**WelcomePage.cs**

using System;

using System.Windows.Forms;

namespace MidTerm

{

public partial class WelcomePage : Form

{

public WelcomePage()

{

InitializeComponent();

}

private void Level1\_Click(object sender, EventArgs e)

{

//Level2Panel.Visible = false;

//Level3Panel.Visible = false;

welcomepanel.Visible = false;

Level1Panel.Visible = true;

Level1Panel.BringToFront();

}

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

{

Level1Panel.Visible = false;

Level2Panel.Visible = false;

Level3Panel.Visible = false;

welcomepanel.Visible = true;

}

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

{

Level1Panel.Visible = false;

Level2Panel.Visible = false;

Level3Panel.Visible = false;

}

private void Level2\_Click(object sender, EventArgs e)

{

welcomepanel.Visible = false;

Level2Panel.Visible = true;

Level2Panel.BringToFront();

}

private void Level3Btn\_Click(object sender, EventArgs e)

{

welcomepanel.Visible = false;

Level3Panel.Visible = true;

Level3Panel.BringToFront();

}

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

{

Form magicSquareGame = new MagicSquare();

magicSquareGame.Show();

}

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

{

Form findAges = new FindAges();

findAges.Show();

}

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

{

Form openLockers = new OpenLockers();

openLockers.Show();

}

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

{

Form BeerGame = new BeerGame();

BeerGame.Show();

}

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

{

Form BucketGame = new BucketGame();

BucketGame.Show();

}

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

{

Form RememberFlag = new RememberFlag();

RememberFlag.Show();

}

private void button6\_Click(object sender, EventArgs e)

{

Form Hangman = new HangMan();

Hangman.Show();

}

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

{

Form formOrder = new FormOrder();

formOrder.Show();

}

}

}

**BeerGame.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 MidTerm

{

public partial class BeerGame : Form

{

// Determine the class level variables

private Color BeerColor = Color.SandyBrown;

private Color EmptyColor = Color.Transparent;

private List<int> current = new List<int> { 0, 0 };

public BeerGame()

{

// Initialize the component

InitializeComponent();

// Let the label size for question text be bigger

label2.MaximumSize = new Size(400, 100);

label2.AutoSize = true;

label2.Text = "A bartender has a three-pint glass and a five-pint glass. A customer walks in and orders four pints of beer. Without a measuring cup but with an unlimited supply of beer how does he get a single pint in either glass?";

}

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

{

// Change the color of the container

threePint.BackColor = BeerColor;

// Assign the current amount

current[0] = GetNumericPint("three");

// Check if the goal has achieved

HasWon();

}

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

{

// Change the color of the container

fivePint.BackColor = BeerColor;

// Assign the current amount

current[1] = GetNumericPint("five");

// Check if the goal has achieved

HasWon();

}

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

{

// Remove the color of the container

threePint.BackColor = EmptyColor;

// Assign the current amount

current[0] = 0;

// Check if the container is filled with any amount

if (this.Controls.Find("innerThree", true).Length > 0)

{

// Determine the layers (filled) element

var element = this.Controls.Find("innerThree", true).First();

// Remove the color

element.BackColor = EmptyColor;

// Send that to back

element.SendToBack();

}

// Check if the goal has achieved

HasWon();

}

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

{

// Remove the color of the container

fivePint.BackColor = EmptyColor;

// Assign the current amount

current[1] = 0;

// Check if the container is filled with any amount

if (this.Controls.Find("innerFive", true).Length > 0)

{

// Determine the layers (filled) element

var element = this.Controls.Find("innerFive", true).First();

// Remove the color

element.BackColor = EmptyColor;

// Send that to back

element.SendToBack();

}

// Check if the goal has achieved

HasWon();

}

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

{

// Check the capacity of target

int targetCapacity = GetNumericPint("five") - current[1];

// If the origin container is empty, trigger an error message

if (current[0] == 0)

