UNIVERSITE PROTESTANTE AU CONGO

FACULTE DE SCIENCES INFORMATIQUES

Département de systèmes informatiques

BP : 4745 KINSHASA/LINGWALA

RAPPORT DU TP 6

PROJET DE PREMIERE ANNEE DE LICENCE

Présenté par :

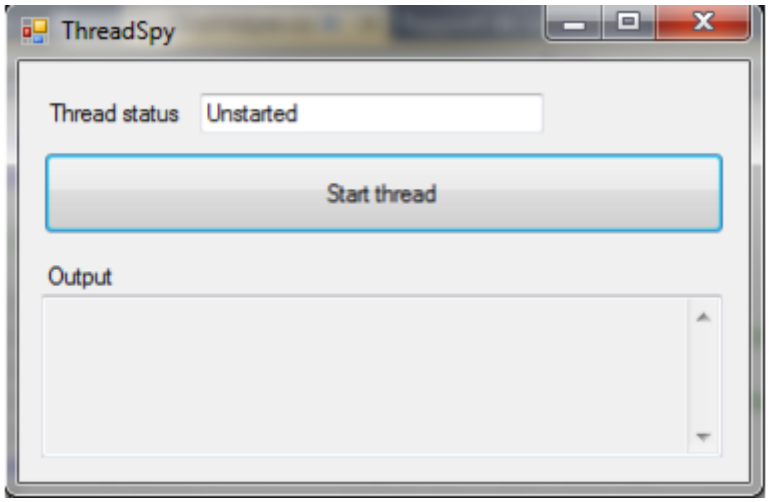
Lionel PENENGE TEBANDIME

Superviseur: Prof.Dr MUKALA Patrick

ANNEE ACADEMIQUE: 2021-2022

In this lesson, we will use the VB locks and semaphores to synchronize threads. First, we continue with the application of lesson 3. In this application, there can shared variables that are used by multiple threads (this depends on the way in which you programmed the assignment). As we learned this week, this is inherently unsafe, so we have to fix this. a. Identify the shared variables and make sure that they are used in a threadsafe way (this means that always at most one thread at a time is allowed to use these variables). Use locks to do this.

Solution



code

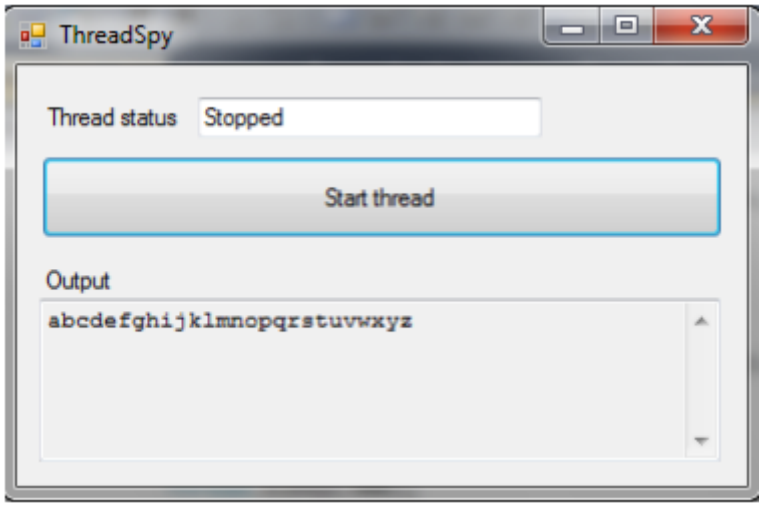
```
Imports System.Threading
Imports System.Collections.Generic

Imports System.Text

Imports System.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 tb, char c)

        textbox = tb
```

```
        textbox.Invoke(new UpdateTextCallback(AddCharSave), c)
end sub
static private sub AddCharSave(char c)
        textbox.Text += c
 end sub
End sub
```

Dans cette application c'est le TextBox textbox qui est partagé.

b. Also, change the program such that at most one thread at a time can put characters in the textbox. Use locks for this also.

