Implementation

## **1. Animate.cs**

using System;

using System.Drawing;

using System.Resources;

namespace Uni\_Form\_Trans\_Test\_01

{

public class Animate

{

#region Declarations

private string name;

private int length;

private Canvas canvas;

private int delay, idx;

private Point[,] eyeLocations;

private string[] fileNames;

static public float rescaleFactor = 1f;

#endregion

public Animate(string name, int width, int length, int delay, float rescaleFactor, ref Canvas canvas, ref ResourceManager rm)

{

this.name = name;

this.length = length;

this.canvas = canvas;

this.delay = delay;

idx = 0;

eyeLocations = new Point[length, 2];

fileNames = new string[length];

Animate.SetScale(rescaleFactor);

for (int i = 0; i < length; i++)

{

string fileName = name;

int l = (i.ToString()).Length;

for (int j = 0; j < width - l; j++)

fileName += '0';

fileName += i.ToString();

LoadBitmap2(fileName, ref fileNames[i]);

}

//Read Location Here

string fn = "Locations/" + name;

string loc = System.IO.File.ReadAllText(fn);

string temp = "";

int flag = 0;

float p = 0;

int spCount = 0;

string xc = "";

string yc = "";

for (int i = 0; i < loc.Length; i++)

{

if (loc[i] != ' ')

{

temp += loc[i];

}

else if (loc[i] == ' ')

{

spCount++;

if (spCount == 1)

{

xc = temp;

}

if (spCount == 2)

{

yc = temp;

spCount = 0;

eyeLocations[(int)p % length, flag].X = int.Parse(xc);

eyeLocations[(int)p % length, flag].Y = int.Parse(yc);

p = p + 0.5f;

flag = 1 - flag;

}

temp = "";

}

}

}

public void SetDelay(int delay)

{

this.delay = delay;

}

public bool Update(long cnt, int x, int y)

{

// Returns True If Current Animation Is Over

idx %= length;

if (cnt % delay == 0)

{

Bitmap tmp = (Bitmap)Bitmap.FromFile(fileNames[idx]);

Bitmap tmp2 = new Bitmap(tmp, new Size((int)(tmp.Width / rescaleFactor),

(int)(tmp.Height / rescaleFactor)));

canvas.AddBitmap(ref tmp2, x, y);

tmp.Dispose();

tmp2.Dispose();

idx++;

}

else

{

Bitmap tmp = (Bitmap)Bitmap.FromFile(fileNames[idx]);

Bitmap tmp2 = new Bitmap(tmp, new Size((int)(tmp.Width / rescaleFactor),

(int)(tmp.Height / rescaleFactor)));

canvas.AddBitmap(ref tmp2, x, y);

tmp.Dispose();

tmp2.Dispose();

}

if(idx == length)

{

return true;

}

return false;

}

public void Reset()

{

idx = 0;

}

public static void LoadBitmap(string fileName, ref Bitmap bmp,double rescaleFactor=1.0f)

{

fileName += ".png";

fileName = "Resources/" + fileName;

bmp = (Bitmap)Bitmap.FromFile(fileName);

Bitmap tmp = new Bitmap(bmp, new Size((int)(bmp.Width / rescaleFactor),(int)(bmp.Height / rescaleFactor)));

bmp.Dispose();

bmp = tmp;

}

public void LoadBitmap2(string fileName, ref string str)

{

fileName += ".png";

fileName = "Resources/" + fileName;

str = fileName;

}

public Point GetLocation(int flag)

{

Point ret = eyeLocations[Math.Max(0, idx - 1), flag];

ret.X = (int)(ret.X / rescaleFactor);

ret.Y = (int)(ret.Y / rescaleFactor);

return ret;

}

public static void SetScale(float scale)

{

rescaleFactor = scale;

}

}

}

## **2. Canvas.cs**

using System;

using System.Drawing;

using System.Drawing.Imaging;

namespace Uni\_Form\_Trans\_Test\_01

{

public class Canvas

{

private Bitmap BitmapObject;

public int Width, Height;

Graphics gfx;

