

**Tema**

TALLER HILOS PELOTAS CON MVC DOTNET

**Tutor**

Ing. Eduardo Mauricio Campaña Ortega

MIS. MDU.CCNA. CCIA.

PhD. (c) Ingeniería de Software

PhD. (c) Seguridad Información

**Fecha**

29/12/2022

TALLER HILOS PELOTAS CON MVC DOTNET

[1 INTRODUCCIÓN 6](#_Toc124904096)

[2 OBJETIVOS 6](#_Toc124904097)

[2.1 OBJETIVO GENERAL 6](#_Toc124904098)

[2.2 OBJETIVOS ESPECÍFICOS 6](#_Toc124904099)

[3 MARCO TEÓRICO 6](#_Toc124904100)

[3.1 PROCESOS 6](#_Toc124904101)

[3.2 HILOS 7](#_Toc124904102)

[3.2.1 GESTIÓN DE HILOS 8](#_Toc124904103)

[3.2.2 CONCURRENCIA 8](#_Toc124904104)

[3.2.3 PARALELISMO 9](#_Toc124904105)

[3.3 CUELLOS DE BOTELLA 10](#_Toc124904106)

[3.4 LENGUAJES DE PROGRAMACIÓN CONCURRENTE 11](#_Toc124904107)

[3.5 PLATAFORMA .NET 12](#_Toc124904108)

[3.5.1 LENGUAJE DE PROGRAMACIÓN C# 13](#_Toc124904109)

[3.5.2 HILOS CON C# 13](#_Toc124904110)

[3.5.3 WINDOWS FORMS 16](#_Toc124904111)

[4 PARTE PRÁCTICA 17](#_Toc124904112)

[4.1 CREACIÓN DEL PROYECTO 17](#_Toc124904113)

[4.2 ARQUITECTURA MVC PARA EL PROYECTO 20](#_Toc124904114)

[4.3 PRÁCTICA VIDEO 168 21](#_Toc124904115)

[4.3.1 CODIFICACIÓN MODELO 21](#_Toc124904116)

[4.3.2 CODIFICACIÓN VISTA 26](#_Toc124904117)

[4.3.3 CODIFICACIÓN CONTROLADOR 36](#_Toc124904118)

[4.4 PRÁCTICA VIDEO 169 39](#_Toc124904119)

[4.4.1 CODIFICACIÓN MODELO 39](#_Toc124904120)

[4.4.2 CODIFICACIÓN VISTA 39](#_Toc124904121)

[4.4.3 CODIFICACIÓN CONTROLADOR 41](#_Toc124904122)

[4.5 PRÁCTICA VIDEO 170 44](#_Toc124904123)

[4.5.1 CODIFICACIÓN VISTA 44](#_Toc124904124)

[4.5.2 CODIFICACIÓN CONTROLADOR 46](#_Toc124904125)

[4.6 ESTRUCTURA DE LA APLICACIÓN 49](#_Toc124904126)

[4.7 EJECUCIÓN DEL PROYECTO 49](#_Toc124904127)

[4.7.1 EJECUCIÓN VIDEO 168 49](#_Toc124904128)

[4.7.2 EJECUCIÓN VIDEO 169 50](#_Toc124904129)

[4.7.3 EJECUCIÓN VIDEO 170 51](#_Toc124904130)

[5 CONCLUSIONES 53](#_Toc124904131)

[6 RECOMENDACIONES 53](#_Toc124904132)

[7 REFERENCIAS 53](#_Toc124904133)

**INDICE DE IMÁGENES**

[Figura 1. Modelo básico de un proceso 7](#_Toc124904134)

[Figura 2. Hilos y Procesos 7](#_Toc124904135)

[Figura 3. Gestión de Hilos 8](#_Toc124904136)

[Figura 4. Concurrencia 9](#_Toc124904137)

[Figura 5. Esquema ideal paralelismo 10](#_Toc124904138)

[Figura 6. Esquema real paralelismo 10](#_Toc124904139)

[Figura 7. Cuello de botella 11](#_Toc124904140)

[Figura 8. Hitos de la Programación Concurrente 12](#_Toc124904141)

[Figura 9. Logotipo identificativo de .NET 12](#_Toc124904142)

[Figura 10. Logotipo identificativo de C# 13](#_Toc124904143)

[Figura 11. Logo identificativo de C# con Hilos 13](#_Toc124904144)

[Figura 12. Logotipo identificativo Windows Forms 17](#_Toc124904145)

[Figura 13. Ventana de proyectos Visual Studio 18](#_Toc124904146)

[Figura 14. Plantillas de proyectos Visual Studio 18](#_Toc124904147)

[Figura 15. Ingreso del nombre y carpeta para el programa 19](#_Toc124904148)

[Figura 16. Selección de la version del Framework .NET 19](#_Toc124904149)

[Figura 17. Archivos generados para el programa 20](#_Toc124904150)

[Figura 18. Creación de carpetas para MVC 20](#_Toc124904151)

[Figura 19. Arquitectura MVC para el programa 21](#_Toc124904152)

[Figura 20. Creación de Clase en el modelo 21](#_Toc124904153)

[Figura 21. Asignación del nombre PelotaModelo para la Clase 22](#_Toc124904154)

[Figura 22. Codificación de la clase PelotaModelo 22](#_Toc124904155)

[Figura 23. Creación de ventana en la carpeta vista 26](#_Toc124904156)

[Figura 24. Asignación del nombre PelotaVista para la Clase 26](#_Toc124904157)

[Figura 25. Ventana generada para PelotaVista 27](#_Toc124904158)

[Figura 26. Creación del panel para la ventana PelotaVista 27](#_Toc124904159)

[Figura 27. Panel generado en la ventana PelotaVista 28](#_Toc124904160)

[Figura 28. Redimensión del panel en la ventana 28](#_Toc124904161)

[Figura 29. Nombre para el panel creado 29](#_Toc124904162)

[Figura 30. Elemento botón en el panel Toolbox 29](#_Toc124904163)

[Figura 31. Texto para el botón Dale 29](#_Toc124904164)

[Figura 32. Nombre para el botón Dale 30](#_Toc124904165)

[Figura 33. Texto para el botón Salir 30](#_Toc124904166)

[Figura 34. Nombre para el botón Salir 31](#_Toc124904167)

[Figura 35. Acceder a la clase PelotaVista 31](#_Toc124904168)

[Figura 36. Codificación de la clase PelotaVista 32](#_Toc124904169)

[Figura 37. Color de fondo para el panel 34](#_Toc124904170)

[Figura 38. Codificación de la clase Program 34](#_Toc124904171)

[Figura 39. Borrado de la ventana inicial generada 36](#_Toc124904172)

[Figura 40. Creación de Clase en el controlador 36](#_Toc124904173)

[Figura 41. Asignación del nombre PelotaControlador para la Clase 37](#_Toc124904174)

[Figura 42. Codificación de la clase PelotaControlador 37](#_Toc124904175)

[Figura 43. Codificación del método pausar 39](#_Toc124904176)

[Figura 44. Agregar botón para detener la pelota 39](#_Toc124904177)

[Figura 45. Propiedades del botón Pausar 40](#_Toc124904178)

[Figura 46. Nombre del botón Pausar 40](#_Toc124904179)

[Figura 47. Texto para el botón Pausar 40](#_Toc124904180)

[Figura 48. Ingreso para el código de la Clase PelotaVista 41](#_Toc124904181)

[Figura 49. Método para obtener el botón Pausar 41](#_Toc124904182)

[Figura 50. Codificación para la Clase PelotaControlador 42](#_Toc124904183)

[Figura 51. Agregación de los botones para las nuevas pelotas 44](#_Toc124904184)

[Figura 52. Codificación de la clase PelotaVista 44](#_Toc124904185)

[Figura 53. Codificación de la clase PelotaControlador 46](#_Toc124904186)

[Figura 54. Métodos para eventos de cada pelota 46](#_Toc124904187)

[Figura 55. Estructura final de la aplicación web con MVC 49](#_Toc124904188)

[Figura 56. Ejecución del programa 49](#_Toc124904189)

[Figura 57. Ventana inicial de la práctica video 168 50](#_Toc124904190)

[Figura 58. Pelotas en movimiento práctica 168 50](#_Toc124904191)

[Figura 59. Ventana inicial de la práctica video 169 51](#_Toc124904192)

[Figura 60. Pelotas en movimiento práctica 169 51](#_Toc124904193)

[Figura 61. Ventana inicial de la práctica video 170 52](#_Toc124904194)

[Figura 62. Pelotas en movimiento práctica 170 52](#_Toc124904195)

**ÍNDICE DE TABLAS**

[Tabla 1. Estados de los hilos 8](#_Toc124904196)

[Tabla 2. Constructores de la Clase Thread 14](#_Toc124904197)

[Tabla 3. Propiedades de la Clase Thread 14](#_Toc124904198)

[Tabla 4. Métodos de la Clase Thread 14](#_Toc124904199)

[Tabla 5. Elementos para Windows Forms 16](#_Toc124904200)

[Tabla 6. Codificación del archivo PelotaModelo.cs práctica video 168 23](file:///C:\Users\santi\Downloads\HILOS_DOTNET_GRUPO05\TALLER_HILOS_PRACTICA1\DOCUMENTACION\TALLER_HILOS_PRACTICA1.docx#_Toc124904201)

[Tabla 7. Codificación del archivo PelotaVista.cs práctica video 168 33](file:///C:\Users\santi\Downloads\HILOS_DOTNET_GRUPO05\TALLER_HILOS_PRACTICA1\DOCUMENTACION\TALLER_HILOS_PRACTICA1.docx#_Toc124904202)

[Tabla 8. Codificación archivo Program.cs práctica video 168 35](file:///C:\Users\santi\Downloads\HILOS_DOTNET_GRUPO05\TALLER_HILOS_PRACTICA1\DOCUMENTACION\TALLER_HILOS_PRACTICA1.docx#_Toc124904203)

[Tabla 9. Codificación archivo PelotaControlador.cs práctica video 168 38](file:///C:\Users\santi\Downloads\HILOS_DOTNET_GRUPO05\TALLER_HILOS_PRACTICA1\DOCUMENTACION\TALLER_HILOS_PRACTICA1.docx#_Toc124904204)

[Tabla 10. Codificación del archivo PelotaModelo.cs práctica video 169 39](file:///C:\Users\santi\Downloads\HILOS_DOTNET_GRUPO05\TALLER_HILOS_PRACTICA1\DOCUMENTACION\TALLER_HILOS_PRACTICA1.docx#_Toc124904205)

[Tabla 11. Codificación del archivo PelotaVista.cs práctica video 169 41](file:///C:\Users\santi\Downloads\HILOS_DOTNET_GRUPO05\TALLER_HILOS_PRACTICA1\DOCUMENTACION\TALLER_HILOS_PRACTICA1.docx#_Toc124904206)

[Tabla 12. Codificación del archivo PelotaControlador.cs práctica video 169 43](file:///C:\Users\santi\Downloads\HILOS_DOTNET_GRUPO05\TALLER_HILOS_PRACTICA1\DOCUMENTACION\TALLER_HILOS_PRACTICA1.docx#_Toc124904207)

[Tabla 13. Codificación del archivo PelotaVista.cs práctica video 170 45](file:///C:\Users\santi\Downloads\HILOS_DOTNET_GRUPO05\TALLER_HILOS_PRACTICA1\DOCUMENTACION\TALLER_HILOS_PRACTICA1.docx#_Toc124904208)

[Tabla 14. Codificación del archivo PelotaControlador.cs práctica video 170 47](file:///C:\Users\santi\Downloads\HILOS_DOTNET_GRUPO05\TALLER_HILOS_PRACTICA1\DOCUMENTACION\TALLER_HILOS_PRACTICA1.docx#_Toc124904209)

# INTRODUCCIÓN

Un hilo también llamado Thread, proceso ligero o subproceso, es la unidad de ejecución de un proceso, y está asociado con una secuencia de instrucciones, un conjunto de registros y una pila (Elorduy, 2020). Los programas que no usan hilos son aquellos que solo tienen una secuencia de ejecución y ejecutan una tarea a la vez. Estos programas pueden ser útiles para tareas sencillas y rápidas, pero pueden ser ineficientes para tareas más complejas o que requieren la ejecución de varias tareas al mismo tiempo.