Solution



code cource

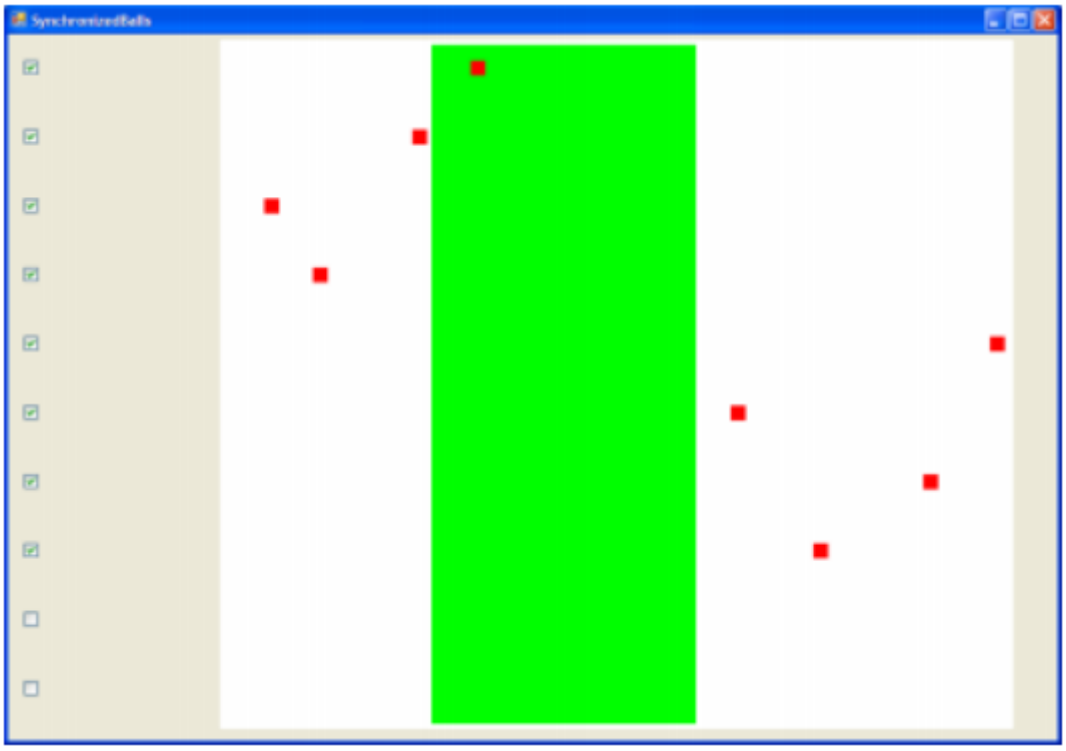
```
public sub Run()
while(true)
        For I As Integer = 0 to 1
                Thread.Sleep(300)
                TextBoxHelper.AddChar(tb, a)
    Next
        For I As Integer = 0 to 1
                Thread.Sleep(300)
                TextBoxHelper.AddChar(tb, b)
    Next

        For I As Integer = 0 to 1
                Thread.Sleep(300)
```

```
            TextBoxHelper.AddChar(tb, c)
Next
    For I As Integer = 0 to 1
            Thread.Sleep(300)
            TextBoxHelper.AddChar(tb, d)
Next

    For I As Integer = 0 to 1
            Thread.Sleep(300)
            TextBoxHelper.AddChar(tb, e)
Next

    For I As Integer = 0 to 1
            Thread.Sleep(300)
            TextBoxHelper.AddChar(tb, f)
Next

    For I As Integer = 0 to 1
            Thread.Sleep(300)
            TextBoxHelper.AddChar(tb, g)
Next

    For I As Integer = 0 to 1
            Thread.Sleep(300)
            TextBoxHelper.AddChar(tb, h)
Next

    For I As Integer = 0 to 1
            Thread.Sleep(300)
            TextBoxHelper.AddChar(tb, i)
Next
    For I As Integer = 0 to 1
            Thread.Sleep(300)
            TextBoxHelper.AddChar(tb, l)
Next

    For I As Integer = 0 to 1
            Thread.Sleep(300)
            TextBoxHelper.AddChar(tb, m)
Next

    For I As Integer = 0 to 1
            Thread.Sleep(300)
```

```
            TextBoxHelper.AddChar(tb, n)
Next
    For I As Integer = 0 to 1
            Thread.Sleep(300)
            TextBoxHelper.AddChar(tb, o)
Next
    For I As Integer = 0 to 1
            Thread.Sleep(300)
            TextBoxHelper.AddChar(tb, p)
Next
    For I As Integer = 0 to 1
            Thread.Sleep(300)
            TextBoxHelper.AddChar(tb, q)
Next
    For I As Integer = 0 to 1
            Thread.Sleep(300)
            TextBoxHelper.AddChar(tb, r)
Next
    For I As Integer = 0 to 1
            Thread.Sleep(300)
            TextBoxHelper.AddChar(tb, s)
Next
    For I As Integer = 0 to 1
            Thread.Sleep(300)
            TextBoxHelper.AddChar(tb, t)
Next
    For I As Integer = 0 to 1
            Thread.Sleep(300)
            TextBoxHelper.AddChar(tb, u)
Next
    For I As Integer = 0 to 1
            Thread.Sleep(300)
            TextBoxHelper.AddChar(tb, v)
Next
    For I As Integer = 0 to 1
            Thread.Sleep(300)
            TextBoxHelper.AddChar(tb, w)
Next
    For I As Integer = 0 to 1
            Thread.Sleep(300)
            TextBoxHelper.AddChar(tb, x)
Next
    For I As Integer = 0 to 1
```

```
            Thread.Sleep(300)
            TextBoxHelper.AddChar(tb, y)
    Next
        For I As Integer = 0 to 1
            Thread.Sleep(300)
            TextBoxHelper.AddChar(tb, z)
    Next

     End sub
```

