Zachary Briggs
May 23rd, 2018
Assessment Five

1. Requirements

1.1. Goal

Name: Intro to C#

Problem Statement: Create a controls-based application to fill the role of a tool application, which is an application that aids in the development of something else, such as a game.

Problem Specifics: A concept diagram that demonstrates the tool being created in a png format, the tool created in C# that can create and edit files that serve some purpose for a potential video game, the tool must also implement custom complex data types, member variables, inheritance, polymorphism, and multiple constructors.

1.2. Input Information: Each of the textboxes take in an int or string from the user.

1.3. Output Information: A Json file representing the character created is saved out to the characters folder if the Save button is clicked.

1.4. User Interface: A listbox that contains all the characters saved out to the characters folder, text boxes for each of the character's variables, and a button for serializing the characters.

2. System Architecture

Character.cs

/*Prototype: public Character();
Description: Default constructor for Character class.
Arguments: None.
Precondition: None.
Postcondition: A default character is created.
Protection: Public.*/

/*Prototype: public Character(string name);
Description: Overloaded constructor for Character class.
Arguments: None.
Precondition: None.
Postcondition: A default character is created.
Protection: Public.*/

/*Prototype: public void CreateStats(List<Stat> stats);
Description: Creates the default stats for the character.
Arguments: A List of stats.
Precondition: A list of stats.
Postcondition: The stats are added to the lists of stats.
Protection: Public.*/

Form1.cs

/*Prototype: private void UiRefresh(Character c)
Description: Changes all the the uiStats stat refrences to the character's stats and then refreshes the uiStats.
Arguments: A character.
Precondition: A character and a global uiStats.
Postcondition: uiStats are set and then refreshed.
Protection: Private.*/
/*Prototype: public void SetBoxes(Character c);
Description: Sets the name,className,expValue,lvlValue, and moveValue's text to the characters equivalent values.
Arguments: A character.
Precondition: A character and text boxes.
Postcondition: The text boxes are set to the values of the character's stats.
Protection: Public.*/
/*Prototype: public void SetOtherValues(Character c);
Description: Sets the character's values that aren't the stats to the values in the equivalent text boxes.
Arguments: A character.
Precondition: A character and text boxes.
Postcondition: The character's stats are set to the values of the text boxes.
Protection: Public.*/
/*Prototype: private void Listchanged(object sender, EventArgs e);
Description: Changes the currentCharacter to the listBox's highlighted character when the highlighted character is changed.
Arguments: object and eventargs
Precondition: uiStats, listbox, currentCharacter, list of all characters.
Postcondition: The currentCharacter is changed.
Protection: Private.*/
/*Prototype: private void saveButton_Click(object sender, EventArgs e);
Description: Upon clicking the saveButton the currentCharacter's stats are set and all of the characters are serialized.
Then the listBox and loadedCharacters lists are cleared out. All of the serialized characters are then loaded back in.
Arguments: object and eventargs
Precondition: A character,text boxes, somewhere to save the serialized files, a list of characters, a listbox.
Postcondition: The characters are serilized out the the folder.
Protection: Private.*/

Stat.cs

/*Prototype: public Stat();
Description: Default constructor for Stat.
Arguments: None.
Precondition: None.

Postcondition: A new stat is created.
Protection: Public.*/
/*Prototype: public Stat(string name, int value, int growth, int max);
Description: Overloaded constructor for Stat.
Arguments: string representing Name and three ints representing Value, Growth, and Max.
Precondition: None.
Postcondition: A new stat is created with the values passed in.
Protection: Public.*/
/*Prototype: public Stat(string name, int value, int growth, int max, string description);
Description: Overloaded constructor for Stat.
Arguments: two strings representing Name and Description and three ints representing Value, Growth, and Max.
Precondition: None.
Postcondition: A new stat is created with the values passed in.
Protection: Public.*/

UIStat.cs

/*Prototype: public UIStat(Stat statref, UIValueGrowthMax vgm);
Description: Overloaded constructor for UIStat
Arguments: A stat representing statRef and a UIValueGrowthMax representing uivalue
Precondition: None.
Postcondition: A UIStat is created with the values passed in.
Protection: Public.*/
/*Prototype: public void Refresh()
Description: Sets the text box refrences' text to the values in the Stat refrence
Arguments: None.
Precondition: The StatRef have values.
Postcondition: The text box refrences are set.
Protection: Public.*/
/*Prototype: public void SetValues();
Description: The stat refrence is set to the values in the text box refrences.
Arguments: None.
Precondition: The UIvalue has values.
Postcondition: The text box refrences are set.
Protection: Public.*/

UIStatGroup.cs

/*Prototype: public void Refresh();
Description: Populates the UIStats field and calls refresh on all the UIStats
Arguments: A character.
Precondition: A character and a global uiStats.
Postcondition: uiStats are set and then refreshed.
Protection: Public.*/

