/\*

\* Name: Adrian Norris

\* Date: 2/26/2013

\* Class and Teacher: Games and Virtual Environments, Dr. Clint Jeffery

\* Task: Homework #3 - Mob Duel Game v 1.5

\* \*/

//Jump points

//combatwork : Work on the various combat systems

//messagework : Work on the combat message system

//optimizework : Work optimizing the code

//mapwork : Work on interchanging maps

//displaywork : Work on displaying weapon, armor, and spell attributes

//spellwork : Work on new spells

//skillwork : Work on new skill system (doesn't include in game functionality)

//inventorywork : Work on the player's inventory system

//itemwork : Work on placing items and elements on map

//questwork : Work on the journal and quest functionality

//levelwork : Work on the PC leveling system

//keywork : Work on new input keys (in game functionality)

//morecombatwork : More work on the various combat systems

//storywork : Work on storyline elements

//UNFINISHED

//Current things to work on:

//-Solidifying where attacks take place (last position or new position?)

//-Finish fleshing out combat messages (current turn, actions left, etc.)

//-Fix focus issues

//-Change picture box, text box, label, etc from default names.

//-PC class rework

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

using System.IO;

using System;

using System.Collections.Generic;

namespace WindowsFormsApplication1

{

public partial class Form1 : Form

{

//Combat engine, random, and time generator globals

int[] currentQuests = new int[2]; //questwork

int[] gc1Storyline;

int[] gc2Storyline;

CombatEngine CE = new CombatEngine();

InventorySystem pcInventory = new InventorySystem(); //inventorywork

Random toHitRand = new Random();

Random trashTalkRand = new Random();

Random lockPickRand = new Random();

Random disableTrapRand = new Random();

Timer wanderTimer = new Timer();

//Drawing globals

Font myFont = new Font("Courier New", 14);

Font textBoxFont = new Font("Courier New", 10);

Pen myPen = new Pen(Brushes.Black);

//Map reader globals

string currentMapStr;

char[] currentMapChars;

StreamReader currentMapSR;

int currentMapFlag;

//mapwork

string currentMapName = "C:\\Users\\Admin\\School\\Adrian's School Work\\Spring 2013\\Games and Virtual Environments\\Homework 2\\Goblin Caves.txt";

//itemwork

BaseItem[] gc1ItemArray;

int[] gc1ItemLoc;

EnvironmentObject[] gc1EnvirArray;

//itemwork

BaseItem[] gc2ItemArray;

int[] gc2ItemLoc;

EnvironmentObject[] gc2EnvirArray;

//Default position

int pcPos = 67;

int mobPos = 51 + (23 \* 66);

PlayerCharacter player = new PlayerCharacter();

MonsterClass goblin = new MonsterClass("Goblin", 1, 51 + (23 \* 66));

//Battle related global declarations

int battleFlag = 0;

int pcTurn = 0;

//int pcCanMove = 0;

int pcMoved = 0;

int pcToAct = 0;

int pcActed = 0;

int attackCommand = 0;

int spellCommand = 0;

//int messageBS = 0; optimizework

//int messageTT = 0; optimizework

int[] messageRepeat = new int[3];

//Weapons, armor, spells, etc. global declarations

Weapon dagger;

Weapon staff;

Weapon sword;

Weapon noWeapon;

Weapon curWeapon;

Armor clothRobes;

Armor lthrRobes;

Armor noArmor;

Armor curArmor;

Consumable berry;

Consumable noConsumable;

UtilitySpell rust; //spellwork

HealingSpell minMendWounds;

OffensiveSpell waveOfPain;

Spell noSpell;

Skill pickLock; //skillwork

Skill disableTrap;

Skill noSkill;

//itemwork

TreasureChest chest1;

TreasureChest chest2;

Trap bearTrapM11;

Trap bearTrapM12;

Trap bearTrapM13;

Trap bearTrapM21;

int initialPaint;

int noKeyInput;

int storylineCounter;

//Initializes various components (weapons, items, etc)

//Sets up certain aspects of the GUI (borders, text box properties, etc.)

//Also loads up the default map

public Form1()

{

InitializeComponent();

this.BackColor = Color.Gray;

textBox1.Font = textBoxFont;

textBox1.BorderStyle = BorderStyle.Fixed3D;

textBox1.ScrollBars = ScrollBars.Vertical;

textBox1.WordWrap = true;

textBox1.ReadOnly = true;

textBox1.Text = "Welcome to Natural Exile.";

textBox2.Font = textBoxFont;

textBox2.BorderStyle = BorderStyle.Fixed3D;

textBox2.ScrollBars = ScrollBars.Vertical;

textBox2.WordWrap = true;

textBox2.ReadOnly = true;

textBox2.Text = "You are safe.";

pictureBox1.BorderStyle = BorderStyle.Fixed3D;

pictureBox2.BorderStyle = BorderStyle.Fixed3D;

pictureBox3.BorderStyle = BorderStyle.Fixed3D;

pictureBox4.BorderStyle = BorderStyle.Fixed3D;

pictureBox5.BorderStyle = BorderStyle.Fixed3D;

pictureBox6.BorderStyle = BorderStyle.Fixed3D;

pictureBox7.BorderStyle = BorderStyle.Fixed3D;

pictureBox8.BorderStyle = BorderStyle.Fixed3D;

pictureBox9.BorderStyle = BorderStyle.Fixed3D;

pictureBox10.BorderStyle = BorderStyle.Fixed3D;

label1.ForeColor = Color.White;

label1.Text = "Current Weapon";

label2.ForeColor = Color.White;

label2.Text = "Current Armor";

label3.ForeColor = Color.White;

label3.Text = "Current Spell";

label4.ForeColor = Color.White;

label4.Text = "Environment Log";

label5.ForeColor = Color.White;

label5.Text = "Combat Log";

label6.ForeColor = Color.White;

label6.Text = "Exile Stats";

label7.ForeColor = Color.White;

label7.Text = "Consumables";

label8.ForeColor = Color.White;

label8.Text = "Inventory";

label9.ForeColor = Color.White;

label9.Text = "Selected Weapon";

label10.ForeColor = Color.White;

label10.Text = "Selected Armor";

label11.ForeColor = Color.White;

label11.Text = "Selected Consumable";

label12.ForeColor = Color.White;

label12.Text = "Selected Spell";

label13.ForeColor = Color.White;

label13.Text = "Selected Skill";

label14.ForeColor = Color.White;

label14.Text = "Coin Purse";

label15.ForeColor = Color.White;

label15.Text = "Journal";

label16.ForeColor = Color.White;

label16.Text = "Current Skill";

comboBox1.DropDownStyle = ComboBoxStyle.DropDownList;

comboBox1.Items.Add("None");

comboBox1.Items.Add("Dagger");

comboBox1.Items.Add("Staff");

comboBox1.SelectedIndex = 1;

dagger = new Weapon("Dagger", 3, 1, 20, 5, 1);

staff = new Weapon("Staff", 2, 3, 15, 20, 1);

sword = new Weapon("Sword", 5, 1, 30, 10, 1); //itemwork

noWeapon = new Weapon("None", 0, 0, 0, 0, 0);

curWeapon = dagger;

//inventorywork

pcInventory.AddToWeapon(dagger);

pcInventory.AddToWeapon(staff);

comboBox2.DropDownStyle = ComboBoxStyle.DropDownList;

comboBox2.Items.Add("None");

comboBox2.Items.Add("Cloth Robes");

comboBox2.SelectedIndex = 1;

clothRobes = new Armor("Cloth Robes", 2, 2);

lthrRobes = new Armor("Lthr Robes", 3, 3);

noArmor = new Armor("None", 0, 0);

curArmor = clothRobes;

//inventorywork

pcInventory.AddToArmor(clothRobes);

comboBox3.DropDownStyle = ComboBoxStyle.DropDownList;

comboBox3.Items.Add("None");

comboBox3.Items.Add("Wave of Pain");

comboBox3.Items.Add("Minor Mend Wounds");

comboBox3.Items.Add("Rust");

comboBox3.SelectedIndex = 0;

waveOfPain = new OffensiveSpell("Wave of Pain", 7, 4, 3, 0, 1, 0, 0);

minMendWounds = new HealingSpell("Minor Mend Wounds", 5, 's', 7); //spellwork

rust = new UtilitySpell("Rust", 10);

noSpell = new OffensiveSpell("None", 0, 0, 0, 0, 0, 0, 0);

comboBox4.DropDownStyle = ComboBoxStyle.DropDownList;

comboBox4.Items.Add("None");

comboBox4.SelectedIndex = 0;

noConsumable = new Consumable("None", 0, 0, 0);

berry = new Consumable("Berry", 3, 5, 5);

button1.Text = "Use Consumable";

//skillwork

comboBox5.DropDownStyle = ComboBoxStyle.DropDownList;

comboBox5.Items.Add("None");

comboBox5.Items.Add("Pick Lock");

comboBox5.Items.Add("Disable Trap");

comboBox5.SelectedIndex = 0;

//skillwork

pickLock = new Skill("Pick Lock");

disableTrap = new Skill("Disable Trap");

noSkill = new Skill("None");

//inventorywork

pcInventory.AddToCoinPurse(0, 0, 5);

//questwork

currentQuests[0] = 1;

//itemwork

Weapon[] tempW = { sword };

Armor[] tempA = new Armor[1];

Consumable[] tempC = { berry };

chest1 = new TreasureChest("Chest 1", 3161, true, false, tempW, tempA, tempC);

//itemwork

tempW = new Weapon[1];

tempA[0] = lthrRobes;

tempC = new Consumable[1];

chest2 = new TreasureChest("Chest 2", 743, true, false, tempW, tempA, tempC);

//itemwork

bearTrapM11 = new Trap("Bear Trap 1", 3243, 5);

bearTrapM12 = new Trap("Bear Trap 2", 3244, 5);

bearTrapM13 = new Trap("Bear Trap 3", 3245, 5);

bearTrapM21 = new Trap("Bear Trap 1", 2218, 5);

//itemwork

gc1ItemArray = new BaseItem[100];

gc1ItemLoc = new int[100];

gc1EnvirArray = new EnvironmentObject[] { chest1, bearTrapM11, bearTrapM12, bearTrapM13 };

gc2ItemArray = new BaseItem[100];

gc2ItemLoc = new int[100];

gc2EnvirArray = new EnvironmentObject[] { chest2, bearTrapM21 };

gc1Storyline = new int[6];

gc2Storyline = new int[2];

this.KeyPreview = true;

this.KeyPress += new KeyPressEventHandler(Form1\_KeyPress);

this.KeyDown += new KeyEventHandler(Form1\_KeyDown);

initialPaint = 0;

noKeyInput = 0;

storylineCounter = 0;

InitializePlayerItems();

currentMapFlag = 0;

LoadCurrentMap();

wanderTimer.Tick += new EventHandler(WanderTimer\_Tick);

wanderTimer.Interval = (1000) \* (3);

wanderTimer.Enabled = true;

wanderTimer.Start();

}

//Loads the goblin in to the mob list (for combat)

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

{

CE.MobList[0] = goblin.mobID;

this.Invalidate();

}

//Initializes base + weapon or armor stats at the start of a game

//or when a new map is loaded

private void InitializePlayerItems()

{

player.meleeDmg2 += curWeapon.meleeDmg;

if (player.meleeDmg2 >= 10)

{

player.meleeDmg1 += player.meleeDmg2 / 10;

player.meleeDmg2 = player.meleeDmg2 % 10;

}

player.magicDmg2 += curWeapon.magicDmg;

if (player.magicDmg2 >= 10)

{

player.magicDmg1 += player.magicDmg2 / 10;

player.magicDmg2 = player.magicDmg2 % 10;

}

player.meleeHit2 += curWeapon.meleeHit;

if (player.meleeHit2 >= 10)

{

player.meleeHit1 += player.meleeHit2 / 10;

player.meleeHit2 = player.meleeHit2 % 10;

}

player.magicHit2 += curWeapon.magicHit;

if (player.magicHit2 >= 10)

{

player.magicHit1 += player.magicHit2 / 10;

player.magicHit2 = player.magicHit2 % 10;

}

player.armor2 += curArmor.prot;

if (player.armor2 >= 10)

{

player.armor1 += player.armor2 / 10;

player.armor2 = player.armor2 % 10;

}

player.magicResist2 += curArmor.mResist;

if (player.magicResist2 >= 10)

{

player.magicResist1 += player.magicResist2 / 10;

player.magicResist2 = player.magicResist2 % 10;

}

}

//Loads up the current map file

private void LoadCurrentMap()

{ //mapwork

//if (currentMapFlag == 0)

//{

//currentMapSR = new StreamReader("C:\\Users\\Admin\\School\\Adrian's School Work\\Spring 2013\\Games and Virtual Environments\\Homework 2\\Goblin Caves.txt");

currentMapSR = new StreamReader(currentMapName);

//}

currentMapStr = currentMapSR.ReadToEnd();

currentMapChars = currentMapStr.ToCharArray();

}

//mapwork

private void ChangeCurrentMap()

{

if (currentMapFlag == 0)

{

currentMapFlag = 1;

currentMapName = "C:\\Users\\Admin\\School\\Adrian's School Work\\Spring 2013\\Games and Virtual Environments\\Homework 2\\Goblin Caves 2.txt";

pcPos = 133;

}

else if (currentMapFlag == 1)

{

currentMapFlag = 0;

currentMapName = "C:\\Users\\Admin\\School\\Adrian's School Work\\Spring 2013\\Games and Virtual Environments\\Homework 2\\Goblin Caves.txt";

pcPos = 3625;

}

this.Invalidate();

}

//Returns the currently selected weapon, or none if there isn't one

private Weapon CurrentWeapon()

{

string selectedWeapon = comboBox1.Items[comboBox1.SelectedIndex].ToString();

if (selectedWeapon.CompareTo("Dagger") == 0)

return dagger;

else if (selectedWeapon.CompareTo("Staff") == 0)

return staff;

else if (selectedWeapon.CompareTo("Sword") == 0)

return sword;

else

return noWeapon;

}

//Returns the currently selected armor set, or none if there isn't one

private Armor CurrentArmor()

{

string selectedArmor = comboBox2.Items[comboBox2.SelectedIndex].ToString();

if (selectedArmor.CompareTo("Cloth Robes") == 0)

return clothRobes;

else if (selectedArmor.CompareTo("Lthr Robes") == 0)

return lthrRobes;

else

return noArmor;

}

//Returns the currently selected consumable, or none if there isn't one

private Consumable CurrentConsumable()

{

string selectedConsumable = comboBox4.Items[comboBox4.SelectedIndex].ToString();

if (selectedConsumable.CompareTo("Berry") == 0)

return berry;

else

return noConsumable;

}

//Returns the currently selected spell, or none if there isn't one

private OffensiveSpell CurrentOffensiveSpell()

{

string selectedSpell = comboBox3.Items[comboBox3.SelectedIndex].ToString();

//if (selectedSpell.CompareTo("Wave of Pain") == 0)

return waveOfPain;

//else

//return noSpell;

}

//keywork

private HealingSpell CurrentHealingSpell()

{

string selectedSpell = comboBox3.Items[comboBox3.SelectedIndex].ToString();

return minMendWounds;

}

//keywork

private UtilitySpell CurrentUtilitySpell()

{

string selectedSpell = comboBox3.Items[comboBox3.SelectedIndex].ToString();

return rust;

}

//keywork displaywork

private Skill CurrentSkill()

{

string selectedSkill = comboBox5.Items[comboBox5.SelectedIndex].ToString();

if (selectedSkill.CompareTo("Pick Lock") == 0)

return pickLock;

else if (selectedSkill.CompareTo("Disable Trap") == 0)

return disableTrap;

else

return noSkill;

}

//displaywork

private Spell CurrentSpell()

{

string selectedSpell = comboBox3.Items[comboBox3.SelectedIndex].ToString();

if (selectedSpell.CompareTo("Wave of Pain") == 0)

return waveOfPain;

else if (selectedSpell.CompareTo("Minor Mend Wounds") == 0)

return minMendWounds;

else if (selectedSpell.CompareTo("Rust") == 0)

return rust;

else

return noSpell;

}

//Function for determining the total (base + weapon or armor)

//for each of the character's attributes

private void CalcStats()

{

Weapon newWeapon = CurrentWeapon();

Armor newArmor = CurrentArmor();

player.ResetStats();

player.meleeDmg2 += newWeapon.meleeDmg;

if (player.meleeDmg2 >= 10)

{

player.meleeDmg1 += player.meleeDmg2 / 10;

player.meleeDmg2 = player.meleeDmg2 % 10;

}

player.magicDmg2 += newWeapon.magicDmg;

if (player.magicDmg2 >= 10)

{

player.magicDmg1 += player.magicDmg2 / 10;

player.magicDmg2 = player.magicDmg2 % 10;

}

player.meleeHit2 += newWeapon.meleeHit;

if (player.meleeHit2 >= 10)

{

player.meleeHit1 += player.meleeHit2 / 10;

player.meleeHit2 = player.meleeHit2 % 10;

}

player.magicHit2 += curWeapon.magicHit;

if (player.magicHit2 >= 10)

{

player.magicHit1 += player.magicHit2 / 10;

player.magicHit2 = player.magicHit2 % 10;

}

curWeapon = newWeapon;

player.armor2 += newArmor.prot;

if (player.armor2 >= 10)

{

player.armor1 += player.armor2 / 10;

player.armor2 = player.armor2 % 10;

}

player.magicResist2 += newArmor.mResist;

if (player.magicResist2 >= 10)

{

player.magicResist1 += player.magicResist2 / 10;

player.magicResist2 = player.magicResist2 % 10;

}

curArmor = newArmor;

}

//Puts each stat number in it's respective place

private void PlaceStats(char[] statChars)

{

statChars[63] = GetChar(player.currentLevel1);

statChars[64] = GetChar(player.currentLevel2);

statChars[90] = GetChar(player.hitPoints1);

statChars[91] = GetChar(player.hitPoints2);

statChars[108] = GetChar(player.mana1);

statChars[109] = GetChar(player.mana2);

statChars[129] = GetChar(player.pcStrength1);

statChars[130] = GetChar(player.pcStrength2);

statChars[147] = GetChar(player.pcDexterity1);

statChars[148] = GetChar(player.pcDexterity2);

statChars[168] = GetChar(player.pcEndurance1);

statChars[169] = GetChar(player.pcEndurance2);

statChars[186] = GetChar(player.pcWillpower1);

statChars[187] = GetChar(player.pcWillpower2);

statChars[207] = GetChar(player.meleeDmg1);

statChars[208] = GetChar(player.meleeDmg2);

statChars[225] = GetChar(player.magicDmg1);

statChars[226] = GetChar(player.magicDmg2);

statChars[246] = GetChar(player.armor1);

statChars[247] = GetChar(player.armor2);

statChars[264] = GetChar(player.magicResist1);

statChars[265] = GetChar(player.magicResist2);

statChars[285] = GetChar(player.meleeHit1);

statChars[286] = GetChar(player.meleeHit2);

statChars[303] = GetChar(player.magicHit1);

statChars[304] = GetChar(player.magicHit2);

statChars[324] = GetChar(player.exp1);

statChars[325] = GetChar(player.exp2);

statChars[342] = GetChar(player.toLevel1);

statChars[343] = GetChar(player.toLevel2);

}

//Loads/reloads the PC's statistics page

private void LoadStatPage()

{

Bitmap DrawArea\_StatsPage;

Graphics statG;

DrawArea\_StatsPage = new Bitmap(pictureBox2.Size.Width, pictureBox2.Size.Height);

pictureBox2.Image = DrawArea\_StatsPage;

statG = Graphics.FromImage(DrawArea\_StatsPage);

statG.Clear(Color.White);

StreamReader statR = new StreamReader("C:\\Users\\Admin\\School\\Adrian's School Work\\Spring 2013\\Games and Virtual Environments\\Homework 2\\StatsPage.txt");

string statStr = statR.ReadToEnd();

char[] statChars = statStr.ToCharArray();

CalcStats();

PlaceStats(statChars);

statStr = new string(statChars);

statG.DrawString(statStr, myFont, Brushes.Black, new Point(0, 0));

statR.Close();

statG.Dispose();

}

//inventorywork

private void LoadInventoryPage()

{

Bitmap DrawArea\_Inventory;

Graphics inventoryG;

DrawArea\_Inventory = new Bitmap(pictureBox7.Size.Width, pictureBox7.Size.Height);

pictureBox7.Image = DrawArea\_Inventory;

inventoryG = Graphics.FromImage(DrawArea\_Inventory);

inventoryG.Clear(Color.White);

StreamReader inventoryR = new StreamReader("C:\\Users\\Admin\\School\\Adrian's School Work\\Spring 2013\\Games and Virtual Environments\\Homework 2\\Inventory.txt");

string inventoryStr = inventoryR.ReadToEnd();

char[] inventoryChars = inventoryStr.ToCharArray();

AddInventoryItems(inventoryChars);

inventoryStr = new string(inventoryChars);

inventoryG.DrawString(inventoryStr, myFont, Brushes.Black, new Point(0, 0));

inventoryR.Close();

inventoryG.Dispose();

}

//inventorywork

private void AddInventoryItems(char[] chars)

{

char[] temp;

int wBase = 103, aBase = 113, cBase = 125;

int i, j;

for (i = 0; i < pcInventory.wCounter; i++)

{

temp = pcInventory.weaponInventory[i].name.ToCharArray();

for (j = 0; j < temp.Length; j++)

{

chars[wBase + (i \* 34) + j] = temp[j];

}

}

for (i = 0; i < pcInventory.aCounter; i++)

{

temp = pcInventory.armorInventory[i].name.ToCharArray();

for (j = 0; j < temp.Length; j++)

{

chars[aBase + (i \* 34) + j] = temp[j];

}

}

for (i = 0; i < pcInventory.cCounter; i++)

{

temp = pcInventory.consumableInventory[i].name.ToCharArray();

for (j = 0; j < temp.Length; j++)

{

chars[cBase + (i \* 34) + j] = temp[j];

}

}

}

//inventorywork

private void LoadCoinPursePage()

{

int[] fNums = new int[2];

Bitmap DrawArea\_CoinPurse;

Graphics coinPurseG;

DrawArea\_CoinPurse = new Bitmap(pictureBox9.Size.Width, pictureBox9.Size.Height);

pictureBox9.Image = DrawArea\_CoinPurse;

coinPurseG = Graphics.FromImage(DrawArea\_CoinPurse);

coinPurseG.Clear(Color.White);

StreamReader coinPurseR = new StreamReader("C:\\Users\\Admin\\School\\Adrian's School Work\\Spring 2013\\Games and Virtual Environments\\Homework 2\\Coin Purse.txt");

string coinPurseStr = coinPurseR.ReadToEnd();

char[] coinPurseChars = coinPurseStr.ToCharArray();

SplitInteger(fNums, pcInventory.coinPurse[0]);

coinPurseChars[29] = GetChar(fNums[0]);

coinPurseChars[30] = GetChar(fNums[1]);

SplitInteger(fNums, pcInventory.coinPurse[1]);

coinPurseChars[49] = GetChar(fNums[0]);

coinPurseChars[50] = GetChar(fNums[1]);

SplitInteger(fNums, pcInventory.coinPurse[2]);

coinPurseChars[69] = GetChar(fNums[0]);

coinPurseChars[70] = GetChar(fNums[1]);

coinPurseStr = new string(coinPurseChars);

coinPurseG.DrawString(coinPurseStr, myFont, Brushes.Black, new Point(0, 0));

coinPurseR.Close();

coinPurseG.Dispose();

}

//questwork

private void LoadJournalPage()

{

Bitmap DrawArea\_Journal;

Graphics journalG;

DrawArea\_Journal = new Bitmap(pictureBox10.Size.Width, pictureBox10.Size.Height);

pictureBox10.Image = DrawArea\_Journal;

journalG = Graphics.FromImage(DrawArea\_Journal);

journalG.Clear(Color.White);

StreamReader journalR = new StreamReader("C:\\Users\\Admin\\School\\Adrian's School Work\\Spring 2013\\Games and Virtual Environments\\Homework 2\\Journal.txt");

string journalStr = journalR.ReadToEnd();

char[] journalChars = journalStr.ToCharArray();

AddCurrentQuests(journalChars);

journalStr = new string(journalChars);

journalG.DrawString(journalStr, myFont, Brushes.Black, new Point(0, 0));

journalR.Close();

journalG.Dispose();

}

//questwork

private void AddCurrentQuests(char[] chars)

{

string n, t, r;

char[] name, task, rewards;

int nLoc, tLoc, rLoc, i;

if (currentQuests[0] == 1)

{

n = "Spelunking for an Exit";

name = n.ToCharArray();

t = "Find a way out of these caves.";

task = t.ToCharArray();

r = "20 Exp";

rewards = r.ToCharArray();

nLoc = 42;

tLoc = 76;

rLoc = 146;

for (i = 0; i < name.Length; i++, nLoc++)

{

chars[nLoc] = name[i];

}

for (i = 0; i < task.Length; i++, tLoc++)

{

if (tLoc == 99)

{

tLoc = 109;

}

chars[tLoc] = task[i];

}

for (i = 0; i < rewards.Length; i++, rLoc++)

{

chars[rLoc] = rewards[i];

}

}

else if (currentQuests[0] == 2)

{

n = "Silence the Shaman";

name = n.ToCharArray();

t = "End the Shaman to lower the Gremlin morale.";

task = t.ToCharArray();

r = "10 Exp";

rewards = r.ToCharArray();

nLoc = 42;

tLoc = 76;

rLoc = 146;

for (i = 0; i < name.Length; i++, nLoc++)

{

chars[nLoc] = name[i];

}

for (i = 0; i < task.Length; i++, tLoc++)

{

if (tLoc == 99)

{

tLoc = 109;

}

chars[tLoc] = task[i];

}

for (i = 0; i < rewards.Length; i++, rLoc++)

{

chars[rLoc] = rewards[i];

}

}

else

{

n = "None";

name = n.ToCharArray();

t = "None";

task = t.ToCharArray();

r = "None";

rewards = r.ToCharArray();

nLoc = 42;

tLoc = 76;

rLoc = 146;

for (i = 0; i < name.Length; i++, nLoc++)

{

chars[nLoc] = name[i];

}

for (i = 0; i < task.Length; i++, tLoc++)

{

if (tLoc == 99)

{

tLoc = 109;

}

chars[tLoc] = task[i];

}

for (i = 0; i < rewards.Length; i++, rLoc++)

{

chars[rLoc] = rewards[i];

}

}

if (currentQuests[1] == 1)

{

n = "Spelunking for an Exit";

name = n.ToCharArray();

t = "Find a way out of these caves.";

task = t.ToCharArray();

r = "20 Exp";

rewards = r.ToCharArray();

nLoc = 212;

tLoc = 246;

rLoc = 316;

for (i = 0; i < name.Length; i++, nLoc++)

{

chars[nLoc] = name[i];

}

for (i = 0; i < task.Length; i++, tLoc++)

{

if (tLoc == 99)

{

tLoc = 109;

}

chars[tLoc] = task[i];

}

for (i = 0; i < rewards.Length; i++, rLoc++)

{

chars[rLoc] = rewards[i];

}

}

else if (currentQuests[1] == 2)

{

n = "Silence the Shaman";

name = n.ToCharArray();

t = "End the Shaman to lower the Gremlin morale.";

task = t.ToCharArray();

r = "10 Exp";

rewards = r.ToCharArray();

nLoc = 212;

tLoc = 246;

rLoc = 316;

for (i = 0; i < name.Length; i++, nLoc++)

{

chars[nLoc] = name[i];

}

for (i = 0; i < task.Length; i++, tLoc++)

{

if (tLoc == 269)

{

tLoc = 279;

}

chars[tLoc] = task[i];

}

for (i = 0; i < rewards.Length; i++, rLoc++)

{

chars[rLoc] = rewards[i];

}

}

else

{

n = "None";

name = n.ToCharArray();

t = "None";

task = t.ToCharArray();

r = "None";

rewards = r.ToCharArray();

nLoc = 212;

tLoc = 246;

rLoc = 316;

for (i = 0; i < name.Length; i++, nLoc++)

{

chars[nLoc] = name[i];

}

for (i = 0; i < task.Length; i++, tLoc++)

{

if (tLoc == 269)

{

tLoc = 279;

}

chars[tLoc] = task[i];

}

for (i = 0; i < rewards.Length; i++, rLoc++)

{

chars[rLoc] = rewards[i];

}

}

}

//displaywork

private void SplitInteger(int[] fNums, int toSplit)

{

fNums[0] = toSplit / 10;

fNums[1] = toSplit % 10;

}

//displaywork

private void LoadWeaponDescriptor()

{

Bitmap DrawArea\_WeaponDescript;

Graphics wDesG;

DrawArea\_WeaponDescript = new Bitmap(pictureBox3.Size.Width, pictureBox3.Size.Height);

pictureBox3.Image = DrawArea\_WeaponDescript;

wDesG = Graphics.FromImage(DrawArea\_WeaponDescript);

wDesG.Clear(Color.White);

StreamReader wDesR = new StreamReader("C:\\Users\\Admin\\School\\Adrian's School Work\\Spring 2013\\Games and Virtual Environments\\Homework 2\\Weapon Descriptor.txt");

string wDesStr = wDesR.ReadToEnd();

char[] wDesChars = wDesStr.ToCharArray();

AddWeaponDescriptStats(wDesChars);

wDesStr = new string(wDesChars);

wDesG.DrawString(wDesStr, myFont, Brushes.Black, new Point(0, 0));

wDesR.Close();

wDesG.Dispose();

}

//displaywork

private void AddWeaponDescriptStats(char[] chars)

{

Weapon cWeapon = CurrentWeapon();

string cName = cWeapon.name;

char[] name = cName.ToCharArray();

int[] splitNums = new int[2];

int i;

for (i = 0; i < name.Length; i++)

chars[47 + i] = name[i];

SplitInteger(splitNums, cWeapon.range);

chars[66] = GetChar(splitNums[0]);

chars[67] = GetChar(splitNums[1]);

SplitInteger(splitNums, cWeapon.meleeDmg);

chars[91] = GetChar(splitNums[0]);

chars[92] = GetChar(splitNums[1]);

SplitInteger(splitNums, cWeapon.meleeHit);

chars[130] = GetChar(splitNums[0]);

chars[131] = GetChar(splitNums[1]);

SplitInteger(splitNums, cWeapon.magicDmg);

chars[105] = GetChar(splitNums[0]);

chars[106] = GetChar(splitNums[1]);

SplitInteger(splitNums, cWeapon.magicHit);

chars[144] = GetChar(splitNums[0]);

chars[145] = GetChar(splitNums[1]);

}

//displaywork

private void LoadArmorDescriptor()

{

Bitmap DrawArea\_ArmorDescript;

Graphics aDesG;

DrawArea\_ArmorDescript = new Bitmap(pictureBox4.Size.Width, pictureBox4.Size.Height);

pictureBox4.Image = DrawArea\_ArmorDescript;

aDesG = Graphics.FromImage(DrawArea\_ArmorDescript);

aDesG.Clear(Color.White);

StreamReader aDesR = new StreamReader("C:\\Users\\Admin\\School\\Adrian's School Work\\Spring 2013\\Games and Virtual Environments\\Homework 2\\Armor Descriptor.txt");

string aDesStr = aDesR.ReadToEnd();

char[] aDesChars = aDesStr.ToCharArray();

AddArmorDescriptStats(aDesChars);

aDesStr = new string(aDesChars);

aDesG.DrawString(aDesStr, myFont, Brushes.Black, new Point(0, 0));

aDesR.Close();

aDesG.Dispose();

}

//displaywork

private void AddArmorDescriptStats(char[] chars)

{

Armor cArmor = CurrentArmor();

string aName = cArmor.name;

char[] name = aName.ToCharArray();

int[] splitNums = new int[2];

int i;

for (i = 0; i < name.Length; i++)

chars[55 + i] = name[i];

SplitInteger(splitNums, cArmor.prot);

chars[102] = GetChar(splitNums[0]);

chars[103] = GetChar(splitNums[1]);

SplitInteger(splitNums, cArmor.mResist);

chars[141] = GetChar(splitNums[0]);

chars[142] = GetChar(splitNums[1]);

}

//displaywork

private void LoadConsumableDescriptor()

{

Bitmap DrawArea\_ConsumeDescript;

Graphics cDesG;

DrawArea\_ConsumeDescript = new Bitmap(pictureBox8.Size.Width, pictureBox8.Size.Height);

pictureBox8.Image = DrawArea\_ConsumeDescript;

cDesG = Graphics.FromImage(DrawArea\_ConsumeDescript);

cDesG.Clear(Color.White);

StreamReader cDesR = new StreamReader("C:\\Users\\Admin\\School\\Adrian's School Work\\Spring 2013\\Games and Virtual Environments\\Homework 2\\Consumable Descriptor.txt");

string cDesStr = cDesR.ReadToEnd();

char[] cDesChars = cDesStr.ToCharArray();

AddConsumableDescriptStats(cDesChars);

cDesStr = new string(cDesChars);

cDesG.DrawString(cDesStr, myFont, Brushes.Black, new Point(0, 0));

cDesR.Close();

cDesG.Dispose();

}

//displaywork

private void AddConsumableDescriptStats(char[] chars)

{

Consumable cConsume = CurrentConsumable();

string cName = cConsume.name;

char[] name = cName.ToCharArray();

int[] splitNums = new int[2];

int i;

for (i = 0; i < name.Length; i++)

chars[28 + i] = name[i];

SplitInteger(splitNums, cConsume.amount);

chars[50] = GetChar(splitNums[0]);

chars[51] = GetChar(splitNums[1]);

SplitInteger(splitNums, cConsume.upHP);

chars[70] = GetChar(splitNums[0]);

chars[71] = GetChar(splitNums[1]);

SplitInteger(splitNums, cConsume.upMP);

chars[73] = GetChar(splitNums[0]);

chars[74] = GetChar(splitNums[1]);

}

//displaywork

private void LoadSpellDescriptor()

{

Bitmap DrawArea\_SpellDescript;

Graphics spDesG;

DrawArea\_SpellDescript = new Bitmap(pictureBox5.Size.Width, pictureBox5.Size.Height);

pictureBox5.Image = DrawArea\_SpellDescript;

spDesG = Graphics.FromImage(DrawArea\_SpellDescript);

spDesG.Clear(Color.White);

StreamReader spDesR = new StreamReader("C:\\Users\\Admin\\School\\Adrian's School Work\\Spring 2013\\Games and Virtual Environments\\Homework 2\\Spell Descriptor.txt");

string spDesStr = spDesR.ReadToEnd();

char[] spDesChars = spDesStr.ToCharArray();

AddSpellDescriptStats(spDesChars);

spDesStr = new string(spDesChars);

spDesG.DrawString(spDesStr, myFont, Brushes.Black, new Point(0, 0));

spDesR.Close();

spDesG.Dispose();

}

//displaywork

private void AddSpellDescriptStats(char[] chars)

{

Spell cSpell = CurrentSpell();

string cName = cSpell.name, d;

char[] name = cName.ToCharArray(), descript;

int[] splitNums = new int[2];

int i, dNum = 93;

for (i = 0; i < name.Length; i++)

chars[47 + i] = name[i];

if (cName.CompareTo("Wave of Pain") == 0)

{

d = "Send forth a cone of psychic pain.";

descript = d.ToCharArray();

}

else if (cName.CompareTo("Minor Mend Wounds") == 0)

{

d = "Repair a small amount of damaged flesh.";

descript = d.ToCharArray();

}

else if (cName.CompareTo("Rust") == 0)

{

d = "Corrode away a metal object in to rust.";

descript = d.ToCharArray();

}

else

{

d = "None";

descript = d.ToCharArray();

}

for (i = 0; i < descript.Length; i++, dNum++)

{

if (dNum == 114)

dNum = 132;

chars[dNum] = descript[i];

}

SplitInteger(splitNums, cSpell.cost);

chars[72] = GetChar(splitNums[0]);

chars[73] = GetChar(splitNums[1]);

}

//displaywork

private void LoadSkillDescriptor()

{

Bitmap DrawArea\_SkillDescript;

Graphics skDesG;

DrawArea\_SkillDescript = new Bitmap(pictureBox6.Size.Width, pictureBox6.Size.Height);

pictureBox6.Image = DrawArea\_SkillDescript;

skDesG = Graphics.FromImage(DrawArea\_SkillDescript);

skDesG.Clear(Color.White);

StreamReader skDesR = new StreamReader("C:\\Users\\Admin\\School\\Adrian's School Work\\Spring 2013\\Games and Virtual Environments\\Homework 2\\Skill Descriptor.txt");

string skDesStr = skDesR.ReadToEnd();

char[] skDesChars = skDesStr.ToCharArray();

AddSkillDescriptStats(skDesChars);

skDesStr = new string(skDesChars);

skDesG.DrawString(skDesStr, myFont, Brushes.Black, new Point(0, 0));

skDesR.Close();

skDesG.Dispose();

}

//displaywork

private void AddSkillDescriptStats(char[] chars)

{

Skill cSkill = CurrentSkill();

string cName = cSkill.name, d;

char[] name = cName.ToCharArray(), descript;

int i, dNum = 93;

for (i = 0; i < name.Length; i++)

chars[47 + i] = name[i];

if (cName.CompareTo("Pick Lock") == 0)

{

d = "Without using a key, unlock a device.";

descript = d.ToCharArray();

}

else if (cName.CompareTo("Disable Trap") == 0)

{

d = "Render a trap inert.";

descript = d.ToCharArray();

}

else

{

d = "None";

descript = d.ToCharArray();

}

for (i = 0; i < descript.Length; i++, dNum++)

{

if (dNum == 114)

dNum = 132;

chars[dNum] = descript[i];

}

}

//Time for combat

//Checks if victory has been met (genocide win condition),

//either running the next turn or taking the player out of combat

private void BattleMode()

{

if (!(CE.MetVictory()))

{

//Whose turn is it now?

if (CE.FocusedActor() == 0)

{

pcTurn = 0;

//pcCanMove = 1; optimizework

}

else

pcTurn = 1;

//If player's turn, else mob's turn

if (pcTurn == 0)

{

//messageRepeat[2]++; messagework

//CombatText();

//Give the player a moment to act

//System.Threading.Thread.Sleep(1000); combatwork

CombatTurn();

}

else if (pcTurn == 1)

{

RunMobTurn(CE.FocusedActor());

CE.NextTurn();

//pcMoved = 0; combatwork

}

}

else

{

battleFlag = 0;

//messageBS = 0;

//messageTT = 0;

messageRepeat[0] = 0;

}

}

//Change turns if the PC has finished his

private void CombatTurn()

{

if (pcMoved == 1 && pcActed == 1)

{

CE.NextTurn();

pcMoved = 0;

pcActed = 0; //combatwork

}

}

//What a mob does each combat turn

private void RunMobTurn(int mobNum)

{

MobMoveToPC();

MobTakeAction();

}

//Checks the pc's position and moves

//the mob towards it

private void MobMoveToPC()

{

//Variables for the difference in locations

int pcRow = pcPos / 66;

int pcCol = pcPos % 66;

int mobRow = mobPos / 66;

int mobCol = mobPos % 66;

int rowDiff = pcRow - mobRow;

int colDiff = pcCol - mobCol;

//Determine which way to move, then move

if (Math.Abs(rowDiff) > Math.Abs(colDiff))

{

if (rowDiff > 0)

{

MoveMob(goblin, 's');

}

else

{

MoveMob(goblin, 'n');

}

}

else

{

if (colDiff > 0)

{

MoveMob(goblin, 'e');

}

else

{

MoveMob(goblin, 'w');

}

}

}

//Checks if the player is within range of the mob's attacks

//Executes an attack if true

private void MobTakeAction()

{

if ((pcPos - 1) == mobPos || (pcPos + 1) == mobPos || (pcPos - 66) == mobPos || (pcPos + 66) == mobPos)

{

MobMeleeAttack();

}

}

//Function for handling a mob attack on the PC

private void MobMeleeAttack()

{

//PC health and armor, and mob to hit variables

int truePCHP = (player.hitPoints1 \* 10) + player.hitPoints2;

int truePCArmor = (player.armor1 \* 10) + player.armor2;

int finalMobHit = goblin.meleeHit - (2 \* truePCArmor);

int isHit = toHitRand.Next(1, 100);

//If hit

if (finalMobHit >= isHit)

{

//Apply damage

truePCHP -= goblin.meleeDmg;

//Check for death

if (truePCHP <= 0)

{

//Status message followed by closing the program

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("You have perished...");

System.Threading.Thread.Sleep(10000);

this.Close();

}

else

{

player.hitPoints1 = truePCHP / 10;

player.hitPoints2 = truePCHP % 10;

}

}

}

//Prints out combat text constants

//I.E that combat has started

private void CombatText()

{

//if (messageRepeat[0] == 1)//messageBS == 0)

//{

// textBox2.AppendText(Environment.NewLine);

// textBox2.AppendText("A hostile creature has spotted you!");

//}

//else if (messageTT == 0)

//{

// textBox1.AppendText(Environment.NewLine);

// textBox1.AppendText("Goblin: \"Dirty human!\"");

//} messagework

if (messageRepeat[1] == 1)

{

textBox2.AppendText(Environment.NewLine);

textBox2.AppendText("Not your turn...");

}

else if (messageRepeat[2] == 1)

{

textBox2.AppendText(Environment.NewLine);

textBox2.AppendText("Player's Turn...");

}

}

//Helper function for converting a number between

//0 and 9 in to a char

private char GetChar(int num)

{

if (num == 0)

return '0';

else if (num == 1)

return '1';

else if (num == 2)

return '2';

else if (num == 3)

return '3';

else if (num == 4)

return '4';

else if (num == 5)

return '5';

else if (num == 6)

return '6';

else if (num == 7)

return '7';

else if (num == 8)

return '8';

else

return '9';

}

private void InitialCombatText()

{

textBox2.AppendText(Environment.NewLine);

textBox2.AppendText("A hostile has spotted you, prepare for battle!");

}

//Controls the flow in and out of combat

//Also calls up combat text

private void GameFlow()

{

if (battleFlag == 0)

{

DrawCurrentMap();

}

else

{

//messageRepeat[0]++; messagework

//CombatText();

//messageBS = 1;

//CombatText();

//messageTT = 1;

if (messageRepeat[0] == 0)

{

messageRepeat[0] = 1;

InitialCombatText();

}

BattleMode();

}

//StoryProgression();

}

private void PlacePlayerCharacter()

{

currentMapChars[pcPos] = '@';

}

private void PlaceMapMobs()

{

//Goblin dead or alive?

if (goblin.dead == 1)

{

currentMapChars[mobPos] = 'X';

}

else

currentMapChars[mobPos] = '%';

}

private void PlaceActors()

{//mapwork

//PlaceMapMobs();

PlacePlayerCharacter();

}

//itemwork

private void PlaceEnvironObjects()

{

int i;

if (currentMapFlag == 0)

{

for (i = 0; i < gc1EnvirArray.Length; i++)

{

if (gc1EnvirArray[i].Equals(chest1))

{

if (chest1.isLocked)

{

currentMapChars[chest1.loc] = '[';

currentMapChars[chest1.loc + 1] = ']';

}

else

{

currentMapChars[chest1.loc] = '{';

currentMapChars[chest1.loc + 1] = '}';

}

}

else if (gc1EnvirArray[i].Equals(bearTrapM11))

{

currentMapChars[bearTrapM11.loc] = '\*';

}

else if (gc1EnvirArray[i].Equals(bearTrapM12))

{

currentMapChars[bearTrapM12.loc] = '\*';

}

else if (gc1EnvirArray[i].Equals(bearTrapM13))

{

currentMapChars[bearTrapM13.loc] = '\*';

}

}

}

else if (currentMapFlag == 1)

{

for (i = 0; i < gc2EnvirArray.Length; i++)

{

if (gc2EnvirArray[i].Equals(chest2))

{

if (chest2.isLocked)

{

currentMapChars[chest2.loc] = '[';

currentMapChars[chest2.loc + 1] = ']';

}

else

{

currentMapChars[chest2.loc] = '{';

currentMapChars[chest2.loc + 1] = '}';

}

}

else if (gc2EnvirArray[i].Equals(bearTrapM21))

{

currentMapChars[bearTrapM21.loc] = '\*';

}

}

}

}

//Ugly function for loading/drawing the map

//Great place for improvement

private void DrawCurrentMap()

{

//Setting up the view screen

Bitmap DrawArea;

Graphics g;

DrawArea = new Bitmap(pictureBox1.Size.Width, pictureBox1.Size.Height);

pictureBox1.Image = DrawArea;

g = Graphics.FromImage(DrawArea);

g.Clear(Color.White);

LoadCurrentMap(); //Load map from file

//Retrieve actor positions

mobPos = goblin.pos;

player.pos = pcPos;

PlaceEnvironObjects(); //itemwork

PlaceActors();

//Variable for new (trimmed) map

char[] newChars = new char[currentMapChars.Length];

CenterCamera(currentMapChars, newChars);

currentMapStr = new string(newChars);

g.DrawString(currentMapStr, myFont, Brushes.Black, new Point(0, 0));

currentMapSR.Close();

g.Dispose();

//mapwork

//CheckAggro(); //Make sure new char position (if there is one) isn't inside a mob's aggro range

}

//Checks to see if the mob is not dead and if the character is within aggro range

//If so combat starts

private void CheckAggro()

{

if (goblin.dead != 1 && CheckAggroRad(mobPos, goblin.aggroRad))//goblin.pos, goblin.aggroRad))

{

battleFlag = 1;

}

}

//Determines if the character is within the aggro radius of the mob

//Checks each spot individually - place for improvement

private bool CheckAggroRad(int mobPos, int aggroRad)

{

int i = aggroRad \* 66, j = mobPos - i, k = mobPos + i;

int a = mobPos - aggroRad, s = mobPos + aggroRad;

int g = 66;

if (a == pcPos || s == pcPos)

{

return true;

}

else if ((s - g) == pcPos || (s - 2 \* g) == pcPos || (s + g) == pcPos || (s + 2 \* g) == pcPos)

{

return true;

}

else if ((a - g) == pcPos || (a - 2 \* g) == pcPos || (a + g) == pcPos || (a + 2 \* g) == pcPos)

{

return true;

}

else if (j == pcPos || k == pcPos)

{

return true;

}

else if ((j - 1) == pcPos || (j - 2) == pcPos)

{

return true;

}

else if ((j + 1) == pcPos || (j + 2) == pcPos)

{

return true;

}

else if ((k - 1) == pcPos || (k - 2) == pcPos)

{

return true;

}

else if ((k + 1) == pcPos || (k + 2) == pcPos)

{

return true;

}

else

return false;

}

//storywork

private void StoryProgression()

{

//gc1: 1 = cave in, 2 = first dead gremlin, 3 = a chest!, 4 = it's a trap, 5 = shaman quest start, 6 = shaman quest end

//gc2: 1 = the champion, 2 = spelunking quest end

if (currentMapFlag == 0)

{

if (initialPaint == 0)

{

textBox1.AppendText(Environment.NewLine);

textBox1.ForeColor = Color.Green;

textBox1.AppendText("The cave in comes suddenly, your reflexes sending you forward in a sloppy roll. The rocks nearly crushing you under the immense weight the path back has been completely sealed off.");

//System.Threading.Thread.Sleep(1000);

noKeyInput = 1;

initialPaint = 1;

}

else if (pcPos == 67 && gc1Storyline[0] != 1 && storylineCounter == 1)

{

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("You stand and brush yourself off, checking to make sure everything is where it should be.");

System.Threading.Thread.Sleep(2000);

textBox1.AppendText(Environment.NewLine);

textBox1.ForeColor = Color.Blue;

textBox1.AppendText("Exile: The air is thinner here...must be getting close to an exit.");

//gc1Storyline[0] = 1;

//System.Threading.Thread.Sleep(1000);

gc1Storyline[0] = 1;

noKeyInput = 0;

}

else if ((pcPos == 1268 || pcPos == 1269 || pcPos == 1270 || pcPos == 1271) && gc1Storyline[1] != 1)

{

System.Threading.Thread.Sleep(1000);

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("Exile: The air is thinner here...must be getting close to an exit.");

}

}

else if (currentMapFlag == 1)

{

}

}

//Centers the camera on the player's avatar

//Only draws the visible portion of the map

private void CenterCamera(char[] chars, char[] newChars)

{

//Variables for position checking

int pcRow = pcPos / 66;

int pcEdge = pcPos % 66;

int maxRows = chars.Length / 66;

int surpN = 0, surpS = 0, surpW = 0, surpE = 0;

char[,] viewSegment = new char[27, 56];

//Calculate distance from edges

if (pcEdge < 28)

{

surpW = 28 - pcEdge;

}

else if (pcEdge >= 37)

{

surpE = pcEdge - 36;

}

if (pcRow < 13)

{

surpN = 13 - pcRow;

}

else if (maxRows < (pcRow + 13))

{

surpS = (pcRow + 13) - maxRows;

}

if (surpN == 0 && surpS == 0 && surpE == 0 && surpW == 0)

{

int zeros = 0;

for (; zeros < chars.Length; zeros++)

{

newChars[zeros] = chars[zeros];

}

}

//Variables for drawing the exact position of the screen

int i, j, l, k;

int rowMod;

int nDiff = 13 - surpN + surpS;

int sDiff = 13 - surpS + surpN;

int eDiff = 28 - surpE + surpW;

int wDiff = 28 - surpW + surpE;

//Saves only the portion of the screen that is visible

for (i = nDiff, l = 0; i > 0; i--, l++)

{

rowMod = pcPos - (i \* 66);

for (j = rowMod - wDiff, k = 0; j < (rowMod + eDiff); j++, k++)

{

viewSegment[l, k] = chars[j];

}

}

for (i = pcPos - wDiff, l = 0; i < (pcPos + eDiff); i++, l++)

{

viewSegment[nDiff, l] = chars[i];

}

for (i = 1, l = nDiff + 1; i <= sDiff; i++, l++)

{

rowMod = pcPos + (i \* 66);

for (j = rowMod - wDiff, k = 0; j < (rowMod + eDiff); j++, k++)

{

viewSegment[l, k] = chars[j];

}

}

for (i = 0, l = 0; i < 27; i++, l++)

{

for (j = 0; j < 56; j++, l++)

{

newChars[l] = viewSegment[i, j];

}

l++;

newChars[l] = '\n';

}

}

//Function called ever time Invalidate() is used

//Refreshes the whole screen

protected override void OnPaint(PaintEventArgs e)

{

LoadStatPage();

//inventorywork

LoadInventoryPage();

LoadCoinPursePage();

//questwork

LoadJournalPage();

//displaywork

LoadWeaponDescriptor();

LoadArmorDescriptor();

LoadConsumableDescriptor();

LoadSpellDescriptor();

LoadSkillDescriptor();

//StoryProgression();

GameFlow();

base.OnPaint(e);

}

private void Form1\_KeyDown(object sender, KeyEventArgs e)

{

storylineCounter = 1;

if (pcToAct == 1 && noKeyInput != 1) // combatwork

{

PerformAction(sender, e);

}

else

e.Handled = true;

}

//Recieves key press events from the GUI

private void Form1\_KeyPress(object sender, KeyPressEventArgs e)

{

storylineCounter = 1;

//If not in battle move normally

if (battleFlag == 0 && noKeyInput != 1)

{

KeyPressHandler(sender, e);

}

else if(noKeyInput != 1)

{

//In battle: is it the player's turn?

if (pcTurn == 0)

{

messageRepeat[1] = 0;

//KeyPressHandler(sender, e); combatwork

if (pcToAct != 1 && pcMoved != 1 && BattleMovePressHandler(sender, e))

pcMoved = 1;

BattleKeyHandler(sender, e);

//if (pcToAct == 1)

//{

//AttackKeyHandler(sender, e);

//}

}

else //messagework

{

//textBox2.AppendText(Environment.NewLine);

//textBox2.AppendText("Not your turn...");

//messageNotTurn = 0;

//messageRepeat[2] = 0;

messageRepeat[1]++;

CombatText();

}

}

e.Handled = true;

}

// combatwork

private bool BattleMovePressHandler(object sender, KeyPressEventArgs e)

{

int tempPcPos = pcPos;

if (e.KeyChar == 'w')

{

tempPcPos -= 66;

if (CanMove(tempPcPos))

{

pcPos = tempPcPos;

IsCurrentPosTrapped(pcPos);

this.Invalidate();

}

return true;

}

else if (e.KeyChar == 's')

{

tempPcPos += 66;

if (CanMove(tempPcPos))

{

pcPos = tempPcPos;

IsCurrentPosTrapped(pcPos);

this.Invalidate();

}

return true;

}

else if (e.KeyChar == 'a')

{

tempPcPos -= 1;

if (CanMove(tempPcPos))

{

pcPos = tempPcPos;

IsCurrentPosTrapped(pcPos);

this.Invalidate();

}

return true;

}

else if (e.KeyChar == 'd')

{

tempPcPos += 1;

if (CanMove(tempPcPos))

{

pcPos = tempPcPos;

IsCurrentPosTrapped(pcPos);

this.Invalidate();

}

return true;

}

else

{

e.Handled = true;

return false;

}

}

//itemwork

private void IsCurrentPosTrapped(int pos)

{

int i;

if (currentMapFlag == 0)

{

for (i = 0; i < gc1EnvirArray.Length; i++)

{

if (pos == gc1EnvirArray[i].loc)

HandleInTrap(i);

}

}

else if (currentMapFlag == 1)

{

for (i = 0; i < gc2EnvirArray.Length; i++)

{

if (pos == gc2EnvirArray[i].loc)

HandleInTrap(i);

}

}

}

//itemwork

private void HandleInTrap(int index)

{

int truePCHP = (player.hitPoints1 \* 10) + player.hitPoints2;

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("The bear trap snaps around your leg, the sharpened edges inflicting grievous damage.");

//Apply damage

truePCHP -= bearTrapM11.damage;

//Check for death

if (truePCHP <= 0)

{

//Status message followed by closing the program

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("You have perished...");

System.Threading.Thread.Sleep(10000);

this.Close();

}

else

{

player.hitPoints1 = truePCHP / 10;

player.hitPoints2 = truePCHP % 10;

}

DeleteTrap(index);

}

//itemwork

private void DeleteTrap(int index)

{

int i, j;

EnvironmentObject[] temp;

if (currentMapFlag == 0)

{

temp = new EnvironmentObject[gc1EnvirArray.Length - 1];

for (i = 0, j = 0; j < gc1EnvirArray.Length; i++, j++)

{

if (j != index)

{

temp[i] = gc1EnvirArray[j];

}

else

i--;

}

gc1EnvirArray = new EnvironmentObject[temp.Length];

for (i = 0; i < temp.Length; i++)

{

gc1EnvirArray[i] = temp[i];

}

}

else if (currentMapFlag == 1)

{

temp = new EnvironmentObject[gc2EnvirArray.Length - 1];

for (i = 0, j = 0; j < gc2EnvirArray.Length; i++, j++)

{

if (j != index)

temp[i] = gc2EnvirArray[j];

}

gc2EnvirArray = new EnvironmentObject[temp.Length];

for (i = 0; i < temp.Length; i++)

{

gc2EnvirArray[i] = temp[i];

}

}

}

//Handles command attack directions during battle

//Includes an option for the player to hold their action

private void BattleKeyHandler(object sender, KeyPressEventArgs e)

{ // combatwork

if (e.KeyChar == 'e')

{

textBox2.AppendText(Environment.NewLine);

textBox2.AppendText("Attack north, east, south, or west?");

//System.Threading.Thread.Sleep(1000); combatwork

//pcActed = 1;

pcToAct = 1;

attackCommand = 1;

}

else if (e.KeyChar == 'q')

{

textBox2.AppendText(Environment.NewLine);

textBox2.AppendText("Cast north, east, south, or west?");

//System.Threading.Thread.Sleep(1000); combatwork

//pcActed = 1;

pcToAct = 1;

spellCommand = 1;

}

else if (e.KeyChar == 'x')

{

textBox2.AppendText(Environment.NewLine);

textBox2.AppendText("Action held...");

pcActed = 1;

}

else

e.Handled = true;

}

//Function for calculating melee hit chance and damage

private void ResolveMeleeAttack()

{

//Variables for calculating to hit and total damage

int pcTrueHit = (player.meleeHit1 \* 10) + player.meleeHit2;

int pcTrueDmg = (player.meleeDmg1 \* 10) + player.meleeDmg2;

int finalHit = pcTrueHit - (goblin.armor \* 2);

int isHit = toHitRand.Next(1, 100);

//If the mob is hit, apply damage and then check for death

if (isHit <= finalHit)

{

goblin.hitPoints -= pcTrueDmg;

if (goblin.hitPoints <= 0)

{

goblin.dead = 1;

CE.MobList[0] = 0;

player.exp2 += goblin.exp;

}

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("You score a solid hit with your weapon.");

TrashTalk();

}

else

{

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("Your enemy skillfully dodges your attack.");

}

}

//Function for calculating magic hit chance and damage

private void ResolveMagicAttack()

{

//Variables for the players chance to hit and total damage

int pcTrueHit = (player.magicHit1 \* 10) + player.magicHit2;

int pcTrueDmg = CurrentOffensiveSpell().damage;

pcTrueDmg += ((player.magicDmg1 \* 10) + player.magicDmg2) / 2;

int finalHit = pcTrueHit - (goblin.mResist \* 2);

int isHit = toHitRand.Next(1, 100);

//If the mob is hit, apply damage and then check for death

if (isHit <= finalHit)

{

goblin.hitPoints -= pcTrueDmg;

if (goblin.hitPoints <= 0)

{

goblin.dead = 1;

CE.MobList[0] = 0;

player.exp2 += goblin.exp;

}

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("Magical energies burn away at your enemy's life force.");

}

else

{

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("Unphased by the attack your enemy continues on.");

}

}

//Function for random trash talk from

//the goblin mob. Includes a chance he will say nothing (40%)

private void TrashTalk()

{

int tt = trashTalkRand.Next(1, 10);

if (tt == 1 || tt == 2)

{

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("Goblin: \"Owie, owie, owie!\"");

}

else if (tt == 4 || tt == 5)

{

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("Goblin: \"I'm gunna eat your eyes in a soup...\"");

}

else if (tt == 7 || tt == 8)

{

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("The goblin snarls, baring his yellow teeth at you.");

}

}

private char SpellType()

{

string selectedSpell = comboBox3.Items[comboBox3.SelectedIndex].ToString();

if (selectedSpell.CompareTo("Wave of Pain") == 0)

return 'O';

else if (selectedSpell.CompareTo("Minor Mend Wounds") == 0)

return 'H';

else if (selectedSpell.CompareTo("Rust") == 0)

return 'U';

else

return 'N';

}

//Function for handling melee and magic attacks

//private void PerformAction(object sender, KeyPressEventArgs e) combatwork

private void PerformAction(object sender, KeyEventArgs e)

{

//Variable for the direction of the attack

char spellCheck;

int dirOffset = 0;

/\*if (e.KeyChar == 'w')

{

dirOffset = -66;

}

else if (e.KeyChar == 'd')

{

dirOffset = 1;

}

else if (e.KeyChar == 's')

{

dirOffset = 66;

}

else if (e.KeyChar == 'a')

{

dirOffset = -1;

}\*/

if (e.KeyCode == Keys.Up)

{

dirOffset = -66;

}

else if (e.KeyCode == Keys.Right)

{

dirOffset = 1;

}

else if (e.KeyCode == Keys.Down)

{

dirOffset = 66;

}

else if (e.KeyCode == Keys.Left)

{

dirOffset = -1;

}

else

{

e.Handled = true;

return;

}

//Give the player a moment to catch up

//System.Threading.Thread.Sleep(1000); combatwork

//If melee attack, else magic attack

if (attackCommand == 1)

{

//Is there a mob directly adjacent to the character in the attack direction?

//Functionality for reach weapons will eventually fall here

if ((pcPos + dirOffset) != mobPos)

{

textBox2.AppendText(Environment.NewLine);

textBox2.AppendText("You swing wildly, hitting nothing but air.");

}

else

{

ResolveMeleeAttack();

}

}

else if (spellCommand == 1)

{

//Handles only cone based offensive spells atm

//Will eventually have more support for other spells (defensive, utility, etc.)

spellCheck = SpellType();

if (spellCheck == 'O')

{

int inSpell = 0, tempPos, i, j;

//Checks if the mob falls within the range of the spell

for (i = 1; i <= CurrentOffensiveSpell().range; i++)

{

tempPos = pcPos + (dirOffset \* i);

for (j = i - 1; j > 0; j--)

{

if (Math.Abs(dirOffset) == 1)

{

if ((tempPos + (j \* 66)) == mobPos || (tempPos - (j \* 66)) == mobPos)

{

inSpell = 1;

break;

}

}

else

{

if ((tempPos + 1) == mobPos || (tempPos - 1) == mobPos)

{

inSpell = 1;

break;

}

}

}

if (tempPos == mobPos || inSpell == 1)

{

inSpell = 1;

break;

}

}

//If a mob is in the spell, apply damage and costs

if (inSpell == 1)

{

int m1 = player.mana1, m2 = player.mana2;

int trueMana = (m1 \* 10) + m2;

if (trueMana >= CurrentOffensiveSpell().cost)

{

player.mana2 -= CurrentOffensiveSpell().cost;

if (player.mana2 < 0)

{

player.mana1--;

player.mana2 += 10;

}

ResolveMagicAttack();

}

else

{

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("You are unable to gather the energy for your spell.");

}

}

else

{

textBox2.AppendText(Environment.NewLine);

textBox2.AppendText("Your spell has no effect.");

}

}

else if (spellCheck == 'H')

{

int m1 = player.mana1, m2 = player.mana2;

int trueMana = (m1 \* 10) + m2;

if (trueMana >= CurrentHealingSpell().cost)

{

player.mana2 -= CurrentHealingSpell().cost;

if (player.mana2 < 0)

{

player.mana1--;

player.mana2 += 10;

}

player.hitPoints2 += CurrentHealingSpell().amountHealed;

if (player.hitPoints2 >= 10)

{

player.hitPoints1++;

player.hitPoints2 -= 10;

}

}

}

else if (spellCheck == 'U')

{//UNFINISHED

}

else

{

}

}

//Reset command flags

attackCommand = 0;

spellCommand = 0;

pcToAct = 0;

pcActed = 1;

e.Handled = true;

}

//A key handler for WASD movement only

private void KeyPressHandler(object sender, KeyPressEventArgs e)

{

int tempPcPos = pcPos;

if (e.KeyChar == 'w')

{

tempPcPos -= 66;

if (CanMove(tempPcPos))

{

pcPos = tempPcPos;

IsCurrentPosTrapped(pcPos);

this.Invalidate();

}

}

else if (e.KeyChar == 's')

{

tempPcPos += 66;

if (CanMove(tempPcPos))

{

pcPos = tempPcPos;

IsCurrentPosTrapped(pcPos);

this.Invalidate();

}

}

else if (e.KeyChar == 'a')

{

tempPcPos -= 1;

if (CanMove(tempPcPos))

{

pcPos = tempPcPos;

IsCurrentPosTrapped(pcPos);

this.Invalidate();

}

}

else if (e.KeyChar == 'd')

{

tempPcPos += 1;

if (CanMove(tempPcPos))

{

pcPos = tempPcPos;

IsCurrentPosTrapped(pcPos);

this.Invalidate();

}

}

else if (e.KeyChar == 'f')

{

HandleSkillUse();

}

else if (e.KeyChar == 'r')

{

HandleNonCombatSpellUse();

}

else if (e.KeyChar == ' ')

{

HandleInteraction();

}

else

e.Handled = true;

CheckNewPosition();

}

//keywork

private void HandleInteraction()

{

int locked = 0;

int north = pcPos - 66, south = pcPos + 66, east = pcPos + 1, west = pcPos - 1;

if (currentMapChars[north] == '[')

{

locked = 1;

}

else if (currentMapChars[north] == ']')

{

locked = 1;

}

else if (currentMapChars[south] == '[')

{

locked = 1;

}

else if (currentMapChars[south] == ']')

{

locked = 1;

}

else if (currentMapChars[east] == '[')

{

locked = 1;

}

else if (currentMapChars[east] == ']')

{

locked = 1;

}

else if (currentMapChars[west] == '[')

{

locked = 1;

}

else if (currentMapChars[west] == ']')

{

locked = 1;

}

else if (currentMapChars[north] == '{')

{

LootChest();

}

else if (currentMapChars[north] == '}')

{

LootChest();

}

else if (currentMapChars[south] == '{')

{

LootChest();

}

else if (currentMapChars[south] == '}')

{

LootChest();

}

else if (currentMapChars[east] == '{')

{

LootChest();

}

else if (currentMapChars[east] == '}')

{

LootChest();

}

else if (currentMapChars[west] == '{')

{

LootChest();

}

else if (currentMapChars[west] == '}')

{

LootChest();

}

if (locked == 1)

{

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("The chest is still locked.");

}

}

//keywork

private void LootChest()

{

if (currentMapFlag == 0)

{

if (!chest1.isEmpty)

{

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("You find an old sword and some magical berries.");

pcInventory.AddToWeapon(sword);

pcInventory.AddToConsumable(berry);

comboBox1.Items.Add("Sword");

comboBox4.Items.Add("Berry");

chest1.RemoveAllContents();

}

else

{

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("The chest is empty.");

}

}

else if (currentMapFlag == 1)

{

if (!chest2.isEmpty)

{

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("You find some leather robes and two silver coins.");

pcInventory.AddToArmor(lthrRobes);

comboBox2.Items.Add("Lthr Robes");

pcInventory.AddToCoinPurse(0, 2, 0);

chest2.RemoveAllContents();

}

else

{

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("The chest is empty.");

}

}

}

//keywork

private void HandleSkillUse()

{

string selectedSkill = comboBox5.Items[comboBox5.SelectedIndex].ToString();

int north = pcPos - 66, south = pcPos + 66, east = pcPos + 1, west = pcPos - 1;

if (selectedSkill.CompareTo("Pick Lock") == 0)

{

if (currentMapChars[north] == '[')

{

HandleLockPick(north);

}

else if (currentMapChars[north] == ']')

{

HandleLockPick(north - 1);

}

else if (currentMapChars[south] == '[')

{

HandleLockPick(south);

}

else if (currentMapChars[south] == ']')

{

HandleLockPick(south - 1);

}

else if (currentMapChars[east] == '[')

{

HandleLockPick(east);

}

else if (currentMapChars[east] == ']')

{

HandleLockPick(east - 1);

}

else if (currentMapChars[west] == '[')

{

HandleLockPick(west);

}

else if (currentMapChars[west] == ']')

{

HandleLockPick(west - 1);

}

}

else if (selectedSkill.CompareTo("Disable Trap") == 0)

{

if (currentMapChars[north] == '\*')

{

HandleDisableTrap(north);

}

else if (currentMapChars[south] == '\*')

{

HandleDisableTrap(south);

}

else if (currentMapChars[east] == '\*')

{

HandleDisableTrap(east);

}

else if (currentMapChars[west] == '\*')

{

HandleDisableTrap(west);

}

}

}

//keywork

private void HandleLockPick(int loc)

{

int i, rNum;

EnvironmentObject temp;

if (currentMapFlag == 0)

{

for (i = 0; i < gc1EnvirArray.Length; i++)

{

if (gc1EnvirArray[i].loc == loc)

{

break;

}

}

temp = gc1EnvirArray[i];

if (temp.name.CompareTo("Chest 1") == 0 && !chest1.isStuck && chest1.isLocked)

{

rNum = lockPickRand.Next(1, 10);

if (rNum <= 4)

{

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("There is a resounding click as the lock pops open.");

chest1.isLocked = false;

}

else

{

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("You fumble around for a moment, unable to force the lock open.");

chest1.IncrementAttempt();

}

if (chest1.pickNum == chest1.attemptNum)

{

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("After an audible snap from the lock you find yourself unable to open it this way. The lock will have to be destroyed.");

chest1.isStuck = true;

}

}

else if (chest1.isStuck)

{

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("The lock is broken and the lid stuck.");

}

}

else if (currentMapFlag == 1)

{

for (i = 0; i < gc2EnvirArray.Length; i++)

{

if (gc2EnvirArray[i].loc == loc)

{

break;

}

}

temp = gc2EnvirArray[i];

if (temp.name.CompareTo("Chest 2") == 0 && !chest2.isStuck && chest2.isLocked)

{

rNum = lockPickRand.Next(1, 10);

if (rNum <= 4)

{

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("There is a resounding click as the lock pops open.");

chest2.isLocked = false;

}

else

{

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("You fumble around for a moment, unable to force the lock open.");

chest2.IncrementAttempt();

}

if (chest2.pickNum == chest2.attemptNum)

{

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("After an audible snap from the lock you find yourself unable to open it this way. The lock will have to be destroyed.");

chest2.isStuck = true;

}

}

else if (chest2.isStuck)

{

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("The lock is broken and the lid stuck.");

}

}

}

//keywork

private void HandleDisableTrap(int loc)

{

int i, rNum;

EnvironmentObject temp;

if (currentMapFlag == 0)

{

for (i = 0; i < gc1EnvirArray.Length; i++)

{

if (gc1EnvirArray[i].loc == loc)

{

break;

}

}

temp = gc1EnvirArray[i];

rNum = disableTrapRand.Next(1, 10);

if (rNum <= 3)

{

DeleteTrap(i);

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("The bear trap harmlessly snaps closed.");

}

else

{

int truePCHP = (player.hitPoints1 \* 10) + player.hitPoints2;

if (temp.name.CompareTo("Bear Trap 1") == 0)

{

bearTrapM11.IncrementAttemp();

if (bearTrapM11.disNum == bearTrapM11.attemptNum)

{

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("The bear trap slams closed grabbing a chunk of your arm as it does.");

DeleteTrap(i);

//Apply damage

truePCHP -= bearTrapM11.damage / 2;

//Check for death

if (truePCHP <= 0)

{

//Status message followed by closing the program

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("You have perished...");

System.Threading.Thread.Sleep(10000);

this.Close();

}

else

{

player.hitPoints1 = truePCHP / 10;

player.hitPoints2 = truePCHP % 10;

}

}

else

{

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("You cautiously attempt to disable the trap to no avail.");

}

}

else if (temp.name.CompareTo("Bear Trap 2") == 0)

{

bearTrapM12.IncrementAttemp();

if (bearTrapM12.disNum == bearTrapM12.attemptNum)

{

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("The bear trap slams closed grabbing a chunk of your arm as it does.");

DeleteTrap(i);

//Apply damage

truePCHP -= bearTrapM11.damage / 2;

//Check for death

if (truePCHP <= 0)

{

//Status message followed by closing the program

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("You have perished...");

System.Threading.Thread.Sleep(10000);

this.Close();

}

else

{

player.hitPoints1 = truePCHP / 10;

player.hitPoints2 = truePCHP % 10;

}

}

else

{

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("You cautiously attempt to disable the trap to no avail.");

}

}

else if (temp.name.CompareTo("Bear Trap 3") == 0)

{

bearTrapM13.IncrementAttemp();

if (bearTrapM13.disNum == bearTrapM13.attemptNum)

{

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("The bear trap slams closed grabbing a chunk of your arm as it does.");

DeleteTrap(i);

//Apply damage

truePCHP -= bearTrapM11.damage / 2;

//Check for death

if (truePCHP <= 0)

{

//Status message followed by closing the program

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("You have perished...");

System.Threading.Thread.Sleep(10000);

this.Close();

}

else

{

player.hitPoints1 = truePCHP / 10;

player.hitPoints2 = truePCHP % 10;

}

}

else

{

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("You cautiously attempt to disable the trap to no avail.");

}

}

}

}

else if (currentMapFlag == 1)

{

for (i = 0; i < gc2EnvirArray.Length; i++)

{

if (gc2EnvirArray[i].loc == loc)

{

break;

}

}

temp = gc2EnvirArray[i];

rNum = disableTrapRand.Next(1, 10);

if (rNum <= 3)

{

DeleteTrap(i);

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("The bear trap harmlessly snaps closed.");

}

else

{

int truePCHP = (player.hitPoints1 \* 10) + player.hitPoints2;

if (temp.name.CompareTo("Bear Trap 1") == 0)

{

bearTrapM21.IncrementAttemp();

if (bearTrapM21.disNum == bearTrapM21.attemptNum)

{

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("The bear trap slams closed grabbing a chunk of your arm as it does.");

DeleteTrap(i);

//Apply damage

truePCHP -= bearTrapM11.damage / 2;

//Check for death

if (truePCHP <= 0)

{

//Status message followed by closing the program

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("You have perished...");

System.Threading.Thread.Sleep(10000);

this.Close();

}

else

{

player.hitPoints1 = truePCHP / 10;

player.hitPoints2 = truePCHP % 10;

}

}

else

{

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("You cautiously attempt to disable the trap to no avail.");

}

}

}

}

}

//keywork

private void HandleNonCombatSpellUse()

{

char currentSpellType = SpellType();

if (currentSpellType == 'H')

{

int truePCHP = (player.hitPoints1 \* 10) + player.hitPoints2;

int truePCMP = (player.mana1 \* 10) + player.mana2;

if (truePCHP < player.maxHP && truePCMP >= CurrentHealingSpell().cost)

{

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("Saturating your wounds with magical energies they begin to heal at a quicker rate.");

truePCHP += CurrentHealingSpell().amountHealed;

truePCMP -= CurrentHealingSpell().cost;

if (truePCHP > player.maxHP)

truePCHP = player.maxHP;

player.hitPoints1 = truePCHP / 10;

player.hitPoints2 = truePCHP % 10;

player.mana1 = truePCMP / 10;

player.mana2 = truePCMP % 10;

}

else if (truePCMP < CurrentHealingSpell().cost)

{

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("You don't have enough mana to cast the spell.");

}

else

{

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("Spell is useless, already at maximum health.");

}

}

else if (currentSpellType == 'U')

{

HandleRustSpell();

}

else

{

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("Not an appropriate spell.");

}

}

//keywork

private void HandleRustSpell()

{

int north = pcPos - 66, south = pcPos + 66, east = pcPos + 1, west = pcPos - 1;

int truePCMP = (player.mana1 \* 10) + player.mana2, i;

if (truePCMP >= CurrentUtilitySpell().cost)

{

if (currentMapFlag == 0)

{

for (i = 0; i < gc1EnvirArray.Length; i++)

{

if (gc1EnvirArray[i].loc == north || gc1EnvirArray[i].loc == south || gc1EnvirArray[i].loc == east || gc1EnvirArray[i].loc == west)

break;

else if (chest1.loc + 1 == north || chest1.loc + 1 == south || chest1.loc + 1 == east || chest1.loc + 1 == west)

break;

}

if (gc1EnvirArray.Length != i)

{

if (gc1EnvirArray[i].name.CompareTo("Chest 1") == 0)

RustTreasureChest(i);

else if (gc1EnvirArray[i].name.CompareTo("Bear Trap 1") == 0)

RustTrap(i);

else if (gc1EnvirArray[i].name.CompareTo("Bear Trap 2") == 0)

RustTrap(i);

else if (gc1EnvirArray[i].name.CompareTo("Bear Trap 3") == 0)

RustTrap(i);

}

}

else if (currentMapFlag == 1)

{

for (i = 0; i < gc2EnvirArray.Length; i++)

{

if (gc2EnvirArray[i].loc == north || gc2EnvirArray[i].loc == south || gc2EnvirArray[i].loc == east || gc2EnvirArray[i].loc == west)

break;

else if (chest2.loc + 1 == north || chest2.loc + 1 == south || chest2.loc + 1 == east || chest2.loc + 1 == west)

break;

}

if (gc2EnvirArray.Length != i)

{

if (gc2EnvirArray[i].name.CompareTo("Chest 2") == 0)

RustTreasureChest(i);

else if (gc2EnvirArray[i].name.CompareTo("Bear Trap 1") == 0)

RustTrap(i);

}

}

}

else

{

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("You don't have enough mana to cast the spell.");

}

}

//keywork

private void RustTreasureChest(int index)

{

if (currentMapFlag == 0)

{

if (chest1.isLocked)

{

int truePCMP = (player.mana1 \* 10) + player.mana2;

truePCMP -= CurrentUtilitySpell().cost;

player.mana1 = truePCMP / 10;

player.mana2 = truePCMP % 10;

chest1.isLocked = false;

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("There is a loud sizzling sound as the chest's lock is eaten away.");

}

else

{

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("The chest is already unlocked.");

}

}

else if (currentMapFlag == 1)

{

if (chest2.isLocked)

{

int truePCMP = (player.mana1 \* 10) + player.mana2;

truePCMP -= CurrentUtilitySpell().cost;

player.mana1 = truePCMP / 10;

player.mana2 = truePCMP % 10;

chest2.isLocked = false;

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("There is a loud sizzling sound as the chest's lock is eaten away.");

}

else

{

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("The chest is already unlocked.");

}

}

}

//keywork

private void RustTrap(int index)

{

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("Arcane energies burn the metallic trap in to a pile of dust.");

int truePCMP = (player.mana1 \* 10) + player.mana2;

truePCMP -= CurrentUtilitySpell().cost;

player.mana1 = truePCMP / 10;

player.mana2 = truePCMP % 10;

DeleteTrap(index);

}

//mapwork

private void CheckNewPosition()

{

if (currentMapFlag == 0)

{

if (pcPos == 3626)

{

ChangeCurrentMap();

}

}

else if (currentMapFlag == 1)

{

if (pcPos == 132)

{

ChangeCurrentMap();

}

}

}

//Moves a mob in some cardinal direction on the map

private void MoveMob(MonsterClass mob, char dir)

{

int tempMobPos = mob.pos;

if (dir == 'n')

{

tempMobPos -= 66;

if (CanMove(tempMobPos))

{

mob.pos = tempMobPos;

}

}

else if (dir == 's')

{

tempMobPos += 66;

if (CanMove(tempMobPos))

{

mob.pos = tempMobPos;

}

}

else if (dir == 'e')

{

tempMobPos += 1;

if (CanMove(tempMobPos))

{

mob.pos = tempMobPos;

}

}

else if (dir == 'w')

{

tempMobPos -= 1;

if (CanMove(tempMobPos))

{

mob.pos = tempMobPos;

}

}

DrawCurrentMap();

}

//Checks if a new position is a valid space for

//a player or mob

private bool CanMove(int newPos)

{

char key = currentMapChars[newPos];

if (key == '-' || key == '|' || key == '+' || key == '@' || key == '#' || key == '\\' || key == '/' || key == '%' || key == '[' || key == ']' || key == '{' || key == '}')

return false;

else

return true;

}

//Invalidates (re-paints) the form every time the timer "ticks"

private void WanderTimer\_Tick(object sender, EventArgs e)

{

this.Invalidate();

}

//Function for performing an action when the form closes

//Want to implement a goodbye message

private void Form1\_FormClosing(object sender, FormClosingEventArgs e)

{

}

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

{

UseConsumable();

}

private void UseConsumable()

{

Consumable selectedConsumable = CurrentConsumable();

int truePCHP = (player.hitPoints1 \* 10) + player.hitPoints2;

int truePCMP = (player.mana1 \* 10) + player.mana2;

if (selectedConsumable.Equals(berry))

{

if (truePCHP < player.maxHP || truePCMP < player.maxMana)

{

berry.amount--;

truePCHP += berry.upHP;

truePCMP += berry.upMP;

if (truePCHP > player.maxHP)

truePCHP = player.maxHP;

if (truePCMP > player.maxMana)

truePCMP = player.maxMana;

player.hitPoints1 = truePCHP / 10;

player.hitPoints2 = truePCHP % 10;

player.mana1 = truePCMP / 10;

player.mana2 = truePCMP % 10;

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("You eat a delicious, plump berry.");

if (berry.amount == 0)

{

comboBox4.SelectedIndex = 0;

comboBox4.Items.Remove("Berry");

pcInventory.RemoveConsumable("Berry");

}

}

}

else if (selectedConsumable.Equals(noConsumable))

{

textBox1.AppendText(Environment.NewLine);

textBox1.AppendText("No consumable selected.");

}

}

}

//Class for controlling combat

//Currently little more than a glorified turn tracker

public class CombatEngine

{

public int[] MobList { get; set; }

public int[] TurnOrder { get; set; }

public int[] TurnHistory { get; set; }

public int CurrentTurn { get; set; }

public CombatEngine()

{

TurnHistory = new int[20];

MobList = new int[20];

TurnOrder = new int[2];

CombatOrder();

}

public void CombatOrder()

{

TurnOrder[0] = 0;

TurnOrder[1] = 1;

CurrentTurn = 0;

}

public int NextTurn()

{

if (CurrentTurn == 1)

return CurrentTurn = 0;

else

return CurrentTurn++;

}

public int FocusedActor()

{

return TurnOrder[CurrentTurn];

}

public bool MetVictory()

{

int allDead = 1;

int i;

for (i = 0; i < MobList.Length; i++)

{

if (MobList[i] != 0)

allDead = 0;

}

if (allDead == 1)

{

return true;

}

else

return false;

}

}

//inventorywork

public class InventorySystem

{

public Weapon[] weaponInventory { get; set; }

public Armor[] armorInventory { get; set; }

public Consumable[] consumableInventory { get; set; }

public int wCounter { get; set; }

public int aCounter { get; set; }

public int cCounter { get; set; }

public int[] coinPurse { get; set; }

public InventorySystem()

{

weaponInventory = new Weapon[100];

wCounter = 0;

armorInventory = new Armor[100];

aCounter = 0;

consumableInventory = new Consumable[100];

cCounter = 0;

coinPurse = new int[3];

}

public void AddToWeapon(Weapon w)

{

weaponInventory[wCounter] = w;

wCounter++;

}

public void AddToArmor(Armor a)

{

armorInventory[aCounter] = a;

aCounter++;

}

public void AddToConsumable(Consumable c)

{

consumableInventory[cCounter] = c;

cCounter++;

}

public void AddToCoinPurse(int g, int s, int c)

{

coinPurse[0] += g;

coinPurse[1] += s;

coinPurse[2] += c;

}

public void RemoveWeapon(string name)

{

int i, j;

Weapon[] temp = new Weapon[weaponInventory.Length];

for (i = 0, j = 0; j < wCounter; i++, j++)

{

if (weaponInventory[j].name.CompareTo(name) != 0)

{

temp[i] = weaponInventory[j];

}

else

i--;

}

wCounter--;

for(i = 0; i < wCounter; i++)

weaponInventory[i] = temp[i];

}

public void RemoveArmor(string name)

{

int i, j;

Armor[] temp = new Armor[armorInventory.Length];

for (i = 0, j = 0; j < aCounter; i++, j++)

{

if (armorInventory[j].name.CompareTo(name) != 0)

{

temp[i] = armorInventory[j];

}

else

i--;

}

aCounter--;

for (i = 0; i < aCounter; i++)

armorInventory[i] = temp[i];

}

public void RemoveConsumable(string name)

{

int i, j;

Consumable[] temp = new Consumable[consumableInventory.Length];

for (i = 0, j = 0; j < cCounter; i++, j++)

{

if (consumableInventory[j].name.CompareTo(name) != 0)

{

temp[i] = consumableInventory[j];

}

else

i--;

}

cCounter--;

for (i = 0; i < cCounter; i++)

consumableInventory[i] = temp[i];

}

}

//Class for the player's character

public class PlayerCharacter

{

public int currentLevel1 { get; set; }

public int currentLevel2 { get; set; }

public int hitPoints1 { get; set; }

public int hitPoints2 { get; set; }

public int maxHP { get; set; }

public int mana1 { get; set; }

public int mana2 { get; set; }

public int maxMana { get; set; }

public int pcStrength1 { get; set; }

public int pcStrength2 { get; set; }

public int pcDexterity1 { get; set; }

public int pcDexterity2 { get; set; }

public int pcEndurance1 { get; set; }

public int pcEndurance2 { get; set; }

public int pcWillpower1 { get; set; }

public int pcWillpower2 { get; set; }

public int meleeDmg1 { get; set; }

public int meleeDmg2 { get; set; }

public int magicDmg1 { get; set; }

public int magicDmg2 { get; set; }

public int armor1 { get; set; }

public int armor2 { get; set; }

public int magicResist1 { get; set; }

public int magicResist2 { get; set; }

public int meleeHit1 { get; set; }

public int meleeHit2 { get; set; }

public int magicHit1 { get; set; }

public int magicHit2 { get; set; }

public int exp1 { get; set; }

public int exp2 { get; set; }

public int toLevel1 { get; set; }

public int toLevel2 { get; set; }

public int pos { get; set; }

public PlayerCharacter()

{

currentLevel1 = 0;

currentLevel2 = 1;

hitPoints1 = 1;

hitPoints2 = 5;

maxHP = 15;

mana1 = 2;

mana2 = 0;

maxMana = 20;

pcStrength1 = 1;

pcStrength2 = 0;

pcDexterity1 = 1;

pcDexterity2 = 2;

pcEndurance1 = 0;

pcEndurance2 = 7;

pcWillpower1 = 1;

pcWillpower2 = 5;

meleeDmg1 = 0;

meleeDmg2 = 2;

magicDmg1 = 0;

magicDmg2 = 3;

armor1 = 0;

armor2 = 1;

magicResist1 = 0;

magicResist2 = 1;

meleeHit1 = 5;

meleeHit2 = 1;

magicHit1 = 5;

magicHit2 = 1;

exp1 = 0;

exp2 = 0;

toLevel1 = 1;

toLevel2 = 0;

}

//This function resets the character's stats to the current base

public void ResetStats()

{

pcStrength1 = 1;

pcStrength2 = 0;

pcDexterity1 = 1;

pcDexterity2 = 2;

pcEndurance1 = 0;

pcEndurance2 = 7;

pcWillpower1 = 1;

pcWillpower2 = 5;

meleeDmg1 = 0;

meleeDmg2 = 2;

magicDmg1 = 0;

magicDmg2 = 3;

armor1 = 0;

armor2 = 1;

magicResist1 = 0;

magicResist2 = 1;

meleeHit1 = 5;

meleeHit2 = 1;

magicHit1 = 5;

magicHit2 = 1;

PlayerLevelOffset();

}

private void PlayerLevelOffset()

{

int lvlOffset = currentLevel2 - 1;

int strP = lvlOffset / 2, dexP = lvlOffset, endP = lvlOffset / 2;

int willP = lvlOffset, medP = lvlOffset / 2, madP = lvlOffset;

int armP = lvlOffset / 2, marP = lvlOffset, mehP = lvlOffset, mahP = lvlOffset \* 2;

pcStrength2 += strP;

pcDexterity2 += dexP;

pcEndurance2 += endP;

pcWillpower2 += willP;

meleeDmg2 += medP;

magicDmg2 += madP;

armor2 += armP;

magicResist2 += marP;

meleeHit2 += mehP;

magicHit2 += mahP;

}

}

//Base class for the various monsters (mobs)

public class MonsterClass

{

Random wanderRand = new Random();

public string Name { get; set; }

public int mobID { get; set; }

public int pos { get; set; }

public int aggroRad { get; set; }

public int hitPoints { get; set; }

public int armor { get; set; }

public int mResist { get; set; }

public int meleeDmg { get; set; }

public int meleeHit { get; set; }

public int exp { get; set; }

public int dead { get; set; }

public MonsterClass(string name, int ID, int initPos)

{

Name = name;

mobID = ID;

pos = initPos;

aggroRad = 6;

hitPoints = 10;

armor = 3;

mResist = 1;

meleeDmg = 4;

meleeHit = 70;

exp = 2;

dead = 0;

}

//Function for determing a random direction to wander in

public char Wander()

{

int numRand = wanderRand.Next(1, 5);

if (numRand == 1)

{

return 'n';

}

else if (numRand == 2)

{

return 's';

}

else if (numRand == 3)

{

return 'w';

}

else

{

return 'e';

}

}

}

//Base class for the items in the game

public class BaseItem

{

public string name { get; set; }

public string type { get; set; }

public BaseItem(string n, string t)

{

name = n;

type = t;

}

}

//Class for a player's weapon

public class Weapon : BaseItem

{

public int meleeDmg { get; set; }

public int magicDmg { get; set; }

public int meleeHit { get; set; }

public int magicHit { get; set; }

public int range { get; set; }

public Weapon(string n, int meD, int maD, int melH, int magH, int r)

: base(n, "Weapon")

{

meleeDmg = meD;

magicDmg = maD;

meleeHit = melH;

magicHit = magH;

range = r;

}

}

//Class for a player's armor

public class Armor : BaseItem

{

public int prot { get; set; }

public int mResist { get; set; }

public Armor(string n, int p, int mR)

: base(n, "Armor")

{

prot = p;

mResist = mR;

}

}

//Class for consumable items

public class Consumable : BaseItem

{

public int amount { get; set; }

public int upHP { get; set; }

public int upMP { get; set; }

public Consumable(string n, int num, int hp, int mp)

: base(n, "Consumable")

{

amount = num;

upHP = hp;

upMP = mp;

}

}

//Base class for various spells

public class Spell

{

public string name { get; set; }

public int cost { get; set; }

public string type { get; set; }

public Spell(string n, int c, string t)

{

name = n;

cost = c;

type = t;

}

}

//Class for combat based spells

public class OffensiveSpell : Spell

{

//Variables for damage, range, AOE, touch, and cone flags,

//as well as where the spells effect takes place.

public int damage { get; set; }

public int range { get; set; }

public int AOE { get; set; }

public int cone { get; set; }

public int touch { get; set; } //spellwork

public int centerEffect { get; set; }

public OffensiveSpell(string n, int c, int d, int r, int aoe, int co, int t, int cE)

: base(n, c, "Offensive")

{

damage = d;

range = r;

AOE = aoe;

cone = co;

touch = t;

centerEffect = cE;

}

}

//spellwork

public class DefensiveSpell : Spell

{

public char centerEffect { get; set; }

public DefensiveSpell(string n, int c, char cE)

: base(n, c, "Defensive")

{

centerEffect = cE;

}

}

//spellwork

public class HealingSpell : DefensiveSpell

{

public int amountHealed { get; set; }

public HealingSpell(string n, int c, char ce, int aH)

: base(n, c, ce)

{

amountHealed = aH;

}

}

//spellwork

public class UtilitySpell : Spell

{

public UtilitySpell(string n, int c)

: base(n, c, "Utility")

{

}

}

//skillwork

public class Skill

{

public string name { get; set; }

public Skill(string n)

{

name = n;

}

}

public class EnvironmentObject

{

public string name { get; set; }

public int loc { get; set; }

public EnvironmentObject(string n, int l)

{

name = n;

loc = l;

}

}

public class TreasureChest : EnvironmentObject

{

public bool isLocked { get; set; }

public int pickNum { get; set; }

public int attemptNum { get; set; }

public bool isStuck { get; set; }

public bool isEmpty { get; set; }

public Weapon[] wContents { get; set; }

public Armor[] aContents { get; set; }

public Consumable[] cContents { get; set; }

public TreasureChest(string n, int l, bool iL, bool iE, Weapon[] wC, Armor[] aC, Consumable[] cC)

: base(n, l)

{

isLocked = iL;

pickNum = 3;

attemptNum = 0;

isStuck = false;

isEmpty = iE;

wContents = new Weapon[100];

aContents = new Armor[100];

cContents = new Consumable[100];

if (!iE)

AddInDefaultContents(wC, aC, cC);

}

private void AddInDefaultContents(Weapon[] w, Armor[] a, Consumable[] c)

{

int i;

for (i = 0; i < w.Length; i++)

wContents[i] = w[i];

for (i = 0; i < a.Length; i++)

aContents[i] = a[i];

for (i = 0; i < c.Length; i++)

cContents[i] = c[i];

}

public void RemoveAllContents()

{

wContents = new Weapon[100];

aContents = new Armor[100];

cContents = new Consumable[100];

isEmpty = true;

}

public void MakeStuck()

{

isStuck = true;

}

public void IncrementAttempt()

{

attemptNum++;

}

}

public class Trap : EnvironmentObject

{

public int damage { get; set; }

public int disNum { get; set; }

public int attemptNum { get; set; }

public Trap(string n, int l, int d)

: base(n, l)

{

damage = d;

disNum = 3;

attemptNum = 0;

}

public void IncrementAttemp()

{

attemptNum++;

}

}

}