**DOCUMENTACIÓN DE CÓDIGO**

**Proyecto:** Mastermind

**Fecha:** 31/05/2022

**Desarrollado por:** Young Development

# Tabla de contenido

**Descripción general**

El código presentado a continuación representa al proyecto Mastermind el cual consiste en la creación del juego con el mismo nombre, para ello se ha divido el informe en varias secciones como: Lógica del programa, Conexión de la lógica con la interfaz, finalmente la Interfaz y los anexos del código fuente; La primera sección es referida a la lógica del programa, todo lo que consiste a cómo debe operar el juego Mastermind, la segunda sección se basa en la manera en la que la lógica se conecta con la interfaz del juego, finalmente, la última sección se observará la interfaz gráfica y sus eventos, sin embargo, después también se anexarán el código fuente.

Adicionalmente, es recomendable conocer cómo funciona y en que consiste el juego Mastermind con el objetivo de entender al máximo la documentación del código.

**Primera sección: Lógica del programa**

***Descripción de la sección***

Para la sección que permite definir la lógica del programa se decidió crear una clase que represente toda la lógica del juego Mastermind a manera que pueda abstraerse y poder operar con él para cumplir todos los objetivos previsibles obtenidos en la toma de requerimientos del proyecto.

***Clase Mastermind***

La clase consta de 4 regiones que conforman al objeto Mastermind: Variables, El constructor, Métodos y Getters. La primera región se refiere a las variables que serán utilizadas por la clase, la región del constructor consiste en la manera en la que se generara el juego, la región de los métodos consiste en métodos substanciales que permiten la operabilidad de la clase, finalmente, la región de getters consiste en métodos que permiten devolver datos del objeto Mastermind.

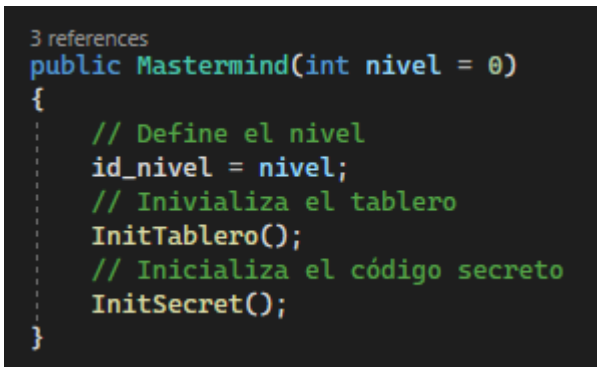
***Región: Variables***

```
#region Variables

// Definición del tablero
int[][] tablero = new int[10][];

// Definición de los colores
// [0]->Blanco, [1]->Rojo, [2]->Amarillo, [3]->Morado, [4]->Verde, [5]->Negro, [6]->Anaranjado, [7]->Rosado, [8]->Azul
//                    [0]        [1]       [2]        [3]        [4]        [5]       [6]         [7]       [8]
string[] colores = { "#FFFFFF", "#FF0000", "#FFA500", "#800080", "#9ACD32", "#000000", "#FF7F50", "#FF00FF", "#000080" };

// Definición del código secreto
int[] secret = new int[4];

// Define un código para el tipo de nivel
// 0 -> No Definido | 1 -> Fácil | 2 -> Medio | 3 -> Difícil
int id_nivel = 0;

#endregion
```

1. La variable tablero es la permite definir el tablero en el cual se realizarán todas las combinaciones realizadas por el jugador.
2. La variable colores consiste en la definición de todos los colores que serán utilizados en la aplicación tanto para las combinaciones como para la validación de los aciertos, los colores son: blanco, rojo, amarillo, morado, verde, negro, anaranjado, rosado y azul.
3. La variable secret es la que permitirá guardar el código secreto que cada juego de Mastermind debe tener.

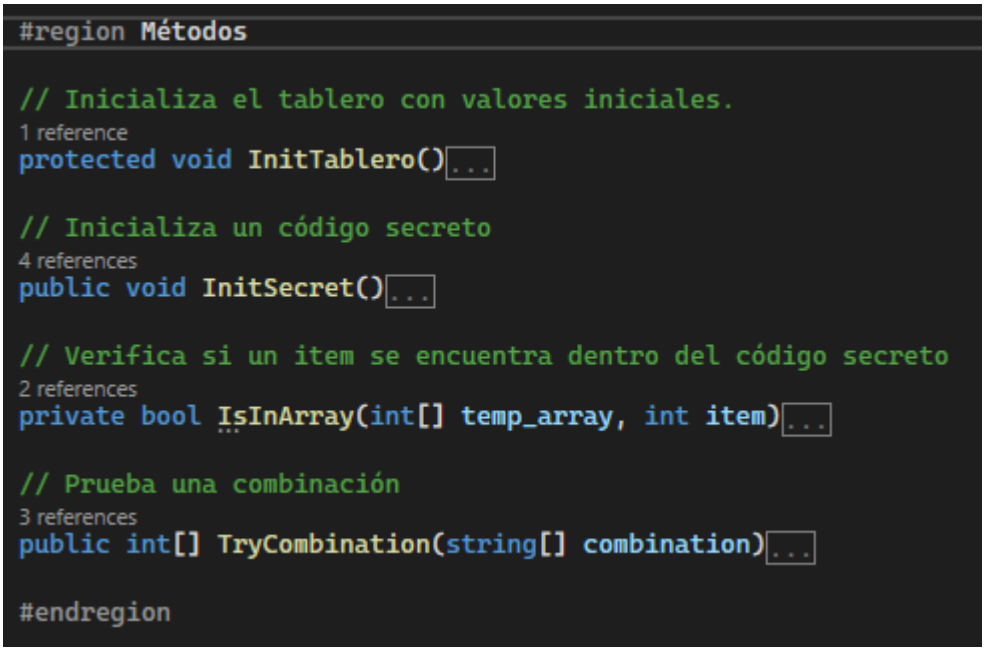
4. La variable id_nivel permite que al momento de crear el objeto Mastermind, este se cree en base a un cierto de nivel de dificultad siendo 1 el nivel fácil, 2 el nivel medio y 3 el nivel difícil; sin embargo, si el nivel es 0, el nivel no está definido.

***Región: Constructor***




El constructor del objeto Mastermind consiste en la definición del nivel que puede ser tanto fácil, medio y difícil, a su vez, permite inicializar el tablero y también inicializar el código secreto del juego.

***Región: Métodos***




**InitTablero**

```
// Inicializa el tablero con valores iniciales.
1 reference
protected void InitTablero()
{
    for (int i = 0; i < 10; i++) tablero[i] = new int[4];
}
```

El método presente permite crear e inicializar el tablero de juego de Mastermind inicializando cada fila con 4 columnas.

### InitSecret

El método InitSecret cumple varias funciones:

1. Primero inicializa los valores del código secreto y crea la variable max_secret que define la cantidad de colores que podrán ser usados en cada nivel.

```
public void InitSecret()
{
    secret = new int[4];
    int max_secret;
```

2. El método también permite crear el código secreto para el nivel fácil el cual tiene únicamente 4 colores para sus combinaciones, después, permite generar un código aleatorio que representará un color de los habilitados en la variables colors, sin embargo, dado que en el nivel fácil los colores no se pueden repetir, el código secreto tampoco puede repetir los colores así pues se valida que mientras algún ítem del código creado ya exista dentro del código secreto, entonces generará un nuevo ítem del código secreto.

```
// Código secreto para el nivel fácil
if (id_nivel == 1)
{
    // Máximo de opciones de color
    max_secret = 4;
    Random rand = new Random();
    // Creación del código secreto
    for (int col = 0; col < 4; col++)
    {
        var temp_color = rand.Next(1, max_secret + 1);
        // Se verifica que no se repitan los colores
        while (IsInArray(secret, temp_color)) temp_color = rand.Next(1, max_secret + 1);
        secret[col] = temp_color;
    }
}
```

3. Por otra parte, también permite definir el código secreto para el nivel medio el cual tiene 8 colores para sus combinaciones, el resto, funciona de igual manera que la creación del nivel fácil.

```
// Código para el nivel medio
else if (id_nivel == 2)
{
    // Máximo de opciones de color
    max_secret = 8;
    Random rand = new Random();
    // Creación del código secreto
    for (int col = 0; col < 4; col++)
    {
        var temp_color = rand.Next(1, max_secret+1);
        // Se verifica que no se repitan los colores
        while (IsInArray(secret, temp_color)) temp_color = rand.Next(1, max_secret + 1);
        secret[col] = temp_color;
    }
}
```

4. Finalmente, permite crear el código secreto para el nivel difícil que de igual manera permite 8 colores, sin embargo, esta vez estos ya se pueden repetir por lo que únicamente se genera ítems del código sin validar si estos ya existían.

```
// Código para el nivel dificil
else if (id_nivel == 3)
{
    // Máximo de opciones de color
    max_secret = 8;
    Random rand = new Random();
    // Creación del código secreto sin importar que se repitan los colores
    for (int col = 0; col < 4; col++)
    {
        secret[col] = rand.Next(1, max_secret);
    }
}
}
```

**IsInArray**

```
// Verifica si un item se encuentra dentro del código secreto
2 references
private bool IsInArray(int[] temp_array, int item)
{
    foreach (var iter in temp_array) if (iter == item) return true;
    return false;
}
```

El método presento permite verificar si un ítem evaluado existe dentro de un arreglo de datos, si este existe devuelve verdadero, si no, falso.

**TryCombination**

Él método TryCombination también permite realizar varias situaciones:

1. Inicializa una variable que almacenará todos los aciertos que el usuario haga en base a la combinación que este realice y el código secreto generado, esto, en base a los siguientes códigos: 0 se refiere a que no hubo aciertos, 1 se refiere a

que hubo un acierto, pero en la posición incorrecta y 2 si hubo un acierto en la posición correcta.

```
// Prueba una combinación
3 references
public int[] TryCombination(string[] combination)
{
    // Definición aciertos y tipo de acierto
    // 0 -> No hubo acierto
    // 1 -> Acierto en la posición incorrecta
    // 2 -> Acierto en la posición correcta
    int[] hits = new int[4];
    // Contador
    int hit_count = 0;
```

2. Por otra parte, permite descubrir los aciertos que el usuario realizo en la posición correcta guardando el código 2 en la variable hits.

```
    // Descubrir aciertos en la posición correcta
    for (int i = 0; i < combination.Length; i++)
    {
        if (colores[secret[i]] == combination[i])
        {
            hits[hit_count++] = 2;
        }
    }
```

3. Asimismo, permite descubrir los aciertos en la posición incorrecta guardando el código 1 en la variable hits.

```
    // Descubrir aciertos en la posición incorrecta
    for (int i = 0; i < combination.Length; i++)
    {
        for (int j = 0; j < secret.Length; j++)
        {
            if (colores[secret[j]] == combination[i] & !(colores[secret[i]] == combination[i]))
            {
                hits[hit_count++] = 1;
            }
        }
    }
    // Devuelve los aciertos
    return hits;
}
```

***Región: Getters***

```
#region Getters


public string[] GetSecret()
{
    string[] temp = new string[4];
    for(int i = 0; i < secret.Length; i++)
    {
        temp[i] = colores[secret[i]];
    }
    return temp;
}


public int GetIdLevel()
{
    return id_nivel;
}

#endregion
```

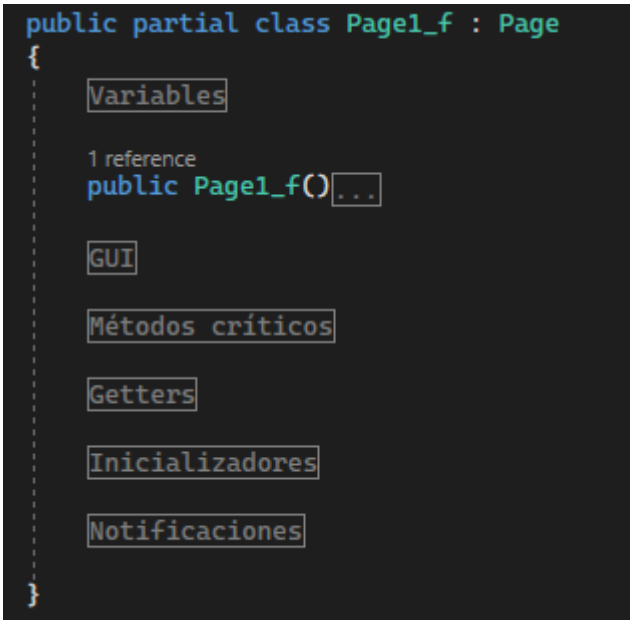
En esta región existen dos métodos, el primero GetSecret permite la devolución del código secreto utilizado, por otra parte, el segundo GetIdLevel permite devolver el id del nivel, es decir, si es el nivel fácil, medio o difícil.

**Segunda sección: Conexión de la lógica con la Interfaz**

***Descripción de la sección***

En esta sección se podrá observar como funciona la integración entre la parte lógica del programa y la interfaz que el usuario podrá observar e interactuar, estos son definidos por cada vista jugable del usuario, es decir, la conexión lógica del juego con la interfaz del nivel fácil, medio y difícil.

***Conexión lógico-gráfico sobre el nivel fácil***




Esta interfaz consiste en 7 regiones: Variables, Constructor, GUI, Métodos críticos, Getters, Inicializadores y Notificaciones; todas ellas permiten la operabilidad y jugabilidad del juego Mastermind.

**Región: Variables**

```
#region Variables

// Inicializamos el juego para usarse.
Mastermind game;
// Permite verificar en que intento se encuentra el usuario
int count_fila;
// Variable del puntaje
int score = 10000;

#endregion
```

Esta región consiste en todas las variables que el nivel fácil va a utilizar, estos son la creación de una variable game que inicialice el objeto Mastermind para luego poder generar el juego de nivel fácil, por otra parte, se tiene otra variable como count_fila que permite validar en que intento el jugador se encuentra y finalmente la variable score que permite verificar el puntaje del jugador.

**Región: Constructor**

```
1 reference
public Page1_f()
{
    InitializeComponent();
    // Crea el juego
    game = new Mastermind(1);
    // Inicializa los intentos
    count_fila = 0;
    // Inicializa la jugabilidad
    InicializarFilas();
    // Inicializa los puntajes
    InicializarPuntaje();
    // Notifica datos del modo de juego
    var thread = new Thread(() => { NotifyGameSettings(); });
    thread.Start();
}
```

1. El constructor permite inicializar todos los componentes de la interfaz que será hablado en la siguiente sección.
2. Por otra parte, se inicializa el juego Mastermind como nivel fácil mandando al constructor del objeto el código 1.
3. Después, se inicializa la variable count_fila para validar que el jugador se encuentre en el primer intento del juego.
4. A continuación, se inicializan las filas con el método InicializarFilas, este método será hablado a continuación a detalle a igual que InicializarPuntaje.
5. Finalmente se genera una notificación de que el juego ha empezado y que debe esperar del nivel, esto, se realiza en base a una metodología asincrónica.

**Región: GUI**

```
#region GUI

//evento de drag and drop
1 reference
private void Ellipse_MouseMove(object sender, MouseEventArgs e)...

// Pinta un circulo con el color deseado
40 references
private void Border_Drop(object sender, DragEventArgs e)...

// Vuelta al menú de selección de niveles
1 reference
private void Button_Click(object sender, RoutedEventArgs e)...

// Reinicia el nivel
1 reference
private void reboot_btn_Click(object sender, RoutedEventArgs e)...

// Permite visualizar las instrucciones del juego
1 reference
private void instrucciones_btn_Click(object sender, RoutedEventArgs e)...

// Confirmación de combinación
1 reference
private void confirmar_btn_Click(object sender, RoutedEventArgs e)...

#endregion
```

Esta región tiene todos los métodos que funcionan en base a eventos que ocurren con la interfaz.

***Método Ellipse_MouseMove***

```
//evento de drag and drop
1 reference
private void Ellipse_MouseMove(object sender, MouseEventArgs e)
{
    if (sender is Ellipse ellipse && e.LeftButton == MouseButtonState.Pressed)
    {
        DragDrop.DoDragDrop(ellipse, ellipse.Fill.ToString(), DragDropEffects.Copy);
    }
}
```

El método permite el evento drag y drop de los colores hacía los espacios para los intentos de combinaciones.

***Método Border_Drop***

```
// Pinta un circulo con el color deseado
40 references
private void Border_Drop(object sender, DragEventArgs e)
{
    if (sender is Border border)
    {
        if (e.Data.GetDataPresent(DataFormats.StringFormat))
        {
            string stringFromDrop = (string)e.Data.GetData(DataFormats.StringFormat);
            BrushConverter convertir = new BrushConverter();
            if (convertir.IsValid(stringFromDrop))
            {
                Brush pincel = (Brush)convertir.ConvertFromString(stringFromDrop);
                // Trae todos los colores utilizados en la fila actual
                string[] combination = GetCombination();
                // Establece el nuevo color del círculo
                border.Background = pincel;
                // Evalua si el color ya fue utilizado en la creación del código
                for (int i = 0; i < combination.Length; i++)
                {
                    if (combination[i] == border.Background.ToString())
                    {
                        // Si lo fue elimina la anterior posición del color y la coloca en la nueva
                        ChangePosition(i);
                    }
                }
            }
        }
    }
}
```

El método permite pintar un espacio de los intentos de combinación tras mover un color hacia ellos, sin embargo, al ser el nivel fácil, se debe tener en cuenta que no se permite la repetición de colores, es por ello por lo que en el siguiente código se valida lo siguiente:

```
// Trae todos los colores utilizados en la fila actual
string[] combination = GetCombination();
// Establece el nuevo color del círculo
border.Background = pincel;
// Evalua si el color ya fue utilizado en la creación del código
for (int i = 0; i < combination.Length; i++)
{
    if (combination[i] == border.Background.ToString())
    {
        // Si lo fue elimina la anterior posición del color y la coloca en la nueva
        ChangePosition(i);
    }
}
```

1. Primero se toma la combinación existente hasta el momento del jugador.
2. Se pinta el elemento deseado en la combinación.
3. Ahora, se verifica si este color pintado ya existía previamente en la combinación, si es así, se ejecuta el método ChangePosition (explicado más adelante), si no, únicamente se pinta el espacio deseado.

***Método Button_Click***

```
// Vuelta al menú de selección de niveles
1 reference
private void Button_Click(object sender, RoutedEventArgs e)
{
    NavigationService.Navigate(new SelectLevel());
}
```

Este método nos permite volver al menú de selección de niveles.

***Método reboot_btn_Click***

```
// Reinicia el nivel
1 reference
private void reboot_btn_Click(object sender, RoutedEventArgs e)
{
    Reboot();
}
```

Este método permite que cuando el jugador desee, el juego se reinicie y se establezcan todos los valores.

***Método instrucciones_btn_Click***

```
// Permite visualizar las instrucciones del juego
1 reference
private void instrucciones_btn_Click(object sender, RoutedEventArgs e)
{
    NotifyRules();
}
```

Este método permite que cuando el jugador desee, pueda visualizar las instrucciones y reglas del nivel y el propio juego.

***Método confirmar_btn_Click***

Este método consiste el más importante de todos ya que permite toda la lógica del juego y posible su propia jugabilidad, sin embargo, consiste en varias acciones que permiten lo antes mencionado.

1. Primero se crea una variable que almacenará la combinación del jugador.
2. Después, dada esta combinación, se transformarán todos los colores traídos como códigos hexadecimales de 8 dígitos a colores de 6 dígitos.

```
// Confirmación de combinación
1 reference
private void confirmar_btn_Click(object sender, RoutedEventArgs e)
{
    // Tomamos la combinación que se esta probando al momento.
    string[] combination = GetCombination();
    // Transformamos los colores a hexadecimal
    string[] combination_colors = GetCombinationHexColors(combination);
```

3. Después, se verifica si la combinación no tiene datos, es decir, si la combinación del usuario tiene todos los espacios de combinación llenos, si toda la combinación está vacía entonces se le notificará al usuario que esta esta vacía; por otro lado, si no estuviera vacía se ejecutaran los siguientes pasos:

```
    // Verificamos si la combinación no tiene datos
    if(!CheckEmpty(combination_colors))
    {
        // Verficamos que todos los colores han sido utilizados y no se han dejado espacios en blanco
        if (!CheckOneEmpty(combination_colors))...
        else...
    }
    else
    {
        NotifyEmpty();
    }
}
```

4. Una vez se evalúe que la combinación no este vacía, se verifica si existe algún espacio dejado en blanco de la propia combinación, si esto es así entonces se notifica que existe un propio espacio en blanco en la combinación; si es que no, entonces se ejecuta el paso 5.

```
// Verficamos que todos los colores han sido utilizados y no se han dejado espacios en blanco
if (!CheckOneEmpty(combination_colors))...
else
{
    NotifyOneEmpty();
}
```

5. Finalmente, sucede la operabilidad y jugabilidad del juego:
   5.1. Primero se toman todos los aciertos que el usuario logro con su combinación en la variable hits.
   5.2. Después, se configura la interfaz dedicada para los aciertos con los datos retomados en la variable hits, el método SetHits se hablará más adelante.
   5.3. A continuación, se suma un intento al identificador de intento actual incrementando la variable count_fila.

```
// Probamos la combinación
var hits = game.TryCombination(combination_colors);
// Configuramos los aciertos dados y los mostramos al usuario
SetHits(hits);
// Incrementamos el contador del intento
count_fila++;
```

   5.4. Después, se verifica si el usuario gano el juego, para esto se utiliza el método PlayerWon (igualmente hablado más adelante).
   5.5. Sí el jugador gano, se verifica si su puntaje obtenido fue mayor al mejor puntaje obtenido hasta la fecha, sí fue así entonces se notifica al usuario su acometido y se guarda su nuevo mejor puntaje. Sí no supero su mejor

puntaje o si lo hizo, de igual manera se notificará al usuario de que el jugador supero el nivel.

```
// Verificar si gano el juego el jugador
if (PlayerWon(hits))
{
    if (CheckNewMaxScore())
    {
        NotifyMaxScore(Int32.Parse(puntaje_actual.Text));
        SetMaxScore();
    }
    NotifyWon(Int32.Parse(puntaje_actual.Text));
}
```

5.6. Después, haya ganado o no, se actualiza el puntaje actual del jugador y se lo representa en su ambiente gráfico.

```
// Actualiza el score del jugador
score = score - 1000;
puntaje_actual.Text = score.ToString();
```

5.7. A continuación, se verifica si el usuario perdió el juego, si es así entonces se le notifica al usuario, pero se le da la oportunidad de volver a intentar superar el nivel.

5.8. Finalmente se inicializan las filas que pueden ser modificadas, el método InicializarFilas como se mencionó anteriormente será explicado más adelante.

```
// Verifica si perdio el juego el jugador
if (PlayerLost() & !PlayerWon(hits))
{
    NotifyLost();
}
// Actualiza las filas que pueden ser modificadas
if (count_fila < 10) InicializarFilas();
```

**Región: Métodos Críticos**

Esta región contiene todos los métodos críticos para el funcionamiento y jugabilidad del juego, consiste en los siguientes métodos:

```
#region Métodos críticos

// Verifica si el jugador gano
2 references
private bool PlayerWon(int[] hits)...

// Verifica si el jugador perdio
1 reference
private bool PlayerLost()...

// Reinicia el nivel
2 references
private void Reboot()...

// Hace que una posición  se inicialice
1 reference
private void ChangePosition(int position)...

// Verifica si la combinación esta vacía
1 reference
private bool CheckEmpty(string[] colors)...

// Verifica que no existan items en blanco para las combinaciones de los intentos
1 reference
private bool CheckOneEmpty(string[] colors)...

// Establece el puntaje máximo
1 reference
private void SetMaxScore()...

// Evalua si se logro un nuevo marcador
1 reference
private bool CheckNewMaxScore()...

#endregion
```

***PlayerWon***

```
// Verifica si el jugador gano
2 references
private bool PlayerWon(int[] hits)
{
    foreach (var iter in hits) if (iter != 2) return false;
    return true;
}
```

Verifica si el jugador paso el nivel.

***PlayerLost***

```
// Verifica si el jugador perdio
1 reference
private bool PlayerLost()
{
    if (count_fila > 9) return true;
    return false;
}
```

Verifica si el jugador perdió el nivel.

***Reboot***

```
// Reinicia el nivel
2 references
private void Reboot()
{
    // Establece un nuevo código secreto
    if(count_fila > 0) game.InitSecret();
    // Reestablece los intentos
    count_fila = 0;
    // Reestablece el puntaje actual
    score = 10000;
    // Inicializa la disponibilidad de las filas
    InicializarFilas();
    // Reinicia todos los datos del juego actual
    ReiniciarFilas();
}
```

Reinicia el nivel, es decir, reestablece un nuevo código secreto, el id del intento que se encuentra el jugador, el puntaje inicial e inicializa filas y las reinicia (ambos métodos serán hablados más adelante).

***ChangePosition***

Este método consiste en el cambio de posición de un color dentro de la combinación de colores que el jugador este realizando, lo que hace es establecer un color blanco en la anterior posición del color deseado. Sin embargo, es importante mencionar que la modificación de cada combinación se da por el id del intento en el que se encuentre el jugador.

```
// Hace que una posición  se inicialice
1 reference
private void ChangePosition(int position)
{
    switch (count_fila)
    {
        case (0):
            if (position == 0) C0_F9.Background = new SolidColorBrush(Colors.White);
            else if (position == 1) C1_F9.Background = new SolidColorBrush(Colors.White);
            else if (position == 2) C2_F9.Background = new SolidColorBrush(Colors.White);
            else if(position == 3) C3_F9.Background = new SolidColorBrush(Colors.White);
            break;
        case (1):
            if (position == 0) C0_F8.Background = new SolidColorBrush(Colors.White);
            else if (position == 1) C1_F8.Background = new SolidColorBrush(Colors.White);
            else if (position == 2) C2_F8.Background = new SolidColorBrush(Colors.White);
            else if (position == 3) C3_F8.Background = new SolidColorBrush(Colors.White);
            break;
        case (2):
            if (position == 0) C0_F7.Background = new SolidColorBrush(Colors.White);
            else if (position == 1) C1_F7.Background = new SolidColorBrush(Colors.White);
            else if (position == 2) C2_F7.Background = new SolidColorBrush(Colors.White);
            else if (position == 3) C3_F7.Background = new SolidColorBrush(Colors.White);
            break;
        case (3):
            if (position == 0) C0_F6.Background = new SolidColorBrush(Colors.White);
            else if (position == 1) C1_F6.Background = new SolidColorBrush(Colors.White);
            else if (position == 2) C2_F6.Background = new SolidColorBrush(Colors.White);
            else if (position == 3) C3_F6.Background = new SolidColorBrush(Colors.White);
            break;
```

```
    case (4):
        if (position == 0) C0_F5.Background = new SolidColorBrush(Colors.White);
        else if (position == 1) C1_F5.Background = new SolidColorBrush(Colors.White);
        else if (position == 2) C2_F5.Background = new SolidColorBrush(Colors.White);
        else if (position == 3) C3_F5.Background = new SolidColorBrush(Colors.White);
        break;
    case (5):
        if (position == 0) C0_F4.Background = new SolidColorBrush(Colors.White);
        else if (position == 1) C1_F4.Background = new SolidColorBrush(Colors.White);
        else if (position == 2) C2_F4.Background = new SolidColorBrush(Colors.White);
        else if (position == 3) C3_F4.Background = new SolidColorBrush(Colors.White);
        break;
    case (6):
        if (position == 0) C0_F3.Background = new SolidColorBrush(Colors.White);
        else if (position == 1) C1_F3.Background = new SolidColorBrush(Colors.White);
        else if (position == 2) C2_F3.Background = new SolidColorBrush(Colors.White);
        else if (position == 3) C3_F3.Background = new SolidColorBrush(Colors.White);
        break;
    case (7):
        if (position == 0) C0_F2.Background = new SolidColorBrush(Colors.White);
        else if (position == 1) C1_F2.Background = new SolidColorBrush(Colors.White);
        else if (position == 2) C2_F2.Background = new SolidColorBrush(Colors.White);
        else if (position == 3) C3_F2.Background = new SolidColorBrush(Colors.White);
        break;
    case (8):
        if (position == 0) C0_F1.Background = new SolidColorBrush(Colors.White);
        else if (position == 1) C1_F1.Background = new SolidColorBrush(Colors.White);
        else if (position == 2) C2_F1.Background = new SolidColorBrush(Colors.White);
        else if (position == 3) C3_F1.Background = new SolidColorBrush(Colors.White);
        break;
```

```
        case (9):
            if (position == 0) C0_F0.Background = new SolidColorBrush(Colors.White);
            else if (position == 1) C1_F0.Background = new SolidColorBrush(Colors.White);
            else if (position == 2) C2_F0.Background = new SolidColorBrush(Colors.White);
            else if (position == 3) C3_F0.Background = new SolidColorBrush(Colors.White);
            break;
    }
}
```

***CheckEmpty***

```
// Verifica si la combinación esta vacía
1 reference
private bool CheckEmpty(string[] colors)
{
    foreach (var iter in colors) if (iter != "#FFFFFF") return false;
    return true;
}
```

El método verifica si la combinación entera está vacía.

***CheckOneEmpty***

```
// Verifica que no existan items en blanco para las combinaciones de los intentos
1 reference
private bool CheckOneEmpty(string[] colors)
{
    foreach (var iter in colors) if (iter == "#FFFFFF") return true;
    return false;
}
```

El método verifica si la combinación tiene una casilla vacía.

***SetMaxScore***

Este método permita el guardado y establecimiento del nuevo mejor puntaje.

1. Para empezar, se define una variable id que toma el id del nivel que se está jugando.
2. Después, se crea una variable que tendrá el nombre del archivo de texto que guardará todos los registros de los mejores puntajes de cada nivel.
3. Después, se crea una variable que almacenará temporalmente todos los mejores puntajes de todos niveles.
4. Finalmente, se creará una variable contadora count como 0.

```
// Establece el puntaje máximo
1 reference
private void SetMaxScore()
{
    // Identifica que nivel se esta jugando
    var id = game.GetIdLevel();
    // Variables para modificar el archivo de puntajes
    string filename = "scores.txt";
    string[] scores = new string[3];
    int count = 0;
```

5. A continuación, se leen los datos del archivo “scores.txt” y se guardan estos datos en la variable scores.

```
// Lee los datos y guarda en scores
var linesRead = File.ReadLines("scores.txt");
foreach (var iter in linesRead) scores[count++] = iter;
```

6. Después, se borra el archivo “scores.txt”
7. A continuación, dado el id del nivel se establece el nuevo mejor puntaje en el id del score.
8. Finalmente, se guardan todos los datos en el archivo “scores.txt” creado nuevamente.

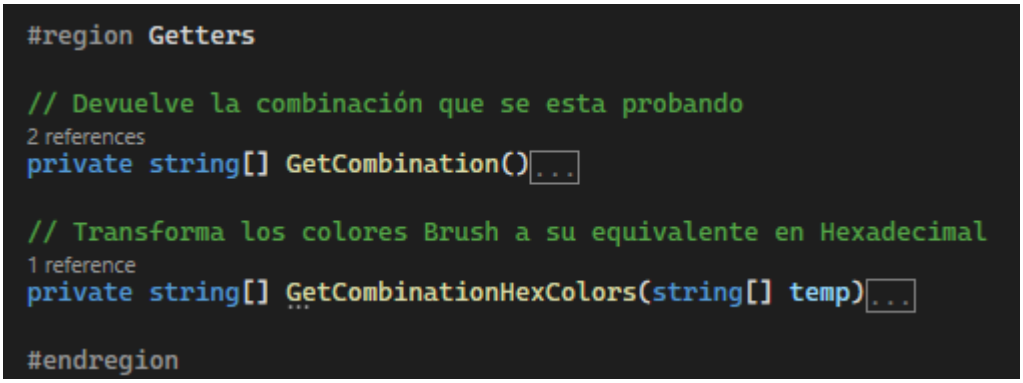
```
    // Borra el archivo
    if (File.Exists(filename)) File.Delete(filename);

    // Establece la nueva información
    if (id == 1) scores[0] = score.ToString();
    else if(id == 2) scores[1] = score.ToString();
    else if(id == 3) scores[2] = score.ToString();

    // Guarda la nueva información
    using (FileStream fs = File.Create(filename))
    {
        Byte[] score_1 = new UTF8Encoding(true).GetBytes(scores[0] + "\n");
        Byte[] score_2 = new UTF8Encoding(true).GetBytes(scores[1] + "\n");
        Byte[] score_3 = new UTF8Encoding(true).GetBytes(scores[2] + "\n");
        fs.Write(score_1,0,score_1.Length);
        fs.Write(score_2, 0, score_2.Length);
        fs.Write(score_3, 0, score_3.Length);
    }
}
```

***CheckNewMaxScore***

```
// Evalua si se logro un nuevo marcador

private bool CheckNewMaxScore()
{
    if (score > Int32.Parse(puntaje_maximo.Text)) return true;
    return false;
}
```

El método permite verificar si se supero el anterior mejor puntaje.

**Región: Getters**




Esta región contiene métodos que permiten retomar datos de la interfaz o de alguna lógica.

***GetCombination***

```
// Devuelve la combinación que se esta probando
2 references
private string[] GetCombination()
{
    string[] temp = new string[4];
    // El número del case + 1 indica la fila que se esta probando.
    switch (count_fila)
    {
        case (0):
            temp[0] = C0_F9.Background.ToString();
            temp[1] = C1_F9.Background.ToString();
            temp[2] = C2_F9.Background.ToString();
            temp[3] = C3_F9.Background.ToString();
            break;
        case (1):
            temp[0] = C0_F8.Background.ToString();
            temp[1] = C1_F8.Background.ToString();
            temp[2] = C2_F8.Background.ToString();
            temp[3] = C3_F8.Background.ToString();
            break;
        case (2):
            temp[0] = C0_F7.Background.ToString();
            temp[1] = C1_F7.Background.ToString();
            temp[2] = C2_F7.Background.ToString();
            temp[3] = C3_F7.Background.ToString();
            break;
        case (3):
            temp[0] = C0_F6.Background.ToString();
            temp[1] = C1_F6.Background.ToString();
            temp[2] = C2_F6.Background.ToString();
            temp[3] = C3_F6.Background.ToString();
            break;
        case (4):
            temp[0] = C0_F5.Background.ToString();
            temp[1] = C1_F5.Background.ToString();
            temp[2] = C2_F5.Background.ToString();
            temp[3] = C3_F5.Background.ToString();
            break;
        case (5):
            temp[0] = C0_F4.Background.ToString();
            temp[1] = C1_F4.Background.ToString();
            temp[2] = C2_F4.Background.ToString();
            temp[3] = C3_F4.Background.ToString();
            break;
        case (6):
            temp[0] = C0_F3.Background.ToString();
            temp[1] = C1_F3.Background.ToString();
            temp[2] = C2_F3.Background.ToString();
            temp[3] = C3_F3.Background.ToString();
            break;
        case (7):
            temp[0] = C0_F2.Background.ToString();
            temp[1] = C1_F2.Background.ToString();
            temp[2] = C2_F2.Background.ToString();
            temp[3] = C3_F2.Background.ToString();
            break;
```

```
        case (8):
            temp[0] = C0_F1.Background.ToString();
            temp[1] = C1_F1.Background.ToString();
            temp[2] = C2_F1.Background.ToString();
            temp[3] = C3_F1.Background.ToString();
            break;
        case (9):
            temp[0] = C0_F0.Background.ToString();
            temp[1] = C1_F0.Background.ToString();
            temp[2] = C2_F0.Background.ToString();
            temp[3] = C3_F0.Background.ToString();
            break;
    }
    return temp;
}
```

El método devuelve la combinación de colores utilizadas por el jugador tras realizar su intento, esto se da en base al id del intento que se encuentre el jugador. Sin embargo, los colores devuletos se basan en códigos hexadecimales de 8 dígitos.

***GetCombinationHexColors***

```
// Transforma los colores Brush a su equivalente en Hexadecimal
1 reference
private string[] GetCombinationHexColors(string[] temp)
{
    for (int i = 0; i < temp.Length; i++)
    {
        if (temp[i] == "#FFFFFFFF") temp[i] = "#FFFFFF"; // Conversión del blanco
        else if (temp[i] == "#00000000") temp[i] = "#000000"; // Conversión del negro
        else if (temp[i] == "#FFFF0000") temp[i] = "#FF0000"; // Conversión del color rojo
        else if (temp[i] == "#FFFFA500") temp[i] = "#FFA500"; // Conversión del color amarillo
        else if (temp[i] == "#FF800080") temp[i] = "#800080"; // Conversión del color morado
        else if (temp[i] == "#FF9ACD32") temp[i] = "#9ACD32"; // Conversión del color verde
    }
    return temp;
}
```

El método permite la transformación de un arreglo de colores con códigos hexadecimales de 8 dígitos a un arreglo de colores con códigos hexadecimales de 6 dígitos. Permite la conversión de 6 colores.

**Región: Inicializadores**

```
#region Inicializadores

// Permite la visualización de los aciertos dados al usuario

private void SetHits(int[] hits)...

// Inicializa las filas que pueden ser modificadas
3 references
private void InicializarFilas()...

// Reinicia todas las filas a su valor inicial -> en blanco

private void ReiniciarFilas()...

// Inicializa el puntaje inicial

private void InicializarPuntaje()...

// Trae puntajes máximos existentes

private string InitMaxScore()...

#endregion
```

La región de inicializadores contiene métodos que permiten la propia inicialización de varias partes del programa.