Next, we use a new application SynchronizedBalls in which the concurrency problems can be shown in a more graphical way. This application has the following screen:



On the left we see 10 checkboxes. Whenever a checkbox is checked, a new thread must be started, that makes a ball move from left to right in the white area. When a checkbox is unchecked the thread must be interrupted. The green area represents the Critical Section, which plays a special role later on. The application has the following classes:

Classe SynchronisationTestForm

public partial class SynchronisationTestForm : Form

public const MINX = 0 As Interger

public const MAXX = 750 As Interger

```
public const CS_MINX = 200 As Interger
public const CS_MAXX = 450 As Interger
public PictureBox[] pba = new PictureBox[10]
public Thread[] ta = new Thread[10]
public SynchronisationTestForm()
pba[0] = pictureBox1
pba[1] = pictureBox2
pba[2] = pictureBox3
pba[3] = pictureBox4
pba[4] = pictureBox5
pba[5] = pictureBox6
pba[6] = pictureBox7
pba[7] = pictureBox8
pba[8] = pictureBox9
pba[9] = pictureBox10
private sub checkBox_CheckedChanged(object sender, EventArgs e)
index As Integer = (((CheckBox)sender).Location.Y - 25) / 65
PictureBox pb = pba[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 pb

public BallMover(PictureBox pb)

this.pb = pb

/// /// 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)

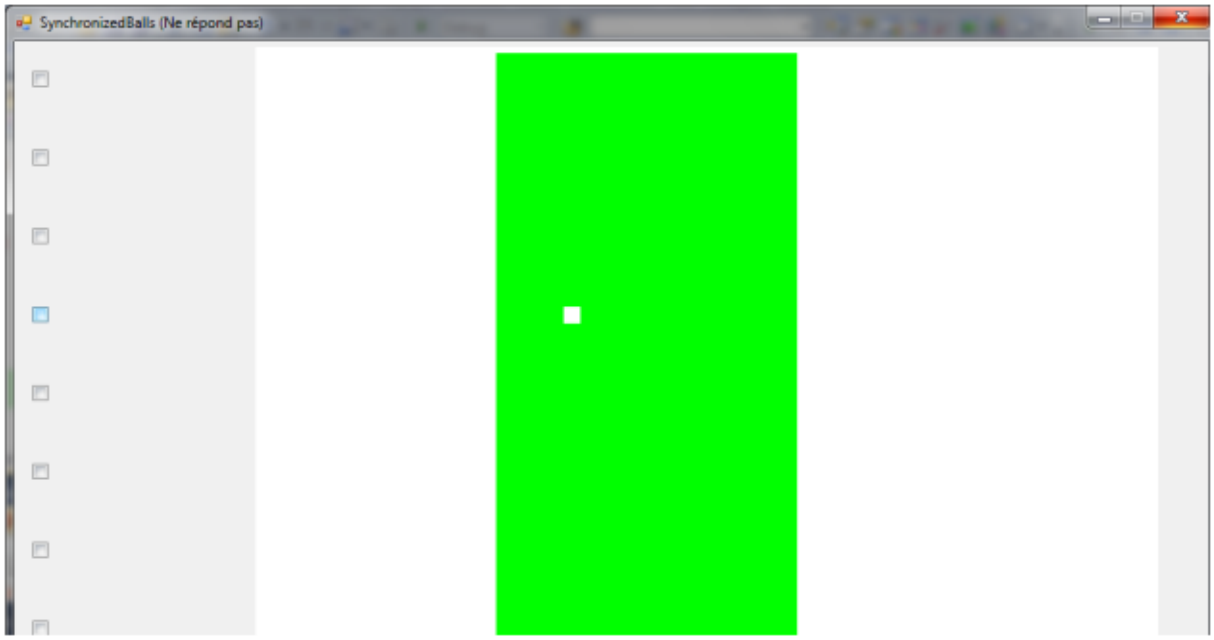
pb.Location = p

End sub

End class

