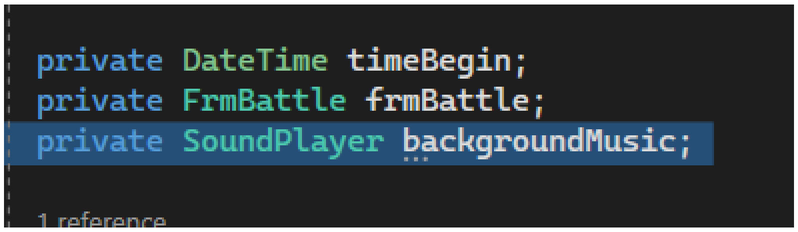
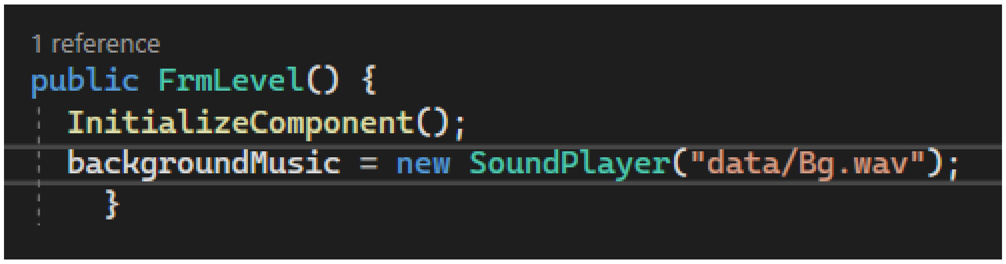
Nabinta Niraula
Background music

frmLevel.cs:
The background music is integrated using the '*SoundPlayer*' class from the '*System.Media'* namespace.



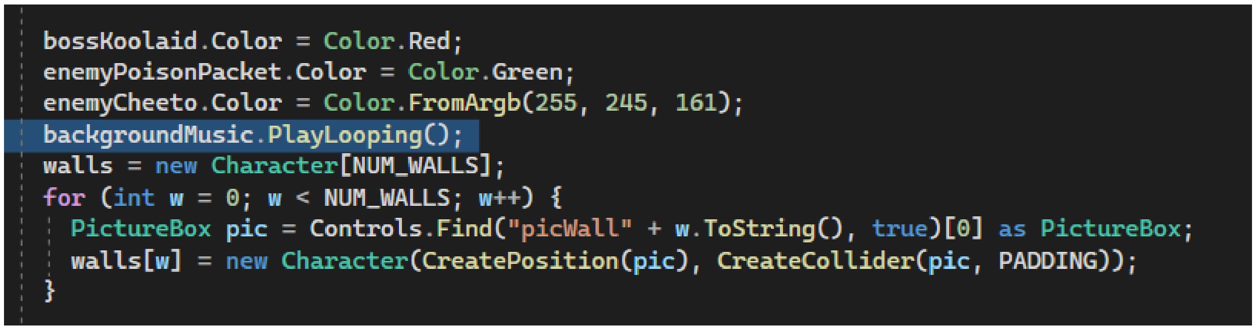
```
private DateTime timeBegin;
private FrmBattle frmBattle;
private SoundPlayer backgroundMusic;

1 reference
```

And the game initializes a '*SoundPlayer*' object with the background music file located at '*data/Bg.wav*' which is within the 'FrmLevel' class. As soon as an instance of the '*FrmLevel*' form is created, the background music plays.



```
1 reference
public FrmLevel() {
  InitializeComponent();
  backgroundMusic = new SoundPlayer("data/Bg.wav");
    }
```

