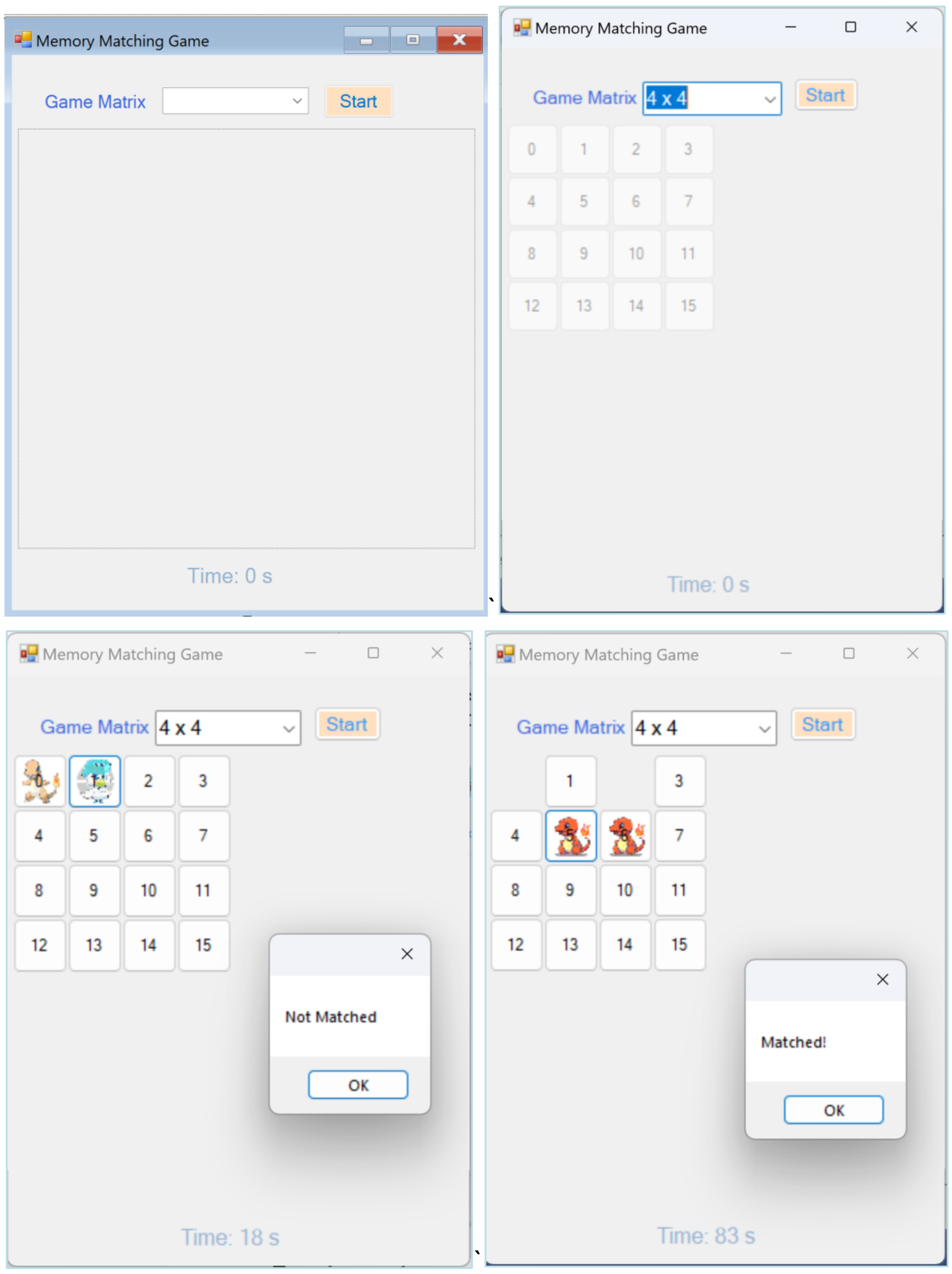
Form 7.

Program Description: Memory Matching Game

Objective:

Write a C# Windows Forms application that implements a memory-matching game. The game consists of an n x n grid of buttons, each hiding a pair of images. Players uncover the images two at a time, attempting to find matching pairs.

