Universidad Mariano Gálvez De Guatemala
Facultad De Ingeniería En Sistemas De La Información
Y Ciencias De La Computación

Tarea 3
BUSQUEDA NO INFORMADA

Alba Gabriela Yool Alvarado
1990-19-15166
Guatemala, marzo 2023

Búsqueda no informada.

La búsqueda a ciegas, también conocida como búsqueda no informada, es un tipo de algoritmo de búsqueda en inteligencia artificial que no utiliza información específica sobre el problema o el objetivo a alcanzar. A diferencia de la búsqueda informada, en la que se utiliza información heurística para guiar la exploración del espacio de búsqueda, la búsqueda a ciegas se basa en la exploración sistemática de diferentes opciones hasta encontrar una solución.

La búsqueda a ciegas es útil cuando se desconoce información específica sobre el problema o la solución deseada, o cuando la información disponible es muy limitada o costosa de obtener. Los algoritmos de búsqueda a ciegas más comunes son la búsqueda en profundidad, la búsqueda en anchura, la búsqueda de costo uniforme, la búsqueda en profundidad limitada y la búsqueda bidireccional.

Los algoritmos de búsqueda no informada, también conocidos como búsqueda a ciegas, no utilizan información específica sobre el problema o el objetivo a alcanzar

1. **Búsqueda en profundidad:** Este algoritmo comienza en el nodo raíz y explora todos los posibles caminos hacia abajo en el árbol de búsqueda. Si no se encuentra la solución, retrocede al nodo anterior y continúa explorando otros caminos. La búsqueda en profundidad se utiliza a menudo en problemas de laberintos o en la exploración de árboles de juego.
2. **Búsqueda de costo uniforme:** Este algoritmo expande el nodo con el costo más bajo en cada iteración, hasta encontrar la solución deseada. Este algoritmo se utiliza a menudo en problemas de optimización en los que se busca la solución de menor costo.
3. **Búsqueda en anchura:** Este algoritmo explora el árbol de búsqueda por niveles, comenzando desde la raíz y explorando todos los nodos de un nivel antes de avanzar al siguiente nivel. La búsqueda en anchura se utiliza a menudo en problemas de optimización en los que se busca la solución más corta.
4. **Búsqueda en profundidad limitada:** Este algoritmo es similar a la búsqueda en profundidad, pero tiene una restricción en la profundidad máxima a la que se puede llegar. Este algoritmo puede ser útil en problemas donde se requiere encontrar una

solución en un límite de profundidad, como en juegos de ajedrez o en laberintos con caminos muy largos.

5. **Búsqueda bidireccional**: Este algoritmo utiliza dos búsquedas, una desde el estado inicial y otra desde el estado objetivo, y se encuentran en el medio. Este algoritmo puede ser útil para reducir el espacio de búsqueda y mejorar la eficiencia en la búsqueda.

En general, los algoritmos de búsqueda no informada son útiles cuando se desconoce información específica sobre el problema o la solución deseada, o cuando la información disponible es muy limitada o costosa de obtener.

Requerimiento del problema de búsqueda no informada.

Características el programa:

El usuario deberá construir un entorno para los movimientos que consisten en:

- 💻 Definir las dimensiones del tablero
- 💻 Definir la posición inicial del camino
- 💻 Definir la posición de los obstáculos (menos de 3 obstáculos)
- 💻 Definir la posición final del camino (punto extra si logran definir más de una posición final)

El sistema deberá marcar todas las etapas de la búsqueda no informada donde deberá indicar en pantalla los siguientes valores.

a) Número de caminos realizados.

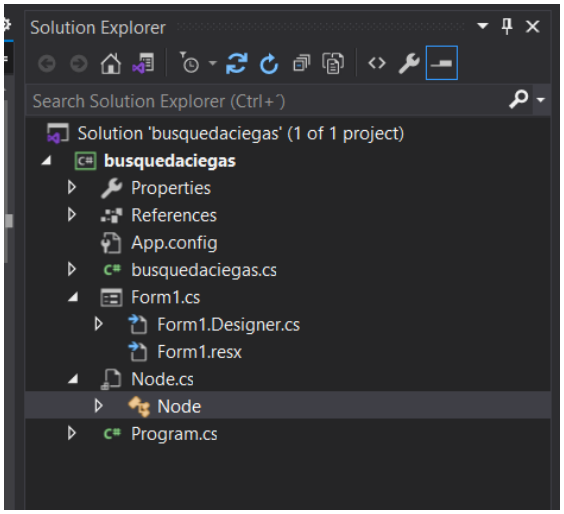
b) Numero de caminos buenos.

c) Número de caminos óptimos.

d) Peso del camino optimo.

Documentación del programa.

