

pitba [6] = pictureBox7

pitba [7] = pictureBox8

pitba [8] = pictureBox9

pitba [9] = pictureBox10

private sub 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

index As Integer = (((CheckBox)sender).Location.Y - 25) / 65

// pbox is the PictureBox that belongs to this CheckBox

PictureBox pbox = pitba[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]

// TODO create thread

BallMover ct = new BallMover(pbox)

ct.Run()

End if

else

// The CheckBox was unchecked, so

// the corresponding thread must be interrupted and

// pbox must get transparant

background color

// TODO interrupt

thread Thread.CurrentThread.Interrupt()

End else

End sub

End class

Save this version.

Nous allons maintenant implémenter le problème lectures/écritures. Pour cela, nous avons besoin de deux sortes de fils, qui seront représentés par deux couleurs de billes différentes :

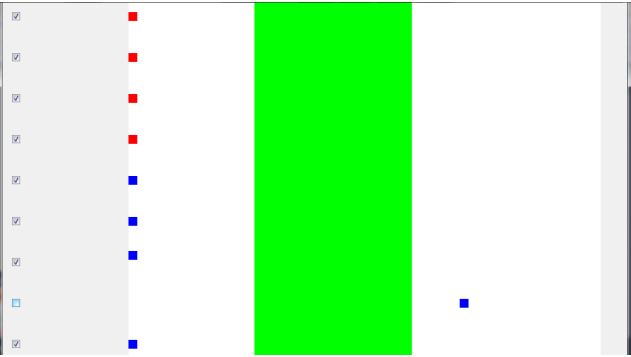
• • rouge (lecture) et

• • bleu (écriture).

• les 5 premières boules sont rouges et les 5 dernières boules sont bleues.

• Utilisation de 2 méthodes Run dans la classe BallMover au lieu de 1. Nommés RunReader et RunWriter. les boules rouges exécutent RunReader et les boules bleues exécutent RunWriter.

**Solution**



**Code source**

public void RunReader()

pb.BackColor = Color.Red

try

Random rd = new Random(200)

while (true)

while (pb.Location.X < SynchronisationTestForm.CS\_MINX)

MoveBall()

Thread.Sleep(5)

Thread.EndCriticalRegion()

while (pb.Location.X < SynchronisationTestForm.CS\_MAXX)

MoveBall()

Thread.Sleep(5)

Thread.EndCriticalRegion()

while (pb.Location.X < SynchronisationTestForm.MAXX)

MoveBall()

Thread.Sleep(5)

Thread.EndCriticalRegion()

Thread.CurrentThread.Interrupt()

ResetBall()

catch (ThreadInterruptedException)

ResetBall()

return

End sub

public void RunWriter()

pb.BackColor = Color.Blue

try

Random rd = new Random(200)

while (true)

while (pb.Location.X < SynchronisationTestForm.CS\_MINX)

MoveBall()

Thread.Sleep(5)

Thread.EndCriticalRegion()

while (pb.Location.X < SynchronisationTestForm.CS\_MAXX)

MoveBall()

Thread.Sleep(5)

Thread.EndCriticalRegion()

while (pb.Location.X < SynchronisationTestForm.MAXX)

MoveBall()

Thread.Sleep(5)

Thread.EndCriticalRegion()

Thread.CurrentThread.Interrupt()

ResetBall()

catch (ThreadInterruptedException)

ResetBall()

return

End sub

• j. Take the solution for the readers/writers problem that was given in the sheets (using semaphores wrt and mutex, and integer readcount) and make it work in this application. The semaphores can be handled in the same way as in question f. To simplify things, you can assume that a thread will never be stopped inside the green area, so you don’t have to handle this situation.

Solution

