Hands On Lab: Pruebas. Calidad y Rendimiento

Contenido

[Introducción 3](#_Toc412131790)

[Calidad 3](#_Toc412131791)

[Rendimiento 3](#_Toc412131792)

[El proyecto: Buscaminas WPF 4](#_Toc412131793)

[Paso 1: Creación de la solución 4](#_Toc412131794)

[Paso 2: Creación del componente TileControl 6](#_Toc412131795)

[Paso 2: Probar el control 12](#_Toc412131796)

[Paso 3: Finalizar la aplicación 14](#_Toc412131797)

[Paso 4: Probar la aplicación 17](#_Toc412131798)

[Paso 5: Fake Objects: Stubs 18](#_Toc412131799)

[Paso 6: Fake Objects: Shims 19](#_Toc412131800)

[Paso 7: Pruebas de Interfaz 19](#_Toc412131801)

[Paso 8: Cobertura de pruebas 23](#_Toc412131802)

[Optimizando el código: Mejorando el rendimiento 23](#_Toc412131803)

[Paso 1: Usando el Profiler para localizar los cuellos de botella 23](#_Toc412131804)

[Paso 2: Mejorando el código 25](#_Toc412131805)

# Introducción

En este ejercicio se pretende introducir al uso tanto de las pruebas de desarrollo como a la mejora de rendimiento mediante el uso de las herramientas que Visual Studio 2013 pone a nuestro alcance.

Antes de comenzar pretendemos hacer una muy breve introducción tanto al concepto de calidad como al de rendimiento en el desarrollo, con el fin de comprender qué beneficios podemos obtener con estas herramientas.

## Calidad

Hablar de calidad en términos absolutos es muy complicado, porque si bien todos entendemos el concepto de manera muy similar no todos coincidimos en qué es tener o no calidad.

Es sin embargo una misión más sencilla entender este concepto en términos relativos. De manera en que podamos responder satisfactoriamente a alguna de estas preguntas, ¿qué puedo hacer para que una aplicación tenga más calidad? O ¿cómo puedo desarrollar con más calidad?

Entendiendo así el concepto desarrollar utilizando pruebas debería incrementar la calidad de una aplicación, el cómo o el por qué será algo que dependerá de que aporte cada una de las técnicas y estrategias de pruebas.

## Rendimiento

Como en el caso de la calidad el rendimiento no es fácil de definir en términos absolutos, aunque aquí y pese a que cada usuario tiene percepciones diferentes, el acuerdo en cuanto a lo que es tener una aplicación con un rendimiento aceptable o no, parece más sencillo de alcanzar.

En cualquier caso en la actualidad no existe un mecanismo para determinar cuánto ha de tardar o no una aplicación en hacer esta o aquella tarea, aunque sin duda habitualmente si podemos señalar que parte de una aplicación se ejecuta con más lentitud.

Visual Studio pretende ayudarnos en esta tarea de medida y localización de cuellos de botella, señalando donde un trabajo de refactorización puede ser más útil a la hora de mejorar el rendimiento de una aplicación.

# El proyecto: Buscaminas WPF

Para la realización de este “Hands on lab” vamos a crear un proyecto de prueba que pretende emular al tradicional Buscaminas que acompañó a Windows durante muchas versiones.

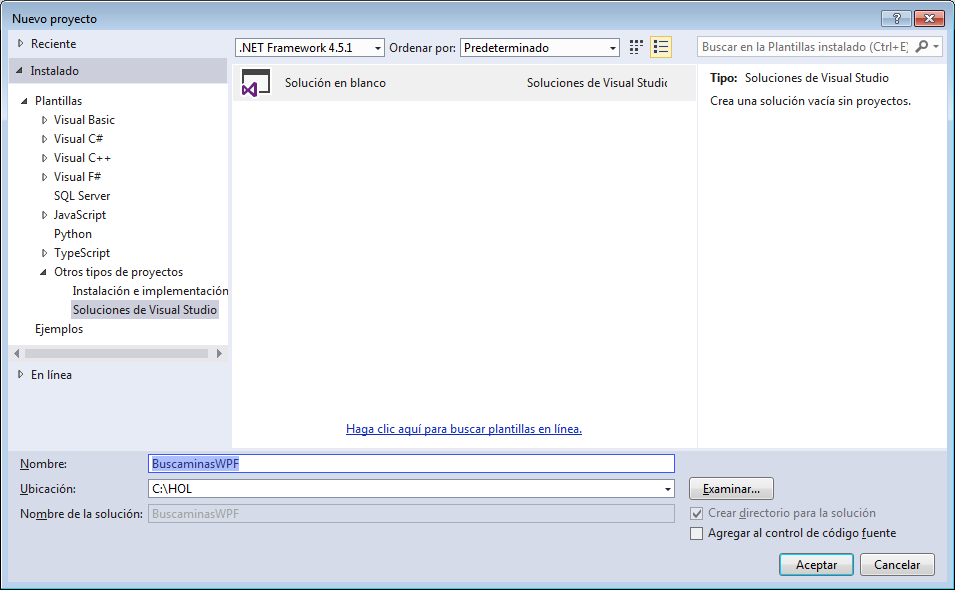
Para aquellos que desconocen la dinámica del juego, en él se nos presenta un suelo enlosado en el que tras cada loseta podemos encontrar o bien una mina o bien un número. Si tenemos la mala suerte de encontrar una mina habremos perdido la partida; si por el contrario lo que encontramos un número este indicará de las hasta ocho losetas circundantes, cuántas esconden una mina. Con estas pistas el objetivo del juego es destapar todas las losetas posibles sin que ninguna sea una bomba.

La tecnología a utilizar va a ser Windows Presentation Foundation y la estructura de la solución va a involucrar dos proyectos, uno en el que desarrollemos un componente para la representación de las diferentes losetas que pueden esconder o no las minas y otro que será la aplicación principal del juego.

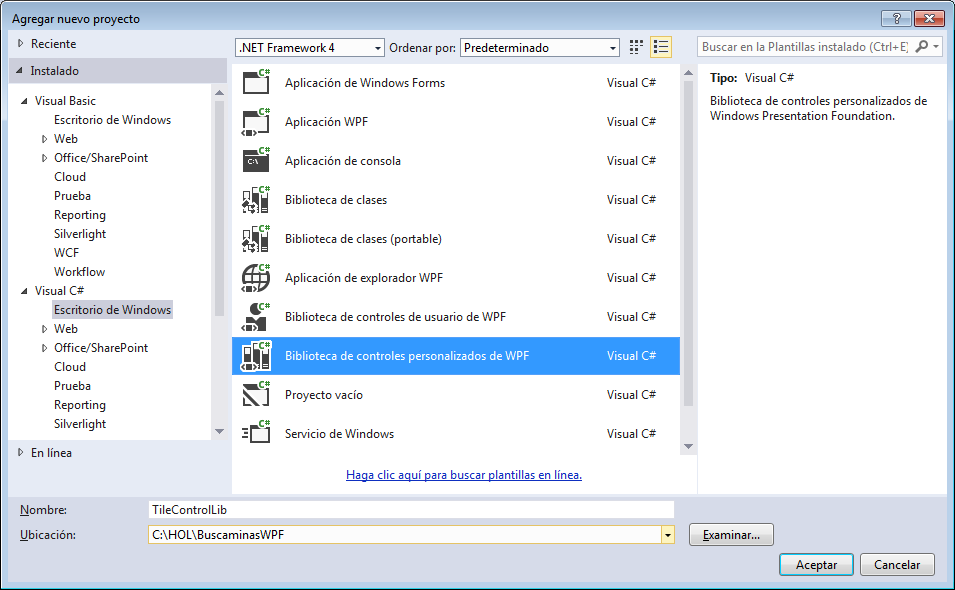
Para la confección de este “Hand on lab” hemos utilizado la versión Premium de Visual Studio 2013 en español con el Update 4 instalado y con la opción de configuración de entorno de inicio establecida a general de desarrollo, sobre un equipo son Windows 7 Professional 32Bit. Cualquier inexactitud en proceso ha de ser achacada a dicha configuración excepto en aquellos casos de error u omisión inadvertida por lo que se pide encarecidamente disculpas de ante mano.