{

MessageBox.Show("Three is empty. Fill that before pouring.");

}

// If the target container is full, trigger an error message

if (targetCapacity <= 0)

{

MessageBox.Show("Five is full. Empty that before pouring.");

}

// If target capacity is smaller that origin, assign amounts on containers

else if (targetCapacity < GetNumericPint("three"))

{

current[1] = GetNumericPint("five");

current[0] = current[0] - targetCapacity;

}

// Otherwise, assign amounts on containers

else

{

current[1] = current[0] + current[1];

current[0] = (current[0] - targetCapacity) < 0

? 0

: (current[0] - targetCapacity);

}

// Update the amounts with visual representation on both containers

PourBeer(GetNumericPint("three"));

PourBeer(GetNumericPint("five"));

// Check if the goal has achieved

HasWon();

}

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

{

// Check the capacity of target

int targetCapacity = GetNumericPint("three") - current[0];

// If the origin container is empty, trigger an error message

if (current[1] == 0)

{

MessageBox.Show("Five is empty. Fill that before pouring.");

}

// If the target container is full, trigger an error message

if (targetCapacity <= 0)

{

MessageBox.Show("Three is full. Empty that before pouring.");

}

// If target capacity is bigger that origin, assign amounts on containers

else if (targetCapacity >= current[1])

{

current[0] = current[1] + current[0];

current[1] = 0;

} else

{

current[0] = targetCapacity + current[0];

current[1] = current[1] - targetCapacity;

}

// Update the amounts with visual representation on both containers

PourBeer(GetNumericPint("three"));

PourBeer(GetNumericPint("five"));

// Check if the goal has achieved

HasWon();

}

private int GetNumericPint(string pint)

{

return pint == "three" ? 3 : 5;

}

private void PourBeer(int target)

{

// Check if the container is 3

if (target == GetNumericPint("three"))

{

// Check if it has the hidden container that visually shows it has amunt in it

if (this.Controls.Find("innerThree", true).Length > 0)

{

// If found, remove that

this.Controls.Find("innerThree", true).First().Dispose();

}

// Remove the color of the container

threePint.BackColor = EmptyColor;

// Create a new panel and assign some properties to it

Panel panel = new Panel();

panel.Name = "innerThree";

panel.Size = new System.Drawing.Size(70, current[0] \* 50);

panel.Location = new Point(79, 131+((GetNumericPint("three") - current[0]) \* 50));

panel.BackColor = BeerColor;

// Add the panel to the form

this.Controls.Add(panel);

// If the panel has been successfully added, bring that to front

if (this.Controls.Find("innerThree", true).Length > 0)

{

this.Controls.Find("innerThree", true).First().BringToFront();

}

}

// Check if the container is 5

if (target == GetNumericPint("five"))

{

// Check if it has the hidden container that visually shows it has amunt in it

if (this.Controls.Find("innerFive", true).Length > 0)

{

this.Controls.Find("innerFive", true).First().Dispose();

}

// If found, remove that

fivePint.BackColor = EmptyColor;

// Remove the color of the container

Panel panel = new Panel();

panel.Name = "innerFive";

panel.Size = new System.Drawing.Size(70, current[1] \* 50);

panel.Location = new Point(218, 31+((GetNumericPint("five") - current[1])\*50));

panel.BackColor = BeerColor;

// Add the panel to the form

this.Controls.Add(panel);

// If the panel has been successfully added, bring that to front

if (this.Controls.Find("innerFive", true).Length > 0)

{

this.Controls.Find("innerFive", true).First().BringToFront();

}

}

// Check if the goal has achieved

HasWon();

}

private void HasWon()

{

// If container 5 has amount 4, then call it success

if (current[1] == 4)

{

label1.Text = "Congratulations !";

} else

{

label1.Text = "Three-Pint: " + current[0] + ", Five-Pint: " + current[1];

