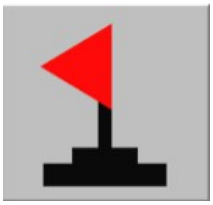
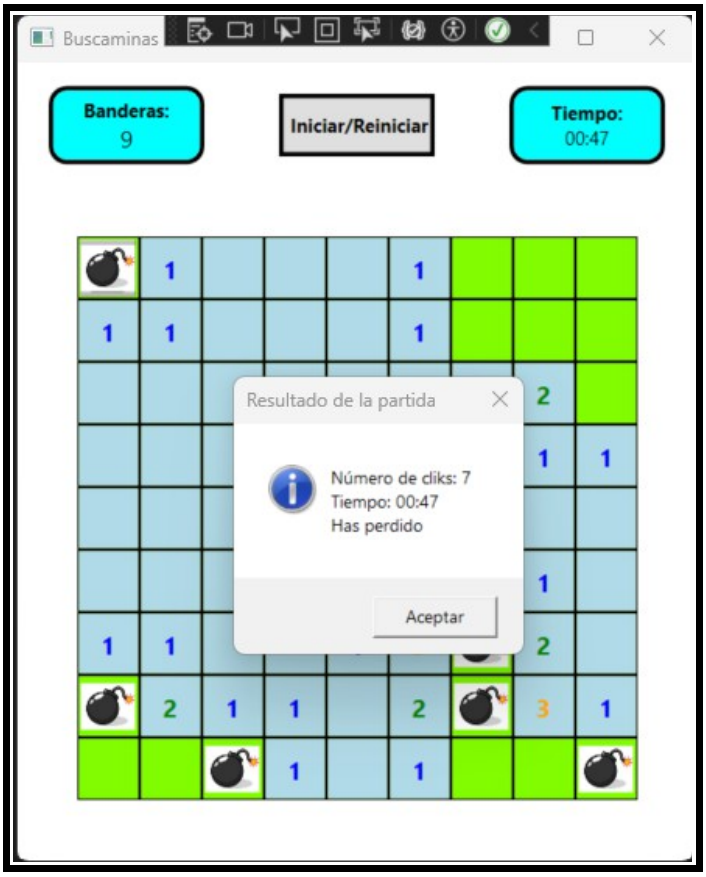
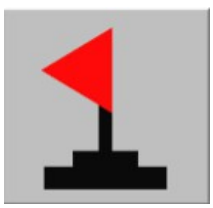


PROYECTO

BUSCAMINAS

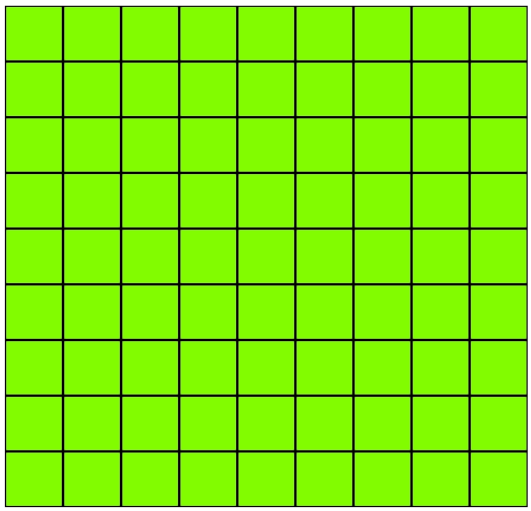




INDICE

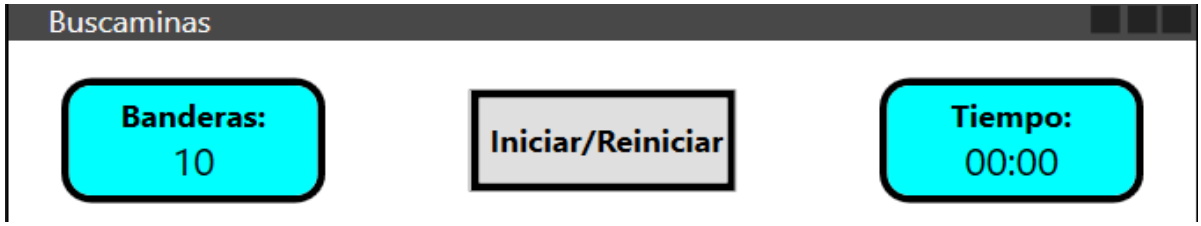
XAML:

La ventana le he puesto un **Grid** que ocupa toda la venta y en este podemos encontrar otro **Grid** para formar el tablero de jugeo, y dentro de él, podemos ven que hemos formado un **Grid.RowDefinitions**, en el cual hemos creado las 9 filas, y un GridColumnDefinicions donde hemos creado 9 columnas.
A los botones le hemos puesto un estilo, llamado **CustomButtonStyle,** donde le hemos agregado un **Backgroung** de color LawnGreen, con tamaño de fuente de 16, un **BorderBrush** en color negro y con un **BorderThickness** de 1, además cada botón tiene unas dimensiones de 40x40.
Todo ello esta recogido en un estilo que lo añadimos por C# al tablero.



A continuación he creado el botón de reinicio, y el temporizador y contador de banderas:

- Botón de reinicio: le he puesto un **Width** de 100 y un **Heigt** de 40, un **HorizontalAlignment** center y un **verticalAlignment Top** , con un **margin** de 20 arriba y abajo, un **BorderBrush** en negro y un **BorderThickness** de 3. Además al botón le hemos puesto un estilo al tipo de letra con un **FontWeight** para que la letar nos salga en negrita con **Bold.**
- Border temporizador y contador de banderas: estos dos **Border** son exactamento solamente que uno con un **HorizontalAlignment** a la izquierda y otro a la derecha. Tienen un **BorderThinckness** de 3 y un **BorderBrusch** negro, Background en colos Aqua, **Width** de 100 y un **Heigt** de 50, **verticalAlignment Top** y un **Margin** de izquierda-derecha de 20 y arriba-abajo de 15 y por último un CornerRadius de 10 para hacer los bordes redondeados, que por esta razón es por lo que el contenedor padre es un Border. Dentro de este hay un Stackpanel con un **HorizontalAlignment** y un **verticalAlignment Top** centrado para que el texto interior este centrado ya que este esta metido en dos **TextBlock** para que queden los enunciados arriba con que es cada visor, y abajo el texto que se modifica gracias a c#.



C#:

Primero declaramos todas la variables y despues comenzamos en la función MainWindow, siendo estas las siguientes:

```
private DispatcherTimer timer;
private int elapsedSecond;
private Button[,] boardButtons; // matriz de botones
private int boardSize = 9; //tamaño del tablero (9x9)
int flag1 = 0; //banderas colocadas de inicio
int flag2 = 10; //banderas permitidas
String relativeBomb = "./img/bomba.png";
String relativeFlag = "./img/flag.png";
private int clickCount = 0;
private bool gameWon = false;
```

Las variables son para crear la matriz del tamaño que queremos, el número de banderas, las rutas relativas de las imagenes, aunque recordemos que solamente he utilizado al final la de la bomba, para controlar el temporizador y el boolean gameWon() en false

Main Window: En esta función, lo primero que hacemos es llamar a otras dos funciones como son InitializeComponent() y InicializaBoard(), las cuales explicaré más tarde. En la función que estamos, lo primero que hacemos es inicializar el temporizados y le ponemos función al boton de reinicio

InitializeComponent(): Esta función se gener automaticamente en proyectos de aplicaciones de interfaz de usuario con tecnologías como WPF o Windows Forms en C#, se encarga de inicializar y configurar componentes de la interfaz de usuario definidos en el archivo de marcado (XAML).