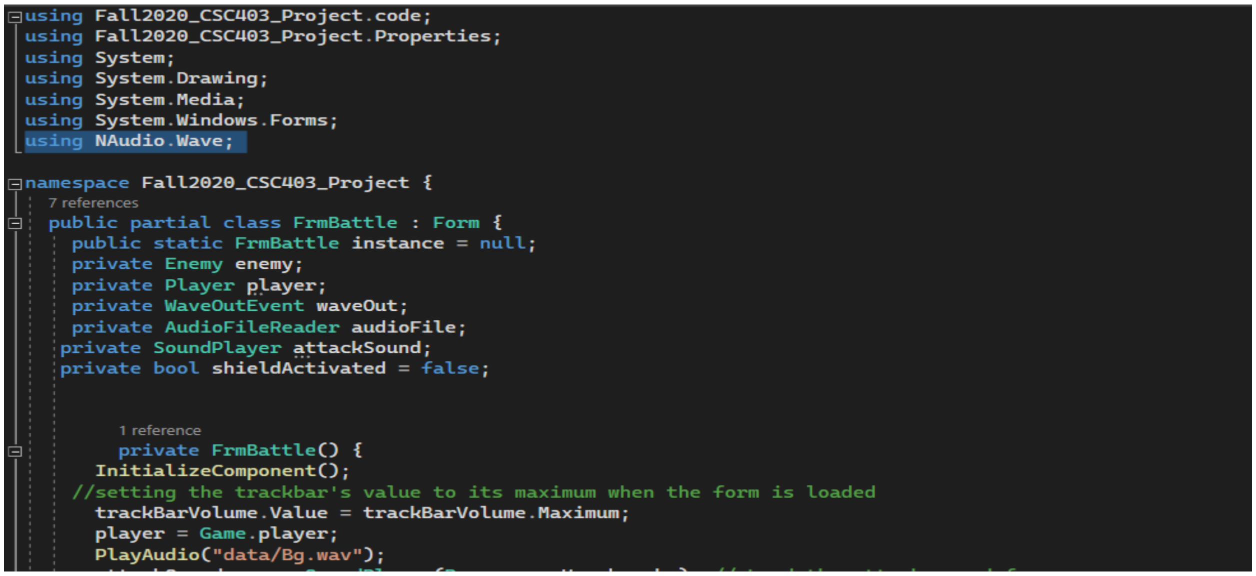
The *backgroundMusic.PlayLooping()* loops the music continuously once started.



```
bossKoolaid.Color = Color.Red;
enemyPoisonPacket.Color = Color.Green;
enemyCheeto.Color = Color.FromArgb(255, 245, 161);
backgroundMusic.PlayLooping();
walls = new Character[NUM_WALLS];
for (int w = 0; w < NUM_WALLS; w++) {
  PictureBox pic = Controls.Find("picWall" + w.ToString(), true)[0] as PictureBox;
  walls[w] = new Character(CreatePosition(pic), CreateCollider(pic, PADDING));
}
```

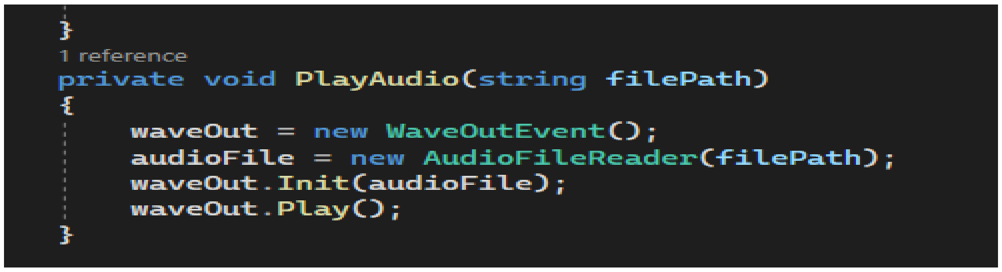
frmBattle.cs:

Here it utilizes ‘*NAudio.Wave*’ for audio playback. The background music is managed through a method named ‘ *PlayAudio("data/Bg.wav")*’ which plays the background music.