***SetHits***

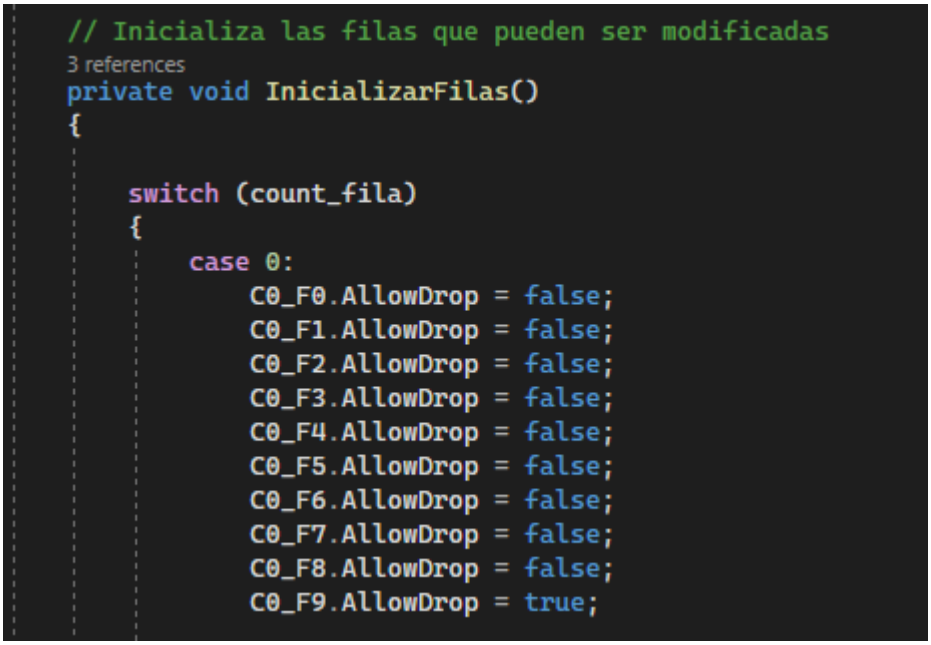
El método presente permite hacer que la interfaz gráfica que permite la visualización de los aciertos del jugador se establezca en base a la combinación dada, un color negro si hay un ítem en la posición correcta, un color gris si hay un ítem en la posición incorrecta y blanco si no existe ítems acertados.

```
// Permite la visualización de los aciertos dados al usuario
1 reference
private void SetHits(int[] hits)
{
    // Coloca color negro si hay un item en la posición correcta.
    // Coloca color gris si hay un item en la posición incorrecta.
    // Deja en blanco si no existe items acertados.
    switch (count_fila)
    {
        case (0):
            // Primer circulo
            if (hits[0] == 2) A1_F10.Background = new SolidColorBrush(Colors.Black);
            else if (hits[0] == 1) A1_F10.Background = new SolidColorBrush(Colors.Gray);
            // Segundo circulo
            if (hits[1] == 2) A2_F10.Background = new SolidColorBrush(Colors.Black);
            else if (hits[1] == 1) A2_F10.Background = new SolidColorBrush(Colors.Gray);
            // Tercer circulo
            if (hits[2] == 2) A3_F10.Background = new SolidColorBrush(Colors.Black);
            else if (hits[3] == 1) A3_F10.Background = new SolidColorBrush(Colors.Gray);
            // Cuarto circulo
            if (hits[3] == 2) A4_F10.Background = new SolidColorBrush(Colors.Black);
            else if (hits[3] == 1) A4_F10.Background = new SolidColorBrush(Colors.Gray);
            break;
```

(Nota: El mismo patrón funciona para cada intento, para más información visualizar el código fuente del proyecto)

***InicializarFilas***

El método presente configura y establece que filas de combinaciones pueden ser modificadas, es decir, a que filas podemos agregar colores para sus combinaciones.



```
// Inicializa las filas que pueden ser modificadas
3 references
private void InicializarFilas()
{

    switch (count_fila)
    {
        case 0:
            C0_F0.AllowDrop = false;
            C0_F1.AllowDrop = false;
            C0_F2.AllowDrop = false;
            C0_F3.AllowDrop = false;
            C0_F4.AllowDrop = false;
            C0_F5.AllowDrop = false;
            C0_F6.AllowDrop = false;
            C0_F7.AllowDrop = false;
            C0_F8.AllowDrop = false;
            C0_F9.AllowDrop = true;
```

(Nota: El mismo patrón funciona para cada fila, para más información visualizar el código fuente del proyecto)

***ReiniciarFilas***

El método permite que todas las filas del juego donde se colocarán y estaban colocadas las combinaciones se reestablezcan a blanco.

```
// Reinicia todas las filas a su valor inicial -> en blanco
1 reference
private void ReiniciarFilas()
{
    // Primeras columnas de todas las filas
    C0_F0.Background = new SolidColorBrush(Colors.White);
    C0_F1.Background = new SolidColorBrush(Colors.White);
    C0_F2.Background = new SolidColorBrush(Colors.White);
    C0_F3.Background = new SolidColorBrush(Colors.White);
    C0_F4.Background = new SolidColorBrush(Colors.White);
    C0_F5.Background = new SolidColorBrush(Colors.White);
    C0_F6.Background = new SolidColorBrush(Colors.White);
    C0_F7.Background = new SolidColorBrush(Colors.White);
    C0_F8.Background = new SolidColorBrush(Colors.White);
    C0_F9.Background = new SolidColorBrush(Colors.White);
```

(Nota: El mismo patrón funciona para cada fila, para más información visualizar el código fuente del proyecto)

***InicializarPuntaje***

```
// Inicializa el puntaje inicial
1 reference
private void InicializarPuntaje()
{
    puntaje_actual.Text = score.ToString();
    puntaje_maximo.Text = InitMaxScore();
    // Identifica que nivel se esta jugando
    var id = game.GetIdLevel();
}
```

El método permite inicializar los puntajes, un puntaje actual del jugador y el mejor puntaje hasta la fecha.

***InitMaxScore***

```
// Trae puntajes máximos existentes
1 reference
private string InitMaxScore()
{
    string curFile = @"scores.txt";
    // Verifica si existen datos guardados
    if (File.Exists(curFile))
    {
        string[] scores = new string[3];
        int count = 0;
        var linesRead = File.ReadLines("scores.txt");
        foreach(var iter in linesRead) scores[count++] = iter;
        if (game.GetIdLevel() == 1) return scores[0];
        else if(game.GetIdLevel() == 2) return scores[1];
        else if(game.GetIdLevel() == 3) return scores[2];
    }
    else
    {
        using(FileStream fs = File.Create(curFile))
        {
            Byte[] temp_score = new UTF8Encoding(true).GetBytes("0\n");
            fs.Write(temp_score, 0, temp_score.Length);
            fs.Write(temp_score, 0, temp_score.Length);
            fs.Write(temp_score, 0, temp_score.Length);
        }
    }
    return "0";
}
```

El método permite verificar si existe el archivo “scores.txt” si lo hace retorna el valor del mejor puntaje hasta la fecha dado el id del nivel, sin embargo, en caso de no que no existe, se crea el archivo e inicializa todos los mejores puntajes como 0.

**Región: Notificaciones**

```
#region Notificaciones

// Muestra un mensaje de que el jugador gano
1 reference
private void NotifyWon(int p_score)...

// Muestra un mensaje de que el jugador perdio
1 reference
private void NotifyLost()...

// Muestra las configuraciones del nivel
1 reference
private void NotifyGameSettings()...

// Notifica que un color ha sido repetido
0 references
private void NotifyRepeatedColor()...

// Notifica si la combinación fue ejecutada vacía
1 reference
private void NotifyEmpty()...

// Notifica que hay un item en blanco en la combinación
1 reference
private void NotifyOneEmpty()...

// Notifica que se logro un nuevo record
1 reference
private void NotifyMaxScore(int p_score)...

// Notifica las instrucciones del juego
1 reference
private void NotifyRules()...

#endregion
```

Esta región se compone por métodos que notifican al usuario de diversas situaciones que suceden durante la operabilidad y jugabilidad del juego.

***NotifyWon***

```
// Muestra un mensaje de que el jugador gano
1 reference
private void NotifyWon(int p_score)
{
    string message = $"¡Felicitaciones!\n\n ¡Has pasado el nivel! Tú puntaje fue de {p_score}.";
    string caption = "¡Has ganado!";
    MessageBoxButton button = MessageBoxButton.OK;
    MessageBoxResult result = MessageBox.Show(message, caption, button);
    switch (result)
    {
        // Vuelve al menú de niveles
        case MessageBoxResult.OK:
            NavigationService.Navigate(new SelectLevel());
            break;
    }
}
```

El método notifica si el jugador supero el nivel.

### *NotifyLost*

```
// Muestra un mensaje de que el jugador perdio
1 reference
private void NotifyLost()
{
    string message = "¡Oh No! No has logrado superar el nivel...\n\n¿Deseas intentar de nuevo?";
    string caption = "¡Perdiste!";
    MessageBoxButton button = MessageBoxButton.YesNo;
    MessageBoxResult result = MessageBox.Show(message, caption, button);
    switch (result)
    {
        // Reinicia el nivel
        case MessageBoxResult.Yes:
            Reboot();
            break;

        // Vuelve al menú de niveles
        case MessageBoxResult.No:
            NavigationService.Navigate(new SelectLevel());
            break;
    }
}
```

El método notifica al jugar si este ha perdido el nivel y le da la opción de intentar nuevamente.

### *NotifyGameSettings*

```
// Muestra las configuraciones del nivel
1 reference
private void NotifyGameSettings()
{
    string message = "¡Bienvenido!\n\nHas seleccionado el nivel de dificultad fácil, tienes 4 colores para crear tus combinaciones
    string caption = "Nivel: Fácil";
    MessageBoxButton button = MessageBoxButton.OK;
    MessageBoxResult result = MessageBox.Show(message, caption, button);
    switch (result)
    {
        // No ocurre nada
        case MessageBoxResult.OK:
            break;
    }
}
```

El método notifica al jugar instrucciones y modo de juego del nivel en el que se encuentra.

### *NotifyRepeatedColor*

```
// Notifica que un color ha sido repetido
0 references
private void NotifyRepeatedColor()
{
    string message = "¡Has repetido un color!\n¡Recuerda que en este nivel no esta permitido su repetición!";
    string caption = "Advertencia";
    MessageBoxButton button = MessageBoxButton.OK;
    MessageBoxResult result = MessageBox.Show(message, caption, button);
    switch (result)
    {
        // No sucede nada
        case MessageBoxResult.OK:
            break;
    }
}
```

El método notifica al jugador si existe un color repetido en la combinación realizada.

### *NotifyEmpty*

```
// Notifica si la combinación fue ejecutada vacía
1 reference
private void NotifyEmpty()
{
    string message = "Parece que no has colocado ningún tipo de combinación...\n\n¡Coloca alguna antes de probar!";
    string caption = "Advertencia";
    MessageBoxButton button = MessageBoxButton.OK;
    MessageBoxResult result = MessageBox.Show(message, caption, button);
}
```

El método notifica al jugador si la combinación realizada se encuentra vacía.

### *NotifyOneEmpty*

```
// Notifica que hay un item en blanco en la combinación
1 reference
private void NotifyOneEmpty()
{
    string message = "Parece que has dejado un espacio en blanco...\n\n¡Ten más cuidado la próxima vez!";
    string caption = "Advertencia";
    MessageBoxButton button = MessageBoxButton.OK;
    MessageBoxResult result = MessageBox.Show(message, caption, button);
}
```

El método notifica al jugador si en la combinación realizada se encuentra un ítem vacío.

### *NotifyMaxScore*

```
// Notifica que se logro un nuevo record
1 reference
private void NotifyMaxScore(int p_score)
{
    string message = $"¡Has logrado un nuevo record!\n\nObtuviste un puntaje de: {p_score}";
    string caption = "Felicitaciones";
    MessageBoxButton button = MessageBoxButton.OK;
    MessageBoxResult result = MessageBox.Show(message, caption, button);
}
```

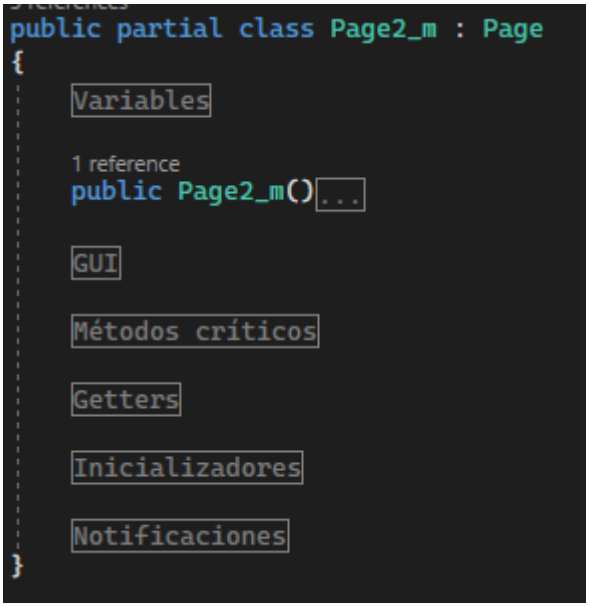
El método notifica al jugar si el jugador supero su anterior mejor puntaje.

### *NotifyRules*

```
// Notifica las instrucciones del juego
1 reference
private void NotifyRules()
{
    string message = "¡Recuerda!\n\nCada vez que se empieza una partida, el juego creará un nuevo código secreto que lo debes de
        "\n\n**********************************************************************\n\n" +
        "Sobre el juego:\n\nPara poder acertar con mayor facilidad el código secreto ¡podrás guiarte de las pistas que el juego
        "1. El color negro significa que tienes un color en la posición correcta!\n" +
        "2. El color gris significa que tienes un color correcta pero en la posición incorrecta!";
    string caption = "Reglas del juego";
    MessageBoxButton buttons = MessageBoxButton.OK;
    MessageBoxResult result = MessageBox.Show(message, caption, buttons);
}
```

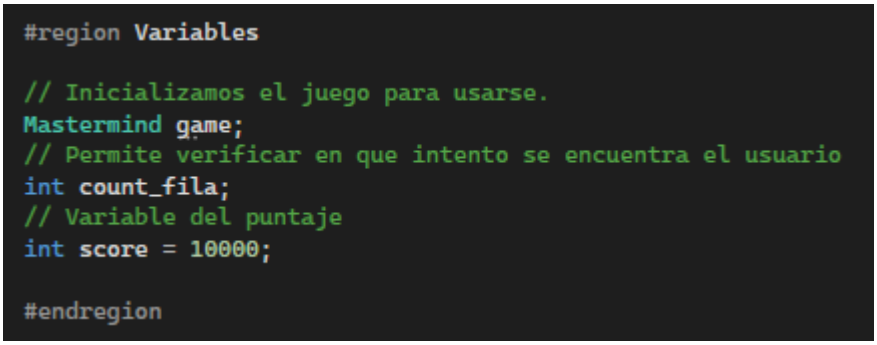
El método notifica las reglas del juego al jugador.

***Conexión lógico-gráfico sobre el nivel medio***




Esta interfaz consiste en 7 regiones: Variables, Constructor, GUI, Métodos críticos, Getters, Inicializadores y Notificaciones; todas ellas permiten la operabilidad y jugabilidad del juego Mastermind.

**Región: Variables**




Esta región consiste en todas las variables que el nivel fácil va a utilizar, estos son la creación de una variable game que inicialice el objeto Mastermind para luego poder generar el juego de nivel fácil, por otra parte, se tiene otra variable como count_fila que permite validar en que intento el jugador se encuentra y finalmente la variable score que permite verificar el puntaje del jugador.

**Región: Constructor**

```
1 reference
public Page2_m()
{
    InitializeComponent();
    // Crea el juego
    game = new Mastermind(2);
    // Inicializa los intentos
    count_fila = 0;
    // Inicializa la jugabilidad
    InicializarFilas();
    // Inicializa los puntajes
    InicializarPuntaje();
    // Notifica datos del modo de juego
    var thread = new Thread(() => { NotifyGameSettings(); });
    thread.Start();
}
```

6. El constructor permite inicializar todos los componentes de la interfaz que será hablado en la siguiente sección.
7. Por otra parte, se inicializa el juego Mastermind como nivel medio mandando al constructor del objeto el código 2.
8. Después, se inicializa la variable count_fila para validar que el jugador se encuentre en el primer intento del juego.
9. A continuación, se inicializan las filas con el método InicializarFilas, este método será hablado a continuación a detalle a igual que InicializarPuntaje.
10. Finalmente se genera una notificación de que el juego ha empezado y que debe esperar del nivel, esto, se realiza en base a una metodología asincrónica.

**Región: GUI**

```
#region GUI

//evento de drag and drop
1 reference
private void Ellipse_MouseMove(object sender, MouseEventArgs e)...

// Pinta un circulo con el color deseado
40 references
private void Border_Drop(object sender, DragEventArgs e)...

// Vuelta al menú de selección de niveles
1 reference
private void Button_Click(object sender, RoutedEventArgs e)...

// Reinicia el nivel
1 reference
private void reboot_btn_Click(object sender, RoutedEventArgs e)...

// Permite visualizar las instrucciones del juego
1 reference
private void instrucciones_btn_Click(object sender, RoutedEventArgs e)...

// Confirmación de combinación
1 reference
private void confirmar_btn_Click(object sender, RoutedEventArgs e)...

#endregion
```

Esta región tiene todos los métodos que funcionan en base a eventos que ocurren con la interfaz.

***Método Ellipse_MouseMove***

```
//evento de drag and drop
1 reference
private void Ellipse_MouseMove(object sender, MouseEventArgs e)
{
    if (sender is Ellipse ellipse && e.LeftButton == MouseButtonState.Pressed)
    {
        DragDrop.DoDragDrop(ellipse, ellipse.Fill.ToString(), DragDropEffects.Copy);
    }
}
```

El método permite el evento drag y drop de los colores hacía los espacios para los intentos de combinaciones.

***Método Border_Drop***

```
// Pinta un circulo con el color deseado
40 references
private void Border_Drop(object sender, DragEventArgs e)
{
    if (sender is Border border)
    {
        if (e.Data.GetDataPresent(DataFormats.StringFormat))
        {
            string stringFromDrop = (string)e.Data.GetData(DataFormats.StringFormat);
            BrushConverter convertir = new BrushConverter();
            if (convertir.IsValid(stringFromDrop))
            {
                Brush pincel = (Brush)convertir.ConvertFromString(stringFromDrop);
                // Trae todos los colores utilizados en la fila actual
                string[] combination = GetCombination();
                // Establece el nuevo color del círculo
                border.Background = pincel;
                // Evalua si el color ya fue utilizado en la creación del código
                for (int i = 0; i < combination.Length; i++)
                {
                    if (combination[i] == border.Background.ToString())
                    {
                        // Si lo fue elimina la anterior posición del color y la coloca en la nueva
                        ChangePosition(i);
                    }
                }
            }
        }
    }
}
```

El método permite pintar un espacio de los intentos de combinación tras mover un color hacia ellos, sin embargo, al ser el nivel medio, se debe tener en cuenta que no se permite la repetición de colores, es por ello por lo que en el siguiente código se valida lo siguiente:

```
// Trae todos los colores utilizados en la fila actual
string[] combination = GetCombination();
// Establece el nuevo color del círculo
border.Background = pincel;
// Evalua si el color ya fue utilizado en la creación del código
for (int i = 0; i < combination.Length; i++)
{
    if (combination[i] == border.Background.ToString())
    {
        // Si lo fue elimina la anterior posición del color y la coloca en la nueva
        ChangePosition(i);
    }
}
```

4. Primero se toma la combinación existente hasta el momento del jugador.
5. Se pinta el elemento deseado en la combinación.
6. Ahora, se verifica si este color pintado ya existía previamente en la combinación, si es así, se ejecuta el método ChangePosition (explicado más adelante), si no, únicamente se pinta el espacio deseado.

***Método Button_Click***

```
// Vuelta al menú de selección de niveles
1 reference
private void Button_Click(object sender, RoutedEventArgs e)
{
    NavigationService.Navigate(new SelectLevel());
}
```

Este método nos permite volver al menú de selección de niveles.

***Método reboot_btn_Click***

```
// Reinicia el nivel
1 reference
private void reboot_btn_Click(object sender, RoutedEventArgs e)
{
    Reboot();
}
```

Este método permite que cuando el jugador desee, el juego se reinicie y se establezcan todos los valores.

***Método instrucciones_btn_Click***

```
// Permite visualizar las instrucciones del juego
1 reference
private void instrucciones_btn_Click(object sender, RoutedEventArgs e)
{
    NotifyRules();
}
```

Este método permite que cuando el jugador desee, pueda visualizar las instrucciones y reglas del nivel y el propio juego.

***Método confirmar_btn_Click***

Este método consiste el más importante de todos ya que permite toda la lógica del juego y posible su propia jugabilidad, sin embargo, consiste en varias acciones que permiten lo antes mencionado.

6. Primero se crea una variable que almacenará la combinación del jugador.
7. Después, dada esta combinación, se transformarán todos los colores traídos como códigos hexadecimales de 8 dígitos a colores de 6 dígitos.

```
// Confirmación de combinación
1 reference
private void confirmar_btn_Click(object sender, RoutedEventArgs e)
{
    // Tomamos la combinación que se esta probando al momento.
    string[] combination = GetCombination();
    // Transformamos los colores a hexadecimal
    string[] combination_colors = GetCombinationHexColors(combination);
```

8. Después, se verifica si la combinación no tiene datos, es decir, si la combinación del usuario tiene todos los espacios de combinación llenos, si toda la combinación está vacía entonces se le notificará al usuario que esta esta vacía; por otro lado, si no estuviera vacía se ejecutaran los siguientes pasos:

```
    // Verificamos si la combinación no tiene datos
    if(!CheckEmpty(combination_colors))
    {
        // Verficamos que todos los colores han sido utilizados y no se han dejado espacios en blanco
        if (!CheckOneEmpty(combination_colors))...
        else...
    }
    else
    {
        NotifyEmpty();
    }
}
```

9. Una vez se evalúe que la combinación no este vacía, se verifica si existe algún espacio dejado en blanco de la propia combinación, si esto es así entonces se notifica que existe un propio espacio en blanco en la combinación; si es que no, entonces se ejecuta el paso 5.

```
// Verficamos que todos los colores han sido utilizados y no se han dejado espacios en blanco
if (!CheckOneEmpty(combination_colors))...
else
{
    NotifyOneEmpty();
}
```

10. Finalmente, sucede la operabilidad y jugabilidad del juego:
    10.1. Primero se toman todos los aciertos que el usuario logro con su combinación en la variable hits.
    10.2. Después, se configura la interfaz dedicada para los aciertos con los datos retomados en la variable hits, el método SetHits se hablará más adelante.
    10.3. A continuación, se suma un intento al identificador de intento actual incrementando la variable count_fila.

```
// Probamos la combinación
var hits = game.TryCombination(combination_colors);
// Configuramos los aciertos dados y los mostramos al usuario
SetHits(hits);
// Incrementamos el contador del intento
count_fila++;
```

    10.4. Después, se verifica si el usuario gano el juego, para esto se utiliza el método PlayerWon (igualmente hablado más adelante).
    10.5. Sí el jugador gano, se verifica si su puntaje obtenido fue mayor al mejor puntaje obtenido hasta la fecha, sí fue así entonces se notifica al usuario su acometido y se guarda su nuevo mejor puntaje. Sí no supero su mejor

puntaje o si lo hizo, de igual manera se notificará al usuario de que el jugador supero el nivel.

```
// Verificar si gano el juego el jugador
if (PlayerWon(hits))
{
    if (CheckNewMaxScore())
    {
        NotifyMaxScore(Int32.Parse(puntaje_actual.Text));
        SetMaxScore();
    }
    NotifyWon(Int32.Parse(puntaje_actual.Text));
}
```

10.6. Después, haya ganado o no, se actualiza el puntaje actual del jugador y se lo representa en su ambiente gráfico.

```
// Actualiza el score del jugador
score = score - 1000;
puntaje_actual.Text = score.ToString();
```

10.7. A continuación, se verifica si el usuario perdió el juego, si es así entonces se le notifica al usuario, pero se le da la oportunidad de volver a intentar superar el nivel.

10.8. Finalmente se inicializan las filas que pueden ser modificadas, el método InicializarFilas como se mencionó anteriormente será explicado más adelante.

```
// Verifica si perdio el juego el jugador
if (PlayerLost() & !PlayerWon(hits))
{
    NotifyLost();
}
// Actualiza las filas que pueden ser modificadas
if (count_fila < 10) InicializarFilas();
```

**Región: Métodos Críticos**

Esta región contiene todos los métodos críticos para el funcionamiento y jugabilidad del juego, consiste en los siguientes métodos:

```
#region Métodos críticos

// Verifica si el jugador gano
2 references
private bool PlayerWon(int[] hits)...

// Verifica si el jugador perdio
1 reference
private bool PlayerLost()...

// Reinicia el nivel
2 references
private void Reboot()...

// Hace que una posición  se inicialice
1 reference
private void ChangePosition(int position)...

// Verifica si la combinación esta vacía
1 reference
private bool CheckEmpty(string[] colors)...

// Verifica que no existan items en blanco para las combinaciones de los intentos
1 reference
private bool CheckOneEmpty(string[] colors)...

// Establece el puntaje máximo
1 reference
private void SetMaxScore()...

// Evalua si se logro un nuevo marcador
1 reference
private bool CheckNewMaxScore()...

#endregion
```

***PlayerWon***

```
// Verifica si el jugador gano
2 references
private bool PlayerWon(int[] hits)
{
    foreach (var iter in hits) if (iter != 2) return false;
    return true;
}
```

Verifica si el jugador paso el nivel.

***PlayerLost***

```
// Verifica si el jugador perdio
1 reference
private bool PlayerLost()
{
    if (count_fila > 9) return true;
    return false;
}
```

Verifica si el jugador perdió el nivel.

***Reboot***

```
// Reinicia el nivel
2 references
private void Reboot()
{
    // Establece un nuevo código secreto
    if(count_fila > 0) game.InitSecret();
    // Reestablece los intentos
    count_fila = 0;
    // Reestablece el puntaje actual
    score = 10000;
    // Inicializa la disponibilidad de las filas
    InicializarFilas();
    // Reinicia todos los datos del juego actual
    ReiniciarFilas();
}
```

Reinicia el nivel, es decir, reestablece un nuevo código secreto, el id del intento que se encuentra el jugador, el puntaje inicial e inicializa filas y las reinicia (ambos métodos serán hablados más adelante).

***ChangePosition***

Este método consiste en el cambio de posición de un color dentro de la combinación de colores que el jugador este realizando, lo que hace es establecer un color blanco en la anterior posición del color deseado. Sin embargo, es importante mencionar que la modificación de cada combinación se da por el id del intento en el que se encuentre el jugador.

```
// Hace que una posición  se inicialice
1 reference
private void ChangePosition(int position)
{
    switch (count_fila)
    {
        case (0):
            if (position == 0) C0_F9.Background = new SolidColorBrush(Colors.White);
            else if (position == 1) C1_F9.Background = new SolidColorBrush(Colors.White);
            else if (position == 2) C2_F9.Background = new SolidColorBrush(Colors.White);
            else if(position == 3) C3_F9.Background = new SolidColorBrush(Colors.White);
            break;
        case (1):
            if (position == 0) C0_F8.Background = new SolidColorBrush(Colors.White);
            else if (position == 1) C1_F8.Background = new SolidColorBrush(Colors.White);
            else if (position == 2) C2_F8.Background = new SolidColorBrush(Colors.White);
            else if (position == 3) C3_F8.Background = new SolidColorBrush(Colors.White);
            break;
        case (2):
            if (position == 0) C0_F7.Background = new SolidColorBrush(Colors.White);
            else if (position == 1) C1_F7.Background = new SolidColorBrush(Colors.White);
            else if (position == 2) C2_F7.Background = new SolidColorBrush(Colors.White);
            else if (position == 3) C3_F7.Background = new SolidColorBrush(Colors.White);
            break;
        case (3):
            if (position == 0) C0_F6.Background = new SolidColorBrush(Colors.White);
            else if (position == 1) C1_F6.Background = new SolidColorBrush(Colors.White);
            else if (position == 2) C2_F6.Background = new SolidColorBrush(Colors.White);
            else if (position == 3) C3_F6.Background = new SolidColorBrush(Colors.White);
            break;
```

```
    case (4):
        if (position == 0) C0_F5.Background = new SolidColorBrush(Colors.White);
        else if (position == 1) C1_F5.Background = new SolidColorBrush(Colors.White);
        else if (position == 2) C2_F5.Background = new SolidColorBrush(Colors.White);
        else if (position == 3) C3_F5.Background = new SolidColorBrush(Colors.White);
        break;
    case (5):
        if (position == 0) C0_F4.Background = new SolidColorBrush(Colors.White);
        else if (position == 1) C1_F4.Background = new SolidColorBrush(Colors.White);
        else if (position == 2) C2_F4.Background = new SolidColorBrush(Colors.White);
        else if (position == 3) C3_F4.Background = new SolidColorBrush(Colors.White);
        break;
    case (6):
        if (position == 0) C0_F3.Background = new SolidColorBrush(Colors.White);
        else if (position == 1) C1_F3.Background = new SolidColorBrush(Colors.White);
        else if (position == 2) C2_F3.Background = new SolidColorBrush(Colors.White);
        else if (position == 3) C3_F3.Background = new SolidColorBrush(Colors.White);
        break;
    case (7):
        if (position == 0) C0_F2.Background = new SolidColorBrush(Colors.White);
        else if (position == 1) C1_F2.Background = new SolidColorBrush(Colors.White);
        else if (position == 2) C2_F2.Background = new SolidColorBrush(Colors.White);
        else if (position == 3) C3_F2.Background = new SolidColorBrush(Colors.White);
        break;
    case (8):
        if (position == 0) C0_F1.Background = new SolidColorBrush(Colors.White);
        else if (position == 1) C1_F1.Background = new SolidColorBrush(Colors.White);
        else if (position == 2) C2_F1.Background = new SolidColorBrush(Colors.White);
        else if (position == 3) C3_F1.Background = new SolidColorBrush(Colors.White);
        break;
```

```
        case (9):
            if (position == 0) C0_F0.Background = new SolidColorBrush(Colors.White);
            else if (position == 1) C1_F0.Background = new SolidColorBrush(Colors.White);
            else if (position == 2) C2_F0.Background = new SolidColorBrush(Colors.White);
            else if (position == 3) C3_F0.Background = new SolidColorBrush(Colors.White);
            break;
    }
}
```

***CheckEmpty***

```
// Verifica si la combinación esta vacía
1 reference
private bool CheckEmpty(string[] colors)
{
    foreach (var iter in colors) if (iter != "#FFFFFF") return false;
    return true;
}
```

El método verifica si la combinación entera está vacía.

***CheckOneEmpty***

```
// Verifica que no existan items en blanco para las combinaciones de los intentos
1 reference
private bool CheckOneEmpty(string[] colors)
{
    foreach (var iter in colors) if (iter == "#FFFFFF") return true;
    return false;
}
```

El método verifica si la combinación tiene una casilla vacía.

***SetMaxScore***

Este método permita el guardado y establecimiento del nuevo mejor puntaje.

9. Para empezar, se define una variable id que toma el id del nivel que se está jugando.
10. Después, se crea una variable que tendrá el nombre del archivo de texto que guardará todos los registros de los mejores puntajes de cada nivel.
11. Después, se crea una variable que almacenará temporalmente todos los mejores puntajes de todos niveles.
12. Finalmente, se creará una variable contadora count como 0.

```
// Establece el puntaje máximo
1 reference
private void SetMaxScore()
{
    // Identifica que nivel se esta jugando
    var id = game.GetIdLevel();
    // Variables para modificar el archivo de puntajes
    string filename = "scores.txt";
    string[] scores = new string[3];
    int count = 0;
```

13. A continuación, se leen los datos del archivo “scores.txt” y se guardan estos datos en la variable scores.

```
// Lee los datos y guarda en scores
var linesRead = File.ReadLines("scores.txt");
foreach (var iter in linesRead) scores[count++] = iter;
```

14. Después, se borra el archivo “scores.txt”
15. A continuación, dado el id del nivel se establece el nuevo mejor puntaje en el id del score.
16. Finalmente, se guardan todos los datos en el archivo “scores.txt” creado nuevamente.

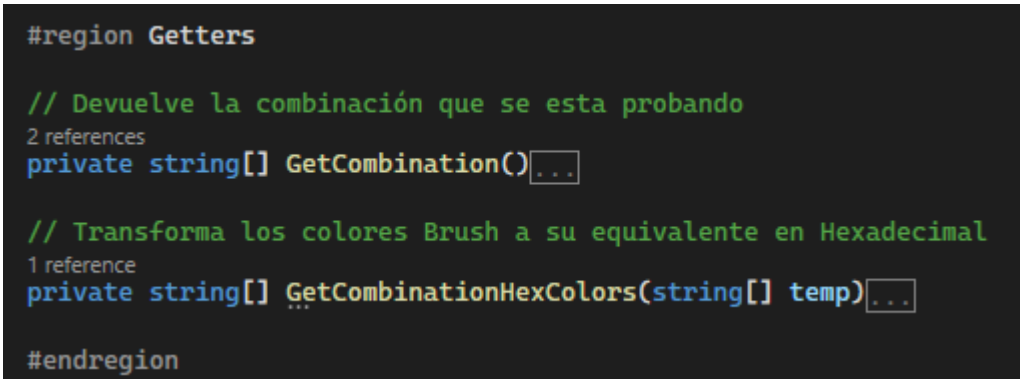
```
    // Borra el archivo
    if (File.Exists(filename)) File.Delete(filename);

    // Establece la nueva información
    if (id == 1) scores[0] = score.ToString();
    else if(id == 2) scores[1] = score.ToString();
    else if(id == 3) scores[2] = score.ToString();

    // Guarda la nueva información
    using (FileStream fs = File.Create(filename))
    {
        Byte[] score_1 = new UTF8Encoding(true).GetBytes(scores[0] + "\n");
        Byte[] score_2 = new UTF8Encoding(true).GetBytes(scores[1] + "\n");
        Byte[] score_3 = new UTF8Encoding(true).GetBytes(scores[2] + "\n");
        fs.Write(score_1,0,score_1.Length);
        fs.Write(score_2, 0, score_2.Length);
        fs.Write(score_3, 0, score_3.Length);
    }
}
```

***CheckNewMaxScore***

```
// Evalua si se logro un nuevo marcador

private bool CheckNewMaxScore()
{
    if (score > Int32.Parse(puntaje_maximo.Text)) return true;
    return false;
}
```

El método permite verificar si se supero el anterior mejor puntaje.

**Región: Getters**




Esta región contiene métodos que permiten retomar datos de la interfaz o de alguna lógica.

***GetCombination***

```
// Devuelve la combinación que se esta probando
2 references
private string[] GetCombination()
{
    string[] temp = new string[4];
    // El número del case + 1 indica la fila que se esta probando.
    switch (count_fila)
    {
        case (0):
            temp[0] = C0_F9.Background.ToString();
            temp[1] = C1_F9.Background.ToString();
            temp[2] = C2_F9.Background.ToString();
            temp[3] = C3_F9.Background.ToString();
            break;
        case (1):
            temp[0] = C0_F8.Background.ToString();
            temp[1] = C1_F8.Background.ToString();
            temp[2] = C2_F8.Background.ToString();
            temp[3] = C3_F8.Background.ToString();
            break;
        case (2):
            temp[0] = C0_F7.Background.ToString();
            temp[1] = C1_F7.Background.ToString();
            temp[2] = C2_F7.Background.ToString();
            temp[3] = C3_F7.Background.ToString();
            break;
        case (3):
            temp[0] = C0_F6.Background.ToString();
            temp[1] = C1_F6.Background.ToString();
            temp[2] = C2_F6.Background.ToString();
            temp[3] = C3_F6.Background.ToString();
            break;
        case (4):
            temp[0] = C0_F5.Background.ToString();
            temp[1] = C1_F5.Background.ToString();
            temp[2] = C2_F5.Background.ToString();
            temp[3] = C3_F5.Background.ToString();
            break;
        case (5):
            temp[0] = C0_F4.Background.ToString();
            temp[1] = C1_F4.Background.ToString();
            temp[2] = C2_F4.Background.ToString();
            temp[3] = C3_F4.Background.ToString();
            break;
        case (6):
            temp[0] = C0_F3.Background.ToString();
            temp[1] = C1_F3.Background.ToString();
            temp[2] = C2_F3.Background.ToString();
            temp[3] = C3_F3.Background.ToString();
            break;
        case (7):
            temp[0] = C0_F2.Background.ToString();
            temp[1] = C1_F2.Background.ToString();
            temp[2] = C2_F2.Background.ToString();
            temp[3] = C3_F2.Background.ToString();
            break;
```

```
            case (8):
                temp[0] = C0_F1.Background.ToString();
                temp[1] = C1_F1.Background.ToString();
                temp[2] = C2_F1.Background.ToString();
                temp[3] = C3_F1.Background.ToString();
                break;
            case (9):
                temp[0] = C0_F0.Background.ToString();
                temp[1] = C1_F0.Background.ToString();
                temp[2] = C2_F0.Background.ToString();
                temp[3] = C3_F0.Background.ToString();
                break;
        }
        return temp;
    }
```

El método devuelve la combinación de colores utilizadas por el jugador tras realizar su intento, esto se da en base al id del intento que se encuentre el jugador. Sin embargo, los colores devuletos se basan en códigos hexadecimales de 8 dígitos.

***GetCombinationHexColors***

```
// Transforma los colores Brush a su equivalente en Hexadecimal
1 reference
private string[] GetCombinationHexColors(string[] temp)
{
    for (int i = 0; i < temp.Length; i++)
    {
        if (temp[i] == "#FFFFFFFF") temp[i] = "#FFFFFF"; // Conversión del blanco
        else if (temp[i] == "#FFFF0000") temp[i] = "#FF0000"; // Conversión del color rojo
        else if (temp[i] == "#FFFFA500") temp[i] = "#FFA500"; // Conversión del color amarillo
        else if (temp[i] == "#FF800080") temp[i] = "#800080"; // Conversión del color morado
        else if (temp[i] == "#FF9ACD32") temp[i] = "#9ACD32"; // Conversión del color verde
        else if (temp[i] == "#00000000") temp[i] = "#000000"; // Conversión del color negro
        else if (temp[i] == "#FFFF7F50") temp[i] = "#FF7F50"; // Conversión del color anaranjado
        else if (temp[i] == "#FFFF00FF") temp[i] = "#FF00FF"; // Conversión del color rosado
        else if (temp[i] == "#FF000080") temp[i] = "#000080"; // Conversión del color azul
    }
    return temp;
}
```

El método permite la transformación de un arreglo de colores con códigos hexadecimales de 8 dígitos a un arreglo de colores con códigos hexadecimales de 6 dígitos. Permite la conversión de 9 colores diferentes.