Los programas que usan hilos, por otro lado, tienen varias secuencias de ejecución que pueden ejecutar tareas de manera concurrente. Esto puede mejorar la eficiencia y la velocidad de la aplicación, ya que permite que varias tareas se ejecuten al mismo tiempo en lugar de tener que esperar a que se completen una a una.

Para implementar hilos en C#, se puede utilizar la clase Thread. Esta clase proporciona varias funciones para crear, controlar y sincronizar hilos. Es importante tener en cuenta que la ejecución concurrente de varios hilos puede ser complicada y requerir una buena gestión para evitar conflictos y errores. Por lo tanto, es importante entender cómo funcionan los hilos y cómo se pueden utilizar de manera efectiva en una aplicación.

# OBJETIVOS

## OBJETIVO GENERAL

Analizar los conceptos, beneficios y aplicaciones de los hilos en sistemas de software que ejecuten más de una tarea por medio de un ejemplo práctico y sencillo, utilizando el lenguaje de programación C#, la plataforma de desarrollo .NET y la arquitectura MVC (Modelo-Vista-Controlador), para implementarlos en sistemas que necesiten ejecutar varias tareas al mismo tiempo.

## OBJETIVOS ESPECÍFICOS

* Analizar los conceptos, ventajas, desventajas y usos de los hilos en sistemas de software complejos que ejecuten más de una tarea.
* Identificar los pasos para implementar un ejemplo del uso de hilos con el lenguaje de programación C#, la plataforma .NET y la arquitectura MVC (Modelo-Vista-Controlador).
* Documentar todo el proceso de desarrollo del ejemplo para tener una guía cuando se necesite implementar hilos en sistemas que ejecuten varias tareas al mismo tiempo.

# MARCO TEÓRICO

## PROCESOS

Un proceso es una entidad compuesta por un código ejecutable que se ejecuta en un orden secuencial, un conjunto de datos y una pila que se utiliza para pasar parámetros, restaurar llamadas recursivas o interrupciones, entre otros (Rossainz, 2015).

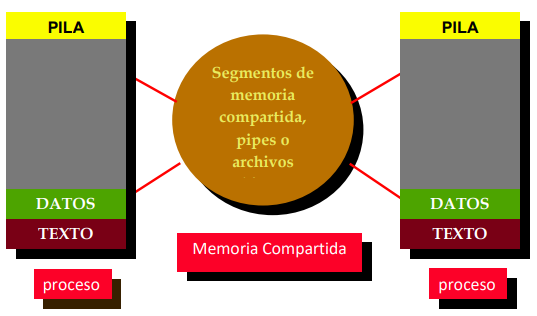


Figura . Modelo básico de un proceso

Los recursos para los procesos incluyen memoria, descriptores de archivos y recursos de E/S, y el estado incluye información como la posición en el código que se está ejecutando y las variables de estado del programa. Los procesos son administrados por el sistema operativo, que se encarga de asignar y liberar recursos, planificar la ejecución y controlar la comunicación entre procesos.

## HILOS

Un hilo es un solo flujo de ejecución de instrucciones dentro de un proceso, debido a que no pueden ejecutarse por sí solos, requieren la supervisión de un proceso (Nakayama, 2009). Los hilos se utilizan a menudo para realizar tareas en segundo plano o para realizar tareas que deben ejecutarse de manera simultánea. Por ejemplo, si una aplicación tiene una interfaz gráfica de usuario (GUI), se puede utilizar un hilo para actualizar la interfaz mientras se realizan otras tareas en segundo plano, como descargar datos de Internet o realizar cálculos.

Timeline

Description automatically generated with medium confidence

Figura . Hilos y Procesos

En algunos lenguajes de programación, los hilos se implementan como una parte integrada del lenguaje, mientras que en otros lenguajes se pueden utilizar bibliotecas o frameworks externos para trabajar con hilos. Los hilos también se pueden utilizar para dividir el trabajo en un sistema con múltiples núcleos o procesadores, lo que permite que el trabajo se ejecute de manera más rápida y eficiente.

### **GESTIÓN DE HILOS**

La gestión de hilos es una técnica importante en programación que permite la ejecución concurrente de diferentes tareas dentro de un mismo proceso. Esto puede mejorar la eficiencia y el rendimiento de una aplicación, especialmente en sistemas con múltiples núcleos o procesadores.

La gestión de hilos también puede ser complicada, ya que los hilos comparten el mismo espacio de memoria y recursos, lo que puede llevar a conflictos y errores si no se gestionan adecuadamente (Davizhe, 2014). Por ejemplo, si dos hilos intentan acceder a la misma variable al mismo tiempo, puede haber errores de sincronización. Por lo tanto, es importante utilizar técnicas de sincronización y control de acceso para garantizar la correcta ejecución de los hilos.

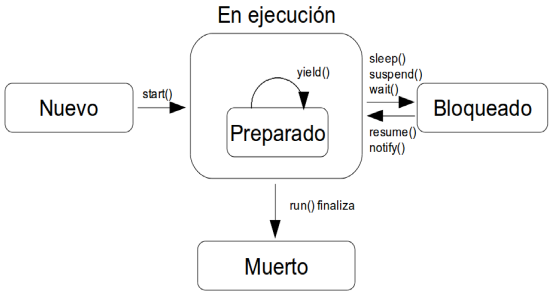


Figura . Gestión de Hilos

Durante el ciclo de vida de un Hilo, éste se puede encontrar en diferentes estados:

Tabla . Estados de los hilos

|  |  |
| --- | --- |
| **Estado** | **Descripción** |
| Ejecución | El hilo está siendo ejecutado. |
| Listo para ejecución | El hilo está en la lista de hilos listos para ser ejecutados, tan pronto disponga cada uno de tiempo de la CPU. |
| Suspensión | La ejecución del hilo es suspendida temporalmente. |
| Reanudación | Un hilo suspendido vuelve a ejecutarse continuando su tarea en el punto donde la dejó. |
| Bloqueado | Un hilo está en espera de un determinado recurso. |

### **CONCURRENCIA**

La concurrencia es la capacidad del CPU para ejecutar varias actividades o procesos al mismo tiempo (Blancarte, 2017). En la concurrencia los procesos no están relacionado, por lo tanto, cualquiera de los procesos puede iniciar y terminar sin afectar a los demás.

Diagram

Description automatically generated

Figura . Concurrencia

La programación concurrente tiene sus orígenes en los sistemas operativos y la programación de sistemas (Rossainz, 2015). Los primeros programas concurrentes fueron los propios sistemas operativos de multiprogramación, en los que un solo procesador de gran capacidad debía compartir su tiempo entre varios usuarios. En la década de 1960, se introdujeron en las computadoras dispositivos controladores independientes de entrada-salida llamados canales, que eran programables por sí mismos. Los sistemas operativos se organizaron como una colección de procesos ejecutándose concurrentemente, algunos en los canales y otros en el procesador principal o CPU. En esa época, la programación de sistemas con capacidades de concurrencia se hacía a bajo nivel, en ensamblador, debido a la falta de lenguajes de alto nivel con capacidades de concurrencia y se daba prioridad a la supuesta eficiencia del código escrito directamente en ensamblador. Sin embargo, en 1972, el lenguaje de alto nivel Concurrent Pascal, desarrollado por Brinch Hansen, abrió la puerta a otros lenguajes de alto nivel que incorporaban concurrencia.

### **PARALELISMO**

El paralelismo consiste en tomar el problema inicial, dividirlo en fracciones más pequeñas y procesarlas aprovechando al máximo la capacidad el procesador (Blancarte, 2017). El propósito del paralelismo es acelerar la ejecución de un programa que originalmente era secuencial, mediante el uso de varios procesadores. La idea principal es dividir un problema grande en varios más pequeños y distribuirlos entre los procesadores disponibles. La principal diferencia con la concurrencia es que todos los procesos están relacionados para resolver el mismo problema.

A picture containing diagram

Description automatically generated

Figura . Esquema ideal paralelismo

Sin embargo, si no se utiliza una estrategia de diseño adecuada, la distribución puede no dar los resultados esperados en términos de aumento de eficiencia del programa y puede complicar significativamente la programación y el mantenimiento. A veces, no es posible paralelizar todo el programa debido a elementos que no pueden ser paralelizados, y otras veces, la distribución de elementos del programa entre los procesadores no es equitativa, lo que resulta en un aumento muy pequeño en la eficiencia en términos de tiempo de ejecución.

Diagram

Description automatically generated

Figura . Esquema real paralelismo

## CUELLOS DE BOTELLA

Los cuellos de botella son situaciones en las que el rendimiento de un sistema se ve limitado por un componente o recurso específico (Microsoft, 2022). Esto puede ocurrir en una variedad de sistemas, como en una red de computadoras, un sistema de base de datos, un sistema de almacenamiento, una aplicación, etc.

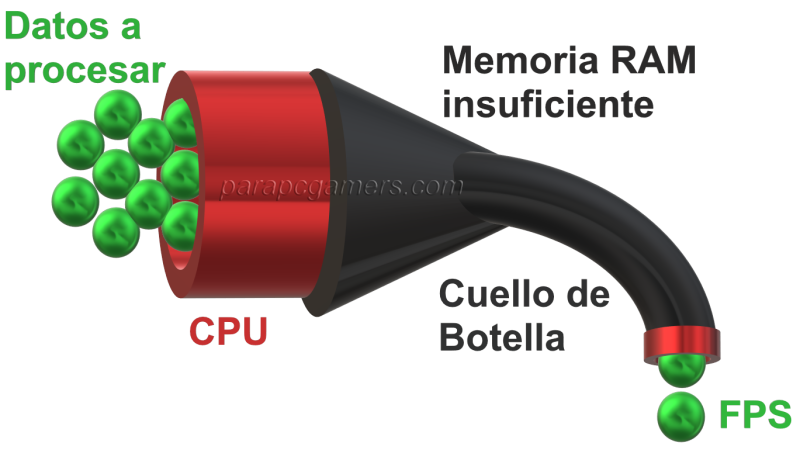


Figura . Cuello de botella

Los cuellos de botella pueden ser causados por una variedad de factores, como la creación, sobrecarga, bloqueo o sincronización ineficiente de varios hilos, mala configuración, un hardware obsoleto, una sobrecarga de tráfico, una mala gestión de recursos, etc. La identificación y resolución de cuellos de botella es importante para mejorar el rendimiento y la escalabilidad de un sistema. Esto se puede lograr a través de técnicas de diagnóstico de rendimiento, optimización de recursos, y mejoras en la infraestructura o en la arquitectura del sistema.

## LENGUAJES DE PROGRAMACIÓN CONCURRENTE

Los lenguajes de programación paralela y concurrente se basan en dos categorías: las abstracciones de programación concurrente basadas en exclusión mutua de accesos a una memoria individual y las abstracciones de procesos que se comunican mediante el envío de mensajes entre sí. El intercambio de mensajes es una acción de alto nivel que se puede implementar físicamente mediante procesadores distribuidos (Rossainz, 2015).