## Paso 1: Creación de la solución

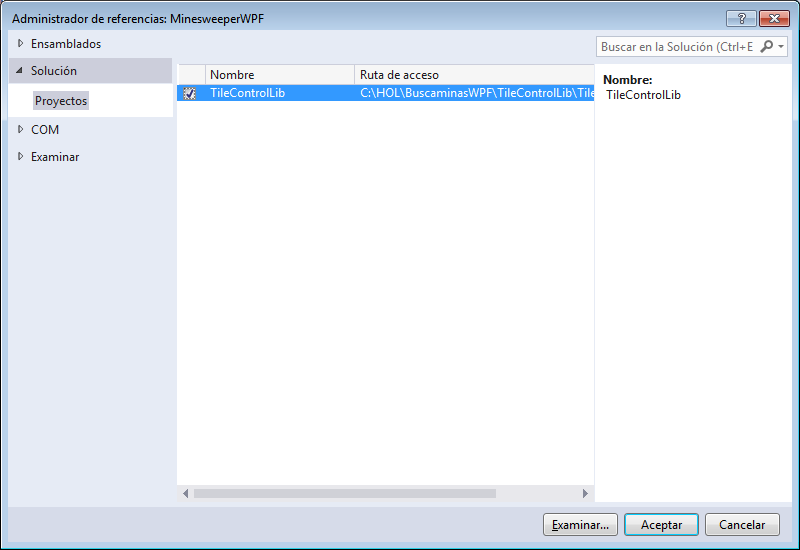
1. Abrir Visual Studio 2013
2. En el menú de *Archivo -> Nuevo* pulsar sobre la opción de *Proyecto…*
3. En la ventana emergente seleccionar dentro del conjunto de plantillas *Instalado* la rama *Otros tipos de proyectos -> Soluciones de Visual Studio* la plantilla *Solución en Blanco* con el nombre “*BuscaminasWPF*” y situándola en la Ubicación *C:\HOL.*



1. Pulsar el botón aceptar y crear la solución.
2. Una vez creada la solución en el explorador de soluciones hacer clic derecho sobre ella y seleccionar el menú *Agregar* y la opción *Nuevo Proyecto.*
3. En la ventana emergente seleccionar dentro del conjunto de plantillas *Instalado* la rama *Visual C#->Escritorio de Windows* y la plantilla *Biblioteca de controles personalizados de WPF*, con el nombre *TileControlLib*



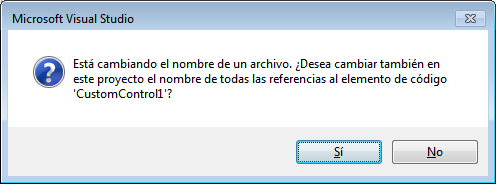
1. Pulsar el botón *Aceptar* y crear el proyecto.
2. En el explorador de soluciones hacer clic derecho sobre la solución y seleccionar el menú *Agregar* y la opción *Nuevo Proyecto.*
3. En la ventana emergente seleccionar dentro del conjunto de plantillas *Instalado* la rama bien *Visual C#->Escritorio de Windows* y la plantilla *Aplicación WPF*, con el nombre *MinesweeperWPF*
4. Pulsar el botón *Aceptar* y crear el proyecto.
5. En el explorador de soluciones hacer clic derecho sobre el nodo *MinesweeperWPF*  y seleccionar la opción *Establecer como proyecto de Inicio*.
6. En el explorador de soluciones hacer clic derecho sobre el nodo *MinesweeperWPF*  y seleccionar el menú *Agregar y* la opción *Referencia..*.
7. En la ventana emergente seleccionar el conjunto de *Solución->Proyectos* y marcar la referencia con el nombre *TileControlLib*



1. Pulsar el botón *Aceptar*

## Paso 2: Creación del componente TileControl

1. En el explorador de soluciones localizar el nodo TileControlLib y hacer clic derecho sobre el archivo CustomControl1.cs y seleccionar la opción *Cambiar nombre*
2. Cambiar el nombre del archivo a TileControl.cs
3. En el diálogo emergente en el que se nos solicita aprobación para cambiar las referencias en todo el código seleccionar el botón Sí.



1. En el explorador de soluciones localizar el nodo TileControlLib y hacer clic derecho sobre él y seleccionar el menú *Agregar* y la opción *Clase…*
2. En la ventana emergente escribir *TileControlState* y pulsar *Agregar.*
3. Sustituir el código de la unidad recién creada por el que convenga según el lenguaje seleccionado:

C#

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace TileControlLib

{

public enum TileControlState

{

Covered,

Text,

Bomb

}

}

1. En el explorador de soluciones localizar el nodo TileControlLib y hacer doble clic sobre el archivo TileControl.cs e insertar debajo del constructor el siguiente código:

C#

public string Text

{

get { return (string)GetValue(TextProperty); }

set { SetValue(TextProperty, value); }

}

public static readonly DependencyProperty TextProperty =

DependencyProperty.Register("Text", typeof(string), typeof(TileControl), new PropertyMetadata(String.Empty));

public TileControlState State

{

get { return (TileControlState)GetValue(StateProperty); }

set

{

TileControlState oldValue = (TileControlState)GetValue(StateProperty);

SetValue(StateProperty, value);

RaiseDiscoverEventEvent(oldValue, value);

}

}

public static readonly DependencyProperty StateProperty =

DependencyProperty.Register("State", typeof(TileControlState), typeof(TileControl), new PropertyMetadata(TileControlState.Covered));

public bool IsBomb

{

get { return (bool)GetValue(IsBombProperty); }

set { SetValue(IsBombProperty, value); }

}

public static readonly DependencyProperty IsBombProperty =

DependencyProperty.Register("IsBomb", typeof(bool), typeof(TileControl), new PropertyMetadata(false));

Hemos insertado tres propiedades: Text, para mostrar el número cuando convenga; State, que indica si el control ha de mostrar el dibujo de la loseta, el de la bomba o el número; y IsBomb, que indica si esta loseta es o no una bomba. Estas propiedades son del tipo DependencyProperty de WPF, que nos permiten que sean usadas en los mecanismos integrados de estilo, enlace a datos, etc… de esta tecnología.