// Constructor for bitmap being passed as default value

public Canvas(Bitmap bmp)

{

BitmapObject = bmp;

gfx = Graphics.FromImage(BitmapObject);

}

// Constructor for only make canvas

public Canvas(int width, int height)

{

BitmapObject = new Bitmap(width, height);

this.Width = width;

this.Height = height;

gfx = Graphics.FromImage(BitmapObject);

}

public void Resize(int width, int height)

{

if (BitmapObject != null)

BitmapObject.Dispose();

BitmapObject = new Bitmap(width, height);

this.Width = width;

this.Height = height;

gfx = Graphics.FromImage(BitmapObject);

}

// Returns current bitmap

public Bitmap GetBitmap()

{

return BitmapObject;

}

// This is an important function. Overlays the new bitmap over the existing canvas

public void AddBitmap(ref Bitmap bmpObj, int x, int y)

{

gfx.DrawImage(bmpObj, new Point(x, y));

}

public void Clear()

{

gfx = Graphics.FromImage(BitmapObject);

gfx.Clear(Color.Transparent);

//gfx.DrawRectangle(Pens.Red, 0, 0, Width - 1, Height - 1);

}

public void FlipBitmap()

{

BitmapObject.RotateFlip(RotateFlipType.Rotate180FlipY);

}

public void SetTransparency(float alphaLevel)

{

Bitmap img = new Bitmap(BitmapObject);

Clear();

Graphics graphics = Graphics.FromImage(BitmapObject);

ColorMatrix colormatrix = new ColorMatrix();

colormatrix.Matrix33 = alphaLevel;

ImageAttributes imgAttribute = new ImageAttributes();

imgAttribute.SetColorMatrix(colormatrix, ColorMatrixFlag.Default, ColorAdjustType.Bitmap);

graphics.DrawImage(img, new Rectangle(0, 0, BitmapObject.Width, BitmapObject.Height), 0, 0, img.Width, img.Height, GraphicsUnit.Pixel, imgAttribute);

graphics.Dispose(); // Releasing all resource used by graphics

img.Dispose();

imgAttribute.Dispose();

}

}

}

## **3. ControlManger.cs**

using System;

using System.Drawing;

using System.Diagnostics;

using System.Windows.Forms;

using TransParentModule;

namespace Uni\_Form\_Trans\_Test\_01

{

public partial class Form1 : TransparentForm

{

#region Enumerations

enum Direction { LEFT, RIGHT, NULL };

enum State { REST, WALK, ACTION\_DANCE, FLY };

enum SubState { RESTV1, RESTV2,

RESTV3S, RESTV3L1, RESTV3L2, RESTV3E,

RESTV4S, RESTV4L, RESTV4E,

WALKV1S, WALKV1L, WALKV1E,

WALKV2S, WALKV2L, WALKV2E,

FLYS1, FLYS2, FLYL, FLYE2, FLYE1};

#endregion

#region Declarations

Timer timer;

Canvas canvas;

Direction direction = Direction.LEFT;

State state = State.REST;

SubState subState = SubState.RESTV1;

long cnt = 0;

int yOffset = 0;

int xPositionCanvas = 100, yPositionCanvas = 100, destX = 0, destY = 0, lim = 5;

float scale = 1.0f;

bool changeState = false;

float transparencyLevel = 1f;

Random rnd;

Bitmap dinoBabyEyeBkgL, dinoBabyEyeBkgR, dinoBabyEyePupilL, dinoBabyEyePupilR;

Animate action\_Dance, restV1, restV2;

Animate restV3S, restV3L1, restV3L2, restV3E;

Animate restV4S, restV4L, restV4E;

Animate walkV1S, walkV1L, walkV1E;

Animate walkV2S, walkV2L, walkV2E;

Animate flyS1, flyS2, flyL, flyE1, flyE2;

double cX = 0, xDestination = 0;

long currentStateCounter = 0;

Point p1 = new Point(0, 0), p2 = new Point(0, 0);

long counter = 0;