GUI of Game:



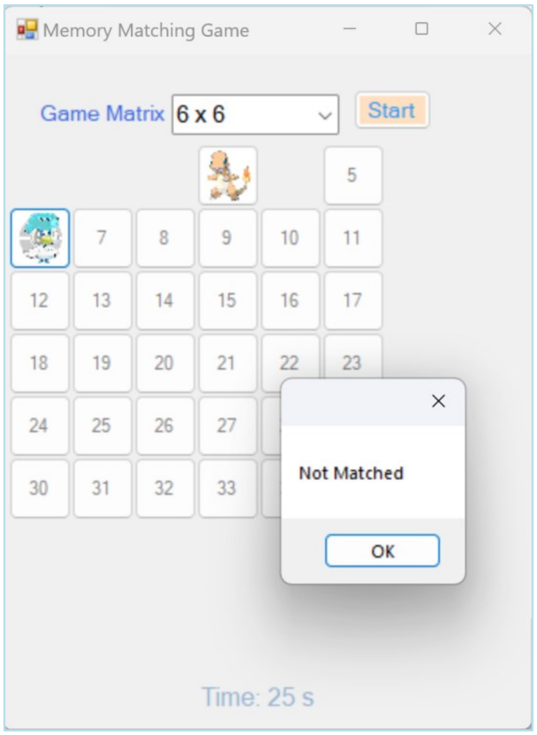
Time: 0 s

Time: 18 s

Not Matched

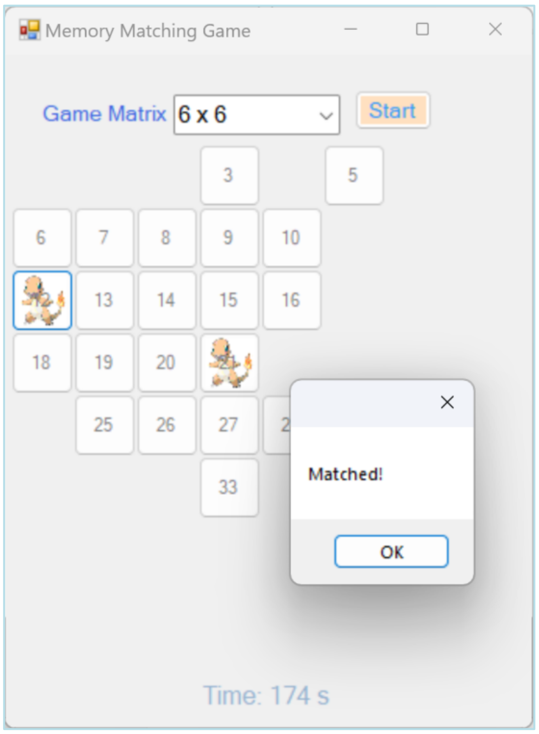
Time: 83 s

Matched!



Not Matched

Time: 25 s



Matched!

Time: 174 s

Requirements:

1. **Generate Game Array (Pairs Generation):**
   - Write a function GenerateArray(int n) that creates an array of n * n elements, ensuring that n is even.
   - Populate the array with pairs of numbers ranging from 0 to n - 1.
   - Randomize the pairs' positions to ensure a shuffled layout.
2. **Button Matrix Setup:**
   - Create an n x n grid of buttons.
   - Each button should initially appear blank and reveal an image or number upon clicking.
   - Assign each button an image corresponding to its randomly assigned pair value.
3. **Button Click Event:**
   - Handle the button click event to reveal the hidden image.
   - Track the clicked buttons in a list.
   - When two buttons are clicked:
     - Check if they match. If so, hide both buttons and increment the match count.
     - If they do not match, reset the buttons after a brief delay by hiding their images.
   - Ensure the same button cannot be clicked twice in succession.
4. **Endgame Logic:**
   - Track matched pairs.
   - Once all pairs are matched, stop the game timer and display a message with the total time elapsed.

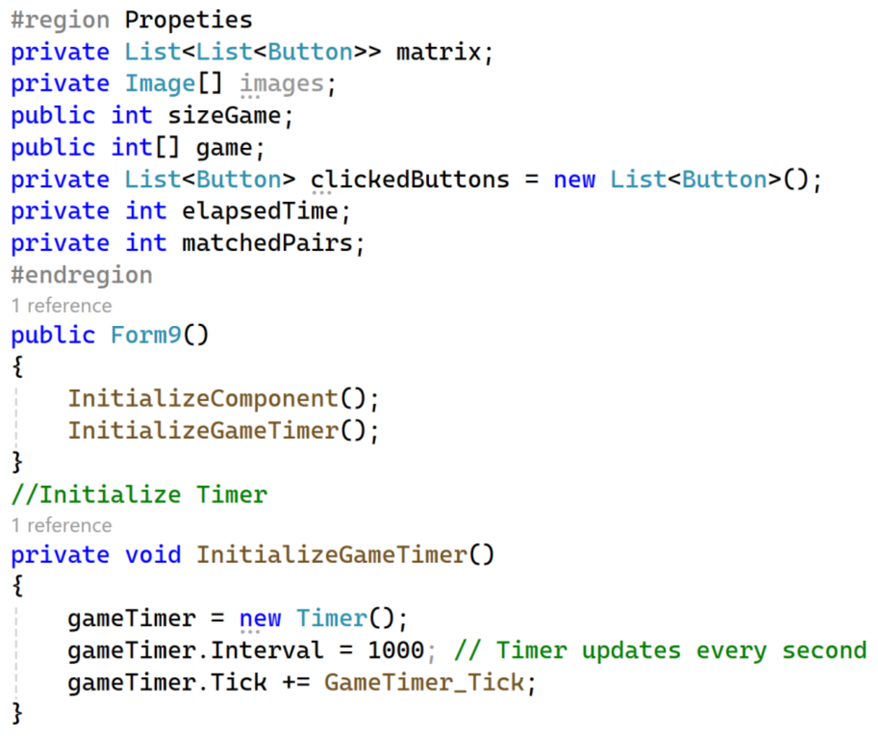
5. **Timer Functionality:**
    - Implement a timer to track the duration of the game, displaying elapsed time to the player.
    - Start the timer when the game begins and stop it upon completion.
6. **UI and Controls:**
    - Add a dropdown (ComboBox) for selecting grid size (4 x 4, 6 x 6, 8 x 8).
    - Add a Start button that initializes the game, resets the timer, and enables the grid.
    - Display a timer label to show elapsed seconds.

Gameplay Flow:

1. Select the grid size from the dropdown.
2. Click Start to begin the game.
3. Click on buttons to reveal hidden images.
4. Match pairs until all images are revealed.
5. View a message with total time once the game is completed.

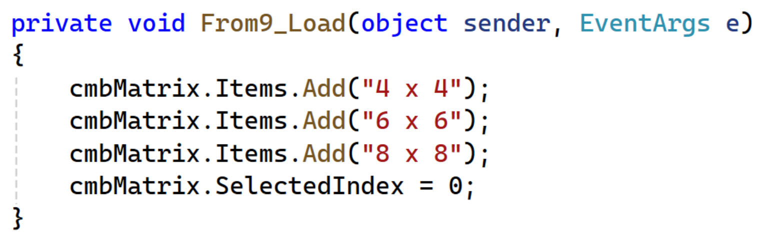
Code:

Step 0. Declare and initialize global variables



```
#region Propeties
private List<List<Button>> matrix;
private Image[] images;
public int sizeGame;
public int[] game;
private List<Button> clickedButtons = new List<Button>();
private int elapsedTime;
private int matchedPairs;
#endregion
1 reference
public Form9()
{
    InitializeComponent();
    InitializeGameTimer();
}
//Initialize Timer
1 reference
private void InitializeGameTimer()
{
    gameTimer = new Timer();
    gameTimer.Interval = 1000; // Timer updates every second
    gameTimer.Tick += GameTimer_Tick;
}
```

Step 1. Set up matrix types for combobox



```
private void From9_Load(object sender, EventArgs e)
{
    cmbMatrix.Items.Add("4 x 4");
    cmbMatrix.Items.Add("6 x 6");
    cmbMatrix.Items.Add("8 x 8");
    cmbMatrix.SelectedIndex = 0;
}
```

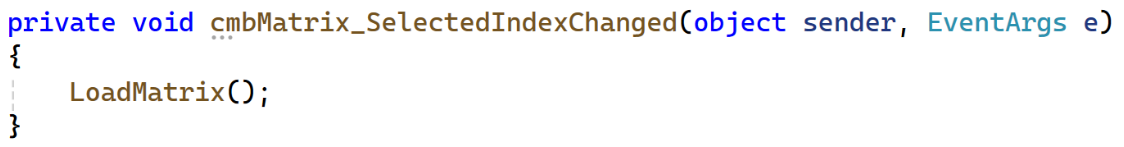
Step 2. Create an array containing pairs of similar shapes corresponding to the matrix of buttons

```
public static int[] GenerateArray(int n)
{
    int size = n * n; //size of array
    int[] array = new int[size];
    Random random = new Random();
    // Create a list of random number pairs from 0 to n - 1.
    int[] pairs = new int[size];
    for (int i = 0; i < size; i += 2) {
        int randomValue = random.Next(0, n);
        pairs[i] = randomValue;
        pairs[i + 1] = randomValue;
    }
    // Shuffle the positions of elements in the pairs array.
    pairs = pairs.OrderBy(x => random.Next()).ToArray();
    // Copy the shuffled pairs into the result array.
    for (int i = 0; i < size; i++) {
        array[i] = pairs[i];
    }
    return array;
}
```

Step 3. Initialize the game matrix

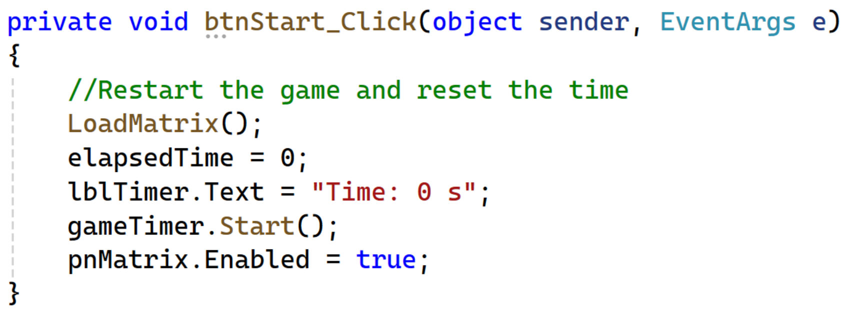
```
void LoadMatrix()
{
    pnMatrix.Controls.Clear();
    matchedPairs = 0; // set of match pair value at start game.
    string[] img = { "0.png", "1.png", "2.png","3.png","4.png","5.png","6.png","7.png" };
    imageList1.ImageSize = new Size(40, 40);
    var image = imageList1.Images;
    foreach(var item in img) {
        image.Add(Image.FromFile("d:\\" + item));
    }
    //return value from selected index of combobox
    sizeGame = cmbMatrix.SelectedIndex == 0 ? 4 : cmbMatrix.SelectedIndex == 1 ? 6 : 8;
    game = GenerateArray(sizeGame); //create random value for game position

    matrix = new List<List<Button>>(); //create button matrix and add to panel (pmMatrix)
    Button x = new Button() { Width = 0, Height = 0, Location = new Point(0, 0) };

    for (int i = 0; i < sizeGame; i++)
    {
        matrix.Add(new List<Button>());
        for (int j = 0; j < sizeGame; j++)
        {
            Button btn = new Button() { Width = 40, Height = 40};
            btn.Location = new Point(x.Location.X + x.Width, x.Location.Y);
            btn.Text = (i * sizeGame + j).ToString(); //set text
            btn.ForeColor = Color.Gray;
            btn.Click += btn_Click; //add click event
            pnMatrix.Controls.Add(btn); //add button to panel
            x = btn;
        }
        x = new Button(){ Width = 0,Height = 0,Location = new Point(0, x.Location.Y + 40)};
    }
    pnMatrix.Enabled = false;
}
```

Step 4. Game matrix changes when combobox value changes



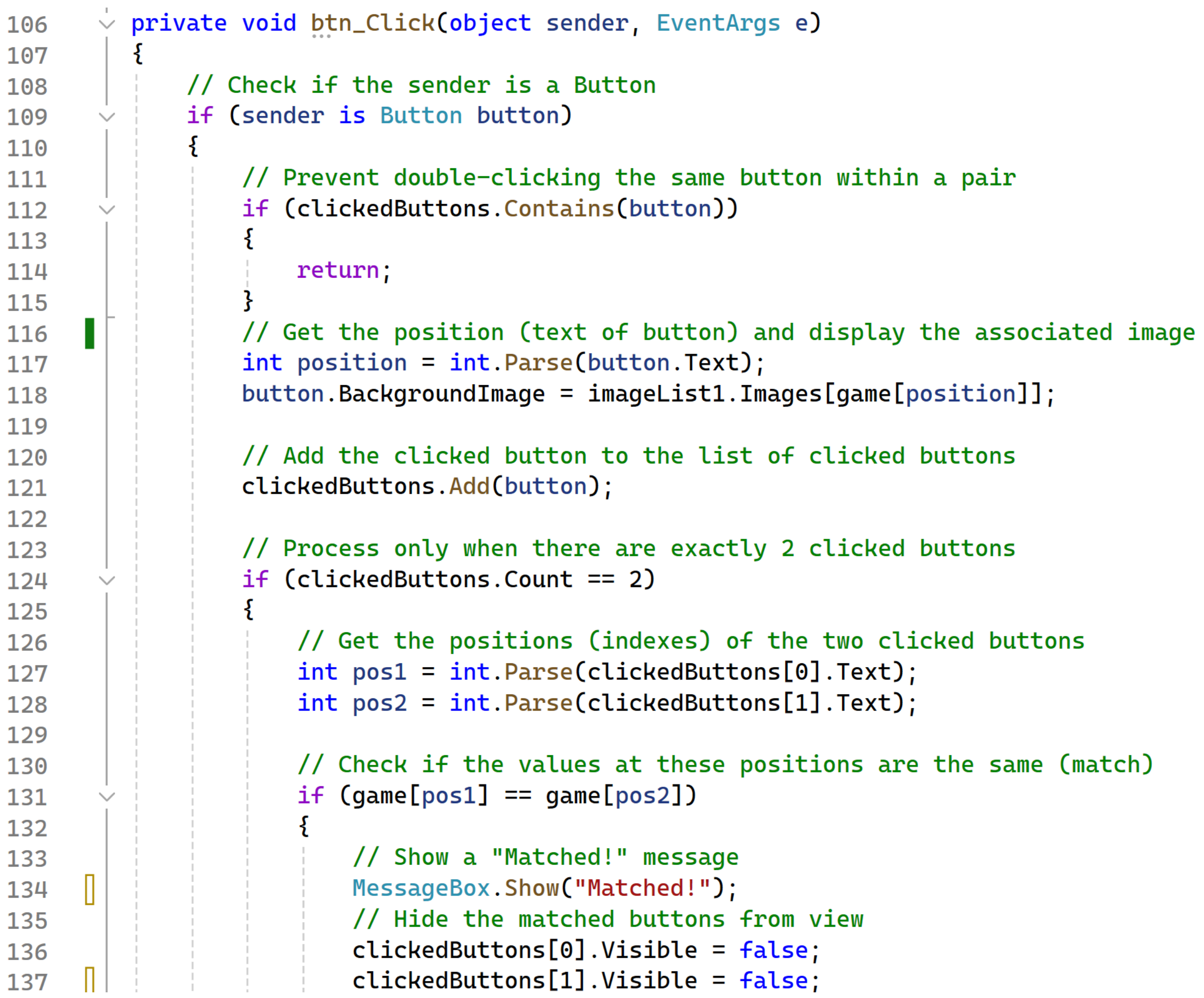
```
private void cmbMatrix_SelectedIndexChanged(object sender, EventArgs e)
{
    LoadMatrix();
}
```

Step 5. Initial setup for the game



```
private void btnStart_Click(object sender, EventArgs e)
{
    //Restart the game and reset the time
    LoadMatrix();
    elapsedTime = 0;
    lblTimer.Text = "Time: 0 s";
    gameTimer.Start();
    pnMatrix.Enabled = true;
}
```

Step 6. Process commands during game play according to game requirements



```
private void btn_Click(object sender, EventArgs e)
{
    // Check if the sender is a Button
    if (sender is Button button)
    {
        // Prevent double-clicking the same button within a pair
        if (clickedButtons.Contains(button))
        {
            return;
        }
        // Get the position (text of button) and display the associated image
        int position = int.Parse(button.Text);
        button.BackgroundImage = imageList1.Images[game[position]];

        // Add the clicked button to the list of clicked buttons
        clickedButtons.Add(button);

        // Process only when there are exactly 2 clicked buttons
        if (clickedButtons.Count == 2)
        {
            // Get the positions (indexes) of the two clicked buttons
            int pos1 = int.Parse(clickedButtons[0].Text);
            int pos2 = int.Parse(clickedButtons[1].Text);

            // Check if the values at these positions are the same (match)
            if (game[pos1] == game[pos2])
            {
                // Show a "Matched!" message
                MessageBox.Show("Matched!");
                // Hide the matched buttons from view
                clickedButtons[0].Visible = false;
                clickedButtons[1].Visible = false;
```

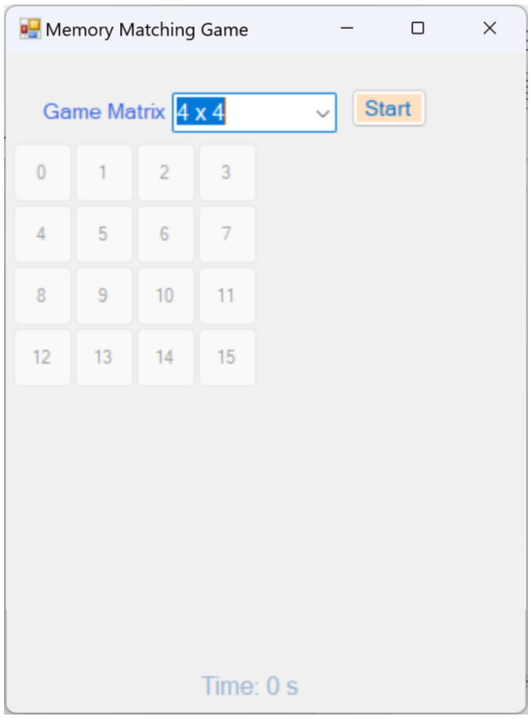
```
                    // Increment the count of matched pairs
                    matchedPairs++;
                    // Check if all pairs have been matched
                    if (matchedPairs == (sizeGame * sizeGame) / 2)
                    {
                        // Stop the game timer and show completion message with time elapsed
                        gameTimer.Stop();
                        MessageBox.Show($"Congratulations! You have completed in {elapsedTime} seconds.");
                    }
                }
                else
                {
                    // Show "Not Matched" message for mismatched pair
                    MessageBox.Show("Not Matched");
                    // Remove the background image of the mismatched buttons to hide them again
                    clickedButtons[0].BackgroundImage = null;
                    clickedButtons[1].BackgroundImage = null;
                }
                // Clear the list of clicked buttons to prepare for the next round of clicks
                clickedButtons.Clear();
            }
        }
    }
```

Step 7. Set up Timer during game play

```
private void GameTimer_Tick(object sender, EventArgs e)
{
    elapsedTime++;
    lblTimer.Text = $"Time: {elapsedTime} s";
}
```

Step 8. Debug > Start Debugging

The game interface starts as shown below:



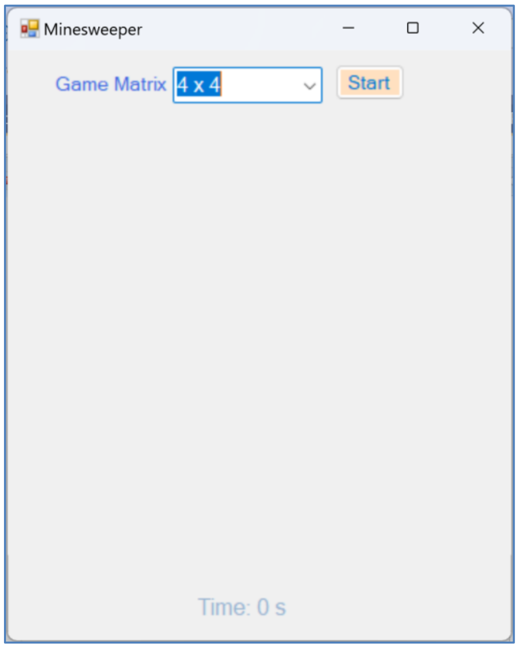
Form 8.

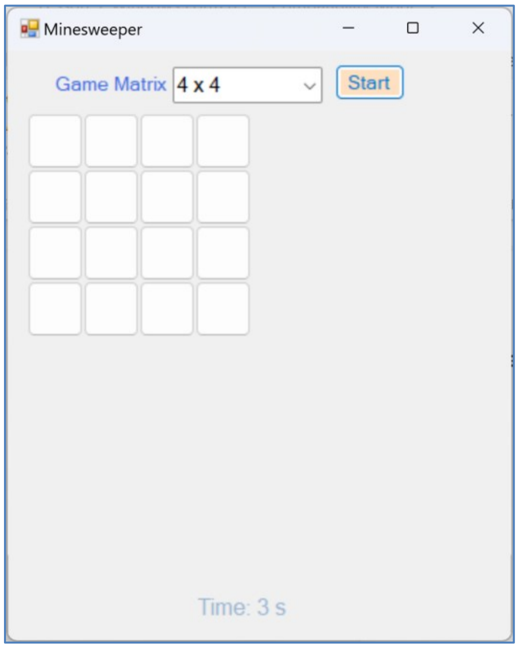
Simple Minesweeper Game Requirements

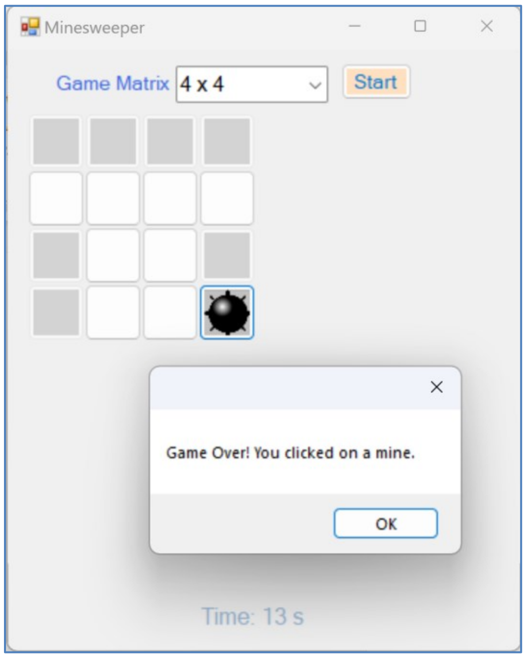
This is a simple Minesweeper game where the player clicks buttons to avoid randomly placed mines. The game includes the following requirements:

1. **Matrix Size and Mines**:
    - The game offers three matrix sizes: 4x4, 6x6, and 8x8.
    - Each matrix contains a set number of randomly placed mines:
        - 4x4 grid: 4 mines
        - 6x6 grid: 6 mines
        - 8x8 grid: 8 mines
    - The mine positions are generated randomly when the game starts.
2. **Gameplay**:
    - Players click on individual buttons to reveal what is underneath.
    - If the clicked button contains a mine, the game immediately ends, displaying a "Game Over" message.
    - If the clicked button is safe (does not contain a mine), the button remains visible or displays an indicator (such as a number of nearby mines, if desired for an advanced version).
    - The game continues until either all safe buttons are clicked or a mine is clicked.
3. **Game End Conditions**:
    - If a mine is clicked, the game ends with a "Game Over" message.
    - If the player successfully avoids all mines and clicks all safe buttons, a "Congratulations" message is displayed.
4. **Additional Requirements (Optional)**:
    - The elapsed time can be displayed to track how long the player takes to complete the game.
    - The player can restart the game, which will reset the board and generate new random mine positions.

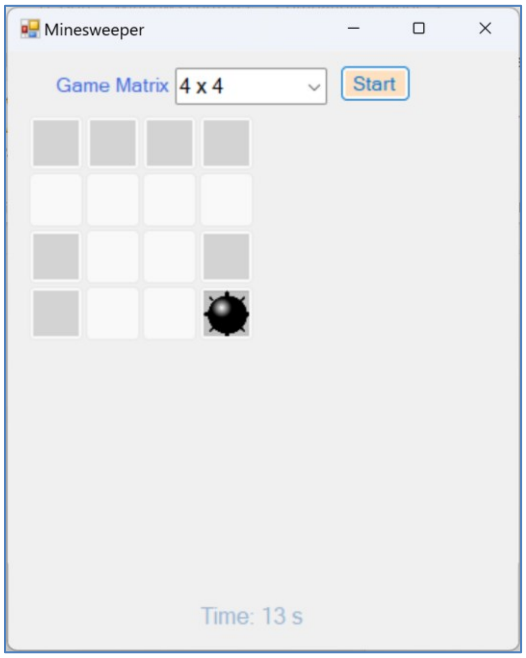
Expected Result:







Game Over! You clicked on a mine.


Minesweeper
Game Matrix
4 x 4
Start
Time: 13 s