TECNOLÓGICO DE ESTUDIOS SUPERIORES DE COACALCO

Desarrollo de una aplicación para el registro y gestión automatizada de recorridos cronometrados en competencias de Robots sigue líneas mediante una interfaz Visual Studio - Arduino

PROYECTO DE INVESTIGACIÓN

PARA OBTENER EL TITULO DE

INGENIERO EN MECATRÓNICA

CON ESPECIALIDAD EN:

AUTOMATIZACIÓN Y CONTROL DE PROCESOS

P R E S E N T A

MADRIGAL RUEDA IAN OMAR
MATRICULA 201821238

DIRECTOR:

M. EN C. MARIO ALBERTO HERNÁNDEZ SORIANO

Coacalco de Berriozábal, julio de 2024

Dedicatorias

A mis padres, por su amor incondicional, su apoyo constante y por inculcarme la importancia de la educación. A mis amigos, por estar siempre a mi lado y darme ánimos cuando más los necesitaba. Y a mis profesores, por guiarme y compartir su conocimiento, haciendo posible este logro académico.

Agradecimientos

Quisiera expresar mi más profundo agradecimiento a todas las personas que han contribuido a la realización de este proyecto. En primer lugar, agradezco a mi asesor, el M. en C. Mario Alberto Hernández Soriano, por su guía, paciencia y apoyo incondicional a lo largo de este proceso. Sus valiosos comentarios y sugerencias fueron fundamentales para el desarrollo de este trabajo.

También me gustaría agradecer a mis profesores y colegas del departamento de Ingeniería Mecatrónica por compartir sus conocimientos y por el ambiente colaborativo que fomentaron. Un agradecimiento especial a mis amigos y compañeros de estudio, quienes me brindaron su amistad, comprensión y ayuda en los momentos más difíciles.

Finalmente, quiero agradecer a mi familia por su amor y apoyo incondicional. A mis padres, por creer siempre en mí y por su constante ánimo, y a mis hermanos, por su constante inspiración y motivación. Este logro no hubiera sido posible sin ustedes.

Resumen

El proyecto que se presenta a continuación tuvo como objetivo principal automatizar una competencia de robots sigue-líneas mediante una interfaz creada en Visual Studio C# con el apoyo de Arduino y una base de datos SQL. El proyecto busco facilitar la organización y ejecución de la competencia, permitiendo la detección precisa del momento en que cada robot ejecuta una vuelta en su respectiva pista de "carreras".

El sistema automatizado de competencia de robots sigue líneas fue diseñado pensando en su facilidad de uso y en la mejora de la precisión y eficiencia del evento. Al integrar la interfaz desarrollada en Visual Studio C# con los microcontroladores Arduino, el proyecto proporciono una solución completa y robusta para la realización de competencias de robots sigue líneas.

Este proyecto en cuestión represento un paso significativo hacia la automatización de eventos similares y demuestra la capacidad para integrar distintas tecnologías en un proyecto práctico y funcional. Además, tiene una plataforma para futuras mejoras y desarrollos en el ámbito de la robótica y la automatización de competencias.

ÍNDICE

Índice de ilustraciones

Capítulo I.
Generalidades

1.1 Introducción

El presente proyecto se centra en el desarrollo de un sistema automatizado para la competencia de robots seguidores de líneas mediante una interfaz de programación en Visual Studio C# con integración de Arduino y una base de datos SQL. El objetivo principal es permitir que los robots sigan una pista de "carreras" definida mediante una línea trazada, y se utilice un sensor de evitación de obstáculos por infrarrojos E18-D80NK para detectar el momento exacto en que el robot ejecuta una vuelta en su respectiva pista.

La importancia de este proyecto radica en el avance de la robótica y la automatización, lo cual presenta un gran potencial en diversos campos, como la industria, la logística y la exploración espacial, entre otros. Asimismo, esta investigación se considera relevante desde el punto de vista académico, ya que implica la aplicación práctica de conocimientos en áreas como programación, electrónica y bases de datos.

El trabajo se plantea con un enfoque práctico y teórico, en el que se utilizará el lenguaje de programación Visual Studio C# para desarrollar la interfaz que permitirá la comunicación entre la clasificación y el usuario. Se empleará el microcontrolador Arduino para la detección del robot ayudando del sensor de evitación de obstáculos E18-D80NK para asegurar la detección precisa del momento en que el robot completa una vuelta en la pista.

La metodología utilizada en este proyecto abarcará tanto la investigación teórica sobre los fundamentos de la programación, la electrónica y el manejo de bases de datos, como la implementación práctica del sistema automatizado. Se realizarán pruebas y análisis exhaustivos para garantizar el correcto funcionamiento y la precisión del circuito seguidor de líneas en la competencia.

Entre las limitaciones del trabajo se encuentran las posibles restricciones de tiempo y recursos para realizar investigaciones más extensas, así como la complejidad que pueda surgir al integrar distintas tecnologías en un solo sistema.

En resumen, este trabajo representa un enfoque multidisciplinario que combina la programación, la electrónica y el manejo de bases de datos para desarrollar un sistema automatizado que permita a los robots competir en una pista de carreras siguiendo líneas de manera autónoma y detectando con precisión el momento en que completan sus vueltas.

1.2 Justificación

Desde una perspectiva académica, el estudiante se encuentra inmerso en el campo de la mecatrónica, una disciplina interdisciplinaria que combina la mecánica, la electrónica y la informática. Esta aplicación representa una oportunidad excepcional para aplicar y consolidar los conocimientos teóricos adquiridos a lo largo de su formación universitaria. El manejo del lenguaje de programación C# en Visual Studio y la interfaz con el microcontrolador Arduino, sumado a la integración de una base de datos SQL, proporcionan una experiencia de aprendizaje integral y valiosa para el estudiante.

Además de los aspectos personales y académicos, la dimensión social de este proyecto también es relevante. La robótica y la automatización son áreas de rápido crecimiento en el mundo actual. Mediante el desarrollo de esta aplicación, el estudiante contribuirá a la promoción y difusión de la mecatrónica como una disciplina innovadora y relevante en la solución de problemas. La posibilidad de organizar competencias de robots sigue líneas a través de un sistema automatizado podría inspirar a otras personas a adentrarse en el campo de la robótica y la ingeniería, fomentando así el desarrollo tecnológico y la creatividad en la comunidad.

Finalmente, la decisión de trabajar en el desarrollo de una aplicación para la gestión automatizada de competencias de Robots sigue líneas mediante una interfaz Visual Studio - Arduino responde al profundo interés personal por la robótica, la necesidad de aplicar los conocimientos académicos adquiridos y el deseo de contribuir positivamente al ámbito de la mecatrónica y la automatización. Este proyecto no solo representa una oportunidad para lograr un trámite universitario, sino también para dejar una huella en el campo tecnológico y social, inspirando a otros a explorar y experimentar con la robótica y la ingeniería en busca de soluciones innovadoras.

1.3 Planteamiento del problema

En el ámbito de la mecatrónica, se ha planteado el desarrollo de una aplicación que automatice la gestión de competencias de robots sigue líneas utilizando una interfaz integrada entre Visual Studio C#, Arduino y una base de datos SQL. El proyecto tiene como objetivo optimizar la realización de competencias de robots sigue líneas, mediante el uso de un sensor de evitación de obstáculos por infrarrojos E18-D80NK. Esta tecnología permitirá detectar de manera precisa el momento en que el robot ejecuta una vuelta en su respectiva pista de "carreras".

El presente proyecto se enmarca en la mecatrónica, una disciplina que combina la ingeniería mecánica, electrónica y sistemas de control para el diseño y desarrollo de sistemas automatizados. En este contexto, el proyecto busca aplicar los conocimientos y habilidades en lenguaje de programación Visual Studio C#, bases de datos SQL, y microcontroladores Arduino, para mejorar la eficiencia y precisión de las competencias de robots sigue líneas.

La propuesta consiste en desarrollar una aplicación que integre la información proveniente del sensor de evitación de obstáculos por infrarrojos E18-D80NK con un microcontrolador Arduino. La información del sensor será procesada y enviada a visual Studio C# para su almacenamiento y posterior análisis, que permita al usuario gestionar la competencia, visualizar en tiempo real el tiempo de conclusión de cada vuelta, y generar informes de resultados.

La necesidad de implantar esta propuesta radica en la importancia de contar con un sistema automatizado y preciso para la gestión de competencias de robots sigue líneas. Actualmente, estas competencias se realizan manualmente, lo que puede llevar a imprecisiones en la medición de los tiempos de vuelta y en la detección de errores. La implementación de este proyecto brindará a los organizadores de competencias y a los participantes una herramienta confiable para realizar mediciones precisas, facilitando la toma de decisiones y la mejora continua de dicha competencia.

En caso de no aplicarse la propuesta, las competencias de robots siguen líneas seguirían siendo gestionadas de forma manual, lo que podría conllevar diversas consecuencias negativas. Entre ellas, se encuentran la falta de precisión en la detección de tiempos de vuelta y posibles errores en el conteo de vueltas. Estos factores podrían afectar la imparcialidad de los resultados y la credibilidad de las competencias.

Además, la gestión manual de las competencias implicaría un mayor tiempo y esfuerzo para los organizadores y participantes. La ausencia de una interfaz automatizada y una base de datos para almacenar los datos limitaría el análisis y la retroalimentación para mejorar el rendimiento de los robots en futuras competencias.

En conclusión, el desarrollo de una aplicación que automatice la gestión de competencias de robots sigue líneas mediante una interfaz Visual Studio - Arduino, basada en el uso del sensor de evitación de obstáculos por infrarrojos E18-D80NK y una base de datos SQL, se presenta como una solución efectiva para mejorar la precisión y eficiencia en la realización de estas competencias. Su implementación contribuirá al avance de la mecatrónica y al desarrollo de soluciones innovadoras en el campo de la automatización y control de sistemas.

1.4 Antecedentes

Las competencias de robots del tipo sigue-líneas hoy en día tienen relevancia en eventos y congresos de robótica a nivel universitario entre estudiantes de carreras como Ing. Electromecánica, Ing. Mecatrónica, Ing. Sistemas, Ing. Electrónica, entre otras. A continuación, se muestra algunas convocatorias en las cuales se han realizado este tipo de competencias.

1. Torneo de sigue líneas, XVI Congreso mexicano de Robótica, llevado a cabo del 6 al 8 de noviembre del 2014, Mazatlán, México. [1]
2. 5ta feria mecatrónica de robot seguidor de línea, instituto tecnológico de Culiacán y el club de robótica, llevado a cabo el 26 Y 27 de noviembre del 2014, Culiacán, México. [2]

3. Concurso de robot seguidor de línea, Universidad Autónoma de la Ciudad de México, llevado a cabo 18 y 19 de mayo del 2023, Ciudad de México. [3]
4. Concurso Robot Seguidor de Líneas, Facultad de Ingeniería y Ciencias de la Universidad Autónoma de Tamaulipas, llevado a cabo el 25 de octubre de 2022, Tamaulipas, México. [4]
5. Concurso de robots seguidor de líneas, Instituto Tecnológico Superior de Calkiní, llevado a cabo el 07 de octubre del 2014, Campeche, México. [5]
6. 1er concurso de Robots seguidores de línea, Facultad de Ingeniería Eléctrica y Electrónica, llevado a cabo el 16 y 17 de noviembre del 2018, Veracruz, México. [6]
7. Concurso semana de ingeniería robot seguidor de línea, Facultad de Ingeniería, llevado a cabo los días 30 de septiembre de 2019, Colombia. [7]
8. 11° encuentro estatal de robótica y prototipos de desarrollo tecnológico, llevado a cabo el 12 de septiembre de 2021, Morelia, Michoacán. [8]

Haciendo una revisión en la manera de como los jueces evalúan al ganador, en todos los casos es un juez quien con un cronometro manual determina el tiempo en el recorrido de cada competidor, sin utilizar sensores que activen y desactiven de manera automática el cronometro.

1.5 Objetivos

1.5.1 Objetivo general

El objetivo general del proyecto en el área de mecatrónica es desarrollar una aplicación que permita la gestión automatizada de competencias de robots seguidores de líneas. Esta aplicación se realizará mediante una interfaz que integrará Visual Studio, lenguaje de programación C#, bases de datos SQL y un microcontrolador Arduino, con el fin de facilitar la competencia y registrar de manera precisa el momento exacto en que cada robot ejecuta una vuelta en su respectiva pista de "carreras".

1.5.2 Objetivos específicos

1. Diseñar e implementar una interfaz de usuario intuitiva en Visual Studio C# que permita a los participantes configurar y monitorear los parámetros de sus robots seguidores de líneas, como la velocidad y sensibilidad de los sensores infrarrojos E18-D80NK.
2. Desarrollar el código necesario en el lenguaje de programación C# para establecer la comunicación bidireccional entre la aplicación y el microcontrolador Arduino, facilitando la recepción de datos del sensor de evitación de obstáculos y el envío de instrucciones a Visual Studio C#.

3. Implementar un sistema de detección y registro de tiempos precisos, utilizando la base de datos SQL, que permita identificar el momento exacto en que cada robot completa una vuelta en la pista de competencia.
4. Validar la aplicación y el sistema automatizado en un entorno real de competencias de robots seguidores de líneas, obteniendo datos reales y retroalimentación de los participantes para realizar ajustes y mejoras, si es necesario.
5. Documentar detalladamente el proceso de desarrollo, desde la configuración del entorno de programación hasta la implementación del sistema automatizado, con el objetivo de que otros interesados puedan replicar y ampliar el proyecto en futuras investigaciones y competencias.

Con la consecución de estos objetivos específicos, se pretende lograr el objetivo general del proyecto, desarrollando una aplicación efectiva y funcional que contribuya a la gestión automatizada de competencias de robots seguidores de líneas, proporcionando una herramienta útil y eficiente en el ámbito de la mecatrónica.

1.6 Hipótesis

Se pretende demostrar que en el proceso de cronometrado de cada concursante abra más fiabilidad en los tiempos registrados que cuando un jurado o un juez cronometra manualmente.

1.7 Alcances y limitaciones

El proyecto se centra en la creación de una aplicación que automatice la gestión de competencias de robots seguidores de líneas, integrando Visual Studio C#, Arduino y una base de datos SQL, permitiendo que la aplicación permitirá a los usuarios configurar y monitorear parámetros de los robots, como velocidad y sensibilidad de los sensores infrarrojos E18-D80NK, mediante una interfaz intuitiva en Visual Studio C#
Se desarrollará un sistema de comunicación entre la aplicación y el microcontrolador Arduino, permitiendo la recepción de datos del sensor de evitación de obstáculos y el envío de instrucciones, con la implementación de un sistema que registre con precisión los tiempos de vuelta de cada robot, almacenando estos datos en una base de datos SQL.

El proyecto puede estar limitado por el tiempo disponible y los recursos financieros y técnicos, lo que podría afectar la profundidad y amplitud de la investigación y desarrollo de igual manera la integración de diversas tecnologías como Visual Studio C#, Arduino y bases de datos SQL puede presentar desafíos técnicos que requieran soluciones complejas y tiempo adicional.
Dado que las pruebas se realizarán en un entorno específico de competencias de robots, los resultados pueden no ser generalizables a otras condiciones o tipos de competencias sin ajustes adicionales; la necesidad de mantener y actualizar tanto el software como el hardware para asegurar un funcionamiento continuo y eficiente puede requerir recursos adicionales a largo plazo.

Capítulo II.
Marco Teórico

2.1 Lenguaje de programación

2.1.1 Definición

En [9] mencionan que una computadora ocupa un sistema de codificación llamado lenguaje maquina el cual consta de procesar una serie de números y patrones de 0 y 1, lo cual para un programador hace difícil la tarea de entender, corregir o memorizar dichos patrones, es necesario trabajar sobre un lenguaje que pueda ser traducido por la maquina y con instrucciones básicas por parte del programador, las cuales suelen ser:

- Instrucciones de entrada y salida:
- Instrucciones de calculo
- Instrucciones de control

Para poder pasar de un lenguaje dado por el programador aun lenguaje maquina es necesario usar programas llamados “Traductores”, dichos traductores son divididos en compiladores e intérpretes.

2.1.2 Interpretes

El intérprete es un traductor que toma un programa fuente para traducirlo y ejecutarlo, consiste en traducir la primera sentencia de programa a lenguaje máquina, se detiene la traducción, se ejecuta la sentencia, se traduce la siguiente sentencia, se detiene la traducción, se ejecuta la sentencia y así sucesivamente hasta terminar el programa. Como se puede observar en [10] el proceso no finaliza hasta que se ha interpretado todo el código, solo se llega a interrumpir si se produce un fallo durante el procesamiento.

2.1.3 Compiladores

De igual manera [11] describe un compilador como un programa informático que traduce todo el código fuente a código máquina antes de ejecutarlo, entendiendo lo anterior, explica como la traducción del programa se realiza en una sola operación denominada compilación; en la compilación es verificado el código para limpiarlo de errores y así ser ejecutado de manera directa cuantas veces sean necesarias.

2.1.4 Aplicaciones

Existe una gran variedad de lenguajes de programación usados actualmente por el área de la informática, a continuación, se aborda algunos de los más usados:

- **JavaScript**

  De acuerdo con [12] trata de un lenguaje de programación que permite la ejecución de funciones complejas en páginas web. Es la tercera capa del desarrollo teniendo como bases las tecnologías de HTML y CSS.

- **HTML**

  Es el lenguaje con el que se definen todos o en su mayoría de los elementos de una página web, así como imágenes, listas, videos, etc. Fue creada con el objetivo de tener la función de divulgación de información sin pensar en lo que más tarde terminaría siendo.

- **Python**

  Es un lenguaje de programación utilizado por varios entornos como las aplicaciones web, el desarrollo de software, la ciencia de datos y el machine learning (ML) como lo plantea [13], siendo un software gratuito, se integra bien a la mayoría de los sistemas aumentando la velocidad de desarrollo.

- **SQL**

  Structured Query Language (SQL) es un lenguaje de consulta, siendo un tipo de lenguaje de programación permite al desarrollador o usuario consultar, manipular o descargar datos de una base de datos. Su uso radica en las empresas en su mayoría de versiones teniendo varias plataformas para su desarrollo.

- **C#**

  Cuando se habla de C# es sobreentendido que tiene como base el lenguaje de programación C teniendo como característica principal la eficiencia del código, usado para sistemas operativos, aplicaciones e incluso juegos. C# es una evolución de C y C++ hecha por Microsoft terminando con una IDE de varios lenguajes con orientación a objetos con su plataforma .NET

2.1.5 Entorno de desarrollo integrado

La manera más fácil de trabajar un programa es con un IDE (Entorno de desarrollo integrado) el cual es un programa que ayuda al desarrollo de códigos en diferentes lenguajes y plataformas, así como, Visual Studio, Eclipse, IntelliJ, entre otros.

En la programación existen varias maneras de resolver un problema o un objetivo, a estas maneras se les denomina paradigmas de programación, de manera que los paradigmas de programación clásico son: procedimental, funcional, declarativo y orientada a objetos.

2.1.6 Programación Orientada a Objetos (POO)

Hablando de la POO [14] menciona que, en este tipo de lenguajes, se construyen objetos complejos de datos y luego designa un conjunto limitado de funciones para que operen. Este tipo de programación se emplea para estructurar un programa de software en piezas simples y reutilizables de planos de código (clases) para crear instancias individuales de objetos.

2.2 Visual Studio C#

2.2.1 Características de Visual Studio

Visual Studio es una herramienta de desarrollo eficaz que permite completar todo el ciclo de desarrollo en un solo entorno de desarrollo integrado (IDE), gracias a su variedad de características y lenguajes, puede ir de escribir un programa básico de "Hola mundo" hasta el desarrollo de aplicaciones. De igual manera su compatibilidad con varios lenguajes como C++, C#, Java, Typescript, entre otros lo hacen de las IDE's más versátiles del mercado afirmado por su sitio [15], la sintaxis básica de Visual Studio en C# es similar a la de cualquier otro entorno de desarrollo que admita este lenguaje.

2.2.2 Sintaxis

En C#, las variables se declaran especificando su tipo y nombre. Aquí hay ejemplos de cómo declarar variables en C#: int edad; double salario; string nombre; bool es Estudiante, etc., los operadores matemáticos en C# son + Suma, - Resta, * Multiplicación, / División, % Módulo (resto de la división), igualmente algunos operadores de comparación == Igual a, != Diferente de, > Mayor que, < Menor que, >= Mayor o igual que, <= Menor o igual que. El condicional `if` se utiliza para ejecutar un bloque de código si una condición es verdadera. Puedes usar `else` para ejecutar un bloque de código alternativo cuando la condición es falsa. Puedes encadenar varios condicionales `if - else` para evaluar múltiples condiciones:

```
intedad = 18 ;
 si (edad >= 18 )
{
    Console.WriteLine( "Eres mayor de edad." );
}
demás
{
    Console.WriteLine( "Eres menor de edad." );
}
```

2.2.3 Programación visual (Windows Forms)

Con base en [16] la programación visual (Windows Forms) es una tecnología de Microsoft que permite crear aplicaciones de escritorio con interfaces gráficas de usuario. Los puntos principales en esta IDE serian:

1. Formularios: Los formularios son ventanas que constituyen la interfaz de usuario de una aplicación. Se pueden diseñar y personalizar usando el diseñador gráfico de Visual Studio.

2. Controles: Windows Forms ofrece una amplia variedad de controles como botones, etiquetas, cajas de texto, listas, entre otros, que pueden ser agregados a los formularios.

3. Eventos: Se pueden asociar eventos a los controles, como hacer clic en un botón, para ejecutar acciones específicas en la aplicación.

4. Manejo de eventos: Los eventos pueden ser manejados a través de métodos específicos, lo que permite una interacción dinámica con el usuario.

2.3 Bases de datos SQL

2.3.1 Definición

El SQL es un lenguaje de programación empleado por casi todas las bases de datos relacionales para consultar, manipular y definir datos, así como para proporcionar control de acceso.

A comparación de las hojas de cálculo como explica [17] están diseñadas principalmente para un solo usuario y son ideales para tareas sencillas y para un pequeño número de usuarios. En contraste, las bases de datos están preparadas para manejar grandes volúmenes de información estructurada y permitir el acceso simultáneo de múltiples usuarios mediante tecnologías complejas.

Existen diversas bases de datos tales como relacionales: Organizan la información en tablas con filas y columnas, facilitando el acceso eficiente a datos estructurados, orientadas a objetos: Utilizan objetos para representar la información, siguiendo el paradigma de programación orientada a objetos, distribuidas: Almacenan datos en múltiples ubicaciones físicas o en varios equipos, facilitando el acceso desde distintas redes, almacenes de datos: Diseñados para consultas y análisis rápidos, actúan como repositorios centrales de información, NoSQL: Permiten gestionar datos no estructurados o semiestructurados, ideales para aplicaciones web complejas, orientadas a grafos: Gestionan datos mediante la representación de entidades y sus relaciones, finalmente esta OLTP: Optimizadas para transacciones rápidas y concurrentes por numerosos usuarios.

Algunos ejemplos incluyen MySQL, Microsoft Access, Microsoft SQL Server, FileMaker Pro, Oracle Database y dBASE, donde MySQL es un sistema de gestión de datos relacionales de código abierto optimizado para aplicaciones web, usado por sitios importantes como Airbnb, Uber, LinkedIn, Facebook, Twitter y YouTube.

2.3.2 Comandos básicos

Citando el sitio web [18] podemos observar los siguientes comandos y ejemplos básicos de su sintaxis:

Definiendo cómo es almacenada la información.

CREATE DATABASE se utiliza para crear una nueva base de datos vacía.

DROP DATABASE se utiliza para eliminar completamente una base de datos existente.

CREATE TABLE se utiliza para crear una nueva tabla, donde la información se almacena realmente.

ALTER TABLE se utiliza para modificar una tabla ya existente.

DROP TABLE se utiliza para eliminar por completo una tabla existente.

Manipulando los datos.

SELECT se utiliza cuando quieres leer (o seleccionar) tus datos.

INSERT se utiliza cuando quieres añadir (o insertar) nuevos datos.

UPDATE se utiliza cuando quieres cambiar (o actualizar) datos existentes.

DELETE se utiliza cuando quieres eliminar (o borrar) datos existentes.

REPLACE se utiliza cuando quieres añadir o cambiar (o reemplazar) datos nuevos o ya existentes.

TRUNCATE se utiliza cuando quieres vaciar (o borrar) todos los datos de la plantilla.

Un ejemplo sencillo.

```
CREAR  BASE DE DATOS mydb;
 UTILIZAR mydb;
 CREAR  TABLA  mitabla (id INT  PRIMARY  KEY , nombre VARCHAR ( 20 ));
 INSERTAR  EN VALORES DE mitabla ( 1 , 'Voluntad' );
 INSERTAR  EN VALORES DE mitabla ( 2 , 'Casarse' );
 INSERTAR  EN VALORES DE mitabla ( 3 , 'Dean' );
 SELECCIONE id, nombre FROM mitabla DONDE id =  1 ;
 ACTUALIZAR mitabla SET nombre =  'Willy'  WHERE id =  1 ;
 SELECCIONE id, nombre FROM mitabla;
 BORRAR  DE mitabla DONDE id =  1 ;
 SELECCIONE id, nombre FROM mitabla;
 BOTAR  BASE DE DATOS mydb;
 SELECCIONE  el recuento ( 1 ) de mitabla; da el número de registros en
la tabla
```

2.4 Microcontroladores

2.4.1 Definición

Un microcontrolador es un dispositivo utilizado para gestionar un proceso en específico [19], aunque la idea del microcontrolador se ha mantenido constante con el paso del tiempo, su forma física ha cambiado. Hoy en día, todos estos elementos se integran en un solo chip conocido como microcontrolador, que es esencialmente un pequeño computador dentro de un circuito integrado.