This class contains the method Run, which is to be executed by every thread. In this method, a ball is moved from left to right on the screen. When the ball is at the far right hand side of the white area, it is placed back to the left hand side. The balls are sll picture boxes.

c. Change the given program such, that the above described behavior is implemented. In order to be able to interrupt a thread when the checkbox is being unchecked, the threads that are created must be put in the array ta, which is already defined in class SynchronisationTestForm. Make sure that all threads stop automatically when the main window is closed.



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
// 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]
  // 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()

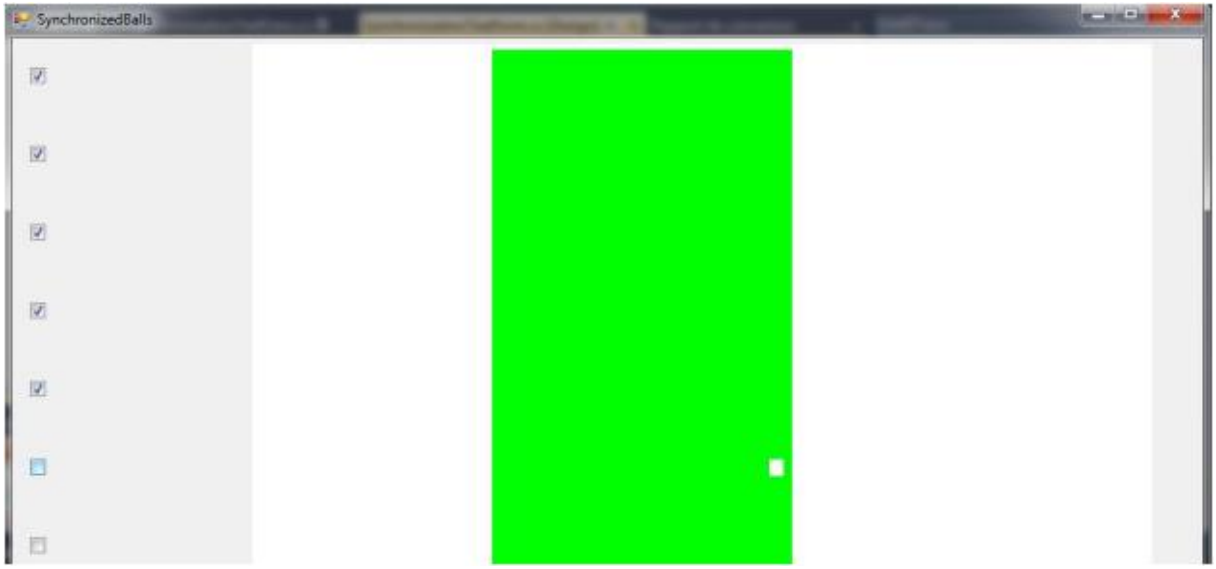
p As Point = pb.Location;

p.X++

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

End sub

f. Change the program such that at most one thread is in the Critical Section at all times. Use a semaphore for this. Since this semaphore will have to be shared by all threads, create it in the class SynchronisationTestForm, and give the constructor of BallMover an extra argument of type Semaphore (and the class BallMover an extra attribute of type Semaphore) to make it known to the threads also. Make sure it also works well when a thread is interrupted within its Critical Section. Solution

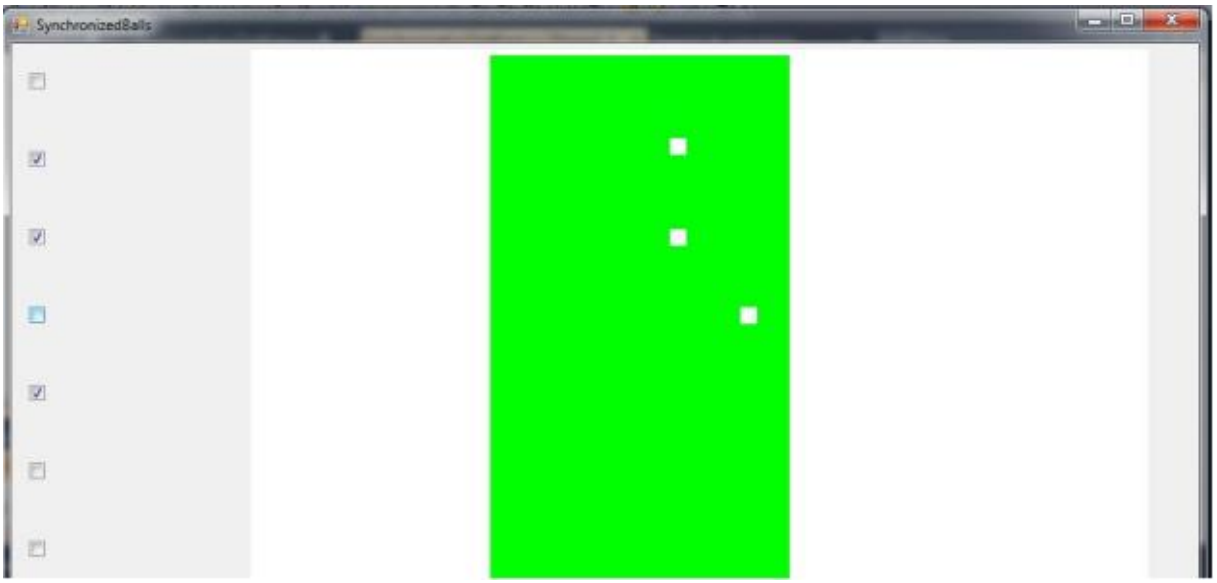


Code source