**Región: Inicializadores**

```
#region Inicializadores

// Permite la visualización de los aciertos dados al usuario

private void SetHits(int[] hits)...

// Inicializa las filas que pueden ser modificadas
3 references
private void InicializarFilas()...

// Reinicia todas las filas a su valor inicial -> en blanco

private void ReiniciarFilas()...

// Inicializa el puntaje inicial

private void InicializarPuntaje()...

// Trae puntajes máximos existentes

private string InitMaxScore()...

#endregion
```

La región de inicializadores contiene métodos que permiten la propia inicialización de varias partes del programa.

***SetHits***

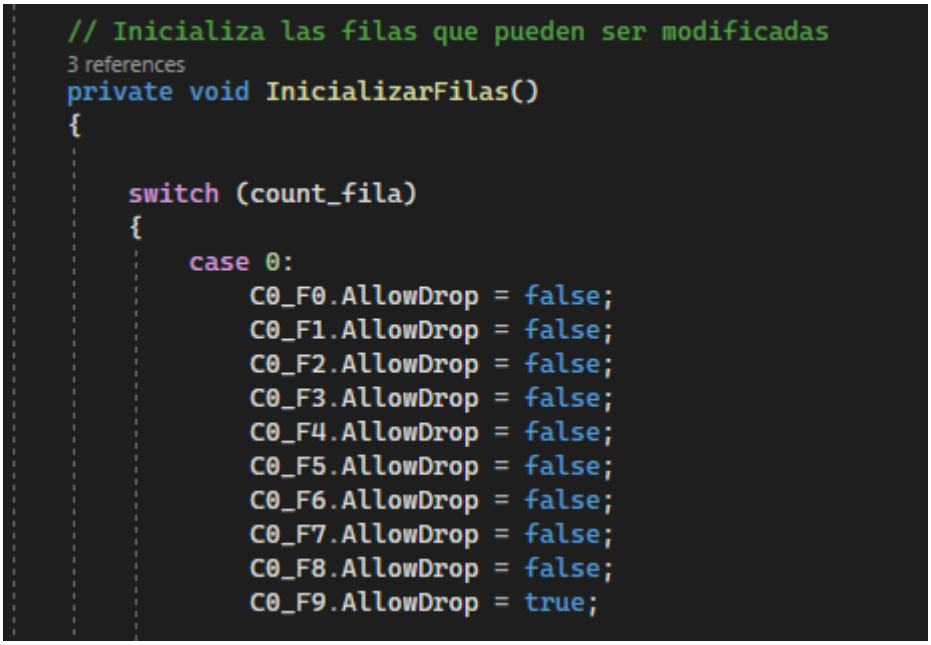
El método presente permite hacer que la interfaz gráfica que permite la visualización de los aciertos del jugador se establezca en base a la combinación dada, un color negro si hay un ítem en la posición correcta, un color gris si hay un ítem en la posición incorrecta y blanco si no existe ítems acertados.

```
// Permite la visualización de los aciertos dados al usuario
1 reference
private void SetHits(int[] hits)
{
    // Coloca color negro si hay un item en la posición correcta.
    // Coloca color gris si hay un item en la posición incorrecta.
    // Deja en blanco si no existe items acertados.
    switch (count_fila)
    {
        case (0):
            // Primer circulo
            if (hits[0] == 2) A1_F10.Background = new SolidColorBrush(Colors.Black);
            else if (hits[0] == 1) A1_F10.Background = new SolidColorBrush(Colors.Gray);
            // Segundo circulo
            if (hits[1] == 2) A2_F10.Background = new SolidColorBrush(Colors.Black);
            else if (hits[1] == 1) A2_F10.Background = new SolidColorBrush(Colors.Gray);
            // Tercer circulo
            if (hits[2] == 2) A3_F10.Background = new SolidColorBrush(Colors.Black);
            else if (hits[3] == 1) A3_F10.Background = new SolidColorBrush(Colors.Gray);
            // Cuarto circulo
            if (hits[3] == 2) A4_F10.Background = new SolidColorBrush(Colors.Black);
            else if (hits[3] == 1) A4_F10.Background = new SolidColorBrush(Colors.Gray);
            break;
```

(Nota: El mismo patrón funciona para cada intento, para más información visualizar el código fuente del proyecto)

***InicializarFilas***

El método presente configura y establece que filas de combinaciones pueden ser modificadas, es decir, a que filas podemos agregar colores para sus combinaciones.



```
// Inicializa las filas que pueden ser modificadas
3 references
private void InicializarFilas()
{

    switch (count_fila)
    {
        case 0:
            C0_F0.AllowDrop = false;
            C0_F1.AllowDrop = false;
            C0_F2.AllowDrop = false;
            C0_F3.AllowDrop = false;
            C0_F4.AllowDrop = false;
            C0_F5.AllowDrop = false;
            C0_F6.AllowDrop = false;
            C0_F7.AllowDrop = false;
            C0_F8.AllowDrop = false;
            C0_F9.AllowDrop = true;
```

(Nota: El mismo patrón funciona para cada fila, para más información visualizar el código fuente del proyecto)

***ReiniciarFilas***

El método permite que todas las filas del juego donde se colocarán y estaban colocadas las combinaciones se reestablezcan a blanco.

```
// Reinicia todas las filas a su valor inicial -> en blanco
1 reference
private void ReiniciarFilas()
{
    // Primeras columnas de todas las filas
    C0_F0.Background = new SolidColorBrush(Colors.White);
    C0_F1.Background = new SolidColorBrush(Colors.White);
    C0_F2.Background = new SolidColorBrush(Colors.White);
    C0_F3.Background = new SolidColorBrush(Colors.White);
    C0_F4.Background = new SolidColorBrush(Colors.White);
    C0_F5.Background = new SolidColorBrush(Colors.White);
    C0_F6.Background = new SolidColorBrush(Colors.White);
    C0_F7.Background = new SolidColorBrush(Colors.White);
    C0_F8.Background = new SolidColorBrush(Colors.White);
    C0_F9.Background = new SolidColorBrush(Colors.White);
```

(Nota: El mismo patrón funciona para cada fila, para más información visualizar el código fuente del proyecto)

***InicializarPuntaje***

```
// Inicializa el puntaje inicial
1 reference
private void InicializarPuntaje()
{
    puntaje_actual.Text = score.ToString();
    puntaje_maximo.Text = InitMaxScore();
    // Identifica que nivel se esta jugando
    var id = game.GetIdLevel();
}
```

El método permite inicializar los puntajes, un puntaje actual del jugador y el mejor puntaje hasta la fecha.

***InitMaxScore***

```
// Trae puntajes máximos existentes
1 reference
private string InitMaxScore()
{
    string curFile = @"scores.txt";
    // Verifica si existen datos guardados
    if (File.Exists(curFile))
    {
        string[] scores = new string[3];
        int count = 0;
        var linesRead = File.ReadLines("scores.txt");
        foreach(var iter in linesRead) scores[count++] = iter;
        if (game.GetIdLevel() == 1) return scores[0];
        else if(game.GetIdLevel() == 2) return scores[1];
        else if(game.GetIdLevel() == 3) return scores[2];
    }
    else
    {
        using(FileStream fs = File.Create(curFile))
        {
            Byte[] temp_score = new UTF8Encoding(true).GetBytes("0\n");
            fs.Write(temp_score, 0, temp_score.Length);
            fs.Write(temp_score, 0, temp_score.Length);
            fs.Write(temp_score, 0, temp_score.Length);
        }
    }
    return "0";
}
```

El método permite verificar si existe el archivo “scores.txt” si lo hace retorna el valor del mejor puntaje hasta la fecha dado el id del nivel, sin embargo, en caso de no que no existe, se crea el archivo e inicializa todos los mejores puntajes como 0.

**Región: Notificaciones**

```
#region Notificaciones

// Muestra un mensaje de que el jugador gano
1 reference
private void NotifyWon(int p_score)...

// Muestra un mensaje de que el jugador perdio
1 reference
private void NotifyLost()...

// Muestra las configuraciones del nivel
1 reference
private void NotifyGameSettings()...

// Notifica que un color ha sido repetido
0 references
private void NotifyRepeatedColor()...

// Notifica si la combinación fue ejecutada vacía
1 reference
private void NotifyEmpty()...

// Notifica que hay un item en blanco en la combinación
1 reference
private void NotifyOneEmpty()...

// Notifica que se logro un nuevo record
1 reference
private void NotifyMaxScore(int p_score)...

// Notifica las instrucciones del juego
1 reference
private void NotifyRules()...

#endregion
```

Esta región se compone por métodos que notifican al usuario de diversas situaciones que suceden durante la operabilidad y jugabilidad del juego.

***NotifyWon***

```
// Muestra un mensaje de que el jugador gano
1 reference
private void NotifyWon(int p_score)
{
    string message = $"¡Felicitaciones!\n\n ¡Has pasado el nivel! Tú puntaje fue de {p_score}.";
    string caption = "¡Has ganado!";
    MessageBoxButton button = MessageBoxButton.OK;
    MessageBoxResult result = MessageBox.Show(message, caption, button);
    switch (result)
    {
        // Vuelve al menú de niveles
        case MessageBoxResult.OK:
            NavigationService.Navigate(new SelectLevel());
            break;
    }
}
```

El método notifica si el jugador supero el nivel.

### *NotifyLost*

```
// Muestra un mensaje de que el jugador perdio
1 reference
private void NotifyLost()
{
    string message = "¡Oh No! No has logrado superar el nivel...\n\n¿Deseas intentar de nuevo?";
    string caption = "¡Perdiste!";
    MessageBoxButton button = MessageBoxButton.YesNo;
    MessageBoxResult result = MessageBox.Show(message, caption, button);
    switch (result)
    {
        // Reinicia el nivel
        case MessageBoxResult.Yes:
            Reboot();
            break;

        // Vuelve al menú de niveles
        case MessageBoxResult.No:
            NavigationService.Navigate(new SelectLevel());
            break;
    }
}
```

El método notifica al jugar si este ha perdido el nivel y le da la opción de intentar nuevamente.

### *NotifyGameSettings*

```
// Muestra las configuraciones del nivel
1 reference
private void NotifyGameSettings()
{
    string message = "¡Bienvenido!\n\nHas seleccionado el nivel de dificultad fácil, tienes 4 colores para crear tus combinaciones
    string caption = "Nivel: Fácil";
    MessageBoxButton button = MessageBoxButton.OK;
    MessageBoxResult result = MessageBox.Show(message, caption, button);
    switch (result)
    {
        // No ocurre nada
        case MessageBoxResult.OK:
            break;
    }
}
```

El método notifica al jugar instrucciones y modo de juego del nivel en el que se encuentra.

### *NotifyRepeatedColor*

```
// Notifica que un color ha sido repetido
0 references
private void NotifyRepeatedColor()
{
    string message = "¡Has repetido un color!\n¡Recuerda que en este nivel no esta permitido su repetición!";
    string caption = "Advertencia";
    MessageBoxButton button = MessageBoxButton.OK;
    MessageBoxResult result = MessageBox.Show(message, caption, button);
    switch (result)
    {
        // No sucede nada
        case MessageBoxResult.OK:
            break;
    }
}
```

El método notifica al jugador si existe un color repetido en la combinación realizada.

### *NotifyEmpty*

```
// Notifica si la combinación fue ejecutada vacía
1 reference
private void NotifyEmpty()
{
    string message = "Parece que no has colocado ningún tipo de combinación...\n\n¡Coloca alguna antes de probar!";
    string caption = "Advertencia";
    MessageBoxButton button = MessageBoxButton.OK;
    MessageBoxResult result = MessageBox.Show(message, caption, button);
}
```

El método notifica al jugador si la combinación realizada se encuentra vacía.

### *NotifyOneEmpty*

```
// Notifica que hay un item en blanco en la combinación
1 reference
private void NotifyOneEmpty()
{
    string message = "Parece que has dejado un espacio en blanco...\n\n¡Ten más cuidado la próxima vez!";
    string caption = "Advertencia";
    MessageBoxButton button = MessageBoxButton.OK;
    MessageBoxResult result = MessageBox.Show(message, caption, button);
}
```

El método notifica al jugador si en la combinación realizada se encuentra un ítem vacío.

### *NotifyMaxScore*

```
// Notifica que se logro un nuevo record
1 reference
private void NotifyMaxScore(int p_score)
{
    string message = $"¡Has logrado un nuevo record!\n\nObtuviste un puntaje de: {p_score}";
    string caption = "Felicitaciones";
    MessageBoxButton button = MessageBoxButton.OK;
    MessageBoxResult result = MessageBox.Show(message, caption, button);
}
```

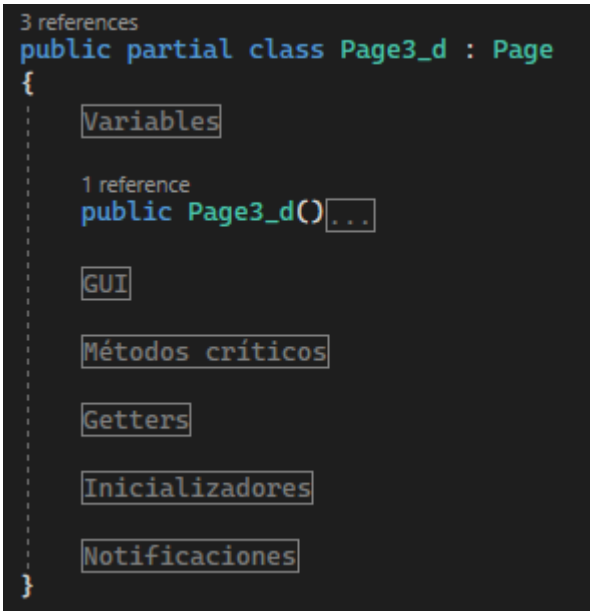
El método notifica al jugar si el jugador supero su anterior mejor puntaje.

### *NotifyRules*

```
// Notifica las instrucciones del juego
1 reference
private void NotifyRules()
{
    string message = "¡Recuerda!\n\nCada vez que se empieza una partida, el juego creará un nuevo código secreto que lo debes de
        "\n\n**********************************************************************\n\n" +
        "Sobre el juego:\n\nPara poder acertar con mayor facilidad el código secreto ¡podrás guiarte de las pistas que el juego
        "1. El color negro significa que tienes un color en la posición correcta!\n" +
        "2. El color gris significa que tienes un color correcta pero en la posición incorrecta!";
    string caption = "Reglas del juego";
    MessageBoxButton buttons = MessageBoxButton.OK;
    MessageBoxResult result = MessageBox.Show(message, caption, buttons);
}
```

El método notifica las reglas del juego al jugador.

***Conexión lógico-gráfico sobre el nivel difícil***




Esta interfaz consiste en 7 regiones: Variables, Constructor, GUI, Métodos críticos, Getters, Inicializadores y Notificaciones; todas ellas permiten la operabilidad y jugabilidad del juego Mastermind.

**Región: Variables**

```
#region Variables

// Inicializamos el juego para usarse.
Mastermind game;
// Permite verificar en que intento se encuentra el usuario
int count_fila;
// Variable del puntaje
int score = 10000;

#endregion
```

Esta región consiste en todas las variables que el nivel fácil va a utilizar, estos son la creación de una variable game que inicialice el objeto Mastermind para luego poder generar el juego de nivel fácil, por otra parte, se tiene otra variable como count_fila que permite validar en que intento el jugador se encuentra y finalmente la variable score que permite verificar el puntaje del jugador.

**Región: Constructor**

```
1 reference
public Page3_d()
{
    InitializeComponent();
    // Crea el juego
    game = new Mastermind(3);
    // Inicializa los intentos
    count_fila = 0;
    // Inicializa la jugabilidad
    InicializarFilas();
    // Inicializa los puntajes
    InicializarPuntaje();
    // Notifica datos del modo de juego
    var thread = new Thread(() => { NotifyGameSettings(); });
    thread.Start();
}
```

11. El constructor permite inicializar todos los componentes de la interfaz que será hablado en la siguiente sección.
12. Por otra parte, se inicializa el juego Mastermind como nivel difícil mandando al constructor del objeto el código 3.
13. Después, se inicializa la variable count_fila para validar que el jugador se encuentre en el primer intento del juego.
14. A continuación, se inicializan las filas con el método InicializarFilas, este método será hablado a continuación a detalle a igual que InicializarPuntaje.
15. Finalmente se genera una notificación de que el juego ha empezado y que debe esperar del nivel, esto, se realiza en base a una metodología asincrónica.

**Región: GUI**

```
#region GUI

//evento de drag and drop
1 reference
private void Ellipse_MouseMove(object sender, MouseEventArgs e)...

// Pinta un circulo con el color deseado
40 references
private void Border_Drop(object sender, DragEventArgs e)...

// Vuelta al menú de selección de niveles
1 reference
private void Button_Click(object sender, RoutedEventArgs e)...

// Reinicia el nivel
1 reference
private void reboot_btn_Click(object sender, RoutedEventArgs e)...

// Permite visualizar las instrucciones del juego
1 reference
private void instrucciones_btn_Click(object sender, RoutedEventArgs e)...

// Confirmación de combinación
1 reference
private void confirmar_btn_Click(object sender, RoutedEventArgs e)...

#endregion
```

Esta región tiene todos los métodos que funcionan en base a eventos que ocurren con la interfaz.

***Método Ellipse_MouseMove***

```
//evento de drag and drop
1 reference
private void Ellipse_MouseMove(object sender, MouseEventArgs e)
{
    if (sender is Ellipse ellipse && e.LeftButton == MouseButtonState.Pressed)
    {
        DragDrop.DoDragDrop(ellipse, ellipse.Fill.ToString(), DragDropEffects.Copy);
    }
}
```

El método permite el evento drag y drop de los colores hacía los espacios para los intentos de combinaciones.

***Método Border_Drop***

```
// Pinta un circulo con el color deseado
40 references
private void Border_Drop(object sender, DragEventArgs e)
{
    if (sender is Border border)
    {
        if (e.Data.GetDataPresent(DataFormats.StringFormat))
        {
            string stringFromDrop = (string)e.Data.GetData(DataFormats.StringFormat);
            BrushConverter convertir = new BrushConverter();
            if (convertir.IsValid(stringFromDrop))
            {
                Brush pincel = (Brush)convertir.ConvertFromString(stringFromDrop);
                border.Background = pincel;
            }
        }
    }
}
```

El método permite pintar un espacio de los intentos de combinación tras mover un color hacia ellos, sin embargo, al ser el nivel difícil, se debe tener en cuenta que se permite la repetición de colores.

***Método Button_Click***

```
// Vuelta al menú de selección de niveles
1 reference
private void Button_Click(object sender, RoutedEventArgs e)
{
    NavigationService.Navigate(new SelectLevel());
}
```

Este método nos permite volver al menú de selección de niveles.

***Método reboot_btn_Click***

```
// Reinicia el nivel
1 reference
private void reboot_btn_Click(object sender, RoutedEventArgs e)
{
    Reboot();
}
```

Este método permite que cuando el jugador desee, el juego se reinicie y se establezcan todos los valores.

***Método instrucciones_btn_Click***

```
// Permite visualizar las instrucciones del juego
1 reference
private void instrucciones_btn_Click(object sender, RoutedEventArgs e)
{
    NotifyRules();
}
```

Este método permite que cuando el jugador desee, pueda visualizar las instrucciones y reglas del nivel y el propio juego.

### *Método confirmar_btn_Click*

Este método consiste el más importante de todos ya que permite toda la lógica del juego y posible su propia jugabilidad, sin embargo, consiste en varias acciones que permiten lo antes mencionado.

11. Primero se crea una variable que almacenará la combinación del jugador.
12. Después, dada esta combinación, se transformarán todos los colores traídos como códigos hexadecimales de 8 dígitos a colores de 6 dígitos.

```
// Confirmación de combinación
1 reference
private void confirmar_btn_Click(object sender, RoutedEventArgs e)
{
    // Tomamos la combinación que se esta probando al momento.
    string[] combination = GetCombination();
    // Transformamos los colores a hexadecimal
    string[] combination_colors = GetCombinationHexColors(combination);
```

13. Después, se verifica si la combinación no tiene datos, es decir, si la combinación del usuario tiene todos los espacios de combinación llenos, si toda la combinación está vacía entonces se le notificará al usuario que esta está vacía; por otro lado, si no estuviera vacía se ejecutaran los siguientes pasos:

```
    // Verificamos si la combinación no tiene datos
    if(!CheckEmpty(combination_colors))
    {
        // Verficamos que todos los colores han sido utilizados y no se han dejado espacios en blanco
        if (!CheckOneEmpty(combination_colors))...
        else...
    }
    else
    {
        NotifyEmpty();
    }
}
```

14. Una vez se evalúe que la combinación no este vacía, se verifica si existe algún espacio dejado en blanco de la propia combinación, si esto es así entonces se notifica que existe un propio espacio en blanco en la combinación; si es que no, entonces se ejecuta el paso 5.

```
// Verficamos que todos los colores han sido utilizados y no se han dejado espacios en blanco
if (!CheckOneEmpty(combination_colors))...
else
{
    NotifyOneEmpty();
}
```

15. Finalmente, sucede la operabilidad y jugabilidad del juego:
    15.1. Primero se toman todos los aciertos que el usuario logro con su combinación en la variable hits.
    15.2. Después, se configura la interfaz dedicada para los aciertos con los datos retomados en la variable hits, el método SetHits se hablará más adelante.
    15.3. A continuación, se suma un intento al identificador de intento actual incrementando la variable count_fila.

```
// Probamos la combinación
var hits = game.TryCombination(combination_colors);
// Configuramos los aciertos dados y los mostramos al usuario
SetHits(hits);
// Incrementamos el contador del intento
count_fila++;
```

    15.4. Después, se verifica si el usuario gano el juego, para esto se utiliza el método PlayerWon (igualmente hablado más adelante).
    15.5. Sí el jugador gano, se verifica si su puntaje obtenido fue mayor al mejor puntaje obtenido hasta la fecha, sí fue así entonces se notifica al usuario su acometido y se guarda su nuevo mejor puntaje. Sí no supero su mejor puntaje o si lo hizo, de igual manera se notificará al usuario de que el jugador supero el nivel.

```
// Verificar si gano el juego el jugador
if (PlayerWon(hits))
{
    if (CheckNewMaxScore())
    {
        NotifyMaxScore(Int32.Parse(puntaje_actual.Text));
        SetMaxScore();
    }
    NotifyWon(Int32.Parse(puntaje_actual.Text));
}
```

    15.6. Después, haya ganado o no, se actualiza el puntaje actual del jugador y se lo representa en su ambiente gráfico.

```
// Actualiza el score del jugador
score = score - 1000;
puntaje_actual.Text = score.ToString();
```

    15.7. A continuación, se verifica si el usuario perdió el juego, si es así entonces se le notifica al usuario, pero se le da la oportunidad de volver a intentar superar el nivel.
    15.8. Finalmente se inicializan las filas que pueden ser modificadas, el método InicializarFilas como se mencionó anteriormente será explicado más adelante.

```
// Verifica si perdio el juego el jugador
if (PlayerLost() & !PlayerWon(hits))
{
    NotifyLost();
}
// Actualiza las filas que pueden ser modificadas
if (count_fila < 10) InicializarFilas();
```

**Región: Métodos Críticos**

Esta región contiene todos los métodos críticos para el funcionamiento y jugabilidad del juego, consiste en los siguientes métodos:

```
#region Métodos críticos

// Verifica si el jugador gano
2 references
private bool PlayerWon(int[] hits)...

// Verifica si el jugador perdio
1 reference
private bool PlayerLost()...

// Reinicia el nivel
2 references
private void Reboot()...

// Hace que una posición  se inicialice
1 reference
private void ChangePosition(int position)...

// Verifica si la combinación esta vacia
1 reference
private bool CheckEmpty(string[] colors)...

// Verifica que no existan items en blanco para las combinaciones de los intentos
1 reference
private bool CheckOneEmpty(string[] colors)...

// Establece el puntaje máximo
1 reference
private void SetMaxScore()...

// Evalua si se logro un nuevo marcador
1 reference
private bool CheckNewMaxScore()...

#endregion
```

***PlayerWon***

```
// Verifica si el jugador gano
2 references
private bool PlayerWon(int[] hits)
{
    foreach (var iter in hits) if (iter != 2) return false;
    return true;
}
```

Verifica si el jugador paso el nivel.

***PlayerLost***

```
// Verifica si el jugador perdio
1 reference
private bool PlayerLost()
{
    if (count_fila > 9) return true;
    return false;
}
```

Verifica si el jugador perdió el nivel.

***Reboot***

```
// Reinicia el nivel
2 references
private void Reboot()
{
    // Establece un nuevo código secreto
    if(count_fila > 0) game.InitSecret();
    // Reestablece los intentos
    count_fila = 0;
    // Reestablece el puntaje actual
    score = 10000;
    // Inicializa la disponibilidad de las filas
    InicializarFilas();
    // Reinicia todos los datos del juego actual
    ReiniciarFilas();
}
```

Reinicia el nivel, es decir, reestablece un nuevo código secreto, el id del intento que se encuentra el jugador, el puntaje inicial e inicializa filas y las reinicia (ambos métodos serán hablados más adelante).

***ChangePosition***

Este método consiste en el cambio de posición de un color dentro de la combinación de colores que el jugador este realizando, lo que hace es establecer un color blanco en la anterior posición del color deseado. Sin embargo, es importante mencionar que la modificación de cada combinación se da por el id del intento en el que se encuentre el jugador.

```
// Hace que una posición  se inicialice
1 reference
private void ChangePosition(int position)
{
    switch (count_fila)
    {
        case (0):
            if (position == 0) C0_F9.Background = new SolidColorBrush(Colors.White);
            else if (position == 1) C1_F9.Background = new SolidColorBrush(Colors.White);
            else if (position == 2) C2_F9.Background = new SolidColorBrush(Colors.White);
            else if(position == 3) C3_F9.Background = new SolidColorBrush(Colors.White);
            break;
        case (1):
            if (position == 0) C0_F8.Background = new SolidColorBrush(Colors.White);
            else if (position == 1) C1_F8.Background = new SolidColorBrush(Colors.White);
            else if (position == 2) C2_F8.Background = new SolidColorBrush(Colors.White);
            else if (position == 3) C3_F8.Background = new SolidColorBrush(Colors.White);
            break;
        case (2):
            if (position == 0) C0_F7.Background = new SolidColorBrush(Colors.White);
            else if (position == 1) C1_F7.Background = new SolidColorBrush(Colors.White);
            else if (position == 2) C2_F7.Background = new SolidColorBrush(Colors.White);
            else if (position == 3) C3_F7.Background = new SolidColorBrush(Colors.White);
            break;
        case (3):
            if (position == 0) C0_F6.Background = new SolidColorBrush(Colors.White);
            else if (position == 1) C1_F6.Background = new SolidColorBrush(Colors.White);
            else if (position == 2) C2_F6.Background = new SolidColorBrush(Colors.White);
            else if (position == 3) C3_F6.Background = new SolidColorBrush(Colors.White);
            break;
```

```
    case (4):
        if (position == 0) C0_F5.Background = new SolidColorBrush(Colors.White);
        else if (position == 1) C1_F5.Background = new SolidColorBrush(Colors.White);
        else if (position == 2) C2_F5.Background = new SolidColorBrush(Colors.White);
        else if (position == 3) C3_F5.Background = new SolidColorBrush(Colors.White);
        break;
    case (5):
        if (position == 0) C0_F4.Background = new SolidColorBrush(Colors.White);
        else if (position == 1) C1_F4.Background = new SolidColorBrush(Colors.White);
        else if (position == 2) C2_F4.Background = new SolidColorBrush(Colors.White);
        else if (position == 3) C3_F4.Background = new SolidColorBrush(Colors.White);
        break;
    case (6):
        if (position == 0) C0_F3.Background = new SolidColorBrush(Colors.White);
        else if (position == 1) C1_F3.Background = new SolidColorBrush(Colors.White);
        else if (position == 2) C2_F3.Background = new SolidColorBrush(Colors.White);
        else if (position == 3) C3_F3.Background = new SolidColorBrush(Colors.White);
        break;
    case (7):
        if (position == 0) C0_F2.Background = new SolidColorBrush(Colors.White);
        else if (position == 1) C1_F2.Background = new SolidColorBrush(Colors.White);
        else if (position == 2) C2_F2.Background = new SolidColorBrush(Colors.White);
        else if (position == 3) C3_F2.Background = new SolidColorBrush(Colors.White);
        break;
    case (8):
        if (position == 0) C0_F1.Background = new SolidColorBrush(Colors.White);
        else if (position == 1) C1_F1.Background = new SolidColorBrush(Colors.White);
        else if (position == 2) C2_F1.Background = new SolidColorBrush(Colors.White);
        else if (position == 3) C3_F1.Background = new SolidColorBrush(Colors.White);
        break;
```

```
        case (9):
            if (position == 0) C0_F0.Background = new SolidColorBrush(Colors.White);
            else if (position == 1) C1_F0.Background = new SolidColorBrush(Colors.White);
            else if (position == 2) C2_F0.Background = new SolidColorBrush(Colors.White);
            else if (position == 3) C3_F0.Background = new SolidColorBrush(Colors.White);
            break;
    }
}
```

***CheckEmpty***

```
// Verifica si la combinación esta vacía
1 reference
private bool CheckEmpty(string[] colors)
{
    foreach (var iter in colors) if (iter != "#FFFFFF") return false;
    return true;
}
```

El método verifica si la combinación entera está vacía.

***CheckOneEmpty***

```
// Verifica que no existan items en blanco para las combinaciones de los intentos
1 reference
private bool CheckOneEmpty(string[] colors)
{
    foreach (var iter in colors) if (iter == "#FFFFFF") return true;
    return false;
}
```

El método verifica si la combinación tiene una casilla vacía.

***SetMaxScore***

Este método permita el guardado y establecimiento del nuevo mejor puntaje.

17. Para empezar, se define una variable id que toma el id del nivel que se está jugando.
18. Después, se crea una variable que tendrá el nombre del archivo de texto que guardará todos los registros de los mejores puntajes de cada nivel.
19. Después, se crea una variable que almacenará temporalmente todos los mejores puntajes de todos niveles.
20. Finalmente, se creará una variable contadora count como 0.

```
// Establece el puntaje máximo
1 reference
private void SetMaxScore()
{
    // Identifica que nivel se esta jugando
    var id = game.GetIdLevel();
    // Variables para modificar el archivo de puntajes
    string filename = "scores.txt";
    string[] scores = new string[3];
    int count = 0;
```

21. A continuación, se leen los datos del archivo “scores.txt” y se guardan estos datos en la variable scores.

```
// Lee los datos y guarda en scores
var linesRead = File.ReadLines("scores.txt");
foreach (var iter in linesRead) scores[count++] = iter;
```

22. Después, se borra el archivo “scores.txt”
23. A continuación, dado el id del nivel se establece el nuevo mejor puntaje en el id del score.
24. Finalmente, se guardan todos los datos en el archivo “scores.txt” creado nuevamente.

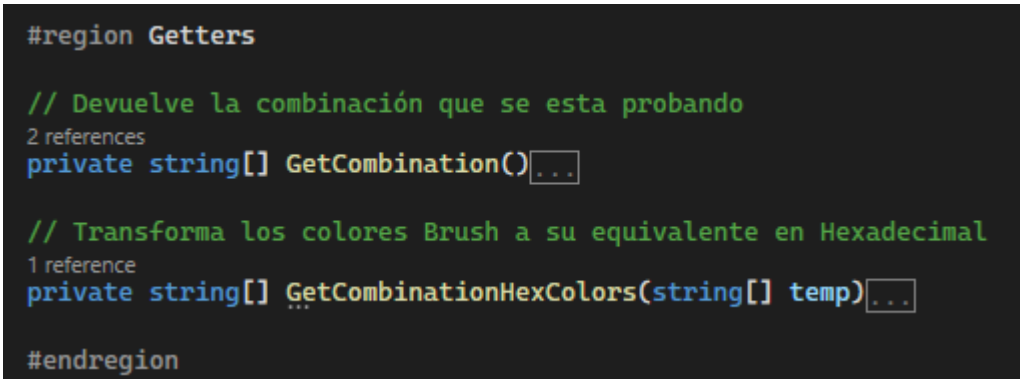
```
    // Borra el archivo
    if (File.Exists(filename)) File.Delete(filename);

    // Establece la nueva información
    if (id == 1) scores[0] = score.ToString();
    else if(id == 2) scores[1] = score.ToString();
    else if(id == 3) scores[2] = score.ToString();

    // Guarda la nueva información
    using (FileStream fs = File.Create(filename))
    {
        Byte[] score_1 = new UTF8Encoding(true).GetBytes(scores[0] + "\n");
        Byte[] score_2 = new UTF8Encoding(true).GetBytes(scores[1] + "\n");
        Byte[] score_3 = new UTF8Encoding(true).GetBytes(scores[2] + "\n");
        fs.Write(score_1,0,score_1.Length);
        fs.Write(score_2, 0, score_2.Length);
        fs.Write(score_3, 0, score_3.Length);
    }
}
```

***CheckNewMaxScore***

```
// Evalua si se logro un nuevo marcador

private bool CheckNewMaxScore()
{
    if (score > Int32.Parse(puntaje_maximo.Text)) return true;
    return false;
}
```

El método permite verificar si se superó el anterior mejor puntaje.

**Región: Getters**




Esta región contiene métodos que permiten retomar datos de la interfaz o de alguna lógica.

***GetCombination***

```
// Devuelve la combinación que se esta probando
2 references
private string[] GetCombination()
{
    string[] temp = new string[4];
    // El número del case + 1 indica la fila que se esta probando.
    switch (count_fila)
    {
        case (0):
            temp[0] = C0_F9.Background.ToString();
            temp[1] = C1_F9.Background.ToString();
            temp[2] = C2_F9.Background.ToString();
            temp[3] = C3_F9.Background.ToString();
            break;
        case (1):
            temp[0] = C0_F8.Background.ToString();
            temp[1] = C1_F8.Background.ToString();
            temp[2] = C2_F8.Background.ToString();
            temp[3] = C3_F8.Background.ToString();
            break;
        case (2):
            temp[0] = C0_F7.Background.ToString();
            temp[1] = C1_F7.Background.ToString();
            temp[2] = C2_F7.Background.ToString();
            temp[3] = C3_F7.Background.ToString();
            break;
        case (3):
            temp[0] = C0_F6.Background.ToString();
            temp[1] = C1_F6.Background.ToString();
            temp[2] = C2_F6.Background.ToString();
            temp[3] = C3_F6.Background.ToString();
            break;
        case (4):
            temp[0] = C0_F5.Background.ToString();
            temp[1] = C1_F5.Background.ToString();
            temp[2] = C2_F5.Background.ToString();
            temp[3] = C3_F5.Background.ToString();
            break;
        case (5):
            temp[0] = C0_F4.Background.ToString();
            temp[1] = C1_F4.Background.ToString();
            temp[2] = C2_F4.Background.ToString();
            temp[3] = C3_F4.Background.ToString();
            break;
        case (6):
            temp[0] = C0_F3.Background.ToString();
            temp[1] = C1_F3.Background.ToString();
            temp[2] = C2_F3.Background.ToString();
            temp[3] = C3_F3.Background.ToString();
            break;
        case (7):
            temp[0] = C0_F2.Background.ToString();
            temp[1] = C1_F2.Background.ToString();
            temp[2] = C2_F2.Background.ToString();
            temp[3] = C3_F2.Background.ToString();
            break;
```

```
            case (8):
                temp[0] = C0_F1.Background.ToString();
                temp[1] = C1_F1.Background.ToString();
                temp[2] = C2_F1.Background.ToString();
                temp[3] = C3_F1.Background.ToString();
                break;
            case (9):
                temp[0] = C0_F0.Background.ToString();
                temp[1] = C1_F0.Background.ToString();
                temp[2] = C2_F0.Background.ToString();
                temp[3] = C3_F0.Background.ToString();
                break;
        }
        return temp;
    }
```

El método devuelve la combinación de colores utilizadas por el jugador tras realizar su intento, esto se da en base al id del intento que se encuentre el jugador. Sin embargo, los colores devuletos se basan en códigos hexadecimales de 8 dígitos.

***GetCombinationHexColors***

```
// Transforma los colores Brush a su equivalente en Hexadecimal
1 reference
private string[] GetCombinationHexColors(string[] temp)
{
    for (int i = 0; i < temp.Length; i++)
    {
        if (temp[i] == "#FFFFFFFF") temp[i] = "#FFFFFF"; // Conversión del blanco
        else if (temp[i] == "#FFFF0000") temp[i] = "#FF0000"; // Conversión del color rojo
        else if (temp[i] == "#FFFFA500") temp[i] = "#FFA500"; // Conversión del color amarillo
        else if (temp[i] == "#FF800080") temp[i] = "#800080"; // Conversión del color morado
        else if (temp[i] == "#FF9ACD32") temp[i] = "#9ACD32"; // Conversión del color verde
        else if (temp[i] == "#00000000") temp[i] = "#000000"; // Conversión del color negro
        else if (temp[i] == "#FFFF7F50") temp[i] = "#FF7F50"; // Conversión del color anaranjado
        else if (temp[i] == "#FFFF00FF") temp[i] = "#FF00FF"; // Conversión del color rosado
        else if (temp[i] == "#FF000080") temp[i] = "#000080"; // Conversión del color azul
    }
    return temp;
}
```

El método permite la transformación de un arreglo de colores con códigos hexadecimales de 8 dígitos a un arreglo de colores con códigos hexadecimales de 6 dígitos. Permite la conversión de 9 colores diferentes.

**Región: Inicializadores**

```
#region Inicializadores

// Permite la visualización de los aciertos dados al usuario

private void SetHits(int[] hits)...

// Inicializa las filas que pueden ser modificadas
3 references
private void InicializarFilas()...

// Reinicia todas las filas a su valor inicial -> en blanco

private void ReiniciarFilas()...

// Inicializa el puntaje inicial

private void InicializarPuntaje()...

// Trae puntajes máximos existentes

private string InitMaxScore()...

#endregion
```

La región de inicializadores contiene métodos que permiten la propia inicialización de varias partes del programa.

***SetHits***

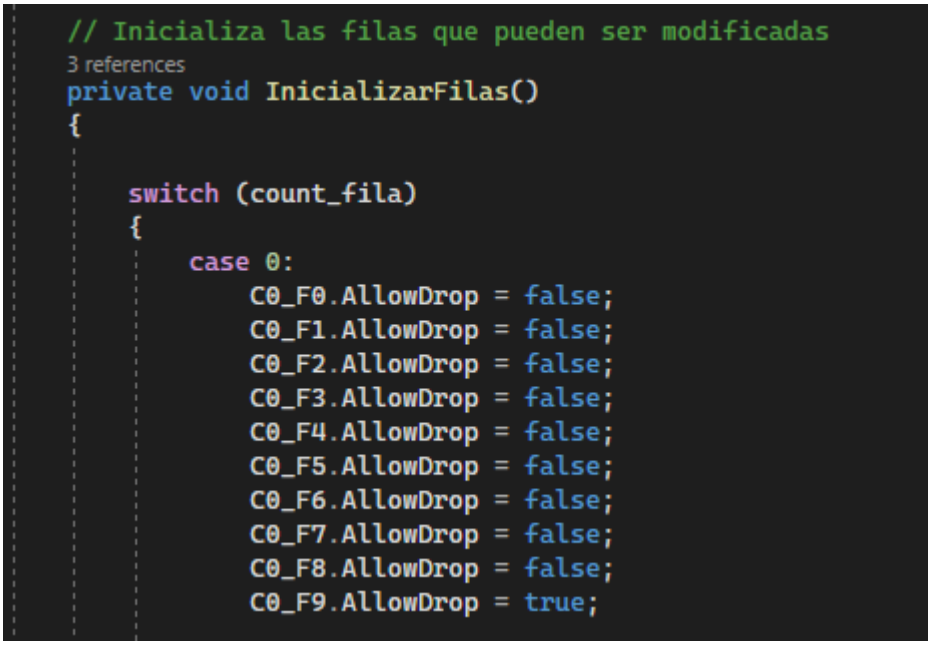
El método presente permite hacer que la interfaz gráfica que permite la visualización de los aciertos del jugador se establezca en base a la combinación dada, un color negro si hay un ítem en la posición correcta, un color gris si hay un ítem en la posición incorrecta y blanco si no existe ítems acertados.

```
// Permite la visualización de los aciertos dados al usuario
1 reference
private void SetHits(int[] hits)
{
    // Coloca color negro si hay un item en la posición correcta.
    // Coloca color gris si hay un item en la posición incorrecta.
    // Deja en blanco si no existe items acertados.
    switch (count_fila)
    {
        case (0):
            // Primer circulo
            if (hits[0] == 2) A1_F10.Background = new SolidColorBrush(Colors.Black);
            else if (hits[0] == 1) A1_F10.Background = new SolidColorBrush(Colors.Gray);
            // Segundo circulo
            if (hits[1] == 2) A2_F10.Background = new SolidColorBrush(Colors.Black);
            else if (hits[1] == 1) A2_F10.Background = new SolidColorBrush(Colors.Gray);
            // Tercer circulo
            if (hits[2] == 2) A3_F10.Background = new SolidColorBrush(Colors.Black);
            else if (hits[3] == 1) A3_F10.Background = new SolidColorBrush(Colors.Gray);
            // Cuarto circulo
            if (hits[3] == 2) A4_F10.Background = new SolidColorBrush(Colors.Black);
            else if (hits[3] == 1) A4_F10.Background = new SolidColorBrush(Colors.Gray);
            break;
```

(Nota: El mismo patrón funciona para cada intento, para más información visualizar el código fuente del proyecto)

***InicializarFilas***

El método presente configura y establece que filas de combinaciones pueden ser modificadas, es decir, a que filas podemos agregar colores para sus combinaciones.



```
// Inicializa las filas que pueden ser modificadas
3 references
private void InicializarFilas()
{

    switch (count_fila)
    {
        case 0:
            C0_F0.AllowDrop = false;
            C0_F1.AllowDrop = false;
            C0_F2.AllowDrop = false;
            C0_F3.AllowDrop = false;
            C0_F4.AllowDrop = false;
            C0_F5.AllowDrop = false;
            C0_F6.AllowDrop = false;
            C0_F7.AllowDrop = false;
            C0_F8.AllowDrop = false;
            C0_F9.AllowDrop = true;
```

(Nota: El mismo patrón funciona para cada fila, para más información visualizar el código fuente del proyecto)

***ReiniciarFilas***

El método permite que todas las filas del juego donde se colocarán y estaban colocadas las combinaciones se reestablezcan a blanco.

```
// Reinicia todas las filas a su valor inicial -> en blanco
1 reference
private void ReiniciarFilas()
{
    // Primeras columnas de todas las filas
    C0_F0.Background = new SolidColorBrush(Colors.White);
    C0_F1.Background = new SolidColorBrush(Colors.White);
    C0_F2.Background = new SolidColorBrush(Colors.White);
    C0_F3.Background = new SolidColorBrush(Colors.White);
    C0_F4.Background = new SolidColorBrush(Colors.White);
    C0_F5.Background = new SolidColorBrush(Colors.White);
    C0_F6.Background = new SolidColorBrush(Colors.White);
    C0_F7.Background = new SolidColorBrush(Colors.White);
    C0_F8.Background = new SolidColorBrush(Colors.White);
    C0_F9.Background = new SolidColorBrush(Colors.White);
```

(Nota: El mismo patrón funciona para cada fila, para más información visualizar el código fuente del proyecto)

***InicializarPuntaje***

```
// Inicializa el puntaje inicial
1 reference
private void InicializarPuntaje()
{
    puntaje_actual.Text = score.ToString();
    puntaje_maximo.Text = InitMaxScore();
    // Identifica que nivel se esta jugando
    var id = game.GetIdLevel();
}
```

El método permite inicializar los puntajes, un puntaje actual del jugador y el mejor puntaje hasta la fecha.

***InitMaxScore***

```
// Trae puntajes máximos existentes
1 reference
private string InitMaxScore()
{
    string curFile = @"scores.txt";
    // Verifica si existen datos guardados
    if (File.Exists(curFile))
    {
        string[] scores = new string[3];
        int count = 0;
        var linesRead = File.ReadLines("scores.txt");
        foreach(var iter in linesRead) scores[count++] = iter;
        if (game.GetIdLevel() == 1) return scores[0];
        else if(game.GetIdLevel() == 2) return scores[1];
        else if(game.GetIdLevel() == 3) return scores[2];
    }
    else
    {
        using(FileStream fs = File.Create(curFile))
        {
            Byte[] temp_score = new UTF8Encoding(true).GetBytes("0\n");
            fs.Write(temp_score, 0, temp_score.Length);
            fs.Write(temp_score, 0, temp_score.Length);
            fs.Write(temp_score, 0, temp_score.Length);
        }
    }
    return "0";
}
```

El método permite verificar si existe el archivo “scores.txt” si lo hace retorna el valor del mejor puntaje hasta la fecha dado el id del nivel, sin embargo, en caso de no que no existe, se crea el archivo e inicializa todos los mejores puntajes como 0.

**Región: Notificaciones**

```
#region Notificaciones

// Muestra un mensaje de que el jugador gano
1 reference
private void NotifyWon(int p_score)...

// Muestra un mensaje de que el jugador perdio
1 reference
private void NotifyLost()...

// Muestra las configuraciones del nivel
1 reference
private void NotifyGameSettings()...

// Notifica que un color ha sido repetido
0 references
private void NotifyRepeatedColor()...

// Notifica si la combinación fue ejecutada vacía
1 reference
private void NotifyEmpty()...

// Notifica que hay un item en blanco en la combinación
1 reference
private void NotifyOneEmpty()...

// Notifica que se logro un nuevo record
1 reference
private void NotifyMaxScore(int p_score)...

// Notifica las instrucciones del juego
1 reference
private void NotifyRules()...

#endregion
```

Esta región se compone por métodos que notifican al usuario de diversas situaciones que suceden durante la operabilidad y jugabilidad del juego.

***NotifyWon***

```
// Muestra un mensaje de que el jugador gano
1 reference
private void NotifyWon(int p_score)
{
    string message = $"¡Felicitaciones!\n\n ¡Has pasado el nivel! Tú puntaje fue de {p_score}.";
    string caption = "¡Has ganado!";
    MessageBoxButton button = MessageBoxButton.OK;
    MessageBoxResult result = MessageBox.Show(message, caption, button);
    switch (result)
    {
        // Vuelve al menú de niveles
        case MessageBoxResult.OK:
            NavigationService.Navigate(new SelectLevel());
            break;
    }
}
```

El método notifica si el jugador supero el nivel.

### *NotifyLost*

```
// Muestra un mensaje de que el jugador perdio
1 reference
private void NotifyLost()
{
    string message = "¡Oh No! No has logrado superar el nivel...\n\n¿Deseas intentar de nuevo?";
    string caption = "¡Perdiste!";
    MessageBoxButton button = MessageBoxButton.YesNo;
    MessageBoxResult result = MessageBox.Show(message, caption, button);
    switch (result)
    {
        // Reinicia el nivel
        case MessageBoxResult.Yes:
            Reboot();
            break;

        // Vuelve al menú de niveles
        case MessageBoxResult.No:
            NavigationService.Navigate(new SelectLevel());
            break;
    }
}
```

El método notifica al jugar si este ha perdido el nivel y le da la opción de intentar nuevamente.

### *NotifyGameSettings*

```
// Muestra las configuraciones del nivel
1 reference
private void NotifyGameSettings()
{
    string message = "¡Bienvenido!\n\nHas seleccionado el nivel de dificultad fácil, tienes 4 colores para crear tus combinaciones
    string caption = "Nivel: Fácil";
    MessageBoxButton button = MessageBoxButton.OK;
    MessageBoxResult result = MessageBox.Show(message, caption, button);
    switch (result)
    {
        // No ocurre nada
        case MessageBoxResult.OK:
            break;
    }
}
```

El método notifica al jugar instrucciones y modo de juego del nivel en el que se encuentra.

### *NotifyRepeatedColor*

```
// Notifica que un color ha sido repetido
0 references
private void NotifyRepeatedColor()
{
    string message = "¡Has repetido un color!\n¡Recuerda que en este nivel no esta permitido su repetición!";
    string caption = "Advertencia";
    MessageBoxButton button = MessageBoxButton.OK;
    MessageBoxResult result = MessageBox.Show(message, caption, button);
    switch (result)
    {
        // No sucede nada
        case MessageBoxResult.OK:
            break;
    }
}
```

El método notifica al jugador si existe un color repetido en la combinación realizada.

### *NotifyEmpty*

```
// Notifica si la combinación fue ejecutada vacía
1 reference
private void NotifyEmpty()
{
    string message = "Parece que no has colocado ningún tipo de combinación...\n\n¡Coloca alguna antes de probar!";
    string caption = "Advertencia";
    MessageBoxButton button = MessageBoxButton.OK;
    MessageBoxResult result = MessageBox.Show(message, caption, button);
}
```

El método notifica al jugador si la combinación realizada se encuentra vacía.

### *NotifyOneEmpty*

```
// Notifica que hay un item en blanco en la combinación
1 reference
private void NotifyOneEmpty()
{
    string message = "Parece que has dejado un espacio en blanco...\n\n¡Ten más cuidado la próxima vez!";
    string caption = "Advertencia";
    MessageBoxButton button = MessageBoxButton.OK;
    MessageBoxResult result = MessageBox.Show(message, caption, button);
}
```

El método notifica al jugador si en la combinación realizada se encuentra un ítem vacío.

### *NotifyMaxScore*

```
// Notifica que se logro un nuevo record
1 reference
private void NotifyMaxScore(int p_score)
{
    string message = $"¡Has logrado un nuevo record!\n\nObtuviste un puntaje de: {p_score}";
    string caption = "Felicitaciones";
    MessageBoxButton button = MessageBoxButton.OK;
    MessageBoxResult result = MessageBox.Show(message, caption, button);
}
```

El método notifica al jugar si el jugador supero su anterior mejor puntaje.

### *NotifyRules*

```
// Notifica las instrucciones del juego
1 reference
private void NotifyRules()
{
    string message = "¡Recuerda!\n\nCada vez que se empieza una partida, el juego creará un nuevo código secreto que lo debes de
        "\n\n**********************************************************************\n\n" +
        "Sobre el juego:\n\nPara poder acertar con mayor facilidad el código secreto ¡podrás guiarte de las pistas que el juego
        "1. El color negro significa que tienes un color en la posición correcta!\n" +
        "2. El color gris significa que tienes un color correcta pero en la posición incorrecta!";
    string caption = "Reglas del juego";
    MessageBoxButton buttons = MessageBoxButton.OK;
    MessageBoxResult result = MessageBox.Show(message, caption, buttons);
}
```

El método notifica las reglas del juego al jugador.

## Tercera sección: Interfaz

### *Descripción de la sección*

En esta sección se podrá observar la interfaz del juego Mastermind, estos serán divididos por su página:

### *Página incial*


La presente es la pantalla inicial del programa, contiene un único botón que permite avanzar hacia la selección de niveles.

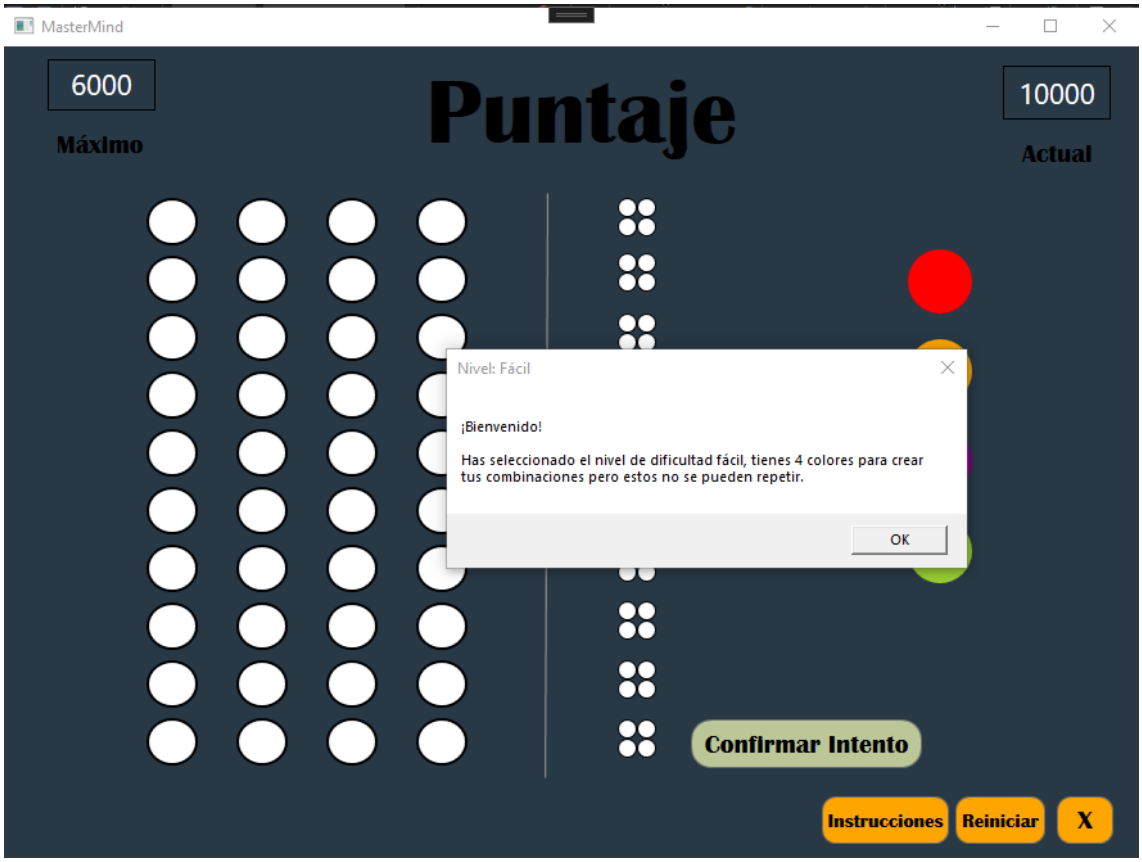
### *Selección de niveles*


En esta página se puede visualizar los niveles de dificultad existentes y una breve descripción de las reglas de juego. Los botones realizan lo siguiente:

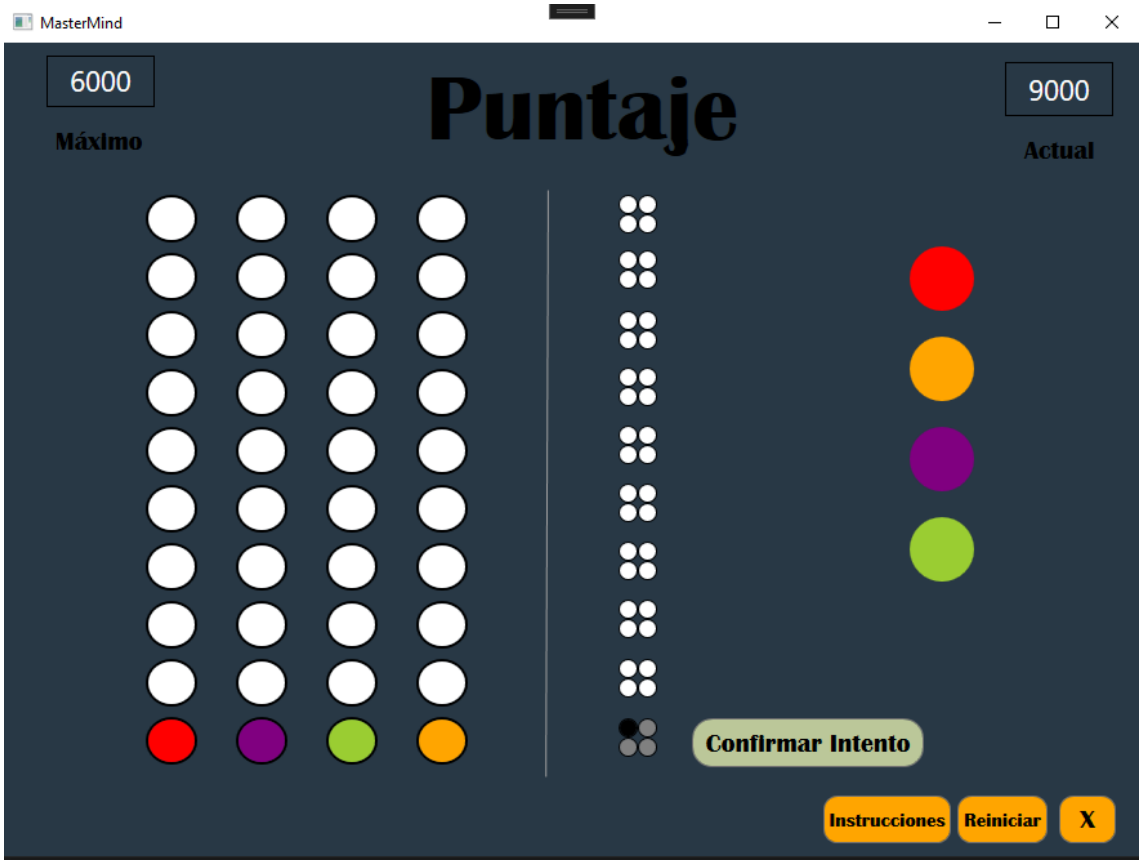
- Fácil: Nos dirige a la interfaz del nivel fácil
- Medio: Nos dirige a la interfaz del nivel medio
- Difícil: Nos dirige a la interfaz del nivel difícil

***Nivel fácil***

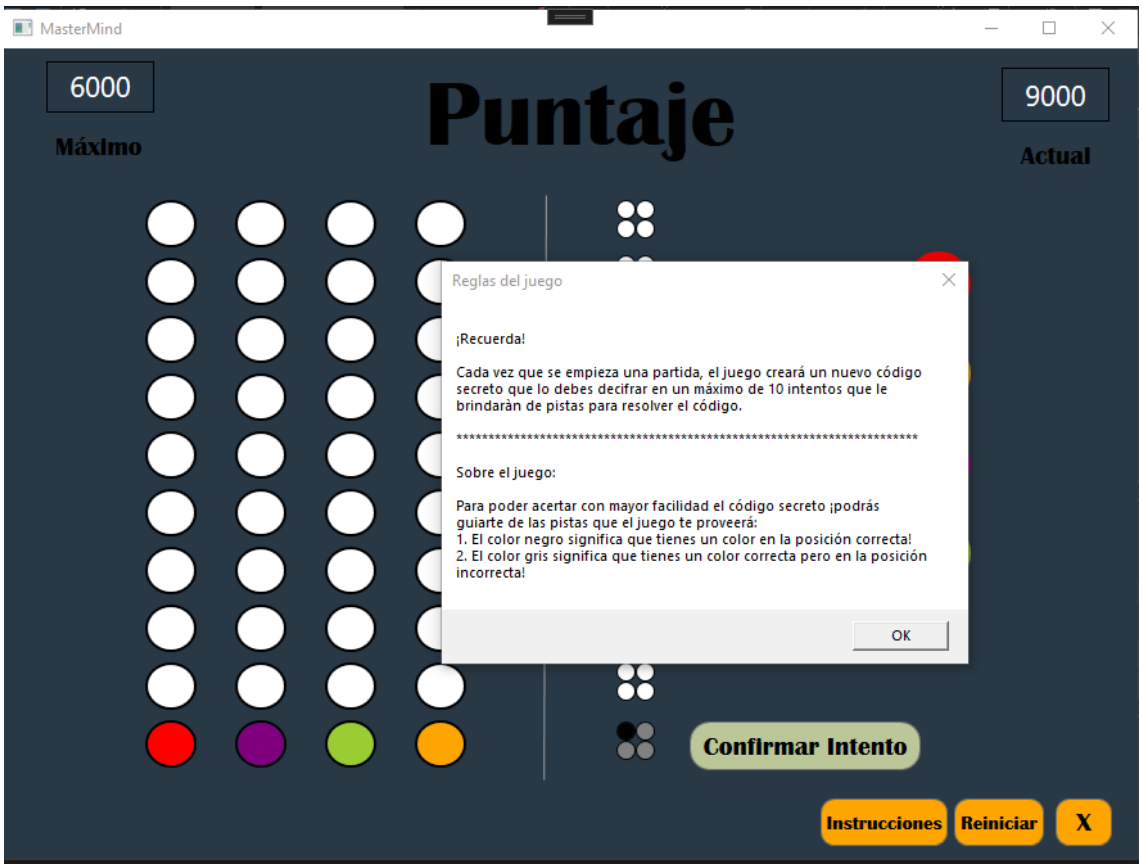
Al comienzo, una vez hemos dado click en el botón Fácil, emergerá una notificación comentando cómo funciona el nivel:



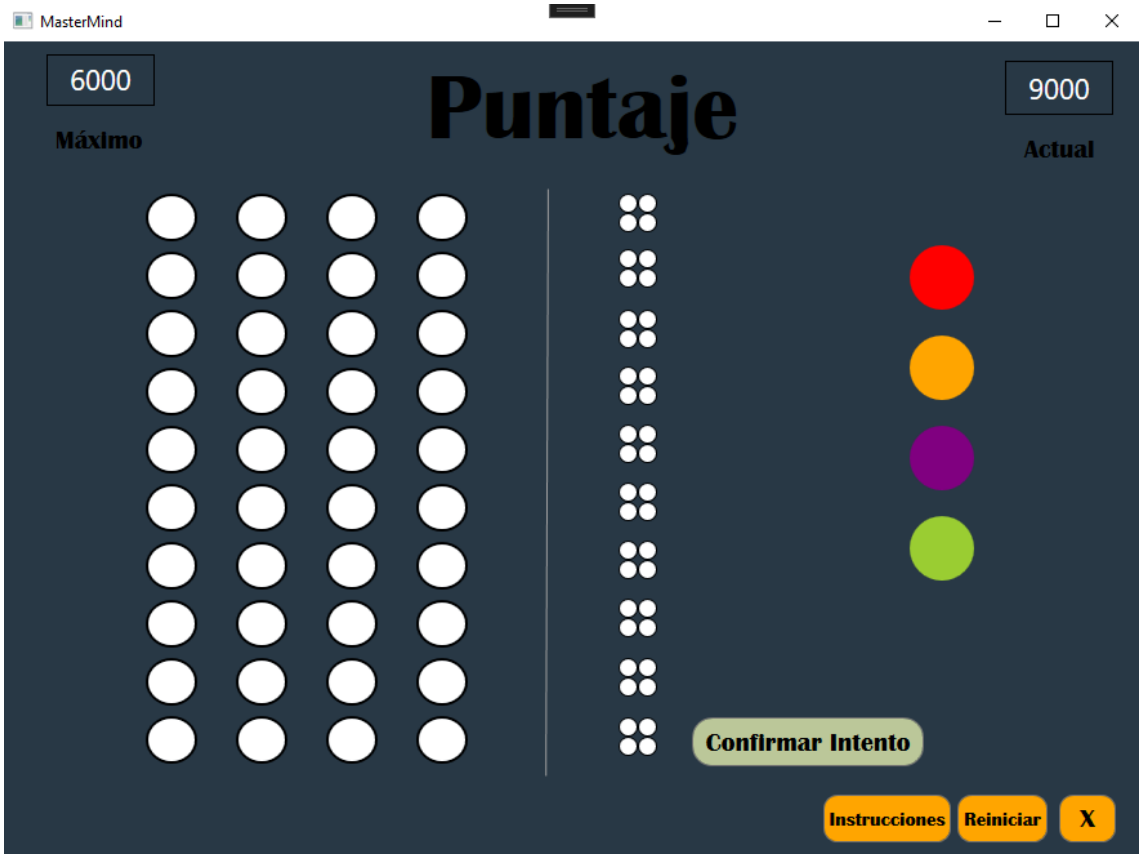

Una vez dado click se podrá operar con la jugabilidad del nivel:



También si damos click en el botón de instrucciones se presentarán las instrucciones del juego:



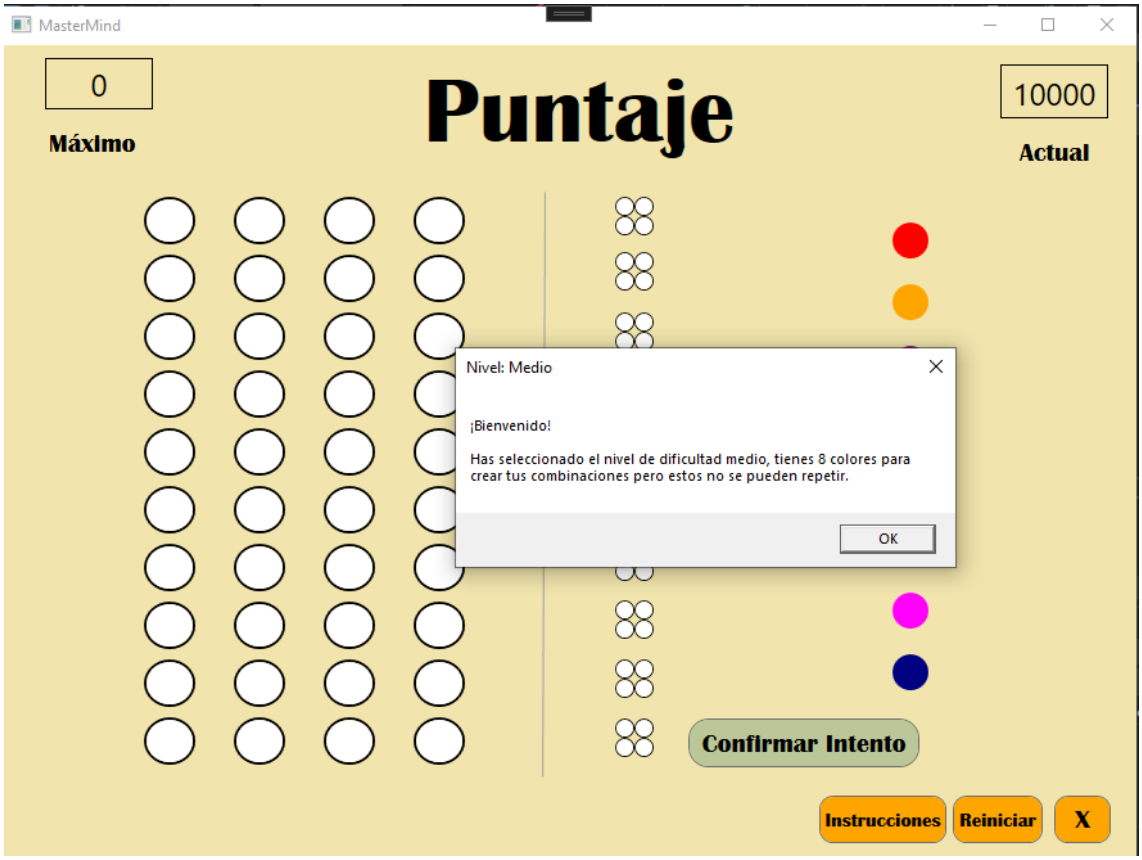

Además, si damos click en reiniciar, se reinicia el nivel:




Finalmente, si damos click en el botón con la “X”, podemos regresar al menú de selección de niveles.

***Nivel medio***

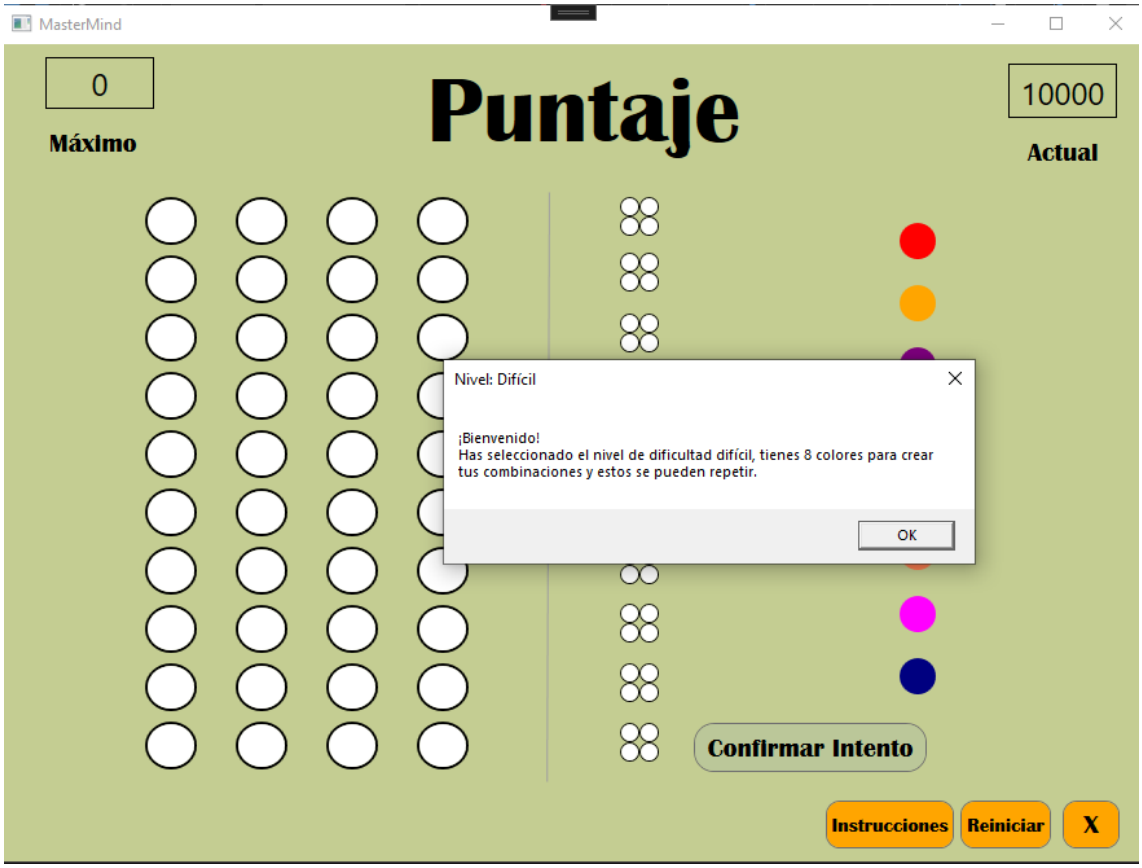
El nivel medio contiene las mismas funcionalidades que el nivel fácil, lo que cambia es que al momento de entrar se notifica el modo de juego del nivel medio:




También se modifica la apariencia del nivel, y las opciones de color posibles.

***Nivel difícil***

El nivel difícil funciona de la misma manera que el nivel medio y fácil, lo único diferente es la notificación del modo de juego del nivel difícil:




## Anexos

### *Anexo 1: Código fuente de la lógica*

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

namespace MasterMind.Classes
{
   internal class Mastermind
   {
      #region Variables

      // Definición del tablero
      int[][] tablero = new int[10][];

      // Definición de los colores
      // [0]->Blanco, [1]->Rojo, [2]->Amarillo, [3]->Morado, [4]->Verde, [5]->Negro,
[6]->Anaranjado, [7]->Rosado, [8]->Azul
      //                [0]      [1]      [2]      [3]      [4]      [5]      [6]      [7]      [8]
```

```
        string[] colores = { "#FFFFFF", "#FF0000", "#FFA500", "#800080", "#9ACD32",
"#000000", "#FF7F50", "#FF00FF", "#000080" };

        // Definición del código secreto
        int[] secret = new int[4];

        // Define un código para el tipo de nivel
        // 0 -> No Definido | 1 -> Fácil | 2 -> Medio | 3 -> Díficil
        int id_nivel = 0;

        #endregion

        public Mastermind(int nivel = 0)
        {
            // Define el nivel
            id_nivel = nivel;
            // Inivializa el tablero
            InitTablero();
            // Inicializa el código secreto
            InitSecret();
        }

        #region Métodos

        // Inicializa el tablero con valores iniciales.
        protected void InitTablero()
        {
            for (int i = 0; i < 10; i++) tablero[i] = new int[4];
        }

        // Inicializa un código secreto
        public void InitSecret()
        {
            secret = new int[4];
            int max_secret;

            // Código secreto para el nivel fácil
            if (id_nivel == 1)
            {
                // Máximo de opciones de color
                max_secret = 4;
                Random rand = new Random();
                // Creación del código secreto
                for (int col = 0; col < 4; col++)
                {
                    var temp_color = rand.Next(1, max_secret + 1);
                    // Se verifica que no se repitan los colores
                    while (IsInArray(secret, temp_color)) temp_color = rand.Next(1,
max_secret + 1);
                    secret[col] = temp_color;
```

```
            }
        }
        // Código para el nivel medio
        else if (id_nivel == 2)
        {
            // Máximo de opciones de color
            max_secret = 8;
            Random rand = new Random();
            // Creación del código secreto
            for (int col = 0; col < 4; col++)
            {
                var temp_color = rand.Next(1, max_secret+1);
                // Se verifica que no se repitan los colores
                while (IsInArray(secret, temp_color)) temp_color = rand.Next(1,
max_secret + 1);
                secret[col] = temp_color;
            }
        }
        // Código para el nivel díficil
        else if (id_nivel == 3)
        {
            // Máximo de opciones de color
            max_secret = 8;
            Random rand = new Random();
            // Creación del código secreto sin importar que se repitan los colores
            for (int col = 0; col < 4; col++)
            {
                secret[col] = rand.Next(1, max_secret);
            }
        }
    }

    // Verifica si un item se encuentra dentro del código secreto
    private bool IsInArray(int[] temp_array, int item)
    {
        foreach (var iter in temp_array) if (iter == item) return true;
        return false;
    }

    // Prueba una combinación
    public int[] TryCombination(string[] combination)
    {
        // Definición aciertos y tipo de acierto
        // 0 -> No hubo acierto
        // 1 -> Acierto en la posición incorrecta
        // 2 -> Acierto en la posición correcta
        int[] hits = new int[4];
        // Contador
        int hit_count = 0;
```

```
            // Descubrir aciertos en la posición correcta
            for (int i = 0; i < combination.Length; i++)
            {
                if (colores[secret[i]] == combination[i])
                {
                    hits[hit_count++] = 2;
                }
            }
            // Descubrir aciertos en la posición incorrecta
            for (int i = 0; i < combination.Length; i++)
            {
                for (int j = 0; j < secret.Length; j++)
                {
                    if (colores[secret[j]] == combination[i] & !(colores[secret[i]] ==
combination[i]))
                    {
                        hits[hit_count++] = 1;
                    }
                }
            }
            // Devuelve los aciertos
            return hits;
        }

        #endregion

        #region Getters

        public string[] GetSecret()
        {
            string[] temp = new string[4];
            for(int i = 0; i < secret.Length; i++)
            {
                temp[i] = colores[secret[i]];
            }
            return temp;
        }

        public int GetIdLevel()
        {
            return id_nivel;
        }

        #endregion
    }
}
```

***Anexo 2: Código fuente del nivel fácil***

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using MasterMind.Classes;
using System.Threading;
using System.IO;

namespace MasterMind.Pages
{
    public class ColoresData : ObservableCollection<SolidColorBrush>
    {
        //Donde se almacenan los colores que se van a utilizar
        public ColoresData() : base()
        {
            Add(new SolidColorBrush(Colors.Red));
            Add(new SolidColorBrush(Colors.Orange));
            Add(new SolidColorBrush(Colors.Purple));
            Add(new SolidColorBrush(Colors.YellowGreen));
        }
```

```
}
/// <summary>
/// Interaction logic for Page1_f.xaml
/// Nivel fácil
/// </summary>
public partial class Page1_f : Page
{
    #region Variables

    // Inicializamos el juego para usarse.
    Mastermind game;
    // Permite verificar en que intento se encuentra el usuario
    int count_fila;
    // Variable del puntaje
    int score = 10000;

    #endregion

    public Page1_f()
    {
        InitializeComponent();
        // Crea el juego
        game = new Mastermind(1);
        // Inicializa los intentos
        count_fila = 0;
        // Inicializa la jugabilidad
        InicializarFilas();
        // Inicializa los puntajes
        InicializarPuntaje();
        // Notifica datos del modo de juego
        var thread = new Thread(() => { NotifyGameSettings(); });
```

```
            thread.Start();
        }

        #region GUI

        //evento de drag and drop
        private void Ellipse_MouseMove(object sender, MouseEventArgs e)
        {
            if (sender is Ellipse ellipse && e.LeftButton == MouseButtonState.Pressed)
            {
                DragDrop.DoDragDrop(ellipse, ellipse.Fill.ToString(),
DragDropEffects.Copy);
            }
        }

        // Pinta un circulo con el color deseado
        private void Border_Drop(object sender, DragEventArgs e)
        {
            if (sender is Border border)
            {
                if (e.Data.GetDataPresent(DataFormats.StringFormat))
                {
                    string stringFromDrop =
(string)e.Data.GetData(DataFormats.StringFormat);
                    BrushConverter convertir = new BrushConverter();
                    if (convertir.IsValid(stringFromDrop))
                    {
                        Brush pincel = (Brush)convertir.ConvertFromString(stringFromDrop);
                        // Trae todos los colores utilizados en la fila actual
                        string[] combination = GetCombination();
                        // Establece el nuevo color del círculo
```

```
                border.Background = pincel;
                // Evalua si el color ya fue utilizado en la creación del código
                for (int i = 0; i < combination.Length; i++)
                {
                    if (combination[i] == border.Background.ToString())
                    {
                        // Si lo fue elimina la anterior posición del color y la coloca en la
nueva
                        ChangePosition(i);
                    }
                }
            }
        }
    }

    // Vuelta al menú de selección de niveles
    private void Button_Click(object sender, RoutedEventArgs e)
    {
        NavigationService.Navigate(new SelectLevel());
    }

    // Reinicia el nivel
    private void reboot_btn_Click(object sender, RoutedEventArgs e)
    {
        Reboot();
    }

    // Permite visualizar las instrucciones del juego
    private void instrucciones_btn_Click(object sender, RoutedEventArgs e)
    {
```

```
            NotifyRules();
        }

        // Confirmación de combinación
        private void confirmar_btn_Click(object sender, RoutedEventArgs e)
        {
            // Tomamos la combinación que se esta probando al momento.
            string[] combination = GetCombination();
            // Transformamos los colores a hexadecimal
            string[] combination_colors = GetCombinationHexColors(combination);
            // Verificamos si la combinación no tiene datos
            if(!CheckEmpty(combination_colors))
            {
                // Verficamos que todos los colores han sido utilizados y no se han dejado
espacios en blanco
                if (!CheckOneEmpty(combination_colors))
                {
                    // Probamos la combinación
                    var hits = game.TryCombination(combination_colors);
                    // Configuramos los aciertos dados y los mostramos al usuario
                    SetHits(hits);
                    // Incrementamos el contador del intento
                    count_fila++;
                    // Verificar si gano el juego el jugador
                    if (PlayerWon(hits))
                    {
                        if (CheckNewMaxScore())
                        {
                            NotifyMaxScore(Int32.Parse(puntaje_actual.Text));
                            SetMaxScore();
                        }
```

```
                NotifyWon(Int32.Parse(puntaje_actual.Text));
            }

            // Actualiza el score del jugador
            score = score - 1000;
            puntaje_actual.Text = score.ToString();

            // Verifica si perdio el juego el jugador
            if (PlayerLost() & !PlayerWon(hits))
            {
                NotifyLost();
            }
            // Actualiza las filas que pueden ser modificadas
            if (count_fila < 10) InicializarFilas();
        }
        else
        {
            NotifyOneEmpty();
        }
    }
    else
    {
        NotifyEmpty();
    }
}

#endregion

#region Métodos críticos

// Verifica si el jugador gano
```

```
private bool PlayerWon(int[] hits)
{
    foreach (var iter in hits) if (iter != 2) return false;
    return true;
}

// Verifica si el jugador perdio
private bool PlayerLost()
{
    if (count_fila > 9) return true;
    return false;
}

// Reinicia el nivel
private void Reboot()
{
    // Establece un nuevo código secreto
    if(count_fila > 0) game.InitSecret();
    // Reestablece los intentos
    count_fila = 0;
    // Reestablece el puntaje actual
    score = 10000;
    // Inicializa la disponibilidad de las filas
    InicializarFilas();
    // Reinicia todos los datos del juego actual
    ReiniciarFilas();
}

// Hace que una posición  se inicialice
private void ChangePosition(int position)
{
```

```
        switch (count_fila)
        {
            case (0):
                if (position == 0) C0_F9.Background = new
SolidColorBrush(Colors.White);
                else if (position == 1) C1_F9.Background = new
SolidColorBrush(Colors.White);
                else if (position == 2) C2_F9.Background = new
SolidColorBrush(Colors.White);
                else if(position == 3) C3_F9.Background = new
SolidColorBrush(Colors.White);
                break;
            case (1):
                if (position == 0) C0_F8.Background = new
SolidColorBrush(Colors.White);
                else if (position == 1) C1_F8.Background = new
SolidColorBrush(Colors.White);
                else if (position == 2) C2_F8.Background = new
SolidColorBrush(Colors.White);
                else if (position == 3) C3_F8.Background = new
SolidColorBrush(Colors.White);
                break;
            case (2):
                if (position == 0) C0_F7.Background = new
SolidColorBrush(Colors.White);
                else if (position == 1) C1_F7.Background = new
SolidColorBrush(Colors.White);
                else if (position == 2) C2_F7.Background = new
SolidColorBrush(Colors.White);
                else if (position == 3) C3_F7.Background = new
SolidColorBrush(Colors.White);
                break;
            case (3):
                if (position == 0) C0_F6.Background = new
SolidColorBrush(Colors.White);
```

```
                else if (position == 1) C1_F6.Background = new
SolidColorBrush(Colors.White);

                else if (position == 2) C2_F6.Background = new
SolidColorBrush(Colors.White);

                else if (position == 3) C3_F6.Background = new
SolidColorBrush(Colors.White);

                break;

            case (4):

                if (position == 0) C0_F5.Background = new
SolidColorBrush(Colors.White);

                else if (position == 1) C1_F5.Background = new
SolidColorBrush(Colors.White);

                else if (position == 2) C2_F5.Background = new
SolidColorBrush(Colors.White);

                else if (position == 3) C3_F5.Background = new
SolidColorBrush(Colors.White);

                break;

            case (5):

                if (position == 0) C0_F4.Background = new
SolidColorBrush(Colors.White);

                else if (position == 1) C1_F4.Background = new
SolidColorBrush(Colors.White);

                else if (position == 2) C2_F4.Background = new
SolidColorBrush(Colors.White);

                else if (position == 3) C3_F4.Background = new
SolidColorBrush(Colors.White);

                break;

            case (6):

                if (position == 0) C0_F3.Background = new
SolidColorBrush(Colors.White);

                else if (position == 1) C1_F3.Background = new
SolidColorBrush(Colors.White);

                else if (position == 2) C2_F3.Background = new
SolidColorBrush(Colors.White);

                else if (position == 3) C3_F3.Background = new
SolidColorBrush(Colors.White);
```

```
                break;
            case (7):
                if (position == 0) C0_F2.Background = new
SolidColorBrush(Colors.White);
                else if (position == 1) C1_F2.Background = new
SolidColorBrush(Colors.White);
                else if (position == 2) C2_F2.Background = new
SolidColorBrush(Colors.White);
                else if (position == 3) C3_F2.Background = new
SolidColorBrush(Colors.White);
                break;
            case (8):
                if (position == 0) C0_F1.Background = new
SolidColorBrush(Colors.White);
                else if (position == 1) C1_F1.Background = new
SolidColorBrush(Colors.White);
                else if (position == 2) C2_F1.Background = new
SolidColorBrush(Colors.White);
                else if (position == 3) C3_F1.Background = new
SolidColorBrush(Colors.White);
                break;
            case (9):
                if (position == 0) C0_F0.Background = new
SolidColorBrush(Colors.White);
                else if (position == 1) C1_F0.Background = new
SolidColorBrush(Colors.White);
                else if (position == 2) C2_F0.Background = new
SolidColorBrush(Colors.White);
                else if (position == 3) C3_F0.Background = new
SolidColorBrush(Colors.White);
                break;
        }
    }

    // Verifica si la combinación esta vacía
```

```
private bool CheckEmpty(string[] colors)
{
    foreach (var iter in colors) if (iter != "#FFFFFF") return false;
    return true;
}

// Verifica que no existan items en blanco para las combinaciones de los intentos
private bool CheckOneEmpty(string[] colors)
{
    foreach (var iter in colors) if (iter == "#FFFFFF") return true;
    return false;
}

// Establece el puntaje máximo
private void SetMaxScore()
{
    // Identifica que nivel se esta jugando
    var id = game.GetIdLevel();
    // Variables para modificar el archivo de puntajes
    string filename = "scores.txt";
    string[] scores = new string[3];
    int count = 0;

    // Lee los datos y guarda en scores
    var linesRead = File.ReadLines("scores.txt");
    foreach (var iter in linesRead) scores[count++] = iter;

    // Borra el archivo
    if (File.Exists(filename)) File.Delete(filename);

    // Establece la nueva información
```

```
    if (id == 1) scores[0] = score.ToString();
    else if(id == 2) scores[1] = score.ToString();
    else if(id == 3) scores[2] = score.ToString();

    // Guarda la nueva información
    using (FileStream fs = File.Create(filename))
    {
        Byte[] score_1 = new UTF8Encoding(true).GetBytes(scores[0] + "\n");
        Byte[] score_2 = new UTF8Encoding(true).GetBytes(scores[1] + "\n");
        Byte[] score_3 = new UTF8Encoding(true).GetBytes(scores[2] + "\n");
        fs.Write(score_1,0,score_1.Length);
        fs.Write(score_2, 0, score_2.Length);
        fs.Write(score_3, 0, score_3.Length);
    }
}

// Evalua si se logro un nuevo marcador
private bool CheckNewMaxScore()
{
    if (score > Int32.Parse(puntaje_maximo.Text)) return true;
    return false;
}

#endregion

#region Getters

// Devuelve la combinación que se esta probando
private string[] GetCombination()
{
    string[] temp = new string[4];
```

```
// El número del case + 1 indica la fila que se esta probando.
switch (count_fila)
{
    case (0):
        temp[0] = C0_F9.Background.ToString();
        temp[1] = C1_F9.Background.ToString();
        temp[2] = C2_F9.Background.ToString();
        temp[3] = C3_F9.Background.ToString();
        break;
    case (1):
        temp[0] = C0_F8.Background.ToString();
        temp[1] = C1_F8.Background.ToString();
        temp[2] = C2_F8.Background.ToString();
        temp[3] = C3_F8.Background.ToString();
        break;
    case (2):
        temp[0] = C0_F7.Background.ToString();
        temp[1] = C1_F7.Background.ToString();
        temp[2] = C2_F7.Background.ToString();
        temp[3] = C3_F7.Background.ToString();
        break;
    case (3):
        temp[0] = C0_F6.Background.ToString();
        temp[1] = C1_F6.Background.ToString();
        temp[2] = C2_F6.Background.ToString();
        temp[3] = C3_F6.Background.ToString();
        break;
    case (4):
        temp[0] = C0_F5.Background.ToString();
        temp[1] = C1_F5.Background.ToString();
        temp[2] = C2_F5.Background.ToString();
```

```
    temp[3] = C3_F5.Background.ToString();
    break;
case (5):
    temp[0] = C0_F4.Background.ToString();
    temp[1] = C1_F4.Background.ToString();
    temp[2] = C2_F4.Background.ToString();
    temp[3] = C3_F4.Background.ToString();
    break;
case (6):
    temp[0] = C0_F3.Background.ToString();
    temp[1] = C1_F3.Background.ToString();
    temp[2] = C2_F3.Background.ToString();
    temp[3] = C3_F3.Background.ToString();
    break;
case (7):
    temp[0] = C0_F2.Background.ToString();
    temp[1] = C1_F2.Background.ToString();
    temp[2] = C2_F2.Background.ToString();
    temp[3] = C3_F2.Background.ToString();
    break;
case (8):
    temp[0] = C0_F1.Background.ToString();
    temp[1] = C1_F1.Background.ToString();
    temp[2] = C2_F1.Background.ToString();
    temp[3] = C3_F1.Background.ToString();
    break;
case (9):
    temp[0] = C0_F0.Background.ToString();
    temp[1] = C1_F0.Background.ToString();
    temp[2] = C2_F0.Background.ToString();
    temp[3] = C3_F0.Background.ToString();
```

```
                break;
            }
            return temp;
        }

        // Transforma los colores Brush a su equivalente en Hexadecimal
        private string[] GetCombinationHexColors(string[] temp)
        {
            for (int i = 0; i < temp.Length; i++)
            {
                if (temp[i] == "#FFFFFFFF") temp[i] = "#FFFFFF"; // Conversión del blanco
                else if (temp[i] == "#00000000") temp[i] = "#000000"; // Conversión del
negro
                else if (temp[i] == "#FFFF0000") temp[i] = "#FF0000"; // Conversión del
color rojo
                else if (temp[i] == "#FFFFA500") temp[i] = "#FFA500"; // Conversión del
color amarillo
                else if (temp[i] == "#FF800080") temp[i] = "#800080"; // Conversión del
color morado
                else if (temp[i] == "#FF9ACD32") temp[i] = "#9ACD32"; // Conversión del
color verde
            }
            return temp;
        }

        #endregion

        #region Inicializadores

        // Permite la visualización de los aciertos dados al usuario
        private void SetHits(int[] hits)
        {
```

```
        // Coloca color negro si hay un item en la posición correcta.
        // Coloca color gris si hay un item en la posición incorrecta.
        // Deja en blanco si no existe items acertados.
        switch (count_fila)
        {
            case (0):
                // Primer circulo
                if (hits[0] == 2) A1_F10.Background = new
SolidColorBrush(Colors.Black);
                else if (hits[0] == 1) A1_F10.Background = new
SolidColorBrush(Colors.Gray);
                // Segundo circulo
                if (hits[1] == 2) A2_F10.Background = new
SolidColorBrush(Colors.Black);
                else if (hits[1] == 1) A2_F10.Background = new
SolidColorBrush(Colors.Gray);
                // Tercer circulo
                if (hits[2] == 2) A3_F10.Background = new
SolidColorBrush(Colors.Black);
                else if (hits[3] == 1) A3_F10.Background = new
SolidColorBrush(Colors.Gray);
                // Cuarto circulo
                if (hits[3] == 2) A4_F10.Background = new
SolidColorBrush(Colors.Black);
                else if (hits[3] == 1) A4_F10.Background = new
SolidColorBrush(Colors.Gray);
                break;
            case (1):
                // Primer circulo
                if (hits[0] == 2) A1_F9.Background = new SolidColorBrush(Colors.Black);
                else if (hits[0] == 1) A1_F9.Background = new
SolidColorBrush(Colors.Gray);
                // Segundo circulo
                if (hits[1] == 2) A2_F9.Background = new SolidColorBrush(Colors.Black);
```

```
                else if (hits[1] == 1) A2_F9.Background = new
SolidColorBrush(Colors.Gray);

                // Tercer circulo

                if (hits[2] == 2) A3_F9.Background = new SolidColorBrush(Colors.Black);

                else if (hits[2] == 1) A3_F9.Background = new
SolidColorBrush(Colors.Gray);

                // Cuarto circulo

                if (hits[3] == 2) A4_F9.Background = new SolidColorBrush(Colors.Black);

                else if (hits[3] == 1) A4_F9.Background = new
SolidColorBrush(Colors.Gray);

                break;

            case (2):

                // Primer circulo

                if (hits[0] == 2) A1_F8.Background = new SolidColorBrush(Colors.Black);

                else if (hits[0] == 1) A1_F8.Background = new
SolidColorBrush(Colors.Gray);

                // Segundo circulo

                if (hits[1] == 2) A2_F8.Background = new SolidColorBrush(Colors.Black);

                else if (hits[1] == 1) A2_F8.Background = new
SolidColorBrush(Colors.Gray);

                // Tercer circulo

                if (hits[2] == 2) A3_F8.Background = new SolidColorBrush(Colors.Black);

                else if (hits[2] == 1) A3_F8.Background = new
SolidColorBrush(Colors.Gray);

                // Cuarto circulo

                if (hits[3] == 2) A4_F8.Background = new SolidColorBrush(Colors.Black);

                else if (hits[3] == 1) A4_F8.Background = new
SolidColorBrush(Colors.Gray);

                break;

            case (3):

                // Primer circulo

                if (hits[0] == 2) A1_F7.Background = new SolidColorBrush(Colors.Black);

                else if (hits[0] == 1) A1_F7.Background = new
SolidColorBrush(Colors.Gray);
```

```
                // Segundo circulo
                if (hits[1] == 2) A2_F7.Background = new SolidColorBrush(Colors.Black);
                else if (hits[1] == 1) A2_F7.Background = new
SolidColorBrush(Colors.Gray);
                // Tercer circulo
                if (hits[2] == 2) A3_F7.Background = new SolidColorBrush(Colors.Black);
                else if (hits[2] == 1) A3_F7.Background = new
SolidColorBrush(Colors.Gray);
                // Cuarto circulo
                if (hits[3] == 2) A4_F7.Background = new SolidColorBrush(Colors.Black);
                else if (hits[3] == 1) A4_F7.Background = new
SolidColorBrush(Colors.Gray);
                break;
            case (4):
                // Primer circulo
                if (hits[0] == 2) A1_F6.Background = new SolidColorBrush(Colors.Black);
                else if (hits[0] == 1) A1_F6.Background = new
SolidColorBrush(Colors.Gray);
                // Segundo circulo
                if (hits[1] == 2) A2_F6.Background = new SolidColorBrush(Colors.Black);
                else if (hits[1] == 1) A2_F6.Background = new
SolidColorBrush(Colors.Gray);
                // Tercer circulo
                if (hits[2] == 2) A3_F6.Background = new SolidColorBrush(Colors.Black);
                else if (hits[2] == 1) A3_F6.Background = new
SolidColorBrush(Colors.Gray);
                // Cuarto circulo
                if (hits[3] == 2) A4_F6.Background = new SolidColorBrush(Colors.Black);
                else if (hits[3] == 1) A4_F6.Background = new
SolidColorBrush(Colors.Gray);
                break;
            case (5):
                // Primer circulo
```

```
            if (hits[0] == 2) A1_F5.Background = new SolidColorBrush(Colors.Black);
            else if (hits[0] == 1) A1_F5.Background = new
SolidColorBrush(Colors.Gray);
            // Segundo circulo
            if (hits[1] == 2) A2_F5.Background = new SolidColorBrush(Colors.Black);
            else if (hits[1] == 1) A2_F5.Background = new
SolidColorBrush(Colors.Gray);
            // Tercer circulo
            if (hits[2] == 2) A3_F5.Background = new SolidColorBrush(Colors.Black);
            else if (hits[2] == 1) A3_F5.Background = new
SolidColorBrush(Colors.Gray);
            // Cuarto circulo
            if (hits[3] == 2) A4_F5.Background = new SolidColorBrush(Colors.Black);
            else if (hits[3] == 1) A4_F5.Background = new
SolidColorBrush(Colors.Gray);
            break;
        case (6):
            // Primer circulo
            if (hits[0] == 2) A1_F4.Background = new SolidColorBrush(Colors.Black);
            else if (hits[0] == 1) A1_F4.Background = new
SolidColorBrush(Colors.Gray);
            // Segundo circulo
            if (hits[1] == 2) A2_F4.Background = new SolidColorBrush(Colors.Black);
            else if (hits[1] == 1) A2_F4.Background = new
SolidColorBrush(Colors.Gray);
            // Tercer circulo
            if (hits[2] == 2) A3_F4.Background = new SolidColorBrush(Colors.Black);
            else if (hits[2] == 1) A3_F4.Background = new
SolidColorBrush(Colors.Gray);
            // Cuarto circulo
            if (hits[3] == 2) A4_F4.Background = new SolidColorBrush(Colors.Black);
            else if (hits[3] == 1) A4_F4.Background = new
SolidColorBrush(Colors.Gray);
            break;
```

```
            case (7):
                // Primer circulo
                if (hits[0] == 2) A1_F3.Background = new SolidColorBrush(Colors.Black);
                else if (hits[0] == 1) A1_F3.Background = new
SolidColorBrush(Colors.Gray);
                // Segundo circulo
                if (hits[1] == 2) A2_F3.Background = new SolidColorBrush(Colors.Black);
                else if (hits[1] == 1) A2_F3.Background = new
SolidColorBrush(Colors.Gray);
                // Tercer circulo
                if (hits[2] == 2) A3_F3.Background = new SolidColorBrush(Colors.Black);
                else if (hits[2] == 1) A3_F3.Background = new
SolidColorBrush(Colors.Gray);
                // Cuarto circulo
                if (hits[3] == 2) A4_F3.Background = new SolidColorBrush(Colors.Black);
                else if (hits[3] == 1) A4_F3.Background = new
SolidColorBrush(Colors.Gray);
                break;
            case (8):
                // Primer circulo
                if (hits[0] == 2) A1_F2.Background = new SolidColorBrush(Colors.Black);
                else if (hits[0] == 1) A1_F2.Background = new
SolidColorBrush(Colors.Gray);
                // Segundo circulo
                if (hits[1] == 2) A2_F2.Background = new SolidColorBrush(Colors.Black);
                else if (hits[1] == 1) A2_F2.Background = new
SolidColorBrush(Colors.Gray);
                // Tercer circulo
                if (hits[2] == 2) A3_F2.Background = new SolidColorBrush(Colors.Black);
                else if (hits[2] == 1) A3_F2.Background = new
SolidColorBrush(Colors.Gray);
                // Cuarto circulo
                if (hits[3] == 2) A4_F10.Background = new
SolidColorBrush(Colors.Black);
```

```
                else if (hits[3] == 1) A4_F2.Background = new
SolidColorBrush(Colors.Gray);
                break;
            case (9):
                // Primer circulo
                if (hits[0] == 2) A1_F1.Background = new SolidColorBrush(Colors.Black);
                else if (hits[0] == 1) A1_F1.Background = new
SolidColorBrush(Colors.Gray);
                // Segundo circulo
                if (hits[1] == 2) A2_F1.Background = new SolidColorBrush(Colors.Black);
                else if (hits[1] == 1) A2_F1.Background = new
SolidColorBrush(Colors.Gray);
                // Tercer circulo
                if (hits[2] == 2) A3_F1.Background = new SolidColorBrush(Colors.Black);
                else if (hits[2] == 1) A3_F1.Background = new
SolidColorBrush(Colors.Gray);
                // Cuarto circulo
                if (hits[3] == 2) A4_F1.Background = new SolidColorBrush(Colors.Black);
                else if (hits[3] == 1) A4_F1.Background = new
SolidColorBrush(Colors.Gray);
                break;
        }
    }

    // Inicializa las filas que pueden ser modificadas
    private void InicializarFilas()
    {

        switch (count_fila)
        {
            case 0:
                C0_F0.AllowDrop = false;
```

```
C0_F1.AllowDrop = false;
C0_F2.AllowDrop = false;
C0_F3.AllowDrop = false;
C0_F4.AllowDrop = false;
C0_F5.AllowDrop = false;
C0_F6.AllowDrop = false;
C0_F7.AllowDrop = false;
C0_F8.AllowDrop = false;
C0_F9.AllowDrop = true;

C1_F0.AllowDrop = false;
C1_F1.AllowDrop = false;
C1_F2.AllowDrop = false;
C1_F3.AllowDrop = false;
C1_F4.AllowDrop = false;
C1_F5.AllowDrop = false;
C1_F6.AllowDrop = false;
C1_F7.AllowDrop = false;
C1_F8.AllowDrop = false;
C1_F9.AllowDrop = true;

C2_F0.AllowDrop = false;
C2_F1.AllowDrop = false;
C2_F2.AllowDrop = false;
C2_F3.AllowDrop = false;
C2_F4.AllowDrop = false;
C2_F5.AllowDrop = false;
C2_F6.AllowDrop = false;
C2_F7.AllowDrop = false;
C2_F8.AllowDrop = false;
C2_F9.AllowDrop = true;
```

```
    C3_F0.AllowDrop = false;
    C3_F1.AllowDrop = false;
    C3_F2.AllowDrop = false;
    C3_F3.AllowDrop = false;
    C3_F4.AllowDrop = false;
    C3_F5.AllowDrop = false;
    C3_F6.AllowDrop = false;
    C3_F7.AllowDrop = false;
    C3_F8.AllowDrop = false;
    C3_F9.AllowDrop = true;
    break;
case 1:
    C0_F0.AllowDrop = false;
    C0_F1.AllowDrop = false;
    C0_F2.AllowDrop = false;
    C0_F3.AllowDrop = false;
    C0_F4.AllowDrop = false;
    C0_F5.AllowDrop = false;
    C0_F6.AllowDrop = false;
    C0_F7.AllowDrop = false;
    C0_F8.AllowDrop = true;
    C0_F9.AllowDrop = false;

    C1_F0.AllowDrop = false;
    C1_F1.AllowDrop = false;
    C1_F2.AllowDrop = false;
    C1_F3.AllowDrop = false;
    C1_F4.AllowDrop = false;
    C1_F5.AllowDrop = false;
    C1_F6.AllowDrop = false;
```

```
    C1_F7.AllowDrop = false;
    C1_F8.AllowDrop = true;
    C1_F9.AllowDrop = false;

    C2_F0.AllowDrop = false;
    C2_F1.AllowDrop = false;
    C2_F2.AllowDrop = false;
    C2_F3.AllowDrop = false;
    C2_F4.AllowDrop = false;
    C2_F5.AllowDrop = false;
    C2_F6.AllowDrop = false;
    C2_F7.AllowDrop = false;
    C2_F8.AllowDrop = true;
    C2_F9.AllowDrop = false;

    C3_F0.AllowDrop = false;
    C3_F1.AllowDrop = false;
    C3_F2.AllowDrop = false;
    C3_F3.AllowDrop = false;
    C3_F4.AllowDrop = false;
    C3_F5.AllowDrop = false;
    C3_F6.AllowDrop = false;
    C3_F7.AllowDrop = false;
    C3_F8.AllowDrop = true;
    C3_F9.AllowDrop = false;
    break;
case 2:
    C0_F0.AllowDrop = false;
    C0_F1.AllowDrop = false;
    C0_F2.AllowDrop = false;
    C0_F3.AllowDrop = false;
```

```
C0_F4.AllowDrop = false;
C0_F5.AllowDrop = false;
C0_F6.AllowDrop = false;
C0_F7.AllowDrop = true;
C0_F8.AllowDrop = false;
C0_F9.AllowDrop = false;

C1_F0.AllowDrop = false;
C1_F1.AllowDrop = false;
C1_F2.AllowDrop = false;
C1_F3.AllowDrop = false;
C1_F4.AllowDrop = false;
C1_F5.AllowDrop = false;
C1_F6.AllowDrop = false;
C1_F7.AllowDrop = true;
C1_F8.AllowDrop = false;
C1_F9.AllowDrop = false;

C2_F0.AllowDrop = false;
C2_F1.AllowDrop = false;
C2_F2.AllowDrop = false;
C2_F3.AllowDrop = false;
C2_F4.AllowDrop = false;
C2_F5.AllowDrop = false;
C2_F6.AllowDrop = false;
C2_F7.AllowDrop = true;
C2_F8.AllowDrop = false;
C2_F9.AllowDrop = false;

C3_F0.AllowDrop = false;
C3_F1.AllowDrop = false;
```

```
    C3_F2.AllowDrop = false;
    C3_F3.AllowDrop = false;
    C3_F4.AllowDrop = false;
    C3_F5.AllowDrop = false;
    C3_F6.AllowDrop = false;
    C3_F7.AllowDrop = true;
    C3_F8.AllowDrop = false;
    C3_F9.AllowDrop = false;
    break;
case 3:
    C0_F0.AllowDrop = false;
    C0_F1.AllowDrop = false;
    C0_F2.AllowDrop = false;
    C0_F3.AllowDrop = false;
    C0_F4.AllowDrop = false;
    C0_F5.AllowDrop = false;
    C0_F6.AllowDrop = true;
    C0_F7.AllowDrop = false;
    C0_F8.AllowDrop = false;
    C0_F9.AllowDrop = false;

    C1_F0.AllowDrop = false;
    C1_F1.AllowDrop = false;
    C1_F2.AllowDrop = false;
    C1_F3.AllowDrop = false;
    C1_F4.AllowDrop = false;
    C1_F5.AllowDrop = false;
    C1_F6.AllowDrop = true;
    C1_F7.AllowDrop = false;
    C1_F8.AllowDrop = false;
    C1_F9.AllowDrop = false;
```

```
    C2_F0.AllowDrop = false;
    C2_F1.AllowDrop = false;
    C2_F2.AllowDrop = false;
    C2_F3.AllowDrop = false;
    C2_F4.AllowDrop = false;
    C2_F5.AllowDrop = false;
    C2_F6.AllowDrop = true;
    C2_F7.AllowDrop = false;
    C2_F8.AllowDrop = false;
    C2_F9.AllowDrop = false;

    C3_F0.AllowDrop = false;
    C3_F1.AllowDrop = false;
    C3_F2.AllowDrop = false;
    C3_F3.AllowDrop = false;
    C3_F4.AllowDrop = false;
    C3_F5.AllowDrop = false;
    C3_F6.AllowDrop = true;
    C3_F7.AllowDrop = false;
    C3_F8.AllowDrop = false;
    C3_F9.AllowDrop = false;
    break;
case 4:
    C0_F0.AllowDrop = false;
    C0_F1.AllowDrop = false;
    C0_F2.AllowDrop = false;
    C0_F3.AllowDrop = false;
    C0_F4.AllowDrop = false;
    C0_F5.AllowDrop = true;
    C0_F6.AllowDrop = false;
```

```
C0_F7.AllowDrop = false;
C0_F8.AllowDrop = false;
C0_F9.AllowDrop = false;

C1_F0.AllowDrop = false;
C1_F1.AllowDrop = false;
C1_F2.AllowDrop = false;
C1_F3.AllowDrop = false;
C1_F4.AllowDrop = false;
C1_F5.AllowDrop = true;
C1_F6.AllowDrop = false;
C1_F7.AllowDrop = false;
C1_F8.AllowDrop = false;
C1_F9.AllowDrop = false;

C2_F0.AllowDrop = false;
C2_F1.AllowDrop = false;
C2_F2.AllowDrop = false;
C2_F3.AllowDrop = false;
C2_F4.AllowDrop = false;
C2_F5.AllowDrop = true;
C2_F6.AllowDrop = false;
C2_F7.AllowDrop = false;
C2_F8.AllowDrop = false;
C2_F9.AllowDrop = false;

C3_F0.AllowDrop = false;
C3_F1.AllowDrop = false;
C3_F2.AllowDrop = false;
C3_F3.AllowDrop = false;
C3_F4.AllowDrop = false;
```

```
    C3_F5.AllowDrop = true;
    C3_F6.AllowDrop = false;
    C3_F7.AllowDrop = false;
    C3_F8.AllowDrop = false;
    C3_F9.AllowDrop = false;
    break;
case 5:
    C0_F0.AllowDrop = false;
    C0_F1.AllowDrop = false;
    C0_F2.AllowDrop = false;
    C0_F3.AllowDrop = false;
    C0_F4.AllowDrop = true;
    C0_F5.AllowDrop = false;
    C0_F6.AllowDrop = false;
    C0_F7.AllowDrop = false;
    C0_F8.AllowDrop = false;
    C0_F9.AllowDrop = false;

    C1_F0.AllowDrop = false;
    C1_F1.AllowDrop = false;
    C1_F2.AllowDrop = false;
    C1_F3.AllowDrop = false;
    C1_F4.AllowDrop = true;
    C1_F5.AllowDrop = false;
    C1_F6.AllowDrop = false;
    C1_F7.AllowDrop = false;
    C1_F8.AllowDrop = false;
    C1_F9.AllowDrop = false;

    C2_F0.AllowDrop = false;
    C2_F1.AllowDrop = false;
```

```
    C2_F2.AllowDrop = false;
    C2_F3.AllowDrop = false;
    C2_F4.AllowDrop = true;
    C2_F5.AllowDrop = false;
    C2_F6.AllowDrop = false;
    C2_F7.AllowDrop = false;
    C2_F8.AllowDrop = false;
    C2_F9.AllowDrop = false;

    C3_F0.AllowDrop = false;
    C3_F1.AllowDrop = false;
    C3_F2.AllowDrop = false;
    C3_F3.AllowDrop = false;
    C3_F4.AllowDrop = true;
    C3_F5.AllowDrop = false;
    C3_F6.AllowDrop = false;
    C3_F7.AllowDrop = false;
    C3_F8.AllowDrop = false;
    C3_F9.AllowDrop = false;
    break;
case 6:
    C0_F0.AllowDrop = false;
    C0_F1.AllowDrop = false;
    C0_F2.AllowDrop = false;
    C0_F3.AllowDrop = true;
    C0_F4.AllowDrop = false;
    C0_F5.AllowDrop = false;
    C0_F6.AllowDrop = false;
    C0_F7.AllowDrop = false;
    C0_F8.AllowDrop = false;
    C0_F9.AllowDrop = false;
```

```
C1_F0.AllowDrop = false;
C1_F1.AllowDrop = false;
C1_F2.AllowDrop = false;
C1_F3.AllowDrop = true;
C1_F4.AllowDrop = false;
C1_F5.AllowDrop = false;
C1_F6.AllowDrop = false;
C1_F7.AllowDrop = false;
C1_F8.AllowDrop = false;
C1_F9.AllowDrop = false;

C2_F0.AllowDrop = false;
C2_F1.AllowDrop = false;
C2_F2.AllowDrop = false;
C2_F3.AllowDrop = true;
C2_F4.AllowDrop = false;
C2_F5.AllowDrop = false;
C2_F6.AllowDrop = false;
C2_F7.AllowDrop = false;
C2_F8.AllowDrop = false;
C2_F9.AllowDrop = false;

C3_F0.AllowDrop = false;
C3_F1.AllowDrop = false;
C3_F2.AllowDrop = false;
C3_F3.AllowDrop = true;
C3_F4.AllowDrop = false;
C3_F5.AllowDrop = false;
C3_F6.AllowDrop = false;
C3_F7.AllowDrop = false;
```

```
    C3_F8.AllowDrop = false;
    C3_F9.AllowDrop = false;
    break;
case 7:
    C0_F0.AllowDrop = false;
    C0_F1.AllowDrop = false;
    C0_F2.AllowDrop = true;
    C0_F3.AllowDrop = false;
    C0_F4.AllowDrop = false;
    C0_F5.AllowDrop = false;
    C0_F6.AllowDrop = false;
    C0_F7.AllowDrop = false;
    C0_F8.AllowDrop = false;
    C0_F9.AllowDrop = false;

    C1_F0.AllowDrop = false;
    C1_F1.AllowDrop = false;
    C1_F2.AllowDrop = true;
    C1_F3.AllowDrop = false;
    C1_F4.AllowDrop = false;
    C1_F5.AllowDrop = false;
    C1_F6.AllowDrop = false;
    C1_F7.AllowDrop = false;
    C1_F8.AllowDrop = false;
    C1_F9.AllowDrop = false;

    C2_F0.AllowDrop = false;
    C2_F1.AllowDrop = false;
    C2_F2.AllowDrop = true;
    C2_F3.AllowDrop = false;
    C2_F4.AllowDrop = false;
```

```
    C2_F5.AllowDrop = false;
    C2_F6.AllowDrop = false;
    C2_F7.AllowDrop = false;
    C2_F8.AllowDrop = false;
    C2_F9.AllowDrop = false;

    C3_F0.AllowDrop = false;
    C3_F1.AllowDrop = false;
    C3_F2.AllowDrop = true;
    C3_F3.AllowDrop = false;
    C3_F4.AllowDrop = false;
    C3_F5.AllowDrop = false;
    C3_F6.AllowDrop = false;
    C3_F7.AllowDrop = false;
    C3_F8.AllowDrop = false;
    C3_F9.AllowDrop = false;
    break;
case 8:
    C0_F0.AllowDrop = false;
    C0_F1.AllowDrop = true;
    C0_F2.AllowDrop = false;
    C0_F3.AllowDrop = false;
    C0_F4.AllowDrop = false;
    C0_F5.AllowDrop = false;
    C0_F6.AllowDrop = false;
    C0_F7.AllowDrop = false;
    C0_F8.AllowDrop = false;
    C0_F9.AllowDrop = false;

    C1_F0.AllowDrop = false;
    C1_F1.AllowDrop = true;
```

```
C1_F2.AllowDrop = false;
C1_F3.AllowDrop = false;
C1_F4.AllowDrop = false;
C1_F5.AllowDrop = false;
C1_F6.AllowDrop = false;
C1_F7.AllowDrop = false;
C1_F8.AllowDrop = false;
C1_F9.AllowDrop = false;

C2_F0.AllowDrop = false;
C2_F1.AllowDrop = true;
C2_F2.AllowDrop = false;
C2_F3.AllowDrop = false;
C2_F4.AllowDrop = false;
C2_F5.AllowDrop = false;
C2_F6.AllowDrop = false;
C2_F7.AllowDrop = false;
C2_F8.AllowDrop = false;
C2_F9.AllowDrop = false;

C3_F0.AllowDrop = false;
C3_F1.AllowDrop = true;
C3_F2.AllowDrop = false;
C3_F3.AllowDrop = false;
C3_F4.AllowDrop = false;
C3_F5.AllowDrop = false;
C3_F6.AllowDrop = false;
C3_F7.AllowDrop = false;
C3_F8.AllowDrop = false;
C3_F9.AllowDrop = false;
break;
```

```
case 9:
    C0_F0.AllowDrop = true;
    C0_F1.AllowDrop = false;
    C0_F2.AllowDrop = false;
    C0_F3.AllowDrop = false;
    C0_F4.AllowDrop = false;
    C0_F5.AllowDrop = false;
    C0_F6.AllowDrop = false;
    C0_F7.AllowDrop = false;
    C0_F8.AllowDrop = false;
    C0_F9.AllowDrop = false;

    C1_F0.AllowDrop = true;
    C1_F1.AllowDrop = false;
    C1_F2.AllowDrop = false;
    C1_F3.AllowDrop = false;
    C1_F4.AllowDrop = false;
    C1_F5.AllowDrop = false;
    C1_F6.AllowDrop = false;
    C1_F7.AllowDrop = false;
    C1_F8.AllowDrop = false;
    C1_F9.AllowDrop = false;

    C2_F0.AllowDrop = true;
    C2_F1.AllowDrop = false;
    C2_F2.AllowDrop = false;
    C2_F3.AllowDrop = false;
    C2_F4.AllowDrop = false;
    C2_F5.AllowDrop = false;
    C2_F6.AllowDrop = false;
```

```
            C2_F7.AllowDrop = false;
            C2_F8.AllowDrop = false;
            C2_F9.AllowDrop = false;

            C3_F0.AllowDrop = true;
            C3_F1.AllowDrop = false;
            C3_F2.AllowDrop = false;
            C3_F3.AllowDrop = false;
            C3_F4.AllowDrop = false;
            C3_F5.AllowDrop = false;
            C3_F6.AllowDrop = false;
            C3_F7.AllowDrop = false;
            C3_F8.AllowDrop = false;
            C3_F9.AllowDrop = false;
            break;

    }
}

// Reinicia todas las filas a su valor inicial -> en blanco
private void ReiniciarFilas()
{
    // Primeras columnas de todas las filas
    C0_F0.Background = new SolidColorBrush(Colors.White);
    C0_F1.Background = new SolidColorBrush(Colors.White);
    C0_F2.Background = new SolidColorBrush(Colors.White);
    C0_F3.Background = new SolidColorBrush(Colors.White);
    C0_F4.Background = new SolidColorBrush(Colors.White);
    C0_F5.Background = new SolidColorBrush(Colors.White);
    C0_F6.Background = new SolidColorBrush(Colors.White);
    C0_F7.Background = new SolidColorBrush(Colors.White);
```

```
C0_F8.Background = new SolidColorBrush(Colors.White);
C0_F9.Background = new SolidColorBrush(Colors.White);
// Segundas columnas de todas las filas
C1_F0.Background = new SolidColorBrush(Colors.White);
C1_F1.Background = new SolidColorBrush(Colors.White);
C1_F2.Background = new SolidColorBrush(Colors.White);
C1_F3.Background = new SolidColorBrush(Colors.White);
C1_F4.Background = new SolidColorBrush(Colors.White);
C1_F5.Background = new SolidColorBrush(Colors.White);
C1_F6.Background = new SolidColorBrush(Colors.White);
C1_F7.Background = new SolidColorBrush(Colors.White);
C1_F8.Background = new SolidColorBrush(Colors.White);
C1_F9.Background = new SolidColorBrush(Colors.White);
// Terceras columnas de todas las filas
C2_F0.Background = new SolidColorBrush(Colors.White);
C2_F1.Background = new SolidColorBrush(Colors.White);
C2_F2.Background = new SolidColorBrush(Colors.White);
C2_F3.Background = new SolidColorBrush(Colors.White);
C2_F4.Background = new SolidColorBrush(Colors.White);
C2_F5.Background = new SolidColorBrush(Colors.White);
C2_F6.Background = new SolidColorBrush(Colors.White);
C2_F7.Background = new SolidColorBrush(Colors.White);
C2_F8.Background = new SolidColorBrush(Colors.White);
C2_F9.Background = new SolidColorBrush(Colors.White);
// Cuartas columnas de todas las filas
C3_F0.Background = new SolidColorBrush(Colors.White);
C3_F1.Background = new SolidColorBrush(Colors.White);
C3_F2.Background = new SolidColorBrush(Colors.White);
C3_F3.Background = new SolidColorBrush(Colors.White);
C3_F4.Background = new SolidColorBrush(Colors.White);
C3_F5.Background = new SolidColorBrush(Colors.White);
```

```
C3_F6.Background = new SolidColorBrush(Colors.White);
C3_F7.Background = new SolidColorBrush(Colors.White);
C3_F8.Background = new SolidColorBrush(Colors.White);
C3_F9.Background = new SolidColorBrush(Colors.White);
// Acierto 1 de todas las filas
A1_F1.Background = new SolidColorBrush(Colors.White);
A1_F2.Background = new SolidColorBrush(Colors.White);
A1_F3.Background = new SolidColorBrush(Colors.White);
A1_F4.Background = new SolidColorBrush(Colors.White);
A1_F5.Background = new SolidColorBrush(Colors.White);
A1_F6.Background = new SolidColorBrush(Colors.White);
A1_F7.Background = new SolidColorBrush(Colors.White);
A1_F8.Background = new SolidColorBrush(Colors.White);
A1_F9.Background = new SolidColorBrush(Colors.White);
A1_F10.Background = new SolidColorBrush(Colors.White);
// Acierto 2 de todas las filas
A2_F1.Background = new SolidColorBrush(Colors.White);
A2_F2.Background = new SolidColorBrush(Colors.White);
A2_F3.Background = new SolidColorBrush(Colors.White);
A2_F4.Background = new SolidColorBrush(Colors.White);
A2_F5.Background = new SolidColorBrush(Colors.White);
A2_F6.Background = new SolidColorBrush(Colors.White);
A2_F7.Background = new SolidColorBrush(Colors.White);
A2_F8.Background = new SolidColorBrush(Colors.White);
A2_F9.Background = new SolidColorBrush(Colors.White);
A2_F10.Background = new SolidColorBrush(Colors.White);
// Acierto 3 de todas las filas
A3_F1.Background = new SolidColorBrush(Colors.White);
A3_F2.Background = new SolidColorBrush(Colors.White);
A3_F3.Background = new SolidColorBrush(Colors.White);
A3_F4.Background = new SolidColorBrush(Colors.White);
```

```
    A3_F5.Background = new SolidColorBrush(Colors.White);
    A3_F6.Background = new SolidColorBrush(Colors.White);
    A3_F7.Background = new SolidColorBrush(Colors.White);
    A3_F8.Background = new SolidColorBrush(Colors.White);
    A3_F9.Background = new SolidColorBrush(Colors.White);
    A3_F10.Background = new SolidColorBrush(Colors.White);
    // Acierto 4 de todas las filas
    A4_F1.Background = new SolidColorBrush(Colors.White);
    A4_F2.Background = new SolidColorBrush(Colors.White);
    A4_F3.Background = new SolidColorBrush(Colors.White);
    A4_F4.Background = new SolidColorBrush(Colors.White);
    A4_F5.Background = new SolidColorBrush(Colors.White);
    A4_F6.Background = new SolidColorBrush(Colors.White);
    A4_F7.Background = new SolidColorBrush(Colors.White);
    A4_F8.Background = new SolidColorBrush(Colors.White);
    A4_F9.Background = new SolidColorBrush(Colors.White);
    A4_F10.Background = new SolidColorBrush(Colors.White);
}

// Inicializa el puntaje inicial
private void InicializarPuntaje()
{
    puntaje_actual.Text = score.ToString();
    puntaje_maximo.Text = InitMaxScore();
    // Identifica que nivel se esta jugando
    var id = game.GetIdLevel();
}

// Trae puntajes máximos existentes
private string InitMaxScore()
{
```

```
    string curFile = @"scores.txt";
    // Verifica si existen datos guardados
    if (File.Exists(curFile))
    {
        string[] scores = new string[3];
        int count = 0;
        var linesRead = File.ReadLines("scores.txt");
        foreach(var iter in linesRead) scores[count++] = iter;
        if (game.GetIdLevel() == 1) return scores[0];
        else if(game.GetIdLevel() == 2) return scores[1];
        else if(game.GetIdLevel() == 3) return scores[2];
    }
    else
    {
        using(FileStream fs = File.Create(curFile))
        {
            Byte[] temp_score = new UTF8Encoding(true).GetBytes("0\n");
            fs.Write(temp_score, 0, temp_score.Length);
            fs.Write(temp_score, 0, temp_score.Length);
            fs.Write(temp_score, 0, temp_score.Length);
        }
    }
    return "0";
}

#endregion

#region Notificaciones

// Muestra un mensaje de que el jugador gano
private void NotifyWon(int p_score)
```

```
        {
            string message = $"¡Felicitaciones!\n\n ¡Has pasado el nivel! Tú puntaje fue de {p_score}.";
            string caption = "¡Has ganado!";
            MessageBoxButton button = MessageBoxButton.OK;
            MessageBoxResult result = MessageBox.Show(message, caption, button);
            switch (result)
            {
                // Vuelve al menú de niveles
                case MessageBoxResult.OK:
                    NavigationService.Navigate(new SelectLevel());
                    break;
            }
        }

        // Muestra un mensaje de que el jugador perdio
        private void NotifyLost()
        {
            string message = "¡Oh No! No has logrado superar el nivel...\n\n¿Deseas intentar de nuevo?";
            string caption = "¡Perdiste!";
            MessageBoxButton button = MessageBoxButton.YesNo;
            MessageBoxResult result = MessageBox.Show(message, caption, button);
            switch (result)
            {
                // Reinicia el nivel
                case MessageBoxResult.Yes:
                    Reboot();
                    break;

                // Vuelve al menú de niveles
```

```
            case MessageBoxResult.No:
                NavigationService.Navigate(new SelectLevel());
                break;
        }
    }

    // Muestra las configuraciones del nivel
    private void NotifyGameSettings()
    {
        string message = "¡Bienvenido!\n\nHas seleccionado el nivel de dificultad fácil, tienes 4 colores para crear tus combinaciones pero estos no se pueden repetir.";
        string caption = "Nivel: Fácil";
        MessageBoxButton button = MessageBoxButton.OK;
        MessageBoxResult result = MessageBox.Show(message, caption, button);
        switch (result)
        {
            // No ocurre nada
            case MessageBoxResult.OK:
                break;
        }
    }

    // Notifica que un color ha sido repetido
    private void NotifyRepeatedColor()
    {
        string message = "¡Has repetido un color!\n¡Recuerda que en este nivel no esta permitido su repetición!";
        string caption = "Advertencia";
        MessageBoxButton button = MessageBoxButton.OK;
        MessageBoxResult result = MessageBox.Show(message, caption, button);
        switch (result)
```

```
            {
                // No sucede nada
                case MessageBoxResult.OK:
                    break;
            }
        }

        // Notifica si la combinación fue ejecutada vacía
        private void NotifyEmpty()
        {
            string message = "Parece que no has colocado ningún tipo de combinación...\n\n¡Coloca alguna antes de probar!";
            string caption = "Advertencia";
            MessageBoxButton button = MessageBoxButton.OK;
            MessageBoxResult result = MessageBox.Show(message, caption, button);
        }

        // Notifica que hay un item en blanco en la combinación
        private void NotifyOneEmpty()
        {
            string message = "Parece que has dejado un espacio en blanco...\n\n¡Ten más cuidado la próxima vez!";
            string caption = "Advertencia";
            MessageBoxButton button = MessageBoxButton.OK;
            MessageBoxResult result = MessageBox.Show(message, caption, button);
        }

        // Notifica que se logro un nuevo record
        private void NotifyMaxScore(int p_score)
        {
            string message = $"¡Has logrado un nuevo record!\n\nObtuviste un puntaje de: {p_score}";
```

```
            string caption = "Felicitaciones";
            MessageBoxButton button = MessageBoxButton.OK;
            MessageBoxResult result = MessageBox.Show(message, caption, button);
        }

        // Notifica las instrucciones del juego
        private void NotifyRules()
        {
            string message = "¡Recuerda!\n\nCada vez que se empieza una partida, el juego creará un nuevo código secreto que lo debes decifrar en un máximo de 10 intentos que le brindaràn de pistas para resolver el código." +

"\n\n**********************************************************************\n\n" +
                "Sobre el juego:\n\nPara poder acertar con mayor facilidad el código secreto ¡podrás guiarte de las pistas que el juego te proveerá:\n" +
                "1. El color negro significa que tienes un color en la posición correcta!\n" +
                "2. El color gris significa que tienes un color correcta pero en la posición incorrecta!";
            string caption = "Reglas del juego";
            MessageBoxButton buttons = MessageBoxButton.OK;
            MessageBoxResult result = MessageBox.Show(message, caption, buttons);
        }

        #endregion

    }
}
```

***Anexo 3: Código fuente del nivel medio***

```
using System;
using System.Collections.Generic;
```

```
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using MasterMind.Classes;
using System.Threading;
using System.IO;

namespace MasterMind.Pages
{
    public class ColoresDataMedio : ObservableCollection<SolidColorBrush>
    {
        //Donde se almacenan los colores que se van a utilizar
        public ColoresDataMedio() : base()
        {
            Add(new SolidColorBrush(Colors.Red));
            Add(new SolidColorBrush(Colors.Orange));
            Add(new SolidColorBrush(Colors.Purple));
            Add(new SolidColorBrush(Colors.YellowGreen));
            Add(new SolidColorBrush(Colors.Black));
            Add(new SolidColorBrush(Colors.Coral));
            Add(new SolidColorBrush(Colors.Fuchsia));
```

```
        Add(new SolidColorBrush(Colors.Navy));
    }
}
/// <summary>
/// Interaction logic for Page2_m.xaml
/// </summary>
public partial class Page2_m : Page
{
    #region Variables

    // Inicializamos el juego para usarse.
    Mastermind game;
    // Permite verificar en que intento se encuentra el usuario
    int count_fila;
    // Variable del puntaje
    int score = 10000;

    #endregion

    public Page2_m()
    {
        InitializeComponent();
        // Crea el juego
        game = new Mastermind(2);
        // Inicializa los intentos
        count_fila = 0;
        // Inicializa la jugabilidad
        InicializarFilas();
        // Inicializa los puntajes
        InicializarPuntaje();
        // Notifica datos del modo de juego
```

```
            var thread = new Thread(() => { NotifyGameSettings(); });
            thread.Start();
        }

        #region GUI

        //evento de drag and drop
        private void Ellipse_MouseMove(object sender, MouseEventArgs e)
        {
            if (sender is Ellipse ellipse && e.LeftButton == MouseButtonState.Pressed)
            {
                DragDrop.DoDragDrop(ellipse, ellipse.Fill.ToString(),
DragDropEffects.Copy);
            }
        }

        // Pinta un circulo con el color deseado
        private void Border_Drop(object sender, DragEventArgs e)
        {
            if (sender is Border border)
            {
                if (e.Data.GetDataPresent(DataFormats.StringFormat))
                {
                    string stringFromDrop =
(string)e.Data.GetData(DataFormats.StringFormat);
                    BrushConverter convertir = new BrushConverter();
                    if (convertir.IsValid(stringFromDrop))
                    {
                        Brush pincel = (Brush)convertir.ConvertFromString(stringFromDrop);
                        // Trae todos los colores utilizados en la fila actual
                        string[] combination = GetCombination();
```

```
                // Establece el nuevo color del círculo
                border.Background = pincel;
                // Evalua si el color ya fue utilizado en la creación del código
                for (int i = 0; i < combination.Length; i++)
                {
                    if (combination[i] == border.Background.ToString())
                    {
                        // Si lo fue elimina la anterior posición del color y la coloca en la
nueva
                        ChangePosition(i);
                    }
                }
            }
        }
    }
}

// Vuelta al menú de selección de niveles
private void Button_Click(object sender, RoutedEventArgs e)
{
    NavigationService.Navigate(new SelectLevel());
}

// Reinicia el nivel
private void reboot_btn_Click(object sender, RoutedEventArgs e)
{
    Reboot();
}

// Permite visualizar las instrucciones del juego
private void instrucciones_btn_Click(object sender, RoutedEventArgs e)
```

```
        {
            NotifyRules();
        }

        // Confirmación de combinación
        private void confirmar_btn_Click(object sender, RoutedEventArgs e)
        {
            // Tomamos la combinación que se esta probando al momento.
            string[] combination = GetCombination();
            // Transformamos los colores a hexadecimal
            string[] combination_colors = GetCombinationHexColors(combination);
            // Verificamos si la combinación no tiene datos
            if (!CheckEmpty(combination_colors))
            {
                // Verficamos que todos los colores han sido utilizados y no se han dejado
espacios en blanco
                if (!CheckOneEmpty(combination_colors))
                {
                    // Probamos la combinación
                    var hits = game.TryCombination(combination_colors);
                    // Configuramos los aciertos dados y los mostramos al usuario
                    SetHits(hits);
                    // Incrementamos el contador del intento
                    count_fila++;
                    // Verificar si gano el juego el jugador
                    if (PlayerWon(hits))
                    {
                        if (CheckNewMaxScore())
                        {
                            NotifyMaxScore(Int32.Parse(puntaje_actual.Text));
                            SetMaxScore();
```

```
            }
            NotifyWon(Int32.Parse(puntaje_actual.Text));
        }

        // Actualiza el score del jugador
        score = score - 1000;
        puntaje_actual.Text = score.ToString();

        // Verifica si perdio el juego el jugador
        if (PlayerLost() & !PlayerWon(hits))
        {
            NotifyLost();
        }
        // Actualiza las filas que pueden ser modificadas
        if (count_fila < 10) InicializarFilas();
      }
      else
      {
        NotifyOneEmpty();
      }
    }
    else
    {
      NotifyEmpty();
    }
}

#endregion

#region Métodos críticos
```

```
// Verifica si el jugador gano
private bool PlayerWon(int[] hits)
{
    foreach (var iter in hits) if (iter != 2) return false;
    return true;
}

// Verifica si el jugador perdio
private bool PlayerLost()
{
    if (count_fila > 9) return true;
    return false;
}

// Reinicia el nivel
private void Reboot()
{
    // Establece un nuevo código secreto
    if (count_fila > 0) game.InitSecret();
    // Reestablece los intentos
    count_fila = 0;
    // Reestablece el puntaje actual
    score = 10000;
    // Inicializa la disponibilidad de las filas
    InicializarFilas();
    // Reinicia todos los datos del juego actual
    ReiniciarFilas();
}

// Hace que una posición  se inicialice
private void ChangePosition(int position)
```

```
    {
        switch (count_fila)
        {
            case (0):
                if (position == 0) C0_F9.Background = new
SolidColorBrush(Colors.White);
                else if (position == 1) C1_F9.Background = new
SolidColorBrush(Colors.White);
                else if (position == 2) C2_F9.Background = new
SolidColorBrush(Colors.White);
                else if (position == 3) C3_F9.Background = new
SolidColorBrush(Colors.White);
                break;
            case (1):
                if (position == 0) C0_F8.Background = new
SolidColorBrush(Colors.White);
                else if (position == 1) C1_F8.Background = new
SolidColorBrush(Colors.White);
                else if (position == 2) C2_F8.Background = new
SolidColorBrush(Colors.White);
                else if (position == 3) C3_F8.Background = new
SolidColorBrush(Colors.White);
                break;
            case (2):
                if (position == 0) C0_F7.Background = new
SolidColorBrush(Colors.White);
                else if (position == 1) C1_F7.Background = new
SolidColorBrush(Colors.White);
                else if (position == 2) C2_F7.Background = new
SolidColorBrush(Colors.White);
                else if (position == 3) C3_F7.Background = new
SolidColorBrush(Colors.White);
                break;
            case (3):
```

```
                if (position == 0) C0_F6.Background = new
SolidColorBrush(Colors.White);

                else if (position == 1) C1_F6.Background = new
SolidColorBrush(Colors.White);

                else if (position == 2) C2_F6.Background = new
SolidColorBrush(Colors.White);

                else if (position == 3) C3_F6.Background = new
SolidColorBrush(Colors.White);

                break;

            case (4):

                if (position == 0) C0_F5.Background = new
SolidColorBrush(Colors.White);

                else if (position == 1) C1_F5.Background = new
SolidColorBrush(Colors.White);

                else if (position == 2) C2_F5.Background = new
SolidColorBrush(Colors.White);

                else if (position == 3) C3_F5.Background = new
SolidColorBrush(Colors.White);

                break;

            case (5):

                if (position == 0) C0_F4.Background = new
SolidColorBrush(Colors.White);

                else if (position == 1) C1_F4.Background = new
SolidColorBrush(Colors.White);

                else if (position == 2) C2_F4.Background = new
SolidColorBrush(Colors.White);

                else if (position == 3) C3_F4.Background = new
SolidColorBrush(Colors.White);

                break;

            case (6):

                if (position == 0) C0_F3.Background = new
SolidColorBrush(Colors.White);

                else if (position == 1) C1_F3.Background = new
SolidColorBrush(Colors.White);

                else if (position == 2) C2_F3.Background = new
SolidColorBrush(Colors.White);
```

```
                else if (position == 3) C3_F3.Background = new
SolidColorBrush(Colors.White);

                break;

            case (7):

                if (position == 0) C0_F2.Background = new
SolidColorBrush(Colors.White);

                else if (position == 1) C1_F2.Background = new
SolidColorBrush(Colors.White);

                else if (position == 2) C2_F2.Background = new
SolidColorBrush(Colors.White);

                else if (position == 3) C3_F2.Background = new
SolidColorBrush(Colors.White);

                break;

            case (8):

                if (position == 0) C0_F1.Background = new
SolidColorBrush(Colors.White);

                else if (position == 1) C1_F1.Background = new
SolidColorBrush(Colors.White);

                else if (position == 2) C2_F1.Background = new
SolidColorBrush(Colors.White);

                else if (position == 3) C3_F1.Background = new
SolidColorBrush(Colors.White);

                break;

            case (9):

                if (position == 0) C0_F0.Background = new
SolidColorBrush(Colors.White);

                else if (position == 1) C1_F0.Background = new
SolidColorBrush(Colors.White);

                else if (position == 2) C2_F0.Background = new
SolidColorBrush(Colors.White);

                else if (position == 3) C3_F0.Background = new
SolidColorBrush(Colors.White);

                break;

        }

    }
```

```
// Verifica si la combinación esta vacía
private bool CheckEmpty(string[] colors)
{
    foreach (var iter in colors) if (iter != "#FFFFFF") return false;
    return true;
}

// Verifica que no existan items en blanco para las combinaciones de los intentos
private bool CheckOneEmpty(string[] colors)
{
    foreach (var iter in colors) if (iter == "#FFFFFF") return true;
    return false;
}

// Establece el puntaje máximo
private void SetMaxScore()
{
    // Identifica que nivel se esta jugando
    var id = game.GetIdLevel();
    // Variables para modificar el archivo de puntajes
    string filename = "scores.txt";
    string[] scores = new string[3];
    int count = 0;

    // Lee los datos y guarda en scores
    var linesRead = File.ReadLines("scores.txt");
    foreach (var iter in linesRead)
    {
        scores[count++] = iter;
    }
```

```
    // Borra el archivo
    if (File.Exists(filename)) File.Delete(filename);

    // Establece la nueva información
    if (id == 1)
    {
        scores[0] = score.ToString();
    }
    else if (id == 2)
    {
        scores[1] = score.ToString();
    }
    else if (id == 3)
    {
        scores[2] = score.ToString();
    }

    // Guarda la nueva información
    using (FileStream fs = File.Create(filename))
    {
        Byte[] score_1 = new UTF8Encoding(true).GetBytes(scores[0] + "\n");
        Byte[] score_2 = new UTF8Encoding(true).GetBytes(scores[1] + "\n");
        Byte[] score_3 = new UTF8Encoding(true).GetBytes(scores[2] + "\n");
        fs.Write(score_1, 0, score_1.Length);
        fs.Write(score_2, 0, score_2.Length);
        fs.Write(score_3, 0, score_3.Length);
    }
}

// Evalua si se logro un nuevo marcador
```

```
private bool CheckNewMaxScore()
{
    if (score > Int32.Parse(puntaje_maximo.Text)) return true;
    return false;
}

#endregion

#region Getters

// Devuelve la combinación que se esta probando
private string[] GetCombination()
{
    string[] temp = new string[4];
    // El número del case + 1 indica la fila que se esta probando.
    switch (count_fila)
    {
        case (0):
            temp[0] = C0_F9.Background.ToString();
            temp[1] = C1_F9.Background.ToString();
            temp[2] = C2_F9.Background.ToString();
            temp[3] = C3_F9.Background.ToString();
            break;
        case (1):
            temp[0] = C0_F8.Background.ToString();
            temp[1] = C1_F8.Background.ToString();
            temp[2] = C2_F8.Background.ToString();
            temp[3] = C3_F8.Background.ToString();
            break;
        case (2):
            temp[0] = C0_F7.Background.ToString();
```

```
    temp[1] = C1_F7.Background.ToString();
    temp[2] = C2_F7.Background.ToString();
    temp[3] = C3_F7.Background.ToString();
    break;
case (3):
    temp[0] = C0_F6.Background.ToString();
    temp[1] = C1_F6.Background.ToString();
    temp[2] = C2_F6.Background.ToString();
    temp[3] = C3_F6.Background.ToString();
    break;
case (4):
    temp[0] = C0_F5.Background.ToString();
    temp[1] = C1_F5.Background.ToString();
    temp[2] = C2_F5.Background.ToString();
    temp[3] = C3_F5.Background.ToString();
    break;
case (5):
    temp[0] = C0_F4.Background.ToString();
    temp[1] = C1_F4.Background.ToString();
    temp[2] = C2_F4.Background.ToString();
    temp[3] = C3_F4.Background.ToString();
    break;
case (6):
    temp[0] = C0_F3.Background.ToString();
    temp[1] = C1_F3.Background.ToString();
    temp[2] = C2_F3.Background.ToString();
    temp[3] = C3_F3.Background.ToString();
    break;
case (7):
    temp[0] = C0_F2.Background.ToString();
    temp[1] = C1_F2.Background.ToString();
```

```
                temp[2] = C2_F2.Background.ToString();
                temp[3] = C3_F2.Background.ToString();
                break;
            case (8):
                temp[0] = C0_F1.Background.ToString();
                temp[1] = C1_F1.Background.ToString();
                temp[2] = C2_F1.Background.ToString();
                temp[3] = C3_F1.Background.ToString();
                break;
            case (9):
                temp[0] = C0_F0.Background.ToString();
                temp[1] = C1_F0.Background.ToString();
                temp[2] = C2_F0.Background.ToString();
                temp[3] = C3_F0.Background.ToString();
                break;
        }
        return temp;
    }

    // Transforma los colores Brush a su equivalente en Hexadecimal
    private string[] GetCombinationHexColors(string[] temp)
    {
        for (int i = 0; i < temp.Length; i++)
        {
            if (temp[i] == "#FFFFFFFF") temp[i] = "#FFFFFF"; // Conversión del blanco
            else if (temp[i] == "#FFFF0000") temp[i] = "#FF0000"; // Conversión del
color rojo
            else if (temp[i] == "#FFFFA500") temp[i] = "#FFA500"; // Conversión del
color amarillo
            else if (temp[i] == "#FF800080") temp[i] = "#800080"; // Conversión del
color morado
```

```
            else if (temp[i] == "#FF9ACD32") temp[i] = "#9ACD32"; // Conversión del color verde
            else if (temp[i] == "#00000000") temp[i] = "#000000"; // Conversión del color negro
            else if (temp[i] == "#FFFF7F50") temp[i] = "#FF7F50"; // Conversión del color anaranjado
            else if (temp[i] == "#FFFF00FF") temp[i] = "#FF00FF"; // Conversión del color rosado
            else if (temp[i] == "#FF000080") temp[i] = "#000080"; // Conversión del color azul
        }
        return temp;
    }

    #endregion

    #region Inicializadores

    // Permite la visualización de los aciertos dados al usuario
    private void SetHits(int[] hits)
    {
        // Coloca color negro si hay un item en la posición correcta.
        // Coloca color gris si hay un item en la posición incorrecta.
        // Deja en blanco si no existe items acertados.
        switch (count_fila)
        {
            case (0):
                // Primer circulo
                if (hits[0] == 2) A1_F10.Background = new SolidColorBrush(Colors.Black);
                else if (hits[0] == 1) A1_F10.Background = new SolidColorBrush(Colors.Gray);
                // Segundo circulo
```

```
            if (hits[1] == 2) A2_F10.Background = new
SolidColorBrush(Colors.Black);

            else if (hits[1] == 1) A2_F10.Background = new
SolidColorBrush(Colors.Gray);

            // Tercer circulo

            if (hits[2] == 2) A3_F10.Background = new
SolidColorBrush(Colors.Black);

            else if (hits[3] == 1) A3_F10.Background = new
SolidColorBrush(Colors.Gray);

            // Cuarto circulo

            if (hits[3] == 2) A4_F10.Background = new
SolidColorBrush(Colors.Black);

            else if (hits[3] == 1) A4_F10.Background = new
SolidColorBrush(Colors.Gray);

            break;

        case (1):

            // Primer circulo

            if (hits[0] == 2) A1_F9.Background = new SolidColorBrush(Colors.Black);

            else if (hits[0] == 1) A1_F9.Background = new
SolidColorBrush(Colors.Gray);

            // Segundo circulo

            if (hits[1] == 2) A2_F9.Background = new SolidColorBrush(Colors.Black);

            else if (hits[1] == 1) A2_F9.Background = new
SolidColorBrush(Colors.Gray);

            // Tercer circulo

            if (hits[2] == 2) A3_F9.Background = new SolidColorBrush(Colors.Black);

            else if (hits[2] == 1) A3_F9.Background = new
SolidColorBrush(Colors.Gray);

            // Cuarto circulo

            if (hits[3] == 2) A4_F9.Background = new SolidColorBrush(Colors.Black);

            else if (hits[3] == 1) A4_F9.Background = new
SolidColorBrush(Colors.Gray);

            break;

        case (2):
```

```
                // Primer circulo
                if (hits[0] == 2) A1_F8.Background = new SolidColorBrush(Colors.Black);
                else if (hits[0] == 1) A1_F8.Background = new
SolidColorBrush(Colors.Gray);
                // Segundo circulo
                if (hits[1] == 2) A2_F8.Background = new SolidColorBrush(Colors.Black);
                else if (hits[1] == 1) A2_F8.Background = new
SolidColorBrush(Colors.Gray);
                // Tercer circulo
                if (hits[2] == 2) A3_F8.Background = new SolidColorBrush(Colors.Black);
                else if (hits[2] == 1) A3_F8.Background = new
SolidColorBrush(Colors.Gray);
                // Cuarto circulo
                if (hits[3] == 2) A4_F8.Background = new SolidColorBrush(Colors.Black);
                else if (hits[3] == 1) A4_F8.Background = new
SolidColorBrush(Colors.Gray);
                break;
            case (3):
                // Primer circulo
                if (hits[0] == 2) A1_F7.Background = new SolidColorBrush(Colors.Black);
                else if (hits[0] == 1) A1_F7.Background = new
SolidColorBrush(Colors.Gray);
                // Segundo circulo
                if (hits[1] == 2) A2_F7.Background = new SolidColorBrush(Colors.Black);
                else if (hits[1] == 1) A2_F7.Background = new
SolidColorBrush(Colors.Gray);
                // Tercer circulo
                if (hits[2] == 2) A3_F7.Background = new SolidColorBrush(Colors.Black);
                else if (hits[2] == 1) A3_F7.Background = new
SolidColorBrush(Colors.Gray);
                // Cuarto circulo
                if (hits[3] == 2) A4_F7.Background = new SolidColorBrush(Colors.Black);
                else if (hits[3] == 1) A4_F7.Background = new
SolidColorBrush(Colors.Gray);
```

```
                break;
            case (4):
                // Primer circulo
                if (hits[0] == 2) A1_F6.Background = new SolidColorBrush(Colors.Black);
                else if (hits[0] == 1) A1_F6.Background = new
SolidColorBrush(Colors.Gray);
                // Segundo circulo
                if (hits[1] == 2) A2_F6.Background = new SolidColorBrush(Colors.Black);
                else if (hits[1] == 1) A2_F6.Background = new
SolidColorBrush(Colors.Gray);
                // Tercer circulo
                if (hits[2] == 2) A3_F6.Background = new SolidColorBrush(Colors.Black);
                else if (hits[2] == 1) A3_F6.Background = new
SolidColorBrush(Colors.Gray);
                // Cuarto circulo
                if (hits[3] == 2) A4_F6.Background = new SolidColorBrush(Colors.Black);
                else if (hits[3] == 1) A4_F6.Background = new
SolidColorBrush(Colors.Gray);
                break;
            case (5):
                // Primer circulo
                if (hits[0] == 2) A1_F5.Background = new SolidColorBrush(Colors.Black);
                else if (hits[0] == 1) A1_F5.Background = new
SolidColorBrush(Colors.Gray);
                // Segundo circulo
                if (hits[1] == 2) A2_F5.Background = new SolidColorBrush(Colors.Black);
                else if (hits[1] == 1) A2_F5.Background = new
SolidColorBrush(Colors.Gray);
                // Tercer circulo
                if (hits[2] == 2) A3_F5.Background = new SolidColorBrush(Colors.Black);
                else if (hits[2] == 1) A3_F5.Background = new
SolidColorBrush(Colors.Gray);
                // Cuarto circulo
```

```
                if (hits[3] == 2) A4_F5.Background = new SolidColorBrush(Colors.Black);
                else if (hits[3] == 1) A4_F5.Background = new
SolidColorBrush(Colors.Gray);
                break;
            case (6):
                // Primer circulo
                if (hits[0] == 2) A1_F4.Background = new SolidColorBrush(Colors.Black);
                else if (hits[0] == 1) A1_F4.Background = new
SolidColorBrush(Colors.Gray);
                // Segundo circulo
                if (hits[1] == 2) A2_F4.Background = new SolidColorBrush(Colors.Black);
                else if (hits[1] == 1) A2_F4.Background = new
SolidColorBrush(Colors.Gray);
                // Tercer circulo
                if (hits[2] == 2) A3_F4.Background = new SolidColorBrush(Colors.Black);
                else if (hits[2] == 1) A3_F4.Background = new
SolidColorBrush(Colors.Gray);
                // Cuarto circulo
                if (hits[3] == 2) A4_F4.Background = new SolidColorBrush(Colors.Black);
                else if (hits[3] == 1) A4_F4.Background = new
SolidColorBrush(Colors.Gray);
                break;
            case (7):
                // Primer circulo
                if (hits[0] == 2) A1_F3.Background = new SolidColorBrush(Colors.Black);
                else if (hits[0] == 1) A1_F3.Background = new
SolidColorBrush(Colors.Gray);
                // Segundo circulo
                if (hits[1] == 2) A2_F3.Background = new SolidColorBrush(Colors.Black);
                else if (hits[1] == 1) A2_F3.Background = new
SolidColorBrush(Colors.Gray);
                // Tercer circulo
                if (hits[2] == 2) A3_F3.Background = new SolidColorBrush(Colors.Black);
```

```
                else if (hits[2] == 1) A3_F3.Background = new
SolidColorBrush(Colors.Gray);

                // Cuarto circulo

                if (hits[3] == 2) A4_F3.Background = new SolidColorBrush(Colors.Black);

                else if (hits[3] == 1) A4_F3.Background = new
SolidColorBrush(Colors.Gray);

                break;

            case (8):

                // Primer circulo

                if (hits[0] == 2) A1_F2.Background = new SolidColorBrush(Colors.Black);

                else if (hits[0] == 1) A1_F2.Background = new
SolidColorBrush(Colors.Gray);