#endregion

private void DrawEye(Point p, ref Bitmap bkg, ref Bitmap pupil, Direction eye)

{

int cX = Cursor.Position.X;

int cY = Cursor.Position.Y;

double dX = 0, dY = 0;

Point offset = new Point(3, 4);

if (direction == Direction.RIGHT)

{

dX = (cX - (canvas.Width - p.X) - xPositionCanvas) / 100;

dY = (cY - p.Y - yPositionCanvas) / 100;

dX = -Math.Atan(dX);

dY = Math.Atan(dY);

}

else

{

dX = (cX - p.X - xPositionCanvas) / 100;

dY = (cY - p.Y - yPositionCanvas) / 100;

dX = Math.Atan(dX);

dY = Math.Atan(dY);

}

if (eye == Direction.LEFT)

{

dX \*= 1.3;

dY \*= 1.4;

}

else

{

dX \*= 1.5;

dY \*= 3;

}

canvas.AddBitmap(ref bkg, p.X + (int)(dX - (offset.X + 3) / scale),

p.Y + (int)((dY - offset.Y - 3) / scale));

canvas.AddBitmap(ref pupil, p.X + (int)((dX \* 1.3f - offset.X) / scale),

p.Y + (int)((dY - offset.Y) \* 1.3f / scale));

}

private double GetProbability()

{

return rnd.NextDouble();

}

private int GetLoc()

{

if (direction == Direction.LEFT)

return p1.X;

else

{

if (subState == SubState.RESTV1 || subState == SubState.RESTV2)

return 250 - p1.X;

else

return 300 - p1.X;

}

}

private void ManageControl(object sender, EventArgs args)

{

// To control flow of program

// Rest of the things will be handeled from here

if(cnt % 100 == 0)

GC.Collect();

cnt++;

if (!changeState)

{

switch (state)

{

case State.REST:

if (Math.Abs(Cursor.Position.X - cX) < 2)

currentStateCounter++;

else

currentStateCounter = 0;

if (currentStateCounter >= 300 && Math.Abs(Cursor.Position.X - xPositionCanvas - GetLoc()) > 100)

{

changeState = true;

currentStateCounter = 0;

xDestination = (int)cX;

if (cX > xPositionCanvas)

direction = Direction.RIGHT;

else

direction = Direction.LEFT;

}

if(currentStateCounter >= 2000 && (subState == SubState.RESTV1 || subState == SubState.RESTV2))

{

double prob = GetProbability();

if (prob < 0.2)

subState = SubState.RESTV3S;

else if (prob > 0.4 && prob < 0.6)

subState = SubState.RESTV4S;

else if (prob > 0.7)

{

state = State.FLY;

subState = SubState.FLYS1;

}

}

if (currentStateCounter > 100)

{

if (cX > xPositionCanvas)

direction = Direction.RIGHT;

else

direction = Direction.LEFT;

}

break;

case State.WALK:

if (Math.Abs(xPositionCanvas + GetLoc() - xDestination) < 20 ||

xPositionCanvas < 20 || xPositionCanvas > Screen.PrimaryScreen.WorkingArea.Width \* 0.8f)

changeState = true;

break;

}

}

cX = Cursor.Position.X;

int xOffset = 0;

int taskBarLocation = Math.Max(0, Screen.PrimaryScreen.WorkingArea.Bottom - (int)canvas.Height + (int)(40 / scale));

// This Part Controls What Will Be Displayed On Screen

switch (state)

{

case State.REST:

{

switch (subState)

{

case SubState.RESTV1:

if(restV1.Update(cnt, 0, 0))

{

if (GetProbability() > 0.4)

subState = SubState.RESTV2;

if (changeState)

{

state = State.WALK;

if (GetProbability() > 0.8)

subState = SubState.WALKV1S;

else

subState = SubState.WALKV2S;

changeState = false;

}

}

p1 = restV1.GetLocation(0);

p2 = restV1.GetLocation(1);

break;

case SubState.RESTV2:

if(restV2.Update(cnt, 0, 0))

{

double prob = GetProbability();

if (prob > 0.8)

subState = SubState.RESTV1;

else if (prob > 0.95)