}

}

}

}

**BucketGame.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 MidTerm

{

public partial class BucketGame : Form

{

// Determine class level variables

private Color WaterColor = Color.SkyBlue;

private Color EmptyColor = Color.Transparent;

private List<int> current = new List<int> { 0, 0 };

public BucketGame()

{

// Initialize the component

InitializeComponent();

// Let the label size for question text be bigger

label2.MaximumSize = new Size(400, 100);

label2.AutoSize = true;

label2.Text = "Using just a five-gallon bucket and a three-gallon bucket, can you put four gallons of water in the five-gallon bucket ? (Assume that you havean unlimited supply of water and that there are no measurement markings of any kind on the buckets.)";

}

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

{

// Change the color of the container

threeBucket.BackColor = WaterColor;

// Assign current amount in the container

current[0] = GetNumericBucket("three");

// Check if the goal has achieved

HasWon();

}

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

{

// Change the color of the container

fiveBucket.BackColor = WaterColor;

// Assign current amount in the container

current[1] = GetNumericBucket("five");

// Check if the goal has achieved

HasWon();

}

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

{

// Remove the color of the container

threeBucket.BackColor = EmptyColor;

// Assign current amount in the container

current[0] = 0;

// Check if the container is filled with any amount

if (this.Controls.Find("innerThree", true).Length > 0)

{

// Determine the layers (filled) element

var element = this.Controls.Find("innerThree", true).First();

// Remove the color

element.BackColor = EmptyColor;

// Send that to back

element.SendToBack();

}

// Check if the goal has achieved

HasWon();

}

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

{

// Remove the color of the container

fiveBucket.BackColor = EmptyColor;

// Assign current amount in the container

current[1] = 0;

// Check if the container is filled with any amount

if (this.Controls.Find("innerFive", true).Length > 0)

{

// Determine the layers (filled) element

var element = this.Controls.Find("innerFive", true).First();

// Remove the color

element.BackColor = EmptyColor;

// Send that to back

element.SendToBack();

}

// Check if the goal has achieved

HasWon();

}

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

{

// Check the capacity of target

int targetCapacity = GetNumericBucket("five") - current[1];

// If the origin container is empty, trigger an error message

if (current[0] == 0)

{

MessageBox.Show("Three is empty. Fill that before pouring.");

}

// If the target container is full, trigger an error message

if (targetCapacity <= 0)

{

MessageBox.Show("Five is full. Empty that before pouring.");

}

// If target capacity is smaller that origin, assign amounts on containers

else if (targetCapacity < GetNumericBucket("three"))

{

current[1] = GetNumericBucket("five");

current[0] = current[0] - targetCapacity;

}

// Otherwise, assign amounts on containers

else

{

current[1] = current[0] + current[1];

current[0] = (current[0] - targetCapacity) < 0

? 0

: (current[0] - targetCapacity);

}

// Update the amounts with visual representation on both containers

PourWater(GetNumericBucket("three"));

PourWater(GetNumericBucket("five"));

// Check if the goal has achieved

HasWon();

}

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

{

// Check the capacity of target

int targetCapacity = GetNumericBucket("three") - current[0];

// If the origin container is empty, trigger an error message

if (current[1] == 0)

{

MessageBox.Show("Five is empty. Fill that before pouring.");

}

// If the target container is full, trigger an error message

if (targetCapacity <= 0)

{

MessageBox.Show("Three is full. Empty that before pouring.");

}

// If target capacity is bigger that origin, assign amounts on containers

else if (targetCapacity >= current[1])

{

current[0] = current[1] + current[0];

current[1] = 0;

}

else

{

current[0] = targetCapacity + current[0];

current[1] = current[1] - targetCapacity;

}

// Update the amounts with visual representation on both containers

PourWater(GetNumericBucket("three"));

PourWater(GetNumericBucket("five"));

// Check if the goal has achieved

HasWon();

}

private int GetNumericBucket(string Bucket)

{

return Bucket == "three" ? 3 : 5;

}

private void PourWater(int target)