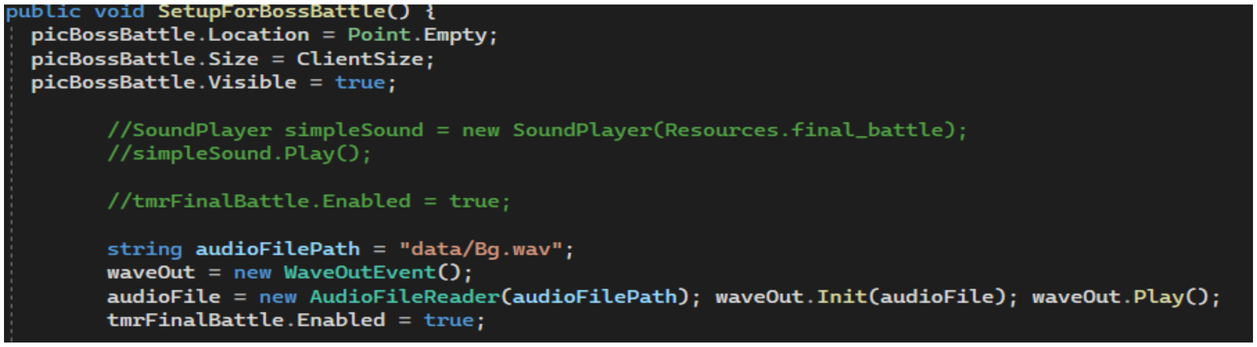
```
using Fall2020_CSC403_Project.code;
using Fall2020_CSC403_Project.Properties;
using System;
using System.Drawing;
using System.Media;
using System.Windows.Forms;
using NAudio.Wave;

namespace Fall2020_CSC403_Project {
  7 references
  public partial class FrmBattle : Form {
    public static FrmBattle instance = null;
    private Enemy enemy;
    private Player player;
    private WaveOutEvent waveOut;
    private AudioFileReader audioFile;
   private SoundPlayer attackSound;
   private bool shieldActivated = false;


        1 reference
        private FrmBattle() {
      InitializeComponent();
    //setting the trackbar's value to its maximum when the form is loaded
      trackBarVolume.Value = trackBarVolume.Maximum;
      player = Game.player;
      PlayAudio("data/Bg.wav");
```

Similarly, the *PlayAudio(string filePath)* method plays the audio files. It takes a file path as an input and uses the *‘NAudio.Wave*’ library for audio operations. First, the method initializes a ‘*WaveOutEvent*’ object, named ‘*waveOut*’, which sends the audio to the sound card. Then, it creates an ‘*AudioFileReader*’ object, ‘*audioFile*’, that reads the specified audio file from file path. Then the ‘*waveOu*t’ object is initialized with this audio file using the ‘*Init*’ method. Finally, the ‘Play’ method of ‘*waveOut*’ is called, which starts the audio playback.



```
}
1 reference
private void PlayAudio(string filePath)
{
    waveOut = new WaveOutEvent();
    audioFile = new AudioFileReader(filePath);
    waveOut.Init(audioFile);
    waveOut.Play();
}
```

This part of the code ensures the background music is played for the boss battle scene.