{

state = State.ACTION\_DANCE;

}

if (changeState)

{

state = State.WALK;

if (GetProbability() > 0.8)

subState = SubState.WALKV1S;

else

subState = SubState.WALKV2S;

changeState = false;

}

}

p1 = restV2.GetLocation(0);

p2 = restV2.GetLocation(1);

break;

case SubState.RESTV3S:

xOffset = -25;

if (restV3S.Update(cnt, 0, 0))

{

subState = SubState.RESTV3L1;

}

p1 = restV3S.GetLocation(0);

p2 = restV3S.GetLocation(1);

break;

case SubState.RESTV3L1:

xOffset = -25;

if (restV3L1.Update(cnt, 0, 0))

{

if (changeState)

subState = SubState.RESTV3E;

else

{

if (GetProbability() > 0.9)

subState = SubState.RESTV3L2;

else

subState = SubState.RESTV3L1;

changeState = false;

}

}

p1 = restV3L1.GetLocation(0);

p2 = restV3L1.GetLocation(1);

break;

case SubState.RESTV3L2:

xOffset = -25;

if (restV3L2.Update(cnt, 0, 0))

{

if (changeState)

subState = SubState.RESTV3E;

else

{

if (GetProbability() > 0.8)

subState = SubState.RESTV3L2;

else

subState = SubState.RESTV3L1;

}

}

p1 = restV3L2.GetLocation(0);

p2 = restV3L2.GetLocation(1);

break;

case SubState.RESTV3E:

xOffset = -25;

if (restV3E.Update(cnt, 0, 0))

{

state = State.WALK;

if (GetProbability() > 0.8)

subState = SubState.WALKV1S;

else

subState = SubState.WALKV2S;

changeState = false;

}

p1 = restV3E.GetLocation(0);

p2 = restV3E.GetLocation(1);

break;

case SubState.RESTV4S:

xOffset = -25;

if (restV4S.Update(cnt, 0, 0))

subState = SubState.RESTV4L;

p1 = restV4S.GetLocation(0);

p2 = restV4S.GetLocation(1);

break;

case SubState.RESTV4L:

xOffset = -25;

if (restV4L.Update(cnt, 0, 0) && changeState)

subState = SubState.RESTV4E;

p1 = restV4L.GetLocation(0);

p2 = restV4L.GetLocation(1);

break;

case SubState.RESTV4E:

xOffset = -25;

if (restV4E.Update(cnt, 0, 0))

{

state = State.WALK;

if (GetProbability() > 0.8)

subState = SubState.WALKV1S;

else

subState = SubState.WALKV2S;

changeState = false;

}

p1 = restV4E.GetLocation(0);

p2 = restV4E.GetLocation(1);

break;

}

break;

}

case State.WALK:

{

switch (subState)

{

case SubState.WALKV1S:

xOffset = -25;

if (walkV1S.Update(cnt, 0, 0))

subState = SubState.WALKV1L;

p1 = walkV1S.GetLocation(0);

p2 = walkV1S.GetLocation(1);

break;

case SubState.WALKV1L:

if (direction == Direction.RIGHT)

xPositionCanvas += 6;

else

xPositionCanvas -= 6;

xOffset = -25;

if (walkV1L.Update(cnt, 0, 0) && changeState)

subState = SubState.WALKV1E;

p1 = walkV1L.GetLocation(0);

p2 = walkV1L.GetLocation(1);

break;

case SubState.WALKV1E:

xOffset = -25;

if (walkV1E.Update(cnt, 0, 0))

{

subState = SubState.RESTV2;

state = State.REST;

changeState = false;