- El programa se realizó en c#



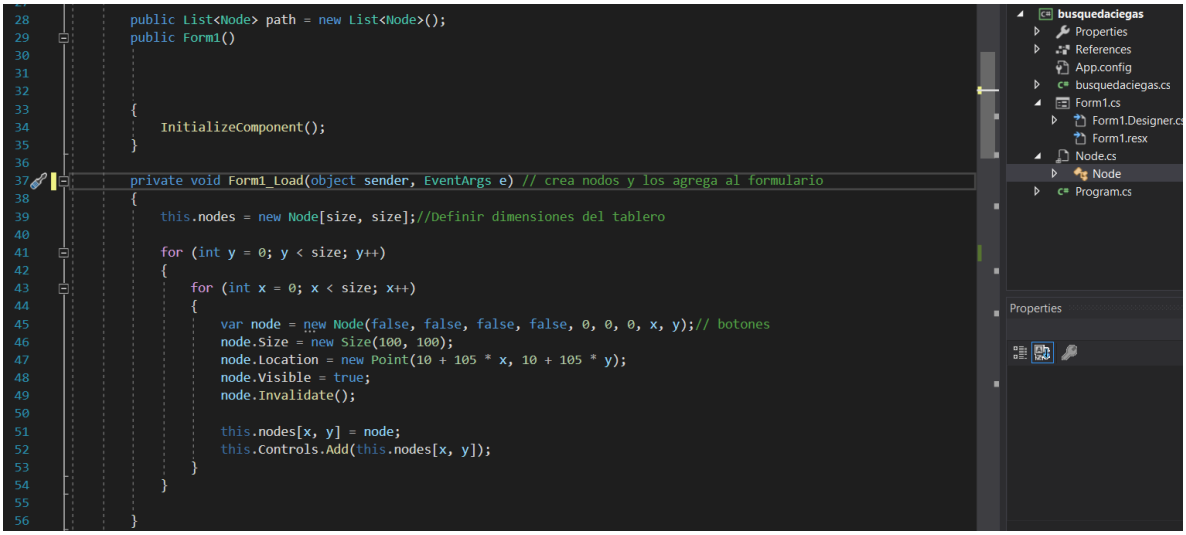
La creación de tres clases en las cual se trabajo es, busquedaciegas, Form1, Node.

Debido a los requerimientos del programa, debemos

- **La clase Form1** en la interfaz gráfica se muestra de esta manera.

En la interfaz gráfica únicamente definiremos los botones de búsqueda inicio, fin, obstáculo, camino, y un textBox donde se mostrarán estadísticas.

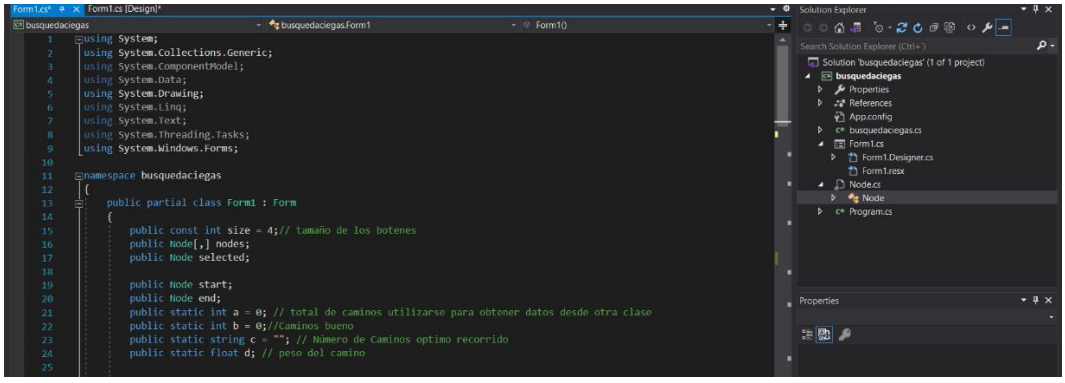
- Ingresamos al codigo de la interfaz, debido a que se nos solicita definir dimensiones del tablero lo haremos a través de codigo.



```
28      public List<Node> path = new List<Node>();
29      public Form1()
30
31
32
33      {
34          InitializeComponent();
35      }
36
37      private void Form1_Load(object sender, EventArgs e) // crea nodos y los agrega al formulario
38      {
39          this.nodes = new Node[size, size];//Definir dimensiones del tablero
40
41          for (int y = 0; y < size; y++)
42          {
43              for (int x = 0; x < size; x++)
44              {
45                  var node = new Node(false, false, false, false, 0, 0, 0, x, y);// botones
46                  node.Size = new Size(100, 100);
47                  node.Location = new Point(10 + 105 * x, 10 + 105 * y);
48                  node.Visible = true;
49                  node.Invalidate();
50
51                  this.nodes[x, y] = node;
52                  this.Controls.Add(this.nodes[x, y]);
53              }
54          }
55
56      }
```

- Este código crea una matriz de nodos que se representan visualmente como botones en un formulario.
- Los nodos se organizan en una cuadrícula con dimensiones definidas por la variable "size". Cada nodo se inicializa con ciertas propiedades, como su posición en la cuadrícula y si es un obstáculo o no. Además, se crea una lista de nodos llamada "path".
- El método "Form1_Load" se ejecuta cuando se carga el formulario y se encarga de crear los nodos y agregarlos al formulario para que se muestren.

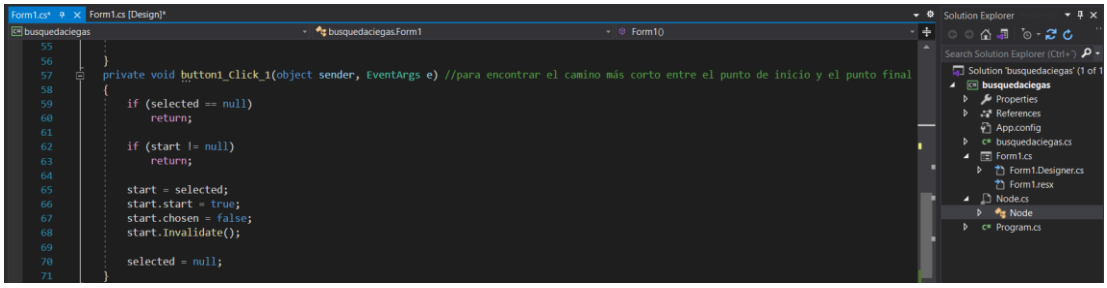
➢ Dentro de esta clase tambien vamos a definir variables, que servirán para visualizar en el textBox, nos dará una serie de estadistas de caminos buenos, caminos óptimos, caminos totales.



```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace busquedaciegas
{
    public partial class Form1 : Form
    {
        public const int size = 4;// tamaño de los botenes
        public Node[,] nodes;
        public Node selected;

        public Node start;
        public Node end;
        public static int a = 0; // total de caminos utilizarse para obtener datos desde otra clase
        public static int b = 0;//Caminos bueno
        public static string c = ""; // Número de Caminos optimo recorrido
        public static float d; // peso del camino
```

Esta clase proporciona un conjunto de variables que se utilizan para realizar un seguimiento del estado de la aplicación, como los nodos seleccionados y los resultados de los cálculos de ruta.

- Tenemos el botón 1 el cual nos servirá para encontrar el camino más corto.

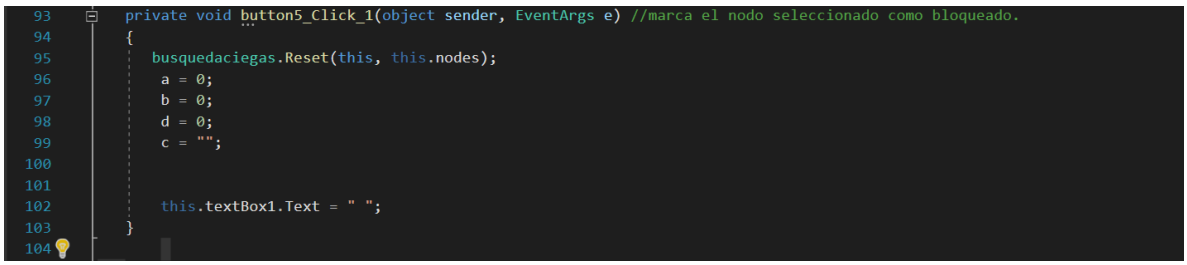


```
}
private void button1_Click_1(object sender, EventArgs e) //para encontrar el camino más corto entre el punto de inicio y el punto final
{
    if (selected == null)
        return;

    if (start != null)
        return;

    start = selected;
    start.start = true;
    start.chosen = false;
    start.Invalidate();

    selected = null;
}
```

- Este código define el método "button4_Click_1", que se ejecuta cuando se hace clic en el botón correspondiente en el formulario. El objetivo de este método es encontrar el camino más corto entre el nodo de inicio y el nodo final en la cuadrícula.

```
private void button4_Click_1(object sender, EventArgs e) //para encontrar el camino más corto entre el punto de inicio y el punto final
{
    if (start == null || end == null)
        return;

    busquedaciegas.FindPath(this, this.nodes, start, end);
     busquedaciegas.ShowPath(this, this.path, nodes, end);

    b++;// caminos buenos tt


     //muestra los datos en nuestra tabla de texto.
     this.textBox1.Text = "\r\n" + "\r\n" + " Número de caminos: " + a.ToString() + "\r\n" + "\r\n" +
         " Número de Caminos buenos: " + b.ToString() + "\r\n" + "\r\n" +

         " Número de Caminos optimo: " + c.ToString() + "\r\n" + "\r\n" +

         " Peso del camino Optimo: " + d.ToString();
}
```

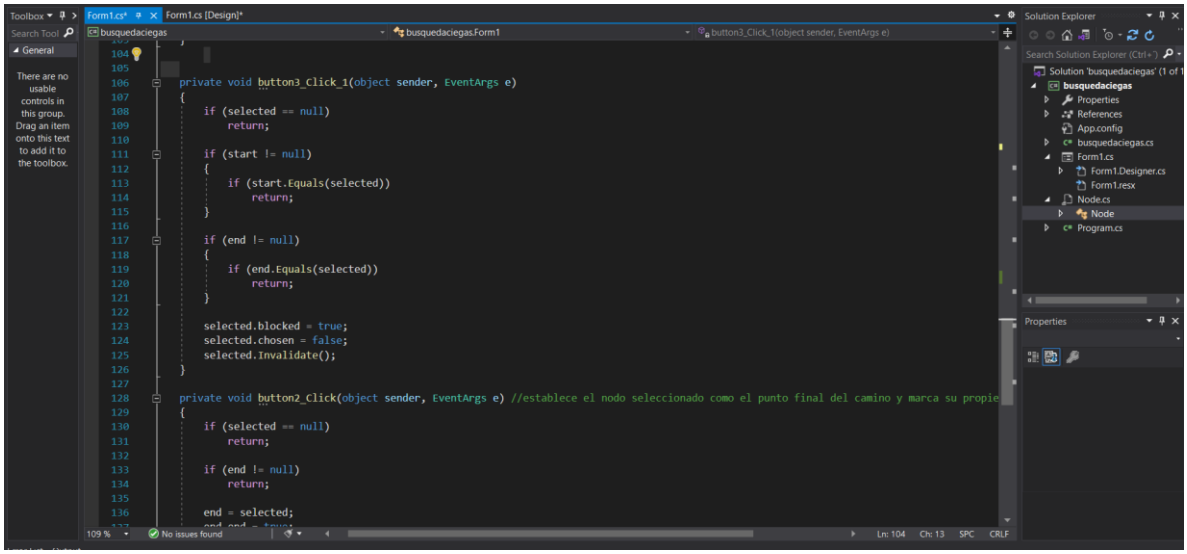
- Tenemos el método "button5_Click_1", que se ejecuta cuando se hace clic en el botón correspondiente en el formulario. El objetivo de este método es reiniciar la cuadrícula y las variables estáticas a su estado inicial.



```
private void button5_Click_1(object sender, EventArgs e) //marca el nodo seleccionado como bloqueado.
{
    busquedaciegas.Reset(this, this.nodes);
     a = 0;
     b = 0;
     d = 0;
     c = "";


     this.textBox1.Text = " ";
}
```

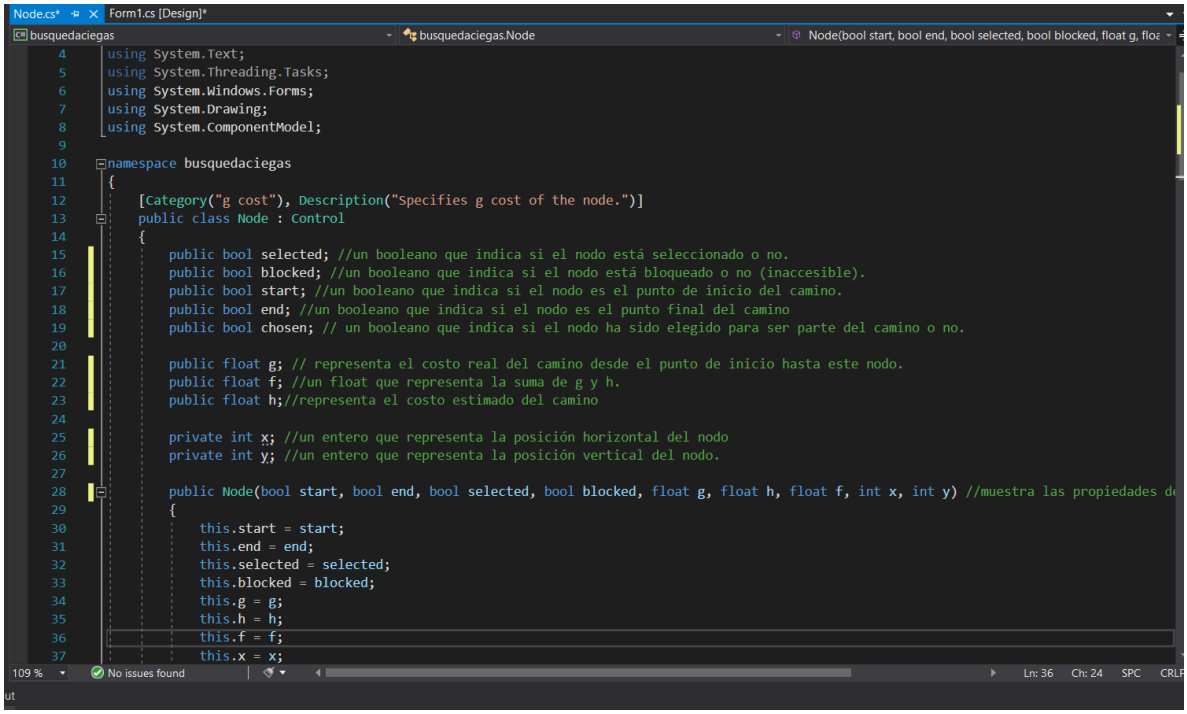
el método llama al método "Reset" de la clase "busquedaciegas", que desbloquea todos los nodos de la cuadrícula y restablece sus valores de peso y distancia. Luego, se reinician las variables estáticas "a" (número total de caminos), "b" (número de caminos buenos), "d" (peso del camino óptimo) y "c" (número de caminos óptimos recorridos).

- Por último, el método "button2_Click" y el"button3_Click_1" marca el nodo seleccionado como bloqueado o que no se puede acceder, para verificar si el nodo seleccionado es nulo, si el punto de inicio o fin es igual al nodo seleccionado y si el nodo ya está bloqueado.



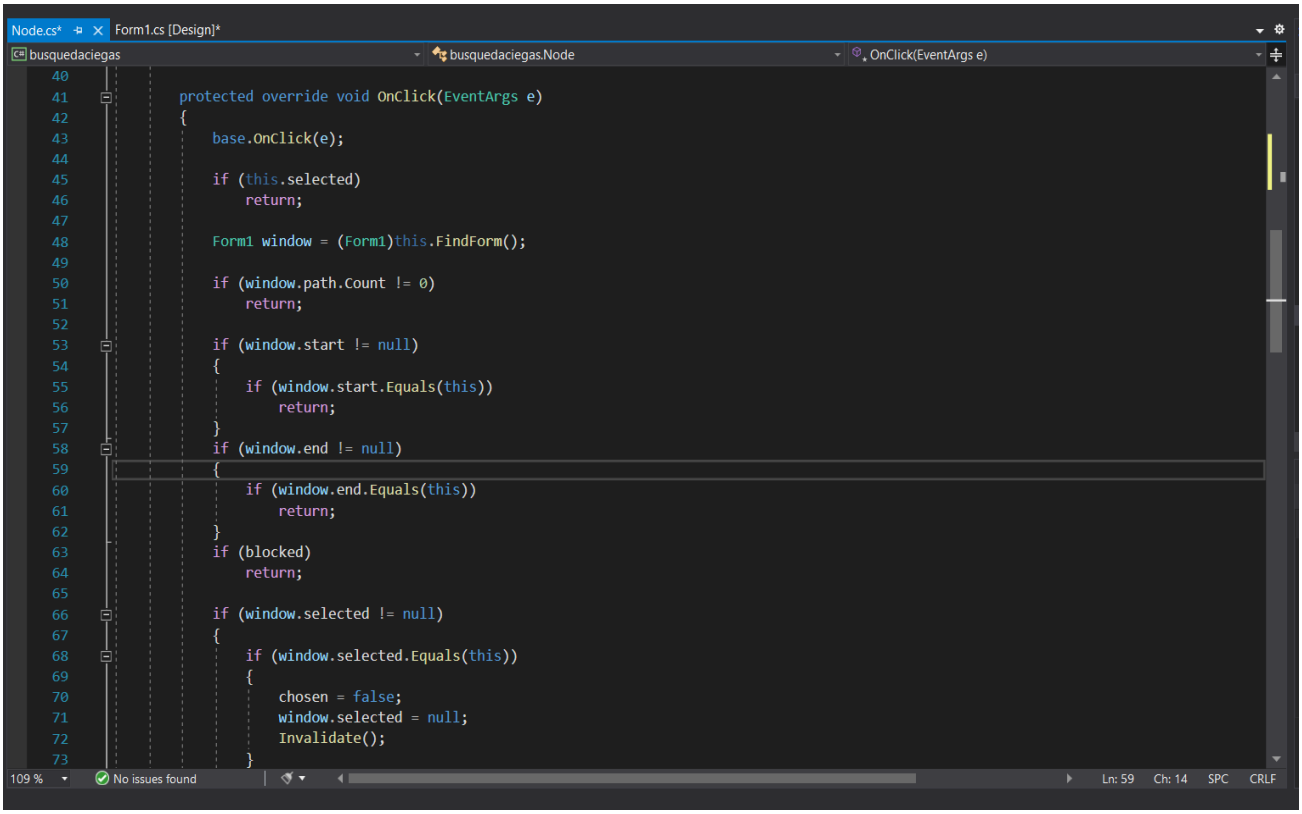
```
private void button3_Click_1(object sender, EventArgs e)
{
    if (selected == null)
        return;

    if (start != null)
    {
        if (start.Equals(selected))
            return;
    }

    if (end != null)
    {
        if (end.Equals(selected))
            return;
    }

    selected.blocked = true;
    selected.chosen = false;
    selected.Invalidate();
}

private void button2_Click(object sender, EventArgs e) //establece el nodo seleccionado como el punto final del camino y marca su propie
{
    if (selected == null)
        return;

    if (end != null)
        return;

    end = selected;
```

- **Clase Node:** En esta clase podemos encontrar el diseño, la información de los nodos, se muestran las variables que hemos definido en Form1.

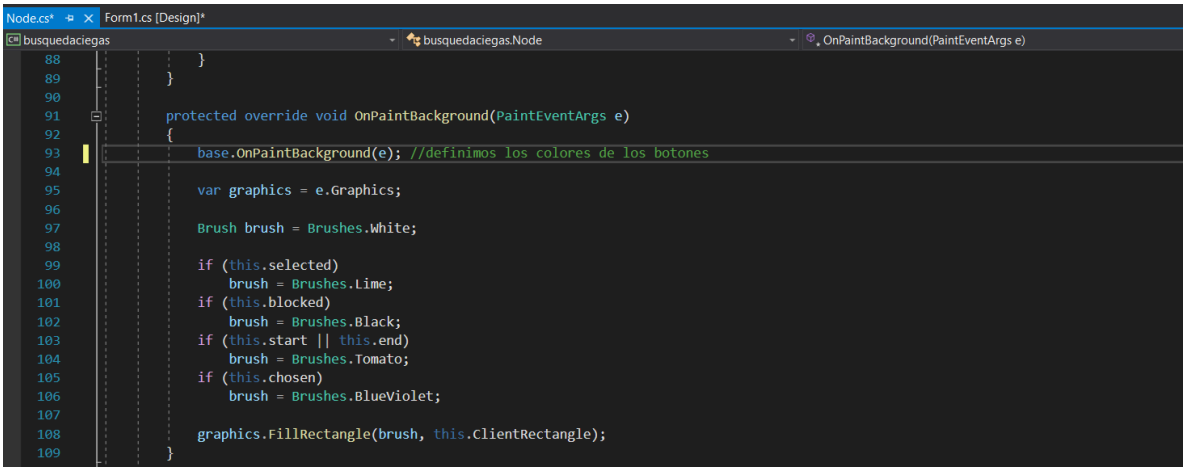


```
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Drawing;
using System.ComponentModel;

namespace busquedaciegas
{
    [Category("g cost"), Description("Specifies g cost of the node.")]
    public class Node : Control
    {
        public bool selected; //un booleano que indica si el nodo está seleccionado o no.
        public bool blocked; //un booleano que indica si el nodo está bloqueado o no (inaccesible).
        public bool start; //un booleano que indica si el nodo es el punto de inicio del camino.
        public bool end; //un booleano que indica si el nodo es el punto final del camino
        public bool chosen; // un booleano que indica si el nodo ha sido elegido para ser parte del camino o no.

        public float g; // representa el costo real del camino desde el punto de inicio hasta este nodo.
        public float f; //un float que representa la suma de g y h.
        public float h;//representa el costo estimado del camino

        private int x; //un entero que representa la posición horizontal del nodo
        private int y; //un entero que representa la posición vertical del nodo.

        public Node(bool start, bool end, bool selected, bool blocked, float g, float h, float f, int x, int y) //muestra las propiedades d
        {
            this.start = start;
            this.end = end;
            this.selected = selected;
            this.blocked = blocked;
            this.g = g;
            this.h = h;
            this.f = f;
            this.x = x;
```

- maneja los clics del mouse en el control del nodo.



- Diseño de los botones y de la interfaz.

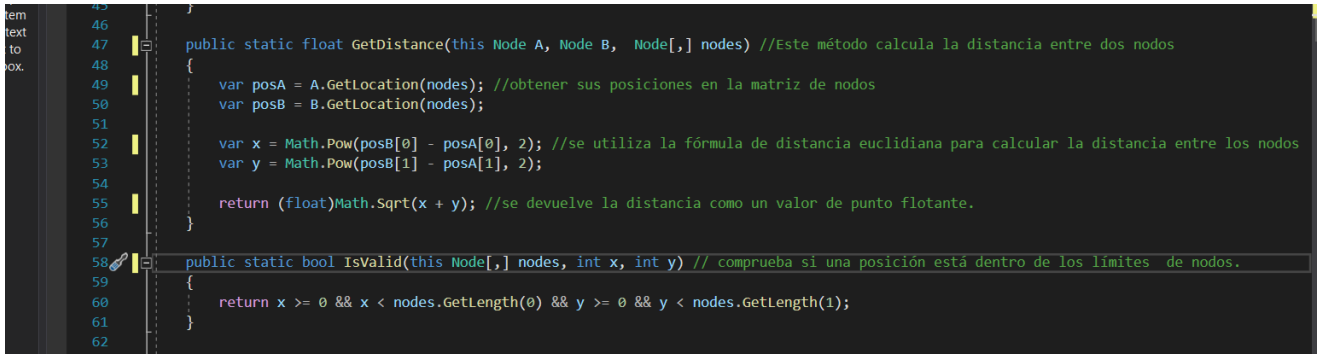


```
Node.cs*  Form1.cs [Design]*
busquedaciegas  busquedaciegas.Node  OnPaintBackground(PaintEventArgs e)

protected override void OnPaint(PaintEventArgs e)
{
    base.OnPaint(e);

    var graphics = e.Graphics;

    var brush = Brushes.White;
    var font = new Font(FontFamily.GenericSansSerif, 10, FontStyle.Bold);

    string pos = string.Format("({0}; {1})", x, y);
    string cost = f.ToString();

    var costPos = new Point(ClientRectangle.Left, ClientRectangle.Bottom - 20);

    if (start)
    {
        var textFont = new Font(FontFamily.GenericSansSerif, 20, FontStyle.Bold);

        var textPos = new PointF(ClientRectangle.Size.Width / 2 - 52.5f, ClientRectangle.Size.Height / 2 - 15);
        graphics.DrawString("Inicio", textFont, Brushes.White, textPos);
    }

    if (end)
    {
        var textFont = new Font(FontFamily.GenericSansSerif, 20, FontStyle.Bold);

        var textPos = new PointF(ClientRectangle.Size.Width / 2 - 52.5f, ClientRectangle.Size.Height / 2 - 15);
        graphics.DrawString("Fin", textFont, Brushes.White, textPos);
    }

    graphics.DrawString(pos, font, brush, ClientRectangle.Location);
    graphics.DrawString(cost, font, brush, costPos);
}

109 %  No issues found  Ln: 93  Ch: 78  SPC  CRLF
```

- **Clase busquedaciegas:** Muestra una serie de estadísticas como caminos buenos, óptimos, peso del camino.

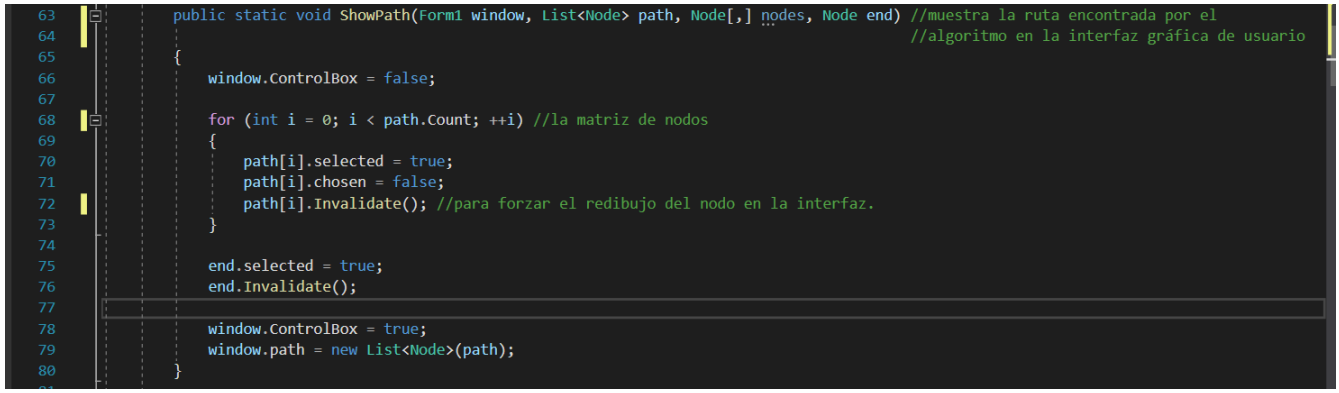
```
busquedaciegas.cs*  Node.cs*  Form1.cs [Design]*
busquedaciegas  busquedaciegas.busquedaciegas  GetLocation(Node node, Node[,] nodes)

public static class busquedaciegas
{
    public static Node MinF(this List<Node> nodes) //recibe como parámetro una lista de nodos y devuelve el nodo con el valor f
    {
        Node min = nodes[nodes.Count - 1];

        for (int i = 1; i < nodes.Count; ++i)
        {
            if (nodes[i].f < min.f)
                min = nodes[i];
        }

        return min;
    }

    public static int[] GetLocation(this Node node,  Node[,] nodes) //recibe como parámetros un nodo y una matriz de nodos.
                                                                    //Devuelve la ubicación del nodo en la matriz
    {
        int[] location = new int[2];

        for (int y = 0; y < nodes.GetLength(1); ++y)
        {
            for (int x = 0; x < nodes.GetLength(0); ++x)
            {
                if (nodes[x, y].Equals(node))
                {
                    location[0] = x;
                    location[1] = y;

                    return location;
                }
            }
        }

        return null;
    }

99 %  No issues found  Ln: 36  Ch: 41  SPC  CRLF
```

- Luego tenemos el método para calcular los nodos.



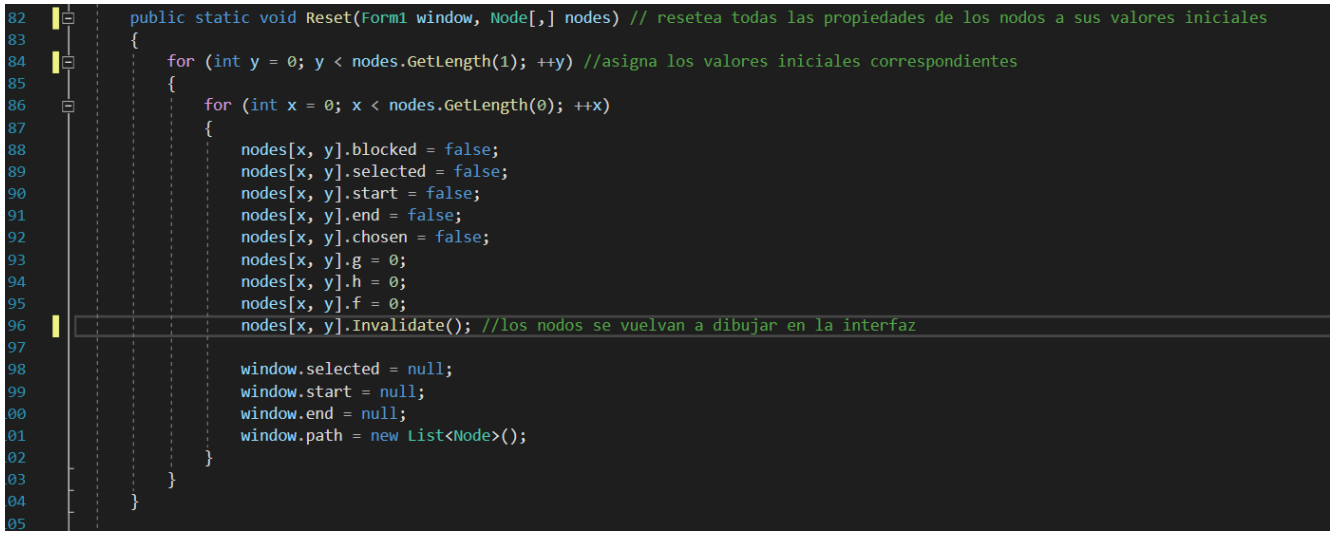
```
    }

    public static float GetDistance(this Node A, Node B,  Node[,] nodes) //Este método calcula la distancia entre dos nodos
    {
        var posA = A.GetLocation(nodes); //obtener sus posiciones en la matriz de nodos
        var posB = B.GetLocation(nodes);

        var x = Math.Pow(posB[0] - posA[0], 2); //se utiliza la fórmula de distancia euclidiana para calcular la distancia entre los nodos
        var y = Math.Pow(posB[1] - posA[1], 2);

        return (float)Math.Sqrt(x + y); //se devuelve la distancia como un valor de punto flotante.
    }

    public static bool IsValid(this Node[,] nodes, int x, int y) // comprueba si una posición está dentro de los límites  de nodos.
    {
        return x >= 0 && x < nodes.GetLength(0) && y >= 0 && y < nodes.GetLength(1);
    }

```

- La ruta encontrada.



```
    public static void ShowPath(Form1 window, List<Node> path, Node[,] nodes, Node end) //muestra la ruta encontrada por el
                                                                                         //algoritmo en la interfaz gráfica de usuario
    {
        window.ControlBox = false;

        for (int i = 0; i < path.Count; ++i) //la matriz de nodos
        {
            path[i].selected = true;
            path[i].chosen = false;
            path[i].Invalidate(); //para forzar el redibujo del nodo en la interfaz.
        }

        end.selected = true;
        end.Invalidate();

        window.ControlBox = true;
        window.path = new List<Node>(path);
    }
```

- Búsqueda de rutas.



```
public static void Reset(Form1 window, Node[,] nodes) // resetea todas las propiedades de los nodos a sus valores iniciales
{
    for (int y = 0; y < nodes.GetLength(1); ++y) //asigna los valores iniciales correspondientes
    {
        for (int x = 0; x < nodes.GetLength(0); ++x)
        {
            nodes[x, y].blocked = false;
            nodes[x, y].selected = false;
            nodes[x, y].start = false;
            nodes[x, y].end = false;
            nodes[x, y].chosen = false;
            nodes[x, y].g = 0;
            nodes[x, y].h = 0;
            nodes[x, y].f = 0;
            nodes[x, y].Invalidate(); //los nodos se vuelvan a dibujar en la interfaz

            window.selected = null;
            window.start = null;
            window.end = null;
            window.path = new List<Node>();
        }
    }
}
```

```
public static void FindPath(Form1 window, Node[,] nodes, Node start, Node end) //busqueda de rutas
{
    List<Node> open = new List<Node>() { start };
    List<Node> closed = new List<Node>();
    StringBuilder sb = new StringBuilder();// almacena el recorrido

    do
    {
        Node q = open.MinF();

        open.Remove(q);
        closed.Add(q);

        int x = q.GetLocation(nodes)[0];
        int y = q.GetLocation(nodes)[1];

        if (nodes.IsValid(x - 1, y)) //Obtiene las coordenadas "x" e "y" del nodo actual.
        {
            var s = nodes[x - 1, y]; //Si es posible, obtiene el nodo adyacente "s" a la izquierda.

            if (s.Equals(end))
            {
                ShowPath(window, closed, nodes, end); //Si el nodo adyacente "s" es igual al nodo objetivo "end", muestra el camino
                Form1.a++; //aumenta el contador
                return;
            }
```

```
            if (!closed.Contains(s) && !s.blocked) //comprueba si el nodo ha sido visitado antes y si no esta bloqueado
            {
                s.g = q.g + s.GetDistance(q, nodes);
                s.h = s.GetDistance(end,  nodes);
                s.f = s.g + s.h;

                open.Add(s);//lista de nodos abiertos
                Form1.d = s.f;//Agrega el nodo de inicio a la lista de nodos abiertos
            }
            string recorrido = string.Format("({0};{1})", x, y);
            sb.Append(recorrido);

            string caminos = sb.ToString();//
            Form1.c = caminos;

        }
```

```
150             if (nodes.IsValid(x + 1, y))
151             {
152                 var s = nodes[x + 1, y];
153
154                 if (s.Equals(end))
155                 {
156                     ShowPath(window, closed, nodes, end);
157                     Form1.a++;
158                     return;
159                 }
160
161                 if (!closed.Contains(s) && !s.blocked)
162                 {
163                     s.g = q.g + s.GetDistance(q, nodes);
164                     s.h = s.GetDistance(end, nodes);
165                     s.f = s.g + s.h;
166
167                     open.Add(s);
168                     Form1.d = s.f;
169                 }
170
171                 string caminos = sb.ToString();
172                 Form1.c = caminos;//
173             }
174
175             if (nodes.IsValid(x, y + 1))
176             {
177                 var s = nodes[x, y + 1];
178
179                 if (s.Equals(end))
180                 {
181                     ShowPath(window, closed, nodes, end);
```

```
199
200             if (nodes.IsValid(x, y - 1))
201             {
202                 var s = nodes[x, y - 1];
203
204                 if (s.Equals(end))
205                 {
206                     ShowPath(window, closed, nodes, end);
207                     Form1.a++;
208                     return;
209                 }
210
211                 if (!closed.Contains(s) && !s.blocked)
212                 {
213                     s.g = q.g + s.GetDistance(q, nodes);
214                     s.h = s.GetDistance(end,  nodes);
215                     s.f = s.g + s.h;
216
217                     open.Add(s);
218                     Form1.d= s.f;//
219                 }
220
221
222                 string caminos = sb.ToString();
223                 Form1.c = caminos;//
224             }
225
226             if (nodes.IsValid(x - 1, y + 1))
227             {
228                 var s = nodes[x - 1, y + 1];
229
230                 if (s.Equals(end))
231                 {
232                     Form1.a++;
```

```
298                 Form1.c = caminos;//
299             }
300
301             if (nodes.IsValid(x + 1, y - 1))
302             {
303                 var s = nodes[x + 1, y - 1];
304
305                 if (s.Equals(end))
306                 {
307                     ShowPath(window, closed, nodes, end);
308                     Form1.a++;
309                     return;
310                 }
311
312                 if (!closed.Contains(s) && !s.blocked)
313                 {
314                     s.g = q.g + s.GetDistance(q, nodes);
315                     s.h = s.GetDistance(end,  nodes);
316                     s.f = s.g + s.h;
317
318                     open.Add(s);
319                     Form1.d = s.f;//
320                 }
321
322
323                 string caminos = sb.ToString();
324                 Form1.c = caminos;//
325             }
326         }
327         while (open.Count != 0); // verifica el tamaño de la coleccion abierta que no sea igual a 0
328     }
329
330
331
```

- **Luego de a ver visualizado en codigo se muestra la interfaz gráfica completa**

- El usuario deberá de ingresar el inicio y el final tambien deberá seleccionar los cuadros que desee para obstáculos, por último el sistema deberá buscar el camino mas cercano para poder llegar a su destino.