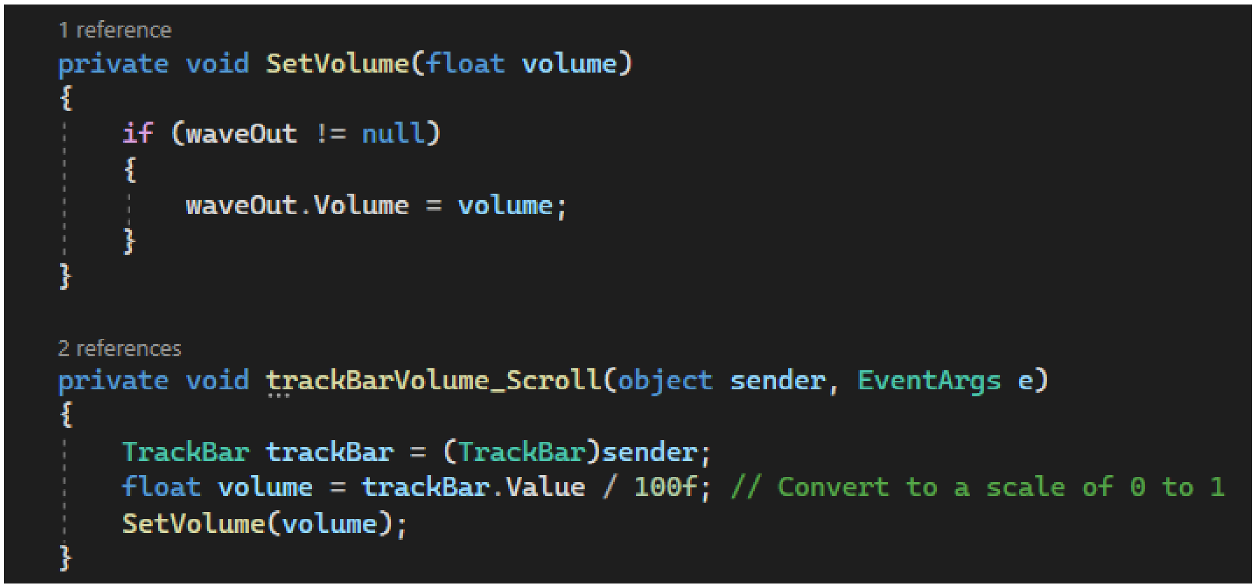


```
public void SetupForBossBattle() {
    picBossBattle.Location = Point.Empty;
    picBossBattle.Size = ClientSize;
    picBossBattle.Visible = true;

        //SoundPlayer simpleSound = new SoundPlayer(Resources.final_battle);
        //simpleSound.Play();

        //tmrFinalBattle.Enabled = true;

        string audioFilePath = "data/Bg.wav";
        waveOut = new WaveOutEvent();
        audioFile = new AudioFileReader(audioFilePath); waveOut.Init(audioFile); waveOut.Play();
        tmrFinalBattle.Enabled = true;
```

Volume TrackBar Slider:

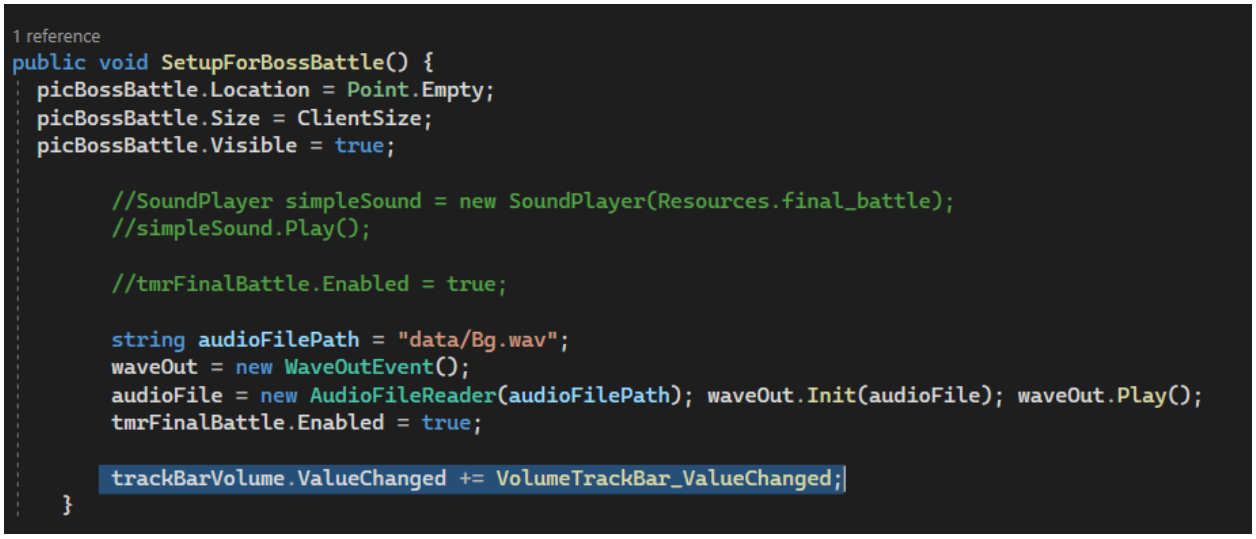
frmBattle.cs:

Here the *setVolume* method sets the volume of an audio playback. If *'waveOut'* is initialized, it sets its volume to the specified value. The volume is a float value between 0 being mute and 1 being the maximum volume. Similarly the *trackBarVolume_Scroll* event handler responds to the scrolling of a trackbar.



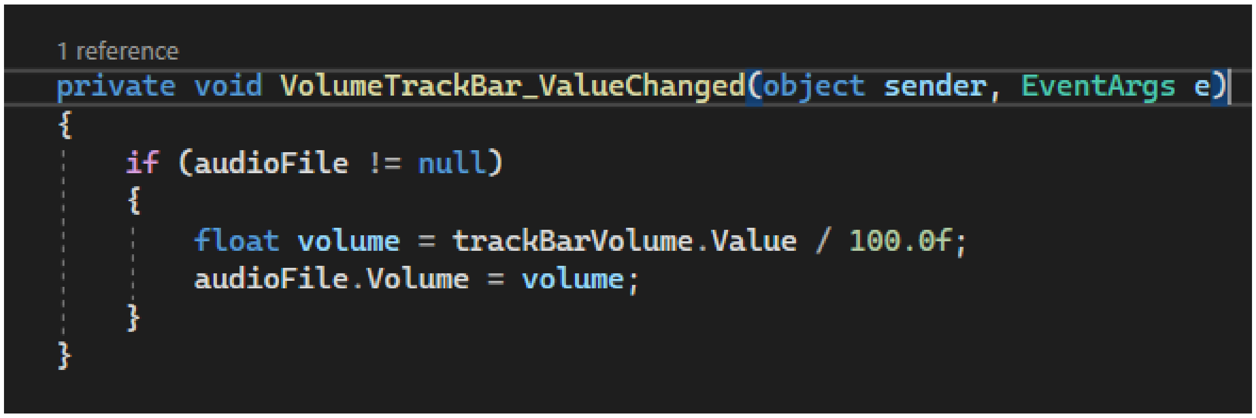
```
1 reference
private void SetVolume(float volume)
{
    if (waveOut != null)
    {
        waveOut.Volume = volume;
    }
}

2 references
private void trackBarVolume_Scroll(object sender, EventArgs e)
{
    TrackBar trackBar = (TrackBar)sender;
    float volume = trackBar.Value / 100f; // Convert to a scale of 0 to 1
    SetVolume(volume);
}
```

The " *trackBarVolume.ValueChanged += VolumeTrackBar_ValueChanged*" in the *setupForBossBattle* method attached an event handler to the '*ValueChanged'* event of '*trackBarVolume*', linking it to '*VolumeTrackBar_ValueChanged*'.



```
1 reference
public void SetupForBossBattle() {
  picBossBattle.Location = Point.Empty;
  picBossBattle.Size = ClientSize;
  picBossBattle.Visible = true;

        //SoundPlayer simpleSound = new SoundPlayer(Resources.final_battle);
        //simpleSound.Play();

        //tmrFinalBattle.Enabled = true;

        string audioFilePath = "data/Bg.wav";
        waveOut = new WaveOutEvent();
        audioFile = new AudioFileReader(audioFilePath); waveOut.Init(audioFile); waveOut.Play();
        tmrFinalBattle.Enabled = true;

        trackBarVolume.ValueChanged += VolumeTrackBar_ValueChanged;
    }
```