}

p1 = walkV1E.GetLocation(0);

p2 = walkV1E.GetLocation(1);

break;

case SubState.WALKV2S:

xOffset = -25;

if (walkV2S.Update(cnt, 0, 0))

subState = SubState.WALKV2L;

p1 = walkV2S.GetLocation(0);

p2 = walkV2S.GetLocation(1);

break;

case SubState.WALKV2L:

if (direction == Direction.RIGHT)

xPositionCanvas += 3;

else

xPositionCanvas -= 3;

xOffset = -25;

if (walkV2L.Update(cnt, 0, 0) && changeState)

subState = SubState.WALKV2E;

p1 = walkV2L.GetLocation(0);

p2 = walkV2L.GetLocation(1);

break;

case SubState.WALKV2E:

xOffset = -25;

if (walkV2E.Update(cnt, 0, 0))

{

state = State.REST;

subState = SubState.RESTV2;

changeState = false;

}

p1 = walkV2E.GetLocation(0);

p2 = walkV2E.GetLocation(1);

break;

}

break;

}

case State.ACTION\_DANCE:

{

xOffset = -40;

if (action\_Dance.Update(cnt, 0, 0))

{

changeState = false;

state = State.REST;

subState = SubState.RESTV2;

}

p1 = action\_Dance.GetLocation(0);

p2 = action\_Dance.GetLocation(1);

break;

}

case State.FLY:

{

xOffset = -40;

switch (subState)

{

case SubState.FLYS1:

if(flyS1.Update(cnt, 0, 0))

subState = SubState.FLYS2;

p1 = flyS1.GetLocation(0);

p2 = flyS1.GetLocation(1);

yOffset = 0;

destY = 0;

break;

case SubState.FLYS2:

if (flyS2.Update(cnt, 0, 0))

subState = SubState.FLYL;

p1 = flyS2.GetLocation(0);

p2 = flyS2.GetLocation(1);

yOffset -= 3;

break;

case SubState.FLYL:

{

if (counter == 0)

{

destX = Cursor.Position.X;

destY = Cursor.Position.Y;

}

bool stable = true;

if (yOffset >= -30)

subState = SubState.FLYE2;

flyL.Update(cnt, 0, 0);

// To change the direction of Dino

if (Math.Abs(destX - xPositionCanvas) > 50)

{

if (destX > xPositionCanvas && counter == 0) // Counter is used for adding delay

direction = Direction.RIGHT; // But in next part of code its also

else if(counter == 0) // used as input to Sin function

direction = Direction.LEFT;

if (direction == Direction.RIGHT)

{

xPositionCanvas += 3;

}

else

{

xPositionCanvas -= 3;

}

xPositionCanvas = (int)Math.Min(xPositionCanvas, Screen.PrimaryScreen.WorkingArea.Width \* 0.8f);

stable = false;

}

if (yPositionCanvas > destY && Math.Abs(destY - yPositionCanvas) > lim)

{

lim = 5;

yOffset -= 2;

if (yOffset < -taskBarLocation)

yOffset = -taskBarLocation;

stable = false;

}

else if (yPositionCanvas < destY && Math.Abs(destY - yPositionCanvas) > lim)

{

lim = 5;

yOffset += 2;

if (yOffset > 0)

yOffset = 0;

stable = false;

}

counter += 10;

if (stable) // This gives effect of hovering using sin function

{

yOffset += (int)(2 \* Math.Sin(counter / 100.0));

lim = 50;

}

if (counter > 628)

counter = 0;

p1 = flyL.GetLocation(0);

p2 = flyL.GetLocation(1);

break;

}

case SubState.FLYE2:

if (flyE2.Update(cnt, 0, 0))

subState = SubState.FLYE1;

p1 = flyE2.GetLocation(0);

p2 = flyE2.GetLocation(1);

yOffset += 4;

break;

case SubState.FLYE1:

if (flyE1.Update(cnt, 0, 0))

{

state = State.REST;

subState = SubState.RESTV2;

}

p1 = flyE1.GetLocation(0);

p2 = flyE1.GetLocation(1);

break;

}

break;

}

}

DrawEye(p1, ref dinoBabyEyeBkgL, ref dinoBabyEyePupilL, Direction.LEFT);

DrawEye(p2, ref dinoBabyEyeBkgR, ref dinoBabyEyePupilR, Direction.RIGHT);

yPositionCanvas = Math.Min(taskBarLocation, taskBarLocation + yOffset);

if (direction == Direction.RIGHT)