Timer_Tick: Con esta función lo que hacermos es dar un seguimiento del tiempo que transcurre en segundos, convirtiendo estos en minutos y segundos con la siguiente linea de codigo `timerText.Text = $"{timeSpan.Minutes:00}:{timeSpan.Seconds:00}".`

```
private void Timer_Tick(object sender, EventArgs e)
{
    elapsedSecond++;
    TimeSpan timeSpan = TimeSpan.FromSeconds(elapsedSecond);
    timerText.Text = $"{timeSpan.Minutes:00}:{timeSpan.Seconds:00}";
}
```

InicializaBoard(): Con esta función lo que hacemos es inicializar el tablero con los botones , creando y configurando una matriz de botones siendo:

- Creo la matriz con un tamaño establecido en una variable llamada **boardSize**
- Con un bucle for anidado se recorre cada fila y cada columna haciendo:
  - creamos un nuevo botón con los siguientes parametros establecidos en la función **CreateButton()**
  - Asignamos a al nuevo boton la posición en la matriz **boardButtons**

- **boardGrid.Children.Add(button)**, agrega el botón al elemento de interfaz llamado boardGrid.
- **Grid.SetRow(button, row) y Grid.SetColumn(button, col)**, establece las propiedades de la fila y la columna del botón en el Grid(boardGrid), estableciendo la posición visual del botón en la cuadricula.
- **Button.Click += Button_Click**, asocia el método de Button_Click al evento Click del botton.
- **Button.MouseRightButtonDown += Buttonflag**, asocia el mettodo Buttonflag al evento Button.MouseRightButtonDown del botón.
- **boardButtons[row, col]. Tag = "Bomb"**, marca el botón en el tablero como una casilla que contiene una bomba, utilizando la propiedad Tag

- Por último llamamos a la función **GenerateBombs()** que será la encargada de generar las bombas.

```
private void InicializeBoard()
{
    boardButtons = new Button[boardSize, boardSize];

    //generamos los botones y los agregamos al grid

    for (int row = 0; row < boardSize; row++)
    {
        for (int col = 0; col < boardSize; col++)
        {
            Button button = CreateButton(row, col);
            boardButtons[row, col] = button;

            //agragar el botón al Grid
            boardGrid.Children.Add(button);

            Grid.SetRow(button, row);

            Grid.SetColumn(button, col);


            button.Click += Button_Click;

            button.MouseRightButtonDown += Buttonflag;


        }
    }

    //Generamos las imagenes de las bombas
    GenerateBombs();
}
```

CreateButton(): Con esta función se crea y se configura el botón de la siguiente manera:

- primero se crea una nueva instancia de la clase **Button** y le da la configuración siguiente:
  - Le da un nombre al botón cocatenando la cadena cell con los valores de row y col.

- Establece el contenido con una cadena vacia
- Se habillita el botón para que sea interactivo
- Y se le da un estilo establecido en "**CustomButtonStyle**" en **XAML**

```
Button button = new Button
{
    Name = $"cell{row}{col}",
    Content = "",
    IsEnabled = true,
    Style = (Style)FindResource("CustomButtonStyle"),
};
```

GenerateBombs(): Esta función se encarga de generar las bombas para el tablero de juego de la siguiente manera:

- Se generan numero aleatorios y una lista bombPositions para almacenar las bombas generadas.
- Con el **for()** se genera el número especifico de bombas que queremos, en cada iteración se genera una posición random de filas y columnas.
- Con el bucle **do – while**, se garantiza que no se dupliquen bombas en la misma posición, si las posiciones generadas ya estan en bombPosition, se generan unas nuevas coordenadas.
- Una vez generadas las coordenadas se añaden a la lista bombPosition.
- Se crea la imagen de la bomba llamando a la función **CreateBombImage()** y se establece su posición en la cuadrícula.
- Sse marca la posición en la matriz **boardButtons** con la etiqueta "**Bomb**" utilizando la etiqueta **.Tag**.

```
private void GenerateBombs()
{
    Random random = new Random();
    List<Tuple<int, int>> bombPositions = new List<Tuple<int, int>>();

    for (int i = 0; i < 10; i++)
    {
        int row, col;
        do
        {
            row = random.Next(boardSize);
            col = random.Next(boardSize);
        }
        while (bombPositions.Contains(Tuple.Create(row, col)));

        bombPositions.Add(Tuple.Create(row, col));

        Image bombImage = CreateBombImage();
        Grid.SetRow(bombImage, row);
        Grid.SetColumn(bombImage, col);
        boardGrid.Children.Add(bombImage);

        boardButtons[row, col].Tag = "Bomb";
    }

}
```