Todos los controladores, al estar contenidos en un chip suelen tener características similares. Todos incluyen procesador, memoria de datos e instrucciones, líneas de entrada y salida, oscilador de reloj y módulos para controlar periféricos, de igual manera cada fabricante destaca los recursos que mejor se adaptan a las aplicaciones específicas para las que están destinados.

Inicialmente, todos los microcontroladores utilizaban la arquitectura clásica de Von Neumann, pero actualmente se prefiere la arquitectura Harvard.

La arquitectura Von Neumann tiene una sola memoria principal para almacenar datos e instrucciones, accediendo a ella a través de un único sistema de buses.

La arquitectura Harvard, en cambio, tiene dos memorias separadas: una para instrucciones y otra para datos, cada una con su propio sistema de buses, permitiendo accesos simultáneos a ambas memorias.

El procesador es el componente principal de un microcontrolador y define sus características esenciales, tanto en hardware como en software. Se encarga de gestionar la memoria de instrucciones, interpretar y ejecutar las instrucciones, buscar operandos y almacenar resultados.

En los microcontroladores, las memorias de instrucciones y datos están integradas en el chip. Parte de esta memoria es no volátil, tipo ROM, y almacena el programa de instrucciones, mientras que otra parte es RAM, volátil, destinada a guardar variables y datos.

La RAM es limitada, ya que solo necesita almacenar variables y datos temporales durante la ejecución del programa, que se ejecuta directamente desde la ROM.

Para poder dar instrucciones a dichos microcontroladores se han desarrollado varios lenguajes para programar microcontroladores, siendo los más comunes Ensamblador, BASIC y C. El Ensamblador produce programas rápidos y compactos, pero puede ser complicado y extenso si no se codifica correctamente. Los lenguajes de alto nivel como BASIC y C son más accesibles y fáciles de usar. Sin embargo, los microcontroladores solo entienden código binario, por lo que los compiladores traducen los programas a código máquina para ser ejecutados.

2.4.2 Arduino

2.4.2.1 Características básicas

Cuando se habla de un microcontrolador Arduino se puede saber por medio de su página web [20], que la placa Arduino UNO utiliza el microcontrolador ATmega328P. Cuenta con 14 pines digitales de entrada/salida (6 de ellos utilizables como salidas PWM), 6 entradas analógicas, un resonador cerámico de 16 MHz, un puerto USB, un conector de alimentación, un cabezal ICSP y un botón de reinicio. Incluye todos los componentes necesarios para funcionar; solo se necesita conectar a una computadora mediante un cable USB o alimentarlo con un adaptador de CA a CC o una batería para comenzar a usarlo. Es un dispositivo fácil de manejar y, en caso de cometer errores, el chip puede ser reemplazado de manera económica.

2.4.2.2 Comandos básicos

En el ámbito de la programación en Arduino, existen comandos esenciales que facilitan el desarrollo de proyectos. A continuación, se presentan algunos comandos básicos más utilizados, con su sintaxis y una breve explicación de su función.

- **`setup()`**

  `void setup() { /* código */ }` Este comando se utiliza para configurar el programa. Se ejecuta una sola vez al iniciar el Arduino y se usa para inicializar variables, configurar pines y establecer las velocidades de comunicación.

- **`loop()`**

  `void loop() { /* código */ }` Define el cuerpo principal del programa que se ejecuta repetidamente mientras el Arduino esté encendido.

- **`pinMode()`**

  `pinMode(pin, mode);` Configura un pin específico del Arduino para que funcione como entrada (`INPUT`) o salida (`OUTPUT`).

- **`digitalWrite()`**

  `digitalWrite(pin, value);` Envía un nivel alto (`HIGH`) o bajo (`LOW`) a un pin digital configurado como salida.

- **`digitalRead()`**

  `digitalRead(pin);` Lee el valor de un pin digital configurado como entrada y devuelve `HIGH` o `LOW`.

- **`analogWrite()`**

  `analogWrite(pin, value);` Escribe un valor analógico (PWM) a un pin. El valor puede ser entre 0 y 255.

- **`analogRead()`**

`analogRead(pin);` Lee el valor de un pin analógico y devuelve un valor entre 0 y 1023.

- **`delay()`**

`delay(ms);` Pausa el programa durante un tiempo específico en milisegundos.

- **`millis()`**

`unsigned long millis();` Devuelve el número de milisegundos desde que la placa Arduino comenzó a ejecutar el programa actual.

- **`Serial.begin()`**

`Serial.begin(baudrate);` Inicializa la comunicación serial a una velocidad específica en baudios.

- **`Serial.print()`**

`Serial.print(data);` Envía datos a través del puerto serial para monitoreo.

- **`Serial.println()`**

`Serial.println(data);` Envía datos a través del puerto serial con un salto de línea al final.

Estos comandos constituyen la base de la programación en Arduino, permitiendo desde la configuración de pines hasta la comunicación serial, y son esenciales para la creación de una amplia gama de proyectos. [21]

2.5 Interfaz Arduino – Visual Studio

La conexión de Arduino con C# .Net como se puede entender en [22] que el comando Serial permite la transferencia de datos entre dispositivos enviando y recibiendo un bit a la vez a través de un solo canal. En el contexto de Arduino, esta técnica se emplea para intercambiar información entre la placa y una computadora u otros dispositivos externos. En el contexto de Arduino y Visual Studio, este protocolo es esencial para la transmisión de datos entre la computadora y el microcontrolador.

La escritura de puertos de Arduino desde Visual Studio permite controlar los pines digitales y analógicos del microcontrolador desde una aplicación de escritorio desarrollada en Visual Studio, del mismo modo, la lectura de puertos de Arduino desde Visual Studio implica obtener datos desde los pines digitales o analógicos del microcontrolador para mostrarlos o utilizarlos en dicha aplicación.

La lectura y escritura simultánea permite comunicarse bidireccionalmente entre Arduino y Visual Studio. Esto permite no solo enviar comandos desde la aplicación de Visual Studio a Arduino, sino también recibir datos y respuestas desde el microcontrolador

Capítulo III.
Desarrollo

3.1 Introducción a Visual Studio.

En el siguiente desarrollo se tratará de dar una explicación detallada acerca del proyecto en cuestión comenzando con la apertura de Visual Studio.

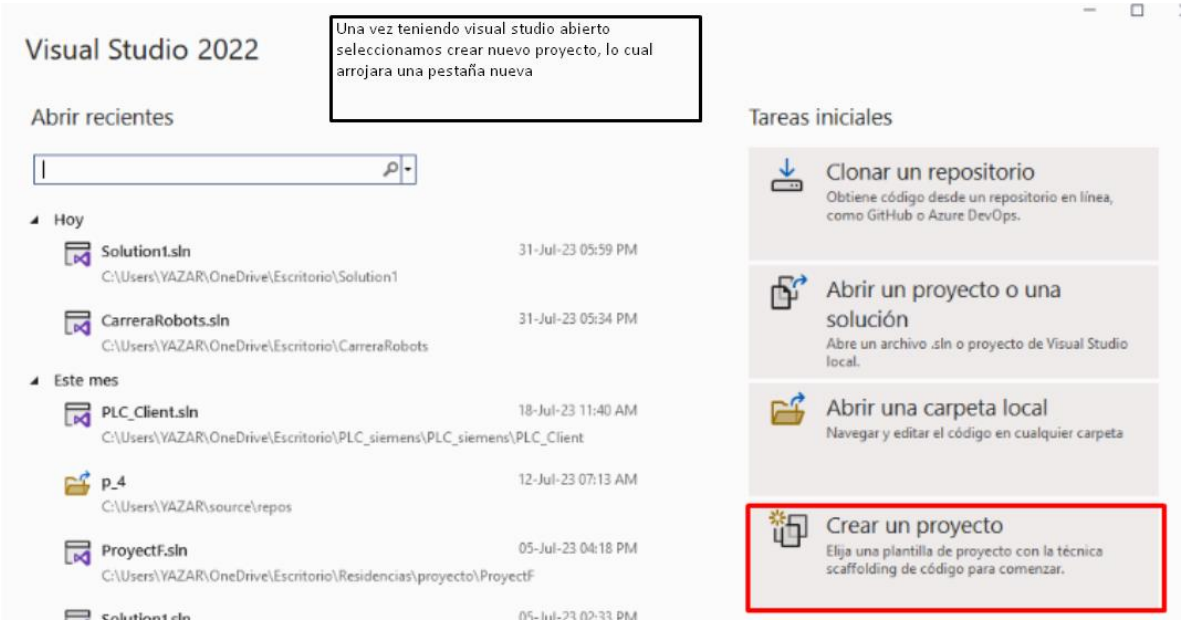

Ilustración 1: Inicio de Visual Studio (Ian Omar Madrigal Rueda)

Abriendo la IDE con una solución en blanco como se muestra a continuación:

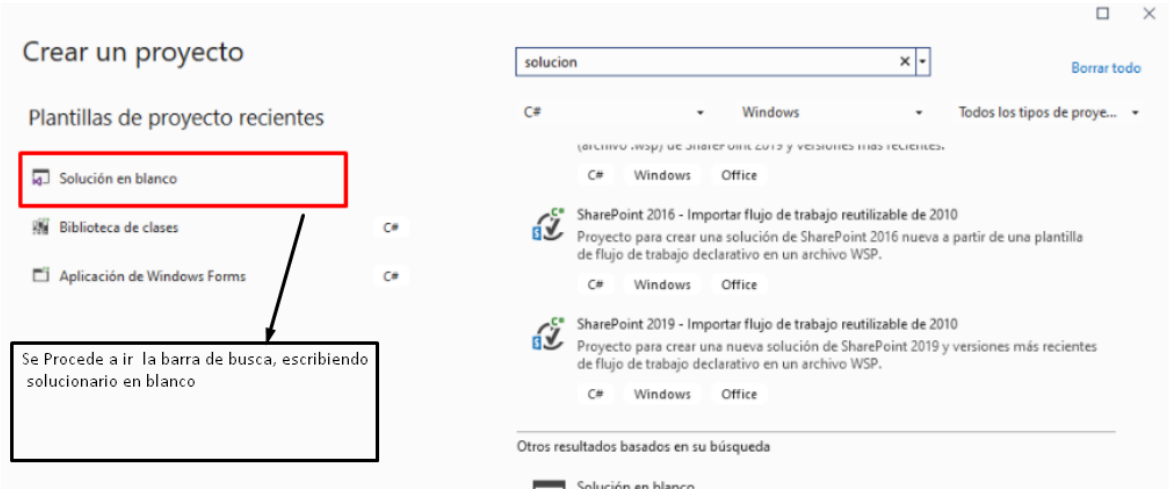

Ilustración 2: Solución en blanco (Ian Omar Madrigal Rueda)

Al tener la pantalla mostrada a continuación se debe organizar en el pc del desarrollador para poder continuar con el proyecto

Ilustración 3: Nombramiento y ubicación del proyecto (Ian Omar Madrigal Rueda)

Con la solución en blanco se necesita usar el explorador de solución.

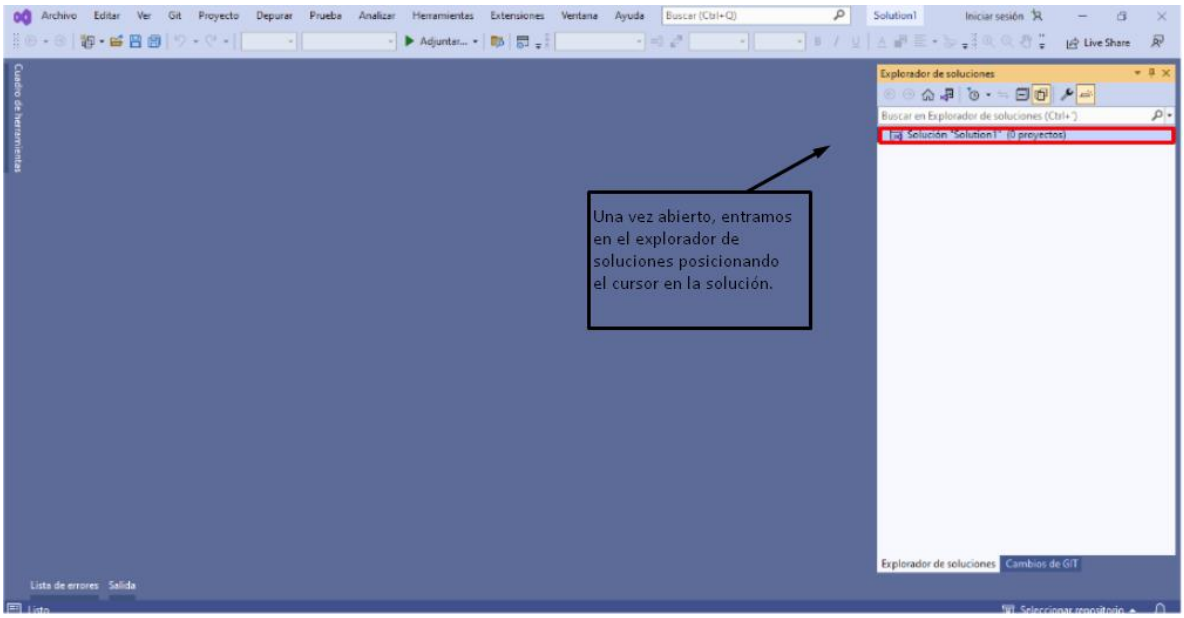

Ilustración 4: Explorador de soluciones (Ian Omar Madrigal Rueda)

Es necesario tener un proyecto dentro del IDE para poder organizar el entorno.

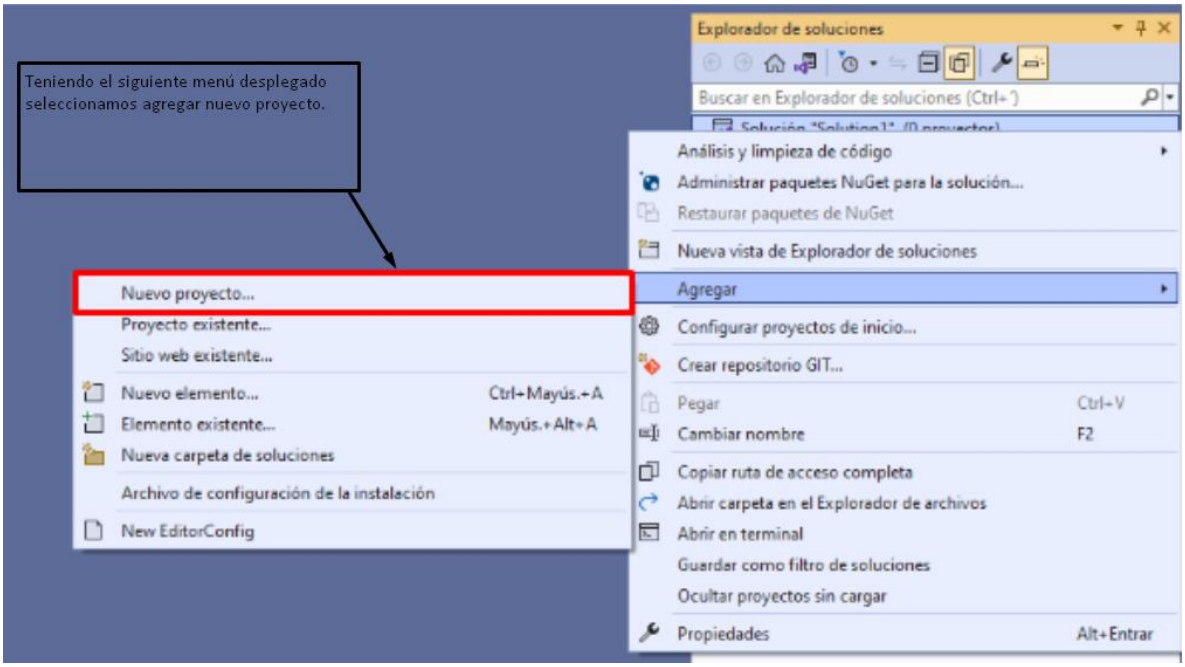

Ilustración 5: Nuevo proyecto (Ian Omar Madrigal Rueda)

3.2 Programación C# utilizando capas

Creando el proyecto es necesario agregar la aplicación .NET Framework .

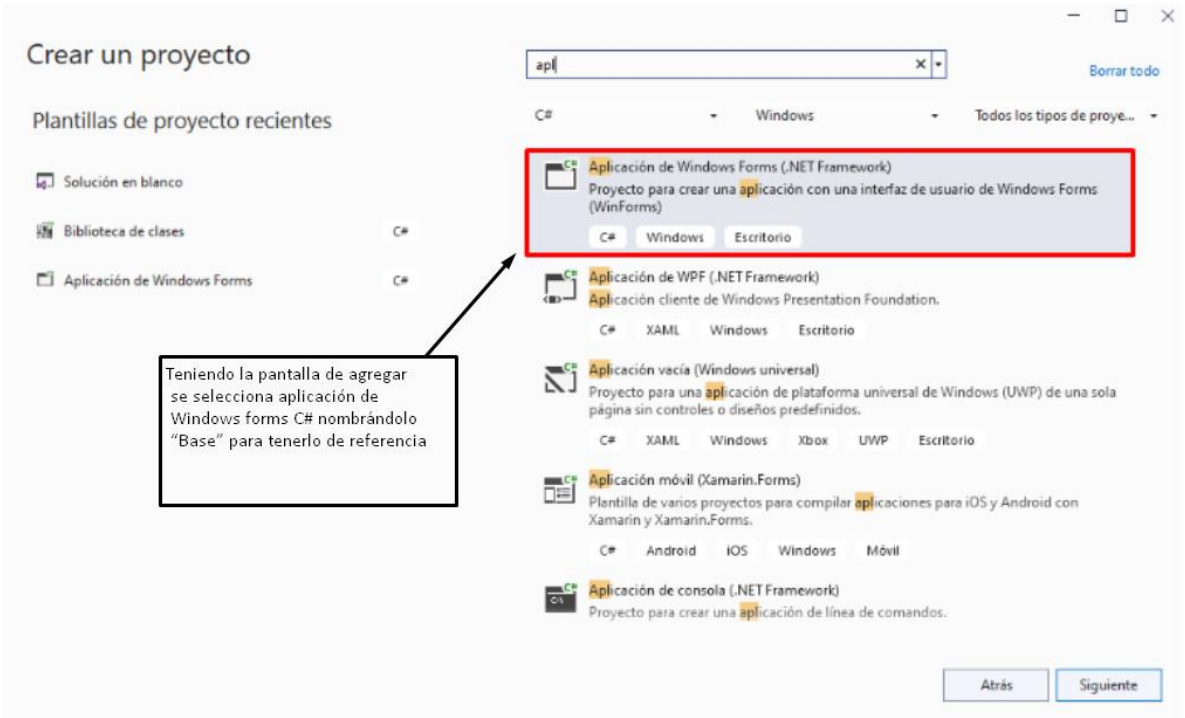

Ilustración 6: Aplicación Windows Forms (Ian Omar Madrigal Rueda)

Dándole un nombre para poder ser identificado más adelante.

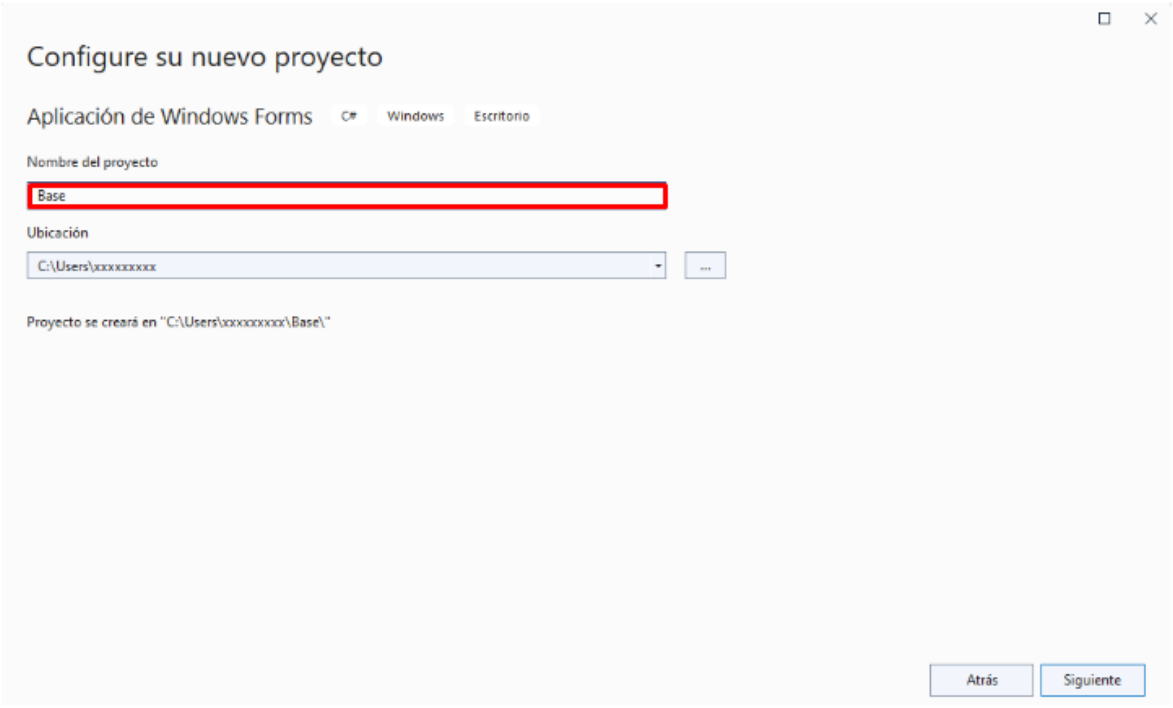

Ilustración 7:Configuración del proyecto (Ian Omar Madrigal Rueda)

Al tener diversas funciones a lo largo del proyecto es necesario separa por proyectos dentro del proyecto base.

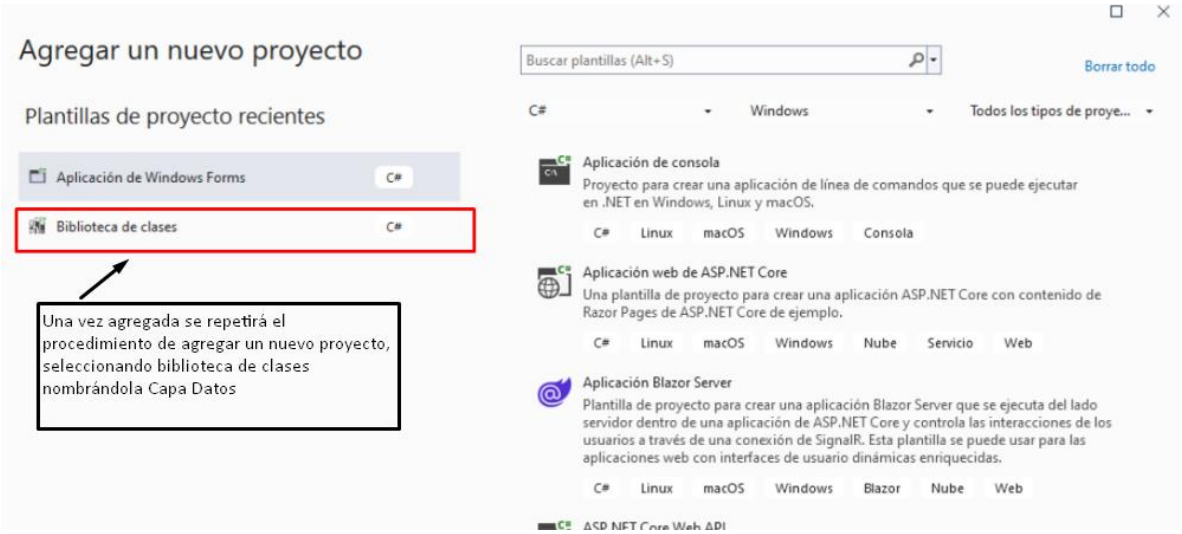

Ilustración 8: Proyecto- Biblioteca de clases (Ian Omar Madrigal Rueda)

Configurando el nombre para ser reconocido.

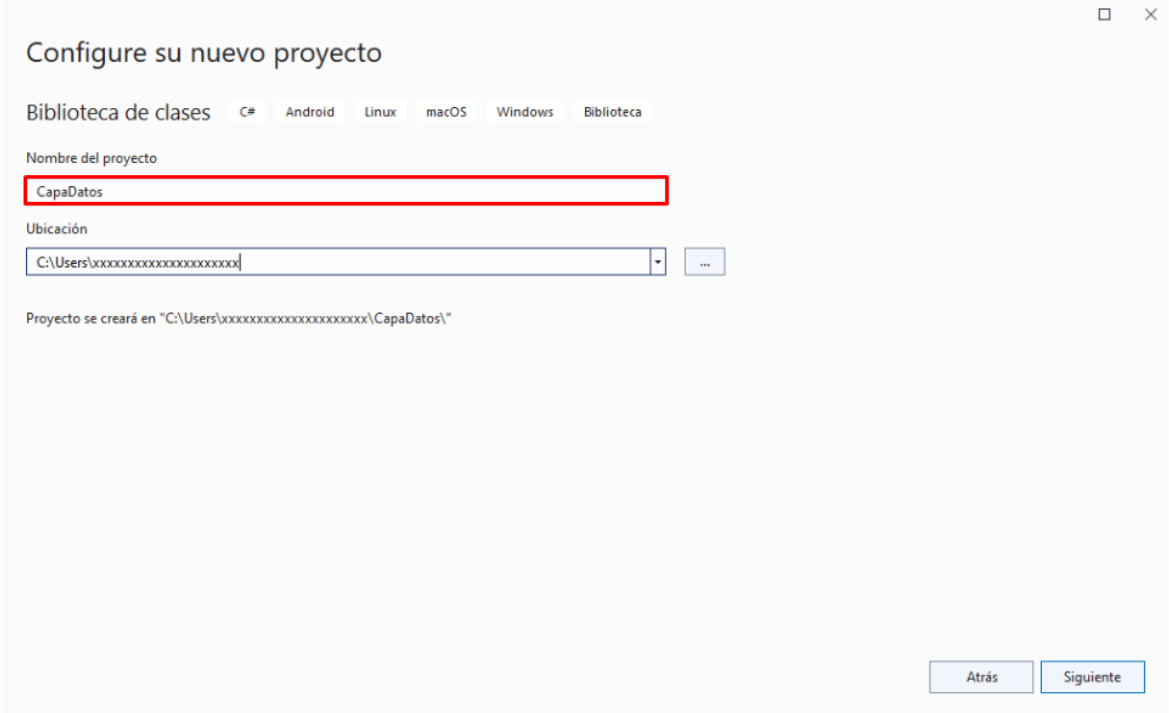

Ilustración 9: Configuración y almacenamiento CapaDatos (Ian Omar Madrigal Rueda)

Es necesario tener las bibliotecas completamente bacias para poder obtener una mayor organización.