1. Debajo del código anterior insertar el siguiente:

C#

public static readonly RoutedEvent DiscoverEvent = EventManager.RegisterRoutedEvent(

"Discovered", RoutingStrategy.Bubble, typeof(RoutedPropertyChangedEventHandler<TileControlState>), typeof(TileControl));

public event RoutedPropertyChangedEventHandler<TileControlState> Discover

{

add { AddHandler(DiscoverEvent, value); }

remove { RemoveHandler(DiscoverEvent, value); }

}

// This method raises the Tap event

void RaiseDiscoverEventEvent(TileControlState oldValue, TileControlState newValue)

{

RoutedPropertyChangedEventArgs<TileControlState> newEventArgs = new RoutedPropertyChangedEventArgs<TileControlState>(oldValue, newValue, TileControl.DiscoverEvent);

RaiseEvent(newEventArgs);

}

Hemos insertado un evento en el control para que este pueda comunicar cuando cambia su propiedad State. De esa manera el elemento puede comunicar cuando esta propiedad cambia. Hemos utilizado un RoutedEvent porque al igual que las DependencyProperty son los eventos propios de WPF. En este caso un RoutedPropertyChangedEvent para poder comunicar el estado antiguo (oldValue) y el nuevo (newValue). Además hemos incluido un método privado RaiseDiscoverEventEvent para simplificar la elevación del método.

1. Por último insertar debajo del código anterior el siguiente:

C#

protected override void OnPreviewMouseLeftButtonUp(MouseButtonEventArgs e)

{

base.OnPreviewMouseLeftButtonUp(e);

if (State == TileControlState.Covered)

if (IsBomb)

State = TileControlState.Bomb;

else

State = TileControlState.Text;

}

Hemos sobrescrito el método OnPreviewMouseLeftButtonUp a fin de cambiar el estado del control cuando se haga clic sobre él. De esa manera el mecanismo básico del juego queda encapsulado en el control.

1. En el explorador de soluciones localizar el nodo TileControlLib y hacer clic derecho sobre él y seleccionar el menú *Agregar* y la opción *Clase…*
2. En la ventana emergente escribir *TileControlStateToVisibilityConverter*  y pulsar *Agregar.*
3. Sustituir el código de la unidad recién creada por el que convenga según el lenguaje seleccionado:

C#

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Windows;

using System.Windows.Data;

namespace TileControlLib

{

public class TileControlStateToVisibilityConverter: IValueConverter

{

public object Convert(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture)

{

Visibility result = Visibility.Hidden;

switch(System.Convert.ToInt32(parameter))

{

case 0:

result = ((TileControlState)value) == TileControlState.Covered ? Visibility.Visible:Visibility.Hidden;

break;

case 1:

result = ((TileControlState)value) == TileControlState.Text ? Visibility.Visible:Visibility.Hidden;

break;

case 2:

result = ((TileControlState)value) == TileControlState.Bomb ? Visibility.Visible:Visibility.Hidden;

break;

}

return result;

}

public object ConvertBack(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture)

{

TileControlState result = TileControlState.Covered;

switch (System.Convert.ToInt32(parameter))

{

case 0:

result = ((Visibility)value) == Visibility.Visible ? TileControlState.Covered : TileControlState.Text;

break;

case 1:

result = ((Visibility)value) == Visibility.Visible ? TileControlState.Text : TileControlState.Covered;

break;

case 2:

result = ((Visibility)value) == Visibility.Visible ? TileControlState.Bomb : TileControlState.Covered;

break;

}

return result;

}

}

}

Hemos creado este conversor para establecer una relación entre lo que el control ha de mostrar en función del valor de su propiedad State.

1. En el explorador de soluciones localizar el nodo TileControlLib -> Themes y dentro hacer doble clic sobre el archivo Generic.xaml para abrirlo.
2. Sustituir el contenido por el siguiente:

XAML

<ResourceDictionary

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:local="clr-namespace:TileControlLib">

<local:TileControlStateToVisibilityConverter x:Key="TileControlStateToVisibilityConverter" />

<Style TargetType="{x:Type local:TileControl}">

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="{x:Type local:TileControl}">

<Border Background="{TemplateBinding Background}"

BorderBrush="{TemplateBinding BorderBrush}"

BorderThickness="{TemplateBinding BorderThickness}">

<Grid Width="{TemplateBinding Width}" Height="{TemplateBinding Height}">

<Viewbox xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" Stretch="Fill" Visibility="{TemplateBinding State,Converter={StaticResource TileControlStateToVisibilityConverter},ConverterParameter=0}">

<Canvas Name="svg3336" Width="512" Height="512">

<Canvas.RenderTransform>

<TranslateTransform X="0" Y="0"/>

</Canvas.RenderTransform>

<Canvas.Resources/>

<!--Unknown tag: metadata-->

<!--Unknown tag: sodipodi:namedview-->

<Canvas Name="g3338"/>

<Rectangle xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Canvas.Left="84.229202" Canvas.Top="140.06654" Width="333.13123" Height="91.800369" Name="rect5602" StrokeLineJoin="Round"/>

<Canvas Name="g5728">

<Canvas.RenderTransform>

<MatrixTransform Matrix="3.5127618 0 0 3.5337977 -585.40133 -620.95596"/>

</Canvas.RenderTransform>

<Path xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Name="path5634" Fill="#FF666666" StrokeThickness="1" Stroke="#FF000000" StrokeLineJoin="Miter" StrokeStartLineCap="Flat" StrokeEndLineCap="Flat">

<Path.Data>

<PathGeometry Figures="m 169.40481 317.98891 c 14.19593 -13.24954 14.19593 -13.24954 14.19593 -13.24954 l 113.56747 0 12.30314 13.24954 z" FillRule="evenodd"/>

</Path.Data>

</Path>

<Path xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Name="path5636" Fill="#FFCCCCCC" StrokeThickness="1" Stroke="#FF000000" StrokeLineJoin="Miter" StrokeStartLineCap="Flat" StrokeEndLineCap="Flat">

<Path.Data>

<PathGeometry Figures="m 297.16821 304.73937 0 -112.62107 12.30314 -13.24954 0 139.12015 z" FillRule="evenodd"/>

</Path.Data>

</Path>

<Path xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Name="path5638" Fill="#FF666666" StrokeThickness="1" Stroke="#FF000000" StrokeLineJoin="Miter" StrokeStartLineCap="Flat" StrokeEndLineCap="Flat">

<Path.Data>

<PathGeometry Figures="m 183.60074 192.1183 -14.19593 -13.24954 140.06654 0 -12.30314 13.24954 z" FillRule="evenodd"/>

</Path.Data>

</Path>

<Path xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Name="path5640" Fill="#FF000000" StrokeThickness="1" Stroke="#FF000000" StrokeLineJoin="Miter" StrokeStartLineCap="Flat" StrokeEndLineCap="Flat">

<Path.Data>

<PathGeometry Figures="m 183.60074 304.73937 0 -112.62107 -14.19593 -13.24954 0 139.12015 z" FillRule="evenodd"/>

</Path.Data>

</Path>

<Path xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Name="path5658" StrokeThickness="1.00906658" Stroke="#FF000000" StrokeLineJoin="Miter" StrokeStartLineCap="Flat" StrokeEndLineCap="Flat">

<Path.Data>

<PathGeometry Figures="m 167.15473 176.79138 144.1873 0 0 143.3102 -144.1873 0 z" FillRule="evenodd"/>

</Path.Data>

</Path>

<Path xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Name="path5664" Fill="#FFCCCCCC" StrokeThickness="1" Stroke="#FF000000" StrokeLineJoin="Miter" StrokeStartLineCap="Flat" StrokeEndLineCap="Flat">

<Path.Data>

<PathGeometry Figures="m 185.49353 194.24769 109.78189 0 0 108.83549 -109.78189 0 z" FillRule="evenodd"/>