{

xOffset \*= -1;

xOffset -= 80;

}

this.Location = new Point(xPositionCanvas + xOffset, yPositionCanvas);

if (direction == Direction.RIGHT)

canvas.FlipBitmap();

int xC, yC;

xC = Cursor.Position.X;

yC = Cursor.Position.Y;

if ((xC > xPositionCanvas && xC < xPositionCanvas + canvas.Width) &&

(yC > yPositionCanvas && yC < yPositionCanvas + canvas.Height))

{

if(transparencyLevel > 0.1f)

transparencyLevel -= 0.05f;

}

else

transparencyLevel += 0.008f;

if (transparencyLevel != 1f)

canvas.SetTransparency(transparencyLevel);

SetBitmap(canvas.GetBitmap());

canvas.Clear();

cnt %= 10000000;

}

}

}

## 

## 

## 

## 

## **4. From1.cs**

using System;

using System.Drawing;

using System.Resources;

using TransParentModule;

using System.Windows.Forms;

namespace Uni\_Form\_Trans\_Test\_01

{

public partial class Form1 : TransparentForm

{

public Form1()

{

InitializeComponent();

Console.WriteLine("Main Program Started!!");

}

public void ChangeSize(float size)

{

scale = size;

canvas.Resize((int)(325 / scale), (int)(250 / scale));

Animate.SetScale(size);

if (dinoBabyEyeBkgL == null)

dinoBabyEyeBkgL.Dispose();

if (dinoBabyEyeBkgR == null)

dinoBabyEyeBkgR.Dispose();

if (dinoBabyEyePupilL == null)

dinoBabyEyePupilL.Dispose();

if (dinoBabyEyePupilR == null)

dinoBabyEyePupilR.Dispose();

Animate.LoadBitmap("DinoBabyEyeBkgL", ref dinoBabyEyeBkgL, scale);

Animate.LoadBitmap("DinoBabyEyeBkgR", ref dinoBabyEyeBkgR, scale);

Animate.LoadBitmap("DinoBabyEyePupilL", ref dinoBabyEyePupilL, scale);

Animate.LoadBitmap("DinoBabyEyePupilR", ref dinoBabyEyePupilR, scale);

}

private void Form1\_Load(object sender, EventArgs e)

{

this.Location = new Point(xPositionCanvas, yPositionCanvas);

this.ShowInTaskbar = false;

ResourceManager rm = Properties.Resources.ResourceManager;

canvas = new Canvas((int)(325 / scale), (int)(250 / scale));

timer = new System.Windows.Forms.Timer()

{

Enabled = true,

Interval = 25

};

timer.Start();

timer.Tick += new EventHandler(ManageControl);

rnd = new Random();

action\_Dance = new Animate("Action\_Dance", 5, 179, 1, scale, ref canvas, ref rm);

restV1 = new Animate("RestV1", 5, 74, 1, scale, ref canvas, ref rm);

restV2 = new Animate("RestV2", 5, 69, 1, scale, ref canvas, ref rm);

restV3S = new Animate("RestV3S", 5, 24, 1, scale, ref canvas, ref rm);

restV3L1 = new Animate("RestV3L1", 5, 29, 1, scale, ref canvas, ref rm);

restV3L2 = new Animate("RestV3L2", 5, 75, 1, scale, ref canvas, ref rm);

restV3E = new Animate("RestV3E", 5, 25, 1, scale, ref canvas, ref rm);

restV4S = new Animate("RestV4S", 5, 73, 1, scale, ref canvas, ref rm);

restV4L = new Animate("RestV4L", 5, 65, 1, scale, ref canvas, ref rm);

restV4E = new Animate("RestV4E", 5, 37, 1, scale, ref canvas, ref rm);

walkV1S = new Animate("WalkV1S", 5, 19, 1, scale, ref canvas, ref rm);

walkV1L = new Animate("WalkV1L", 5, 18, 1, scale, ref canvas, ref rm);