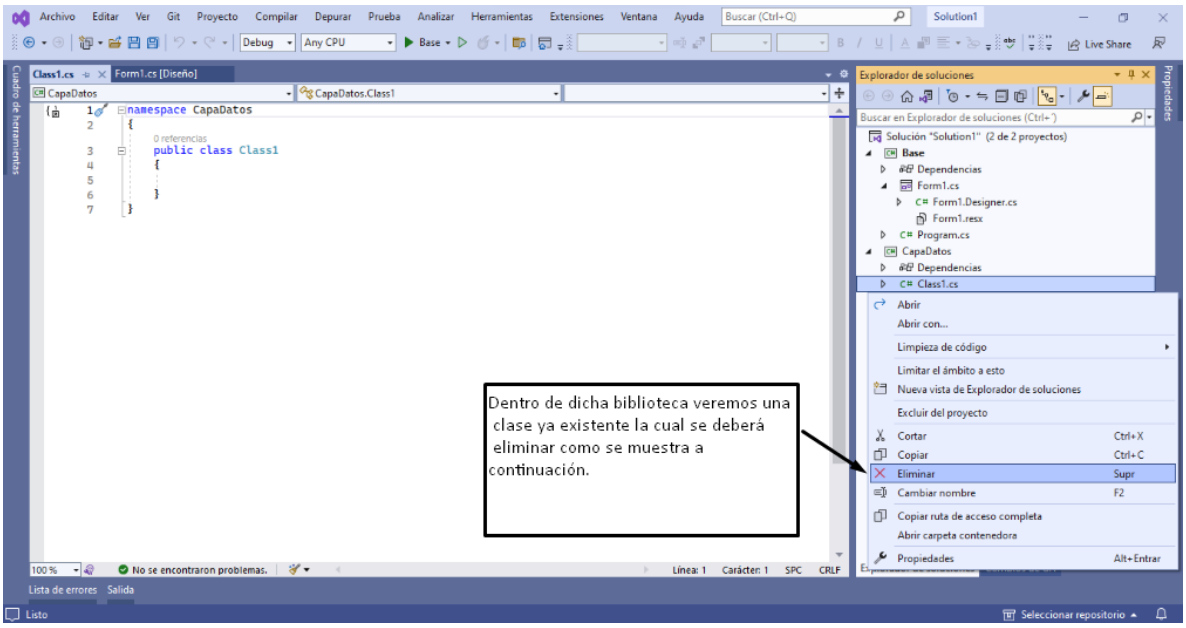

Ilustración 10: Eliminación de clase (Ian Omar Madrigal Rueda)

Por cada capa a trabajar es necesario un proyecto.

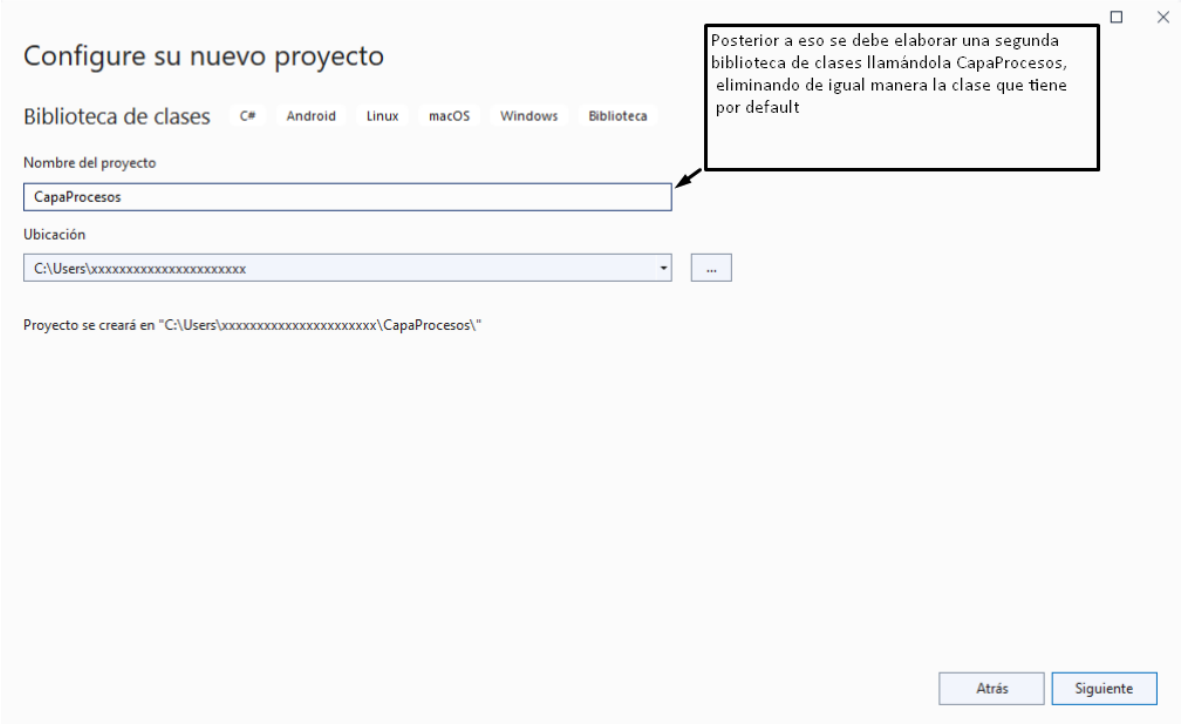

Ilustración 11: Configuración y almacenamiento CapaProcesos (Ian Omar Madrigal Rueda)

Usar referencias entre clases es lo que permite la comunicación a la hora de ser ejecutado el programa como se muestra.

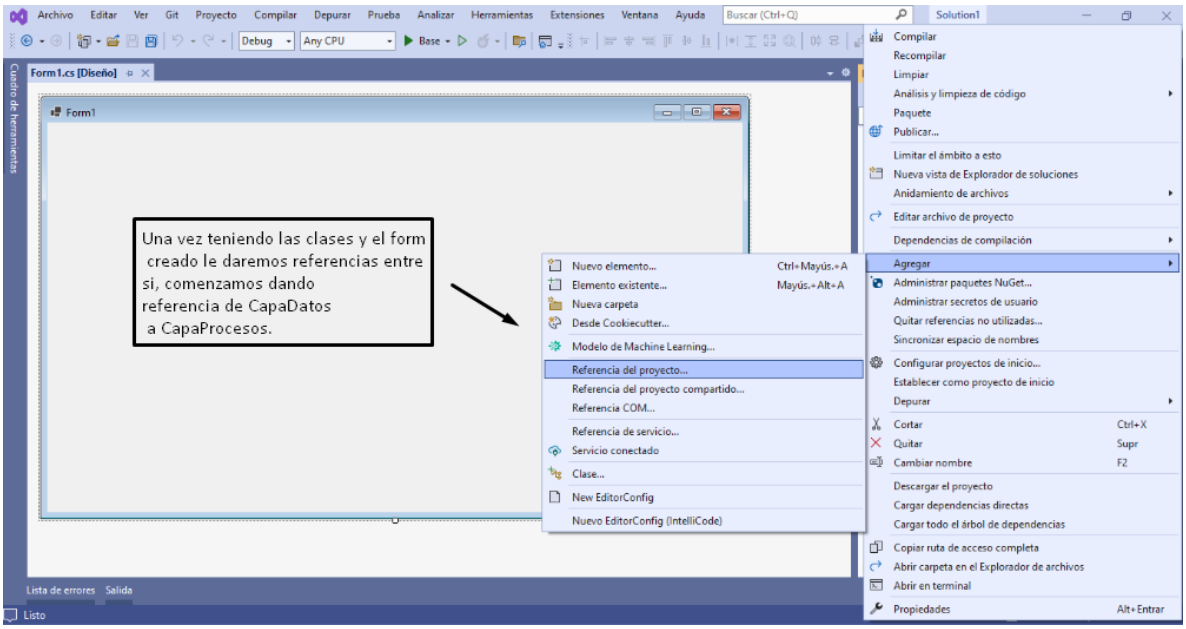

Ilustración 12: Creación de referencias (Ian Omar Madrigal Rueda)

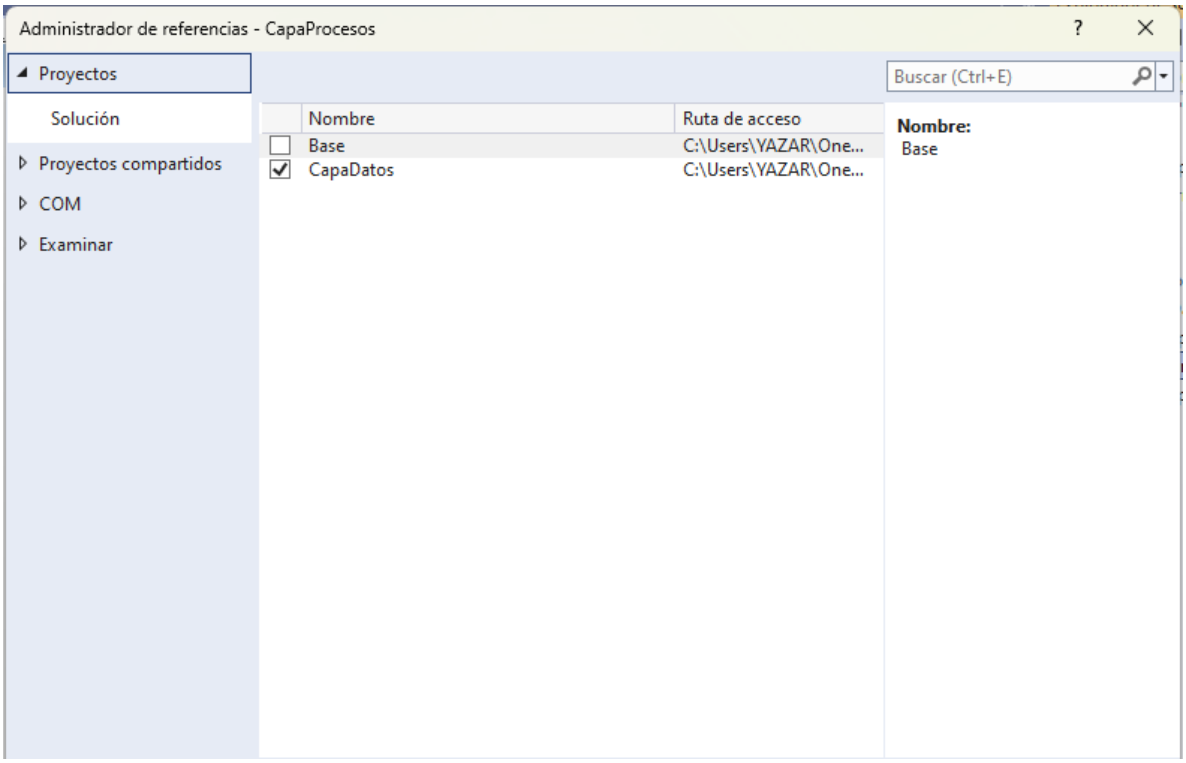

Ilustración 13: Referencia CapaProcesos-CapaDatos (Ian Omar Madrigal Rueda)

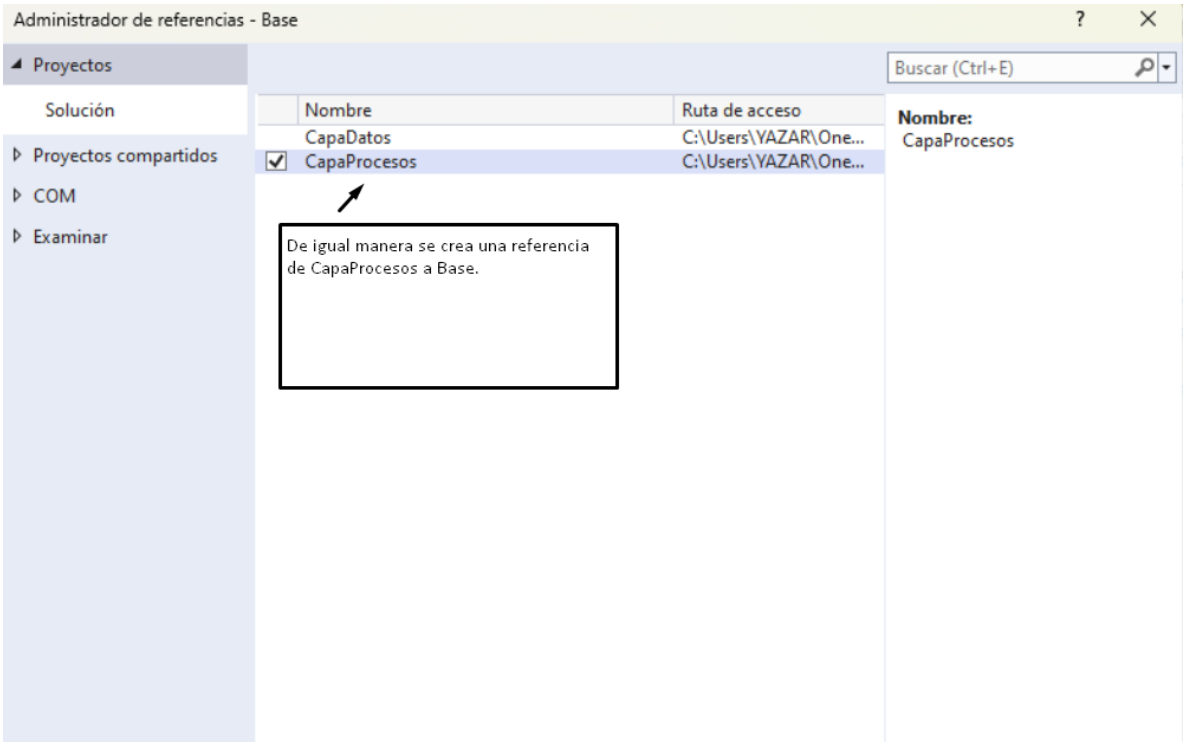

Ilustración 14: Referencia Base-CapaProcesos (Ian Omar Madrigal Rueda)

Se inicia el desarrollo de clases dentro de las bibliotecas para poder comenzar la conexión.

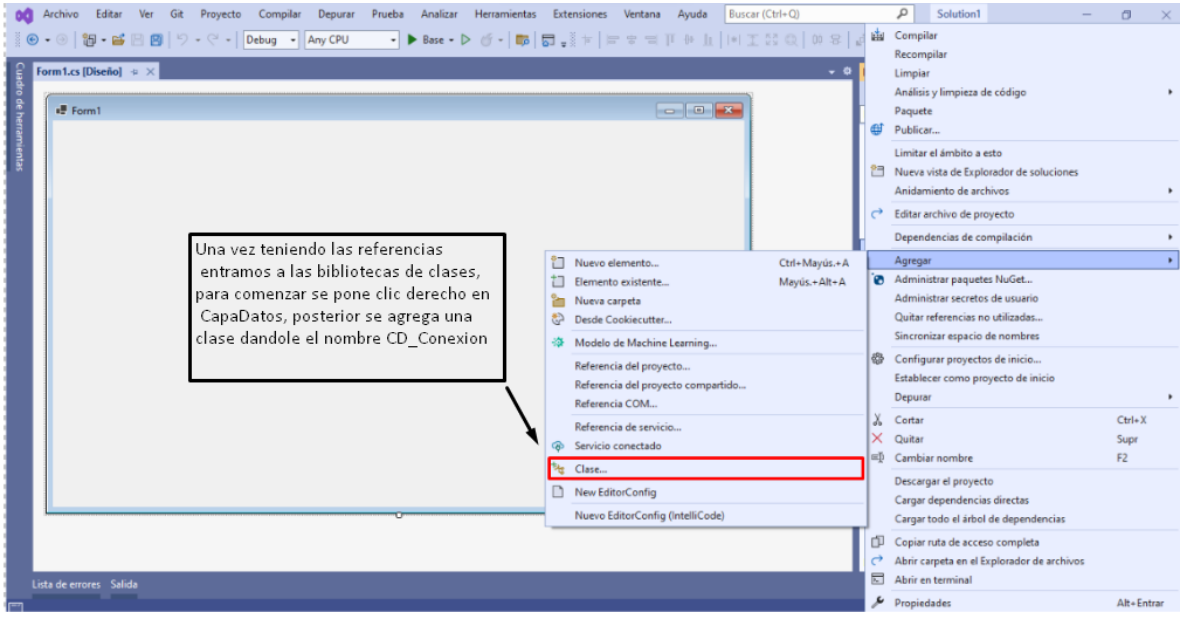

Ilustración 15: Creación de CD_Conexion (Ian Omar Madrigal Rueda)

Las clases al estar trabajando con C# deben tener el mismo lenguaje de programación.

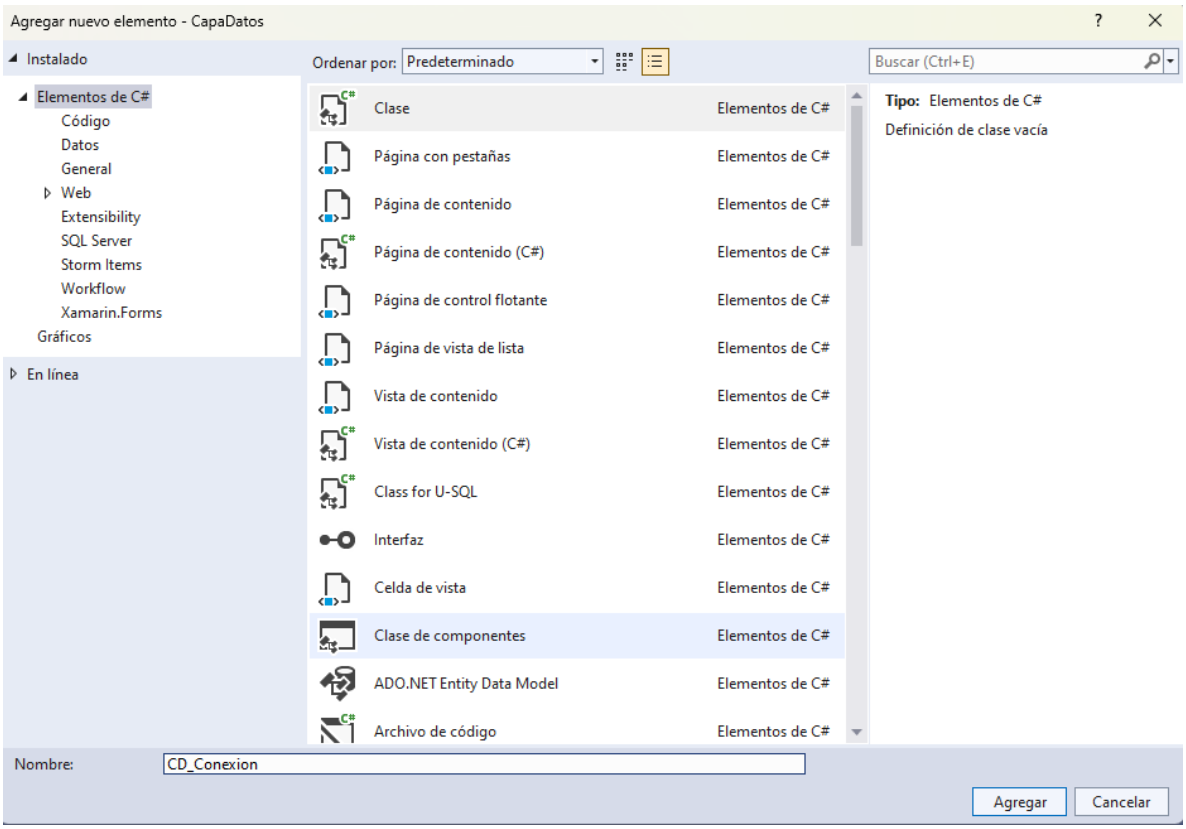

Ilustración 16: Clase de tipo C# (Ian Omar Madrigal Rueda)

Declarando el nombre de acuerdo con su propósito.

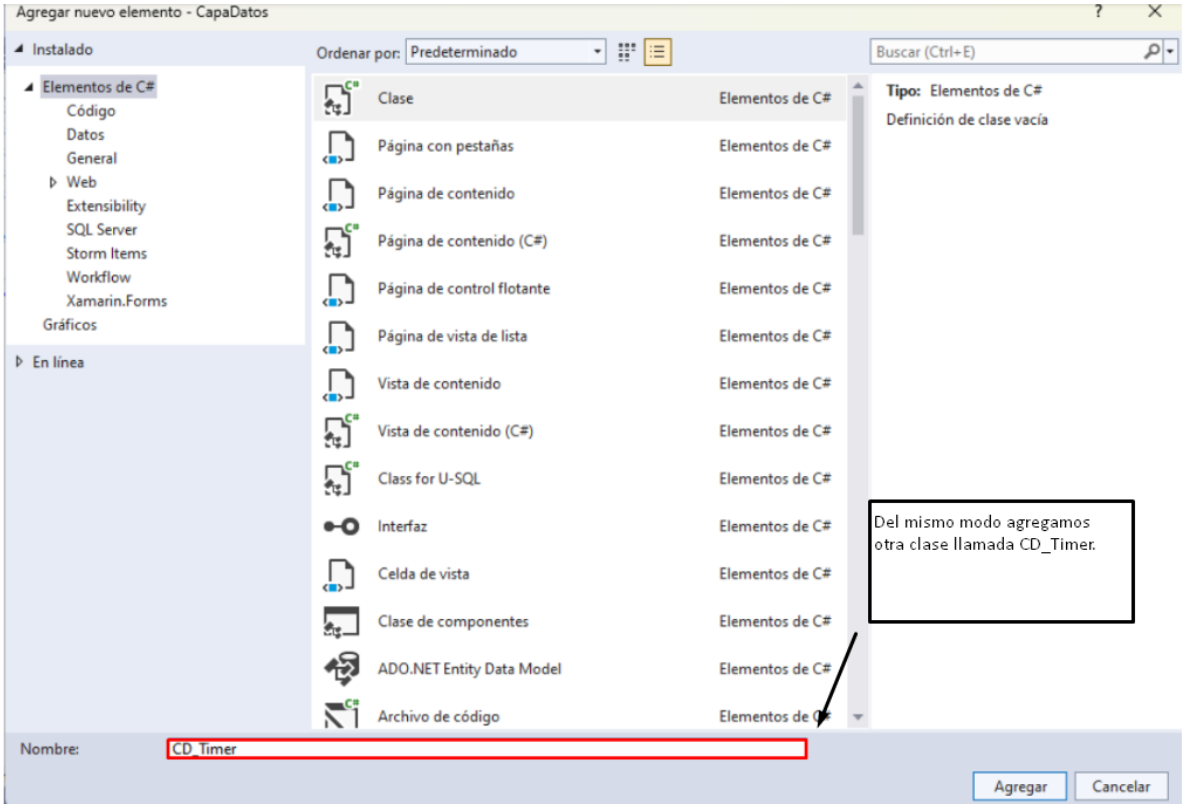

Ilustración 17: Clase CD_Timer (Ian Omar Madrigal Rueda)

3.3 Introducción a las bases de datos

Para hacer la conexión se trabajará en la base de datos SQL Server por la compatibilidad de la misma empresa Microsoft.

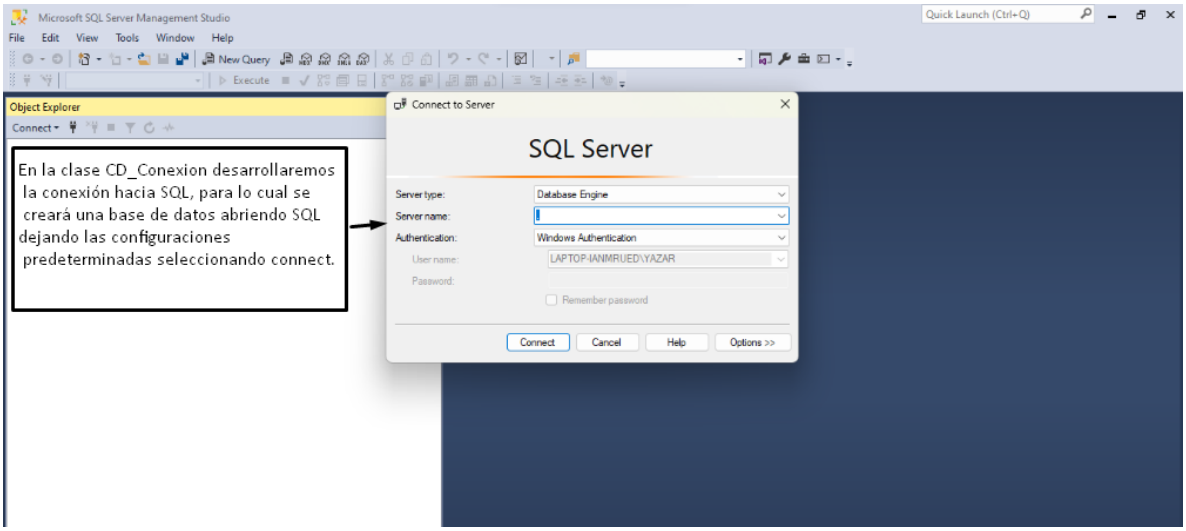

Ilustración 18: Conexión con SQL Server (Ian Omar Madrigal Rueda)

La creación de la base de datos es el primer paso en el desarrollo de SQL.