</Path.Data>

</Path>

</Canvas>

</Canvas>

</Viewbox>

<Viewbox Stretch="Fill" Visibility="{TemplateBinding State,Converter={StaticResource TileControlStateToVisibilityConverter},ConverterParameter=1}">

<TextBlock TextWrapping="Wrap" Text="{TemplateBinding Text}"></TextBlock>

</Viewbox>

<Viewbox xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" Stretch="Fill" Visibility="{TemplateBinding State,Converter={StaticResource TileControlStateToVisibilityConverter},ConverterParameter=2}">

<Canvas Name="svg5666" Width="512" Height="512">

<Canvas.RenderTransform>

<TranslateTransform X="0" Y="0"/>

</Canvas.RenderTransform>

<Canvas.Resources/>

<!--Unknown tag: metadata-->

<!--Unknown tag: sodipodi:namedview-->

<Canvas Name="g5668"/>

<Canvas Name="g5736">

<Canvas.RenderTransform>

<MatrixTransform Matrix="1.0542309 0 0 1.371993 -276.9426 -412.40735"/>

</Canvas.RenderTransform>

<Path xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Name="path5670" Fill="#FF000000">

<Path.Data>

<PathGeometry Figures="m 659.89235 419.1727 81.49 56.002 5.079 -10.189 z" FillRule="NonZero"/>

</Path.Data>

</Path>

<Path xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Name="path5672" Fill="#FF000000">

<Path.Data>

<PathGeometry Figures="m 731.20435 393.7057 0 -10.189 -66.222 15.268 z" FillRule="NonZero"/>

</Path.Data>

</Path>

<Path xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Name="path5674" Fill="#FF000000">

<Path.Data>

<PathGeometry Figures="m 552.98735 302.0477 -10.23 10.178 66.181 66.192 z" FillRule="NonZero"/>

</Path.Data>

</Path>

<Path xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Name="path5676" Fill="#FF000000">

<Path.Data>

<PathGeometry Figures="m 710.82635 302.0477 -56.034 76.37 66.233 -66.192 z" FillRule="NonZero"/>

</Path.Data>

</Path>

<Path xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Name="path5678" Fill="#FF000000">

<Path.Data>

<PathGeometry Figures="m 593.69135 526.1087 25.447 -91.678 -40.724 86.579 z" FillRule="NonZero"/>

</Path.Data>

</Path>

<Path xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Name="path5680" Fill="#FF000000">

<Path.Data>

<PathGeometry Figures="m 629.32635 373.3077 10.178 -71.26 -15.257 0 z" FillRule="NonZero"/>

</Path.Data>

</Path>

<Path xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Name="path5682" Fill="#FF000000">

<Path.Data>

<PathGeometry Figures="m 659.89235 500.6517 10.178 0 -25.457 -66.222 z" FillRule="NonZero"/>

</Path.Data>

</Path>

<Path xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Name="path5684" Fill="#FF000000">

<Path.Data>

<PathGeometry Figures="m 624.24735 403.8837 -5.099 0 c -10.178 5.1 -15.257 10.179 -25.436 10.179 -15.278 0 -40.724 -15.268 -71.301 -40.745 l -15.268 -10.158 c -30.566 -25.467 -50.934 -40.755 -76.4 -40.755 -15.268 0 -25.477 5.1 -35.645 20.388 -10.178 10.178 -15.268 20.368 -15.268 40.745 l 0 5.048 c -15.288 0 -25.436 5.12 -30.576 5.12 l 0 35.645 c -55.992 25.446 -86.558 61.112 -86.558 117.095 0 35.676 15.278 66.233 35.645 91.699 25.457 20.357 56.054 35.625 91.658 35.625 35.645 0 66.222 -15.257 86.579 -35.625 25.457 -25.467 40.745 -56.023 40.745 -91.699 0 -55.982 -30.587 -96.738 -86.579 -117.094 l 0 -35.645 c -10.199 0 -20.368 -5.12 -30.566 -5.12 0 -30.515 10.199 -40.714 30.566 -40.714 15.278 0 35.656 10.189 61.092 30.535 l 15.288 10.178 c 35.635 30.577 61.082 45.844 86.569 45.844 10.178 0 20.347 0 30.546 -5.069 5.079 -5.11 10.199 -10.189 10.199 -15.298 0.01 -10.179 0.01 -10.179 -10.189 -10.179 z m -266.014 248.433 c -83.569 -48.384 -69.181 -130.447 -49.9 -167.076 l 26.02 11.09 c 0 0 -0.348 1.065 -0.368 1.096 l -0.512 1.003 c -1.731 3.338 -41.646 82.637 38.994 129.341 l -14.234 24.546 z" FillRule="NonZero"/>

</Path.Data>

</Path>

</Canvas>

</Canvas>

</Viewbox>

</Grid>

</Border>

</ControlTemplate>

</Setter.Value>

</Setter>

</Style>

</ResourceDictionary>

El archivo Generic.xaml es la especificación visual, en XAML, del componente que estamos construyendo. Aunque largo en él se pueden distinguir tres “ViewBox” cada uno de los cuales representa un estado visual del control. El primero dibuja una loseta, el segundo muestra un texto y el tercero una bomba. Los tres o sus contenidos tienen diversas propiedades enlazadas mediante TemplateBinding a las propiedades de la clase del control, pero destaca especialmente la de la propiedad Visibility. Esta se enlaza con la propiedad State que es debidamente convertida mediante el conversor *TileControlStateToVisibilityConverter*  debidamente parametrizado mediante el ConverterParameter, que indica con cuál de las tres vistas alternativas del objeto estamos enlazando, de esa manera la vista del control cambia en relación a su estado.

## Paso 2: Probar el control

1. En el explorador de soluciones hacer clic derecho sobre la solución y seleccionar el menú *Agregar* y la opción *Nuevo Proyecto.*
2. En la ventana emergente seleccionar dentro del conjunto de plantillas *Instalado* la rama *Visual Basic->Prueba* o bien *Visual C#->Prueba* según sea la elección de nuestro lenguaje y la plantilla *Proyecto de prueba unitaria*, con el nombre asignado por defecto.
3. Pulsar el botón *Aceptar* y crear el proyecto.
4. En el explorador de soluciones hacer clic derecho sobre el nodo *UnitTestProject1*  y seleccionar el menú *Agregar y* la opción *Referencia..*.
5. En la ventana emergente seleccionar el conjunto de *Solución->Proyectos* y marcar la referencia con el nombre *TileControlLib*
6. En la ventana emergente seleccionar el conjunto de *Solución->Ensamblados* y marcar las referencias con los nombres *PresentationFramework, PresentationCore, WindowsBase y System.Xaml*
7. Pulsar el botón *Aceptar*
8. En el explorador de soluciones hacer clic derecho sobre el archivo *UnitTest1.cs* y seleccionar la opción *Cambiar nombre*
9. Cambiar el nombre del archivo a TileControlUnitTest.cs
10. En el diálogo emergente en el que se nos solicita aprobación para cambiar las referencias en todo el código seleccionar el botón Sí.
11. Hacer doble clic sobre el archivo TileControlUnitTest.cs para abrirlo.
12. Sustituir el contenido de la clase TileControlUnitTest por lo siguiente:

C#

[TestMethod]

public void DiscoverEventTestMethod()

{

TileControlLib.TileControl control = new TileControlLib.TileControl();

control.State = TileControlLib.TileControlState.Covered;

control.Discover += delegate(object sender,

System.Windows.RoutedPropertyChangedEventArgs<TileControlLib.TileControlState> e)