La programación concurrente ha ganado importancia y se utiliza ampliamente en la implementación de diferentes sistemas. Existen varios factores importantes que contribuyen a la importancia de la programación concurrente.

* La introducción del concepto de hilos (Threads) permite que los programas ejecuten de manera más rápida en comparación con aquellos que utilizan el concepto de procesos.
* Los lenguajes recientes como JAVA, que son lenguajes orientados a objetos de uso general, ofrecen soporte directo para la programación concurrente mediante la inclusión de primitivas específicas.
* La aparición de Internet ha aumentado el campo de desarrollo y uso de programas concurrentes, ya que cualquier aplicación de Internet, como un navegador o un chat, está programada utilizando técnicas de programación concurrente.



Figura . Hitos de la Programación Concurrente

Cada enfoque de programación concurrente y paralela requiere una configuración de hardware específica, y la mejor será aquella que se adapte a las primitivas del lenguaje utilizado para la programación paralela y concurrente. Actualmente existen muchos lenguajes de programación con primitivas diseñadas para manejar procesos de manera asíncrona o síncrona. La programación asíncrona se utiliza para la programación de multiprocesadores o sistemas distribuidos, mientras que las soluciones paralelas síncronas son adecuadas para el uso de arrays o vectores de procesadores. En cuanto a la programación orientada a objetos, se puede utilizar como un buen enfoque de programación paralela y concurrente ya que permite encapsular y abstraer patrones comunes de comunicación paralela y llevar esta abstracción hacia un estilo estructurado de programación paralela/concurrente.

## PLATAFORMA .NET

Microsoft .NET es una plataforma para el desarrollo de software lanzada por Microsoft con la finalidad de fusionar su amplio catálogo de productos, que va desde sus múltiples sistemas operativos hasta herramientas de desarrollo (Pellicer, 2021). Se utiliza para crear aplicaciones para dispositivos web, móviles, escritorio, juegos y de IoT (Internet de las cosas). El marco .NET proporciona un modelo de programación coherente y un entorno de tiempo de ejecución que simplifica el desarrollo y la implementación de aplicaciones. También incluye una gran biblioteca de componentes prediseñados que se pueden usar para crear e implementar rápidamente aplicaciones con un código mínimo.



Figura . Logotipo identificativo de .NET

### **LENGUAJE DE PROGRAMACIÓN C#**

C# es un lenguaje de programación de propósito general y de alto nivel desarrollado por Microsoft, soporta paradigmas de programación como el orientado a objetos, funcional, imperativos (Microsoft, 2022). Fue creado en el año 2000 como parte de la plataforma .NET y ha ganado popularidad gracias a su uso en aplicaciones de escritorio y en la creación de videojuegos.

Una de las ventajas de C# es su sintaxis clara y legible, lo que la convierte en un lenguaje fácil de aprender tanto para programadores principiantes como para aquellos con experiencia en otros lenguajes de programación. Además, C# es compatible con una amplia variedad de plataformas, incluyendo Windows, Linux y MacOS, lo que lo hace ideal para el desarrollo de aplicaciones multiplataforma.



Figura . Logotipo identificativo de C#

### **HILOS CON C#**

En C#, se pueden utilizar hilos para mejorar la eficiencia y el rendimiento de una aplicación al permitir que diferentes tareas se ejecuten al mismo tiempo en lugar de secuencialmente (Microsoft, 2022). Para utilizar hilos en C#, es necesario crear una clase que herede de la clase "Thread" y sobreescribir el método "Run()". Este método contiene el código que se ejecutará en el hilo. Una vez creada la clase, se puede instanciar y ejecutar mediante el método "Start()".

Los hilos también pueden tener prioridades, lo que permite controlar el orden en que se ejecutan en caso de que haya más de un hilo en ejecución al mismo tiempo. Además, es posible detener, pausar o reanudar la ejecución de un hilo en cualquier momento utilizando los métodos "Abort()", "Suspend()" y "Resume()", respectivamente.



Figura . Logo identificativo de C# con Hilos

**CONSTRUCTORES**

Tabla . Constructores de la Clase Thread

|  |  |
| --- | --- |
| **Constructor** | **Descripción** |
| Thread(ParameterizedThreadStart) | Inicializa una nueva instancia especificando un delegado para pasar un objeto al subproceso cuando este último se inicia. |
| Thread(ParameterizedThreadStart, Int32) | Inicializa una nueva instancia especificando un delegado que pase un objeto al subproceso cuando este último se inicia, especificando el tamaño de pila máximo para el subproceso. |
| Thread(ThreadStart) | Inicializa una nueva instancia de la clase. |
| Thread(ThreadStart, Int32) | Inicializa una nueva instancia especificando el tamaño de pila máximo para el subproceso. |

**PROPIEDADES**

Tabla . Propiedades de la Clase Thread

|  |  |
| --- | --- |
| **Propiedad** | **Descripción** |
| ApartmentState | Obsoleto.  Obtiene o establece el estado de apartamento de este subproceso. |
| CurrentCulture | Obtiene o establece la referencia cultural del subproceso actual. |
| CurrentPrincipal | Obtiene o establece la entidad de seguridad actual del subproceso (de la seguridad basada en roles). |
| [CurrentThread](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.currentthread?view=net-7.0#system-threading-thread-currentthread) | Obtiene el subproceso actualmente en ejecución. |
|  |  |
| [CurrentUICulture](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.currentuiculture?view=net-7.0#system-threading-thread-currentuiculture) | Obtiene o establece la referencia cultural actual utilizada por el administrador de recursos para buscar recursos específicos de la referencia cultural en tiempo de ejecución. |
| [ExecutionContext](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.executioncontext?view=net-7.0#system-threading-thread-executioncontext) | Obtiene un objeto [ExecutionContext](https://learn.microsoft.com/es-es/dotnet/api/system.threading.executioncontext?view=net-7.0) que contiene información sobre los distintos contextos del subproceso actual. |
| [IsAlive](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.isalive?view=net-7.0#system-threading-thread-isalive) | Obtiene un valor que indica el estado de ejecución del subproceso actual. |
| [IsBackground](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.isbackground?view=net-7.0#system-threading-thread-isbackground) | Obtiene o establece un valor que indica si un subproceso es o no un subproceso en segundo plano. |
| [IsThreadPoolThread](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.isthreadpoolthread?view=net-7.0#system-threading-thread-isthreadpoolthread) | Obtiene un valor que indica si un subproceso pertenece al grupo de subprocesos administrados o no. |
| [ManagedThreadId](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.managedthreadid?view=net-7.0#system-threading-thread-managedthreadid) | Obtiene un identificador único para el actual subproceso administrado. |
| [Name](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.name?view=net-7.0#system-threading-thread-name) | Obtiene o establece el nombre del subproceso. |
| [Priority](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.priority?view=net-7.0#system-threading-thread-priority) | Obtiene o establece un valor que indica la prioridad de programación de un subproceso. |
| [ThreadState](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.threadstate?view=net-7.0#system-threading-thread-threadstate) | Obtiene un valor que contiene los estados del subproceso actual. |

**MÉTODOS**

Tabla . Métodos de la Clase Thread

|  |  |
| --- | --- |
| **Métodos** | **Descripción** |
| [Abort()](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.abort?view=net-7.0#system-threading-thread-abort) | **Obsoleto.**  Produce una excepción [ThreadAbortException](https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadabortexception?view=net-7.0) en el subproceso en el que se invoca, para iniciar el proceso de finalización del subproceso. Normalmente, una llamada a este método finaliza el subproceso. |
| [Abort(Object)](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.abort?view=net-7.0#system-threading-thread-abort(system-object)) | **Obsoleto.**  Produce una excepción [ThreadAbortException](https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadabortexception?view=net-7.0) en el subproceso en el que se invoca, para iniciar el proceso de finalización del subproceso, proporcionando al mismo tiempo información sobre excepciones relativa a la terminación del subproceso. Normalmente, una llamada a este método finaliza el subproceso. |
| [AllocateDataSlot()](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.allocatedataslot?view=net-7.0#system-threading-thread-allocatedataslot) | Asigna una ranura de datos sin nombre en todos los subprocesos. Para mejorar el rendimiento, en su lugar use campos marcados con el atributo [ThreadStaticAttribute](https://learn.microsoft.com/es-es/dotnet/api/system.threadstaticattribute?view=net-7.0). |
| [AllocateNamedDataSlot(String)](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.allocatenameddataslot?view=net-7.0#system-threading-thread-allocatenameddataslot(system-string)) | Asigna una ranura de datos con nombre en todos los subprocesos. Para mejorar el rendimiento, en su lugar use campos marcados con el atributo [ThreadStaticAttribute](https://learn.microsoft.com/es-es/dotnet/api/system.threadstaticattribute?view=net-7.0). |
| [BeginCriticalRegion()](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.begincriticalregion?view=net-7.0#system-threading-thread-begincriticalregion) | Notifica a un host que la ejecución está a punto de entrar en una región del código donde los efectos de una anulación del subproceso o de una excepción no controlada podrían constituir un riesgo para otras tareas del dominio de aplicaciones. |
| [DisableComObjectEagerCleanup()](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.disablecomobjecteagercleanup?view=net-7.0#system-threading-thread-disablecomobjecteagercleanup) | Desactiva la limpieza automática de contenedores RCW (Runtime Callable Wrappers) para el subproceso actual. |
| [EndCriticalRegion()](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.endcriticalregion?view=net-7.0#system-threading-thread-endcriticalregion) | Notifica a un host que la ejecución está a punto de entrar en una región de código donde los efectos de una anulación del subproceso o de una excepción no controlada se limitan a la tarea actual. |
| [EndThreadAffinity()](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.endthreadaffinity?view=net-7.0#system-threading-thread-endthreadaffinity) | Notifica a un host que el código administrado ha terminado de ejecutar instrucciones que dependen de la identidad del subproceso del sistema operativo físico actual. |
| [Equals(Object)](https://learn.microsoft.com/es-es/dotnet/api/system.object.equals?view=net-7.0#system-object-equals(system-object)) | Determina si el objeto especificado es igual que el objeto actual.  (Heredado de [Object](https://learn.microsoft.com/es-es/dotnet/api/system.object?view=net-7.0)) |
| [Finalize()](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.finalize?view=net-7.0#system-threading-thread-finalize) | Se asegura de que los recursos se liberan y que se llevan a cabo otras operaciones de limpieza cuando el recolector de elementos no utilizados recupere el objeto [Thread](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread?view=net-7.0). |
| [Sleep(Int32)](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.sleep?view=net-7.0#system-threading-thread-sleep(system-int32)) | Suspende el subproceso actual durante el número de milisegundos especificado. |
| [Sleep(TimeSpan)](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.sleep?view=net-7.0#system-threading-thread-sleep(system-timespan)) | Suspende el subproceso actual durante la cantidad de tiempo especificada. |
| [SpinWait(Int32)](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.spinwait?view=net-7.0#system-threading-thread-spinwait(system-int32)) | Hace que un subproceso espere el número de veces definido por el parámetro iterations. |
| [Start()](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.start?view=net-7.0#system-threading-thread-start) | Hace que el sistema operativo cambie el estado de la instancia actual a [Running](https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadstate?view=net-7.0#system-threading-threadstate-running). |
| [Start(Object)](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.start?view=net-7.0#system-threading-thread-start(system-object)) | Hace que el sistema operativo cambie el estado de la instancia actual a [Running](https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadstate?view=net-7.0#system-threading-threadstate-running); también puede proporcionar un objeto que contiene datos para que los use el método ejecutado por el subproceso. |
| [Suspend()](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.suspend?view=net-7.0#system-threading-thread-suspend) | **Obsoleto.**  Suspende el subproceso o, si este ya se ha suspendido, no tiene efecto alguno. |
| [ToString()](https://learn.microsoft.com/es-es/dotnet/api/system.object.tostring?view=net-7.0#system-object-tostring) | Devuelve una cadena que representa el objeto actual.  (Heredado de [Object](https://learn.microsoft.com/es-es/dotnet/api/system.object?view=net-7.0)) |
| [TrySetApartmentState(ApartmentState)](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.trysetapartmentstate?view=net-7.0#system-threading-thread-trysetapartmentstate(system-threading-apartmentstate)) | Establece el estado del apartamento de un subproceso antes de iniciarse. |
| [UnsafeStart()](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.unsafestart?view=net-7.0#system-threading-thread-unsafestart) | Hace que el sistema operativo cambie el estado de la instancia actual a [Running](https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadstate?view=net-7.0#system-threading-threadstate-running). |
| [UnsafeStart(Object)](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.unsafestart?view=net-7.0#system-threading-thread-unsafestart(system-object)) | Hace que el sistema operativo cambie el estado de la instancia actual a [Running](https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadstate?view=net-7.0#system-threading-threadstate-running); también puede proporcionar un objeto que contiene datos para que los use el método ejecutado por el subproceso. |
| [Yield()](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.yield?view=net-7.0#system-threading-thread-yield) | Hace que el subproceso que realiza la llamada ceda la ejecución a otro subproceso que está listo para ejecutarse en el procesador actual. El sistema operativo selecciona el subproceso al que se va a ceder la ejecución. |