walkV1E = new Animate("WalkV1E", 5, 50, 1, scale, ref canvas, ref rm);

walkV2S = new Animate("WalkV2S", 5, 11, 1, scale, ref canvas, ref rm);

walkV2L = new Animate("WalkV2L", 5, 28, 1, scale, ref canvas, ref rm);

walkV2E = new Animate("WalkV2E", 5, 25, 1, scale, ref canvas, ref rm);

flyS1 = new Animate("FlyS1", 5, 19, 1, scale, ref canvas, ref rm);

flyS2 = new Animate("FlyS2", 5, 13, 1, scale, ref canvas, ref rm);

flyL = new Animate("FlyL", 5, 32, 1, scale, ref canvas, ref rm);

flyE1 = new Animate("FlyE1", 5, 19, 1, scale, ref canvas, ref rm);

flyE2 = new Animate("FlyE2", 5, 13, 1, scale, ref canvas, ref rm);

Animate.LoadBitmap("DinoBabyEyeBkgL", ref dinoBabyEyeBkgL, scale);

Animate.LoadBitmap("DinoBabyEyeBkgR", ref dinoBabyEyeBkgR, scale);

Animate.LoadBitmap("DinoBabyEyePupilL", ref dinoBabyEyePupilL, scale);

Animate.LoadBitmap("DinoBabyEyePupilR", ref dinoBabyEyePupilR, scale);

Console.WriteLine("Loading Completed");

}

}

}

## 

## 

## 

## 

## 

## **5. Program.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Uni\_Form\_Trans\_Test\_01

{

static class Program

{

/// <summary>

/// The main entry point for the application.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new Setting());

}

}

}

## **6. Setting.cs**

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 Uni\_Form\_Trans\_Test\_01

{

public partial class Setting : Form

{

Form1 dino;

public Setting()

{

InitializeComponent();

}

private void Setting\_Load(object sender, EventArgs e)

{

dino = new Form1();

dino.Show();

}

private void button1\_Click(object sender, EventArgs e)

{

this.Hide();

}

private void exitToolStripMenuItem\_Click(object sender, EventArgs e)

{

this.notifyIcon1.Dispose();

this.Close();

}

private void settingsToolStripMenuItem\_Click(object sender, EventArgs e)

{

this.Show();

}

private void button2\_Click(object sender, EventArgs e)

{

this.notifyIcon1.Dispose();

this.Close();

}

private void trackBar1\_Scroll(object sender, EventArgs e)

{

float size = 9 - trackBar1.Value;

size = size / 10f + 1;

dino.ChangeSize(size);

}

private void checkBox1\_CheckedChanged(object sender, EventArgs e)

{

if (checkBox1.CheckState == CheckState.Checked)

notifyIcon1.Visible = false;

else

notifyIcon1.Visible = true;

}

}

}

## **7. TransparentForm.cs**

using System;

using System.Drawing;

using System.Drawing.Imaging;

using System.Windows.Forms;

using System.Runtime.InteropServices;

namespace TransParentModule

{

class Win32

{

public enum Bool

{

False = 0,

True

};

[StructLayout(LayoutKind.Sequential)]

public struct Point

{

public Int32 x;

public Int32 y;

public Point(Int32 x, Int32 y)

{

this.x = x;

this.y = y;

}

}

[StructLayout(LayoutKind.Sequential)]

public struct Size

{

public Int32 cx;

public Int32 cy;

public Size(Int32 cx, Int32 cy)

{

this.cx = cx;

this.cy = cy;

}

}

[StructLayout(LayoutKind.Sequential, Pack = 1)]

struct ARGB

{

public byte blue;

public byte green;

public byte red;

public byte alpha;

}

[StructLayout(LayoutKind.Sequential, Pack = 1)]

public struct BLENDFUNCTION

{

public byte blendOp;

public byte blendFlags;

public byte sourceConstantAlpha;

public byte alphaFormat;

}

public const Int32 ULW\_COLORKEY = 0x00000001;

public const Int32 ULW\_ALPHA = 0x00000002;

public const Int32 ULW\_OPAQUE = 0x00000003;

public const Int32 AC\_SRC\_OVER = 0x00;

public const Int32 AC\_SRC\_ALPHA = 0x01;