{

if (e.NewValue == e.OldValue)

Assert.Fail();

else

Assert.IsTrue(true);

};

control.State = TileControlLib.TileControlState.Bomb;

}

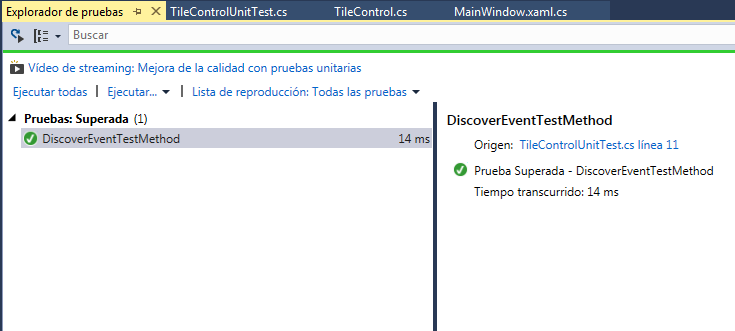
Un proyecto de pruebas es una biblioteca que contiene clases marcadas con el atributo TestClassAttribute que son instanciadas por Visual Studio. Dichas clases contienen métodos de pruebas, marcados con el atributo TestMethodAttribute. Dichos métodos son ejecutados por Visual Studio, el cual espera una respuesta como invocación de uno de los métodos de la clase Assert, que le comunica el resultado de la prueba.

El concepto de prueba unitaria representa el mismo “ejercicio” de código que el que realiza un desarrollador con las pruebas que realiza al desarrollar. La principal diferencia es que la prueba unitaria queda encapsulada en un método para poder ser invocada sistemáticamente a lo largo del tiempo.

1. En el editor de código hacer clic derecho sobre el nombre del método DiscoverEventTestMethod y seleccionar ejecutar pruebas.



1. En la nueva pestaña creada observar el resultado de la ejecución.



## Paso 3: Finalizar la aplicación

1. En el explorador de soluciones localizar el nodo MinesweeperWPF y hacer clic derecho sobre el seleccionando el menú Agregar y seleccionando Referencia, seleccionando en el grupo Solución el ensamblado TileControlLib.
2. Abrir el archivo MainWindow.xml y reemplazar el contenido por:

XAML

<Window x:Class="MinesweeperWPF.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

Title="MainWindow" Height="350" Width="525"

xmlns:lib="clr-namespace:TileControlLib;assembly=TileControlLib"

xmlns:local="clr-namespace:MinesweeperWPF">

<Window.Resources>

<CommandBinding x:Key="RestartCommand" Command="{x:Static local:MainWindow.RestartCommand}"

Executed="reStartCommandExecute" CanExecute="CommandBinding\_CanExecute"/>

</Window.Resources>

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="20" />

<RowDefinition />

</Grid.RowDefinitions>

<Grid Height="20">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="120" />

<ColumnDefinition Width="50" />

<ColumnDefinition />

</Grid.ColumnDefinitions>

<TextBox Grid.Column="0" Text="{Binding Path=Size}"></TextBox>

<Button Grid.Column="1" Command="{x:Static local:MainWindow.RestartCommand}">Iniciar</Button>

</Grid>

<Grid Name="field" Grid.Row="1">

<Grid.RowDefinitions>

<RowDefinition />

<RowDefinition />

<RowDefinition />

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition />

<ColumnDefinition />

<ColumnDefinition />

</Grid.ColumnDefinitions>

<lib:TileControl Grid.Row="0" Grid.Column="0" Discover="TileControl\_Discover" />

<lib:TileControl Grid.Row="0" Grid.Column="1" Discover="TileControl\_Discover" />

<lib:TileControl Grid.Row="0" Grid.Column="2" Discover="TileControl\_Discover" />

<lib:TileControl Grid.Row="1" Grid.Column="0" Discover="TileControl\_Discover" />

<lib:TileControl Grid.Row="1" Grid.Column="1" Discover="TileControl\_Discover" />

<lib:TileControl Grid.Row="1" Grid.Column="2" Discover="TileControl\_Discover" />

<lib:TileControl Grid.Row="2" Grid.Column="0" Discover="TileControl\_Discover" />

<lib:TileControl Grid.Row="2" Grid.Column="1" Discover="TileControl\_Discover" />

<lib:TileControl Grid.Row="2" Grid.Column="2" Discover="TileControl\_Discover" />

</Grid>

</Grid>

</Window>

1. Abrir el archivo MainWindow.cs según convenga y sustituir el código por el siguiente:

C#

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

using TileControlLib;

namespace MinesweeperWPF

{

public partial class MainWindow : Window

{

public static RoutedCommand RestartCommand = new RoutedCommand();

public MainWindow()

{

InitializeComponent();

this.DataContext = this;

CleanField();

GenerateField();

SpreadBombs();

CommandBindings.Add(new CommandBinding(RestartCommand, reStartCommandExecute, CommandBinding\_CanExecute));

}

public void reStartCommandExecute(object sender, ExecutedRoutedEventArgs e)

{

CleanField();

GenerateField();

SpreadBombs();

}

public void CommandBinding\_CanExecute(object sender, CanExecuteRoutedEventArgs e)

{

e.CanExecute = true;

}

public void CleanField()

{

field.Children.Clear();

field.ColumnDefinitions.Clear();

field.RowDefinitions.Clear();

}

public void GenerateField()

{

for (int i = 0; i < this.Size; i++)

field.ColumnDefinitions.Add(new ColumnDefinition());

for (int j = 0; j < this.Size; j++)

field.RowDefinitions.Add(new RowDefinition());

for (int i = 0; i < this.Size; i++)

for (int j = 0; j < this.Size; j++)

{

TileControl control = new TileControl();

control.Discover += TileControl\_Discover;

field.Children.Add(control);

Grid.SetRow(control, i);

Grid.SetColumn(control, j);

}

}

public int Size

{

get { return (int)GetValue(SizeProperty); }

set { SetValue(SizeProperty, value); }

}

public Grid Field

{

get

{

return this.field;

}

}

public static readonly DependencyProperty SizeProperty =

DependencyProperty.Register("Size", typeof(int), typeof(MainWindow), new PropertyMetadata(10));

int emptyTiles = 0;

public void SpreadBombs()

{

int bombs = Convert.ToInt32(Math.Truncate((decimal)(Math.Pow(this.Size, 2)) / 4));

emptyTiles = Convert.ToInt32(Math.Pow(this.Size, 2)) - bombs;

int e = 0;

Random generator = new Random();

for (int i = 0; i < bombs; i++)

{

do

{

e = generator.Next(field.Children.Count);

} while (((TileControl)field.Children[e]).IsBomb);

((TileControl)field.Children[e]).IsBomb = true;

}

foreach (UIElement element in field.Children)

{

((TileControl)element).Text = GetSuroundingBombs(Grid.GetRow(element), Grid.GetColumn(element));

}

}

public string GetSuroundingBombs(int row, int column)

{

int result = 0;

int rows = field.RowDefinitions.Count();

int columns = field.ColumnDefinitions.Count();

for (int x = row - 1; x <= row + 1; x++)

for (int y = column - 1; y <= column + 1; y++)

{

if ((x < 0) || (x >= rows) || (y < 0) || (y >= columns))

continue;

else

{

var elements = field.Children.Cast<TileControl>().