### **WINDOWS FORMS**

Windows Forms es una biblioteca de clases incluida en el marco de desarrollo .NET de Microsoft que permite crear aplicaciones de interfaz gráfica de usuario (GUI) para el sistema operativo Windows (Microsoft, 2022). Con Windows Forms, es posible crear aplicaciones de escritorio con una amplia variedad de controles y elementos gráficos, como botones, menús, etiquetas y tablas.

Tabla . Elementos para Windows Forms

|  |  |  |
| --- | --- | --- |
| **Nombre** | **Abreviatura** | **Descripción** |
| Botón | btn | Un elemento de interfaz de usuario que permite al usuario hacer clic para realizar una acción específica. |
| Caja de texto | txt | Un elemento de interfaz de usuario que permite al usuario escribir o editar texto. |
| Etiqueta | lbl | Un elemento de interfaz de usuario que muestra texto o imágenes para proporcionar información o contexto. |
| Panel | pnl | Un elemento de interfaz de usuario que agrupar otros controles y organizarlos de manera lógica. |
| Menú | mnu | Un elemento de interfaz de usuario que proporciona una lista de opciones para el usuario. |
| Barra de menús | mnubar | Un elemento de interfaz de usuario que proporciona una barra de menús con opciones para el usuario. |
| Lista | lst | Un elemento de interfaz de usuario que muestra una lista de elementos que el usuario puede seleccionar. |
| Cuadro de lista | lstbox | Un elemento de interfaz de usuario que muestra una lista de elementos que el usuario puede seleccionar. |
| Cuadro combinado | cbo | Un elemento de interfaz de usuario que combina un cuadro de lista y una caja de texto. El usuario puede seleccionar un elemento de la lista o escribir uno nuevo. |
| Cuadro de diálogo | dlg | Un elemento de interfaz de usuario que se muestra como una ventana emergente y proporciona opciones o información adicionales al usuario. |
| Árbol | trv | Un elemento de interfaz de usuario que muestra una lista jerárquica de elementos que el usuario puede expandir o contraer. |

Para utilizar Windows Forms en C#, es necesario importar la biblioteca de clases correspondiente y crear una clase que herede de la clase "Form". Esta clase se utiliza para definir la apariencia y el comportamiento de la ventana principal de la aplicación. A continuación, se pueden añadir controles y elementos gráficos a la ventana mediante el diseñador de formularios o mediante código.



Figura . Logotipo identificativo Windows Forms

# PARTE PRÁCTICA

Para esta práctica se implementará un programa que dibuje tres pelotas en una pantalla, a cada pelota se le asignara un hilo para que esta se mueva, se activara y pausara por medio de botones, además de incluir un botón adicional para salir de la aplicación.

## CREACIÓN DEL PROYECTO

1. Para comenzar con la creación del proyecto, abrir Visual Studio, seleccionar y dar clic en la opción Create a new project.

Graphical user interface, text, application

Description automatically generated

Figura . Ventana de proyectos Visual Studio

1. A continuación, buscar la plantilla para el proyecto, se requiere una interfaz gráfica para dibujar las pelotas, por lo tanto, seleccionar la opción Windows Form App y clic en Next.

Text

Description automatically generated

Figura . Plantillas de proyectos Visual Studio

1. En los campos que se muestran a continuación, ingresar el nombre del proyecto y la ubicación de almacenamiento para los archivos que se generen, luego clic en Next.

Graphical user interface, text, application

Description automatically generated

Figura . Ingreso del nombre y carpeta para el programa

1. En la siguiente ventana se debe seleccionar la version del Framework de .NET, en este caso la version 6.0 y clic en Create.

A screenshot of a computer

Description automatically generated with medium confidence

Figura . Selección de la version del Framework .NET

1. Por último, se muestra la estructura inicial del proyecto con algunas clases, archivos de configuración y un formulario que se crea de forma predeterminada.

A picture containing text, screenshot, monitor, computer

Description automatically generated

Figura . Archivos generados para el programa

## ARQUITECTURA MVC PARA EL PROYECTO

Para definir la arquitectura MVC (Modelo-Vista-Controlador), se crean tres carpetas para organizar y clasificar las clases del proyecto.

1. Para crear una carpeta, dirigirse a la sección Solution Explorer y dar clic derecho en el segundo item de la lista, después dirigirse a la opción Add y dar clic en New Folder. Como se requieren tres carpetas, repetir estre proceso tres veces.

A screenshot of a computer

Description automatically generated with medium confidence

Figura . Creación de carpetas para MVC

1. Una vez creadas las carpetas dar clic sobre cada una y presionar la tecla F2 para renombrarlas con los nombres modelo, vista y controlador.

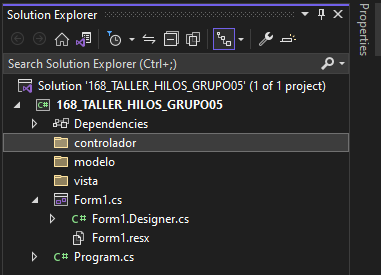


Figura . Arquitectura MVC para el programa

## PRÁCTICA VIDEO 168

Para esta parte de la práctica se creará una ventana que dibuje una pelota, esta se moverá en un panel, se activara al dar clic en un botón, y para cerrar o salir de la aplicación se creara un botón adicional.

### **CODIFICACIÓN** **MODELO**

1. Para añadir una nueva clase, dirigirse a la sección Solution Explorer, dar clic derecho sobre la carpeta modelo, luego Add y clic en la opción Class.

A screenshot of a computer

Description automatically generated with medium confidence

Figura . Creación de Clase en el modelo

1. En la siguiente ventana, definir el nombre de la clase como PelotaModelo y clic en Add, esta clase tendrá las propiedades y métodos para dibujar y mover la pelota.

A screenshot of a computer

Description automatically generated with medium confidence

Figura . Asignación del nombre PelotaModelo para la Clase

1. A continuación, se muestra la codificación de la clase, en donde se definen las propiedades para las coordenadas de la pelota, campos auxiliares para mover la pelota y dos objetos para dibujar y definir el hilo que moverá la pelota.

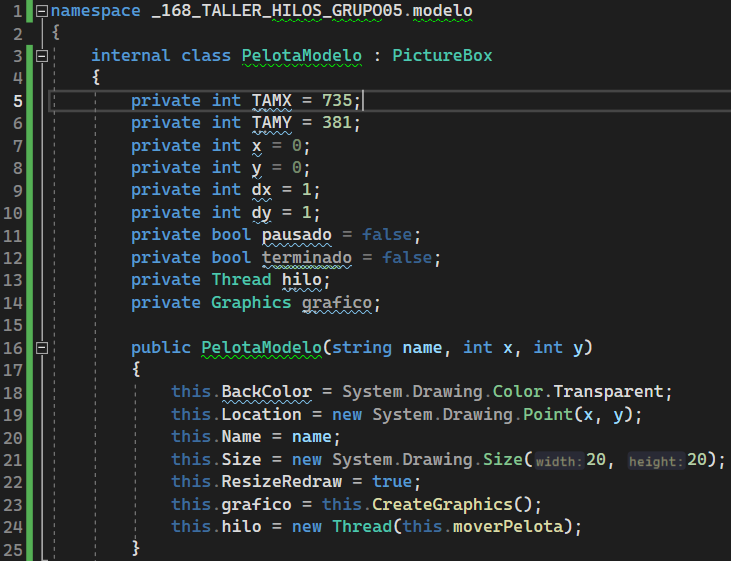


Figura . Codificación de la clase PelotaModelo

Tabla . Codificación del archivo PelotaModelo.cs práctica video 168

namespace \_168\_TALLER\_HILOS\_GRUPO05.modelo

{

internal class PelotaModelo : PictureBox

{

private int TAMX = 735;

private int TAMY = 381;

private int x = 0;

private int y = 0;

private int dx = 1;

private int dy = 1;

private bool pausado = false;

private bool terminado = false;

private Thread hilo;

private Graphics grafico;

public PelotaModelo(string name, int x, int y)

{

this.BackColor = System.Drawing.Color.Transparent;

this.Location = new System.Drawing.Point(x, y);

this.Name = name;

this.Size = new System.Drawing.Size(20, 20);

this.ResizeRedraw = true;

this.grafico = this.CreateGraphics();

this.hilo = new Thread(this.moverPelota);

}

public void dibujarPelota(Brush color)

{

Image bmp = new Bitmap(20, 20);

using (Graphics g = Graphics.FromImage(bmp))

{

g.FillEllipse(color, 0, 0, 20, 20);

}

this.Image = bmp;

}

public void ejecutar()

{

if (this.hilo.ThreadState == ThreadState.Unstarted)

{

this.hilo.IsBackground = true;

this.hilo.Start();

}

else

{

lock (this)

{

this.pausado = false;

Monitor.PulseAll(this);

}

}

}

public void finalizar()

{

lock (this)

{

this.terminado = true;

Monitor.PulseAll(this);

}

}

public void moverPelota()

{

while (true)

{

x = this.Location.X + dx;

y = this.Location.Y + dy;

this.Location = new System.Drawing.Point(x, y);

Thread.Sleep(1);

lock (this)

{

if (this.pausado) Monitor.Wait(this);

}

if (this.Location.X > this.TAMX - 10)

{

dx = -dx;

}

else if (this.Location.X < 10)

{

dx = -dx;

}

if (this.Location.Y > this.TAMY - 10)

{

dy = -dy;

}

else if (this.Location.Y < 10)

{

dy = -dy;

}

}

}

}

}

### **CODIFICACIÓN VISTA**

1. Para crear la ventana en donde se dibujará la pelota, dar clic derecho sobre la carpeta vista y dar clic sobre la opción Form (Windows Forms).