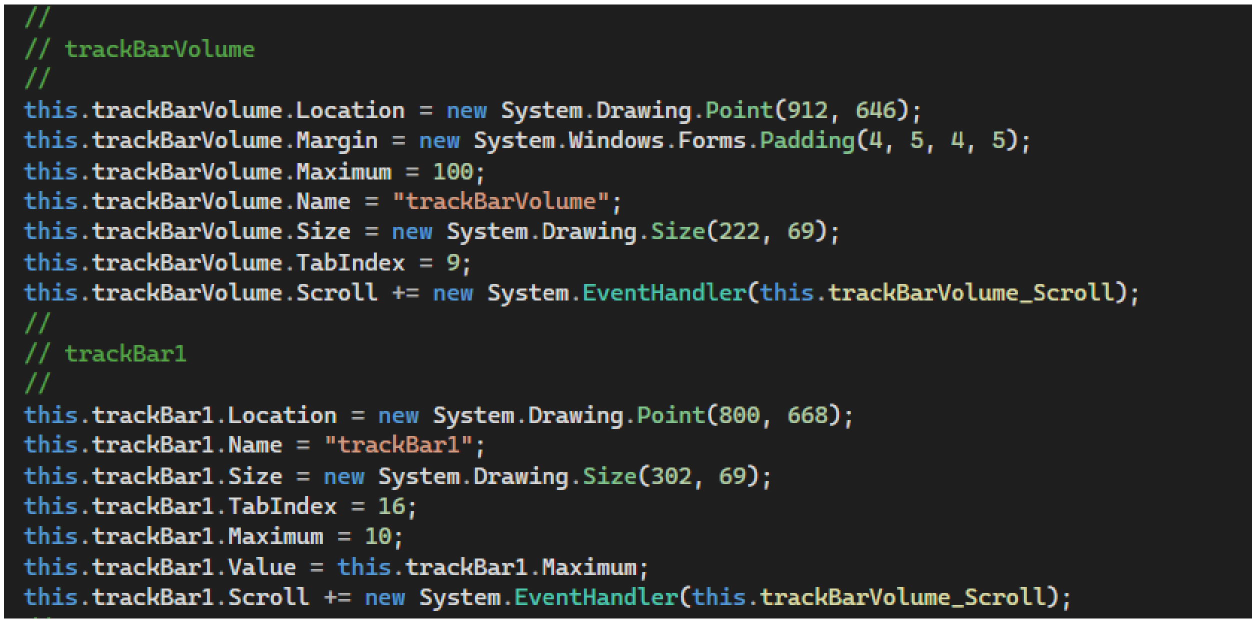
The *volumeTrackBar_ValueChanged* method is an event handler for changes in the volume trackbars' value. This method adjusts the volume of the audio playback based on the value of the trackbar.



```
1 reference
private void VolumeTrackBar_ValueChanged(object sender, EventArgs e)
{
    if (audioFile != null)
    {
        float volume = trackBarVolume.Value / 100.0f;
        audioFile.Volume = volume;
    }
}
```

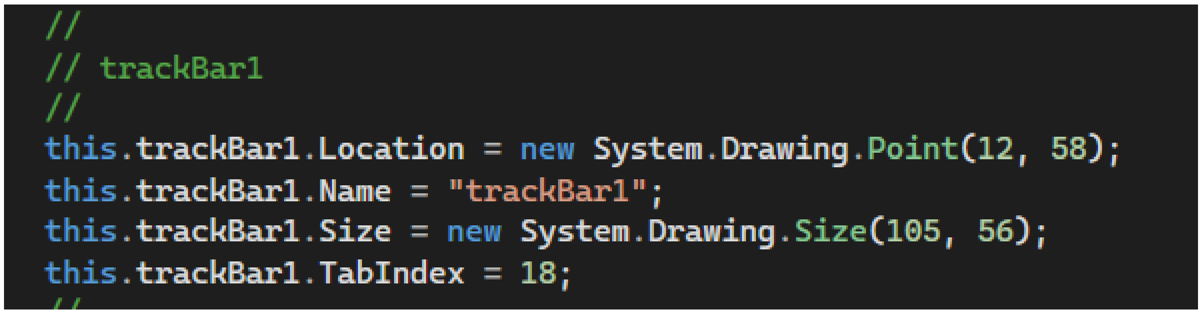
frmBattle.Designer.cs:

The '*trackBarVolume*' and *'trackBar1'* are implemented as a UI component of a trackbar.



```
//
// trackBarVolume
//
this.trackBarVolume.Location = new System.Drawing.Point(912, 646);
this.trackBarVolume.Margin = new System.Windows.Forms.Padding(4, 5, 4, 5);
this.trackBarVolume.Maximum = 100;
this.trackBarVolume.Name = "trackBarVolume";
this.trackBarVolume.Size = new System.Drawing.Size(222, 69);
this.trackBarVolume.TabIndex = 9;
this.trackBarVolume.Scroll += new System.EventHandler(this.trackBarVolume_Scroll);
//
// trackBar1
//
this.trackBar1.Location = new System.Drawing.Point(800, 668);
this.trackBar1.Name = "trackBar1";
this.trackBar1.Size = new System.Drawing.Size(302, 69);
this.trackBar1.TabIndex = 16;
this.trackBar1.Maximum = 10;
this.trackBar1.Value = this.trackBar1.Maximum;
this.trackBar1.Scroll += new System.EventHandler(this.trackBarVolume_Scroll);
```

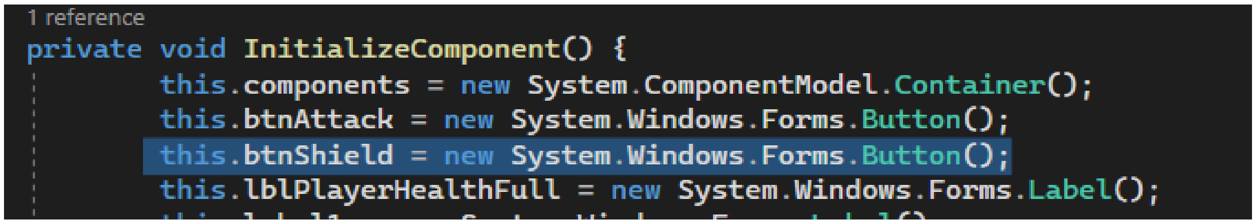
frmLevelDesigner.cs:
It's the same as above. Where the trackbar is positioned at the coordinates *(12,58)* within the form. And the size is set to *(105, 56*) which determines the width and height of the trackbar.



```
//
// trackBar1
//
this.trackBar1.Location = new System.Drawing.Point(12, 58);
this.trackBar1.Name = "trackBar1";
this.trackBar1.Size = new System.Drawing.Size(105, 56);
this.trackBar1.TabIndex = 18;
```

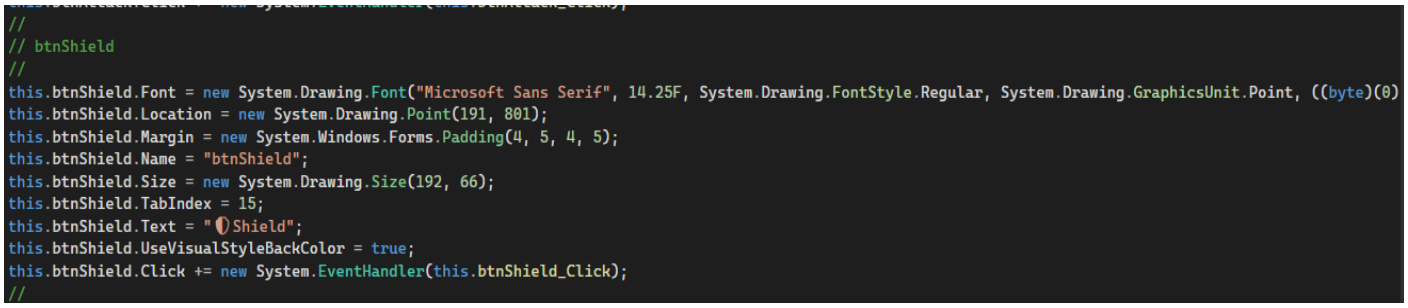
Shield Button

frmBattle.Designer.cs:
The shield button is implemented . “*this.btnShield = new System.Windows.Forms.Button();”* this line creates an instance of a button control, naming it *‘btnShield’*.