CreateBombImage(): primero creamos una nueva instancia de la clase Image,

- se establece la propiedad **Source** de la imagen para que sea una instancia de **BitmapImage** que carga la imagen en este caso, de una ruta relativa que esta declarada como variable.
- Inicialmente se le da una visivilidad **Hidden,** lo que significa que la bomba no será visible.
- Por último le damos un **Width** y un **Height** de 35 para que entre correctamente en el botón.

```
private Image CreateBombImage()
{
    return new Image

    {
        Source = new BitmapImage(new Uri(relativeBomb, UriKind.Relative)),
        Visibility = Visibility.Hidden,
        Width = 35,
        Height = 35,
    };
}
```

La imagen de la bomba la elegí por ser una opción vistosa.

StartTimer(): Función que realiza el restablecimiento del tiempo, actualiza el timerText a 00:00 y loego lo inicia.

```
private void StartTimer()
{
    elapsedSecond = 0;
    timerText.Text = "00:00";
    timer.Start();
}
```

StopTimer(): Función que realiza la parada del tiempo.

Button_Click(): Función que se utiliza para controlar los eventos del clic del botón en la interfaz:

- **clickCount**: incrementa el contador de clicks realizados.
- **Button button = (Button)sender, int row = Grid.GetRow(button), int col = Gris.GetColumn(button)**: obtiene el botón donde se clica con la fila y la columna.
- **If(){...}**: verifica si la casilla en la que se clica contiene una bomba, siendo así se realiza todo el interior como mostrar la imagen de la bomba, para el tiempo, poner la funcion gameWon() en falso(que ya comentaré esta función), y mostrar la función PopUp() (que también comentare más tarde). En caso contrario, si lo que hay es un número, simplemente descubre ese número, cambiando el color del número dependiendo de cual es con SetNumberColor().Si el espacio esta completamente vacio, es decir, que no tiene ninguna bomba en los 8 botones que hay a su alrededor, llama a la función ExpandCells() que detallaré más adelante, pero que realiza la "apertura" de los botones que estan igualmente vasios hasta que se encuentre botones con un número de bombas

adjacentes.
Cuando se abren los botones, cambian de color verde a color azul diferenciando de esta manera que el botón esta abierto o cerrado.

```
private void Button_Click(object sender, RoutedEventArgs e)
    {
        clickCount++; // Incrementa el contador de clicks
        Button button = (Button)sender;
        int row = Grid.GetRow(button);
        int col = Grid.GetColumn(button);

        Console.WriteLine("Button_Click2");

        if (boardButtons[row, col].Tag != null)
        {
            //La casilla hace click contiene una bomba
            //Mostrar la imagen de la bomba y realizar otras acciones

            Image bombImage = boardGrid.Children
                .OfType<Image>()
                .FirstOrDefault(image => Grid.GetRow(image) == row && Grid.GetColumn(image) == col);

            bombImage.Visibility = Visibility.Visible;
            button.IsHitTestVisible = false;
            timer.Stop();
            gameWon = false;
            ShowAllBombs(); //para mostrar todas las bombas
            ShowGameResult();


        }
        else
        {
        //La casilla hace click no contiene una bomba
        //Revelar el contenido de la casilla, como un número o un espacio vacio

            int bombsAdjacent = CountAdjacentBombs(row, col);
            if (bombsAdjacent > 0)
            {
                button.Content = bombsAdjacent;
                SetNumberColor(button, bombsAdjacent);
            }
            else
            {
                //Es para que no me muestre ninguna cifra
                ExpandCells(row, col);
            }

            //establecer el color de los numero en función de la cantidad

            SetNumberColor(button, bombsAdjacent);

            //cambia el color de fondo

            button.Background = Brushes.LightBlue;
            button.IsHitTestVisible = false;
        }
    }
```