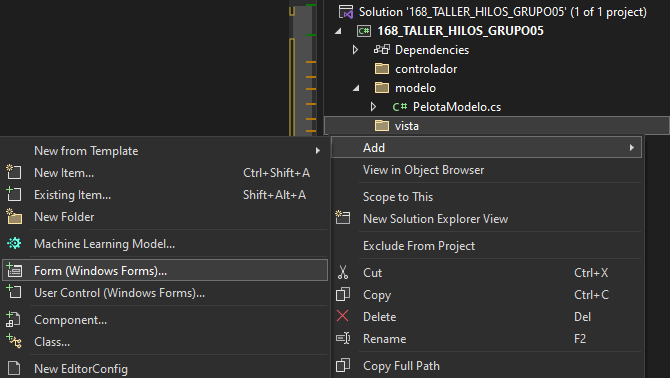


Figura . Creación de ventana en la carpeta vista

1. En la siguiente ventana ingresar el nombre de la clase como PelotaVista y clic en Add.

A screenshot of a computer

Description automatically generated with medium confidence

Figura . Asignación del nombre PelotaVista para la Clase

1. Se genera una nueva ventana con sus clases y archivos de configuración para el diseño y control de la vista.

A picture containing text, screenshot, monitor, computer

Description automatically generated

Figura . Ventana generada para PelotaVista

1. Para añadir elementos a la ventana, dirigirse al panel Toolbox y se muestra una lista extensa de elementos que se pueden añadir a la ventana, seleccionar la opción panel y darle clic.

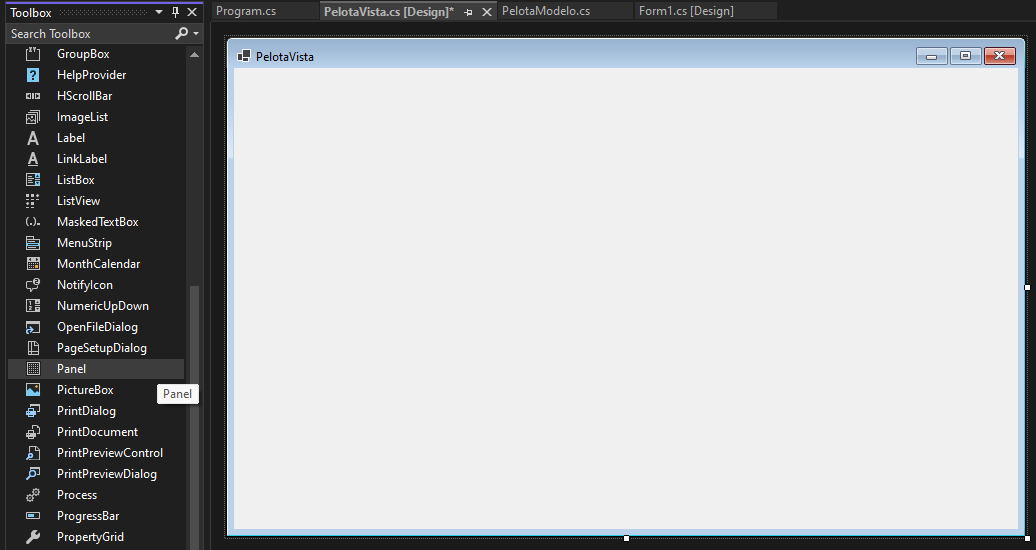


Figura . Creación del panel para la ventana PelotaVista

1. Para añadir el panel seleccionado a la ventana, dar clic en donde se ubicará el panel, en este caso en la esquina superior izquierda de la ventana.

A picture containing graphical user interface

Description automatically generated

Figura . Panel generado en la ventana PelotaVista

1. Para dimensionar el panel, dar clic sobre los botones que se muestran alrededor del panel hasta que ocupe todo el ancho de la ventana y casi todo el alto como se muestra en la siguiente figura.

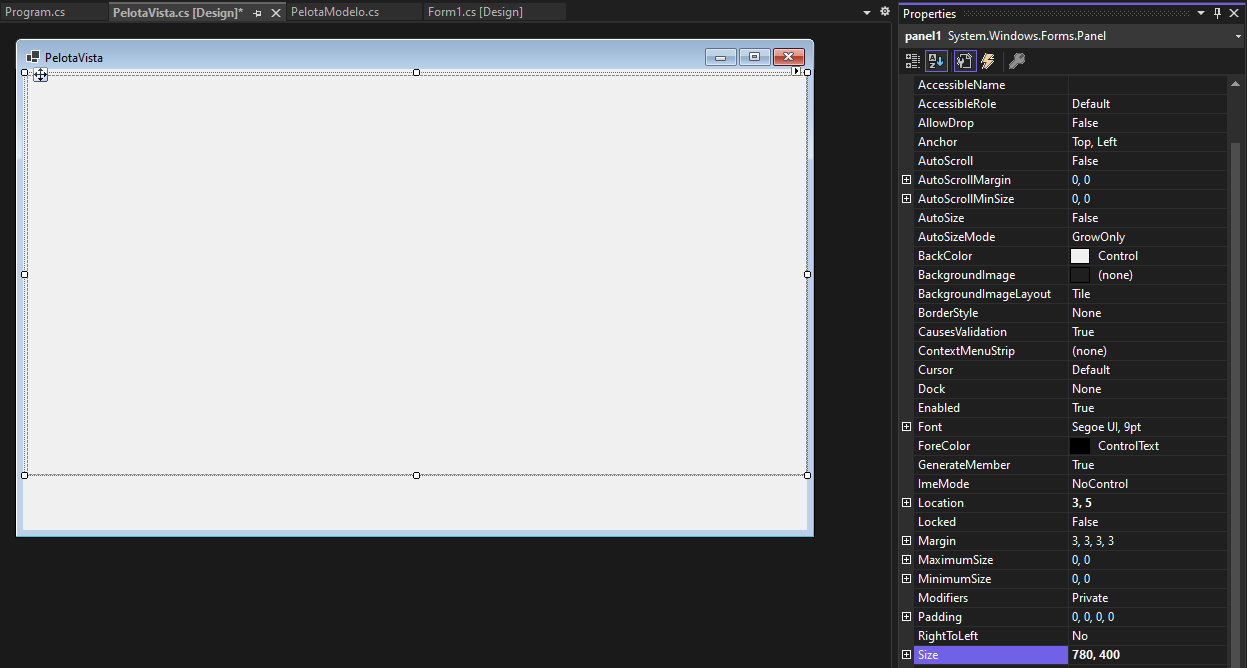


Figura . Redimensión del panel en la ventana

1. A continuación, se debe definir el nombre del panel tal y como se muestra en la tabla de Nombres y Abreviaturas, en la sección Properties buscar la opción (Name) e ingresar el nombre del panel como pnlPelotas.

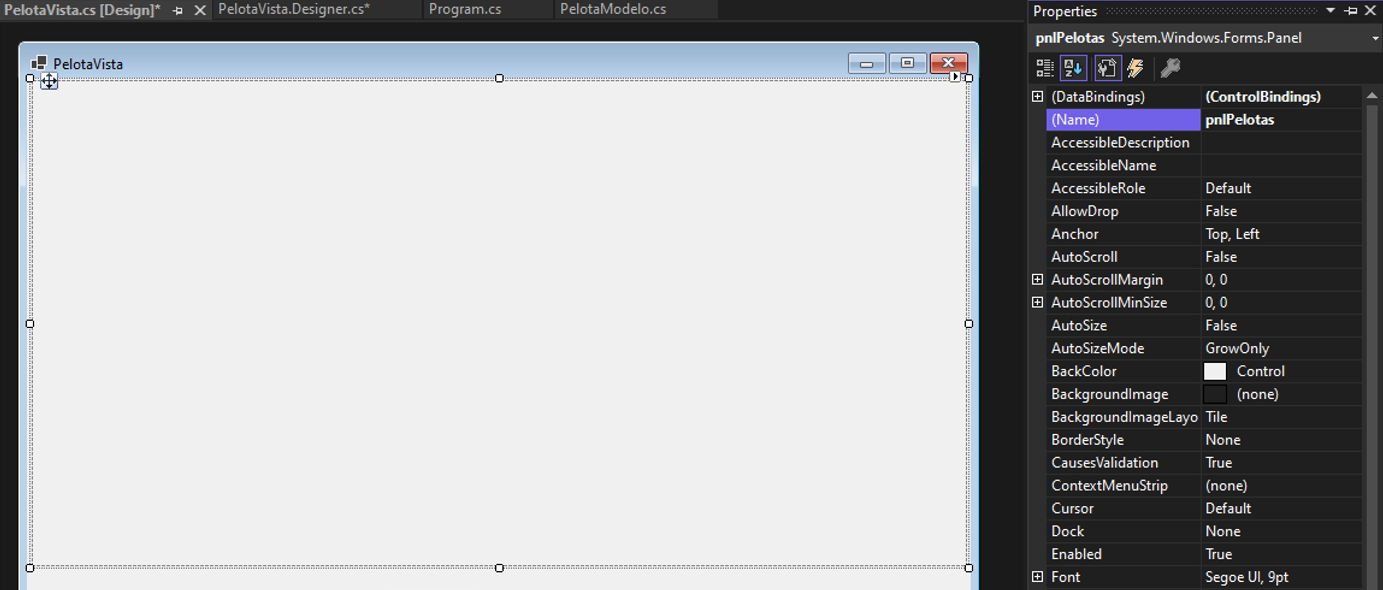


Figura . Nombre para el panel creado

1. De la misma forma como se añadió el panel, dirigirse a la sección Toobox y dar clic en la opción Button.

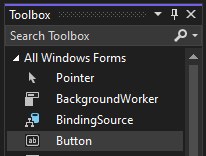


Figura . Elemento botón en el panel Toolbox

1. Dirigirse a la sección Properties y buscar la opción Text, en este campo ingresar el texto que se mostrará en el botón, en este caso Dale.