3. Source Code

Character.cs

```
using System.Collections.Generic;

namespace CharacterCreator
{
   public class Character
   {
      public int Experience; //Amount of experience the character has.
      public string Job; //Name of character's class
      public int Level; //Current level of character. Goes up by 1 when the character gets
100 experience.
      public int Move; //Number of spaces this character can move.
      public string Name; //Name of the character.
      public List<Stat> Stats; //Combination of this character's base stats, the class base
stats,and any stat increases from level ups.
      public Character()
      {
         Name = "Default";
         Job = "Class";
         Stats = new List<Stat>();
         Level = 1;
         Experience = 0;
         Move = 0;
         CreateStats(Stats);
      }
      public Character(string name)
      {
         Name = name;
         Job = "Class";
         Stats = new List<Stat>();
         Level = 1;
         Experience = 0;
         Move = 0;
         CreateStats(Stats);
      }
      public void CreateStats(List<Stat> stats)
      {
         stats.Add(new Stat("HP", 0, 0, 0, ""));
         stats.Add(new Stat("Str", 0, 0, 0, ""));
         stats.Add(new Stat("Mag", 0, 0, 0, ""));
         stats.Add(new Stat("Skill", 0, 0, 0, ""));
```

```
            stats.Add(new Stat("Spd", 0, 0, 0, ""));
            stats.Add(new Stat("Luck", 0, 0, 0, ""));
            stats.Add(new Stat("Def", 0, 0, 0, ""));
            stats.Add(new Stat("Res", 0, 0, 0, ""));
        }
    }
}
```

Form1.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Net.NetworkInformation;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using Newtonsoft.Json;

namespace CharacterCreator
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private Character _currentCharacter;

        private Character CurrentCharacter
        {
            get { return _currentCharacter; }
            set
            {
                _currentCharacter = new Character();
                _currentCharacter = value;
                if(onCharacterChanged != null)
                    onCharacterChanged.Invoke(_currentCharacter);
            }
        }
```

```
        public delegate void OnCharacterChanged(Character c);

        public OnCharacterChanged onCharacterChanged;

        public int CurrentIndex
        {
            get { return _currentIndex; }
            set
            {
                _currentIndex = value;
                if (_currentIndex > loadedCharacters.Count)
                    _currentIndex = loadedCharacters.Count - 1;
                if (_currentIndex <= 0)
                    _currentIndex = 0;
            }
        }
        private void UiRefresh(Character c)
        {
            for (int i = 0; i < uiStats.Count; i++)
                uiStats[i].StatRef = c.Stats[i];

            uiStats.ForEach(s => s.Refresh());
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            loadedCharacters = new List<Character>();
            uiStats = new List<UIStat>();

            onCharacterChanged += UiRefresh;
            onCharacterChanged += SetBoxes;

            var path = System.IO.Path.Combine(System.Environment.CurrentDirectory,
"Characters");
            var files = Directory.GetFiles(path);

            foreach (var f in files)
            {
                var jsonstring = File.ReadAllText(f);
                var character = JsonConvert.DeserializeObject<Character>(jsonstring);
                loadedCharacters.Add(character);
            }
```

```
        foreach (var c in loadedCharacters)
            listBox1.Items.Add(c.Name);

        ///----------GROUP TEXTBOXES///
        var hpgroup = new UIValueGrowthMax { ValueRef = hpBox, GrowthRef =
hpGrowthBox, MaxRef = maxHpBox };
        var strgroup = new UIValueGrowthMax { ValueRef = strBox, GrowthRef =
strGrowthBox, MaxRef = maxStrBox };
        var maggroup = new UIValueGrowthMax { ValueRef = magBox, GrowthRef =
magGrowthBox, MaxRef = maxMagBox };
        var skillgroup = new UIValueGrowthMax { ValueRef = skillBox, GrowthRef =
skillGrowthBox, MaxRef = maxSkillBox };
        var spdgroup = new UIValueGrowthMax { ValueRef = spdBox, GrowthRef =
spdGrowthBox, MaxRef = maxSpdBox };
        var luckgroup = new UIValueGrowthMax { ValueRef = luckBox, GrowthRef =
luckGrowthBox, MaxRef = maxLuckBox };
        var defgroup = new UIValueGrowthMax { ValueRef = defBox, GrowthRef =
defGrowthBox, MaxRef = maxDefBox };
        var resgroup = new UIValueGrowthMax { ValueRef = resBox, GrowthRef =
resGrowthBox, MaxRef = maxResBox };

        ///----------STUFF UI IN A LIST/////
        uiStats.Add(new UIStat(new Stat("HP", 0, 0, 0, ""), hpgroup));
        uiStats.Add(new UIStat(new Stat("Str", 0, 0, 0, ""), strgroup));
        uiStats.Add(new UIStat(new Stat("Mag", 0, 0, 0, ""), maggroup));
        uiStats.Add(new UIStat(new Stat("Skill", 0, 0, 0, ""), skillgroup));
        uiStats.Add(new UIStat(new Stat("Spd", 0, 0, 0, ""), spdgroup));
        uiStats.Add(new UIStat(new Stat("Luck", 0, 0, 0, ""), luckgroup));
        uiStats.Add(new UIStat(new Stat("Def", 0, 0, 0, ""), defgroup));
        uiStats.Add(new UIStat(new Stat("Res", 0, 0, 0, ""), resgroup));

        if (loadedCharacters.Capacity == 0)
        {
            var one = new Character("Default");
            listBox1.Items.Add(one.Name);
            loadedCharacters.Add(one);
            listBox1.SelectedIndex = CurrentIndex;
        }