ShowAllBombs(): Función que realiza la "apertura" de los botones que tienen bomba una vez finalizada la partida, recorriendo todas los botones con los dos for anidados, y verifica si el .Tag del inicio en la celda es nulo o si es igual a la cadena "Bomb", y si es así, busca la imagen y si encuentra la imagen de la bomba, la hace visible.

```
private void ShowAllBombs()
{
    for(int i = 0; i< boardSize; i++)
    {
        for (int j = 0; j < boardSize; j++)
        {
            if (boardButtons[i, j].Tag != null && boardButtons[i, j].Tag.ToString() == "Bomb")
            {
                Image bombImage = boardGrid.Children
                    .OfType<Image>()
                    .FirstOrDefault(image => Grid.GetRow(image) == i && Grid.GetColumn(image) == j);
                if(bombImage != null)
                {
                    bombImage.Visibility = Visibility.Visible;
                }
            }
        }
    }
}
```

Buttonflag(): En si la función es para poner y quitar banderas que lo realiza de la siguiente manera:

- Button button...., obtiene el botón que desencadena el evento.
- if(flag1 < flag2){´...}, lo que se ve es si hay banderas para poner ya que estas variables han sido creadas al principio donde flag1 = 0 y flag2 = 10, si esto se cumple, se coloca la imagen de la bandera, en este caso he utilizado una ruta relativa, aunque la variable de esta la tenia creada pero queria hacer una imagen de cada. Le damos un contenido al botón con la imagen de la bandera y añadimos a flag1, 1 unidad. En caso de que lo que hayamos hecho es quitar la bandera, devolvemos el contenido del botón y restamos una unidad a flag1. Por último, al Border de Bandera, le actualizamos la información, restando a flag2, flag1.
- Imagen de la bandera, aposte por la bandera clasica haciendo un homenaje al juego.

```
private void Buttonflag(object sender, RoutedEventArgs e)
{
    Button button = (Button)sender;


    if(flag1 < flag2)
    {
        //se colocan las banderas
        Image flagImage = button.Content as Image;
        if(flagImage == null)

        {
            flagImage = new Image();

            flagImage.Source = new BitmapImage(new Uri("./img/flag.png", UriKind.Relative));

            button.Content = flagImage;
            flag1++;
        }
        else
        {
            //cuando quitamos la bandera
            button.Content = null;
            flag1--;
        }
        //para contabilizar las banderas que nos quedan
        Bandera.Text = (flag2 - flag1).ToString();
    }
}
```

ShowGameResult(): Esa función lo que realiza es mostrar una pantalla emergente donde nos informa del resultado, el tiempo trascurrido y el numero de clicks.

```
private void ShowGameResult()
{
    string result = gameWon ? "Has ganado" : "Has perdido ";
    MessageBox.Show($"Número de cliks: {clickCount}\nTiempo: {timerText.Text}\n{result}", "Resultado de la partida", MessageBoxButton.OK, MessageBoxImage.Information);
}
```

CountAdjacentBombs(): Con esta fuunción lo que se pretende es contar el numero de bombas adyacentes de una celda de la siguiente manera:

- Primero inicializamos la variable bombCount que se utiliza para contar las bombas.
- Creamos dos arreglos dx y dy, representando las direcciones adyacentes, y con ellas exploramos cada una de las celdas adyacentes.

```
int[] dx = { -1, -1, -1, 0, 0, 1, 1, 1 };
int[] dy = { -1, 0, 1, -1, 1, -1, 0, 1 };
```

- Con un for(){...} iteramos sobre todas las direcciones y verifica las celdas adyacentes.
- En el bucles se calculan las nuevas posiciones para verificar las celdas
- Se verifican que las nuevas posiciones esten dentro de los limites del tablero

```
if(newRow >= 0 && newRow < boardSize && newCol >= 0 && newCol < boardSize)
```