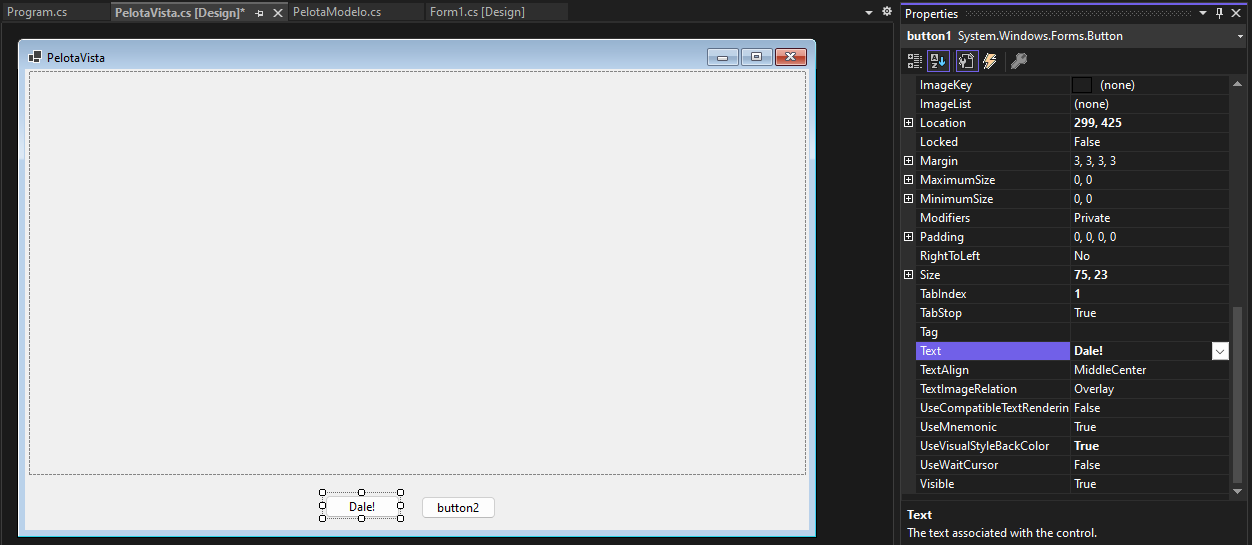


Figura . Texto para el botón Dale

1. Luego buscar la opción (Name) e ingresar el nombre del botón como btnDale.

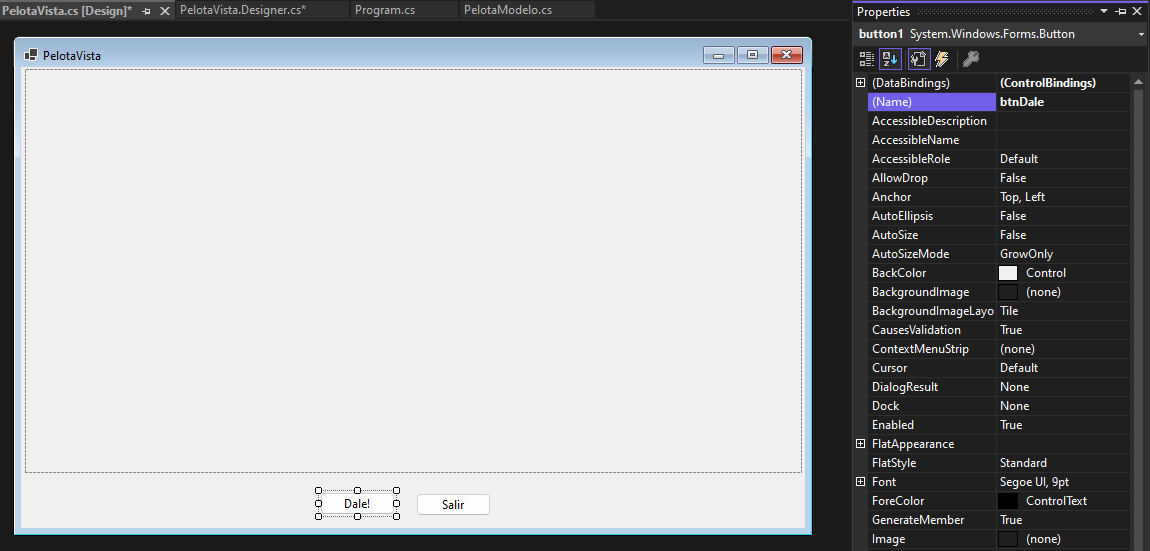


Figura . Nombre para el botón Dale

1. De la misma forma para el botón Salir, buscar la opción Text e ingresar el texto que se mostrara en el botón.

A picture containing text, screenshot, monitor

Description automatically generated

Figura . Texto para el botón Salir

1. Para el botón Salir, buscar el campo (Name) en ingresar btnSalir.

A picture containing text, screenshot, computer

Description automatically generated

Figura . Nombre para el botón Salir

1. Para acceder al panel y los botones que se agregaron a la ventana se deben definir sus métodos getters, para esto dirigirse a la sección Solution Explorer, dar clic derecho sobre el archivo PelotaVista.cs y clic en View Code.

A screenshot of a computer

Description automatically generated with medium confidence

Figura . Acceder a la clase PelotaVista

1. En el archivo de clase que se abre a continuación, codificar los métodos getter para cada elemento que se agregó a la ventana.

Text

Description automatically generated

Figura . Codificación de la clase PelotaVista

Tabla . Codificación del archivo PelotaVista.cs práctica video 168

namespace \_168\_TALLER\_HILOS\_GRUPO05.vista

{

public partial class PelotaVista : Form

{

public PelotaVista()

{

InitializeComponent();

}

public Panel getPnlPelotas()

{

return this.pnlPelotas;

}

public Button getBtnDale()

{

return this.btnDale;

}

public Button getBtnSalir()

{

return this.btnSalir;

}

}

}

1. Para que el panel en el que se dibujaran las pelotas sea más legible, dirigirse a la sección Properties, buscar la opción BackColor y seleccionar un color más oscuro.

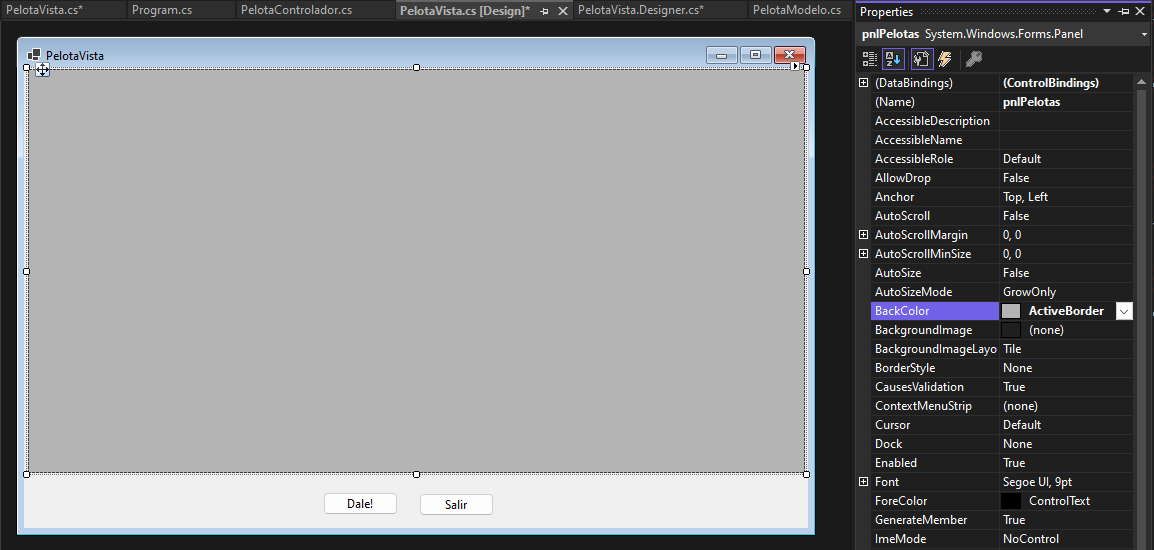


Figura . Color de fondo para el panel

1. Para que la ventana PelotaVista se muestra al ejecutar el proyecto, dirigirse a la clase Program.cs y cambiar el código por el que se muestra a continuación.

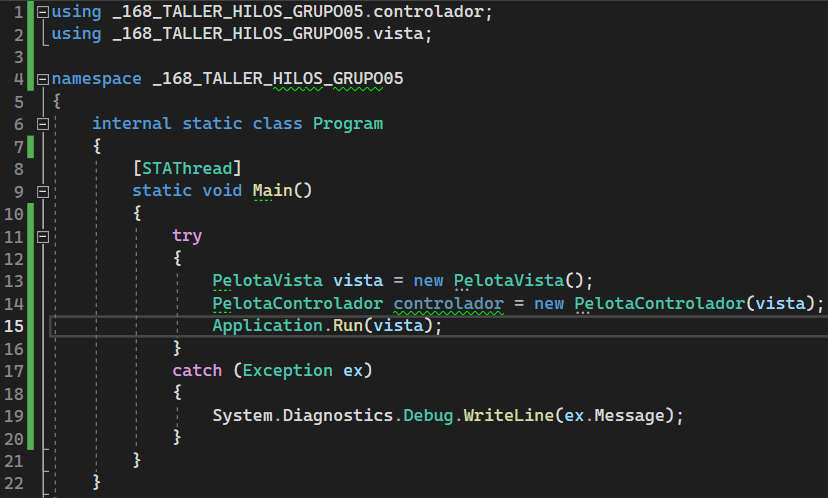


Figura . Codificación de la clase Program

using \_168\_TALLER\_HILOS\_GRUPO05.controlador;

using \_168\_TALLER\_HILOS\_GRUPO05.vista;

namespace \_168\_TALLER\_HILOS\_GRUPO05

{

internal static class Program

{

[STAThread]

static void Main()

{

try

{

PelotaVista vista = new PelotaVista();

PelotaControlador controlador = new PelotaControlador(vista);

Application.Run(vista);

}

catch (Exception ex)

{

System.Diagnostics.Debug.WriteLine(ex.Message);

}

}

}

}

Tabla . Codificación archivo Program.cs práctica video 168

1. Al crear el proyecto se generó de forma predeterminada una ventana llamada Form1, esta ventana es innecesaria, por lo tanto, dirigirse a la sección Solution Explorer, dar clic derecho sobre Form1.cs y clic en Delete para borrar la ventana.

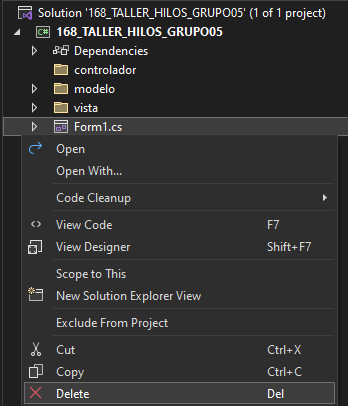


Figura . Borrado de la ventana inicial generada

### **CODIFICACIÓN CONTROLADOR**

1. Para la codificación del controlador, dar clic derecho sobre la carpeta controlador y clic en Class.

A screenshot of a computer

Description automatically generated with medium confidence

Figura . Creación de Clase en el controlador

1. En la siguiente ventana, definir el nombre de la clase como PelotaControlador y clic en Add.

A screenshot of a computer

Description automatically generated with medium confidence

Figura . Asignación del nombre PelotaControlador para la Clase

1. En esta clase se definen los procesos que se ejecutaran cuando el usuario haga clic en los botones Dale y Salir, por lo tanto, se relaciona y controla la clase PelotaModelo con PelotaVista.

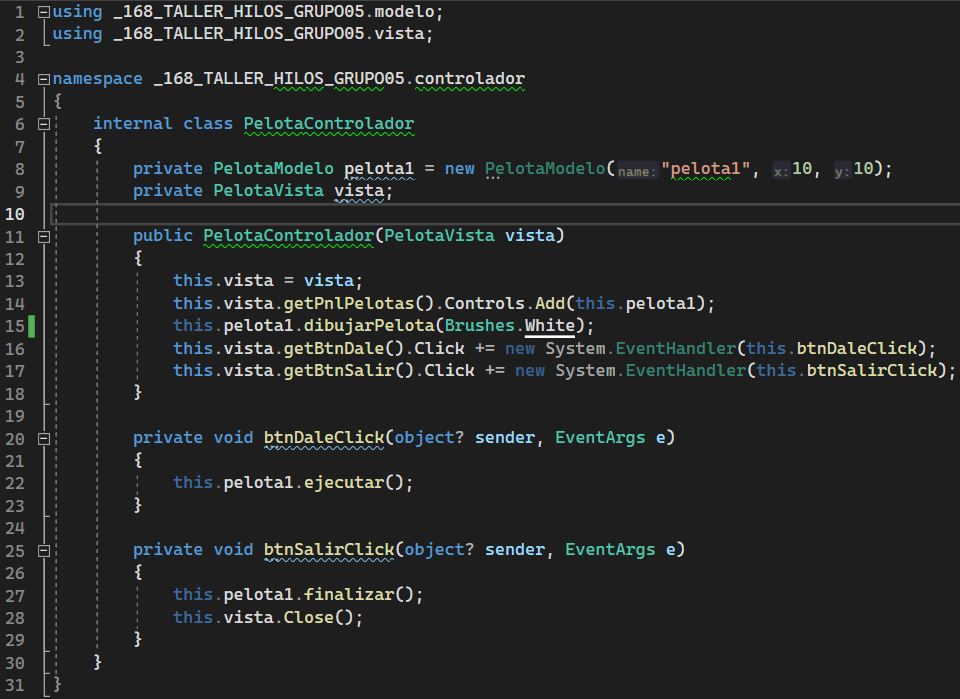


Figura . Codificación de la clase PelotaControlador

Tabla . Codificación archivo PelotaControlador.cs práctica video 168

using \_168\_TALLER\_HILOS\_GRUPO05.modelo;

using \_168\_TALLER\_HILOS\_GRUPO05.vista;

namespace \_168\_TALLER\_HILOS\_GRUPO05.controlador

{

internal class PelotaControlador

{

private PelotaModelo pelota1 = new PelotaModelo("pelota1", 10, 10);

private PelotaVista vista;

public PelotaControlador(PelotaVista vista)

{

this.vista = vista;

this.vista.getPnlPelotas().Controls.Add(this.pelota1);

this.pelota1.dibujarPelota(Brushes.White);

this.vista.getBtnDale().Click += new System.EventHandler(this.btnDaleClick);

this.vista.getBtnSalir().Click += new System.EventHandler(this.btnSalirClick);

}

private void btnDaleClick(object? sender, EventArgs e)

{

this.pelota1.ejecutar();

}

private void btnSalirClick(object? sender, EventArgs e)

{

this.pelota1.finalizar();

this.vista.Close();

}

}

}

## PRÁCTICA VIDEO 169

Para esta parte de la práctica, se agregará un nuevo botón, este será el encargado de detener el movimiento de la pelota, pausando la ejecución en el hilo.