Where(e => Grid.GetRow(e) == x && Grid.GetColumn(e) == y && e.IsBomb);

result += elements.Any() ? 1 : 0;

}

}

return result.ToString();

}

private void TileControl\_Discover(object sender, RoutedPropertyChangedEventArgs<TileControlLib.TileControlState> e)

{

if (e.NewValue == TileControlState.Bomb)

MessageBox.Show("Juego terminado. Tu pierdes");

else

{

emptyTiles--;

if (emptyTiles == 0)

MessageBox.Show("Juego terminado. Tu Ganas");

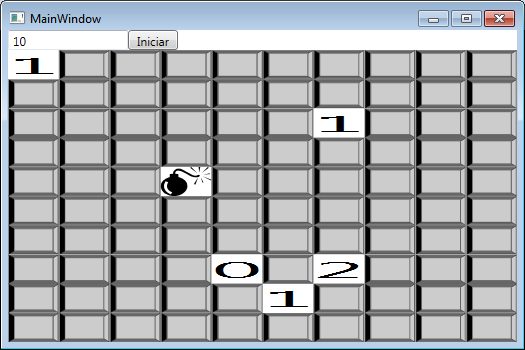
}

}

}

}

1. Compilar y probar la solución.



## Paso 4: Probar la aplicación

1. En el explorador de soluciones localizar el nodo *UnitTestProject1*  y hacer clic derecho sobre el seleccionando el menú Agregar y seleccionando Referencia, seleccionando en el grupo Solución el ensamblado MinesweeperWPF.
2. Agregar una nueva Prueba unitaria renombrándola como MainWindowUnitTest.
3. Dentro de la clase disponer los siguientes métodos de prueba:

C#

[TestMethod]

public void CleanFieldTestMethod()

{

MinesweeperWPF.MainWindow window = new MinesweeperWPF.MainWindow();

window.Size = 10;

window.CleanField();

Assert.AreEqual(0, window.Field.ColumnDefinitions.Count);

Assert.AreEqual(0, window.Field.RowDefinitions.Count);

Assert.AreEqual(0, window.Field.Children.Count);

}

[TestMethod]

public void GenerateFieldTestMethod()

{

MinesweeperWPF.MainWindow window = new MinesweeperWPF.MainWindow();

window.Size = 10;

window.CleanField();

window.GenerateField();

Assert.AreEqual(10, window.Field.ColumnDefinitions.Count);

Assert.AreEqual(10, window.Field.RowDefinitions.Count);

Assert.AreEqual(100, window.Field.Children.Count);

}

1. Ejecutar los métodos de prueba

## Paso 5: Fake Objects: Stubs

1. En el explorador de soluciones localizar el nodo MinesweeperWPF y hacer clic derecho sobre el seleccionando el menú Agregar y seleccionando Nuevo Elemento, buscar dentro del grupo Visual C# o Visual Basic según convenga la plantilla Interfaz, dándole el nombre IGenerator y pulsar el botón Agregar.
2. Sustituir el contenido del interfaz recién creado por el siguiente:

C#

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace MinesweeperWPF

{

public interface IGenerator

{

int Next(int number);

}

}

1. En la clase MainWindow.cs agregar el método:

C#

public void SpreadBombs(IGenerator generator)

{

int bombs = Convert.ToInt32(Math.Truncate((decimal)(Math.Pow(this.Size, 2)) / 4));

emptyTiles = Convert.ToInt32(Math.Pow(this.Size, 2)) - bombs;

int e = 0;

for (int i = 0; i < bombs; i++)

{

do

{

e = generator.Next(field.Children.Count);

} while (((TileControl)field.Children[e]).IsBomb);

((TileControl)field.Children[e]).IsBomb = true;

}

foreach (UIElement element in field.Children)

{

((TileControl)element).Text = GetSuroundingBombs(Grid.GetRow(element), Grid.GetColumn(element));

}

}

1. En el explorador de soluciones localizar el nodo *UnitTestProject1*  y desplegar el nodo references dentro de él. Hacer clic derecho sobre elemento MinesweeperWPF seleccionando Agregar ensamblado de Fakes.
2. En el archivo MainWindowUnitTest.cs, incluir dentro de la clase MainWindowUnitText el siguiente método de pruebas.

C#

[TestMethod]

public void SpreadBombsTestMethod()

{

int numero = 0;

MinesweeperWPF.Fakes.StubIGenerator generator = new MinesweeperWPF.Fakes.StubIGenerator();

generator.NextInt32 = (n) => {

numero++;

return numero;

};

MinesweeperWPF.MainWindow window = new MinesweeperWPF.MainWindow();

window.Size = 10;

window.CleanField();

window.GenerateField();

window.SpreadBombs(generator);

int bombs = 0;

for (int i = 0; i < window.Field.Children.Count; i++)

if (((TileControlLib.TileControl)window.Field.Children[i]).IsBomb)

bombs++;

Assert.AreEqual(25, bombs);

Assert.AreEqual(true, ((TileControlLib.TileControl)window.Field.Children[25]).IsBomb);

Assert.AreEqual(false, ((TileControlLib.TileControl)window.Field.Children[26]).IsBomb);

}

1. Ejecutar las pruebas.

## Paso 6: Fake Objects: Shims

1. En el archivo MainWindowUnitTest.cs, incluir dentro de la clase MainWindowUnitText el siguiente método de pruebas.

C#

[TestMethod]

public void GetSuroundingBombsTestMethod()

{

using (ShimsContext.Create())

{

MinesweeperWPF.MainWindow window = new MinesweeperWPF.MainWindow();

new MinesweeperWPF.Fakes.ShimMainWindow(window)

{

SpreadBombs = () =>

{

((TileControlLib.TileControl)window.Field.Children[0]).IsBomb = true;

((TileControlLib.TileControl)window.Field.Children[1]).IsBomb = true;

}

};

window.Size = 10;

window.CleanField();

window.GenerateField();

window.SpreadBombs();

Assert.AreEqual("2", window.GetSuroundingBombs(1, 1));

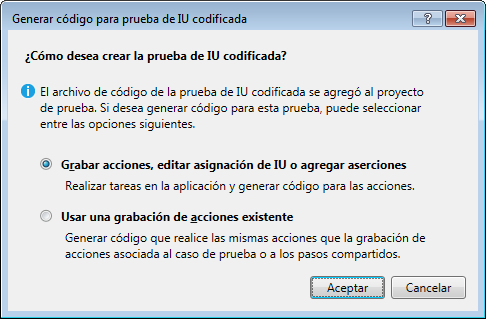
}

} }

1. Ejecutar las pruebas.

## Paso 7: Pruebas de Interfaz

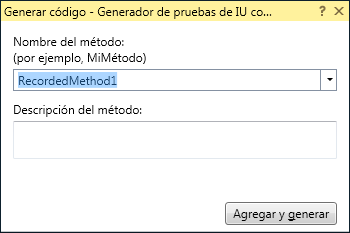
1. En el escritorio de la máquina crear un nuevo acceso directo al ejecutable de la aplicación MinesweeperWPF (comúnmente “C:\HOL\BuscaminasWPF\MinesweeperWPF\bin\Debug\MinesweeperWPF.exe”).
2. En el explorador de soluciones localizar el nodo *UnitTestProject1*  y hacer clic derecho sobre el seleccionando el menú Agregar una nueva Prueba de IU Codificada.
3. En la ventana emergente seleccionar la opción Grabar acciones, editar asignación de IU o agregar aserciones.