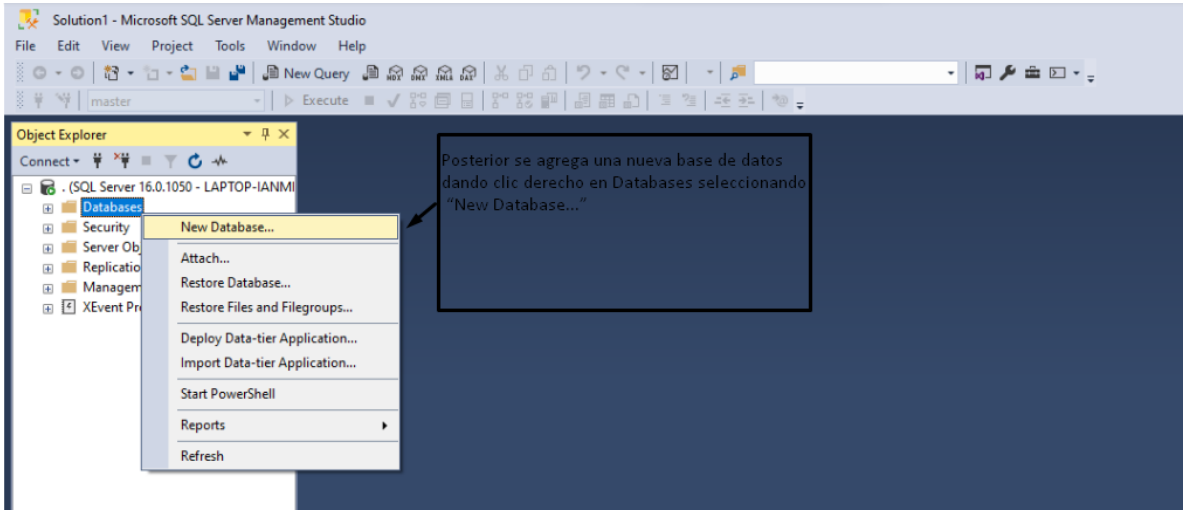

Ilustración 19: Creación de la base de datos (Ian Omar Madrigal Rueda)

El primer paso en la base de datos es la configuración de la tabla pensando en los datos obtenidos durante las pruebas.

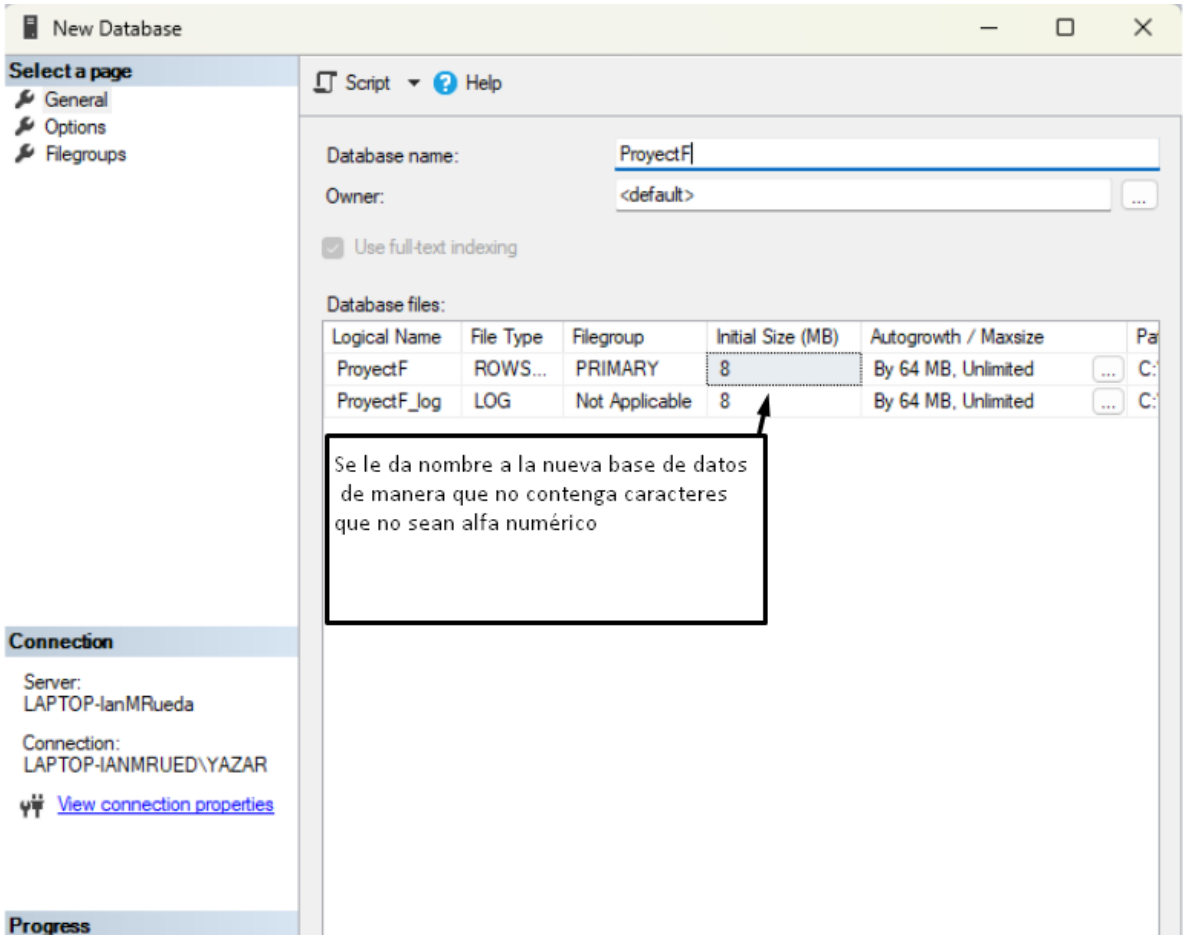

Ilustración 20: Nombre de la base de datos (Ian Omar Madrigal Rueda)

El código suele ser desarrollado en un New Query el cual es guardado automáticamente en la dirección del programa de SQL.

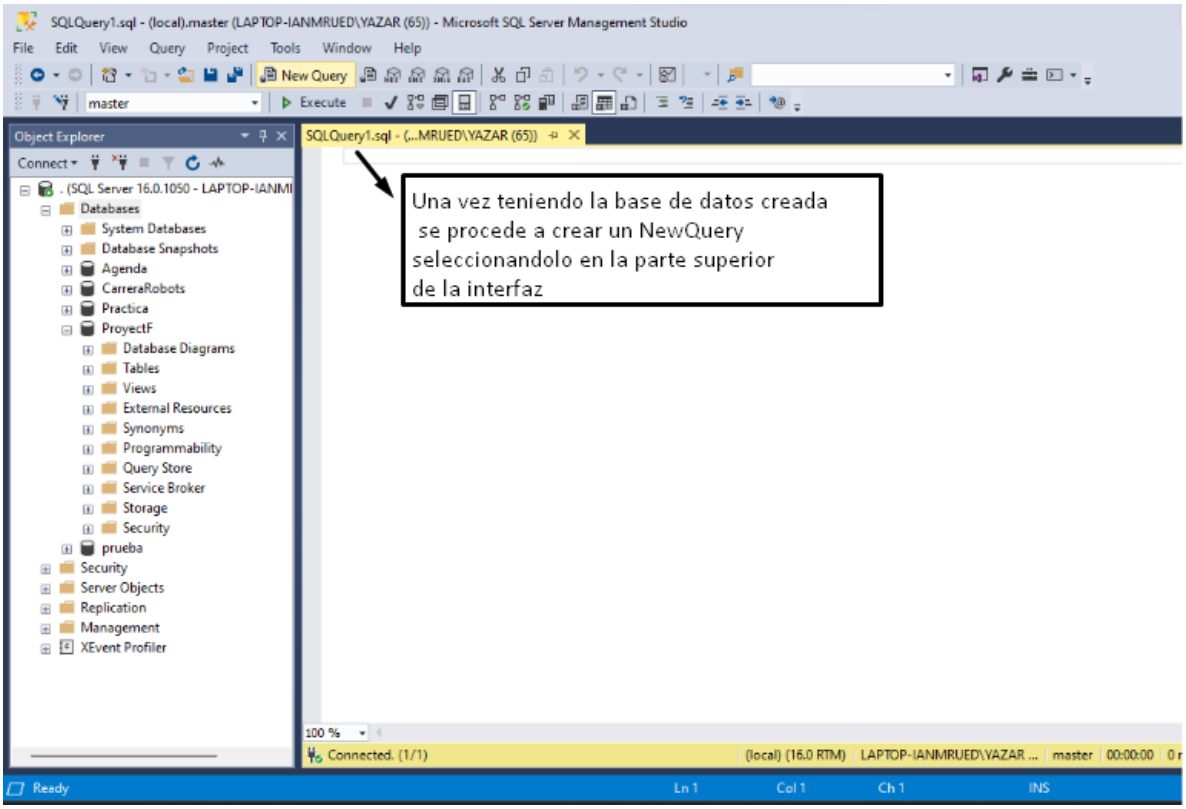

Ilustración 21: Elaboración del NewQuery (Ian Omar Madrigal Rueda)

3.4 Lenguaje SQL Consultas básicas

Una vez en el Query se digita el siguiente código en la hoja en blanco

```
use ProyectF
create table Timer
(
ID int identity (1,1) primary
key,
TIMER nvarchar (100)
)
ID int identity (1,1) primary
key,
create proc ClasificarZ
as
declare @n as int, @c as int
set @n = (select COUNT (*) from
timer)
if @n >64
begin
set @c = 64
end
```

```
else if @n > 32
begin
set @c = 32
end
else if @n > 16
begin
set @c = 16;
end
else if @n > 8
begin
set @c = 8;
end
select top (@c) * from timer
order by Timer
go
exec Clasificar

create proc Clasificar
```

```
as
select * from Timer
order by Timer
go
create proc Mostrar1
as
select * from timer
go
create proc InsertarCopm
```

```
@Timer nvarchar(200)
as
insert into timer values
(@Timer)
go
exec Mostrar1
insert into timer
Values
```

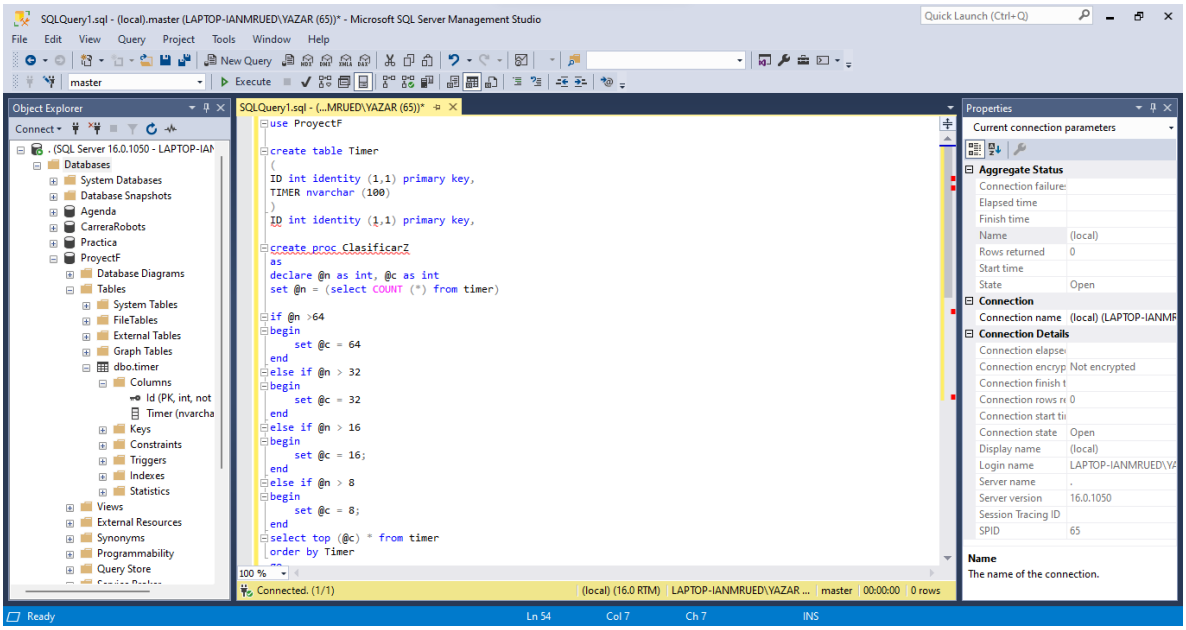

Ilustración 22: Código SQL Server (Ian Omar Madrigal Rueda)

Donde podemos observar cada línea y su propósito:

1. `use ProyectF`: Esta línea especifica que se va a utilizar la base de datos llamada "ProyectF" para realizar las operaciones posteriores. Esencialmente, establece el contexto de la base de datos en la que se ejecutarán las siguientes instrucciones.

2. `create table Timer (...)`: Esta línea crea una nueva tabla llamada "Timer" en la base de datos "ProyectF". La tabla tiene tres columnas: "ID" (entero autoincremental), "TIMER" (cadena de texto de hasta 100 caracteres). La columna "ID" se define como clave primaria (primary key), lo que garantiza la unicidad de cada registro en la tabla.

3. `create proc ClasificarZ as ...`: Esta línea crea un procedimiento almacenado (stored procedure) llamado "ClasificarZ". Los procedimientos almacenados son un conjunto de instrucciones SQL que se guardan en el servidor y se pueden ejecutar como una sola unidad. Este procedimiento no acepta

parámetros y se utilizará para clasificar los registros de la tabla "Timer" basándose en el número de registros en la tabla.

4. `declare @n as int, @c as int`: Aquí se declaran dos variables locales, "@n" y "@c", ambas de tipo entero (int). Estas variables se utilizarán para almacenar el número total de registros en la tabla "Timer" (@n) y para determinar la cantidad de registros a seleccionar después de la clasificación (@c).

5. `set @n = (select COUNT (*) from timer)`: Esta línea asigna a la variable "@n" el resultado de la consulta que cuenta el número total de registros en la tabla "Timer". En otras palabras, "@n" contendrá la cantidad de registros en la tabla.

6. `if @n >64 begin ...`: Estas líneas son una serie de condiciones IF-ELSE que determinan el valor de la variable "@c", que será la cantidad de registros a seleccionar después de la clasificación. Dependiendo del valor de "@n", se establecerá el valor de "@c" en 64, 32, 16 o 8.

7. `select top (@c) * from timer order by Timer`: Esta consulta seleccionará los primeros "@c" registros de la tabla "Timer" después de clasificarlos por la columna "Timer". En esencia, esta consulta devolverá los primeros registros ordenados alfabéticamente según la columna "Timer", donde "@c" es el número determinado en el paso anterior.

8. `go`: La instrucción "go" indica el final del procedimiento almacenado "ClasificarZ". Es una forma de separar distintas instrucciones en un script SQL.

9. `exec Clasificar`: Esta línea ejecuta el procedimiento almacenado "ClasificarZ" que se definió anteriormente. En otras palabras, clasifica los registros de la tabla "Timer" y selecciona los registros según la lógica descrita en el procedimiento.

10. `create proc Clasificar ...`: Esta línea crea otro procedimiento almacenado llamado "Clasificar". Este procedimiento almacenado no acepta parámetros y se utiliza para seleccionar todos los registros de la tabla "Timer" y ordenarlos alfabéticamente por la columna "Timer".

11. `create proc Mostrar1 ...`: Esta línea crea un tercer procedimiento almacenado llamado "Mostrar1". Este procedimiento no acepta parámetros y se utilizará para seleccionar todos los registros de la tabla "Timer" y mostrarlos tal como están en el orden en que se encuentran en la tabla.

12. `create proc InsertarCopm ...`: Esta línea crea otro procedimiento almacenado llamado "InsertarCopm". Este procedimiento acepta un parámetro "@Timer" de tipo nvarchar(200), que se utilizará para insertar un nuevo registro en la tabla "Timer" con el valor proporcionado en "@Timer".

13. `insert into timer values ...`: Esta línea realiza una inserción en la tabla "Timer" utilizando el procedimiento almacenado "InsertarCopm". Se inserta un nuevo registro en la tabla con el valor proporcionado en el parámetro "@Timer".

El código SQL proporcionado realiza diversas operaciones con una tabla llamada 'Timer' en la base de datos 'ProyectF'. Primero, crea la tabla 'Timer' con dos columnas, 'ID' y 'TIMER'. Luego, define un procedimiento almacenado llamado 'ClasificarZ', que clasifica los registros de la tabla 'Timer' según el número de registros que contiene. A continuación, se crea un procedimiento almacenado 'Clasificar' que ordena los registros de la tabla 'Timer' alfabéticamente. Además, hay otro procedimiento almacenado llamado 'Mostrar1', que simplemente selecciona y muestra todos los registros en el orden en que se encuentran en la tabla. También se proporciona un procedimiento almacenado 'InsertarCopm' para insertar un nuevo registro en la tabla 'Timer' con un valor específico.

Finalmente, se realiza una inserción utilizando el procedimiento almacenado 'InsertarCopm', lo que agrega un nuevo registro a la tabla 'Timer' con un valor determinado. En general, estos procedimientos almacenados permiten realizar diversas operaciones con la tabla 'Timer', como clasificar y mostrar registros, así como insertar nuevos datos."

3.5 Conexión entre C# y SQL Server

Una vez teniendo la base de datos, en la clase creada previamente llamada CD_Conexion se escribió el siguiente código.

CapaDatos | CapaDatos.CD_Conexion | AbrirConexion()

```
1   using System;
2   using System.Collections.Generic;
3   using System.Data;
4   using System.Linq;
5   using System.Text;
6   using System.Threading.Tasks;
7   using System.Data.SqlClient;
8
9   namespace CapaDatos
10  {
        2 referencias
11      public class CD_Conexion
12      {
13          private SqlConnection Conexion = new SqlConnection("Server=(local); DataBase=ProyectF; Integrated Security=true");
14
            4 referencias
15          public SqlConnection AbrirConexion()
16          {
17              if (Conexion.State == ConnectionState.Closed)
18                  Conexion.Open();
19              return Conexion;
20          }
            3 referencias
21          public SqlConnection CerrarConexion()
22          {
23              if (Conexion.State == ConnectionState.Open)
24                  Conexion.Close();
25              return Conexion;
26          }
27      }
28  }
29
```

Ilustración 23: Código de conexión Visual Studio-SQL Server (Ian Omar Madrigal Rueda)

Es una clase llamada `CD_Conexion` en C# que forma parte de una capa de datos en una aplicación. Su propósito es gestionar una conexión a la base de datos mediante el uso de la librería `System.Data.SqlClient`. Veamos una explicación detallada de cada línea del código:

1. `using System;`: Esta línea importa el espacio de nombres `System`, que contiene las clases y tipos fundamentales de C#.

2. `using System.Collections.Generic;`: Este `using` importa el espacio de nombres `System.Collections.Generic`, que proporciona clases genéricas para trabajar con colecciones de datos, como listas y diccionarios.

3. `using System.Data;`: Aquí, se importa el espacio de nombres `System.Data`, que proporciona clases para trabajar con datos y bases de datos.

4. `using System.Linq;`: Esta línea importa el espacio de nombres `System.Linq`, que contiene clases y extensiones para realizar consultas y operaciones en colecciones de datos.

5. `using System.Text;`: Aquí, se importa el espacio de nombres `System.Text`, que proporciona clases para manipular cadenas de texto y caracteres.

6. `using System.Threading.Tasks;`: Este `using` importa el espacio de nombres `System.Threading.Tasks`, que contiene clases y tipos para trabajar con tareas y operaciones asíncronas.

7. `using System.Data.SqlClient;`: Se importa el espacio de nombres `System.Data.SqlClient`, que proporciona clases para trabajar con SQL Server y permite establecer conexiones, ejecutar consultas y realizar otras operaciones relacionadas con bases de datos SQL Server.

8. `namespace CapaDatos`: Aquí, se define un espacio de nombres llamado `CapaDatos`, que es donde se encuentra la clase `CD_Conexion`.

9. `public class CD_Conexion`: Esta línea declara la clase pública `CD_Conexion`, que se encarga de gestionar la conexión a la base de datos.

10. `private SqlConnection Conexion = new SqlConnection("Server=(local); DataBase=ProyectF; Integrated Security=true");`: Se declara una variable privada llamada `Conexion`, que es del tipo `SqlConnection`. En esta línea, se inicializa la variable con una nueva instancia de `SqlConnection`, pasando como parámetro la cadena de conexión a la base de datos. La cadena de conexión indica el servidor de base de datos, el nombre de la base de datos y la autenticación integrada de Windows.

11. `public SqlConnection AbrirConexion()`: Se define un método público llamado `AbrirConexion()` que devuelve una instancia de `SqlConnection`.

12. `{`: Inicio del bloque de código del método `AbrirConexion()`.

13. `if (Conexion.State == ConnectionState.Closed)`: Aquí se utiliza una estructura condicional `if` para comprobar si el estado de la conexión (`Conexion.State`) es igual a `ConnectionState.Closed`, es decir, si la conexión está cerrada.

14. `Conexion.Open();`: Si el estado de la conexión es cerrado, se invoca el método `Open()` de la instancia `Conexion`, lo que abre la conexión a la base de datos.

15. `return Conexion;`: Se devuelve la instancia de `SqlConnection` `Conexion`.

16. `}`: Fin del bloque de código del método `AbrirConexion()`.

17. `public SqlConnection CerrarConexion()`: Se define otro método público llamado `CerrarConexion()` que devuelve una instancia de `SqlConnection`.

18. `{`: Inicio del bloque de código del método `CerrarConexion()`.

19. `if (Conexion.State == ConnectionState.Open)`: Se verifica mediante un condicional `if` si el estado de la conexión (`Conexion.State`) es igual a `ConnectionState.Open`, lo que significa que la conexión está abierta.

20. `Conexion.Close(); `: Si la conexión está abierta, se llama al método `Close()` de la instancia `Conexion` para cerrar la conexión.

21. `return Conexion;`: Se devuelve la instancia de `SqlConnection` `Conexion`.

22. `} `: Fin del bloque de código del método `CerrarConexion()`.

En resumen, la clase `CD_Conexion` proporciona dos métodos públicos: `AbrirConexion()` y `CerrarConexion()`, que se utilizan para abrir y cerrar la conexión a la base de datos SQL Server. La cadena de conexión se especifica en el constructor de la clase y contiene la información necesaria para conectarse al servidor y a la base de datos. Estos métodos son útiles cuando se necesita interactuar con la base de datos en otras partes del programa, como la capa de negocios o la interfaz de usuario.

Posteriormente en la misma CapaDatos se crea una clase llamada CD_Timer en la cual se agrega el siguiente código (En la imagen se muestra solo un fragmento):

```
8
9   namespace CapaDatos
10  {
        6 referencias
11      public class CD_Timer
12      {
13          private CD_Conexion conexion = new CD_Conexion();
14
15          SqlDataReader leer;
16          DataTable tabla = new DataTable();
17          SqlCommand comando = new SqlCommand();
18
            1 referencia
19          public DataTable Mostrar()
20          {
21              //TRANSACT SQL
22              comando.Connection = conexion.AbrirConexion();
23              comando.CommandText = "Mostrar1";
24              comando.CommandType = CommandType.StoredProcedure;
25              leer = comando.ExecuteReader();
26              tabla.Load(leer);
27              conexion.CerrarConexion();
28              return tabla;
29              //PROCEDIMIENTO
30          }
            1 referencia
31          public void Insertar(string Timer)
32          {
33              comando.Connection = conexion.AbrirConexion();
34              comando.CommandText = "InsertarCopm";
35              comando.CommandType = CommandType.StoredProcedure;
36              comando.Parameters.AddWithValue("@Timer", Timer);
37              comando.ExecuteNonQuery();
38              comando.Parameters.Clear();
39
40          }
```

Ilustración 24: Colocación del tiempo en SQL Server -Visual Studio (Ian Omar Madrigal Rueda)

A continuación, se explica el código a profundidad, código en C# proporcionado corresponde a una clase llamada `CD_Timer` en el espacio de nombres `CapaDatos`. Esta clase se encarga de interactuar con la base de datos y contiene tres métodos principales: `Mostrar () `, `Insertar (string Timer)`, `Clasificar () `, y `ClasificarZ()`. Vamos a explicar línea por línea el funcionamiento y el propósito de cada parte del código:

```
using System;
using System.Collections.Generic;
using System.Data.SqlClient;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

Estas son las directivas de `using`, que permiten importar diferentes espacios de nombres (namespaces) y sus tipos relacionados para que puedan ser utilizados en el código. En este caso, se incluyen namespaces relacionados con el manejo de bases de datos y operaciones de entrada/salida.

```
namespace CapaDatos
{
    public class CD_Timer
    {
        // ...
    }
}
```

El código está contenido dentro del espacio de nombres `CapaDatos` y define una clase llamada `CD_Timer`.

```
private CD_Conexion conexion = new CD_Conexion();
```

Se declara e instancia una variable `conexion` de tipo `CD_Conexion`. Esto sugiere que hay una clase llamada `CD_Conexion` que maneja la conexión con la base de datos.

```
SqlDataReader leer;
DataTable tabla = new DataTable();
SqlCommand comando = new SqlCommand();
```

Se declaran tres variables: `leer`, `tabla` y `comando`. `leer` es de tipo `SqlDataReader`, utilizado para leer datos de la base de datos, `tabla` es de tipo `DataTable`, una estructura de datos que almacena los resultados de una consulta y `comando` es de tipo `SqlCommand`, que se utiliza para ejecutar comandos SQL en la base de datos.