### **CODIFICACIÓN MODELO**

1. Para poder pausar la ejecución del hilo y así detener el movimiento de la pelota, se debe agregar el método que se muestra a continuación en la clase PelotaModelo.

Graphical user interface

Description automatically generated with medium confidence

Figura . Codificación del método pausar

Tabla . Codificación del archivo PelotaModelo.cs práctica video 169

public void pausar()

{

lock (this)

{

this.pausado = true;

}

}

### **CODIFICACIÓN VISTA**

1. Para detener el movimiento de la pelota, se agregará un botón a la ventana.

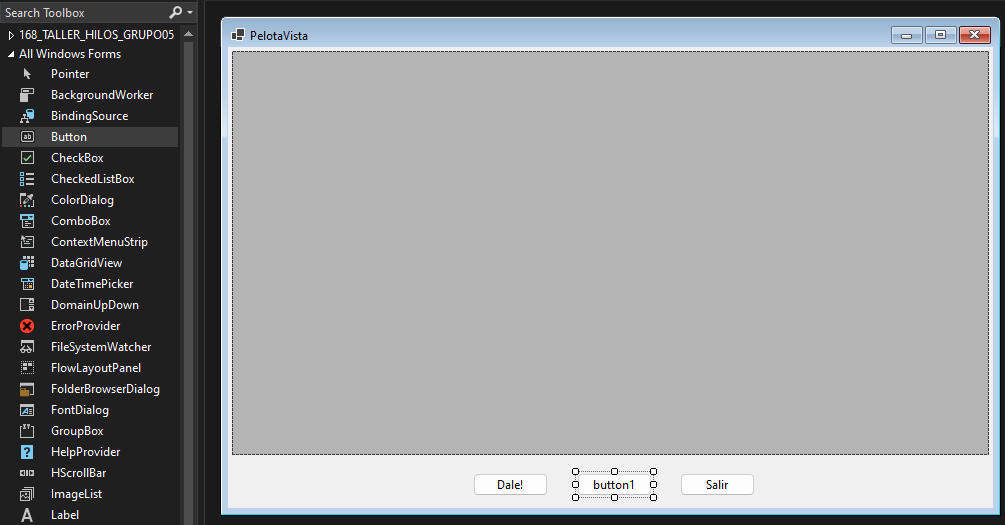


Figura . Agregar botón para detener la pelota

1. A continuación, se debe definir el nombre y texto del botón, para esto dar clic derecho sobre el botón nuevo y clic en Properties.

A screenshot of a computer

Description automatically generated with medium confidence

Figura . Propiedades del botón Pausar

1. En el campo (Name) definir el nombre del botón como btnPausar.

Graphical user interface, application

Description automatically generated

Figura . Nombre del botón Pausar

1. En el campo Text ingresar el texto que se mostrará en el botón como Pausar.

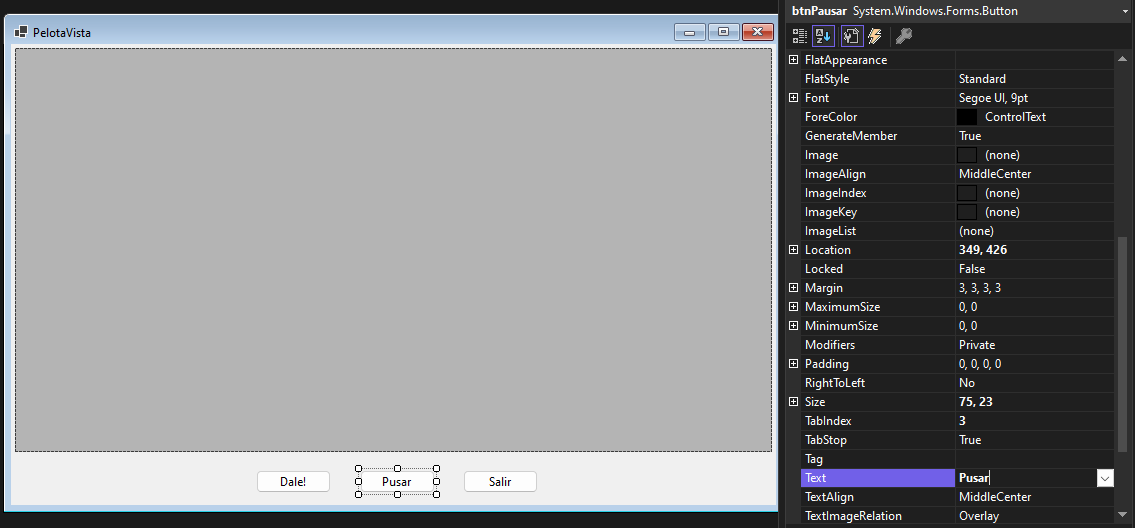


Figura . Texto para el botón Pausar

1. Para agregar los accesores del nuevo botón dirigirse a la sección Solution Explorer y dar clic derecho sobre PelotaVista.cs y clic en View Code.

A screenshot of a computer

Description automatically generated with medium confidence

Figura . Ingreso para el código de la Clase PelotaVista

1. En el archivo de clase que se abre a continuación, agregar el siguiente método getter para el botón Pausar.

Text

Description automatically generated with medium confidence

Figura . Método para obtener el botón Pausar

Tabla . Codificación del archivo PelotaVista.cs práctica video 169

public Button getBtnPausar()

{

return this.btnPausar;

}

### **CODIFICACIÓN CONTROLADOR**

1. En la clase PelotaControlador se debe agregar el método que se ejecutara al dar clic en el botón Pausar, el cual se encarga de detener el movimiento de la pelota.

Text

Description automatically generated

Figura . Codificación para la Clase PelotaControlador

using \_168\_TALLER\_HILOS\_GRUPO05.modelo;

using \_168\_TALLER\_HILOS\_GRUPO05.vista;

namespace \_168\_TALLER\_HILOS\_GRUPO05.controlador

{

internal class PelotaControlador

{

private PelotaModelo pelota1 = new PelotaModelo("pelota1", 10, 10);

private PelotaVista vista;

public PelotaControlador(PelotaVista vista)

{

this.vista = vista;

this.vista.getPnlPelotas().Controls.Add(this.pelota1);

this.pelota1.dibujarPelota(Brushes.White);

this.vista.getBtnDale().Click += new System.EventHandler(this.btnDaleClick);

this.vista.getBtnSalir().Click += new System.EventHandler(this.btnSalirClick);

this.vista.getBtnPausar().Click += new System.EventHandler(this.btnPausarClick);

}

private void btnDaleClick(object? sender, EventArgs e)

{

this.pelota1.ejecutar();

}

private void btnSalirClick(object? sender, EventArgs e)

{

this.pelota1.finalizar();

this.vista.Close();

}

private void btnPausarClick(object? sender, EventArgs e)

{

this.pelota1.pausar();

}

}

}

Tabla . Codificación del archivo PelotaControlador.cs práctica video 169

## PRÁCTICA VIDEO 170

Para esta parte de la práctica, se agregarán dos pelotas más, de esta forma se podrá apreciar de mejora manera el uso de los hilos y como operan de forma independiente por cada pelota.

### **CODIFICACIÓN VISTA**

1. De la misma forma como se agregaron los botones en las practicas anteriores, se deben agregar botones de iniciar y pausar para las dos nuevas pelotas, tal y como se muestra en la siguiente figura.

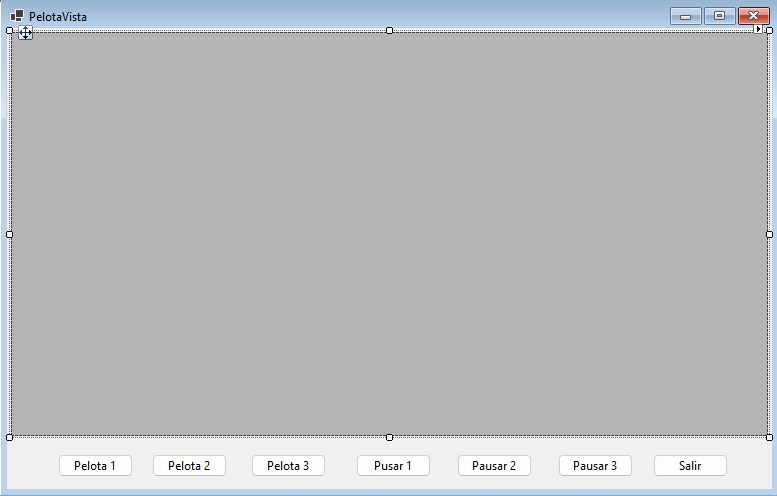


Figura . Agregación de los botones para las nuevas pelotas

1. En la clase PelotaVista se deben agregar los getters para los nuevos botones.

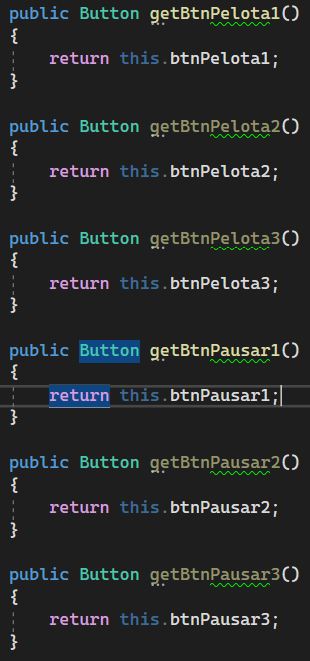


Figura . Codificación de la clase PelotaVista

Tabla . Codificación del archivo PelotaVista.cs práctica video 170

namespace \_168\_TALLER\_HILOS\_GRUPO05.vista{

public partial class PelotaVista : Form

{

public PelotaVista(){

InitializeComponent();

}

public Panel getPnlPelotas(){

return this.pnlPelotas;

}

public Button getBtnPelota1(){

return this.btnPelota1;

}

public Button getBtnPelota2(){

return this.btnPelota2;

}

public Button getBtnPelota3(){

return this.btnPelota3;

}

public Button getBtnPausar1(){

return this.btnPausar1;

}

public Button getBtnPausar2(){

return this.btnPausar2;

}

public Button getBtnPausar3(){

return this.btnPausar3;

}

public Button getBtnSalir(){

return this.btnSalir;

}

}

}

### **CODIFICACIÓN CONTROLADOR**

1. En la clase PelotaControlador se debe agregar dos instancias más de la clase PelotaModelo, además de agregar los eventos para cada nueva pelota. El código para las nuevas pelotas será el mismo, por lo tanto, se debe tomar de modelo la pelota que ya estaba creada con anterioridad.