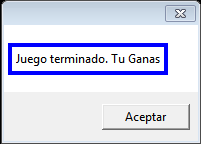
1. Pulsar el botón Aceptar.
2. Tras minimizarse Visual Studio se nos muestra el escritorio con una pequeña herramienta para la grabación.



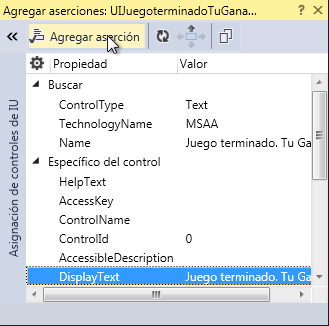
1. Pulsar sobre el botón de Iniciar Grabación 
2. Hacer doble clic sobre el acceso directo de la aplicación antes creado.
3. Una vez en la aplicación cambiar el valor del TextBox de 10 a 1 y pulsar el botón Iniciar
4. Pulsar sobre la única loseta del juego
5. Pulsar sobre el botón Pausar la Grabación 
6. A continuación pulsar sobre el botón Generar código  y en la ventana emergente pulsar el botón Agregar y generar



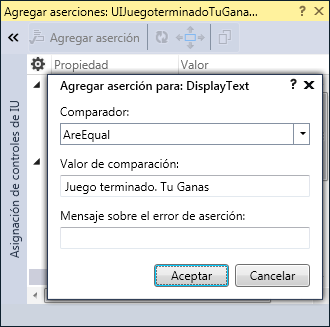
1. A continuación pulsar sobre el botón Generar nueva aserción  y sin dejar de presionar arrastrar hasta el texto de la ventana modal



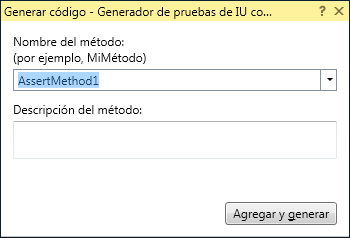
1. Tras esto aparece una ventana emergente con el nombre Agregar aserciones, en ella hacer clic sobre la propiedad DisplayText y pulsar el botón Agregar aserción



1. Sobre la ventana emergente pulsar el botón aceptar



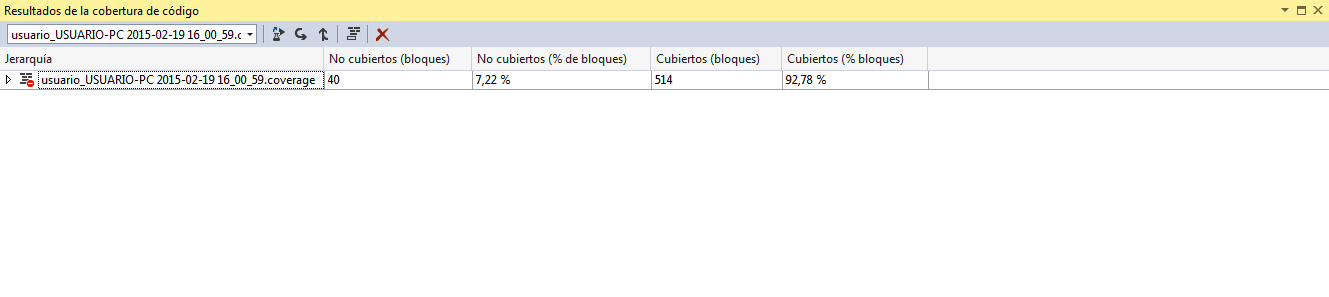
1. A continuación pulsar sobre el botón Generar código  y en la ventana emergente pulsar el botón Agregar y generar



1. Pulsar sobre el botón de Iniciar Grabación 
2. Cerrar la aplicación, cerrando primero el diálogo modal.
3. Pulsar sobre el botón Pausar la Grabación 
4. A continuación pulsar sobre el botón Generar código  y en la ventana emergente pulsar el botón Agregar y generar.
5. Por ultimo cerrar la herramienta de grabación.
6. Ya en Visual Studio ejecutar la prueba buscando y abriendo el archivo CodedUITest1.cs ejecutando la prueba CodedUITestMethod1 como en ocasiones anteriores.

## Paso 8: Cobertura de pruebas

1. Buscar el menú de Visual Studio llamado Prueba->Ventanas y hacer clic sobre el elemento Explorador de pruebas.
2. Pulsar sobre el menú Ejecutar->Analizar cobertura de código para todas las pruebas
3. Pulsar sobre el menú de Visual Studio llamado Prueba->Ventanas y hacer clic sobre el Resultados de la cobertura de código.

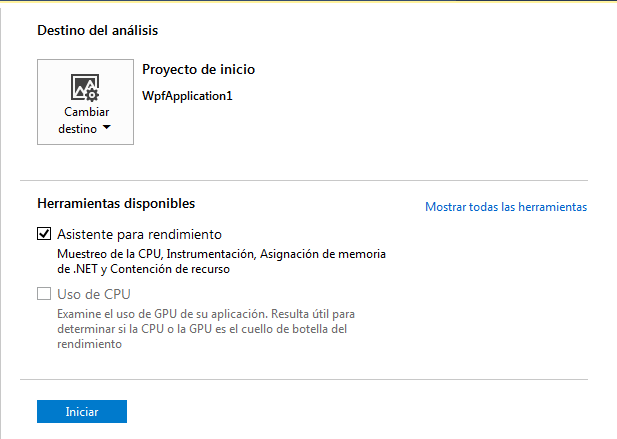


# Optimizando el código: Mejorando el rendimiento

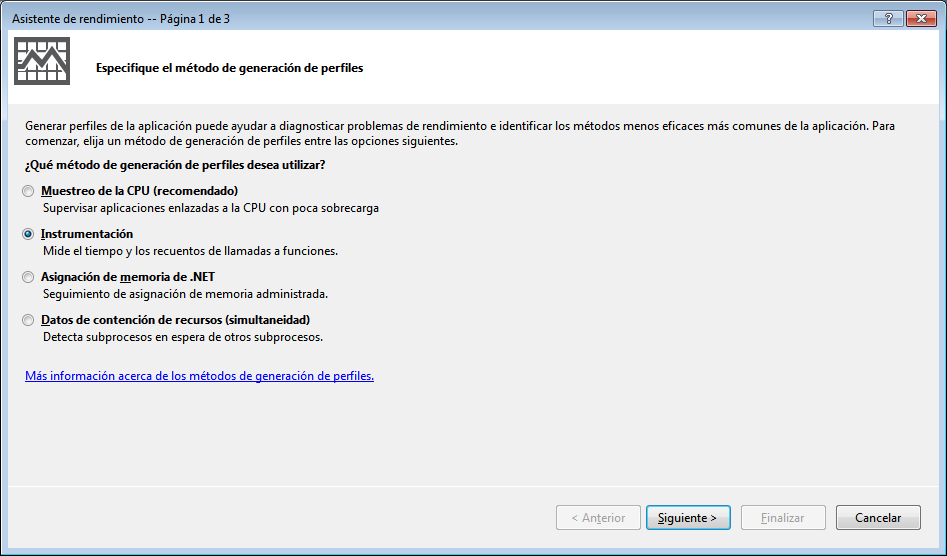
El “Profiler” o Explorador de rendimiento es una aplicación que nos permite detectar los cuellos de botella de una aplicación y con ellos los caminos de mayor esfuerzo. Señalándolos podemos descubrir donde al realizar el esfuerzo de optimización podemos obtener mejores resultados.