```
public DataTable Mostrar()
{
    //TRANSACT SQL
    comando.Connection = conexion.AbrirConexion();
    comando.CommandText = "Mostrar1";
    comando.CommandType = CommandType.StoredProcedure;
    leer = comando.ExecuteReader();
    tabla.Load(leer);
    conexion.CerrarConexion();
    return tabla;
    //PROCEDIMIENTO
}
```

Este método `Mostrar()` devuelve un objeto `DataTable` que contiene los datos obtenidos de la base de datos. El método se encarga de ejecutar una consulta almacenada en la base de datos llamada "Mostrar1" y cargar los resultados en la tabla. Luego, se cierra la conexión y se retorna la tabla con los datos.

```
public void Insertar(string Timer)
{
    comando.Connection = conexion.AbrirConexion();
    comando.CommandText = "InsertarCopm";
    comando.CommandType = CommandType.StoredProcedure;
    comando.Parameters.AddWithValue("@Timer", Timer);
    comando.ExecuteNonQuery();
    comando.Parameters.Clear();
}
```

El método `Insertar(string Timer)` permite insertar un registro en la base de datos. Utiliza un procedimiento almacenado llamado "InsertarCopm" y asigna el valor del parámetro `@Timer` con el valor pasado al método. Luego, ejecuta el comando utilizando `ExecuteNonQuery()`, lo que indica que no se espera una respuesta de la base de datos.

```
public DataTable Clasificar()
{
    //TRANSACT SQL
    comando.Connection = conexion.AbrirConexion();
    comando.CommandText = "Orden1";
    comando.CommandType = CommandType.StoredProcedure;
    leer = comando.ExecuteReader();
    tabla.Load(leer);
    conexion.CerrarConexion();
    return tabla;
    //PROCEDIMIENTO
}
```

El método `Clasificar()` es similar a `Mostrar()` pero ejecuta una consulta diferente almacenada en la base de datos llamada "Orden1".

```
public DataTable ClasificarZ()
{
    //TRANSACT SQL
    comando.Connection = conexion.AbrirConexion();
    comando.CommandText = "Clasificar";
    comando.CommandType = CommandType.StoredProcedure;
    leer = comando.ExecuteReader();
    tabla.Load(leer);
    conexion.CerrarConexion();
    return tabla;
    //PROCEDIMIENTO
}
```

El método `ClasificarZ()` también es similar a `Mostrar()` pero ejecuta un procedimiento almacenado diferente llamado "Clasificar".

En resumen, la clase `CD_Timer` proporciona métodos para interactuar con la base de datos, como obtener datos, insertar registros y realizar clasificaciones mediante el uso de procedimientos almacenados y consultas en la base de datos. La clase se apoya en una clase de conexión `CD_Conexion` para establecer y cerrar la conexión con la base de datos.

Del mismo modo en la biblioteca de clases CapaProcesos se agregó una clase llamada CP_Timer como ya se explicó con anterioridad, en la cual se desarrolló el siguiente código.

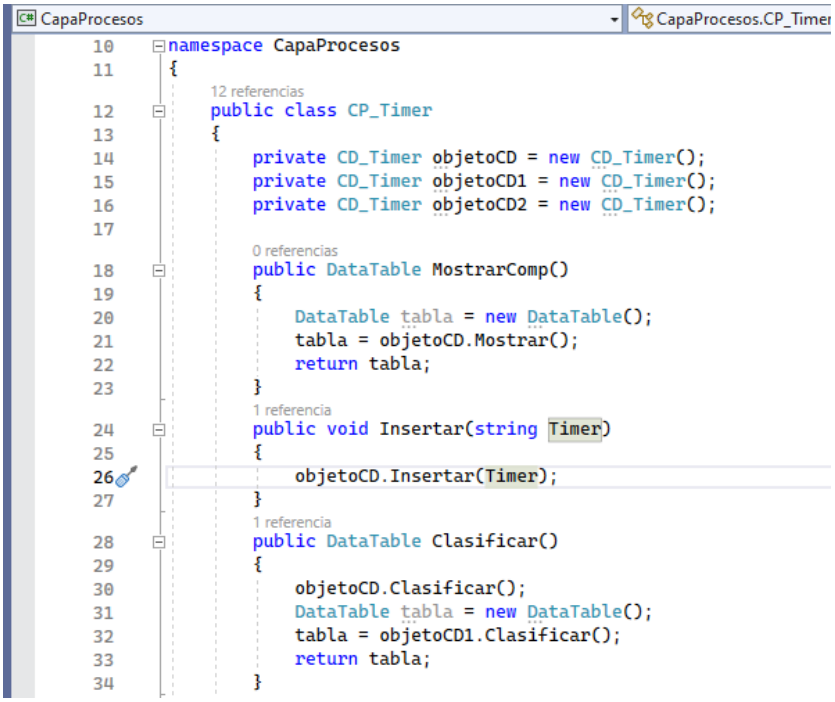

Ilustración 25: Desarrollo del código CapaProcesos (Ian Omar Madrigal Rueda)

El código es una clase en C# llamada `CP_Timer` que se encuentra dentro del namespace `CapaProcesos`. A continuación, se explicará línea por línea para comprender su funcionalidad:

1. `using System;`: Esta línea importa el namespace `System`, que es necesario para utilizar tipos y funciones básicas de C#.

2. `using System.Collections.Generic;`: Importa el namespace `System.Collections.Generic`, que proporciona tipos y clases para trabajar con colecciones genéricas, como listas y diccionarios.

3. `using System.Data;`: Importa el namespace `System.Data`, que contiene clases para trabajar con datos y bases de dato.

4. `using System.Linq;`: Importa el namespace `System.Linq`, que proporciona extensiones para consultar colecciones y realizar operaciones de LINQ (Language Integrated Query).

5. `using System.Text;`: Importa el namespace `System.Text`, que contiene clases para trabajar con codificaciones de caracteres y manipulación de cadenas.

6. `using System.Threading.Tasks;`: Importa el namespace `System.Threading.Tasks`, que proporciona clases y métodos para trabajar con tareas asincrónicas.

7. `using CapaDatos;`: Importa el namespace `CapaDatos`, que debe contener la definición de la clase `CD_Timer`, utilizada en el código.

8. `using System.Data.SqlClient;`: Importa el namespace `System.Data.SqlClient`, que proporciona clases para trabajar con el proveedor de datos de SQL Server.

9. `namespace CapaProcesos`: Define el namespace `CapaProcesos`, que agrupa todas las clases y tipos relacionados con la lógica de procesos de la aplicación.

10. `public class CP_Timer`: Declara una clase pública llamada `CP_Timer`. Esta clase actúa como la capa de procesos para interactuar con los datos relacionados con el "Timer".

11. `private CD_Timer objetoCD = new CD_Timer();`: Crea una instancia privada de la clase `CD_Timer` llamada `objetoCD`. Esto permite que la clase `CP_Timer` acceda a los métodos y propiedades de la clase `CD_Timer`.

12. `private CD_Timer objetoCD1 = new CD_Timer();`: Crea otra instancia privada de la clase `CD_Timer` llamada `objetoCD1`.

13. `private CD_Timer objetoCD2 = new CD_Timer();`: Crea otra instancia privada de la clase `CD_Timer` llamada `objetoCD2`.

14. `public DataTable MostrarComp()`: Declara un método público llamado `MostrarComp` que devuelve un objeto DataTable. Este método es responsable de obtener datos relacionados con el "Timer" desde la capa de datos (`objetoCD.Mostrar()`) y devolverlos.

15. `public void Insertar (string Timer)`: Declara un método público llamado `Insertar` que recibe un parámetro `Timer` de tipo string. Este método es responsable de insertar un registro relacionado con el "Timer" en la capa de datos (`objetoCD.Insertar(Timer)`).

16. `public DataTable Clasificar ()`: Declara un método público llamado `Clasificar` que devuelve un objeto DataTable. Este método es responsable de clasificar datos relacionados con el "Timer" en la capa de datos (`objetoCD.Clasificar()`) y devolverlos.

17. `public DataTable ClasificarZ()`: Declara otro método público llamado `ClasificarZ` que devuelve un objeto DataTable. Este método también clasifica datos relacionados con el "Timer" en la capa de datos (`objetoCD.Clasificar()`), pero utiliza otra instancia (`objetoCD2`) de la clase `CD_Timer` para realizar la clasificación (`objetoCD2.ClasificarZ() `) antes de devolver los resultados.

En resumen, la clase `CP_Timer` representa la capa de procesos de una aplicación. Utiliza instancias de la clase `CD_Timer` (ubicada en el namespace `CapaDatos`) para interactuar con los datos relacionados con el "Timer". Los métodos de esta clase se encargan de mostrar, insertar y clasificar información sobre el "Timer" utilizando la capa de datos correspondiente.

3.6 Diseño de la interfaz gráfica en C#

El siguiente paso para desarrollar es el form llamado base en el cual se usará la barra de herramientas visualizada en la parte izquierda de la interfaz de visual studio.

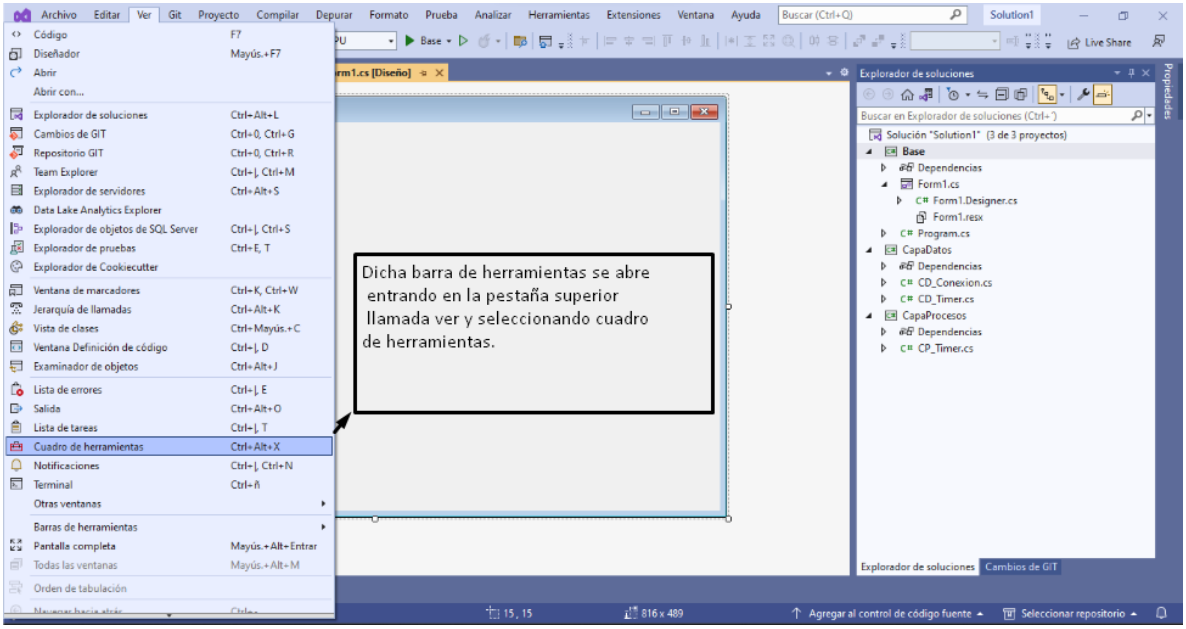

Ilustración 26: Herramientas de Windows Forms (Ian Omar Madrigal Rueda)

Una vez teniendo el cuadro de herramientas, se familiarizo con los siguientes comandos arrastrándolos hacia la ventana del formulario.

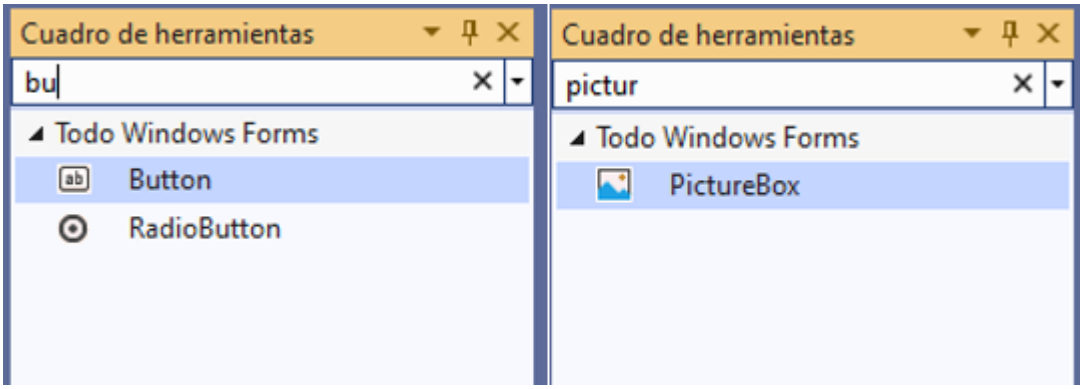

Ilustración 27: Button y PictureBox en el Cuadro de herramientas (Ian Omar Madrigal Rueda)

Usando PictureBox para darle estética al formulario con un encabezado de preferencia; ya teniendo el siguiente formato se trabajó con la ventana de propiedades mostrada a la derecha de la interfaz, la cual aparece cuando ponemos clic en cualquiera de los elementos deseados.

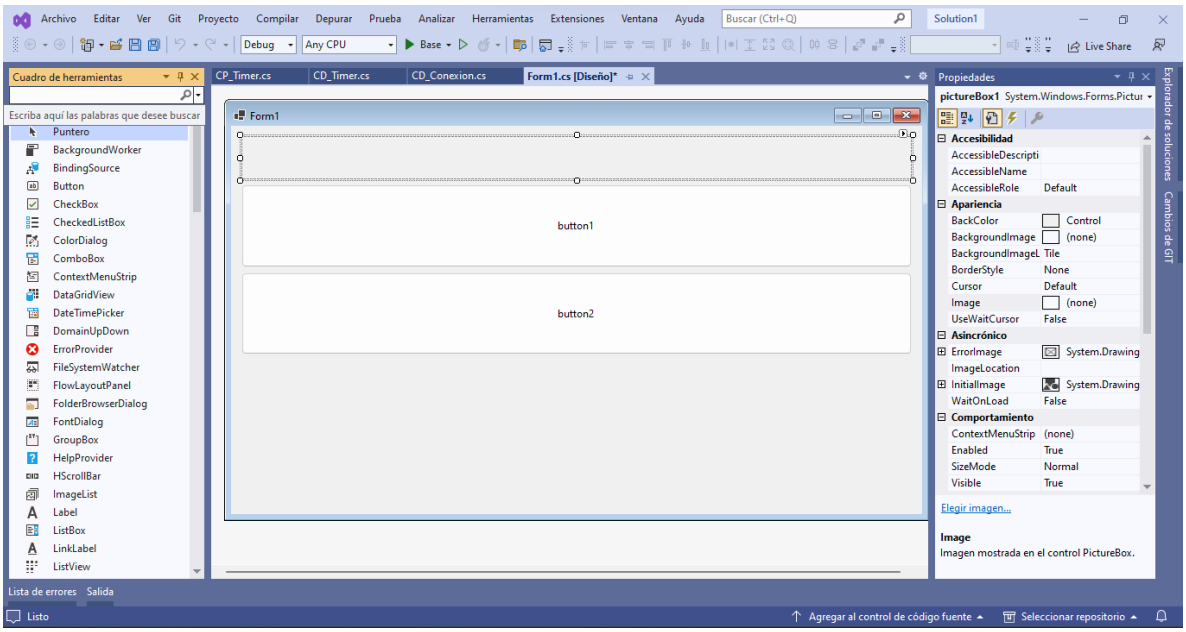

Ilustración 28: Botones principales en la interfaz de usuario (Ian Omar Madrigal Rueda)

En la ventana de propiedades que nos muestra al seleccionar el PictureBox se encuentra el menú apariencia donde seleccionaremos el apartado “image”, en dicho apartado importaremos la imagen de preferencia para el encabezado.

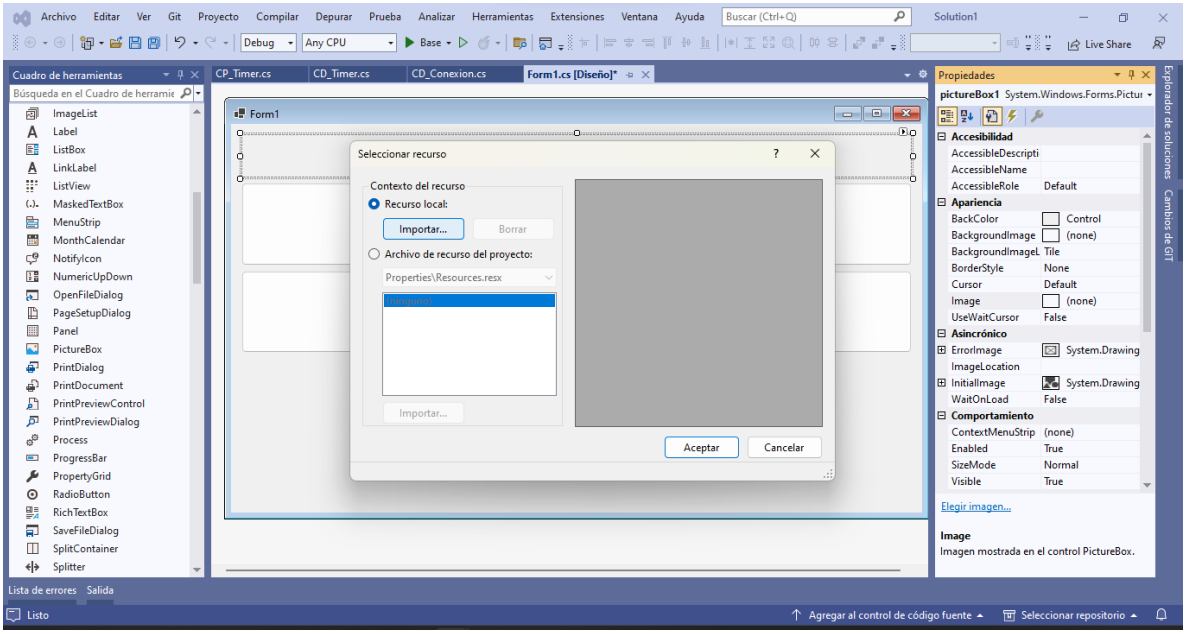

Ilustración 29: importación de imagen para presentación (Ian Omar Madrigal Rueda)

Del mismo modo posicionándonos en el formulario principal se buscaron los apartados “(name)” y “text” para escribir en ellos el nombre del formulario (Base), se hizo el mismo procedimiento con botton1 y botton2 cambiando “(name)” y “text” a Fase1 y Fase2 correspondientemente.

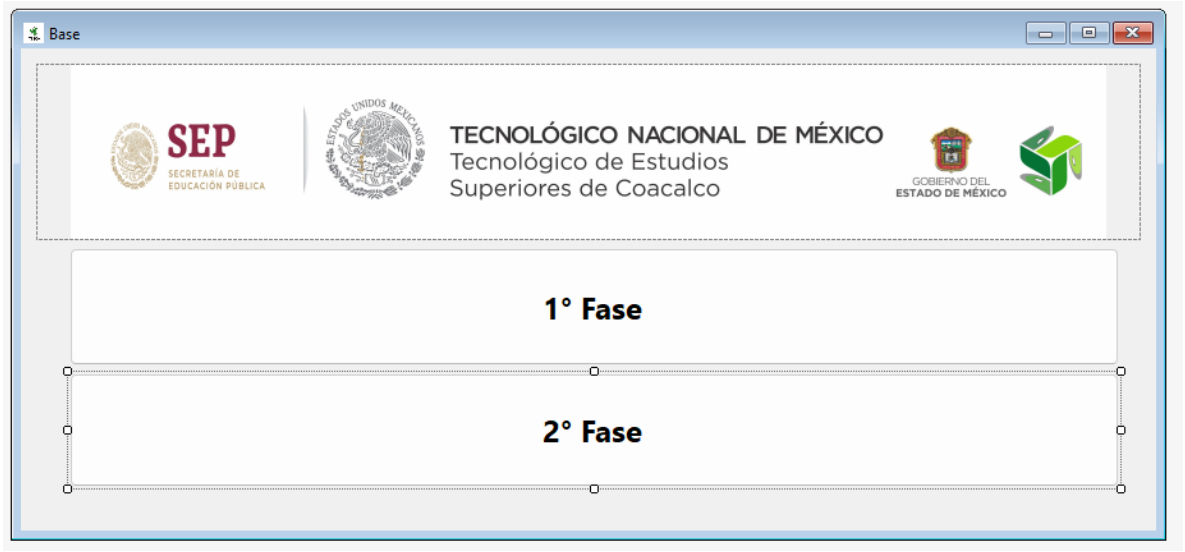

Ilustración 30: Diseño final de la interfaz de usuario (Ian Omar Madrigal Rueda)

Ya teniendo la base creada se añadió otros tres proyectos de Windows forms llamados Fase_1, Fase_2_1 y Fase_2_2 dando clic derecho en proyecto ya existente llamado Base localizado en el explorador de soluciones con la misma configuración de C#, dando doble clic en cualquier parte del formulario Base donde no existan elementos se desarrolló el siguiente código con la siguiente pantalla:

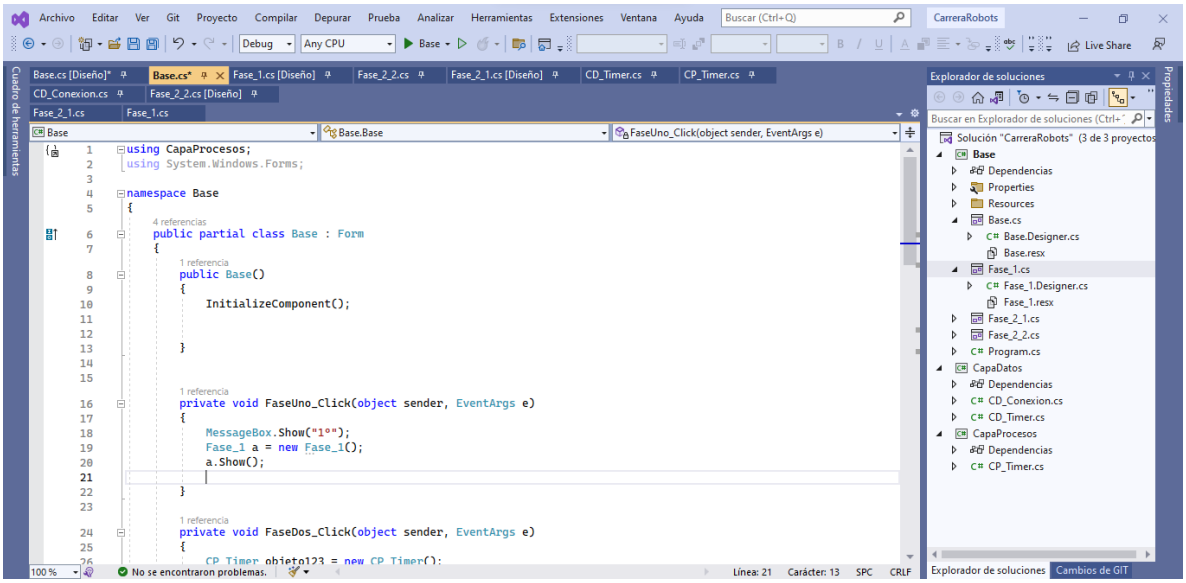

Ilustración 31: Código fuente de la base (Ian Omar Madrigal Rueda)

El código es una clase C# en Visual Studio que pertenece al espacio de nombres "Base". Esta clase representa una ventana de formulario que contiene dos eventos de clic asociados a botones ("FaseUno_Click" y "FaseDos_Click") y algunas funciones internas.

El primer bloque de código contiene las directivas "using" para importar los espacios de nombres necesarios en la aplicación:

1. `using CapaProcesos;`: Importa el espacio de nombres "CapaProcesos", que probablemente contenga algunas clases y funcionalidades relacionadas con el procesamiento de datos en la aplicación.

2. `using System.Windows.Forms;`: Importa el espacio de nombres "System.Windows.Forms", que proporciona las clases necesarias para trabajar con formularios y controles en una aplicación de Windows Forms.

Luego, se define una clase llamada "Base", que extiende la clase "Form" de Windows Forms. Esta clase representa la ventana principal de la aplicación.

Dentro de la clase "Base", hay un constructor público llamado "Base()", que es el constructor de la clase. Este constructor se ejecuta cuando se crea una instancia de la clase "Base". El constructor utiliza el método "InitializeComponent()" para inicializar los componentes del formulario. El método "InitializeComponent()" es generado automáticamente por el diseñador de formularios de Visual Studio y se encarga de configurar todos los controles y propiedades del formulario.

A continuación, la clase "Base" tiene dos métodos de evento:

1. `private void FaseUno_Click(object sender, EventArgs e)`: Este es el evento que se activa cuando el botón con el nombre "FaseUno" es clicado. Al hacer clic en el botón, se mostrará un mensaje emergente que dice "1°". Luego, se crea una instancia de la clase "Fase_1" (probablemente definida en el espacio de nombres "CapaProcesos") y se muestra esa instancia.

2. `private void FaseDos_Click(object sender, EventArgs e)`: Este es el evento que se activa cuando el botón con el nombre "FaseDos" es clicado. Al hacer clic en el botón, se ejecuta una serie de operaciones que involucran el uso de un objeto de tipo "CP_Timer" (probablemente definido en el espacio de nombres "CapaProcesos") y un control "DataGridView" llamado "dataGridView1".

Primero, se crea una instancia de la clase "CP_Timer" con el nombre "objeto123".

Luego, se establece la propiedad "DataSource" del control "dataGridView1" para mostrar los datos obtenidos llamando al método "ClasificarZ()" del objeto "objeto123".

A continuación, se calcula el valor de la variable "n" utilizando una serie de cálculos matemáticos. La variable "aux" se inicializa con el número de filas en el control "dataGridView1" menos uno. Luego, se itera un bucle while hasta que la variable "finalP" sea igual a cero. En cada iteración, se incrementa el valor de "n" y se recalcula "finalP" utilizando la fórmula proporcionada.

Después de calcular "n" y "final", se realiza una estructura de control "if-else" para decidir qué formulario mostrar a continuación. Si "final" es igual a 8, se muestra una instancia de la clase "Fase_2_2". Si "final" es mayor o igual a 16, se muestra una instancia de la clase "Fase_2_1".

En resumen, el código representa una ventana de formulario con dos botones. Al hacer clic en el botón "FaseUno", se muestra un mensaje emergente y se abre una nueva ventana llamada "Fase_1". Al hacer clic en el botón "FaseDos", se realizan ciertas operaciones con datos y se muestra una de las dos ventanas diferentes ("Fase_2_2" o "Fase_2_1") según el resultado de los cálculos realizados.

En la Fase_1 se buscó hacer una clasificación por tiempo de los competidores, para ello se creó un formulario que contenga un cronometro automatizado de la siguiente manera con componentes como DataGridView, textBox, CheckBox, Button y Timer.

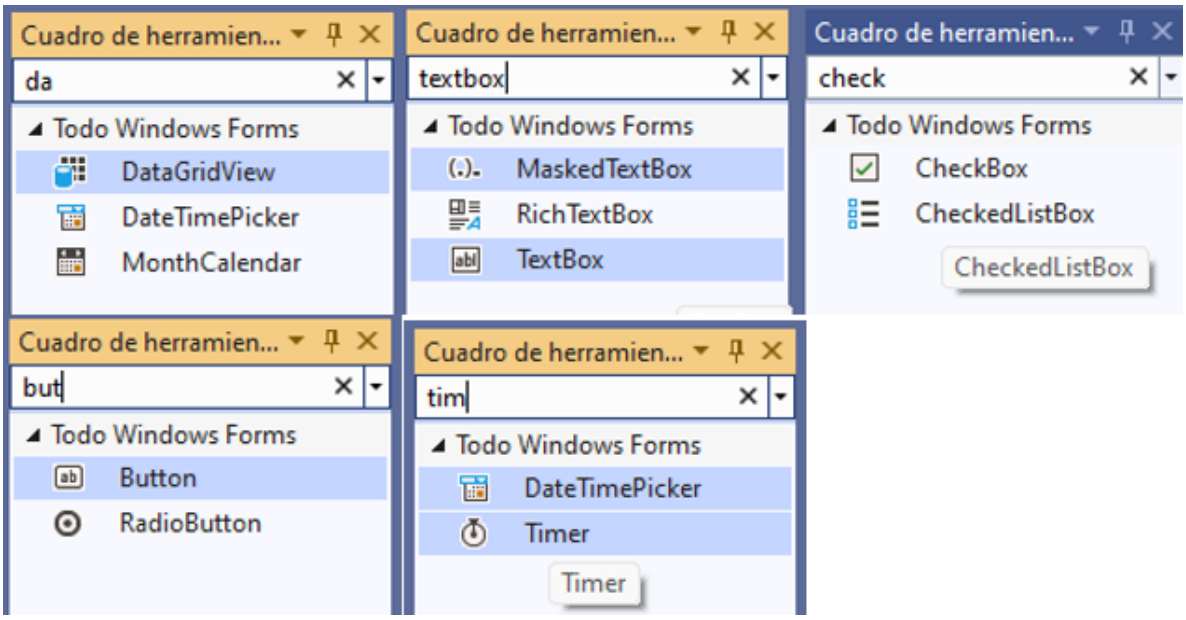

Ilustración 32: Herramientas necesarias para Fase_1 (Ian Omar Madrigal Rueda)

Como punto de partida se añadió un TextBox Dándole el formato de preferencia en la pestaña de propiedades quitando los bordes seleccionando “none” en BorderStyle, acomodando el estilo de fuente en Font, y haciendo “True” la opción de ReadOnly, de igual manera se corrige el “(name)” y el Text poniendo TextF y Digite el número de vueltas respectivamente.

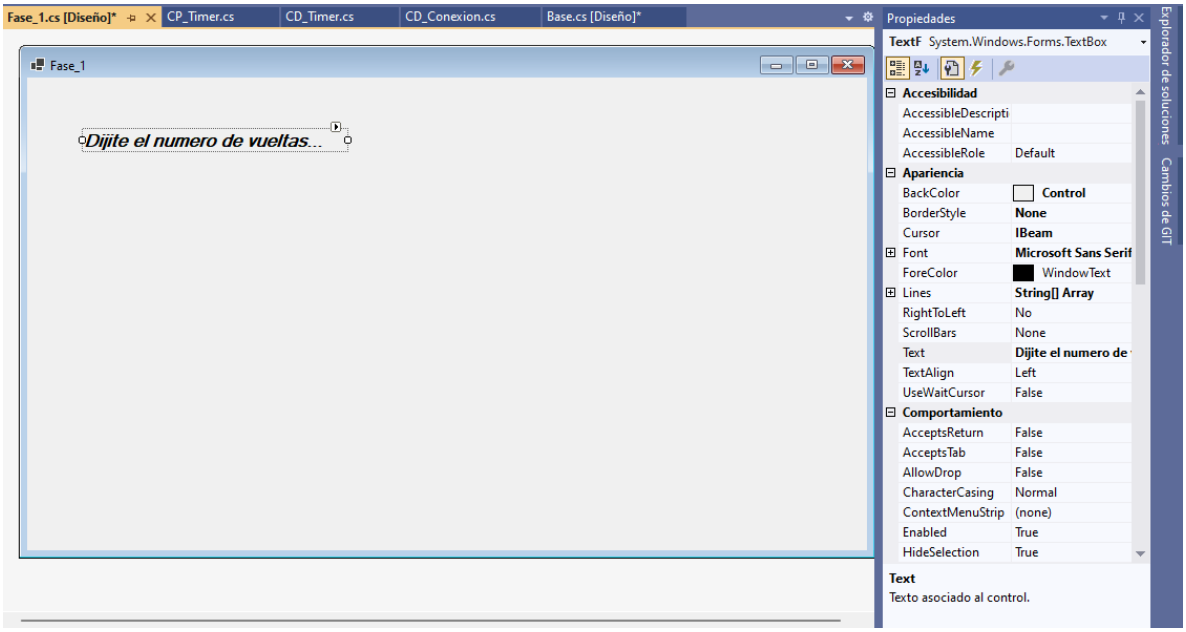

Ilustración 33: Fase_1 numero de vueltas (Ian Omar Madrigal Rueda)

Lo primero a desarrollar es el numero de vueltas que el usuario desea cronometrar.

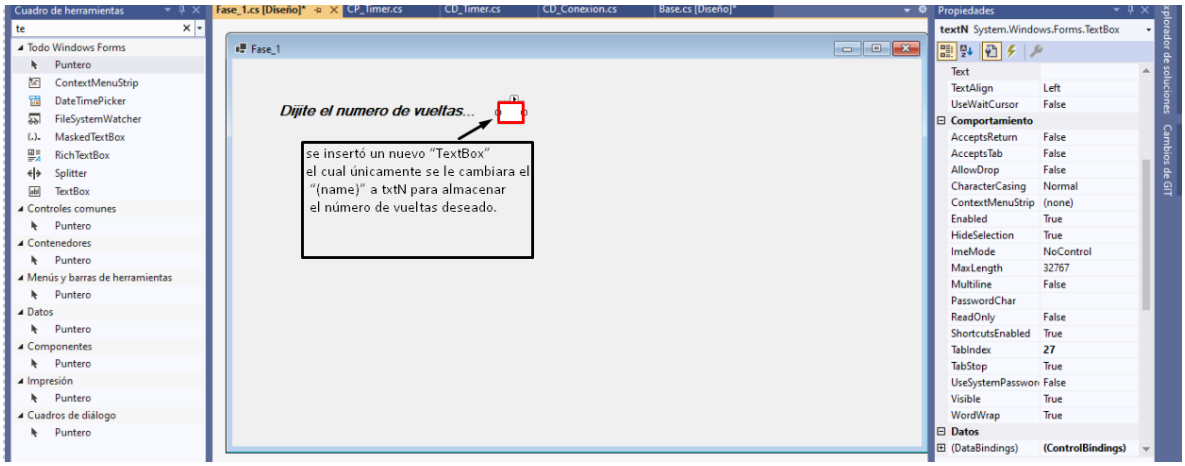

Ilustración 34: Interacción del usuario en el TextBox (Ian Omar madrigal Rueda)

Para posteriormente ver el tiempo ya en la parte grafica.

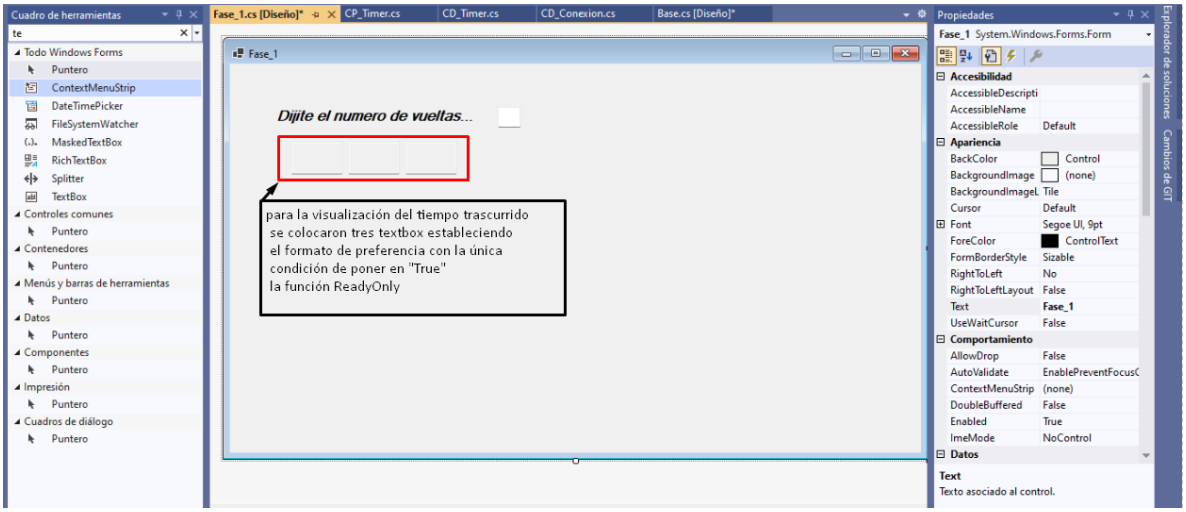

Ilustración 35: Visualización del tiempo (Ian Omar Madrigal Rueda)

Se agregan los botones básicos como son el reset y la clasificación de la información obtenida.

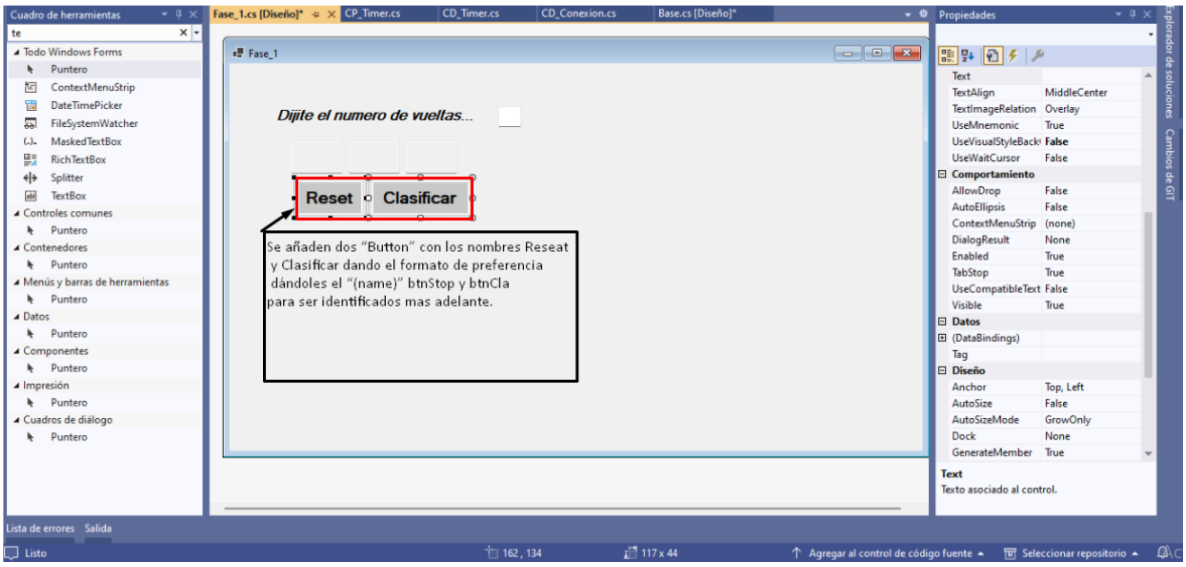

Ilustración 36: Diseño de Reset y Clasificar (Ian Omar Madrigal Rueda)

En un espacio mas se implementa la visualización de la carrera en curso.

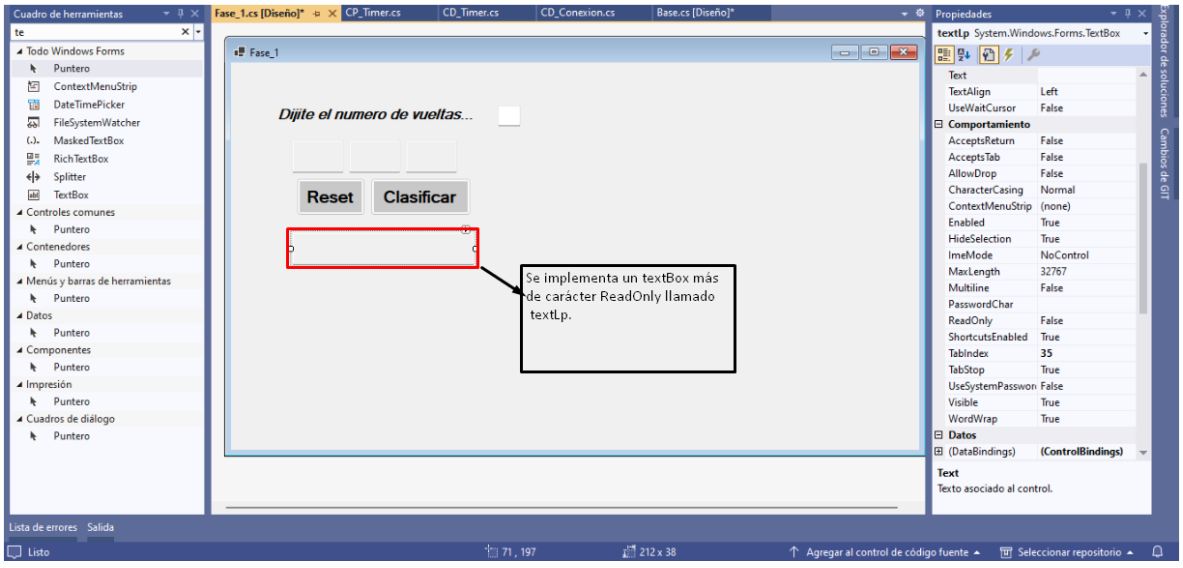

Ilustración 37: Configuración de un TextBox en ReadOnly (Ian Omar Madrigal Rueda)

Obteniendo los tiempos en un recuadro más.

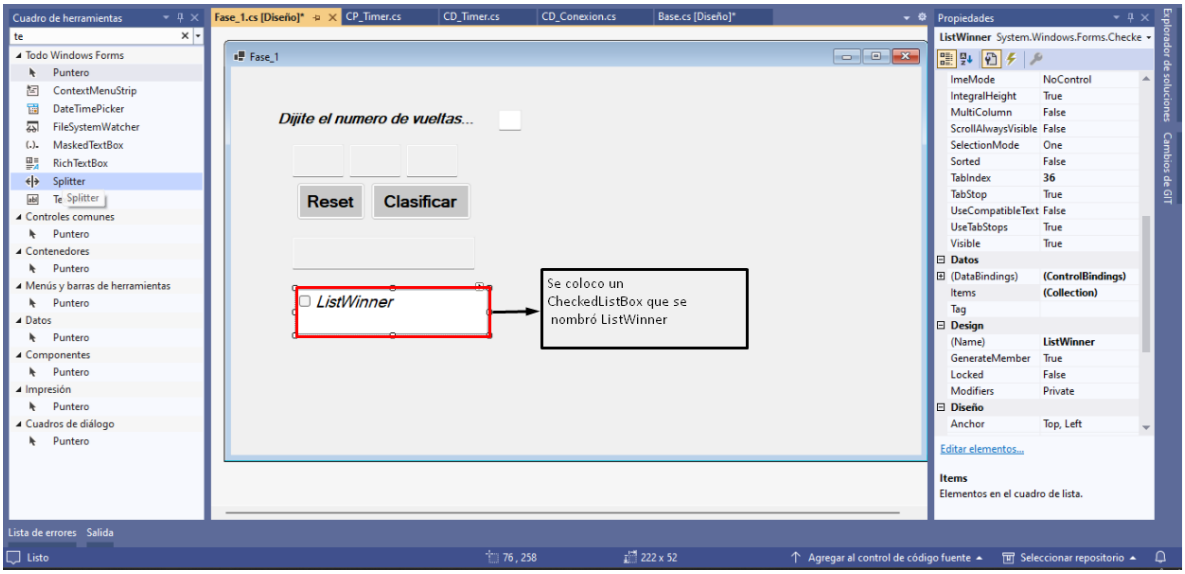

Ilustración 38: Visualización de las vueltas transcurridas (Ian Omar Madrigal Rueda)

Así mostrando la base de datos de SQL en un espacio sobrante.

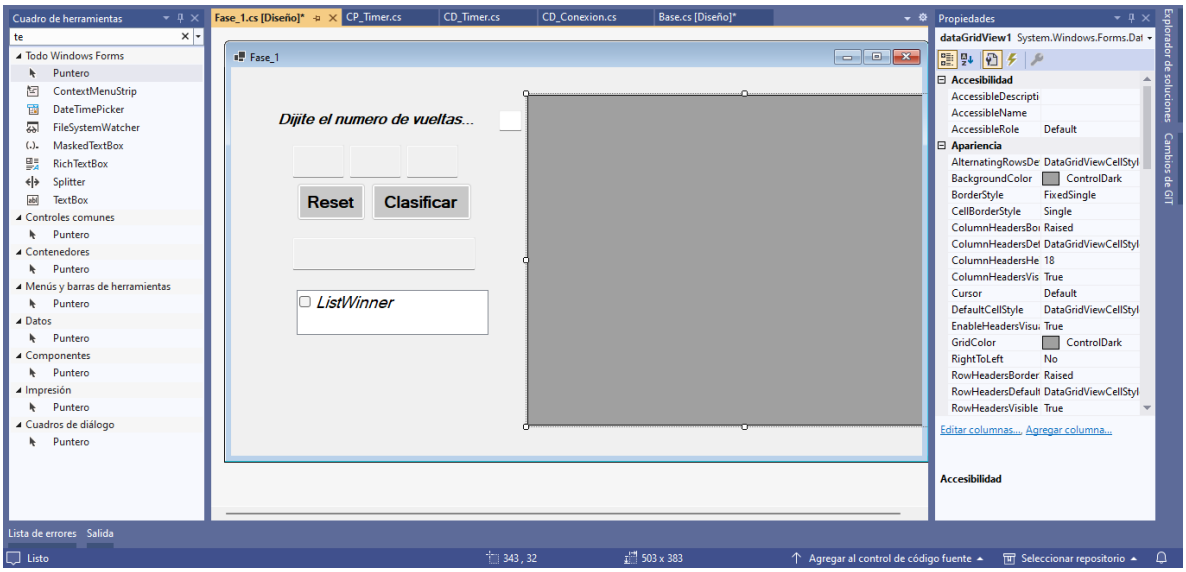

Ilustración 39: Muestreo de datos almacenados (Ian Omar Madrigal Rueda)

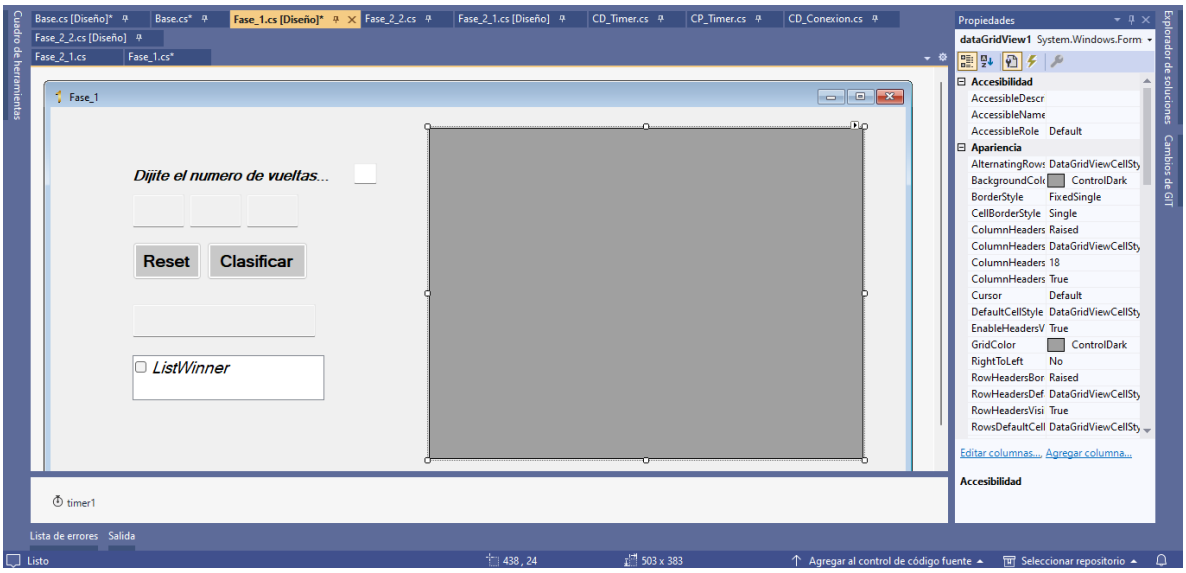

Ilustración 40: Manejo del timer1 (Ian Omar Madrigal Rueda)

3.7 Arduino

A partir de este punto se desarrolló la conexión del microcontrolador Arduino (Placa UNO) con el sensor de evitación de obstáculos por infrarrojos E18-D80NK comenzando con la conexión física mostrada a continuación donde nuestra terminal de color negro será la señal proporcionada al microcontrolador de manera digital, entendiendo el color azul como GND y el color café como 5v.

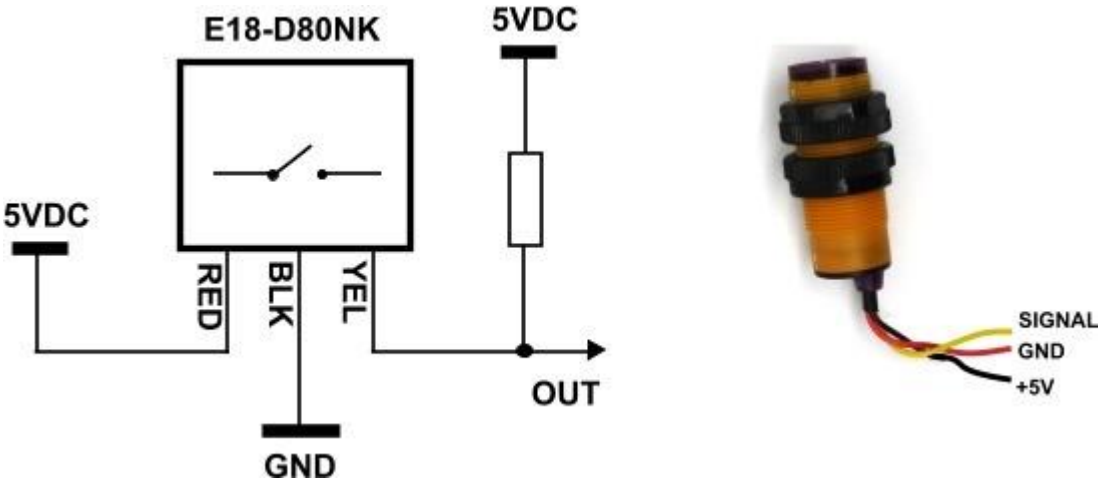

Ilustración 41: E18-D80N [21]

Ilustración 42: Conexión Arduino-E18-D80N [22]

Una vez hecha la conexión física se procedió a la programación de la propia placa implementando el siguiente código, para ello se establece la conexión de Arduino IDE con la placa seleccionando herramientas, puerto y el COM donde aparece la placa correspondiente.

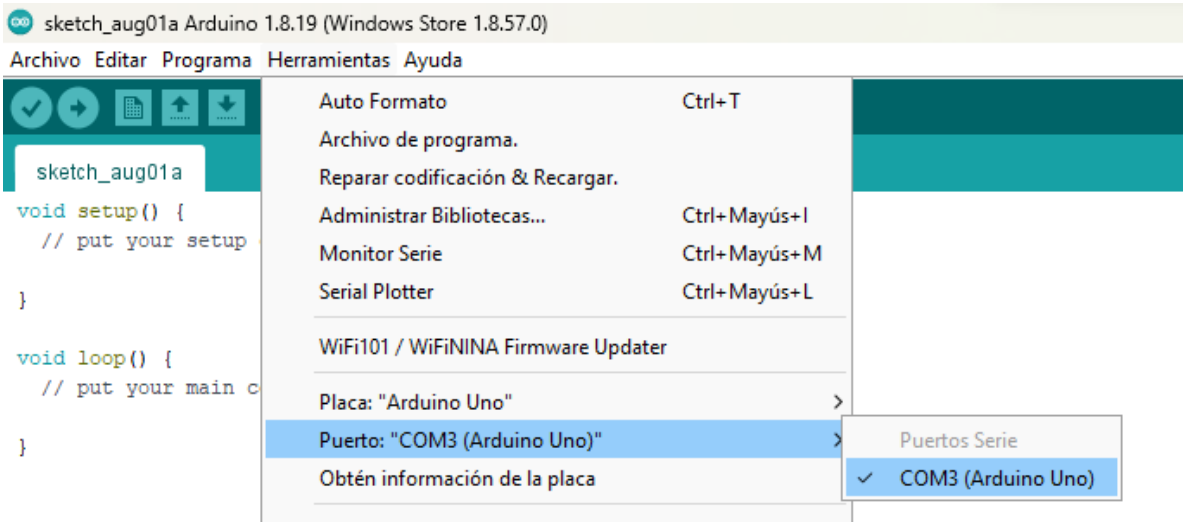

Ilustración 43: Conexión de Arduino con pc (Ian Omar Madrigal Rueda)

3.8 Comunicación serial entre Arduino - Visual Studio

```
void setup() {
  // put your setup code here, to run once:
  // Establece la velocidad de comunicación en 9600 baudios entre
Arduino y el puerto serie (computadora)
  Serial.begin(9600);
  // Establece el pin 8 como entrada, lo que significa que se leerá el
valor de este pin
  pinMode(8, INPUT);
  // Establece el pin 13 como salida, lo que significa que se enviará
una señal desde Arduino a través de este pin
  pinMode(13, OUTPUT);
}
void loop() {
  // put your main code here, to run repeatedly:
  // Si el valor leído del pin 8 es bajo (LOW)
  if (digitalRead(8) == LOW) {
    // Imprime el valor 1 en el puerto serie (computadora)
    Serial.println(1);
  }
  // Si el valor leído del pin 8 es alto (HIGH)
  if (digitalRead(8) == HIGH) {
    // Imprime el valor 0 en el puerto serie (computadora)
    Serial.println(0);
  }
  // Aguarda 150 milisegundos antes de continuar con la siguiente
iteración del bucle
  delay(150);
}
```

Ilustración 44: Almacenamiento del código en Arduino (Ian Omar Madrigal Rueda)

3.9 Integración en la programación en C# con Arduino

Regresando al formulario Fase_1 dando clic en el mismo desarrollamos el siguiente código; esta clase contiene varios métodos y variables para llevar a cabo diferentes tareas. A continuación, se explicará cada línea de código y su función:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Diagnostics;
using System.Data.SqlClient;
using CapaProcesos;
using System.Timers;
using System.Security.Cryptography.X509Certificates;
```

Estas son las directivas "using" que permiten acceder a diferentes bibliotecas y espacios de nombres para utilizar sus clases y funciones en el código.

```
namespace Base
{
    public partial class Fase_1 : Form
    {
```

Aquí se declara la clase `Fase_1` que hereda de `Form`, lo que significa que representa una ventana de formulario en la interfaz de usuario.

```
public int x;
```

Se declara una variable pública llamada `x` de tipo entero. La variable está disponible para acceder desde fuera de la clase.

```
CP_Timer objetoCN = new CP_Timer();
```

Se crea una instancia de la clase `CP_Timer` y se asigna a la variable `objetoCN`. Esta clase parece estar definida en un espacio de nombres llamado "CapaProcesos".

```
int conteo = 0;
```

Se declara una variable `conteo` de tipo entero e inicializada en 0.

```
Stopwatch oSW = new Stopwatch();
```

Se declara una instancia de la clase `Stopwatch` llamada `oSW`, que se utilizará para medir el tiempo transcurrido.

```
List<string> vs = new List<string>();
```

Se declara una lista genérica llamada `vs` que almacenará cadenas (strings).

```
System.IO.Ports.SerialPort Port;
```

Se declara una instancia de la clase `SerialPort` llamada `Port`, que se utilizará para comunicarse con un puerto serie.

```
bool IsClosed = false;
```

Se declara una variable booleana llamada `IsClosed` y se inicializa en `false`.

```
public Fase_1()
{
InitializeComponent();
```

El constructor de la clase `Fase_1`. Aquí se inicializan los componentes del formulario llamando al método `InitializeComponent()`.

```
Port = new System.IO.Ports.SerialPort();
Port.PortName = "COM3";
Port.BaudRate = 9600;
Port.ReadTimeout = 500;
```

Se inicializa la instancia de `SerialPort` llamada `Port`. Se configura el nombre del puerto como "COM3", la velocidad de transmisión en 9600 baudios y el tiempo de espera de lectura en 500 milisegundos.

```
try
{
Port.Open();
}
catch (Exception ex)
{
}
```

Se intenta abrir el puerto serie con `Port.Open()`. Si ocurre algún error, se captura la excepción, pero no se realiza ninguna acción.

Hasta aquí, termina el constructor de la clase.

Continuando con los métodos de la clase `Fase_1`:

```
private void Clasificar()
{
CP_Timer objeto12 = new CP_Timer();
dataGridView1.DataSource = objeto12.Clasificar();
}
```

El método `Clasificar()` crea una nueva instancia de `CP_Timer` llamada `objeto12` y asigna el resultado de su método `Clasificar()` como origen de datos para un control DataGridView llamado `dataGridView1`.

```
private void ClasificarZ()
{
CP_Timer objeto123 = new CP_Timer();
dataGridView1.DataSource = objeto123.ClasificarZ();
}
```

El método `ClasificarZ()` crea una nueva instancia de `CP_Timer` llamada `objeto123` y asigna el resultado de su método `ClasificarZ()` como origen de datos para el control `dataGridView1`.

```
private void EscucharSerial()
{
```

El método `EscucharSerial()` es donde ocurre el ciclo principal de escucha del puerto serie y la gestión de los datos recibidos.

```
StringComparer stringComparer = StringComparer.OrdinalIgnoreCase;
```

Se crea un objeto de tipo `StringComparer` para realizar comparaciones de cadenas sin distinción entre mayúsculas y minúsculas.

```
while (!IsClosed)
{
```

Comienza un bucle `while` que se ejecutará siempre que `IsClosed` sea `false`.

```
try
{
string cadena = Port.ReadLine();
```

Se lee una línea (hasta el salto de línea) del puerto serie y se almacena en una variable llamada `cadena`.

```
if (int.Parse(cadena) == 1) {
if (textN.Text.Length == 0) {
MessageBox.Show("Dijita un numero de vueltas.");
}
else
{
conteo = conteo + int.Parse(cadena);
Console.WriteLine(conteo);
int aux = int.Parse(textN.Text) + 2;
```

Si el valor numérico de `cadena` es igual a 1, se ejecutan las siguientes acciones. Se verifica si el campo `textN` está vacío; si es así, se muestra un mensaje emergente que solicita ingresar un número de vueltas.

```
if (conteo == 1)
{
oSW.Start();
timer1.Enabled = true;
Console.WriteLine("Hi");
}
```

Si el `conteo` es igual a 1, se inicia el cronómetro (`oSW.Start()`) y se habilita un temporizador (`timer1.Enabled = true`).

```
if (conteo > 1 && conteo < aux)
{
Console.WriteLine("Low");
textN.Invoke(new MethodInvoker(
delegate
{
textLp.Text = textMin.Text + ":" + textSeg.Text + ":" + textMil.Text;
vs.Add(textLp.Text);
ListWinner.Items.Clear();
for (int i = 0; i <= vs.Count - 1; i++)
{
ListWinner.Items.Add(vs.ElementAt(i));
}
}));
}
```

Si `conteo` es mayor que 1 y menor que `aux` (calculado previamente), se imprime "Low" en la consola. Luego, se actualiza el campo `textLp` con el tiempo actual formateado y se agrega a la lista `vs`. A continuación, se actualiza el contenido del control `ListWinner` con los elementos de la lista `vs`.

```
if (conteo == aux - 1)
{
oSW.Stop();
try
{
objetoCN.Insertar(textMin.Text + ":" + textSeg.Text + ":" +
textMil.Text);
MessageBox.Show("Se inserto correctamente.");
}
catch (Exception ex)
{
MessageBox.Show("No se pudo insertar el tiempo. ");
}
}
```

Si `conteo` es igual a `aux - 1`, se detiene el cronómetro (`oSW.Stop()`) y se intenta insertar el tiempo actual en una base de datos utilizando el método `Insertar()` de `objetoCN`. Si hay algún error, se muestra un mensaje de error en un cuadro de diálogo emergente.

```
}
}
}
catch { }
```

Se captura cualquier excepción que ocurra al intentar convertir `cadena` a un número entero.

```
    }
}
```

Aquí termina el bucle `while` y el método `EscucharSerial()`.

```
private void timer1_Tick_1(object sender, EventArgs e)
{
TimeSpan ts = new TimeSpan(0, 0, 0, 0, (int)oSW.ElapsedMilliseconds);
textMin.Text = ts.Minutes.ToString().Length < 2 ? "0" +
ts.Minutes.ToString() : ts.Minutes.ToString();
textSeg.Text = ts.Seconds.ToString().Length < 2 ? "0" +
ts.Seconds.ToString() : ts.Seconds.ToString();
textMil.Text = ts.Milliseconds.ToString();
}
```

Este método se activa cada vez que ocurre el evento `timer1_Tick`, lo que significa que se ejecuta periódicamente mientras el temporizador esté habilitado. El método actualiza los campos de texto `textMin`, `textSeg` y `textMil` con el tiempo transcurrido obtenido del cronómetro (`oSW.ElapsedMilliseconds`).

```
private void Stop_Click_1(object sender, EventArgs e)
{
oSW.Reset();
textMin.Text = "00";
textSeg.Text = "00";
textMil.Text = "000";
textLp.Text = "                ";
ListWinner.Items.Clear();
vs.Clear();
timer1.Enabled = true;
conteo = 0;
textN.Clear();
textN.ReadOnly = false;
dataGridView1.ClearSelection();
Clasificar();
}
```

Este método se ejecuta cuando se hace clic en el botón "Stop". El método restablece el cronómetro, establece algunos campos de texto y listas a sus valores predeterminados y llama al método `Clasificar ()` para actualizar los datos en el control `dataGridView1`.

```
private void BtnCla_Click(object sender, EventArgs e)
{
int n = 1;
double aux = dataGridView1.RowCount - 1;
int finalP = 1;
while (finalP != 0)
{
n++;
finalP = (int)Math.Round(aux / Math.Pow(2, n), 1);
}
n--;
int final = (int)Math.Round(Math.Pow(2, n));
MessageBox.Show("Seleccionados " + final + " participantes");
ClasificarZ();
x = final;
Fase_1 fase_1 = new Fase_1();
fase_1.Close();
}
```

Este método se ejecuta cuando se hace clic en el botón "BtnCla". Calcula el número de participantes seleccionados y muestra un mensaje emergente con la cantidad. Luego llama al método `ClasificarZ()` para actualizar los datos en el control `dataGridView1`, asigna el valor calculado de `final` a la variable pública `x` y cierra la ventana actual de la fase 1 creando una nueva instancia de `Fase_1` y cerrándola.

```
private void Fase_1_Load(object sender, EventArgs e)
{
Thread Hilo = new Thread(EscucharSerial);
Hilo.Start();
        }
    }
}
```

Este método se ejecuta cuando se carga el formulario. Crea un nuevo hilo (`Thread`) que ejecutará el método `EscucharSerial()`, el cual es responsable de escuchar y procesar los datos del puerto serie.

En resumen, el código implementa una aplicación de Windows que interactúa con un dispositivo conectado al puerto serie (COM3) para realizar un seguimiento del tiempo de vuelta de los participantes. La aplicación muestra los datos en un formulario y permite clasificar y guardar los tiempos en una base de datos. Además, se muestra una lista de los tiempos de los participantes, y se pueden seleccionar un número específico de participantes para una siguiente fase.

3.10 Desarrollo de fases

El siguiente paso para seguir es desarrollar Fase_2_1 y Fase_2_2 respectivamente elaborando un cuadro de clasificatorias de 8 y 16 participantes.

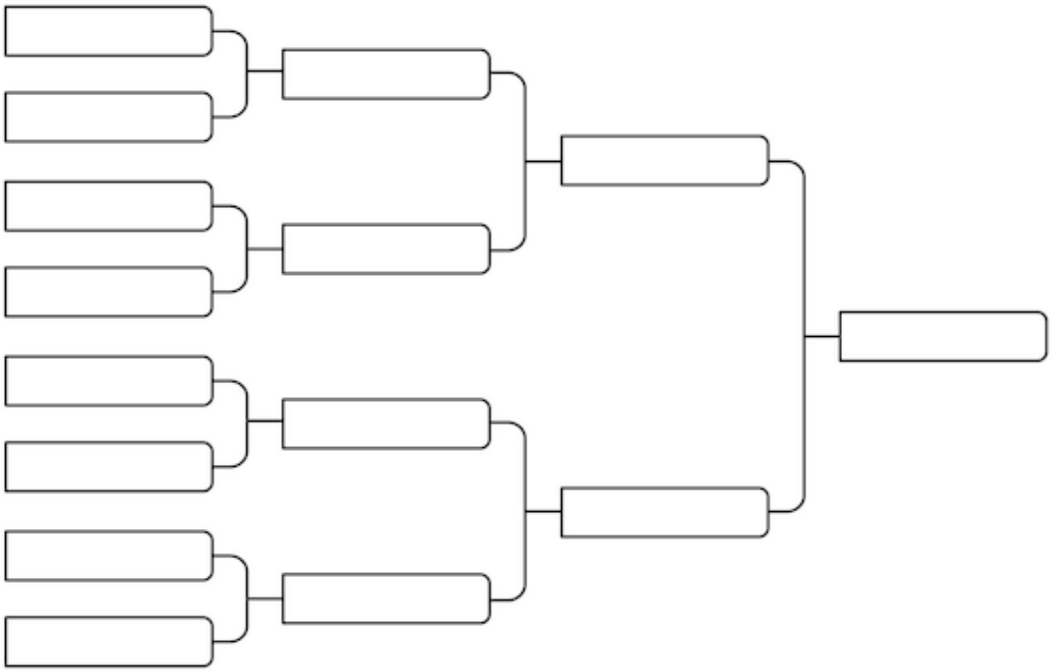

Ilustración 45: Esquema de selección del ganador [23]

Para ello se usará un CheckBox junto con sus respectivos textBox que tendrá el número del participante seleccionado y su contrincante en cuestión, el cuadro de clasificación tendrá un llenado automatizado expresado en el siguiente Código.

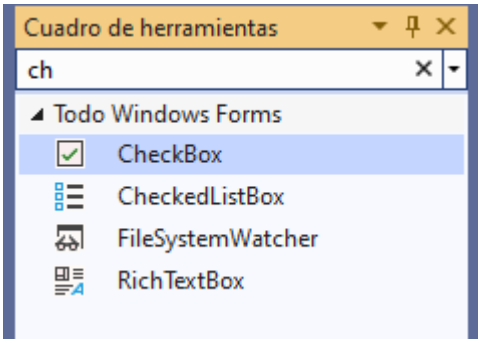

Ilustración 46: CheckBox en Windows form (Ian Omar Madrigal Rueda)

Se mantiene la idea de una buena distribución de la clasificación de los jugadores para una mejor lectura por parte del usuario.

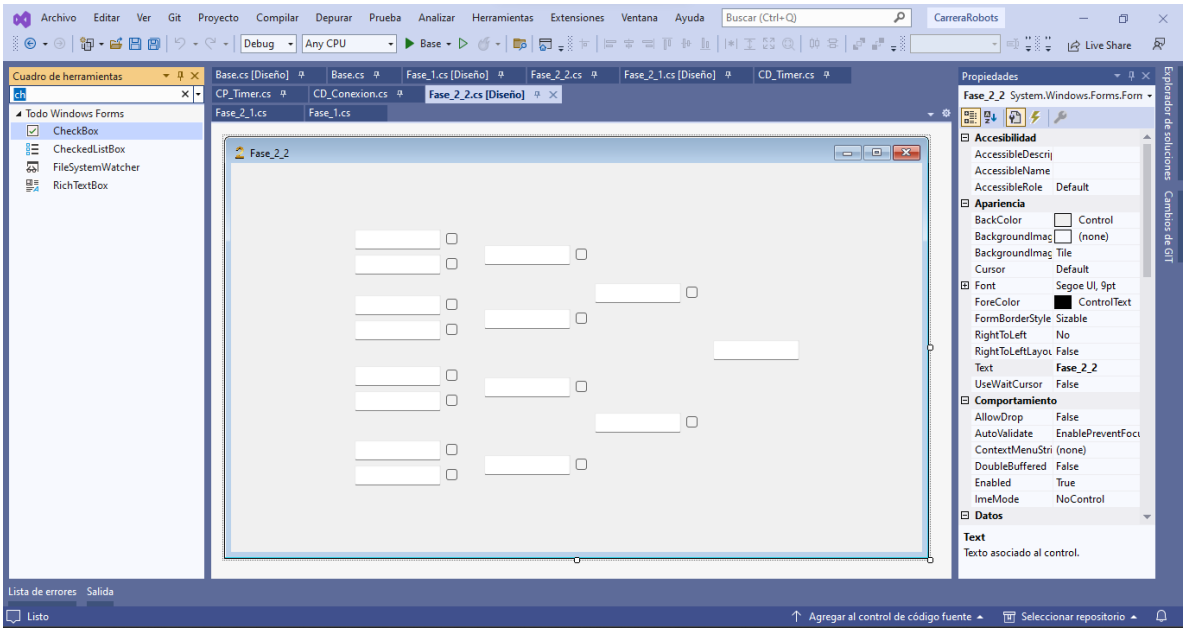

Ilustración 47: Esquema chequeo de ganadores 8 jugadores (Ian Omar Madrigal Rueda)

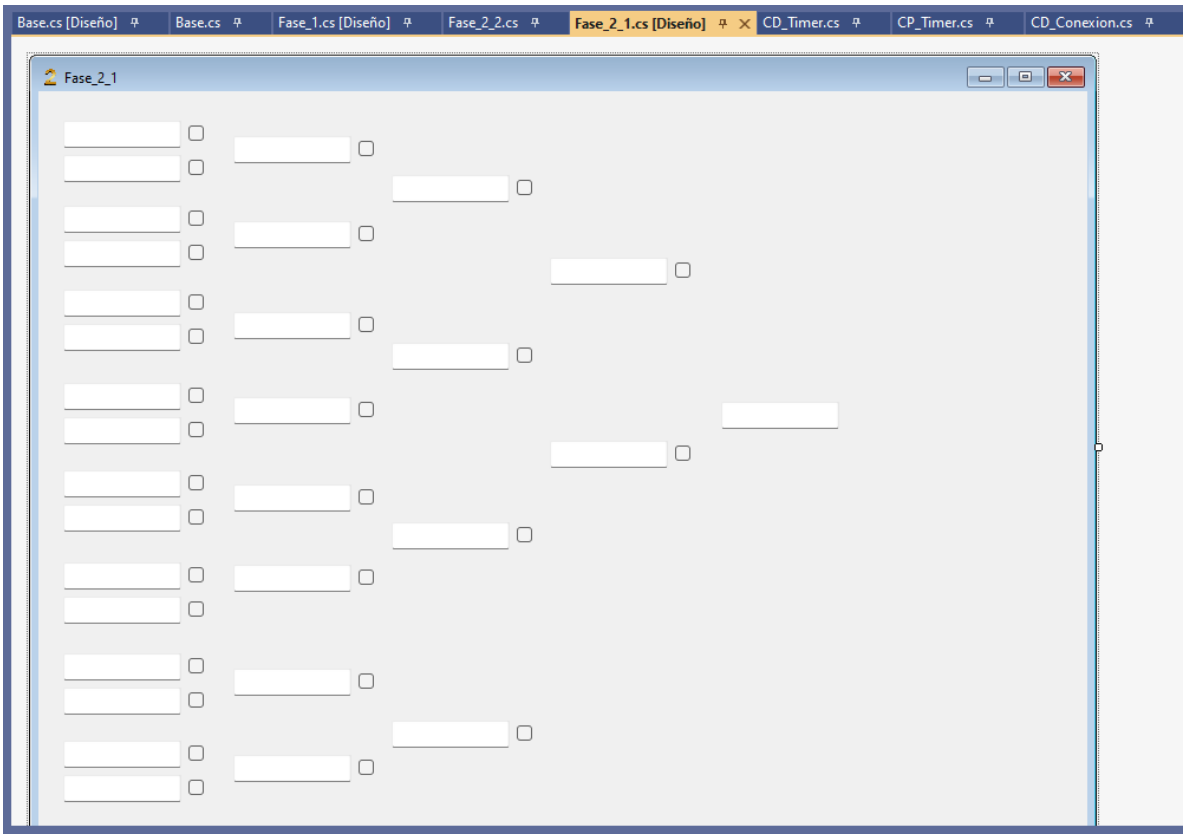

Ilustración 48: Esquema chequeo de ganadores 16 jugadores (Ian Omar Madrigal Rueda)

El código proporcionado es un formulario en C# creado en Visual Studio. El formulario se llama "Fase_2_2" y contiene varias casillas de verificación y cajas de texto donde se muestran resultados y se pueden realizar selecciones.

La clase `Fase_2_2` se encuentra dentro del namespace "Base". Para su correcto funcionamiento, el código importa algunas bibliotecas usando la declaración "using". A continuación, se explicará cada línea del código en tercera persona:

1. `using CapaProcesos;`: Importa el espacio de nombres "CapaProcesos", que probablemente contenga definiciones de clases y métodos relacionados con la lógica de negocio o procesamiento de datos de la aplicación.

2. `using System;`: Importa el espacio de nombres "System", que contiene tipos y funcionalidades fundamentales proporcionados por el .NET Framework.

3. `using System.Collections.Generic;`: Importa el espacio de nombres "System.Collections.Generic", que contiene tipos y clases genéricas para trabajar con colecciones.

4. `using System.ComponentModel;`: Importa el espacio de nombres "System.ComponentModel", que contiene clases e interfaces para extender el modelo de componentes y la implementación de componentes.

5. `using System.Data;`: Importa el espacio de nombres "System.Data", que contiene clases que admiten el acceso a datos y la manipulación de datos en una base de datos.

6. `using System.Drawing;`: Importa el espacio de nombres "System.Drawing", que proporciona clases para dibujar gráficos y trabajar con imágenes.

7. `using System.Linq;`: Importa el espacio de nombres "System.Linq", que proporciona extensiones para consultas LINQ (Language Integrated Query) en colecciones.

8. `using System.Text;`: Importa el espacio de nombres "System.Text", que contiene clases para trabajar con cadenas de caracteres y codificaciones de caracteres.

9. `using System.Threading.Tasks;`: Importa el espacio de nombres "System.Threading.Tasks", que proporciona clases para admitir la programación basada en tareas.

10. `using System.Windows.Forms;`: Importa el espacio de nombres "System.Windows.Forms", que contiene clases para crear aplicaciones de Windows Forms y componentes visuales.

El formulario "Fase_2_2" tiene una serie de casillas de verificación (`CheckBox`) y cajas de texto (`TextBox`) en los que se muestra información y se realizan operaciones. Cuando se carga el formulario, se ejecuta el evento `Fase_2_2_Load`. En este evento, se crea una instancia de la clase `CP_Timer` y se usa para realizar algunas operaciones en las cajas de texto.

Luego, hay varios eventos asociados con las casillas de verificación, como `checkBox1_CheckedChanged`, `checkBox2_CheckedChanged`, `checkBox3_CheckedChanged`, etc. Estos eventos se activan cuando el estado de las casillas de verificación cambia (seleccionado/deseleccionado). En cada evento, se realizan algunas operaciones en las cajas de texto dependiendo del estado de las casillas de verificación seleccionadas y deshabilitando otras casillas según el flujo lógico del programa.

Finalmente, hay dos eventos, `checkBox13_CheckedChanged` y `checkBox14_CheckedChanged`, que muestran un mensaje emergente (MessageBox) indicando el "Ganador de la competencia" al seleccionar una casilla de verificación específica.

En resumen, el código crea un formulario con casillas de verificación y cajas de texto que interactúan entre sí según el estado de las casillas de verificación seleccionadas y deshabilitadas. Al seleccionar determinadas casillas de verificación, se mostrarán los resultados en las cajas de texto correspondientes y se presentará un mensaje emergente cuando se seleccione una casilla de verificación específica. La funcionalidad exacta y el propósito más amplio del formulario dependen de la lógica de negocio que esté implementada en la clase `CP_Timer` y cómo se utilizan los datos en las casillas de texto.

Base.cs [Diseño] | Base.cs | Fase_1.cs [Diseño] | Fase_2_2.cs | Fase_2_1.cs [Diseño] | CD_Timer.cs

Base | Base.Fase_2_2

```
12  namespace Base
13  {
        5 referencias
14      public partial class Fase_2_2 : Form
15      {
            1 referencia
16          public Fase_2_2()
17          {
18              InitializeComponent();
19          }
20
            1 referencia
21          private void Fase_2_2_Load(object sender, EventArgs e)
22          {
23              CP_Timer think = new CP_Timer();
24              textBox1.Text = think.ClasificarZ().Rows[0][0].ToString();
25              textBox2.Text = think.ClasificarZ().Rows[1][0].ToString();
26              textBox3.Text = think.ClasificarZ().Rows[2][0].ToString();
27              textBox4.Text = think.ClasificarZ().Rows[3][0].ToString();
28              textBox5.Text = think.ClasificarZ().Rows[4][0].ToString();
29              textBox6.Text = think.ClasificarZ().Rows[5][0].ToString();
30              textBox7.Text = think.ClasificarZ().Rows[6][0].ToString();
31              textBox8.Text = think.ClasificarZ().Rows[7][0].ToString();
32          }
            1 referencia
33          private void checkBox1_CheckedChanged(object sender, EventArgs e)
34          {
35              if (checkBox1.Checked == true)
36              {
37                  textBox9.Text = textBox1.Text;
38                  checkBox2.Enabled = false;
39
40              }
41          }
```

Ilustración 49: Código de Fase_2 8 jugadores (Ian Omar Madrigal Rueda)

Del mismo modo se repite el proceso para los 16 participantes

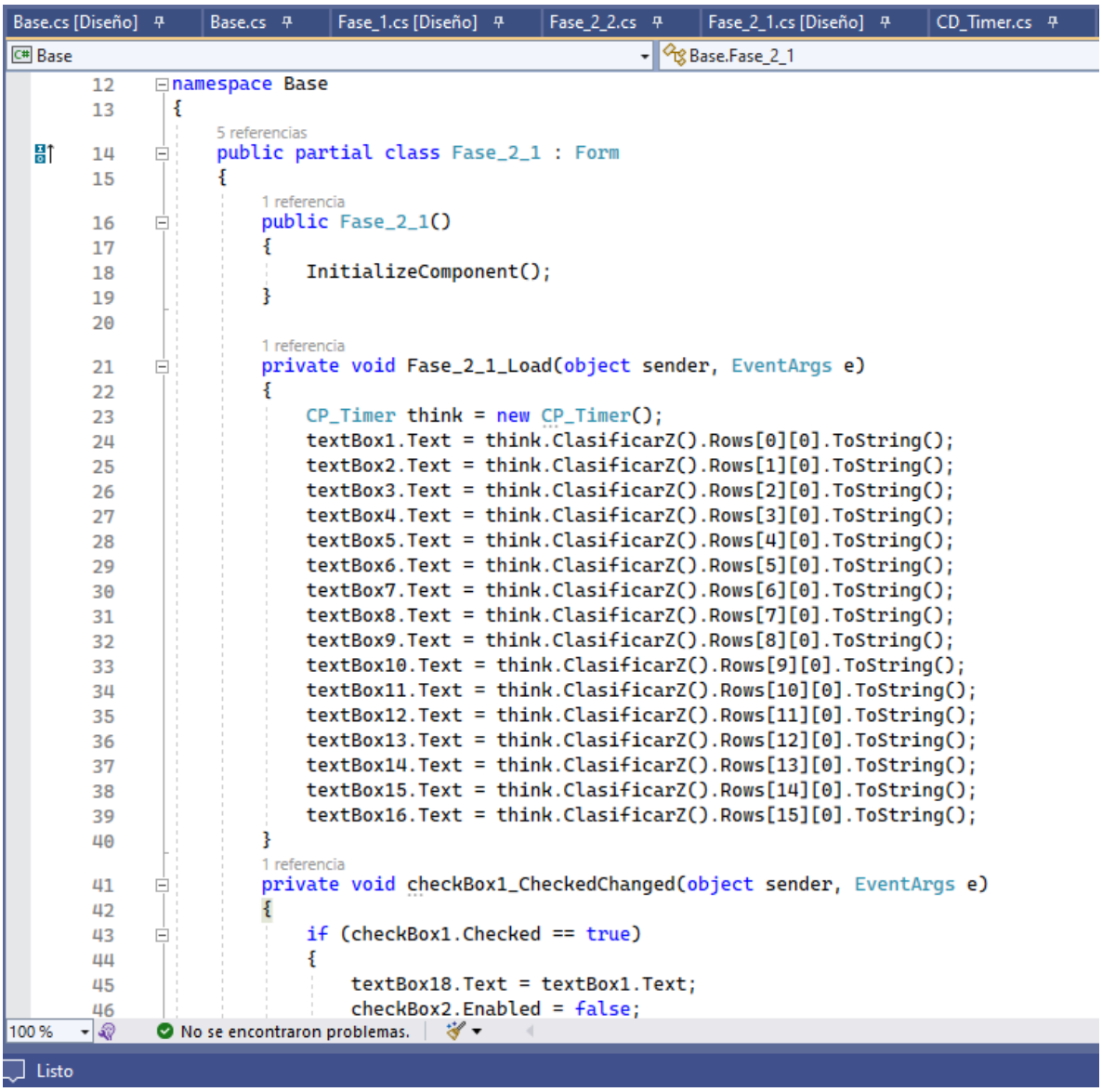

Ilustración 50: Código de Fase_2 16 jugadores (Ian Omar Madrigal Rueda)

3.11 Ejecución

La ejecución del proyecto debe ser iniciada desde el proyecto base nombrado en un principio.

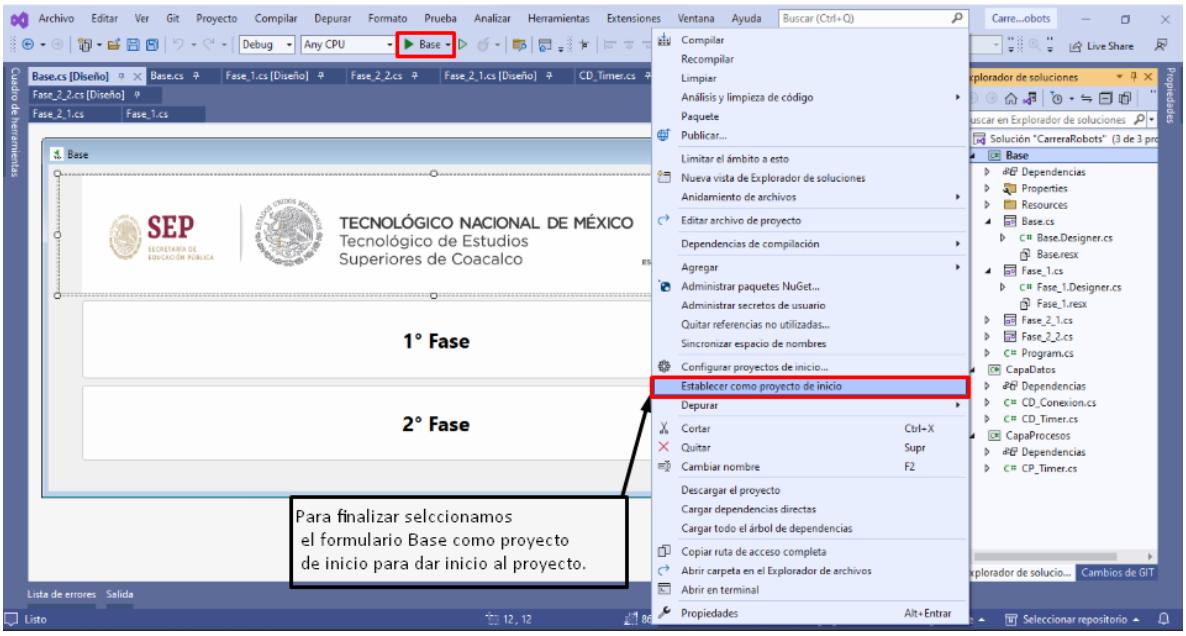

Ilustración 51: Base como proyecto de inicio (Ian Omar Madrigal Rueda)

Capítulo IV.
Resultados

Implementación del sistema: La implementación fue un reto emocionante. Programar el Arduino y trabajar con Visual Studio para desarrollar la interfaz fue una experiencia enriquecedora. Ver cómo mi código cobra vida y controla las carreras de los Robots sigue líneas es realmente gratificante.

Velocidad y eficiencia en la competencia: La aplicación ha mejorado significativamente la velocidad y eficiencia del robot sigue líneas durante las competencias. La gestión automatizada de las competencias permite una toma de decisiones rápida y precisa, lo que se traduce en una mejora en el rendimiento general del robot en comparación con métodos manuales.

Interfaz de usuario amigable: La interfaz desarrollada en Visual Studio proporciona una experiencia de usuario amigable para el control y la gestión del robot sigue líneas. Los comandos y ajustes son más fáciles de configurar y cambiar, lo que facilita su uso por parte del operador.

Simplicidad en la configuración: La programación y configuración de la aplicación han sido diseñadas para ser lo más simples y comprensibles posible, permitiendo que incluso estudiantes y principiantes en robótica puedan utilizarla con facilidad.

Comparativa antes y después del proyecto: Ver los resultados de la comparativa me hace sentir orgulloso del trabajo realizado. La automatización ha demostrado ser mucho más eficiente que los métodos manuales previos, y eso me motiva a seguir explorando nuevas soluciones tecnológicas.

Ilustración 52: Resultado de primera interfaz (Ian Omar Madrigal rueda)

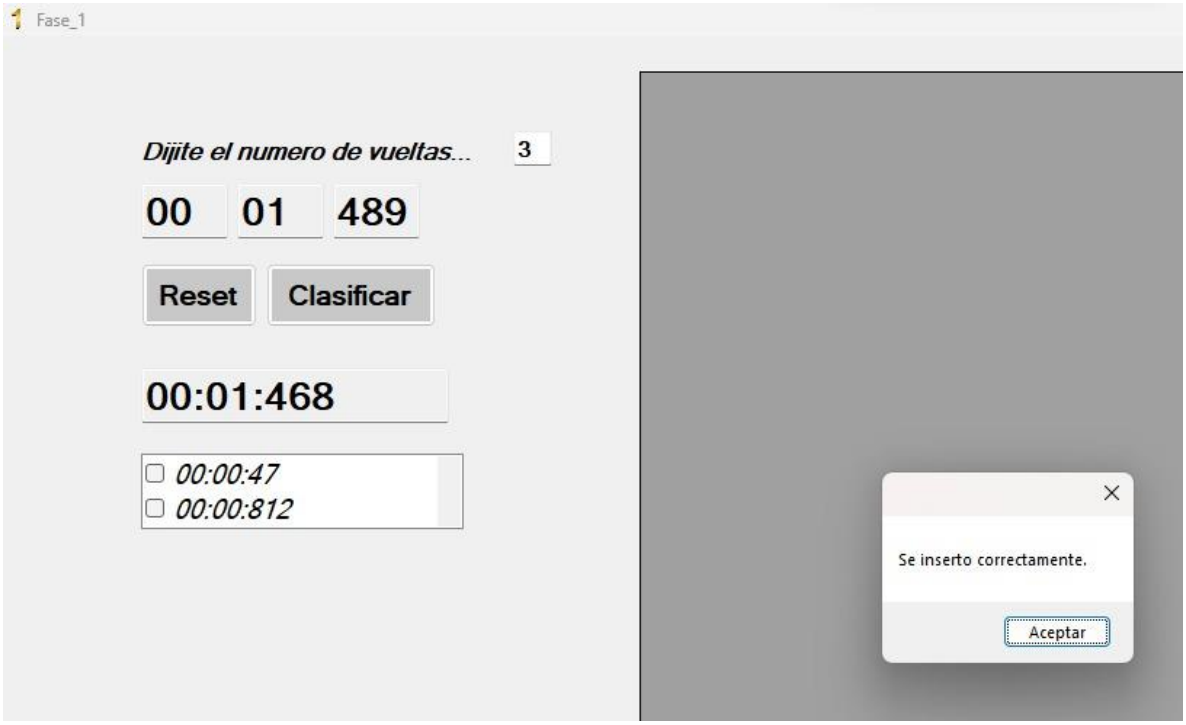

Ilustración 53: Resultado de cronometro (Ian Omar Madrigal Rueda)

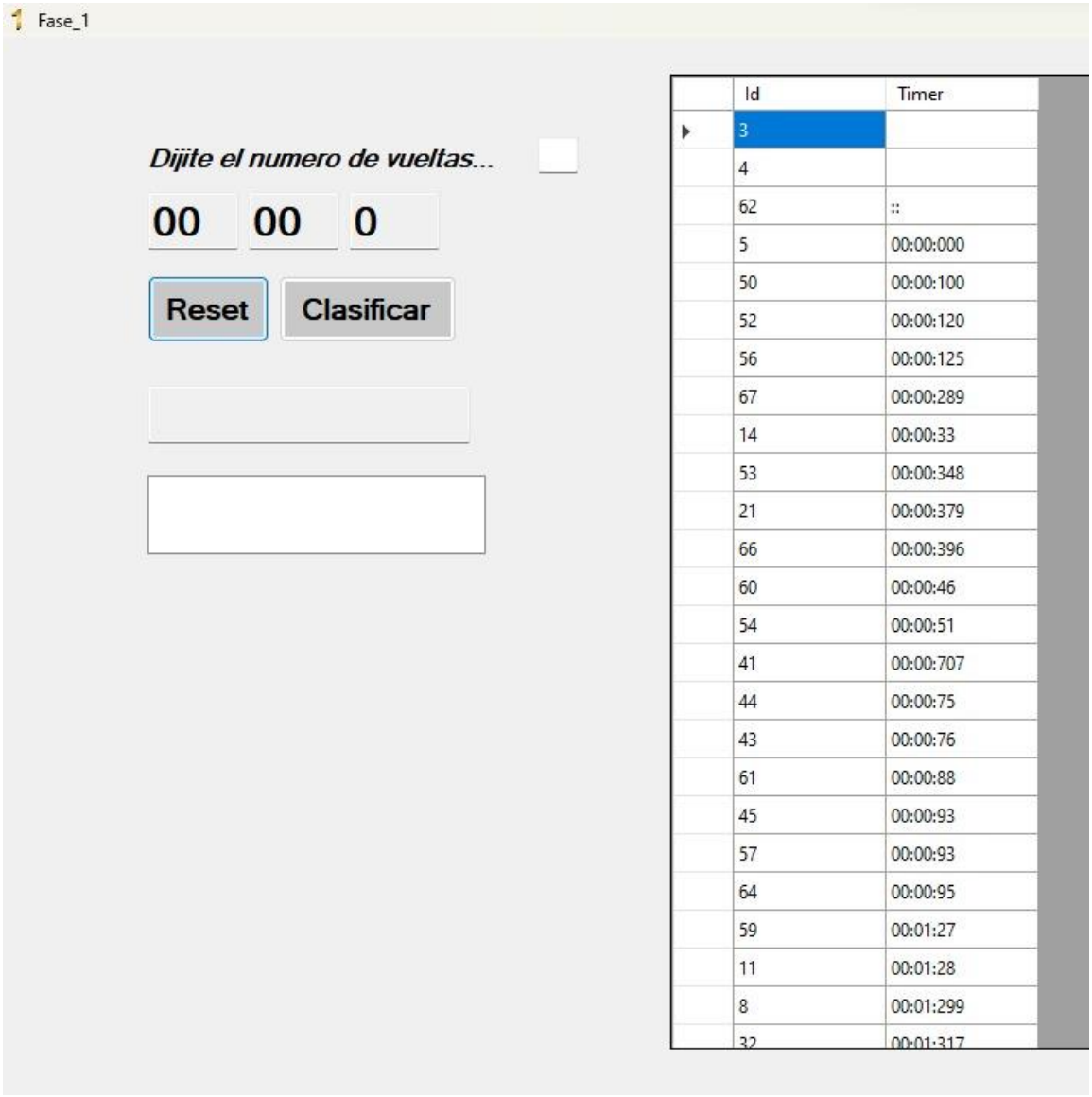

Ilustración 54: Lista de jugadores en orden (Ian Omar Madrigal Rueda)

Ilustración 55: Espina de ganadores (Ian Omar Madrigal Rueda)

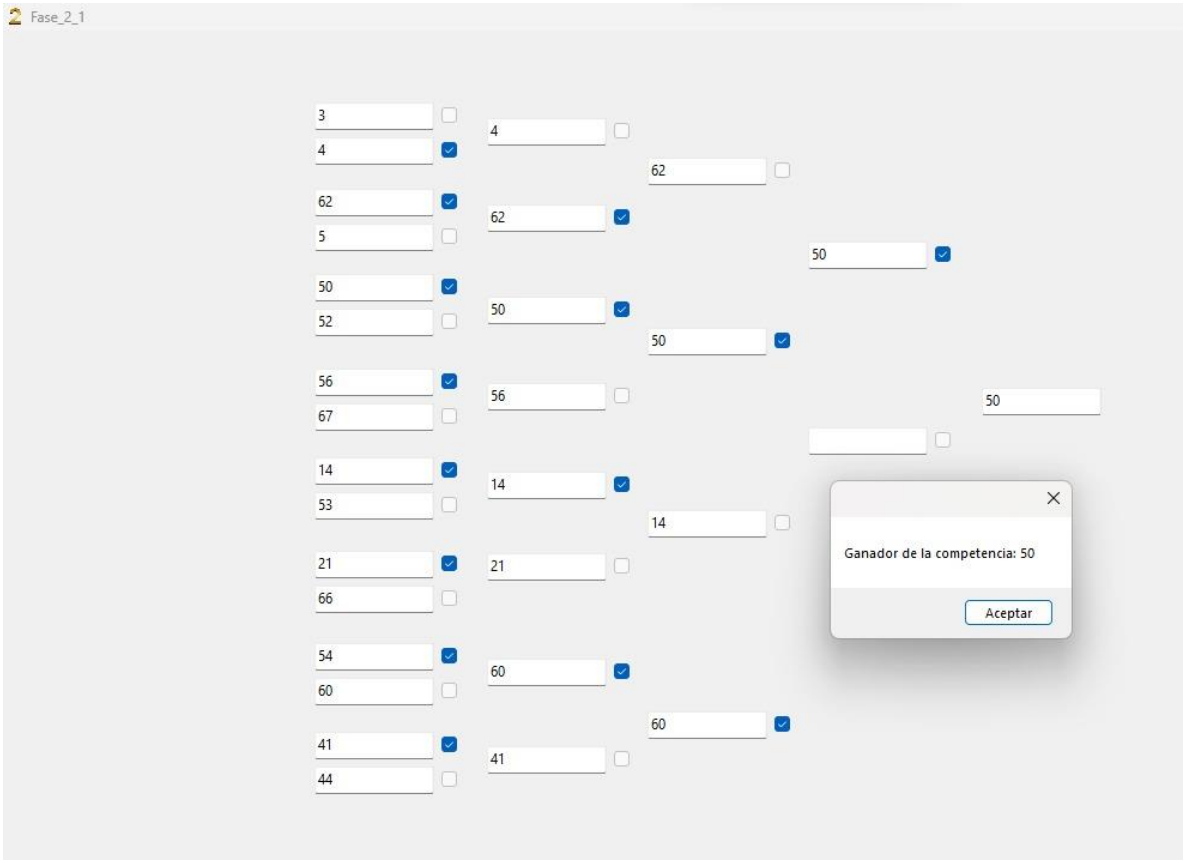

Ilustración 56: Selección de ganador (Ian Omar Madrigal Rueda)

Capítulo V.
Conclusiones

A través de la investigación realizada, se ha demostrado que la combinación de tecnologías y herramientas utilizadas en el proyecto permitió la creación de una solución eficiente y efectiva para gestionar las competencias de robots sigue líneas. La utilización del lenguaje de programación C# en Visual Studio proporcionó un entorno de desarrollo robusto y versátil, facilitando la implementación de la interfaz de usuario y la interacción con el microcontrolador Arduino.

La integración del sensor de evitación de obstáculos por infrarrojos E18-D80NK permitió al sistema detectar con precisión el momento exacto en que cada robot ejecutaba una vuelta en su respectiva pista de "carreras". Esto aseguró un seguimiento preciso de los desempeños de los robots, lo que es esencial para evaluar el rendimiento y la efectividad de los algoritmos de seguimiento de línea implementados en cada uno de ellos.

Además de cumplir con los objetivos planteados, el proyecto arrojó resultados adicionales que pueden tener aplicaciones más allá de las competencias de robots sigue líneas, como la mejora de la navegación autónoma en entornos restringidos. Esta versatilidad sugiere que la aplicación desarrollada puede tener un impacto significativo en la industria de la robótica y la automatización.

No obstante, se identificaron ciertas limitaciones que obstaculizaron el proyecto. Estas limitaciones pueden incluir posibles problemas de calibración del sensor infrarrojo, la falta de recursos suficientes para realizar pruebas exhaustivas en diferentes escenarios.

A pesar de las excepciones u omisiones encontradas durante el desarrollo del proyecto, se logró demostrar que la automatización de las competencias de robots sigue líneas es factible y altamente prometedora. Las conclusiones extraídas de esta investigación brindan una base sólida para futuros desarrollos en el campo de la mecatrónica, alentando a estudiantes e investigadores a continuar explorando y mejorando esta área de estudio.

En conclusión, el proyecto representa un paso significativo hacia la automatización de competencias de robots sigue líneas y ha demostrado el potencial de combinar el lenguaje de programación C#, Visual Studio, bases de datos, microcontroladores y la interfaz Arduino para crear soluciones mecatrónicas innovadoras y eficientes.

Fuentes de Información

[1] comrob.org, «comrob.org,» [En línea]. Available: https://comrob.org/2014/wp-content/uploads/2014/07/Reglamento-del-concurso_siguelinea.pdf. [Último acceso: 5 Agosto 2023].

[2] cetis107.edu.mx, «cetis107.edu.mx,» [En línea]. Available: https://cetis107.edu.mx/portal/archivos/Convocatoria-Robot-Seguidorde-Linea.pdf. [Último acceso: 5 Agosto 2023].

[3] uacm, «www.uacm.edu.mx,» [En línea]. Available: https://www.uacm.edu.mx/publicaciones/2023/4/24/ConvocatoriaConcursoRobot_Seguidor_Linea.pdf. [Último acceso: 5 Agosto 2023].

[4] uat, «fic.uat.edu.mx,» [En línea]. Available: https://fic.uat.edu.mx/assets/convocatorias/aniversario/ConvocatoriaSiguelineas.pdf. [Último acceso: 5 Agosto 2023].

[5] itescam, «www.itescam.edu.mx,» [En línea]. Available: https://www.itescam.edu.mx/portal/convocatorias.php?id=437. [Último acceso: 5 Agosto 2023].

[6] UV, «www.uv.mx,» [En línea]. Available: https://www.uv.mx/veracruz/uvca415-sistemas-dinamicos-autonomos/files/2018/11/Reglamento-RSL-FIEE.pdf. [Último acceso: 7 Agosto 2023].

[7] ucatolica, «www.ucatolica.edu.co,» [En línea]. Available: https://www.ucatolica.edu.co/portal/wp-content/uploads/2019/09/concurso-robot-seguidor-de-linea.pdf. [Último acceso: 7 Agosto 2023].

[8] icti, «icti.michoacan.gob.mx,» [En línea]. Available: https://icti.michoacan.gob.mx/wp-content/uploads/2022/11/B_Carrera-de-Seguidores-De-Linea.pdf. [Último acceso: 7 Agosto 2023].

[9] L. J. Aguilar, «Fundamentos de programación,» de *Fundamentos de programación Algoritmos, estructura de datos y objetivos*, España, Mc Graw Hill, 2008, p. 800.

[10] ionos, «www.ionos.mx,» [En línea]. Available: https://www.ionos.mx/digitalguide/paginas-web/desarrollo-web/compilador-e-interprete/. [Último acceso: 21 Mayo 2024].

[11] freecodecamp, «www.freecodecamp.org,» [En línea]. Available: https://www.freecodecamp.org/espanol/news/lenguajes-compilados-vs-interpretados/. [Último acceso: 5 Nobiembre 2023].

[12] m. w. docs, «developer.mozilla.org,» [En línea]. Available: https://developer.mozilla.org/es/docs/Learn/JavaScript/First_steps/What_is_JavaScript. [Último acceso: 23 Noviembre 2023].

[13] Amazon, «aws.amazon.com,» [En línea]. Available: https://aws.amazon.com/es/what-is/python/. [Último acceso: 25 Noviembre 2023].

[14] fcasua, «fcasua.contad.unam.mx,» [En línea]. Available: http://fcasua.contad.unam.mx/apuntes/interiores/docs/98/4/informatica_4.pdf. [Último acceso: 2024 Mayo 21].

[15] microsoft, «learn.microsoft.com,» [En línea]. Available: https://learn.microsoft.com/es-es/visualstudio/get-started/visual-studio-ide?view=vs-2022. [Último acceso: 7 Agosto 2023].

[16] microsoft, «learn.microsoft.com,» [En línea]. Available: https://learn.microsoft.com/es-es/dotnet/desktop/winforms/overview/?view=netdesktop-8.0. [Último acceso: 7 Agosto 2023].

[17] oracle, «www.oracle.com,» [En línea]. Available: https://www.oracle.com/mx/database/what-is-database/. [Último acceso: 9 Agosto 2023].

[18] mariadb, «mariadb.com,» [En línea]. Available: https://mariadb.com/kb/es/basic-sql-statements/. [Último acceso: 10 Noviembre 2023].

[19] biblus.us.es, «biblus.us.es,» [En línea]. Available: https://biblus.us.es/bibing/proyectos/abreproy/11301/fichero/Memoria%252FCap%C3%ADtulo+2.pdf+. [Último acceso: 10 Noviembre 2023].

[20] arduino, «docs.arduino.cc,» [En línea]. Available: https://docs.arduino.cc/hardware/uno-rev3/. [Último acceso: 13 Noviembre 2023].

[21] Arduino, «www.arduino.cc,» [En línea]. Available: https://www.arduino.cc/reference/en/. [Último acceso: 26 noviembre 2023].

[22] Hdeleon.net, «Conectar Arduino con C# .Net | Prender un LED con C# .Net | Novatos,» YouTube, 2018.

[23] Electronica Valtierra, «electronicavaltierra.com.mx,» [En línea]. Available: https://electronicavaltierra.com.mx/producto/sensor-infrarrojo-de-proximidad-e18-d80nk/. [Último acceso: 1 agosto 2023].

[24] Solectro, «solectroshop.com,» [En línea]. Available: https://solectroshop.com/es/content/82-como-detectar-objetos-con-sensor-de-proximidad-infrarrojo-e18-d80nk-en-arduino. [Último acceso: 1 agosto 2023].

[25] Freepik, «www.freepik.es,» [En línea]. Available: https://www.freepik.es/vector-premium/ilustracion-vector-diseno-estilo-plano-plantilla-campeonato-soporte-torneo-8-equipos_25244746.htm. [Último acceso: 1 Agosto 2023].

[26] fp.uoc.fje.edu, «fp.uoc.fje.edu,» [En línea]. Available: https://fp.uoc.fje.edu/blog/que-lenguajes-de-programacion-desarrollan-interfaces-graficas/. [Último acceso: 7 Agosto 2023].