{

// Check if the container is 3

if (target == GetNumericBucket("three"))

{

// Check if it has the hidden container that visually shows it has amunt in it

if (this.Controls.Find("innerThree", true).Length > 0)

{

// If found, remove that

this.Controls.Find("innerThree", true).First().Dispose();

}

// Remove the color of the container

threeBucket.BackColor = EmptyColor;

// Create a new panel and assign some properties to it

Panel panel = new Panel();

panel.Name = "innerThree";

panel.Size = new System.Drawing.Size(70, current[0] \* 50);

panel.Location = new Point(79, 131+((GetNumericBucket("three") - current[0]) \* 50));

panel.BackColor = WaterColor;

// Add the panel to the form

this.Controls.Add(panel);

// If the panel has been successfully added, bring that to front

if (this.Controls.Find("innerThree", true).Length > 0)

{

this.Controls.Find("innerThree", true).First().BringToFront();

}

}

// Check if the container is 5

if (target == GetNumericBucket("five"))

{

// Check if it has the hidden container that visually shows it has amunt in it

if (this.Controls.Find("innerFive", true).Length > 0)

{

// If found, remove that

this.Controls.Find("innerFive", true).First().Dispose();

}

// Remove the color of the container

fiveBucket.BackColor = EmptyColor;

// Create a new panel and assign some properties to it

Panel panel = new Panel();

panel.Name = "innerFive";

panel.Size = new System.Drawing.Size(70, current[1] \* 50);

panel.Location = new Point(218, 31+((GetNumericBucket("five") - current[1])\*50));

panel.BackColor = WaterColor;

// Add the panel to the form

this.Controls.Add(panel);

// If the panel has been successfully added, bring that to front

if (this.Controls.Find("innerFive", true).Length > 0)

{

this.Controls.Find("innerFive", true).First().BringToFront();

}

}

// Check if the goal has achieved

HasWon();

}

private void HasWon()

{

// If one of the container has amount 4, then call it success

if (current[0] == 4 || current[1] == 4)

{

label1.Text = "Congratulations !";

}

// Otherwise, keep displaying the current amounts on each container

else

{

label1.Text = "Three-Bucket: " + current[0] + ", Five-Bucket: " + current[1];

}

}

}

}

**RememberFlag.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 MidTerm

{

public partial class RememberFlag : Form

{

// Determine the class variables and objects

private List<int> imagePositions = new List<int>();

private List<int> imagePositionsSetOne = new List<int>();

private List<int> imagePositionsSetTwo = new List<int>();

private Random random = new Random();

private List<int> immediatePosition = new List<int>();

private int counter = 30; // (seconds)

private Timer timer = null;

public RememberFlag()

{

// Initialize the component

InitializeComponent();

// Let the label size for question text be bigger

label1.MaximumSize = new Size(280, 100);

label1.AutoSize = true;

label1.Text = "Click on mask-images to reveal the flag inside. At one time, two images can be revealed. Click to pair up to the another un-revealed flag. Finish that within the given time to win the game.";

// Instantiate a new Timer

timer = new Timer();

}

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

{

// Set the hidden images

GetImagesHidden();

// Assign image positions

setImagePositions();

}

private void setImagePositions()

{

// Break down the image positions into two parts

imagePositionsSetOne = new List<int> { 1, 2, 3, 4, 5, 6, 7, 8 };

imagePositionsSetTwo = new List<int> { 1, 2, 3, 4, 5, 6, 7, 8 };

// Determine the immediate positions

immediatePosition = new List<int> { 0, 0, 0 };

// If the timer has been created, stop the timer

if (timer!=null)

{

timer.Stop();

}

// Shuffle both image positions to spead those pairs in random positions

ShuffleMe(imagePositionsSetOne);

ShuffleMe(imagePositionsSetTwo);

// Merge both sets and shuffle again

imagePositions = ShuffleMe(imagePositionsSetOne.Concat(imagePositionsSetTwo).ToList());