```
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. Change the program such that at most three threads are in the Critical Section at all times.

Solution



Code source

```
Imports System
Imports System.Collections.Generic
Imports System.ComponentModel
Imports System.Data
Imports System.Drawing
Imports System.Text
Imports System.Windows.Forms
Imports System.Threading
public partial class SynchronisationTestForm
public const MINX As Integer= 3
public const MAXX As Integer = 750
public const CS_MINX As Integer = 100
public const CS_MAXX As Integer = 200
private PictureBox[] pba = new PictureBox[10]
private Thread[] ta = new Thread[10]
Random rd
public SynchronisationTestForm()
pba[0] = pictureBox1
```

```
pba[1] = pictureBox2
pba[2] = pictureBox3
pba[3] = pictureBox4
pba[4] = pictureBox5
pba[5] = pictureBox6
pba[6] = pictureBox7
pba[7] = pictureBox8
pba[8] = pictureBox9
pba[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
// 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]
// 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

End class

Save this version.

Now we will implement the readers/writers problem. For that, we need two kinds of threads, which will be represented by two different ball-colors:

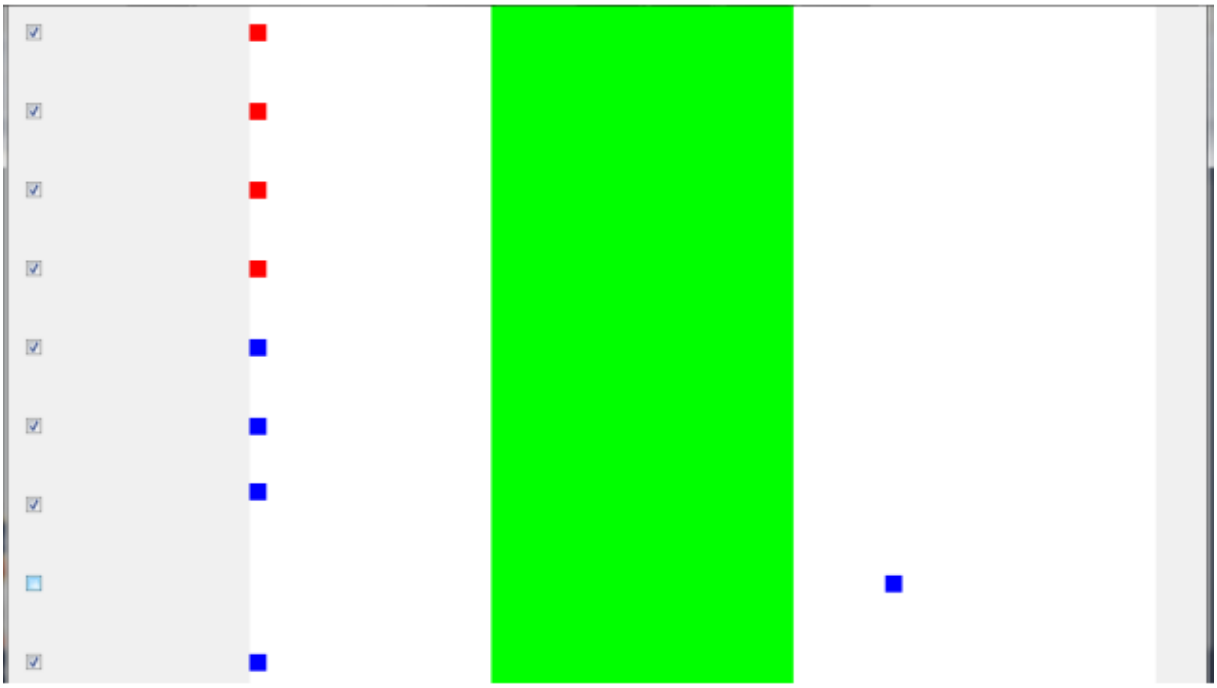
• • red (readers) and

• • blue (writers).

• h. Change the program such that the first 5 balls are red, and the last 5 balls are blue.

• i. Also, use 2 Run-methods inside class BallMover instead of 1. Name them RunReader and RunWriter. Let the red balls execute RunReader and the blue balls execute RunWriter.

Solution



Code source

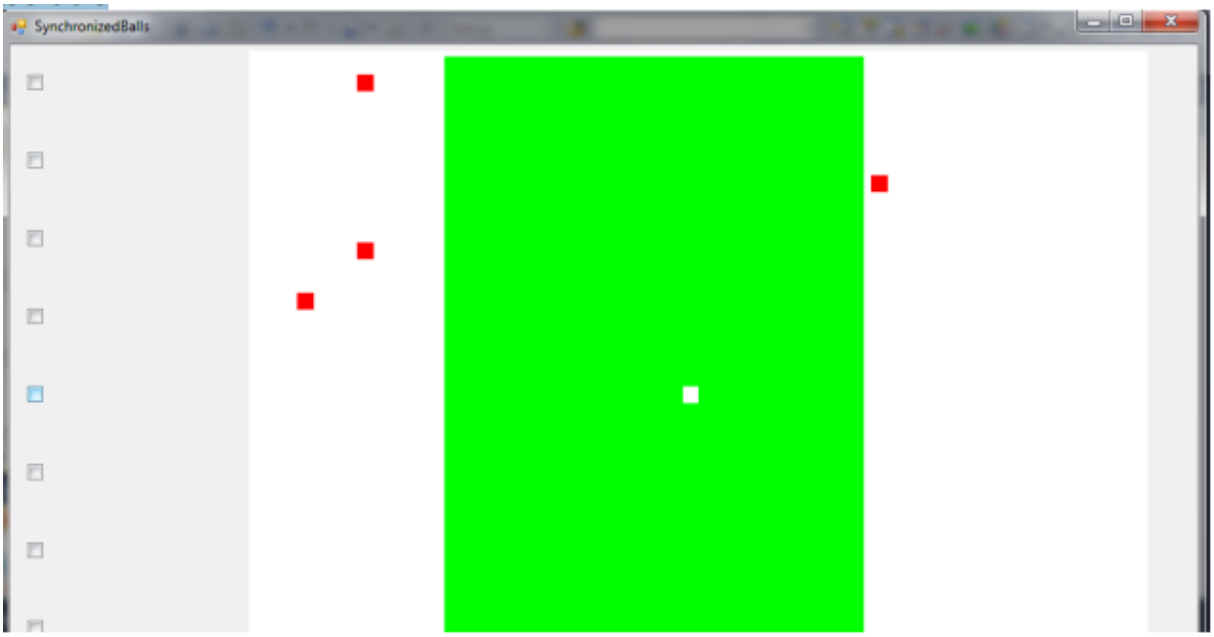
public void RunReader()

```
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


SynchronizedBalls