- Por último verificamos si la celda adyacente tiene una bomba (Bomb), siendo asi, se

incrementa el numero de bombas(bombCount++)

```
private int CountAdjacentBombs(int row, int col)
{
    int bombCount = 0;

    //Definir las direcciones para verificar las casillas adyacentes
    int[] dx = { -1, -1, -1, 0, 0, 1, 1, 1 };
    int[] dy = { -1, 0, 1, -1, 1, -1, 0, 1 };

    for(int i= 0; i < 8; i++)
    {
        int newRow = row + dx[i];
        int newCol = col + dy[i];

        //verificamos si las casillas adyacentes está dentro de los limites del tablero.

        if(newRow >= 0 && newRow < boardSize && newCol >= 0 && newCol < boardSize)
        {
            if (boardButtons[newRow, newCol].Tag != null && boardButtons[newRow, newCol].Tag.ToString() == "Bomb")
            {
                bombCount++;
            }
        }
    }
     return bombCount ;

}
```

ExpandCells(): Esta función realiza la la operación de que cuando clicamos en un botón y esta vacio, "abre" los botones que hay a su alrededor vacios, y deterner esta "apertura" si la celda contiene un número. Realiza esta operación de la siguiente manera:

- Si las casillas adyacentes estan fuera de los limites del tablero, retorna a la función.

```
if (row < 0 || col < 0 || row >= boardSize || col >= boardSize)
```

- Si las casillas ya se han expandido/abierto:

```
if (boardButtons[row, col].Content == null)
```

- Después desactivamos el poder hacer click en la misma celda.
- Calcula la cantidad de bombas adyacentes con la función **CountAdjacentBombs()**, y si la la cantidad de bombas es 0, verifica si esta vacia, y si es así, se colorea el Background de color LightBlue, y se utiliza el bucle anidado para llamar recursivamente a la funcion **ExpandCells().**
- Else{...}, Si la celda tiene bombas adyacentes, se revela el numero de bombas, se le da color al número y al background

```
private void ExpandCells(int row, int col)
{
    //casillas fuera de los limites del tablero

    if (row < 0 || col < 0 || row >= boardSize || col >= boardSize)
    {
        return;
    }
    //casillas que ya se han expandido/abierto
    if (boardButtons[row, col].Content == null)
    {
        return;
    }
    boardButtons[row, col].IsHitTestVisible = false;
    int bombsAdjacent = CountAdjacentBombs(row, col);

    if (bombsAdjacent == 0)
    {
        //la casilla está vacía, la expandimos y continuamos con las casillas adyacentes
        boardButtons[row, col].Content = null;
        boardButtons[row, col].Background = Brushes.LightBlue;
        for (int i = -1; i <= 1; i++)
        {
            for (int j = -1; j <= 1; j++)
            {
                ExpandCells(row + i, col + j);

            }
        }
    }
    else
    {
        // las casillas tienen número, las revelamos y detenemos la expansión
        boardButtons[row, col].Content = bombsAdjacent;
        SetNumberColor(boardButtons[row, col], bombsAdjacent);
        boardButtons[row, col].Background = Brushes.LightBlue;
    }
}
```

ResetButton_Click() : Esta función es para resetear el juego de la siguiente manera:

- primero llama a la función **InicializeBoard();**
- Después resetea el tiempo con **StartTimer();**
- Reinicia las banderas del flag1 a 0 y le dice al contador de banderas que marque las de inicio.

```
private void ResertButton_Click(object sender, RoutedEventArgs e)
{
    //reiniciar el tablero
    InicializeBoard();
    //reiniciar el temporizador
    StartTimer();
    //reiniciar las banderas
    flag1 = 0;
    Bandera.Text = flag2.ToString();
}
```

SetNumberColor(): Esta es la función que le da color a los número de las bombas adyacentes, siendo los colores y números los siguientes:

```
if(bombCount == 1)
{
    button.Foreground = Brushes.Blue;
}
else if(bombCount == 2)
{
    button.Foreground= Brushes.Green;
}else if(bombCount == 3)
{
    button.Foreground= Brushes.Orange;
}else if(bombCount == 4){

    button.Foreground= Brushes.Purple;
}else if(bombCount == 5)
{
    button.Foreground= Brushes.Red;
}
```

IMAGENES DEL JUEGO:

INICIO **JUGANDO** **FINALIZADO**