}

void GetImagesHidden()

{

// Iterate through all the controls

foreach (Control i in this.Controls)

{

// Make sure it's a picture-box

if (i is PictureBox)

{

// Make it visible

(i as PictureBox).Show();

// Assign mask image

(i as PictureBox).Image = Image.FromFile("0.png");

}

}

}

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

{

// Restart the application

Application.Restart();

// Prevent it from being closed

Environment.Exit(0);

}

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

{

// If the picture is revealed, do nothing

if ((sender as PictureBox).ImageLocation != null && (sender as PictureBox).ImageLocation != "0.png")

{

return;

}

// If user clicks on the first image-box, start the countdown

if (immediatePosition[0] == 0 && winorloss.Text == "")

{

Countdown();

}

// Find the name of the clicked picture-box

var nameOnly = (sender as PictureBox).Name.ToString();

// Find the length of the name

var length = nameOnly.Length;

// Find the position of the clicked picture-box

var imagePosition = Convert.ToInt32(nameOnly.Substring(10, length - 10));

// Set the immediate position

setImmediatePosition(imagePosition);

// Make sure if the images are matched, if so, remove that pair

if (immediatePosition[0] != 0 && immediatePosition[1] != 0 && imagePositions[immediatePosition[0] - 1] == imagePositions[immediatePosition[1] - 1])

{

(sender as PictureBox).ImageLocation = imagePositions[imagePosition - 1] + ".png";

MatchBox();

}

else

{

// Make sure that bottom immediate picture-box position is already set

if (immediatePosition[2] != 0)

{

// Hide that picture-box

HideBox();

}

// Reveal the image

(sender as PictureBox).ImageLocation = imagePositions[imagePosition - 1] + ".png";

}

}

private void MatchBox()

{

// Determine the pair boxes

var boxToHide1 = this.Controls.Find("pictureBox" + (immediatePosition[0]), false);

var boxToHide2 = this.Controls.Find("pictureBox" + (immediatePosition[1]), false);

// If they are still visible,

if (boxToHide1.Length > 0 && boxToHide2.Length > 0)

{

// Hide both boxes

boxToHide1.First().Hide();

boxToHide2.First().Hide();

// Update the immediate positions

immediatePosition[0] = immediatePosition[2];

immediatePosition[1] = 0;

immediatePosition[2] = 0;

}

// If there are no remaining picture-boxes to reveal

if (Remaining()==false)

{

// Display the message

DisplayMessage();

}

}

private void HideBox()

{

// Determine the name of the picture-box

var pictureBoxName = "pictureBox" + immediatePosition[2];

// Make sure the box exists

if (this.Controls.Find(pictureBoxName, false).Length > 0)

{

// Determine the box

var theImageBox = this.Controls.Find(pictureBoxName, true).First();

// Make sure that it's a picture-box

if (theImageBox is PictureBox)

{

// Assign the mask image to it

(theImageBox as PictureBox).ImageLocation = "0.png";

}

}

}

List<int> ShuffleMe(List<int> list)

{

// Iterate through the list

for (int i = list.Count - 1; i > 1; i--)

{

// Determine a random number

int rnd = random.Next(i + 1);

// Determine the value and assign to the list

int value = list[rnd];

list[rnd] = list[i];

list[i] = value;

}

// Return the shuffled list

return list;

}

private void setImmediatePosition(int position)

{

// Mae sure the position is not set already

if (position != immediatePosition[1])

{

// Update the positions

immediatePosition[2] = immediatePosition[1];

immediatePosition[1] = immediatePosition[0];

immediatePosition[0] = position;

}

}

void Countdown()

{

// Increament the tick

timer.Tick += new EventHandler(DisplayTimer);

// Determine the interval

timer.Interval = 1000;

// Start the counter

timer.Start();

// Display the counter value on the label

displaycounter.Text = counter.ToString();