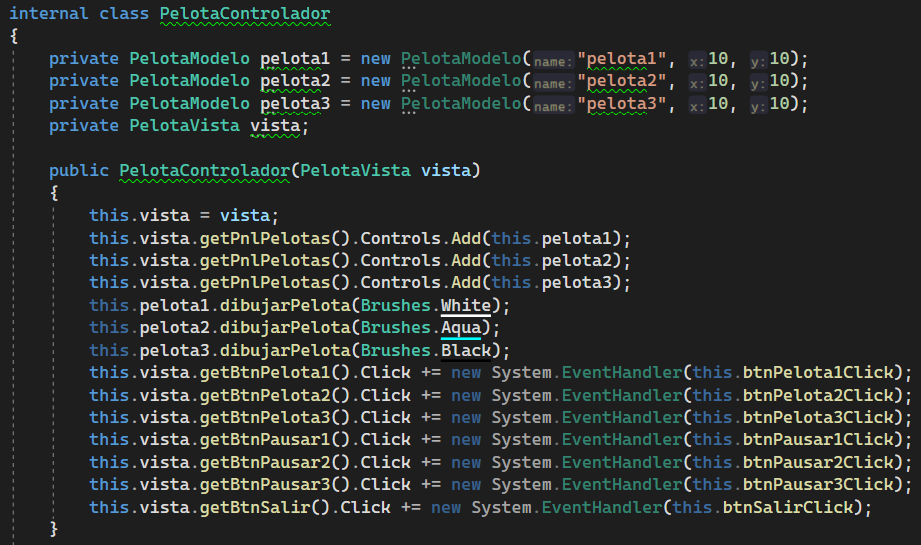


Figura . Codificación de la clase PelotaControlador

1. De la misma forma se deben definir los métodos que se ejecutaran al dar clic en los respectivos botones de la ventana.

Text

Description automatically generated

Figura . Métodos para eventos de cada pelota

using \_168\_TALLER\_HILOS\_GRUPO05.modelo;

using \_168\_TALLER\_HILOS\_GRUPO05.vista;

namespace \_168\_TALLER\_HILOS\_GRUPO05.controlador

{

internal class PelotaControlador

{

private PelotaModelo pelota1 = new PelotaModelo("pelota1", 10, 10);

private PelotaModelo pelota2 = new PelotaModelo("pelota2", 10, 10);

private PelotaModelo pelota3 = new PelotaModelo("pelota3", 10, 10);

private PelotaVista vista;

public PelotaControlador(PelotaVista vista)

{

this.vista = vista;

this.vista.getPnlPelotas().Controls.Add(this.pelota1);

this.vista.getPnlPelotas().Controls.Add(this.pelota2);

this.vista.getPnlPelotas().Controls.Add(this.pelota3);

this.pelota1.dibujarPelota(Brushes.White);

this.pelota2.dibujarPelota(Brushes.Aqua);

this.pelota3.dibujarPelota(Brushes.Black);

this.vista.getBtnPelota1().Click += new System.EventHandler(this.btnPelota1Click);

this.vista.getBtnPelota2().Click += new System.EventHandler(this.btnPelota2Click);

this.vista.getBtnPelota3().Click += new System.EventHandler(this.btnPelota3Click);

this.vista.getBtnPausar1().Click += new System.EventHandler(this.btnPausar1Click);

this.vista.getBtnPausar2().Click += new System.EventHandler(this.btnPausar2Click);

this.vista.getBtnPausar3().Click += new System.EventHandler(this.btnPausar3Click);

this.vista.getBtnSalir().Click += new System.EventHandler(this.btnSalirClick);

}

private void btnPelota1Click(object? sender, EventArgs e)

{

this.pelota1.ejecutar();

}

Tabla . Codificación del archivo PelotaControlador.cs práctica video 170

private void btnPelota2Click(object? sender, EventArgs e)

{

this.pelota2.ejecutar();

}

private void btnPelota3Click(object? sender, EventArgs e)

{

this.pelota3.ejecutar();

}

private void btnPausar1Click(object? sender, EventArgs e)

{

this.pelota1.pausar();

}

private void btnPausar2Click(object? sender, EventArgs e)

{

this.pelota2.pausar();

}

private void btnPausar3Click(object? sender, EventArgs e)

{

this.pelota3.pausar();

}

private void btnSalirClick(object? sender, EventArgs e)

{

this.pelota1.finalizar();

this.vista.Close();

}

}

}

## ESTRUCTURA DE LA APLICACIÓN

A screenshot of a computer

Description automatically generated with medium confidence

Figura . Estructura final de la aplicación web con MVC

## EJECUCIÓN DEL PROYECTO

Para ejecutar el proyecto dirigirse a la barra de herramientas superior y dar clic en Start Without Debugging o también pulsando las teclas Ctrl+F5.

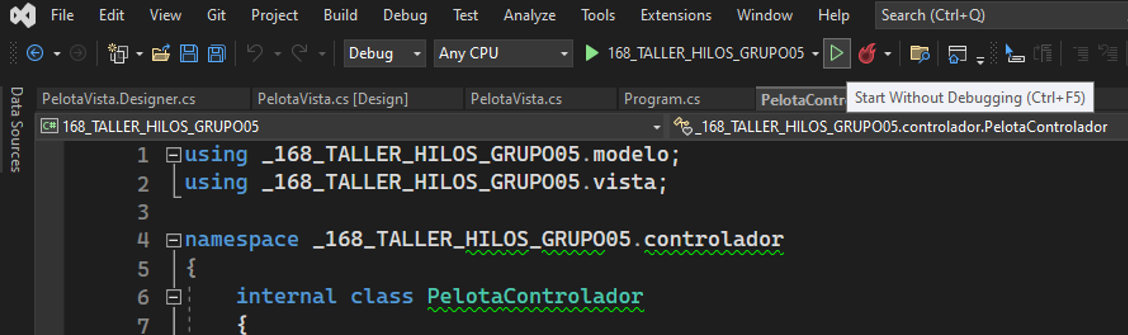


Figura . Ejecución del programa

### **EJECUCIÓN VIDEO 168**

Una vez se ejecute el proyecto, se muestra una ventana con la pelota en su posición inicial.

Graphical user interface, application, Word

Description automatically generated

Figura . Ventana inicial de la práctica video 168

Para mover la pelota se debe dar clic en el botón Dale, la pelota se moverá cuando el hilo comience a ejecutar el método para mover la pelota, y para salir de la aplicación se debe dar clic en el botón Salir.

Graphical user interface, application, Word

Description automatically generated

Figura . Pelotas en movimiento práctica 168

### **EJECUCIÓN VIDEO 169**

De la misma forma que en la sección anterior, una vez se ejecute el proyecto se muestra la pelota en la posición inicial.

Graphical user interface, application, Word

Description automatically generated

Figura . Ventana inicial de la práctica video 169

Dar clic en el botón Dale para que la pelota se mueva, para esta práctica se agregó un botón adicional, el cual pausara la ejecución el proceso del hilo de la pelota, y para hacer que la pelota vuelva a moverse se debe presionar en el botón Dala nuevamente, de esta forma el hilo que controla el proceso para que la pelota se mueva o detenga estará controlado por los botones Dale y Pausar.

Graphical user interface, application

Description automatically generated

Figura . Pelotas en movimiento práctica 169

### **EJECUCIÓN VIDEO 170**

En esta parte de la práctica, al ejecutar el proyecto se mostrarán tres pelotas, aunque no se pueden apreciar al inicio, pues se encuentran una sobre otra en la misma posición.

Graphical user interface

Description automatically generated with medium confidence

Figura . Ventana inicial de la práctica video 170

Al iniciar el movimiento de las pelotas con los botones Pelota1, Pelota2 y Pelota3 se puede apreciar como el aplicativo puede mover tres pelotas al mismo tiempo, aquí se puede evidenciar el uso de los hilos, se puede detener cualquier pelota sin afectar a las demás, y de la misma forma hacer que se muevan nuevamente, esto se da gracias a que cada pelota está siendo controlada por un hilo diferente, si las pelotas se movieran por un solo proceso, se detendrían o moverían al presionar cualquiera de los botones para detener o mover.

Chart

Description automatically generated with medium confidence

Figura . Pelotas en movimiento práctica 170

# CONCLUSIONES

* Los hilos son unidades de ejecución de un proceso que permiten la ejecución concurrente de varias tareas, mejorando la eficiencia y velocidad de una aplicación. La implementación de hilos en C# se puede realizar mediante la clase Thread y proporciona varias funciones para crear, controlar y sincronizar hilos. Los hilos pueden ser utilizados en la arquitectura MVC para implementar tareas concurrentes en un sistema.
* Los programas que no utilizan hilos solo tienen una secuencia de ejecución y ejecutan una tarea a la vez, lo que puede ser ineficiente para tareas complejas o que requieren la ejecución de varias tareas al mismo tiempo.
* La ejecución concurrente de varios hilos puede ser complicada y requerir una buena gestión para evitar conflictos y errores. Por lo tanto, es importante entender cómo funcionan los hilos y cómo se pueden utilizar de manera efectiva en una aplicación.

# RECOMENDACIONES

* Entender cómo funcionan los hilos y cuándo deben utilizarse. Los hilos son útiles para mejorar la eficiencia y el rendimiento de una aplicación al permitir la ejecución concurrente de varias tareas, pero también pueden ser complicados de gestionar y pueden requerir una buena planificación y diseño.
* Utiliza el depurador de C# para detectar y solucionar problemas relacionados con los hilos. El depurador te permite ver el estado de los hilos y el código que están ejecutando, lo que ayuda a identificar problemas y a encontrar soluciones.
* Al trabajar con la arquitectura MVC, se puede utilizar hilos para implementar tareas concurrentes en el sistema. Asegúrse de diseñar el sistema de manera que los hilos sean eficientes y se integren de manera adecuada con el resto de la aplicación.

# REFERENCIAS

Blancarte, O. (29 de Marzo de 2017). *Concurrencia vs Paralelismo*. Obtenido de oscarblancarteblog: https://www.oscarblancarteblog.com/2017/03/29/concurrencia-vs-paralelismo/

Davizhe. (7 de Octubre de 2014). *GESTIÓN DE HILOS*. Obtenido de davidizquierdoheras: https://davidizquierdoheras.wordpress.com/2014/10/07/gestion-de-hilos/

Elorduy, G. (2020). *Introducción a los hilos*. Obtenido de open source initiative: https://javaparajavatos.wordpress.com/2017/05/06/introduccion-a-los-hilos/

Microsoft. (22 de Septiembre de 2022). *Paseo por el lenguaje C#*. Obtenido de Learn Microsoft: https://learn.microsoft.com/es-es/dotnet/csharp/tour-of-csharp/

Milagros, B. (20 de Marzo de 2022). *¿Qué es Visual Studio? ¡El desarrollo de software nunca fue más fácil!* Obtenido de crehana: https://www.crehana.com/blog/transformacion-digital/que-es-visual-studio/

Nakayama, A. (2009). Guía práctica de estudio 12: Hilos. México.

Pellicer, P. (21 de Octubre de 2021). *¿Qué es el .NET? ¿Para qué sirve?* Obtenido de emagister: https://www.emagister.com/blog/que-es-el-net-para-que-sirve/

Rossainz, M. L. (2015). *Programación Concurrente y Paralela.* Puebla.