                // Segundo circulo

                if (hits[1] == 2) A2_F2.Background = new SolidColorBrush(Colors.Black);

                else if (hits[1] == 1) A2_F2.Background = new
SolidColorBrush(Colors.Gray);

                // Tercer circulo

                if (hits[2] == 2) A3_F2.Background = new SolidColorBrush(Colors.Black);

                else if (hits[2] == 1) A3_F2.Background = new
SolidColorBrush(Colors.Gray);

                // Cuarto circulo

                if (hits[3] == 2) A4_F10.Background = new
SolidColorBrush(Colors.Black);

                else if (hits[3] == 1) A4_F2.Background = new
SolidColorBrush(Colors.Gray);

                break;

            case (9):

                // Primer circulo

                if (hits[0] == 2) A1_F1.Background = new SolidColorBrush(Colors.Black);

                else if (hits[0] == 1) A1_F1.Background = new
SolidColorBrush(Colors.Gray);

                // Segundo circulo

                if (hits[1] == 2) A2_F1.Background = new SolidColorBrush(Colors.Black);
```

```
            else if (hits[1] == 1) A2_F1.Background = new
SolidColorBrush(Colors.Gray);
            // Tercer circulo
            if (hits[2] == 2) A3_F1.Background = new SolidColorBrush(Colors.Black);
            else if (hits[2] == 1) A3_F1.Background = new
SolidColorBrush(Colors.Gray);
            // Cuarto circulo
            if (hits[3] == 2) A4_F1.Background = new SolidColorBrush(Colors.Black);
            else if (hits[3] == 1) A4_F1.Background = new
SolidColorBrush(Colors.Gray);
            break;
        }
    }

    // Inicializa las filas que pueden ser modificadas
    private void InicializarFilas()
    {

        switch (count_fila)
        {
            case 0:
                C0_F0.AllowDrop = false;
                C0_F1.AllowDrop = false;
                C0_F2.AllowDrop = false;
                C0_F3.AllowDrop = false;
                C0_F4.AllowDrop = false;
                C0_F5.AllowDrop = false;
                C0_F6.AllowDrop = false;
                C0_F7.AllowDrop = false;
                C0_F8.AllowDrop = false;
                C0_F9.AllowDrop = true;
```

```
C1_F0.AllowDrop = false;
C1_F1.AllowDrop = false;
C1_F2.AllowDrop = false;
C1_F3.AllowDrop = false;
C1_F4.AllowDrop = false;
C1_F5.AllowDrop = false;
C1_F6.AllowDrop = false;
C1_F7.AllowDrop = false;
C1_F8.AllowDrop = false;
C1_F9.AllowDrop = true;

C2_F0.AllowDrop = false;
C2_F1.AllowDrop = false;
C2_F2.AllowDrop = false;
C2_F3.AllowDrop = false;
C2_F4.AllowDrop = false;
C2_F5.AllowDrop = false;
C2_F6.AllowDrop = false;
C2_F7.AllowDrop = false;
C2_F8.AllowDrop = false;
C2_F9.AllowDrop = true;

C3_F0.AllowDrop = false;
C3_F1.AllowDrop = false;
C3_F2.AllowDrop = false;
C3_F3.AllowDrop = false;
C3_F4.AllowDrop = false;
C3_F5.AllowDrop = false;
C3_F6.AllowDrop = false;
C3_F7.AllowDrop = false;
C3_F8.AllowDrop = false;
```

```
    C3_F9.AllowDrop = true;
    break;
case 1:
    C0_F0.AllowDrop = false;
    C0_F1.AllowDrop = false;
    C0_F2.AllowDrop = false;
    C0_F3.AllowDrop = false;
    C0_F4.AllowDrop = false;
    C0_F5.AllowDrop = false;
    C0_F6.AllowDrop = false;
    C0_F7.AllowDrop = false;
    C0_F8.AllowDrop = true;
    C0_F9.AllowDrop = false;

    C1_F0.AllowDrop = false;
    C1_F1.AllowDrop = false;
    C1_F2.AllowDrop = false;
    C1_F3.AllowDrop = false;
    C1_F4.AllowDrop = false;
    C1_F5.AllowDrop = false;
    C1_F6.AllowDrop = false;
    C1_F7.AllowDrop = false;
    C1_F8.AllowDrop = true;
    C1_F9.AllowDrop = false;

    C2_F0.AllowDrop = false;
    C2_F1.AllowDrop = false;
    C2_F2.AllowDrop = false;
    C2_F3.AllowDrop = false;
    C2_F4.AllowDrop = false;
    C2_F5.AllowDrop = false;
```

```
    C2_F6.AllowDrop = false;
    C2_F7.AllowDrop = false;
    C2_F8.AllowDrop = true;
    C2_F9.AllowDrop = false;

    C3_F0.AllowDrop = false;
    C3_F1.AllowDrop = false;
    C3_F2.AllowDrop = false;
    C3_F3.AllowDrop = false;
    C3_F4.AllowDrop = false;
    C3_F5.AllowDrop = false;
    C3_F6.AllowDrop = false;
    C3_F7.AllowDrop = false;
    C3_F8.AllowDrop = true;
    C3_F9.AllowDrop = false;
    break;
case 2:
    C0_F0.AllowDrop = false;
    C0_F1.AllowDrop = false;
    C0_F2.AllowDrop = false;
    C0_F3.AllowDrop = false;
    C0_F4.AllowDrop = false;
    C0_F5.AllowDrop = false;
    C0_F6.AllowDrop = false;
    C0_F7.AllowDrop = true;
    C0_F8.AllowDrop = false;
    C0_F9.AllowDrop = false;

    C1_F0.AllowDrop = false;
    C1_F1.AllowDrop = false;
    C1_F2.AllowDrop = false;
```

```
    C1_F3.AllowDrop = false;
    C1_F4.AllowDrop = false;
    C1_F5.AllowDrop = false;
    C1_F6.AllowDrop = false;
    C1_F7.AllowDrop = true;
    C1_F8.AllowDrop = false;
    C1_F9.AllowDrop = false;

    C2_F0.AllowDrop = false;
    C2_F1.AllowDrop = false;
    C2_F2.AllowDrop = false;
    C2_F3.AllowDrop = false;
    C2_F4.AllowDrop = false;
    C2_F5.AllowDrop = false;
    C2_F6.AllowDrop = false;
    C2_F7.AllowDrop = true;
    C2_F8.AllowDrop = false;
    C2_F9.AllowDrop = false;

    C3_F0.AllowDrop = false;
    C3_F1.AllowDrop = false;
    C3_F2.AllowDrop = false;
    C3_F3.AllowDrop = false;
    C3_F4.AllowDrop = false;
    C3_F5.AllowDrop = false;
    C3_F6.AllowDrop = false;
    C3_F7.AllowDrop = true;
    C3_F8.AllowDrop = false;
    C3_F9.AllowDrop = false;
    break;
case 3:
```

```
C0_F0.AllowDrop = false;
C0_F1.AllowDrop = false;
C0_F2.AllowDrop = false;
C0_F3.AllowDrop = false;
C0_F4.AllowDrop = false;
C0_F5.AllowDrop = false;
C0_F6.AllowDrop = true;
C0_F7.AllowDrop = false;
C0_F8.AllowDrop = false;
C0_F9.AllowDrop = false;

C1_F0.AllowDrop = false;
C1_F1.AllowDrop = false;
C1_F2.AllowDrop = false;
C1_F3.AllowDrop = false;
C1_F4.AllowDrop = false;
C1_F5.AllowDrop = false;
C1_F6.AllowDrop = true;
C1_F7.AllowDrop = false;
C1_F8.AllowDrop = false;
C1_F9.AllowDrop = false;

C2_F0.AllowDrop = false;
C2_F1.AllowDrop = false;
C2_F2.AllowDrop = false;
C2_F3.AllowDrop = false;
C2_F4.AllowDrop = false;
C2_F5.AllowDrop = false;
C2_F6.AllowDrop = true;
C2_F7.AllowDrop = false;
C2_F8.AllowDrop = false;
```

```
    C2_F9.AllowDrop = false;

    C3_F0.AllowDrop = false;
    C3_F1.AllowDrop = false;
    C3_F2.AllowDrop = false;
    C3_F3.AllowDrop = false;
    C3_F4.AllowDrop = false;
    C3_F5.AllowDrop = false;
    C3_F6.AllowDrop = true;
    C3_F7.AllowDrop = false;
    C3_F8.AllowDrop = false;
    C3_F9.AllowDrop = false;
    break;
case 4:
    C0_F0.AllowDrop = false;
    C0_F1.AllowDrop = false;
    C0_F2.AllowDrop = false;
    C0_F3.AllowDrop = false;
    C0_F4.AllowDrop = false;
    C0_F5.AllowDrop = true;
    C0_F6.AllowDrop = false;
    C0_F7.AllowDrop = false;
    C0_F8.AllowDrop = false;
    C0_F9.AllowDrop = false;

    C1_F0.AllowDrop = false;
    C1_F1.AllowDrop = false;
    C1_F2.AllowDrop = false;
    C1_F3.AllowDrop = false;
    C1_F4.AllowDrop = false;
    C1_F5.AllowDrop = true;
```

```
    C1_F6.AllowDrop = false;
    C1_F7.AllowDrop = false;
    C1_F8.AllowDrop = false;
    C1_F9.AllowDrop = false;

    C2_F0.AllowDrop = false;
    C2_F1.AllowDrop = false;
    C2_F2.AllowDrop = false;
    C2_F3.AllowDrop = false;
    C2_F4.AllowDrop = false;
    C2_F5.AllowDrop = true;
    C2_F6.AllowDrop = false;
    C2_F7.AllowDrop = false;
    C2_F8.AllowDrop = false;
    C2_F9.AllowDrop = false;

    C3_F0.AllowDrop = false;
    C3_F1.AllowDrop = false;
    C3_F2.AllowDrop = false;
    C3_F3.AllowDrop = false;
    C3_F4.AllowDrop = false;
    C3_F5.AllowDrop = true;
    C3_F6.AllowDrop = false;
    C3_F7.AllowDrop = false;
    C3_F8.AllowDrop = false;
    C3_F9.AllowDrop = false;
    break;
case 5:
    C0_F0.AllowDrop = false;
    C0_F1.AllowDrop = false;
    C0_F2.AllowDrop = false;
```

```
C0_F3.AllowDrop = false;
C0_F4.AllowDrop = true;
C0_F5.AllowDrop = false;
C0_F6.AllowDrop = false;
C0_F7.AllowDrop = false;
C0_F8.AllowDrop = false;
C0_F9.AllowDrop = false;

C1_F0.AllowDrop = false;
C1_F1.AllowDrop = false;
C1_F2.AllowDrop = false;
C1_F3.AllowDrop = false;
C1_F4.AllowDrop = true;
C1_F5.AllowDrop = false;
C1_F6.AllowDrop = false;
C1_F7.AllowDrop = false;
C1_F8.AllowDrop = false;
C1_F9.AllowDrop = false;

C2_F0.AllowDrop = false;
C2_F1.AllowDrop = false;
C2_F2.AllowDrop = false;
C2_F3.AllowDrop = false;
C2_F4.AllowDrop = true;
C2_F5.AllowDrop = false;
C2_F6.AllowDrop = false;
C2_F7.AllowDrop = false;
C2_F8.AllowDrop = false;
C2_F9.AllowDrop = false;

C3_F0.AllowDrop = false;
```

```
    C3_F1.AllowDrop = false;
    C3_F2.AllowDrop = false;
    C3_F3.AllowDrop = false;
    C3_F4.AllowDrop = true;
    C3_F5.AllowDrop = false;
    C3_F6.AllowDrop = false;
    C3_F7.AllowDrop = false;
    C3_F8.AllowDrop = false;
    C3_F9.AllowDrop = false;
    break;
case 6:
    C0_F0.AllowDrop = false;
    C0_F1.AllowDrop = false;
    C0_F2.AllowDrop = false;
    C0_F3.AllowDrop = true;
    C0_F4.AllowDrop = false;
    C0_F5.AllowDrop = false;
    C0_F6.AllowDrop = false;
    C0_F7.AllowDrop = false;
    C0_F8.AllowDrop = false;
    C0_F9.AllowDrop = false;

    C1_F0.AllowDrop = false;
    C1_F1.AllowDrop = false;
    C1_F2.AllowDrop = false;
    C1_F3.AllowDrop = true;
    C1_F4.AllowDrop = false;
    C1_F5.AllowDrop = false;
    C1_F6.AllowDrop = false;
    C1_F7.AllowDrop = false;
    C1_F8.AllowDrop = false;
```

```
    C1_F9.AllowDrop = false;

    C2_F0.AllowDrop = false;
    C2_F1.AllowDrop = false;
    C2_F2.AllowDrop = false;
    C2_F3.AllowDrop = true;
    C2_F4.AllowDrop = false;
    C2_F5.AllowDrop = false;
    C2_F6.AllowDrop = false;
    C2_F7.AllowDrop = false;
    C2_F8.AllowDrop = false;
    C2_F9.AllowDrop = false;

    C3_F0.AllowDrop = false;
    C3_F1.AllowDrop = false;
    C3_F2.AllowDrop = false;
    C3_F3.AllowDrop = true;
    C3_F4.AllowDrop = false;
    C3_F5.AllowDrop = false;
    C3_F6.AllowDrop = false;
    C3_F7.AllowDrop = false;
    C3_F8.AllowDrop = false;
    C3_F9.AllowDrop = false;
    break;
case 7:
    C0_F0.AllowDrop = false;
    C0_F1.AllowDrop = false;
    C0_F2.AllowDrop = true;
    C0_F3.AllowDrop = false;
    C0_F4.AllowDrop = false;
    C0_F5.AllowDrop = false;
```

```
C0_F6.AllowDrop = false;
C0_F7.AllowDrop = false;
C0_F8.AllowDrop = false;
C0_F9.AllowDrop = false;

C1_F0.AllowDrop = false;
C1_F1.AllowDrop = false;
C1_F2.AllowDrop = true;
C1_F3.AllowDrop = false;
C1_F4.AllowDrop = false;
C1_F5.AllowDrop = false;
C1_F6.AllowDrop = false;
C1_F7.AllowDrop = false;
C1_F8.AllowDrop = false;
C1_F9.AllowDrop = false;

C2_F0.AllowDrop = false;
C2_F1.AllowDrop = false;
C2_F2.AllowDrop = true;
C2_F3.AllowDrop = false;
C2_F4.AllowDrop = false;
C2_F5.AllowDrop = false;
C2_F6.AllowDrop = false;
C2_F7.AllowDrop = false;
C2_F8.AllowDrop = false;
C2_F9.AllowDrop = false;

C3_F0.AllowDrop = false;
C3_F1.AllowDrop = false;
C3_F2.AllowDrop = true;
C3_F3.AllowDrop = false;
```

```
    C3_F4.AllowDrop = false;
    C3_F5.AllowDrop = false;
    C3_F6.AllowDrop = false;
    C3_F7.AllowDrop = false;
    C3_F8.AllowDrop = false;
    C3_F9.AllowDrop = false;
    break;
case 8:
    C0_F0.AllowDrop = false;
    C0_F1.AllowDrop = true;
    C0_F2.AllowDrop = false;
    C0_F3.AllowDrop = false;
    C0_F4.AllowDrop = false;
    C0_F5.AllowDrop = false;
    C0_F6.AllowDrop = false;
    C0_F7.AllowDrop = false;
    C0_F8.AllowDrop = false;
    C0_F9.AllowDrop = false;

    C1_F0.AllowDrop = false;
    C1_F1.AllowDrop = true;
    C1_F2.AllowDrop = false;
    C1_F3.AllowDrop = false;
    C1_F4.AllowDrop = false;
    C1_F5.AllowDrop = false;
    C1_F6.AllowDrop = false;
    C1_F7.AllowDrop = false;
    C1_F8.AllowDrop = false;
    C1_F9.AllowDrop = false;

    C2_F0.AllowDrop = false;
```

```
    C2_F1.AllowDrop = true;
    C2_F2.AllowDrop = false;
    C2_F3.AllowDrop = false;
    C2_F4.AllowDrop = false;
    C2_F5.AllowDrop = false;
    C2_F6.AllowDrop = false;
    C2_F7.AllowDrop = false;
    C2_F8.AllowDrop = false;
    C2_F9.AllowDrop = false;

    C3_F0.AllowDrop = false;
    C3_F1.AllowDrop = true;
    C3_F2.AllowDrop = false;
    C3_F3.AllowDrop = false;
    C3_F4.AllowDrop = false;
    C3_F5.AllowDrop = false;
    C3_F6.AllowDrop = false;
    C3_F7.AllowDrop = false;
    C3_F8.AllowDrop = false;
    C3_F9.AllowDrop = false;
    break;

case 9:
    C0_F0.AllowDrop = true;
    C0_F1.AllowDrop = false;
    C0_F2.AllowDrop = false;
    C0_F3.AllowDrop = false;
    C0_F4.AllowDrop = false;
    C0_F5.AllowDrop = false;
    C0_F6.AllowDrop = false;
    C0_F7.AllowDrop = false;
```

```
C0_F8.AllowDrop = false;
C0_F9.AllowDrop = false;

C1_F0.AllowDrop = true;
C1_F1.AllowDrop = false;
C1_F2.AllowDrop = false;
C1_F3.AllowDrop = false;
C1_F4.AllowDrop = false;
C1_F5.AllowDrop = false;
C1_F6.AllowDrop = false;
C1_F7.AllowDrop = false;
C1_F8.AllowDrop = false;
C1_F9.AllowDrop = false;

C2_F0.AllowDrop = true;
C2_F1.AllowDrop = false;
C2_F2.AllowDrop = false;
C2_F3.AllowDrop = false;
C2_F4.AllowDrop = false;
C2_F5.AllowDrop = false;
C2_F6.AllowDrop = false;
C2_F7.AllowDrop = false;
C2_F8.AllowDrop = false;
C2_F9.AllowDrop = false;

C3_F0.AllowDrop = true;
C3_F1.AllowDrop = false;
C3_F2.AllowDrop = false;
C3_F3.AllowDrop = false;
C3_F4.AllowDrop = false;
C3_F5.AllowDrop = false;
```

```
            C3_F6.AllowDrop = false;
            C3_F7.AllowDrop = false;
            C3_F8.AllowDrop = false;
            C3_F9.AllowDrop = false;
            break;

    }
}

// Reinicia todas las filas a su valor inicial -> en blanco
private void ReiniciarFilas()
{
    // Primeras columnas de todas las filas
    C0_F0.Background = new SolidColorBrush(Colors.White);
    C0_F1.Background = new SolidColorBrush(Colors.White);
    C0_F2.Background = new SolidColorBrush(Colors.White);
    C0_F3.Background = new SolidColorBrush(Colors.White);
    C0_F4.Background = new SolidColorBrush(Colors.White);
    C0_F5.Background = new SolidColorBrush(Colors.White);
    C0_F6.Background = new SolidColorBrush(Colors.White);
    C0_F7.Background = new SolidColorBrush(Colors.White);
    C0_F8.Background = new SolidColorBrush(Colors.White);
    C0_F9.Background = new SolidColorBrush(Colors.White);
    // Segundas columnas de todas las filas
    C1_F0.Background = new SolidColorBrush(Colors.White);
    C1_F1.Background = new SolidColorBrush(Colors.White);
    C1_F2.Background = new SolidColorBrush(Colors.White);
    C1_F3.Background = new SolidColorBrush(Colors.White);
    C1_F4.Background = new SolidColorBrush(Colors.White);
    C1_F5.Background = new SolidColorBrush(Colors.White);
    C1_F6.Background = new SolidColorBrush(Colors.White);
```

```
C1_F7.Background = new SolidColorBrush(Colors.White);
C1_F8.Background = new SolidColorBrush(Colors.White);
C1_F9.Background = new SolidColorBrush(Colors.White);
// Terceras columnas de todas las filas
C2_F0.Background = new SolidColorBrush(Colors.White);
C2_F1.Background = new SolidColorBrush(Colors.White);
C2_F2.Background = new SolidColorBrush(Colors.White);
C2_F3.Background = new SolidColorBrush(Colors.White);
C2_F4.Background = new SolidColorBrush(Colors.White);
C2_F5.Background = new SolidColorBrush(Colors.White);
C2_F6.Background = new SolidColorBrush(Colors.White);
C2_F7.Background = new SolidColorBrush(Colors.White);
C2_F8.Background = new SolidColorBrush(Colors.White);
C2_F9.Background = new SolidColorBrush(Colors.White);
// Cuartas columnas de todas las filas
C3_F0.Background = new SolidColorBrush(Colors.White);
C3_F1.Background = new SolidColorBrush(Colors.White);
C3_F2.Background = new SolidColorBrush(Colors.White);
C3_F3.Background = new SolidColorBrush(Colors.White);
C3_F4.Background = new SolidColorBrush(Colors.White);
C3_F5.Background = new SolidColorBrush(Colors.White);
C3_F6.Background = new SolidColorBrush(Colors.White);
C3_F7.Background = new SolidColorBrush(Colors.White);
C3_F8.Background = new SolidColorBrush(Colors.White);
C3_F9.Background = new SolidColorBrush(Colors.White);
// Acierto 1 de todas las filas
A1_F1.Background = new SolidColorBrush(Colors.White);
A1_F2.Background = new SolidColorBrush(Colors.White);
A1_F3.Background = new SolidColorBrush(Colors.White);
A1_F4.Background = new SolidColorBrush(Colors.White);
A1_F5.Background = new SolidColorBrush(Colors.White);
```

```
A1_F6.Background = new SolidColorBrush(Colors.White);
A1_F7.Background = new SolidColorBrush(Colors.White);
A1_F8.Background = new SolidColorBrush(Colors.White);
A1_F9.Background = new SolidColorBrush(Colors.White);
A1_F10.Background = new SolidColorBrush(Colors.White);
// Acierto 2 de todas las filas
A2_F1.Background = new SolidColorBrush(Colors.White);
A2_F2.Background = new SolidColorBrush(Colors.White);
A2_F3.Background = new SolidColorBrush(Colors.White);
A2_F4.Background = new SolidColorBrush(Colors.White);
A2_F5.Background = new SolidColorBrush(Colors.White);
A2_F6.Background = new SolidColorBrush(Colors.White);
A2_F7.Background = new SolidColorBrush(Colors.White);
A2_F8.Background = new SolidColorBrush(Colors.White);
A2_F9.Background = new SolidColorBrush(Colors.White);
A2_F10.Background = new SolidColorBrush(Colors.White);
// Acierto 3 de todas las filas
A3_F1.Background = new SolidColorBrush(Colors.White);
A3_F2.Background = new SolidColorBrush(Colors.White);
A3_F3.Background = new SolidColorBrush(Colors.White);
A3_F4.Background = new SolidColorBrush(Colors.White);
A3_F5.Background = new SolidColorBrush(Colors.White);
A3_F6.Background = new SolidColorBrush(Colors.White);
A3_F7.Background = new SolidColorBrush(Colors.White);
A3_F8.Background = new SolidColorBrush(Colors.White);
A3_F9.Background = new SolidColorBrush(Colors.White);
A3_F10.Background = new SolidColorBrush(Colors.White);
// Acierto 4 de todas las filas
A4_F1.Background = new SolidColorBrush(Colors.White);
A4_F2.Background = new SolidColorBrush(Colors.White);
A4_F3.Background = new SolidColorBrush(Colors.White);
```

```
    A4_F4.Background = new SolidColorBrush(Colors.White);
    A4_F5.Background = new SolidColorBrush(Colors.White);
    A4_F6.Background = new SolidColorBrush(Colors.White);
    A4_F7.Background = new SolidColorBrush(Colors.White);
    A4_F8.Background = new SolidColorBrush(Colors.White);
    A4_F9.Background = new SolidColorBrush(Colors.White);
    A4_F10.Background = new SolidColorBrush(Colors.White);
}


// Inicializa el puntaje inicial
private void InicializarPuntaje()
{
    puntaje_actual.Text = score.ToString();
    puntaje_maximo.Text = InitMaxScore();
}

// Trae puntajes máximos existentes
private string InitMaxScore()
{
    string curFile = @"scores.txt";
    // Verifica si existen datos guardados
    if (File.Exists(curFile))
    {
        string[] scores = new string[3];
        int count = 0;
        var linesRead = File.ReadLines("scores.txt");
        foreach (var iter in linesRead)
        {
            scores[count++] = iter;
        }
```

```
        if (game.GetIdLevel() == 1)
        {
            return scores[0];
        }
        else if (game.GetIdLevel() == 2)
        {
            return scores[1];
        }
        else if (game.GetIdLevel() == 3)
        {
            return scores[2];
        }
    }
    else
    {
        using (FileStream fs = File.Create(curFile))
        {
            Byte[] temp_score = new UTF8Encoding(true).GetBytes("0\n");
            fs.Write(temp_score, 0, temp_score.Length);
            fs.Write(temp_score, 0, temp_score.Length);
            fs.Write(temp_score, 0, temp_score.Length);
        }
    }
    return "0";
}

#endregion

#region Notificaciones

// Muestra un mensaje de que el jugador gano
```

```
        private void NotifyWon(int p_score)
        {
            string message = $"¡Felicitaciones!\n\n ¡Has pasado el nivel! Tú puntaje fue de {p_score}.";
            string caption = "¡Has ganado!";
            MessageBoxButton button = MessageBoxButton.OK;
            MessageBoxResult result = MessageBox.Show(message, caption, button);
            switch (result)
            {
                // Vuelve al menú de niveles
                case MessageBoxResult.OK:
                    NavigationService.Navigate(new SelectLevel());
                    break;
            }
        }

        // Muestra un mensaje de que el jugador perdio
        private void NotifyLost()
        {
            string message = "¡Oh No! No has logrado superar el nivel...\n\n¿Deseas intentar de nuevo?";
            string caption = "¡Perdiste!";
            MessageBoxButton button = MessageBoxButton.YesNo;
            MessageBoxResult result = MessageBox.Show(message, caption, button);
            switch (result)
            {
                // Reinicia el nivel
                case MessageBoxResult.Yes:
                    Reboot();
                    break;
```

```
            // Vuelve al menú de niveles
            case MessageBoxResult.No:
                NavigationService.Navigate(new SelectLevel());
                break;
        }
    }

    // Muestra las configuraciones del nivel
    private void NotifyGameSettings()
    {
        string message = "¡Bienvenido!\n\nHas seleccionado el nivel de dificultad medio, tienes 8 colores para crear tus combinaciones pero estos no se pueden repetir.";
        string caption = "Nivel: Medio";
        MessageBoxButton button = MessageBoxButton.OK;
        MessageBoxResult result = MessageBox.Show(message, caption, button);
        switch (result)
        {
            // No ocurre nada
            case MessageBoxResult.OK:
                break;
        }
    }

    // Notifica que un color ha sido repetido
    private void NotifyRepeatedColor()
    {
        string message = "¡Has repetido un color!\n¡Recuerda que en este nivel no esta permitido su repetición!";
        string caption = "Advertencia";
        MessageBoxButton button = MessageBoxButton.OK;
        MessageBoxResult result = MessageBox.Show(message, caption, button);
```

```
            switch (result)
            {
                // No sucede nada
                case MessageBoxResult.OK:
                    break;
            }
        }

        // Notifica si la combinación fue ejecutada vacía
        private void NotifyEmpty()
        {
            string message = "Parece que no has colocado ningún tipo de
combinación...\n\n¡Coloca alguna antes de probar!";
            string caption = "Advertencia";
            MessageBoxButton button = MessageBoxButton.OK;
            MessageBoxResult result = MessageBox.Show(message, caption, button);
        }

        // Notifica que hay un item en blanco en la combinación
        private void NotifyOneEmpty()
        {
            string message = "Parece que has dejado un espacio en blanco...\n\n¡Ten más
cuidado la próxima vez!";
            string caption = "Advertencia";
            MessageBoxButton button = MessageBoxButton.OK;
            MessageBoxResult result = MessageBox.Show(message, caption, button);
        }

        // Notifica que se logro un nuevo record
        private void NotifyMaxScore(int p_score)
        {
```

```
            string message = $"¡Has logrado un nuevo record!\n\nObtuviste un puntaje de: {p_score}";
            string caption = "Felicitaciones";
            MessageBoxButton button = MessageBoxButton.OK;
            MessageBoxResult result = MessageBox.Show(message, caption, button);
        }

        // Notifica las instrucciones del juego
        private void NotifyRules()
        {
            string message = "¡Recuerda!\n\nCada vez que se empieza una partida, el juego creará un nuevo código secreto que lo debes decifrar en un máximo de 10 intentos que le brindaràn de pistas para resolver el código." +

"\n\n**********************************************************************\n\n" +
                "Sobre el juego:\n\nPara poder acertar con mayor facilidad el código secreto ¡podrás guiarte de las pistas que el juego te proveerá:\n" +
                "1. El color negro significa que tienes un color en la posición correcta!\n" +
                "2. El color gris significa que tienes un color correcta pero en la posición incorrecta!";
            string caption = "Reglas del juego";
            MessageBoxButton buttons = MessageBoxButton.OK;
            MessageBoxResult result = MessageBox.Show(message, caption, buttons);
        }

        #endregion
    }
}
```