}

private void DisplayTimer(object sender, EventArgs e)

{

// Decrease the counter

counter--;

// Display the counter value on the label

displaycounter.Text = counter.ToString();

// Check if the counter has reached to zero (0)

if (counter == 0)

{

// Stop the timer

timer.Stop();

// Display the message

DisplayMessage();

}

}

private void DisplayMessage()

{

// Make sure that the label is empty

if (winorloss.Text == "")

{

// If there are some remaining picture-boxes, display lost message

if (Remaining())

{

winorloss.Text = "You Lost!";

winorloss.ForeColor = Color.Red;

}

// Otherwise, display win message

else

{

winorloss.Text = "You Won!";

winorloss.ForeColor = Color.Green;

}

}

}

private bool Remaining()

{

// Initialize the variable as false

bool isRemaining = false;

// Iterate through each controls

foreach (Control i in this.Controls)

{

// Make sure that the control is picture-box

if (i is PictureBox)

{

// Check if there are any picture-boxes remaining

if ((i as PictureBox).ImageLocation == "0.png" || (i as PictureBox).ImageLocation == null)

{

// If so, update the remaining as true

isRemaining = true;

}

}

}

// Return the remaing

return isRemaining;

}

}

}

**HangMan.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 MidTerm

{

public partial class HangMan : Form

{

// Determine the class level variables

List<string> States = new List<string> { "Mississippi", "Connecticut", "Louisiana", "Minnesota", "California", "Washington", "Arkansas", "Colorado", "Delaware", "Illinois", "Kentucky", "Maryland", "Michigan", "Missouri", "Nebraska", "Pennsylvania", "Oklahoma", "Virginia" };

List<char> Alphabets = new List<char>();

List<char> StateAlpha = new List<char>();

int totalMistakes = 0;

int boxSize = 40;

string selectedState = "";

public HangMan()

{

// Initialize the component

InitializeComponent();

}

private void SetAlphabets()

{

// Determine the alhabets as a character array

char[] alpha = Enumerable.Range('A', 26).Select(x => (char)x).ToArray();

// Shuffe the list

Alphabets = ShuffleMe(new List<char>(alpha));

// Instantiate the random object

Random random = new Random();

// Select a state randomly

selectedState = States[random.Next(States.Count)].ToUpper();

// Break down the state-name into a character array

StateAlpha = new List<char>(selectedState.ToArray());

// Determine the positions

int positionX = 300;

int positionY = 230;

// Create an array of labels

Label[] labels = new Label[Alphabets.Count];

// Determine j to break the line

int j = 0;

// Iterate through the alphabets

for (int i = 0; i < Alphabets.Count; i++)

{

// After 10 alphabets, list them one line below

if (i == 10)

{

j = 10;

positionY = positionY + boxSize;

}

// After 20 alphabets, list them two lines below

if (i == 20)

{

j = 20;

positionY = positionY + boxSize;

}

// After 30 alphabets, list them three line below

if (i == 30)

{

j = 30;

positionY = positionY + boxSize;

}

// Create a new label and assign attributes

labels[i] = new Label();

labels[i].Text = Alphabets[i].ToString();

labels[i].ForeColor = Color.Black;

labels[i].BorderStyle = BorderStyle.FixedSingle;

labels[i].Font = new Font("Arial", 15,FontStyle.Bold);

labels[i].Size = new System.Drawing.Size(boxSize - 2, boxSize - 2);

labels[i].Location = new Point(positionX+(boxSize \* (i-j)), positionY);

labels[i].Name = "alpha"+i;

labels[i].TextAlign = ContentAlignment.MiddleCenter;

labels[i].Cursor = Cursors.Hand;

labels[i].Click += Alphabet\_ClickedEvent;

}

// Iterate through all alphabets and add the labels for them

for (int i = 0; i < Alphabets.Count; i++)

{

this.Controls.Add(labels[i]);

}

}

private void SetStateAplphabets()