## Paso 1: Usando el Profiler para localizar los cuellos de botella

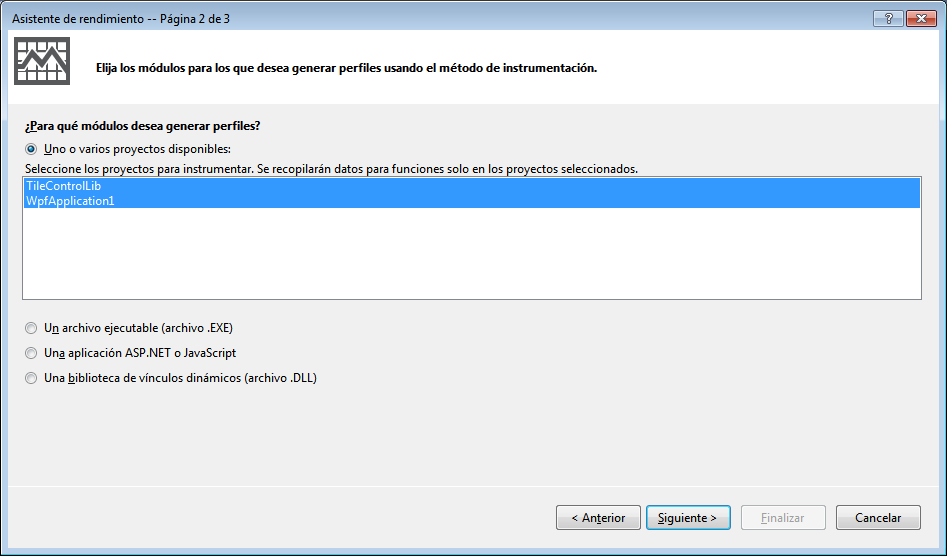
1. Pulsar sobre el menú de Visual Studio llamado Analizar y hacer clic sobre el elemento Rendimiento y diagnósticos.
2. En la pestaña de Rendimiento y Diagnósticos, comprobar que está marcada la opción de Asistente para rendimiento y pulsar el botón Iniciar.



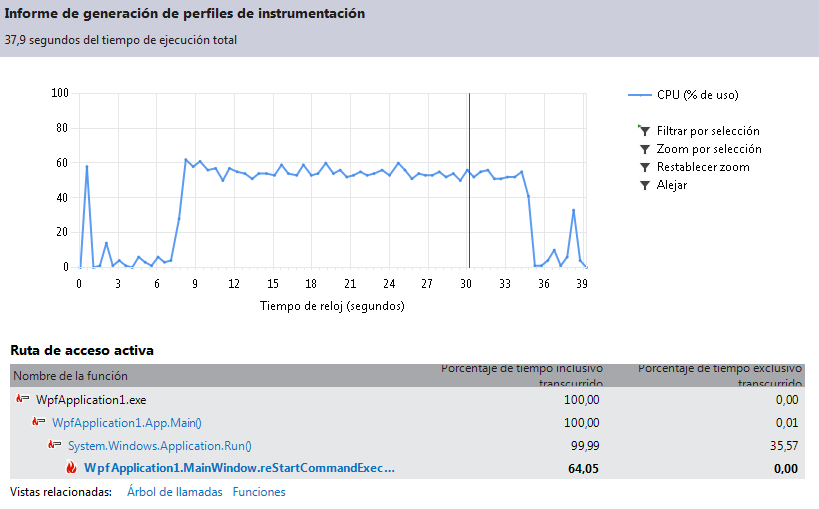
1. En el asistente seleccionar la opción Instrumentación y pulsar siguiente



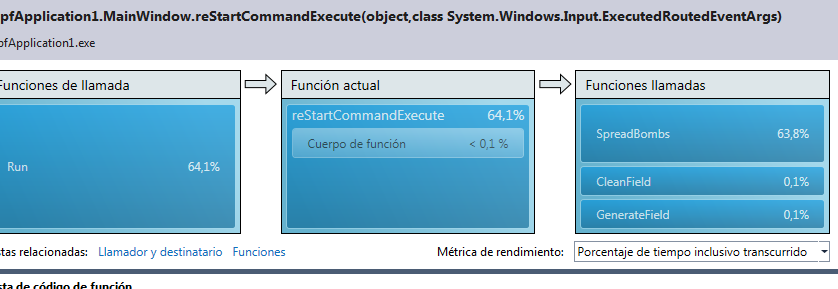
1. En el paso 2 seleccionar la opción de Uno o varios proyectos disponibles y seleccionar sólo el proyecto principal.



1. Pulsar siguiente y después finalizar
2. Esto lanzará una instancia de la aplicación, en ella escribiremos 40 sobre el Textbox y pulsaremos el Botón Iniciar. Por último cerraremos la ventana.
3. Esto producirá la finalización de la sesión de optimización y la generación automática de un informe. En él, y entre otras cosas se nos indica la “ruta caliente” de ejecución, o ejecución de mayor coste. Hacer clic sobre la función de mayor coste.



1. Con esto se nos muestra el árbol de invocación en un gráfico de cajas y en la correspondiente a llamadas pulsamos sobre la de mayor coste.



1. Esto nos muestra el código de la función

## Paso 2: Mejorando el código

1. En el clase MainWindow.cs localizar el método SpreadBombs() y sustituir el siguiente código :

C#

for (int i = 0; i < bombs; i++)

{

do

{

e = generator.Next(field.Children.Count);

} while (((TileControl)field.Children[e]).IsBomb);

((TileControl)field.Children[e]).IsBomb = true;

}

1. Por este:

C#

List<TileControl> list = field.Children.Cast<TileControl>().ToList();

for (int i = 0; i < bombs; i++)

{

do

{

e = generator.Next(list.Count);

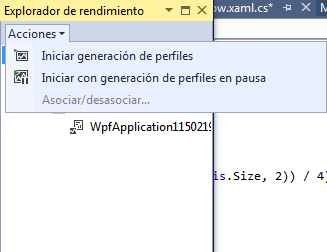
} while (list[e].IsBomb);

list[e].IsBomb = true;

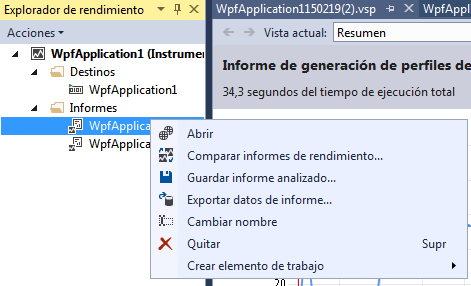
list.RemoveAt(e);

}

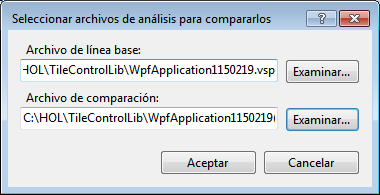
1. Pulsar sobre el menú de Visual Studio llamado Analizar->Ventanas y hacer clic sobre el elemento Explorador de rendimiento. Pulsar sobre la acción Acciones->Iniciar generación de perfiles



1. Repetir la operatividad de la sesión de rendimiento anterior.
2. Una vez finalizada y cerrada la aplicación en el Explorador de rendimiento, en el nodo Informes hacer clic derecho sobre uno de ellos y seleccionar la opción Comparar informes de rendimiento.



1. Configurar en la ventana el segundo de los informes generados y pulsar el botón Aceptar



1. Observar los resultados