***Anexo 4: Código fuente del nivel difícil***

```
using System;
using System.Collections.Generic;
```

```
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using MasterMind.Classes;
using System.Threading;
using System.IO;

namespace MasterMind.Pages
{
    public class ColoresDataDificil : ObservableCollection<SolidColorBrush>
    {
        //Donde se almacenan los colores que se van a utilizar
        public ColoresDataDificil() : base()
        {
            Add(new SolidColorBrush(Colors.Red));
            Add(new SolidColorBrush(Colors.Orange));
            Add(new SolidColorBrush(Colors.Purple));
            Add(new SolidColorBrush(Colors.YellowGreen));
            Add(new SolidColorBrush(Colors.Black));
            Add(new SolidColorBrush(Colors.Coral));
            Add(new SolidColorBrush(Colors.Fuchsia));
```

```
        Add(new SolidColorBrush(Colors.Navy));
    }
}
/// <summary>
/// Interaction logic for Page3_d.xaml
/// </summary>
public partial class Page3_d : Page
{
    #region Variables

    // Inicializamos el juego para usarse.
    Mastermind game;
    // Permite verificar en que intento se encuentra el usuario
    int count_fila;
    // Variable del puntaje
    int score = 10000;

    #endregion

    public Page3_d()
    {
        InitializeComponent();
        // Crea el juego
        game = new Mastermind(3);
        // Inicializa los intentos
        count_fila = 0;
        // Inicializa la jugabilidad
        InicializarFilas();
        // Inicializa los puntajes
        InicializarPuntaje();
        // Notifica datos del modo de juego
```

```
            var thread = new Thread(() => { NotifyGameSettings(); });
            thread.Start();
        }

        #region GUI

        //evento de drag and drop
        private void Ellipse_MouseMove(object sender, MouseEventArgs e)
        {
            if (sender is Ellipse ellipse && e.LeftButton == MouseButtonState.Pressed)
            {
                DragDrop.DoDragDrop(ellipse, ellipse.Fill.ToString(),
DragDropEffects.Copy);
            }
        }

        // Pinta un circulo con el color deseado
        private void Border_Drop(object sender, DragEventArgs e)
        {
            if (sender is Border border)
            {
                if (e.Data.GetDataPresent(DataFormats.StringFormat))
                {
                    string stringFromDrop =
(string)e.Data.GetData(DataFormats.StringFormat);
                    BrushConverter convertir = new BrushConverter();
                    if (convertir.IsValid(stringFromDrop))
                    {
                        Brush pincel = (Brush)convertir.ConvertFromString(stringFromDrop);
                        border.Background = pincel;
                    }
```

```
        }
    }
}

// Vuelta al menú de selección de niveles
private void Button_Click(object sender, RoutedEventArgs e)
{
    NavigationService.Navigate(new SelectLevel());
}

// Reinicia el nivel
private void reboot_btn_Click(object sender, RoutedEventArgs e)
{
    Reboot();
}

// Permite visualizar las instrucciones del juego
private void instrucciones_btn_Click(object sender, RoutedEventArgs e)
{
    NotifyRules();
}

// Confirmación de combinación
private void confirmar_btn_Click(object sender, RoutedEventArgs e)
{
    // Tomamos la combinación que se esta probando al momento.
    string[] combination = GetCombination();
    // Transformamos los colores a hexadecimal
    string[] combination_colors = GetCombinationHexColors(combination);
    // Verificamos si la combinación no tiene datos
    if (!CheckEmpty(combination_colors))
```

```
{
    if (!CheckOneEmpty(combination_colors))
    {
        // Probamos la combinación
        var hits = game.TryCombination(combination_colors);
        // Configuramos los aciertos dados y los mostramos al usuario
        SetHits(hits);
        // Incrementamos el contador del intento
        count_fila++;
        // Verificar si gano el juego el jugador
        if (PlayerWon(hits))
        {
            if (CheckNewMaxScore())
            {
                NotifyMaxScore(Int32.Parse(puntaje_actual.Text));
                SetMaxScore();
            }
            NotifyWon(Int32.Parse(puntaje_actual.Text));
        }

        // Actualiza el score del jugador
        score = score - 1000;
        puntaje_actual.Text = score.ToString();

        // Verifica si perdio el juego el jugador
        if (PlayerLost() & !PlayerWon(hits))
        {
            NotifyLost();
        }
        // Actualiza las filas que pueden ser modificadas
        if (count_fila < 10) InicializarFilas();
```

```
        }
        else
        {
            NotifyOneEmpty();
        }
    }
    else
    {
        NotifyEmpty();
    }
}

#endregion

#region Métodos críticos

// Verifica si el jugador gano
private bool PlayerWon(int[] hits)
{
    foreach (var iter in hits) if (iter != 2) return false;
    return true;
}

// Verifica si el jugador perdio
private bool PlayerLost()
{
    if (count_fila > 9) return true;
    return false;
}

// Reinicia el nivel
```

```
        private void Reboot()
        {
            // Establece un nuevo código secreto
            if (count_fila > 0) game.InitSecret();
            // Reestablece los intentos
            count_fila = 0;
            // Reestablece el puntaje actual
            score = 10000;
            // Inicializa la disponibilidad de las filas
            InicializarFilas();
            // Reinicia todos los datos del juego actual
            ReiniciarFilas();
        }

        // Hace que una posición  se inicialice
        private void ChangePosition(int position)
        {
            switch (count_fila)
            {
                case (0):
                    if (position == 0) C0_F9.Background = new
SolidColorBrush(Colors.White);
                    else if (position == 1) C1_F9.Background = new
SolidColorBrush(Colors.White);
                    else if (position == 2) C2_F9.Background = new
SolidColorBrush(Colors.White);
                    else if (position == 3) C3_F9.Background = new
SolidColorBrush(Colors.White);
                    break;
                case (1):
                    if (position == 0) C0_F8.Background = new
SolidColorBrush(Colors.White);
```

```
                else if (position == 1) C1_F8.Background = new
SolidColorBrush(Colors.White);

                else if (position == 2) C2_F8.Background = new
SolidColorBrush(Colors.White);

                else if (position == 3) C3_F8.Background = new
SolidColorBrush(Colors.White);

                break;

            case (2):

                if (position == 0) C0_F7.Background = new
SolidColorBrush(Colors.White);

                else if (position == 1) C1_F7.Background = new
SolidColorBrush(Colors.White);

                else if (position == 2) C2_F7.Background = new
SolidColorBrush(Colors.White);

                else if (position == 3) C3_F7.Background = new
SolidColorBrush(Colors.White);

                break;

            case (3):

                if (position == 0) C0_F6.Background = new
SolidColorBrush(Colors.White);

                else if (position == 1) C1_F6.Background = new
SolidColorBrush(Colors.White);

                else if (position == 2) C2_F6.Background = new
SolidColorBrush(Colors.White);

                else if (position == 3) C3_F6.Background = new
SolidColorBrush(Colors.White);

                break;

            case (4):

                if (position == 0) C0_F5.Background = new
SolidColorBrush(Colors.White);

                else if (position == 1) C1_F5.Background = new
SolidColorBrush(Colors.White);

                else if (position == 2) C2_F5.Background = new
SolidColorBrush(Colors.White);

                else if (position == 3) C3_F5.Background = new
SolidColorBrush(Colors.White);
```

```
                break;
            case (5):
                if (position == 0) C0_F4.Background = new
SolidColorBrush(Colors.White);
                else if (position == 1) C1_F4.Background = new
SolidColorBrush(Colors.White);
                else if (position == 2) C2_F4.Background = new
SolidColorBrush(Colors.White);
                else if (position == 3) C3_F4.Background = new
SolidColorBrush(Colors.White);
                break;
            case (6):
                if (position == 0) C0_F3.Background = new
SolidColorBrush(Colors.White);
                else if (position == 1) C1_F3.Background = new
SolidColorBrush(Colors.White);
                else if (position == 2) C2_F3.Background = new
SolidColorBrush(Colors.White);
                else if (position == 3) C3_F3.Background = new
SolidColorBrush(Colors.White);
                break;
            case (7):
                if (position == 0) C0_F2.Background = new
SolidColorBrush(Colors.White);
                else if (position == 1) C1_F2.Background = new
SolidColorBrush(Colors.White);
                else if (position == 2) C2_F2.Background = new
SolidColorBrush(Colors.White);
                else if (position == 3) C3_F2.Background = new
SolidColorBrush(Colors.White);
                break;
            case (8):
                if (position == 0) C0_F1.Background = new
SolidColorBrush(Colors.White);
                else if (position == 1) C1_F1.Background = new
SolidColorBrush(Colors.White);
```

```
                else if (position == 2) C2_F1.Background = new
SolidColorBrush(Colors.White);
                else if (position == 3) C3_F1.Background = new
SolidColorBrush(Colors.White);
                break;
            case (9):
                if (position == 0) C0_F0.Background = new
SolidColorBrush(Colors.White);
                else if (position == 1) C1_F0.Background = new
SolidColorBrush(Colors.White);
                else if (position == 2) C2_F0.Background = new
SolidColorBrush(Colors.White);
                else if (position == 3) C3_F0.Background = new
SolidColorBrush(Colors.White);
                break;
        }
    }

    // Verifica si la combinación esta vacía
    private bool CheckEmpty(string[] colors)
    {
        foreach (var iter in colors) if (iter != "#FFFFFF") return false;
        return true;
    }

    // Verifica que no existan items en blanco para las combinaciones de los intentos
    private bool CheckOneEmpty(string[] colors)
    {
        foreach (var iter in colors) if (iter == "#FFFFFF") return true;
        return false;
    }

    // Establece el puntaje máximo
```

```
private void SetMaxScore()
{
    // Identifica que nivel se esta jugando
    var id = game.GetIdLevel();
    // Variables para modificar el archivo de puntajes
    string filename = "scores.txt";
    string[] scores = new string[3];
    int count = 0;

    // Lee los datos y guarda en scores
    var linesRead = File.ReadLines("scores.txt");
    foreach (var iter in linesRead)
    {
        scores[count++] = iter;
    }

    // Borra el archivo
    if (File.Exists(filename)) File.Delete(filename);

    // Establece la nueva información
    if (id == 1)
    {
        scores[0] = score.ToString();
    }
    else if (id == 2)
    {
        scores[1] = score.ToString();
    }
    else if (id == 3)
    {
        scores[2] = score.ToString();
```

```
    }

    // Guarda la nueva información
    using (FileStream fs = File.Create(filename))
    {
        Byte[] score_1 = new UTF8Encoding(true).GetBytes(scores[0] + "\n");
        Byte[] score_2 = new UTF8Encoding(true).GetBytes(scores[1] + "\n");
        Byte[] score_3 = new UTF8Encoding(true).GetBytes(scores[2] + "\n");
        fs.Write(score_1, 0, score_1.Length);
        fs.Write(score_2, 0, score_2.Length);
        fs.Write(score_3, 0, score_3.Length);
    }
}

// Evalua si se logro un nuevo marcador
private bool CheckNewMaxScore()
{
    if (score > Int32.Parse(puntaje_maximo.Text)) return true;
    return false;
}

#endregion

#region Getters

// Devuelve la combinación que se esta probando
private string[] GetCombination()
{
    string[] temp = new string[4];
    // El número del case + 1 indica la fila que se esta probando.
    switch (count_fila)
```

```
{
    case (0):
        temp[0] = C0_F9.Background.ToString();
        temp[1] = C1_F9.Background.ToString();
        temp[2] = C2_F9.Background.ToString();
        temp[3] = C3_F9.Background.ToString();
        break;
    case (1):
        temp[0] = C0_F8.Background.ToString();
        temp[1] = C1_F8.Background.ToString();
        temp[2] = C2_F8.Background.ToString();
        temp[3] = C3_F8.Background.ToString();
        break;
    case (2):
        temp[0] = C0_F7.Background.ToString();
        temp[1] = C1_F7.Background.ToString();
        temp[2] = C2_F7.Background.ToString();
        temp[3] = C3_F7.Background.ToString();
        break;
    case (3):
        temp[0] = C0_F6.Background.ToString();
        temp[1] = C1_F6.Background.ToString();
        temp[2] = C2_F6.Background.ToString();
        temp[3] = C3_F6.Background.ToString();
        break;
    case (4):
        temp[0] = C0_F5.Background.ToString();
        temp[1] = C1_F5.Background.ToString();
        temp[2] = C2_F5.Background.ToString();
        temp[3] = C3_F5.Background.ToString();
        break;
```

```
    case (5):
        temp[0] = C0_F4.Background.ToString();
        temp[1] = C1_F4.Background.ToString();
        temp[2] = C2_F4.Background.ToString();
        temp[3] = C3_F4.Background.ToString();
        break;
    case (6):
        temp[0] = C0_F3.Background.ToString();
        temp[1] = C1_F3.Background.ToString();
        temp[2] = C2_F3.Background.ToString();
        temp[3] = C3_F3.Background.ToString();
        break;
    case (7):
        temp[0] = C0_F2.Background.ToString();
        temp[1] = C1_F2.Background.ToString();
        temp[2] = C2_F2.Background.ToString();
        temp[3] = C3_F2.Background.ToString();
        break;
    case (8):
        temp[0] = C0_F1.Background.ToString();
        temp[1] = C1_F1.Background.ToString();
        temp[2] = C2_F1.Background.ToString();
        temp[3] = C3_F1.Background.ToString();
        break;
    case (9):
        temp[0] = C0_F0.Background.ToString();
        temp[1] = C1_F0.Background.ToString();
        temp[2] = C2_F0.Background.ToString();
        temp[3] = C3_F0.Background.ToString();
        break;
}
```

```
            return temp;
        }

        // Transforma los colores Brush a su equivalente en Hexadecimal
        private string[] GetCombinationHexColors(string[] temp)
        {
            for (int i = 0; i < temp.Length; i++)
            {
                if (temp[i] == "#FFFFFFFF") temp[i] = "#FFFFFF"; // Conversión del blanco
                else if (temp[i] == "#FFFF0000") temp[i] = "#FF0000"; // Conversión del color rojo
                else if (temp[i] == "#FFFFA500") temp[i] = "#FFA500"; // Conversión del color amarillo
                else if (temp[i] == "#FF800080") temp[i] = "#800080"; // Conversión del color morado
                else if (temp[i] == "#FF9ACD32") temp[i] = "#9ACD32"; // Conversión del color verde
                else if (temp[i] == "#00000000") temp[i] = "#000000"; // Conversión del color negro
                else if (temp[i] == "#FFFF7F50") temp[i] = "#FF7F50"; // Conversión del color anaranjado
                else if (temp[i] == "#FFFF00FF") temp[i] = "#FF00FF"; // Conversión del color rosado
                else if (temp[i] == "#FF000080") temp[i] = "#000080"; // Conversión del color azul
            }
            return temp;
        }

        #endregion

        #region Inicializadores

        // Permite la visualización de los aciertos dados al usuario
```

```
        private void SetHits(int[] hits)
        {
            // Coloca color negro si hay un item en la posición correcta.
            // Coloca color gris si hay un item en la posición incorrecta.
            // Deja en blanco si no existe items acertados.
            switch (count_fila)
            {
                case (0):
                    // Primer circulo
                    if (hits[0] == 2) A1_F10.Background = new
SolidColorBrush(Colors.Black);
                    else if (hits[0] == 1) A1_F10.Background = new
SolidColorBrush(Colors.Gray);
                    // Segundo circulo
                    if (hits[1] == 2) A2_F10.Background = new
SolidColorBrush(Colors.Black);
                    else if (hits[1] == 1) A2_F10.Background = new
SolidColorBrush(Colors.Gray);
                    // Tercer circulo
                    if (hits[2] == 2) A3_F10.Background = new
SolidColorBrush(Colors.Black);
                    else if (hits[3] == 1) A3_F10.Background = new
SolidColorBrush(Colors.Gray);
                    // Cuarto circulo
                    if (hits[3] == 2) A4_F10.Background = new
SolidColorBrush(Colors.Black);
                    else if (hits[3] == 1) A4_F10.Background = new
SolidColorBrush(Colors.Gray);
                    break;
                case (1):
                    // Primer circulo
                    if (hits[0] == 2) A1_F9.Background = new SolidColorBrush(Colors.Black);
                    else if (hits[0] == 1) A1_F9.Background = new
SolidColorBrush(Colors.Gray);
```

```
                // Segundo circulo
                if (hits[1] == 2) A2_F9.Background = new SolidColorBrush(Colors.Black);
                else if (hits[1] == 1) A2_F9.Background = new
SolidColorBrush(Colors.Gray);
                // Tercer circulo
                if (hits[2] == 2) A3_F9.Background = new SolidColorBrush(Colors.Black);
                else if (hits[2] == 1) A3_F9.Background = new
SolidColorBrush(Colors.Gray);
                // Cuarto circulo
                if (hits[3] == 2) A4_F9.Background = new SolidColorBrush(Colors.Black);
                else if (hits[3] == 1) A4_F9.Background = new
SolidColorBrush(Colors.Gray);
                break;
            case (2):
                // Primer circulo
                if (hits[0] == 2) A1_F8.Background = new SolidColorBrush(Colors.Black);
                else if (hits[0] == 1) A1_F8.Background = new
SolidColorBrush(Colors.Gray);
                // Segundo circulo
                if (hits[1] == 2) A2_F8.Background = new SolidColorBrush(Colors.Black);
                else if (hits[1] == 1) A2_F8.Background = new
SolidColorBrush(Colors.Gray);
                // Tercer circulo
                if (hits[2] == 2) A3_F8.Background = new SolidColorBrush(Colors.Black);
                else if (hits[2] == 1) A3_F8.Background = new
SolidColorBrush(Colors.Gray);
                // Cuarto circulo
                if (hits[3] == 2) A4_F8.Background = new SolidColorBrush(Colors.Black);
                else if (hits[3] == 1) A4_F8.Background = new
SolidColorBrush(Colors.Gray);
                break;
            case (3):
                // Primer circulo
```

```
            if (hits[0] == 2) A1_F7.Background = new SolidColorBrush(Colors.Black);
            else if (hits[0] == 1) A1_F7.Background = new
SolidColorBrush(Colors.Gray);
            // Segundo circulo
            if (hits[1] == 2) A2_F7.Background = new SolidColorBrush(Colors.Black);
            else if (hits[1] == 1) A2_F7.Background = new
SolidColorBrush(Colors.Gray);
            // Tercer circulo
            if (hits[2] == 2) A3_F7.Background = new SolidColorBrush(Colors.Black);
            else if (hits[2] == 1) A3_F7.Background = new
SolidColorBrush(Colors.Gray);
            // Cuarto circulo
            if (hits[3] == 2) A4_F7.Background = new SolidColorBrush(Colors.Black);
            else if (hits[3] == 1) A4_F7.Background = new
SolidColorBrush(Colors.Gray);
            break;
        case (4):
            // Primer circulo
            if (hits[0] == 2) A1_F6.Background = new SolidColorBrush(Colors.Black);
            else if (hits[0] == 1) A1_F6.Background = new
SolidColorBrush(Colors.Gray);
            // Segundo circulo
            if (hits[1] == 2) A2_F6.Background = new SolidColorBrush(Colors.Black);
            else if (hits[1] == 1) A2_F6.Background = new
SolidColorBrush(Colors.Gray);
            // Tercer circulo
            if (hits[2] == 2) A3_F6.Background = new SolidColorBrush(Colors.Black);
            else if (hits[2] == 1) A3_F6.Background = new
SolidColorBrush(Colors.Gray);
            // Cuarto circulo
            if (hits[3] == 2) A4_F6.Background = new SolidColorBrush(Colors.Black);
            else if (hits[3] == 1) A4_F6.Background = new
SolidColorBrush(Colors.Gray);
            break;
```

```
            case (5):
                // Primer circulo
                if (hits[0] == 2) A1_F5.Background = new SolidColorBrush(Colors.Black);
                else if (hits[0] == 1) A1_F5.Background = new
SolidColorBrush(Colors.Gray);
                // Segundo circulo
                if (hits[1] == 2) A2_F5.Background = new SolidColorBrush(Colors.Black);
                else if (hits[1] == 1) A2_F5.Background = new
SolidColorBrush(Colors.Gray);
                // Tercer circulo
                if (hits[2] == 2) A3_F5.Background = new SolidColorBrush(Colors.Black);
                else if (hits[2] == 1) A3_F5.Background = new
SolidColorBrush(Colors.Gray);
                // Cuarto circulo
                if (hits[3] == 2) A4_F5.Background = new SolidColorBrush(Colors.Black);
                else if (hits[3] == 1) A4_F5.Background = new
SolidColorBrush(Colors.Gray);
                break;
            case (6):
                // Primer circulo
                if (hits[0] == 2) A1_F4.Background = new SolidColorBrush(Colors.Black);
                else if (hits[0] == 1) A1_F4.Background = new
SolidColorBrush(Colors.Gray);
                // Segundo circulo
                if (hits[1] == 2) A2_F4.Background = new SolidColorBrush(Colors.Black);
                else if (hits[1] == 1) A2_F4.Background = new
SolidColorBrush(Colors.Gray);
                // Tercer circulo
                if (hits[2] == 2) A3_F4.Background = new SolidColorBrush(Colors.Black);
                else if (hits[2] == 1) A3_F4.Background = new
SolidColorBrush(Colors.Gray);
                // Cuarto circulo
                if (hits[3] == 2) A4_F4.Background = new SolidColorBrush(Colors.Black);
```

```
                else if (hits[3] == 1) A4_F4.Background = new
SolidColorBrush(Colors.Gray);

                break;

            case (7):

                // Primer circulo

                if (hits[0] == 2) A1_F3.Background = new SolidColorBrush(Colors.Black);

                else if (hits[0] == 1) A1_F3.Background = new
SolidColorBrush(Colors.Gray);

                // Segundo circulo

                if (hits[1] == 2) A2_F3.Background = new SolidColorBrush(Colors.Black);

                else if (hits[1] == 1) A2_F3.Background = new
SolidColorBrush(Colors.Gray);

                // Tercer circulo

                if (hits[2] == 2) A3_F3.Background = new SolidColorBrush(Colors.Black);

                else if (hits[2] == 1) A3_F3.Background = new
SolidColorBrush(Colors.Gray);

                // Cuarto circulo

                if (hits[3] == 2) A4_F3.Background = new SolidColorBrush(Colors.Black);

                else if (hits[3] == 1) A4_F3.Background = new
SolidColorBrush(Colors.Gray);

                break;

            case (8):

                // Primer circulo

                if (hits[0] == 2) A1_F2.Background = new SolidColorBrush(Colors.Black);

                else if (hits[0] == 1) A1_F2.Background = new
SolidColorBrush(Colors.Gray);

                // Segundo circulo

                if (hits[1] == 2) A2_F2.Background = new SolidColorBrush(Colors.Black);

                else if (hits[1] == 1) A2_F2.Background = new
SolidColorBrush(Colors.Gray);

                // Tercer circulo

                if (hits[2] == 2) A3_F2.Background = new SolidColorBrush(Colors.Black);

                else if (hits[2] == 1) A3_F2.Background = new
SolidColorBrush(Colors.Gray);
```

```
                // Cuarto circulo
                if (hits[3] == 2) A4_F10.Background = new
SolidColorBrush(Colors.Black);
                else if (hits[3] == 1) A4_F2.Background = new
SolidColorBrush(Colors.Gray);
                break;
            case (9):
                // Primer circulo
                if (hits[0] == 2) A1_F1.Background = new SolidColorBrush(Colors.Black);
                else if (hits[0] == 1) A1_F1.Background = new
SolidColorBrush(Colors.Gray);
                // Segundo circulo
                if (hits[1] == 2) A2_F1.Background = new SolidColorBrush(Colors.Black);
                else if (hits[1] == 1) A2_F1.Background = new
SolidColorBrush(Colors.Gray);
                // Tercer circulo
                if (hits[2] == 2) A3_F1.Background = new SolidColorBrush(Colors.Black);
                else if (hits[2] == 1) A3_F1.Background = new
SolidColorBrush(Colors.Gray);
                // Cuarto circulo
                if (hits[3] == 2) A4_F1.Background = new SolidColorBrush(Colors.Black);
                else if (hits[3] == 1) A4_F1.Background = new
SolidColorBrush(Colors.Gray);
                break;
        }
    }

    // Inicializa las filas que pueden ser modificadas
    private void InicializarFilas()
    {

        switch (count_fila)
        {
```

```
case 0:
    C0_F0.AllowDrop = false;
    C0_F1.AllowDrop = false;
    C0_F2.AllowDrop = false;
    C0_F3.AllowDrop = false;
    C0_F4.AllowDrop = false;
    C0_F5.AllowDrop = false;
    C0_F6.AllowDrop = false;
    C0_F7.AllowDrop = false;
    C0_F8.AllowDrop = false;
    C0_F9.AllowDrop = true;

    C1_F0.AllowDrop = false;
    C1_F1.AllowDrop = false;
    C1_F2.AllowDrop = false;
    C1_F3.AllowDrop = false;
    C1_F4.AllowDrop = false;
    C1_F5.AllowDrop = false;
    C1_F6.AllowDrop = false;
    C1_F7.AllowDrop = false;
    C1_F8.AllowDrop = false;
    C1_F9.AllowDrop = true;

    C2_F0.AllowDrop = false;
    C2_F1.AllowDrop = false;
    C2_F2.AllowDrop = false;
    C2_F3.AllowDrop = false;
    C2_F4.AllowDrop = false;
    C2_F5.AllowDrop = false;
    C2_F6.AllowDrop = false;
    C2_F7.AllowDrop = false;
```

```
    C2_F8.AllowDrop = false;
    C2_F9.AllowDrop = true;

    C3_F0.AllowDrop = false;
    C3_F1.AllowDrop = false;
    C3_F2.AllowDrop = false;
    C3_F3.AllowDrop = false;
    C3_F4.AllowDrop = false;
    C3_F5.AllowDrop = false;
    C3_F6.AllowDrop = false;
    C3_F7.AllowDrop = false;
    C3_F8.AllowDrop = false;
    C3_F9.AllowDrop = true;
    break;
case 1:
    C0_F0.AllowDrop = false;
    C0_F1.AllowDrop = false;
    C0_F2.AllowDrop = false;
    C0_F3.AllowDrop = false;
    C0_F4.AllowDrop = false;
    C0_F5.AllowDrop = false;
    C0_F6.AllowDrop = false;
    C0_F7.AllowDrop = false;
    C0_F8.AllowDrop = true;
    C0_F9.AllowDrop = false;

    C1_F0.AllowDrop = false;
    C1_F1.AllowDrop = false;
    C1_F2.AllowDrop = false;
    C1_F3.AllowDrop = false;
    C1_F4.AllowDrop = false;
```

```
    C1_F5.AllowDrop = false;
    C1_F6.AllowDrop = false;
    C1_F7.AllowDrop = false;
    C1_F8.AllowDrop = true;
    C1_F9.AllowDrop = false;

    C2_F0.AllowDrop = false;
    C2_F1.AllowDrop = false;
    C2_F2.AllowDrop = false;
    C2_F3.AllowDrop = false;
    C2_F4.AllowDrop = false;
    C2_F5.AllowDrop = false;
    C2_F6.AllowDrop = false;
    C2_F7.AllowDrop = false;
    C2_F8.AllowDrop = true;
    C2_F9.AllowDrop = false;

    C3_F0.AllowDrop = false;
    C3_F1.AllowDrop = false;
    C3_F2.AllowDrop = false;
    C3_F3.AllowDrop = false;
    C3_F4.AllowDrop = false;
    C3_F5.AllowDrop = false;
    C3_F6.AllowDrop = false;
    C3_F7.AllowDrop = false;
    C3_F8.AllowDrop = true;
    C3_F9.AllowDrop = false;
    break;
case 2:
    C0_F0.AllowDrop = false;
    C0_F1.AllowDrop = false;
```

```
C0_F2.AllowDrop = false;
C0_F3.AllowDrop = false;
C0_F4.AllowDrop = false;
C0_F5.AllowDrop = false;
C0_F6.AllowDrop = false;
C0_F7.AllowDrop = true;
C0_F8.AllowDrop = false;
C0_F9.AllowDrop = false;

C1_F0.AllowDrop = false;
C1_F1.AllowDrop = false;
C1_F2.AllowDrop = false;
C1_F3.AllowDrop = false;
C1_F4.AllowDrop = false;
C1_F5.AllowDrop = false;
C1_F6.AllowDrop = false;
C1_F7.AllowDrop = true;
C1_F8.AllowDrop = false;
C1_F9.AllowDrop = false;

C2_F0.AllowDrop = false;
C2_F1.AllowDrop = false;
C2_F2.AllowDrop = false;
C2_F3.AllowDrop = false;
C2_F4.AllowDrop = false;
C2_F5.AllowDrop = false;
C2_F6.AllowDrop = false;
C2_F7.AllowDrop = true;
C2_F8.AllowDrop = false;
C2_F9.AllowDrop = false;
```

```
    C3_F0.AllowDrop = false;
    C3_F1.AllowDrop = false;
    C3_F2.AllowDrop = false;
    C3_F3.AllowDrop = false;
    C3_F4.AllowDrop = false;
    C3_F5.AllowDrop = false;
    C3_F6.AllowDrop = false;
    C3_F7.AllowDrop = true;
    C3_F8.AllowDrop = false;
    C3_F9.AllowDrop = false;
    break;
case 3:
    C0_F0.AllowDrop = false;
    C0_F1.AllowDrop = false;
    C0_F2.AllowDrop = false;
    C0_F3.AllowDrop = false;
    C0_F4.AllowDrop = false;
    C0_F5.AllowDrop = false;
    C0_F6.AllowDrop = true;
    C0_F7.AllowDrop = false;
    C0_F8.AllowDrop = false;
    C0_F9.AllowDrop = false;

    C1_F0.AllowDrop = false;
    C1_F1.AllowDrop = false;
    C1_F2.AllowDrop = false;
    C1_F3.AllowDrop = false;
    C1_F4.AllowDrop = false;
    C1_F5.AllowDrop = false;
    C1_F6.AllowDrop = true;
    C1_F7.AllowDrop = false;
```

```
    C1_F8.AllowDrop = false;
    C1_F9.AllowDrop = false;

    C2_F0.AllowDrop = false;
    C2_F1.AllowDrop = false;
    C2_F2.AllowDrop = false;
    C2_F3.AllowDrop = false;
    C2_F4.AllowDrop = false;
    C2_F5.AllowDrop = false;
    C2_F6.AllowDrop = true;
    C2_F7.AllowDrop = false;
    C2_F8.AllowDrop = false;
    C2_F9.AllowDrop = false;

    C3_F0.AllowDrop = false;
    C3_F1.AllowDrop = false;
    C3_F2.AllowDrop = false;
    C3_F3.AllowDrop = false;
    C3_F4.AllowDrop = false;
    C3_F5.AllowDrop = false;
    C3_F6.AllowDrop = true;
    C3_F7.AllowDrop = false;
    C3_F8.AllowDrop = false;
    C3_F9.AllowDrop = false;
    break;
case 4:
    C0_F0.AllowDrop = false;
    C0_F1.AllowDrop = false;
    C0_F2.AllowDrop = false;
    C0_F3.AllowDrop = false;
    C0_F4.AllowDrop = false;
```

```
C0_F5.AllowDrop = true;
C0_F6.AllowDrop = false;
C0_F7.AllowDrop = false;
C0_F8.AllowDrop = false;
C0_F9.AllowDrop = false;

C1_F0.AllowDrop = false;
C1_F1.AllowDrop = false;
C1_F2.AllowDrop = false;
C1_F3.AllowDrop = false;
C1_F4.AllowDrop = false;
C1_F5.AllowDrop = true;
C1_F6.AllowDrop = false;
C1_F7.AllowDrop = false;
C1_F8.AllowDrop = false;
C1_F9.AllowDrop = false;

C2_F0.AllowDrop = false;
C2_F1.AllowDrop = false;
C2_F2.AllowDrop = false;
C2_F3.AllowDrop = false;
C2_F4.AllowDrop = false;
C2_F5.AllowDrop = true;
C2_F6.AllowDrop = false;
C2_F7.AllowDrop = false;
C2_F8.AllowDrop = false;
C2_F9.AllowDrop = false;

C3_F0.AllowDrop = false;
C3_F1.AllowDrop = false;
C3_F2.AllowDrop = false;
```

```
    C3_F3.AllowDrop = false;
    C3_F4.AllowDrop = false;
    C3_F5.AllowDrop = true;
    C3_F6.AllowDrop = false;
    C3_F7.AllowDrop = false;
    C3_F8.AllowDrop = false;
    C3_F9.AllowDrop = false;
    break;
case 5:
    C0_F0.AllowDrop = false;
    C0_F1.AllowDrop = false;
    C0_F2.AllowDrop = false;
    C0_F3.AllowDrop = false;
    C0_F4.AllowDrop = true;
    C0_F5.AllowDrop = false;
    C0_F6.AllowDrop = false;
    C0_F7.AllowDrop = false;
    C0_F8.AllowDrop = false;
    C0_F9.AllowDrop = false;

    C1_F0.AllowDrop = false;
    C1_F1.AllowDrop = false;
    C1_F2.AllowDrop = false;
    C1_F3.AllowDrop = false;
    C1_F4.AllowDrop = true;
    C1_F5.AllowDrop = false;
    C1_F6.AllowDrop = false;
    C1_F7.AllowDrop = false;
    C1_F8.AllowDrop = false;
    C1_F9.AllowDrop = false;
```

```
    C2_F0.AllowDrop = false;
    C2_F1.AllowDrop = false;
    C2_F2.AllowDrop = false;
    C2_F3.AllowDrop = false;
    C2_F4.AllowDrop = true;
    C2_F5.AllowDrop = false;
    C2_F6.AllowDrop = false;
    C2_F7.AllowDrop = false;
    C2_F8.AllowDrop = false;
    C2_F9.AllowDrop = false;

    C3_F0.AllowDrop = false;
    C3_F1.AllowDrop = false;
    C3_F2.AllowDrop = false;
    C3_F3.AllowDrop = false;
    C3_F4.AllowDrop = true;
    C3_F5.AllowDrop = false;
    C3_F6.AllowDrop = false;
    C3_F7.AllowDrop = false;
    C3_F8.AllowDrop = false;
    C3_F9.AllowDrop = false;
    break;
case 6:
    C0_F0.AllowDrop = false;
    C0_F1.AllowDrop = false;
    C0_F2.AllowDrop = false;
    C0_F3.AllowDrop = true;
    C0_F4.AllowDrop = false;
    C0_F5.AllowDrop = false;
    C0_F6.AllowDrop = false;
    C0_F7.AllowDrop = false;
```

```
C0_F8.AllowDrop = false;
C0_F9.AllowDrop = false;

C1_F0.AllowDrop = false;
C1_F1.AllowDrop = false;
C1_F2.AllowDrop = false;
C1_F3.AllowDrop = true;
C1_F4.AllowDrop = false;
C1_F5.AllowDrop = false;
C1_F6.AllowDrop = false;
C1_F7.AllowDrop = false;
C1_F8.AllowDrop = false;
C1_F9.AllowDrop = false;

C2_F0.AllowDrop = false;
C2_F1.AllowDrop = false;
C2_F2.AllowDrop = false;
C2_F3.AllowDrop = true;
C2_F4.AllowDrop = false;
C2_F5.AllowDrop = false;
C2_F6.AllowDrop = false;
C2_F7.AllowDrop = false;
C2_F8.AllowDrop = false;
C2_F9.AllowDrop = false;

C3_F0.AllowDrop = false;
C3_F1.AllowDrop = false;
C3_F2.AllowDrop = false;
C3_F3.AllowDrop = true;
C3_F4.AllowDrop = false;
C3_F5.AllowDrop = false;
```

```
    C3_F6.AllowDrop = false;
    C3_F7.AllowDrop = false;
    C3_F8.AllowDrop = false;
    C3_F9.AllowDrop = false;
    break;
case 7:
    C0_F0.AllowDrop = false;
    C0_F1.AllowDrop = false;
    C0_F2.AllowDrop = true;
    C0_F3.AllowDrop = false;
    C0_F4.AllowDrop = false;
    C0_F5.AllowDrop = false;
    C0_F6.AllowDrop = false;
    C0_F7.AllowDrop = false;
    C0_F8.AllowDrop = false;
    C0_F9.AllowDrop = false;

    C1_F0.AllowDrop = false;
    C1_F1.AllowDrop = false;
    C1_F2.AllowDrop = true;
    C1_F3.AllowDrop = false;
    C1_F4.AllowDrop = false;
    C1_F5.AllowDrop = false;
    C1_F6.AllowDrop = false;
    C1_F7.AllowDrop = false;
    C1_F8.AllowDrop = false;
    C1_F9.AllowDrop = false;

    C2_F0.AllowDrop = false;
    C2_F1.AllowDrop = false;
    C2_F2.AllowDrop = true;
```

```
    C2_F3.AllowDrop = false;
    C2_F4.AllowDrop = false;
    C2_F5.AllowDrop = false;
    C2_F6.AllowDrop = false;
    C2_F7.AllowDrop = false;
    C2_F8.AllowDrop = false;
    C2_F9.AllowDrop = false;

    C3_F0.AllowDrop = false;
    C3_F1.AllowDrop = false;
    C3_F2.AllowDrop = true;
    C3_F3.AllowDrop = false;
    C3_F4.AllowDrop = false;
    C3_F5.AllowDrop = false;
    C3_F6.AllowDrop = false;
    C3_F7.AllowDrop = false;
    C3_F8.AllowDrop = false;
    C3_F9.AllowDrop = false;
    break;
case 8:
    C0_F0.AllowDrop = false;
    C0_F1.AllowDrop = true;
    C0_F2.AllowDrop = false;
    C0_F3.AllowDrop = false;
    C0_F4.AllowDrop = false;
    C0_F5.AllowDrop = false;
    C0_F6.AllowDrop = false;
    C0_F7.AllowDrop = false;
    C0_F8.AllowDrop = false;
    C0_F9.AllowDrop = false;
```

```
C1_F0.AllowDrop = false;
C1_F1.AllowDrop = true;
C1_F2.AllowDrop = false;
C1_F3.AllowDrop = false;
C1_F4.AllowDrop = false;
C1_F5.AllowDrop = false;
C1_F6.AllowDrop = false;
C1_F7.AllowDrop = false;
C1_F8.AllowDrop = false;
C1_F9.AllowDrop = false;

C2_F0.AllowDrop = false;
C2_F1.AllowDrop = true;
C2_F2.AllowDrop = false;
C2_F3.AllowDrop = false;
C2_F4.AllowDrop = false;
C2_F5.AllowDrop = false;
C2_F6.AllowDrop = false;
C2_F7.AllowDrop = false;
C2_F8.AllowDrop = false;
C2_F9.AllowDrop = false;

C3_F0.AllowDrop = false;
C3_F1.AllowDrop = true;
C3_F2.AllowDrop = false;
C3_F3.AllowDrop = false;
C3_F4.AllowDrop = false;
C3_F5.AllowDrop = false;
C3_F6.AllowDrop = false;
C3_F7.AllowDrop = false;
C3_F8.AllowDrop = false;
```

```
    C3_F9.AllowDrop = false;
    break;

case 9:
    C0_F0.AllowDrop = true;
    C0_F1.AllowDrop = false;
    C0_F2.AllowDrop = false;
    C0_F3.AllowDrop = false;
    C0_F4.AllowDrop = false;
    C0_F5.AllowDrop = false;
    C0_F6.AllowDrop = false;
    C0_F7.AllowDrop = false;
    C0_F8.AllowDrop = false;
    C0_F9.AllowDrop = false;

    C1_F0.AllowDrop = true;
    C1_F1.AllowDrop = false;
    C1_F2.AllowDrop = false;
    C1_F3.AllowDrop = false;
    C1_F4.AllowDrop = false;
    C1_F5.AllowDrop = false;
    C1_F6.AllowDrop = false;
    C1_F7.AllowDrop = false;
    C1_F8.AllowDrop = false;
    C1_F9.AllowDrop = false;

    C2_F0.AllowDrop = true;
    C2_F1.AllowDrop = false;
    C2_F2.AllowDrop = false;
    C2_F3.AllowDrop = false;
    C2_F4.AllowDrop = false;
```

```
            C2_F5.AllowDrop = false;
            C2_F6.AllowDrop = false;
            C2_F7.AllowDrop = false;
            C2_F8.AllowDrop = false;
            C2_F9.AllowDrop = false;

            C3_F0.AllowDrop = true;
            C3_F1.AllowDrop = false;
            C3_F2.AllowDrop = false;
            C3_F3.AllowDrop = false;
            C3_F4.AllowDrop = false;
            C3_F5.AllowDrop = false;
            C3_F6.AllowDrop = false;
            C3_F7.AllowDrop = false;
            C3_F8.AllowDrop = false;
            C3_F9.AllowDrop = false;
            break;

    }
}

// Reinicia todas las filas a su valor inicial -> en blanco
private void ReiniciarFilas()
{
    // Primeras columnas de todas las filas
    C0_F0.Background = new SolidColorBrush(Colors.White);
    C0_F1.Background = new SolidColorBrush(Colors.White);
    C0_F2.Background = new SolidColorBrush(Colors.White);
    C0_F3.Background = new SolidColorBrush(Colors.White);
    C0_F4.Background = new SolidColorBrush(Colors.White);
    C0_F5.Background = new SolidColorBrush(Colors.White);
```

```
C0_F6.Background = new SolidColorBrush(Colors.White);
C0_F7.Background = new SolidColorBrush(Colors.White);
C0_F8.Background = new SolidColorBrush(Colors.White);
C0_F9.Background = new SolidColorBrush(Colors.White);
// Segundas columnas de todas las filas
C1_F0.Background = new SolidColorBrush(Colors.White);
C1_F1.Background = new SolidColorBrush(Colors.White);
C1_F2.Background = new SolidColorBrush(Colors.White);
C1_F3.Background = new SolidColorBrush(Colors.White);
C1_F4.Background = new SolidColorBrush(Colors.White);
C1_F5.Background = new SolidColorBrush(Colors.White);
C1_F6.Background = new SolidColorBrush(Colors.White);
C1_F7.Background = new SolidColorBrush(Colors.White);
C1_F8.Background = new SolidColorBrush(Colors.White);
C1_F9.Background = new SolidColorBrush(Colors.White);
// Terceras columnas de todas las filas
C2_F0.Background = new SolidColorBrush(Colors.White);
C2_F1.Background = new SolidColorBrush(Colors.White);
C2_F2.Background = new SolidColorBrush(Colors.White);
C2_F3.Background = new SolidColorBrush(Colors.White);
C2_F4.Background = new SolidColorBrush(Colors.White);
C2_F5.Background = new SolidColorBrush(Colors.White);
C2_F6.Background = new SolidColorBrush(Colors.White);
C2_F7.Background = new SolidColorBrush(Colors.White);
C2_F8.Background = new SolidColorBrush(Colors.White);
C2_F9.Background = new SolidColorBrush(Colors.White);
// Cuartas columnas de todas las filas
C3_F0.Background = new SolidColorBrush(Colors.White);
C3_F1.Background = new SolidColorBrush(Colors.White);
C3_F2.Background = new SolidColorBrush(Colors.White);
C3_F3.Background = new SolidColorBrush(Colors.White);
```

```
C3_F4.Background = new SolidColorBrush(Colors.White);
C3_F5.Background = new SolidColorBrush(Colors.White);
C3_F6.Background = new SolidColorBrush(Colors.White);
C3_F7.Background = new SolidColorBrush(Colors.White);
C3_F8.Background = new SolidColorBrush(Colors.White);
C3_F9.Background = new SolidColorBrush(Colors.White);
// Acierto 1 de todas las filas
A1_F1.Background = new SolidColorBrush(Colors.White);
A1_F2.Background = new SolidColorBrush(Colors.White);
A1_F3.Background = new SolidColorBrush(Colors.White);
A1_F4.Background = new SolidColorBrush(Colors.White);
A1_F5.Background = new SolidColorBrush(Colors.White);
A1_F6.Background = new SolidColorBrush(Colors.White);
A1_F7.Background = new SolidColorBrush(Colors.White);
A1_F8.Background = new SolidColorBrush(Colors.White);
A1_F9.Background = new SolidColorBrush(Colors.White);
A1_F10.Background = new SolidColorBrush(Colors.White);
// Acierto 2 de todas las filas
A2_F1.Background = new SolidColorBrush(Colors.White);
A2_F2.Background = new SolidColorBrush(Colors.White);
A2_F3.Background = new SolidColorBrush(Colors.White);
A2_F4.Background = new SolidColorBrush(Colors.White);
A2_F5.Background = new SolidColorBrush(Colors.White);
A2_F6.Background = new SolidColorBrush(Colors.White);
A2_F7.Background = new SolidColorBrush(Colors.White);
A2_F8.Background = new SolidColorBrush(Colors.White);
A2_F9.Background = new SolidColorBrush(Colors.White);
A2_F10.Background = new SolidColorBrush(Colors.White);
// Acierto 3 de todas las filas
A3_F1.Background = new SolidColorBrush(Colors.White);
A3_F2.Background = new SolidColorBrush(Colors.White);
```

```
    A3_F3.Background = new SolidColorBrush(Colors.White);
    A3_F4.Background = new SolidColorBrush(Colors.White);
    A3_F5.Background = new SolidColorBrush(Colors.White);
    A3_F6.Background = new SolidColorBrush(Colors.White);
    A3_F7.Background = new SolidColorBrush(Colors.White);
    A3_F8.Background = new SolidColorBrush(Colors.White);
    A3_F9.Background = new SolidColorBrush(Colors.White);
    A3_F10.Background = new SolidColorBrush(Colors.White);
    // Acierto 4 de todas las filas
    A4_F1.Background = new SolidColorBrush(Colors.White);
    A4_F2.Background = new SolidColorBrush(Colors.White);
    A4_F3.Background = new SolidColorBrush(Colors.White);
    A4_F4.Background = new SolidColorBrush(Colors.White);
    A4_F5.Background = new SolidColorBrush(Colors.White);
    A4_F6.Background = new SolidColorBrush(Colors.White);
    A4_F7.Background = new SolidColorBrush(Colors.White);
    A4_F8.Background = new SolidColorBrush(Colors.White);
    A4_F9.Background = new SolidColorBrush(Colors.White);
    A4_F10.Background = new SolidColorBrush(Colors.White);
}


// Inicializa el puntaje inicial
private void InicializarPuntaje()
{
    puntaje_actual.Text = score.ToString();
    puntaje_maximo.Text = InitMaxScore();
}

// Trae puntajes máximos existentes
private string InitMaxScore()
```

```
{
    string curFile = @"scores.txt";
    // Verifica si existen datos guardados
    if (File.Exists(curFile))
    {
        string[] scores = new string[3];
        int count = 0;
        var linesRead = File.ReadLines("scores.txt");
        foreach (var iter in linesRead)
        {
            scores[count++] = iter;
        }
        if (game.GetIdLevel() == 1)
        {
            return scores[0];
        }
        else if (game.GetIdLevel() == 2)
        {
            return scores[1];
        }
        else if (game.GetIdLevel() == 3)
        {
            return scores[2];
        }
    }
    else
    {
        using (FileStream fs = File.Create(curFile))
        {
            Byte[] temp_score = new UTF8Encoding(true).GetBytes("0\n");
            fs.Write(temp_score, 0, temp_score.Length);
```

```
                fs.Write(temp_score, 0, temp_score.Length);
                fs.Write(temp_score, 0, temp_score.Length);
            }
        }
        return "0";
    }

    #endregion

    #region Notificaciones

    // Muestra un mensaje de que el jugador gano
    private void NotifyWon(int p_score)
    {
        string message = $"¡Felicitaciones!\n\n ¡Has pasado el nivel! Tú puntaje fue de {p_score}.";
        string caption = "¡Has ganado!";
        MessageBoxButton button = MessageBoxButton.OK;
        MessageBoxResult result = MessageBox.Show(message, caption, button);
        switch (result)
        {
            // Vuelve al menú de niveles
            case MessageBoxResult.OK:
                NavigationService.Navigate(new SelectLevel());
                break;
        }
    }

    // Muestra un mensaje de que el jugador perdio
    private void NotifyLost()
    {
```

```
        string message = "¡Oh No! No has logrado superar el nivel...\n\n¿Deseas intentar de nuevo?";
        string caption = "¡Perdiste!";
        MessageBoxButton button = MessageBoxButton.YesNo;
        MessageBoxResult result = MessageBox.Show(message, caption, button);
        switch (result)
        {
            // Reinicia el nivel
            case MessageBoxResult.Yes:
                Reboot();
                break;

            // Vuelve al menú de niveles
            case MessageBoxResult.No:
                NavigationService.Navigate(new SelectLevel());
                break;
        }
    }

    // Muestra las configuraciones del nivel
    private void NotifyGameSettings()
    {
        string message = "¡Bienvenido!\nHas seleccionado el nivel de dificultad difícil, tienes 8 colores para crear tus combinaciones y estos se pueden repetir.";
        string caption = "Nivel: Difícil";
        MessageBoxButton button = MessageBoxButton.OK;
        MessageBoxResult result = MessageBox.Show(message, caption, button);
        switch (result)
        {
            // No ocurre nada
            case MessageBoxResult.OK:
```

```
                break;
            }
        }

        // Notifica que un color ha sido repetido
        private void NotifyRepeatedColor()
        {
            string message = "¡Has repetido un color!\n¡Recuerda que en este nivel no esta permitido su repetición!";
            string caption = "Advertencia";
            MessageBoxButton button = MessageBoxButton.OK;
            MessageBoxResult result = MessageBox.Show(message, caption, button);
            switch (result)
            {
                // No sucede nada
                case MessageBoxResult.OK:
                    break;
            }
        }

        // Notifica si la combinación fue ejecutada vacía
        private void NotifyEmpty()
        {
            string message = "Parece que no has colocado ningún tipo de combinación...\n\n¡Coloca alguna antes de probar!";
            string caption = "Advertencia";
            MessageBoxButton button = MessageBoxButton.OK;
            MessageBoxResult result = MessageBox.Show(message, caption, button);
        }

        // Notifica que hay un item en blanco en la combinación
```

```
        private void NotifyOneEmpty()
        {
            string message = "Parece que has dejado un espacio en blanco...\n\n¡Ten más cuidado la próxima vez!";
            string caption = "Advertencia";
            MessageBoxButton button = MessageBoxButton.OK;
            MessageBoxResult result = MessageBox.Show(message, caption, button);
        }

        // Notifica que se logro un nuevo record
        private void NotifyMaxScore(int p_score)
        {
            string message = $"¡Has logrado un nuevo record!\n\nObtuviste un puntaje de: {p_score}";
            string caption = "Felicitaciones";
            MessageBoxButton button = MessageBoxButton.OK;
            MessageBoxResult result = MessageBox.Show(message, caption, button);
        }

        // Notifica las instrucciones del juego
        private void NotifyRules()
        {
            string message = "¡Recuerda!\n\nCada vez que se empieza una partida, el juego creará un nuevo código secreto que lo debes decifrar en un máximo de 10 intentos que le brindaràn de pistas para resolver el código." +
"\n\n**********************************************************************\n\n" +
                "Sobre el juego:\n\nPara poder acertar con mayor facilidad el código secreto ¡podrás guiarte de las pistas que el juego te proveerá:\n" +
                "1. El color negro significa que tienes un color en la posición correcta!\n" +
                "2. El color gris significa que tienes un color correcta pero en la posición incorrecta!";
            string caption = "Reglas del juego";
```

```
            MessageBoxButton buttons = MessageBoxButton.OK;
            MessageBoxResult result = MessageBox.Show(message, caption, buttons);
        }

        #endregion
    }
}
```