{

// Create an array of labels

Label[] labels = new Label[StateAlpha.Count];

// Assign the positions

int positionX = 300;

int positionY = 100;

// Iterate through each state name alphabets

for (int i = 0; i < StateAlpha.Count; i++)

{

// Create a new label and assign attributes to them

labels[i] = new Label();

labels[i].Text = "\_";

labels[i].ForeColor = Color.Black;

labels[i].BorderStyle = BorderStyle.FixedSingle;

labels[i].Font = new Font("Arial", 15, FontStyle.Bold);

labels[i].Size = new System.Drawing.Size(boxSize - 2, boxSize - 2);

labels[i].Location = new Point(positionX + (boxSize\*i), positionY);

labels[i].Name = "stateAlpha" + i;

labels[i].TextAlign = ContentAlignment.MiddleCenter;

labels[i].Click += Alphabet\_ClickedEvent;

}

// Iterate through state name alphabets and add the labels for each

for (int i = 0; i < StateAlpha.Count; i++)

{

this.Controls.Add(labels[i]);

}

}

private void Alphabet\_ClickedEvent(object sender, EventArgs e)

{

// Check if total mistakes is 8 (complete hang); if so terminate here

if (totalMistakes >= 8)

{

return;

}

// Find the clicked character

char clickedChar = (sender as Label).Text.ToCharArray()[0];

// Make sure the clicked character in included in the current state characters

if (StateAlpha.Contains(clickedChar))

{

// Find the character

char position = StateAlpha.Where(i => i == clickedChar).First();

// Find the index

int indexPosition = StateAlpha.FindIndex(i => i == clickedChar);

// Iterate through all the controls

foreach (Control i in this.Controls)

{

// Make sure it's a label

if (i is Label)

{

// Make sure that the label is the state alphabet label

if ((i as Label).Name.ToString() == "stateAlpha" + indexPosition)

{

// Make sure that the label name is valid

if ((i as Label).Name.ToString().Contains("stateAlpha")) {

// Change the character in the box

(i as Label).Text = clickedChar.ToString();

// Rename it to something different to avoid selecting next time

(i as Label).Name = "stateAlphaAddressed" + i;

// Replace that character with dash sign to exclude next time

StateAlpha[indexPosition] = "-".ToCharArray()[0];

// Terminate right here

return;

}

}

}

}

}

else

{

// Increment the total mistakes

totalMistakes = totalMistakes + 1;

// Check if it's already hung

Hang();

}

}

private void Hang()

{

// Make sure the mistakes are already 8

if (totalMistakes >= 8)

{

// It's hung so lets reveal the answer

Label label = new Label();

label.Text = "Answer: " + selectedState.ToString();

label.ForeColor = Color.Black;

label.Font = new Font("Arial", 15, FontStyle.Bold);

label.Location = new Point(340, 150);

label.Name = "answer";

label.TextAlign = ContentAlignment.MiddleCenter;

label.AutoSize = true;

this.Controls.Add(label);

// Show the game restart button

button1.Show();

}

// Otherwise, increase the higher steps of hangman

pictureBox1.ImageLocation = "hm\_" + totalMistakes + ".png";

pictureBox1.SizeMode = PictureBoxSizeMode.StretchImage;

}

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

{

// Hide the restart game button at the beginning

button1.Hide();

// Lay out the all alphabets

SetAlphabets();

// Lay out the state alphabets

SetStateAplphabets();

}

List<char> ShuffleMe(List<char> list)

{

// Create a random object

Random random = new Random();

// Iterate through the list

for (int i = list.Count - 1; i > 1; i--)

{

// Determine a random number

int rnd = random.Next(i + 1);

// Determine the value and assign to the list

char value = list[rnd];

list[rnd] = list[i];

list[i] = value;

}

// Return the shuffled list

return list;

}

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

{

// Restart the application

Application.Restart();

// Prevent it from being closed

Environment.Exit(0);

}

}

}