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Design", "CA1060:MovePInvokesToNativeMethodsClass")]

[DllImport("user32.dll", ExactSpelling = true, SetLastError = true)]

public static extern Bool UpdateLayeredWindow(IntPtr hwnd, IntPtr hdcDst, ref Point pptDst, ref Size psize, IntPtr hdcSrc, ref Point pprSrc, Int32 crKey, ref BLENDFUNCTION pblend, Int32 dwFlags);

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Design", "CA1060:MovePInvokesToNativeMethodsClass")]

[DllImport("user32.dll", ExactSpelling = true, SetLastError = true)]

public static extern IntPtr GetDC(IntPtr hWnd);

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Design", "CA1060:MovePInvokesToNativeMethodsClass")]

[DllImport("user32.dll", ExactSpelling = true)]

public static extern int ReleaseDC(IntPtr hWnd, IntPtr hDC);

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Design", "CA1060:MovePInvokesToNativeMethodsClass")]

[DllImport("gdi32.dll", ExactSpelling = true, SetLastError = true)]

public static extern IntPtr CreateCompatibleDC(IntPtr hDC);

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Design", "CA1060:MovePInvokesToNativeMethodsClass")]

[DllImport("gdi32.dll", ExactSpelling = true, SetLastError = true)]

public static extern Bool DeleteDC(IntPtr hdc);

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Design", "CA1060:MovePInvokesToNativeMethodsClass")]

[DllImport("gdi32.dll", ExactSpelling = true)]

public static extern IntPtr SelectObject(IntPtr hDC, IntPtr hObject);

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Design", "CA1060:MovePInvokesToNativeMethodsClass")]

[DllImport("gdi32.dll", ExactSpelling = true, SetLastError = true)]

public static extern Bool DeleteObject(IntPtr hObject);

}

public class TransparentForm : Form

{

public TransparentForm()

{

FormBorderStyle = FormBorderStyle.None;

TopMost = true;

}

public void SetBitmap(Bitmap bitmap)

{

SetBitmap(bitmap, 255);

}

public void SetBitmap(Bitmap bitmap, byte opacity)

{

if (bitmap.PixelFormat != PixelFormat.Format32bppArgb)

throw new ApplicationException("The bitmap must be 32ppp with alpha-channel.");

IntPtr screenDc = Win32.GetDC(IntPtr.Zero);

IntPtr memDc = Win32.CreateCompatibleDC(screenDc);

IntPtr hBitmap = IntPtr.Zero;

IntPtr oldBitmap = IntPtr.Zero;

try

{

hBitmap = bitmap.GetHbitmap(Color.FromArgb(0));

oldBitmap = Win32.SelectObject(memDc, hBitmap);

Win32.Size size = new Win32.Size(bitmap.Width, bitmap.Height);

Win32.Point pointSource = new Win32.Point(0, 0);

Win32.Point topPos = new Win32.Point(Left, Top);

Win32.BLENDFUNCTION blend = new Win32.BLENDFUNCTION()

{

blendOp = Win32.AC\_SRC\_OVER,

blendFlags = 0,

sourceConstantAlpha = opacity,

alphaFormat = Win32.AC\_SRC\_ALPHA

};

Win32.UpdateLayeredWindow(Handle, screenDc, ref topPos, ref size, memDc, ref pointSource, 0, ref blend, Win32.ULW\_ALPHA);

}

finally

{

Win32.ReleaseDC(IntPtr.Zero, screenDc);

if (hBitmap != IntPtr.Zero)

{

Win32.SelectObject(memDc, oldBitmap);

Win32.DeleteObject(hBitmap);

}

Win32.DeleteDC(memDc);

}

}

protected override CreateParams CreateParams

{

get

{

CreateParams cp = base.CreateParams;

cp.ExStyle |= 0x00080000 | 0x20; // To Set Windows Layered And Transparent + Click Through

return cp;

}

}

}}