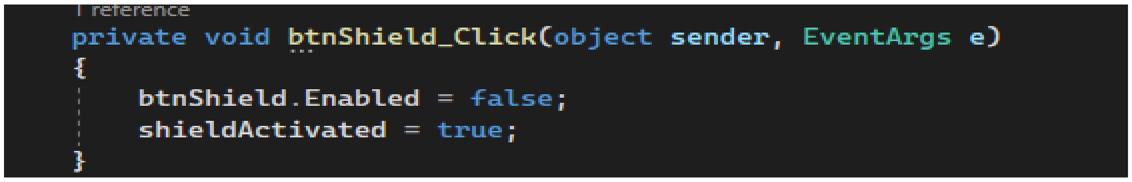
```
1 reference
private void InitializeComponent() {
        this.components = new System.ComponentModel.Container();
        this.btnAttack = new System.Windows.Forms.Button();
        this.btnShield = new System.Windows.Forms.Button();
        this.lblPlayerHealthFull = new System.Windows.Forms.Label();
```

btnShield is the basic implementation of the size and position of the button itself.



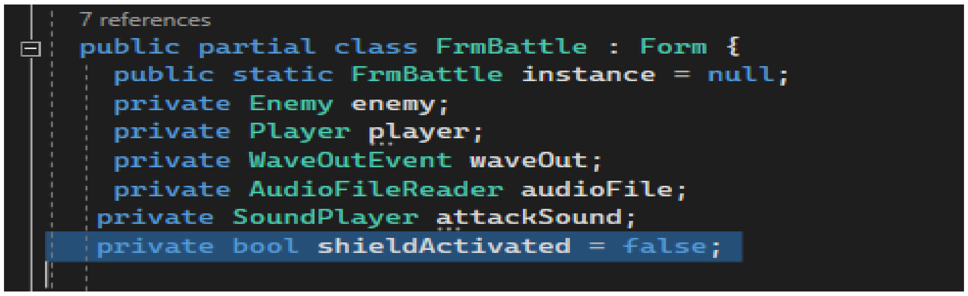
```
//
// btnShield
//
this.btnShield.Font = new System.Drawing.Font("Microsoft Sans Serif", 14.25F, System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((byte)(0)
this.btnShield.Location = new System.Drawing.Point(191, 801);
this.btnShield.Margin = new System.Windows.Forms.Padding(4, 5, 4, 5);
this.btnShield.Name = "btnShield";
this.btnShield.Size = new System.Drawing.Size(192, 66);
this.btnShield.TabIndex = 15;
this.btnShield.Text = "🛡Shield";
this.btnShield.UseVisualStyleBackColor = true;
this.btnShield.Click += new System.EventHandler(this.btnShield_Click);
//
```

frmBattle.cs:
The *‘btnShield_Click’* is an event handler method that is triggered when the shield button *‘btnShield’* is clicked. The line *‘ btnShield.Enabled = false’* disables the shield button immediately after it’s clicked. Then the line *‘shieldActivated = true’* indicates the shield is now active.



```
1 reference
private void btnShield_Click(object sender, EventArgs e)
{
    btnShield.Enabled = false;
    shieldActivated = true;
}
```

The *‘private bool shieldActivated = false’* is a boolean variable declaration which indicates the boolean is not active initially.



```
7 references
public partial class FrmBattle : Form {
  public static FrmBattle instance = null;
  private Enemy enemy;
  private Player player;
  private WaveOutEvent waveOut;
  private AudioFileReader audioFile;
 private SoundPlayer attackSound;
 private bool shieldActivated = false;
```