        CurrentCharacter = loadedCharacters[CurrentIndex];
        listBox1.SelectedIndexChanged += new System.EventHandler(Listchanged);
```

```
        }

        public void SetBoxes(Character c)
        {
            nameBox.Text = c.Name;
            classNameBox.Text = c.Job;
            expValueBox.Text = c.Experience.ToString();
            lvlValueBox.Text = c.Level.ToString();
            moveValueBox.Text = c.Move.ToString();
        }
        public void SetOtherValues(Character c)
        {
            c.Name = nameBox.Text;
            c.Job = classNameBox.Text;
            c.Experience = int.Parse(expValueBox.Text);
            c.Level = int.Parse(lvlValueBox.Text);
            c.Move = int.Parse(moveValueBox.Text);
        }
        private void Listchanged(object sender, EventArgs e)
        {
            SetOtherValues(CurrentCharacter);
            uiStats.ForEach(s => s.SetValues());
            var listbox = sender as ListBox;
            CurrentIndex = listBox1.SelectedIndex;
            CurrentCharacter = loadedCharacters[CurrentIndex];
        }

        private void saveButton_Click(object sender, EventArgs e)
        {
            SetOtherValues(CurrentCharacter);
            uiStats.ForEach(s => s.SetValues());
            foreach (var c in loadedCharacters)
            {
                SetOtherValues(CurrentCharacter);
                var jsonstring = JsonConvert.SerializeObject(c);
                var path = System.IO.Path.Combine(System.Environment.CurrentDirectory,
"Characters/" + c.Name + ".json");
                File.WriteAllText(path, jsonstring);
            }
            var otherpath = System.IO.Path.Combine(System.Environment.CurrentDirectory,
"Characters");
            var files = Directory.GetFiles(otherpath);
```

```
            loadedCharacters = new List<Character>();
            listBox1.Items.Clear();
            foreach (var f in files)
            {
                var jsonstring = File.ReadAllText(f);
                var character = JsonConvert.DeserializeObject<Character>(jsonstring);
                loadedCharacters.Add(character);
            }
            foreach (var c in loadedCharacters)
                listBox1.Items.Add(c.Name);
        }

        private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
        {

        }
    }
}
```

Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace CharacterCreator
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

Stat.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace CharacterCreator
{
    public class Stat
    {
        public string Name;
        public int Value;
        public int Growth;
        public int Max;
        public string Description;
        public Stat()
        {

        }
        public Stat(string name, int value, int growth, int max)
        {
            Name = name;
            Value = value;
            Max = max;
            Growth = growth;
        }
        public Stat(string name, int value, int growth, int max, string description)
        {
            Name = name;
            Value = value;
            Growth = growth;
            Max = max;
            Description = description;
        }
    }
}
```

UIStat.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
namespace CharacterCreator
{
    public class UIStat
    {
        public UIStat(Stat statref, UIValueGrowthMax vgm)
        {
            StatRef = statref;
            uivalue = vgm;
        }
        public Stat StatRef;
        public UIValueGrowthMax uivalue;

        public void Refresh()
        {
            uivalue.ValueRef.Text = StatRef.Value.ToString();
            uivalue.GrowthRef.Text = StatRef.Growth.ToString();
            uivalue.MaxRef.Text = StatRef.Max.ToString();
        }

        public void SetValues()
        {
            StatRef.Value = int.Parse(uivalue.ValueRef.Text);
            StatRef.Growth = int.Parse(uivalue.GrowthRef.Text);
            StatRef.Max = int.Parse(uivalue.MaxRef.Text);

        }
    }
}
```

UIStatGroup.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace CharacterCreator
{
    public class UIStatGroup
    {
        UIValueGrowthMax[] vgms = new UIValueGrowthMax[9];
        Stat[] StatsRef = new Stat[9];
        List<UIStat> UIStats;
```

```
        public void Refresh()
        {
            for (int i = 0; i < StatsRef.Length; i++)
                UIStats.Add(new UIStat(StatsRef[i], vgms[i]));
            foreach (var stat in UIStats)
            {
                stat.Refresh();
            }
        }
    }
}
```

UIValueGrowthMax.cs

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

namespace CharacterCreator
{
    public class UIValueGrowthMax
    {
        public TextBox ValueRef;
        public TextBox GrowthRef;
        public TextBox MaxRef;
    }
}
```

4. Read Me

Open charactercreator.exe to run the program.
Fill out the textboxes to edit a characters stats.
You can select your character from the list in the top left.
If there are no characters in the characters folder the only character that will show up in the list is the default character.
Press the "Save" button to save what you've created to the characters folder.
If you want to delete a character you can delete their corresponding file in the characters folder. ex If I wanted to delete my character named "Marth" I'd find the file by the same name in the characters
folder and delete.

If the program crashes or doesn't work make sure there is a "characters" folder in the same folder as the exe.

Repositories

Character Creator: https://github.com/ZacharyBriggs/CharacterCreator

IntroToC#: