PRG655 Manuel

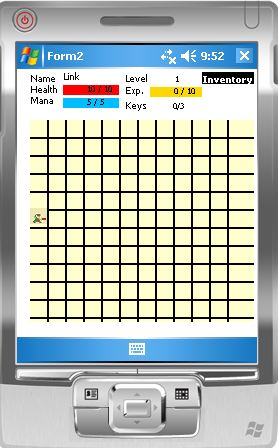
Link’s Nightmare

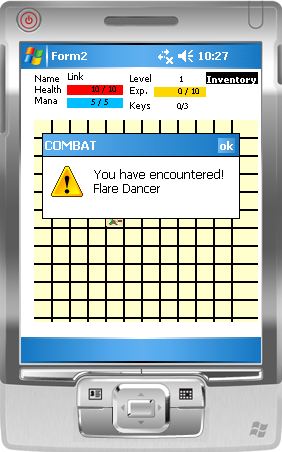
By: Jordan James Turner

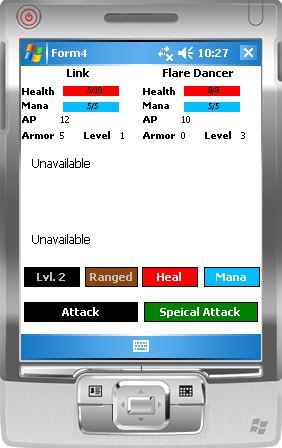
Login Menu

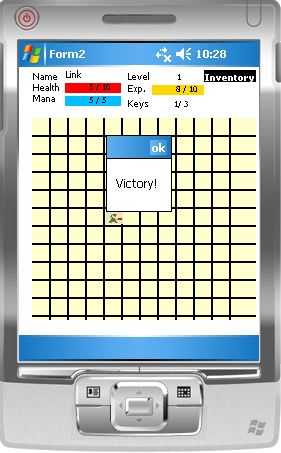
 This is the first thing the user will see, what is happening behind the scenes here is the game resources are being loaded into the game map before the user even clicks play.

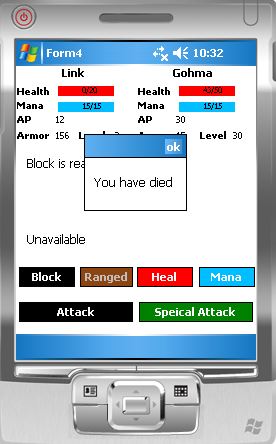
Map Layout

 The user starts on the middle left of the map. The top part contains all the basic user attributes along with a button for the user’s inventory. The map is randomly populated with monsters and items. Some items have level restrictions on them but, once the user levels up they will be able to access these items once they find them. Leveling up is done through monster fights.

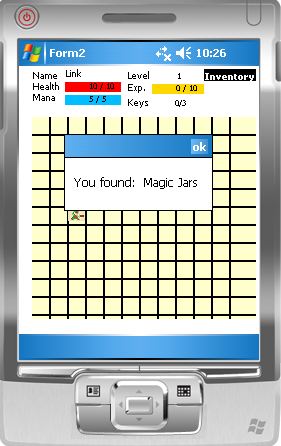
 The user can will walk around and eventually run into a monster. The screen will look like this. It will notify the user that it has a encountered a monster and will wait for the user to respond before entering battle.

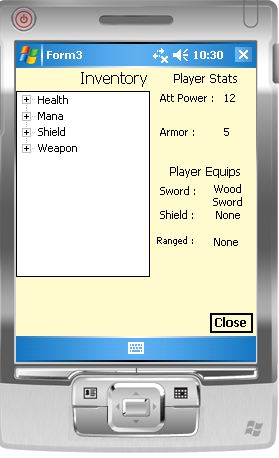
 This form will open and will allow the user to have combat with the monster. The first move of combat is randomly determined by the combat moderator. The user at a low level will not have access to all abilities but as the user levels up they will unlock new abilities that they can use.

 If the user manages to defeat the monster they will be brought back to the map and displayed a message rewarding them on their victory. On top of that, they will be rewarded experience points based on which monster they killed, every monster gives different experience points, the more experience points the user gains the higher their level will become. The final thing they’re rewarded is a key. It takes a total of three keys to unlock the boss.

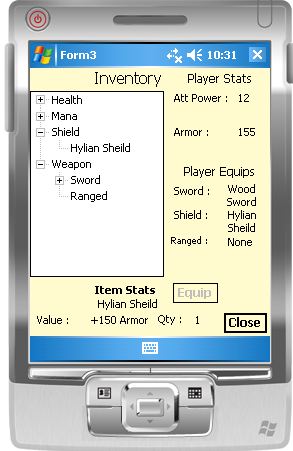
 If the user manages to be defeated by the monster in battle then this message will be displayed.

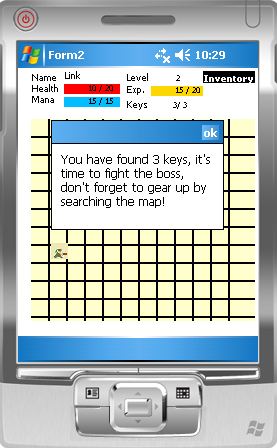
The user will then have to decide if they want to play again or not. They will be brought back to the starting point if they wish to play again.

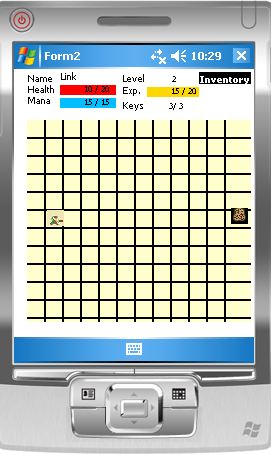
Users can find items throughout the map. This is an example of the message that will be displayed when they find an item.

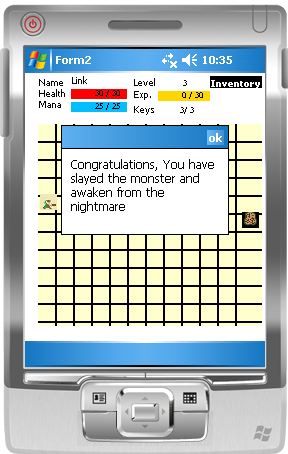
 If the user clicks the Inventory button in the top right corner then they will be shown their inventory with all the items they have found.

The user can pick a new item they have found to equip. In this example they are equipping a newly found shield they have just found.

Once the user equips the item there stats are updated along with their currently equipped weapons. The equipped button is disabled since the user has this item equipped already.

 Once the user has obtained 3 keys they will be displayed this message. All monsters that the user didn’t encounter will be removed so that the user can walk around freely to obtain new items.

The boss will displayed to show the user where they have to move to once they are ready to fight the boss.

Once the user has defeated the boss this message will be displayed before the game ends.

**File I/O**

/// <summary>

/// This function uses FILE I/O to load the game resources into class lists that are used by the rest of the project to populate the map.

/// </summary>

private void GetItems()

{

if (File.Exists(healthPath) == true)

{

using (StreamReader reader = new StreamReader(healthPath))

{

string line;

while ((line = reader.ReadLine()) != null)

{

if(line.IndexOf("Name") != -1)

{

int index = line.IndexOf(':');

string result = line.Remove(0,index +1);

game.HealthItemNames.Add(result);

}

if (line.IndexOf("Value") != -1)

{

int index = line.IndexOf(':');

string result = line.Remove(0, index + 1);

game.HealthItemValues.Add(Convert.ToInt32(result));

}

}

}

}

if (File.Exists(manaPath) == true)

{

using (StreamReader reader = new StreamReader(manaPath))

{

string line;

while ((line = reader.ReadLine()) != null)

{

if (line.IndexOf("Name") != -1)

{

int index = line.IndexOf(':');

string result = line.Remove(0, index + 1);

game.ManaItemNames.Add(result);

}

if (line.IndexOf("Value") != -1)

{

int index = line.IndexOf(':');

string result = line.Remove(0, index + 1);

game.ManaItemValues.Add(Convert.ToInt32(result));

}

}

}

}

if(File.Exists(swordsPath))

{

using (StreamReader reader = new StreamReader(swordsPath))

{

string line;

while ((line = reader.ReadLine()) != null)

{

if (line.IndexOf("Name") != -1)

{

int index = line.IndexOf(':');

string result = line.Remove(0, index + 1);

game.SwordNames.Add(result);

}

if (line.IndexOf("Value") != -1)

{

int index = line.IndexOf(':');

string result = line.Remove(0, index + 1);

game.SwordValues.Add(Convert.ToInt32(result));

}

}

}

}

if (File.Exists(sheildsPath))

{

using (StreamReader reader = new StreamReader(sheildsPath))

{

string line;

while ((line = reader.ReadLine()) != null)

{

if (line.IndexOf("Name") != -1)

{

int index = line.IndexOf(':');

string result = line.Remove(0, index + 1);

game.SheildNames.Add(result);

}

if (line.IndexOf("Value") != -1)

{

int index = line.IndexOf(':');

string result = line.Remove(0, index + 1);

game.SheildValues.Add(Convert.ToInt32(result));

}

}

}

}

if (File.Exists(rangedPath))

{

using (StreamReader reader = new StreamReader(rangedPath))

{

string line;

while ((line = reader.ReadLine()) != null)

{

if (line.IndexOf("Name") != -1)

{

int index = line.IndexOf(':');

string result = line.Remove(0, index + 1);

game.RangedNames.Add(result);

}

if (line.IndexOf("Value") != -1)

{

int index = line.IndexOf(':');

string result = line.Remove(0, index + 1);

game.RangedValues.Add(Convert.ToInt32(result));

}

}

}

}

if (File.Exists(keysPath))

{

using (StreamReader reader = new StreamReader(keysPath))

{

string line;

while ((line = reader.ReadLine()) != null)

{

if (line.IndexOf("Name") != -1)

{

int index = line.IndexOf(':');

string result = line.Remove(0, index + 1);

game.KeyNames.Add(result);

}

}

}

}

if (File.Exists(enemyPath))

{

using (StreamReader reader = new StreamReader(enemyPath))

{

string line;

while ((line = reader.ReadLine()) != null)

{

if (line.IndexOf("Name") != -1)

{

int index = line.IndexOf(':');

string result = line.Remove(0, index + 1);

game.EnemyNames.Add(result);

}

if (line.IndexOf("Health") != -1)

{

int index = line.IndexOf(':');

string result = line.Remove(0, index + 1);

game.EnemyHealth.Add(Convert.ToInt32(result));

}

if (line.IndexOf("AP") != -1)

{

int index = line.IndexOf(':');

string result = line.Remove(0, index + 1);

game.EnemyAP.Add(Convert.ToInt32(result));

}

if (line.IndexOf("Mana") != -1)

{

int index = line.IndexOf(':');

string result = line.Remove(0, index + 1);

game.EnemyMana.Add(Convert.ToInt32(result));

}

if (line.IndexOf("Armor") != -1)

{

int index = line.IndexOf(':');

string result = line.Remove(0, index + 1);

game.EnemyArmor.Add(Convert.ToInt32(result));

}

if (line.IndexOf("Level") != -1)

{

int index = line.IndexOf(':');

string result = line.Remove(0, index + 1);

game.EnemyLevel.Add(Convert.ToInt32(result));

}

if (line.IndexOf("Exp") != -1)

{

int index = line.IndexOf(':');

string result = line.Remove(0, index + 1);

game.EnemyExpAward.Add(Convert.ToInt32(result));

}

}

}

}

if (File.Exists(bossPath))

{

using (StreamReader reader = new StreamReader(bossPath))

{

string line;

while ((line = reader.ReadLine()) != null)

{

if (line.IndexOf("Name") != -1)

{

int index = line.IndexOf(':');

string result = line.Remove(0, index + 1);

game.BossNames.Add(result);

}

if (line.IndexOf("Health") != -1)

{

int index = line.IndexOf(':');

string result = line.Remove(0, index + 1);

game.BossHealth.Add(Convert.ToInt32(result));

}

if (line.IndexOf("AP") != -1)

{

int index = line.IndexOf(':');

string result = line.Remove(0, index + 1);

game.BossAP.Add(Convert.ToInt32(result));

}

if (line.IndexOf("Mana") != -1)

{

int index = line.IndexOf(':');

string result = line.Remove(0, index + 1);

game.BossMana.Add(Convert.ToInt32(result));

}

if (line.IndexOf("Armor") != -1)

{

int index = line.IndexOf(':');

string result = line.Remove(0, index + 1);

game.BossArmor.Add(Convert.ToInt32(result));

}

if (line.IndexOf("Level") != -1)

{

int index = line.IndexOf(':');

string result = line.Remove(0, index + 1);

game.BossLevel.Add(Convert.ToInt32(result));

}

if (line.IndexOf("Exp") != -1)

{

int index = line.IndexOf(':');

string result = line.Remove(0, index + 1);

game.BossExpAward.Add(Convert.ToInt32(result));

}

}

}

}

if (File.Exists(baHumanPath))

{

string name="";

int value=-1, level=-1, cd=-1;

using (StreamReader reader = new StreamReader(baHumanPath))

{

string line;

while ((line = reader.ReadLine()) != null)

{

if (line.IndexOf("Name") != -1)

{

int index = line.IndexOf(':');

string result = line.Remove(0, index + 1);

name = result;

}

if (line.IndexOf("Value") != -1)

{

int index = line.IndexOf(':');

string result = line.Remove(0, index + 1);

value = Convert.ToInt32(result);

}

if (line.IndexOf("Level") != -1)

{

int index = line.IndexOf(':');

string result = line.Remove(0, index + 1);

level = Convert.ToInt32(result);

}

if (line.IndexOf("Cooldown") != -1)

{

int index = line.IndexOf(':');

string result = line.Remove(0, index + 1);

cd = Convert.ToInt32(result);

}

if (name != "" && value != -1 && level != -1 && cd != -1)

{

userPlayer.UpdateBasicAttacks(name, value, level, cd);

name = "";

value = -1;

level = -1;

cd = -1;

}

}

}

}

if (File.Exists(saHumanPath))

{

string name = "";

int value = -1, level = -1, cost = -1;

using (StreamReader reader = new StreamReader(saHumanPath))

{

string line;

while ((line = reader.ReadLine()) != null)

{

if (line.IndexOf("Name") != -1)

{

int index = line.IndexOf(':');

string result = line.Remove(0, index + 1);

name = result;

}

if (line.IndexOf("Value") != -1)

{

int index = line.IndexOf(':');

string result = line.Remove(0, index + 1);

value = Convert.ToInt32(result);

}

if (line.IndexOf("Cost") != -1)

{

int index = line.IndexOf(':');

string result = line.Remove(0, index + 1);

cost = Convert.ToInt32(result);

}

if (line.IndexOf("Level") != -1)

{

int index = line.IndexOf(':');

string result = line.Remove(0, index + 1);

level = Convert.ToInt32(result);

}

if (name != "" && value != -1 && level != -1 && cost != -1)

{

userPlayer.UpdateSpecialAttacks(name, value, level,cost);

name = "";

value = -1;

level = -1;

cost = -1;

}

}

}

}

if (File.Exists(saComputerPath))

{

using (StreamReader reader = new StreamReader(saHumanPath))

{

string line;

while ((line = reader.ReadLine()) != null)

{

if (line.IndexOf("Name") != -1)

{

int index = line.IndexOf(':');

string result = line.Remove(0, index + 1);

saCName = result;

}

if (line.IndexOf("Value") != -1)

{

int index = line.IndexOf(':');

string result = line.Remove(0, index + 1);

saCValue = Convert.ToInt32(result);

}

if (line.IndexOf("Cost") != -1)

{

int index = line.IndexOf(':');

string result = line.Remove(0, index + 1);

saCCost = Convert.ToInt32(result);

}

}

}

}

}

**Abstract Class**

namespace Project\_Game

{

public abstract class Player

{

private Int32 health;

private Int32 attackPower;

private Int32 mana;

private Int32 armor;

private Int32 level;

private String name;

protected Int32 exp;

protected Int32 totalHealth;

protected Int32 totalMana;

protected List<String> specAttack;

protected List<String> baseAttack;

protected List<Int32> baseLevel;

protected List<Int32> specLevel;

protected List<Int32> baseCD;

protected List<Int32> saDamage;

protected List<Int32> baDamage;

protected List<Int32> specCost;

public Player(String \_name, Int32 \_health, Int32 \_attackPower, Int32 \_mana, Int32 \_armor, Int32 \_level, Int32 \_exp )

{

name = \_name;

health = \_health;

attackPower = \_attackPower;

mana = \_mana;

armor = \_armor;

level = \_level;

}

public Int32 Health

{

get { return health; }

set { health = value; }

}

public Int32 AttackPower

{

get { return attackPower; }

set { attackPower = value; }

}

public Int32 Mana

{

get { return mana; }

set { mana = value; }

}

public Int32 Armor

{

get { return armor; }

set { armor = value; }

}

public Int32 Level

{

get { return level; }

set { level = value; }

}

public String Name

{

get { return name; }

set { name = value; }

}

public abstract Int32 Exp

{

get;

set;

}

public abstract Int32 TotalHealth

{

get;

set;

}

public abstract Int32 TotalMana

{

get;

set;

}

public abstract List<String> SpecAttack

{

get;

}

public abstract List<String> BaseAttack

{

get;

}

public abstract List<Int32> SaDamage

{

get;

}

public abstract List<Int32> BaDamage

{

get;

}

public abstract List<Int32> BaseLevel

{

get;

}

public abstract List<Int32> BaseCD

{

get;

}

public abstract List<Int32> SpecLevel

{

get;

}

public abstract List<Int32> SpecCost

{

get;

}

public abstract void Move(Object combatMove);

public abstract void UpdateStat(Int32 hp, StatAction action, Stats stat, CombatType cType);

public abstract void UpdateSpecialAttacks(String attack, Int32 damage, Int32 level, Int32 cost);

public abstract void UpdateBasicAttacks(String attack, Int32 damage, Int32 level, Int32 cd);

}

/// <summary>

/// This class represents a Computer Player

/// </summary>

public class ComputerPlayer : Player

{

private String keyName;

private Int32 numKeys;

private MonsterType mType;

private Int32 lastAttackPower;

private Int32 lastArmor;

public ComputerPlayer(String name, Int32 health, Int32 attackPower, Int32 mana, Int32 armor, Int32 level, Int32 exp, String \_keyName, Int32 \_numKeys, MonsterType \_mType )

: base(name, health, attackPower, mana, armor, level, exp)

{

keyName = \_keyName;

numKeys = \_numKeys;

mType = \_mType;

totalHealth += health;

totalMana += mana;

Exp = exp;

}

public String KeyName

{

get { return keyName; }

}

public Int32 NumKeys

{

get { return numKeys; }

}

public MonsterType MType

{

get { return MType; }

}

public override Int32 Exp

{

get { return exp; }

set { exp = value; }

}

public override Int32 TotalHealth

{

get { return totalHealth; }

set { totalHealth = value; }

}

public override Int32 TotalMana

{

get { return totalMana; }

set { totalMana = value; }

}

public override List<String> SpecAttack

{

get { return specAttack; }

}

public override List<String> BaseAttack

{

get { return baseAttack; }

}

public override List<Int32> SaDamage

{

get { return saDamage; }

}

public override List<Int32> BaDamage

{

get { return baDamage; }

}

public override List<Int32> BaseLevel

{

get { return baseLevel; }

}

public override List<Int32> BaseCD

{

get { return baseCD; }

}

public override List<int> SpecCost

{

get { return specCost; }

}

public override List<Int32> SpecLevel

{

get { return specLevel; }

}

public Int32 LastArmor

{

get { return lastArmor; }

}

public Int32 LastAttackPower

{

get { return lastAttackPower; }

}

/// <summary>

/// Used to assign special attacks to player. parameters are added to there respected list.

/// </summary>

/// <param name="attack"></param>

/// <param name="damage"></param>

public override void UpdateSpecialAttacks(String attack, Int32 damage, Int32 level, Int32 cost)

{

if (specAttack == null)

{

specAttack = new List<String>();

specAttack.Capacity = 3;

}

if (saDamage == null)

{

saDamage = new List<Int32>();

saDamage.Capacity = 3;

}

if (specLevel == null)

{

specLevel = new List<Int32>();

specLevel.Capacity = 3;

}

if (specCost == null)

{

specCost = new List<Int32>();

specCost.Capacity = 3;

}

if (!specAttack.Contains(attack))

{

specAttack.Add(attack);

saDamage.Add(damage);

specLevel.Add(level);

specCost.Add(cost);

}

}

/// <summary>

/// Used to assign basic attacks to player. parameters are added to there respected list.

/// </summary>

/// <param name="attack"></param>

/// <param name="damage"></param>

public override void UpdateBasicAttacks(String attack, Int32 damage, Int32 level, Int32 cd)

{

if (baseAttack == null)

{

baseAttack = new List<String>();

baseAttack.Capacity = 3;

}

if (baDamage == null)

{

baDamage = new List<Int32>();

baDamage.Capacity = 3;

}

if (baseLevel == null)

{

baseLevel = new List<Int32>();

baseLevel.Capacity = 3;

}

if (baseCD == null)

{

baseCD = new List<Int32>();

baseCD.Capacity = 3;

}

if (!baseAttack.Contains(attack))

{

baseAttack.Add(attack);

baDamage.Add(damage);

baseLevel.Add(level);

baseCD.Add(cd);

}

}

public override void UpdateStat(Int32 amount, StatAction action, Stats stat, CombatType cType)

{

switch (stat)

{

case Stats.Health:

if (action == StatAction.Add)

{

if (Health != TotalHealth)

Health += amount;

}

else if (action == StatAction.Sub)

{

Health -= amount;

if (Health < 0)

Health = 0;

}

break;

case Stats.Mana:

if (action == StatAction.Add)

{

if (Mana != TotalMana)

Mana += amount;

}

else if (action == StatAction.Sub)

{

Mana -= amount;

if (Mana < 0)

Mana = 0;

}

break;

case Stats.Armor:

if (action == StatAction.Add)

{

if (cType != CombatType.Ranged && cType != CombatType.SpecialAttack)

lastArmor = Armor;

Armor += amount;

}

else if (action == StatAction.Sub)

{

Armor -= amount;

}

break;

case Stats.AttackPower:

if (action == StatAction.Add)

{

if (cType != CombatType.Ranged || cType != CombatType.SpecialAttack)

lastAttackPower = amount;

AttackPower += amount;

}

else if (action == StatAction.Sub)

{

AttackPower -= amount;

}

break;

/\*case Stats.Level:

if (action == StatAction.Add)

{

exp += amount;

if (exp == totalExp)

{

TotalExp += 10;

exp = 0;

Level += 1;

TotalHealth += 10;

Health = TotalHealth;

}

}

break;\*/

}

}

public override void Move(object combatMove)

{

throw new NotImplementedException();

}

}

/// <summary>

/// This class represents a Human Player

/// </summary>

public class HumanPlayer : Player

{

protected bool alreadyMoved = false;

private String weapon;

private Int32 weaponDamage;

private String shield;

private Int32 shieldValue;

private String ranged;

private Int32 rangedDamage;

private List<String> inventory;

private List<InventoryTyp> inventoryType;

private List<Int32> inventoryValue;

private Int32 totalExp;

private Int32 lastAttackAmount;

private Int32 lastArmor;

private Boolean winStatus;

private Int32 keyCount;

private Int32 curMonsterPos;

private Boolean bossWin;

/// <summary>

/// Create a human player that will be linked to the user of the game, you.

/// </summary>

/// <param name="name">The name of the player goes here, we've picked Link since this a tribute to the Zelda games.</param>

/// <param name="health">Your typical video game has health, just like Zelda, so we start with 10Hp.</param>

/// <param name="attackPower">How much damage you do to the enemy.</param>

/// <param name="mana">The amount of points you need to use Special Attacks.</param>

/// <param name="armor">The amount of damage you can take before you lose health.</param>

/// <param name="level">The higher the level the greater the weapons you can find.</param>

/// <param name="\_weapon">Better swords can be found on the map from searching around.</param>

/// <param name="\_weaponDamage">How much damage you do with your sword. the attack amount is added to the players weapon.</param>

/// <param name="\_shield">Adds to Armor</param>

/// <param name="\_shieldValue">Adds to Armor</param>

/// <param name="\_ranged">Adds to Attack Power, has cooldown.</param>

/// <param name="\_rangedDamage">Adds to Attack Power, has cooldown.</param>

/// <param name="exp">Once the total amount is reached you level up.</param>

public HumanPlayer(String name, Int32 health, Int32 attackPower, Int32 mana, Int32 armor, Int32 level,String \_weapon, Int32 \_weaponDamage, String \_shield, Int32 \_shieldValue, String \_ranged, Int32 \_rangedDamage, Int32 exp)

: base(name,health,attackPower,mana,armor,level, exp)

{

weapon = \_weapon;

weaponDamage = \_weaponDamage;

shield = \_shield;

shieldValue = \_shieldValue;

ranged = \_ranged;

rangedDamage = \_rangedDamage;

inventory = new List<String>();//List to hold the items that the user finds.

inventoryType = new List<InventoryTyp>();

inventoryValue = new List<Int32>();

}

public Boolean BoosWin

{

get { return bossWin; }

set { bossWin = value; }

}

public Int32 CurMonsterPos

{

get { return curMonsterPos; }

set { curMonsterPos = value; }

}

public Int32 KeyCount

{

get { return keyCount; }

set { keyCount = value; }

}

public Boolean WinStatus

{

get { return winStatus; }

set { winStatus = value; }

}

public Int32 LastAttackAmount

{

get { return lastAttackAmount; }

}

public Int32 LastArmor

{

get { return lastArmor; }

}

public Int32 TotalExp

{

get { return totalExp; }

set { totalExp = value; }

}

public String Weapon

{

get { return weapon; }

}

public Int32 WeaponDamage

{

get { return weaponDamage; }

}

public String Shield

{

get { return shield; }

}

public Int32 ShieldValue

{

get { return shieldValue; }

}

public String Ranged

{

get { return ranged; }

}

public Int32 RangedDamage

{

get { return rangedDamage; }

}

public List<String> Inventory

{

get { return inventory; }

}

public List<InventoryTyp> InventoryType

{

get { return inventoryType; }

}

public List<Int32> InventoryValue

{

get { return inventoryValue; }

}

public override Int32 Exp

{

get { return exp; }

set { exp = value; }

}

public override Int32 TotalHealth

{

get { return totalHealth; }

set { totalHealth = value; }

}

public override Int32 TotalMana

{

get { return totalMana; }

set { totalMana = value; }

}

public override List<String> SpecAttack

{

get { return specAttack; }

}

public override List<String> BaseAttack

{

get { return baseAttack; }

}

public override List<Int32> SaDamage

{

get { return saDamage; }

}

public override List<Int32> BaDamage

{

get { return baDamage; }

}

public override List<Int32> BaseLevel

{

get { return baseLevel; }

}

public override List<Int32> BaseCD

{

get { return baseCD; }

}

public override List<int> SpecCost

{

get { return specCost; }

}

public override List<Int32> SpecLevel

{

get { return specLevel; }

}

public void UpdateInventory(String item, InventoryTyp type, Int32 value)

{

if (inventory == null)

inventory = new List<String>();

if (inventoryType == null)

inventoryType = new List<InventoryTyp>();

if (inventoryValue == null)

inventoryValue = new List<Int32>();

inventory.Add(item);

inventoryType.Add(type);

inventoryValue.Add(value);

}

/// <summary>

/// Used to assign special attacks to player. parameters are added to there respected list.

/// </summary>

/// <param name="attack"></param>

/// <param name="damage"></param>

/// <summary>

/// Used to assign special attacks to player. parameters are added to there respected list.

/// </summary>

/// <param name="attack"></param>

/// <param name="damage"></param>

public override void UpdateSpecialAttacks(String attack, Int32 damage, Int32 level, Int32 cost)

{

if (specAttack == null)

{

specAttack = new List<String>();

specAttack.Capacity = 3;

}

if (saDamage == null)

{

saDamage = new List<Int32>();

saDamage.Capacity = 3;

}

if (specLevel == null)

{

specLevel = new List<Int32>();

specLevel.Capacity = 3;

}

if (specCost == null)

{

specCost = new List<Int32>();

specCost.Capacity = 3;

}

if (!specAttack.Contains(attack))

{

specAttack.Add(attack);

saDamage.Add(damage);

specLevel.Add(level);

specCost.Add(cost);

}

}

/// <summary>

/// Used to assign basic attacks to player. parameters are added to there respected list.

/// </summary>

/// <param name="attack"></param>

/// <param name="damage"></param>

public override void UpdateBasicAttacks(String attack, Int32 damage, Int32 level, Int32 cd)

{

if (baseAttack == null)

{

baseAttack = new List<String>();

baseAttack.Capacity = 3;

}

if (baDamage == null)

{

baDamage = new List<Int32>();

baDamage.Capacity = 3;

}

if (baseLevel == null)

{

baseLevel = new List<Int32>();

baseLevel.Capacity = 3;

}

if (baseCD == null)

{

baseCD = new List<Int32>();

baseCD.Capacity = 3;

}

if (!baseAttack.Contains(attack))

{

baseAttack.Add(attack);

baDamage.Add(damage);

baseLevel.Add(level);

baseCD.Add(cd);

}

}

public override void UpdateStat(Int32 amount, StatAction action, Stats stat, CombatType cType)

{

switch (stat)

{

case Stats.Health:

if (action == StatAction.Add)

{

if(Health != TotalHealth)

Health += amount;

if (Health > TotalHealth)

Health = TotalHealth;

}

else if (action == StatAction.Sub)

{

Health -= amount;

if (Health < 0)

Health = 0;

}

break;

case Stats.Mana:

if (action == StatAction.Add)

{

if(Mana != TotalMana)

Mana += amount;

}

else if (action == StatAction.Sub)

{

Mana -= amount;

if (Mana < 0)

Mana = 0;

}

break;

case Stats.Armor:

if (action == StatAction.Add)

{

if (cType != CombatType.Ranged && cType != CombatType.SpecialAttack)

lastArmor = Armor;

Armor += amount;

}

else if (action == StatAction.Sub)

{

Armor -= amount;

}

break;

case Stats.AttackPower:

if (action == StatAction.Add)

{

if(cType != CombatType.Ranged && cType != CombatType.SpecialAttack)

lastAttackAmount = amount;

AttackPower += amount;

}

else if (action == StatAction.Sub)

{

AttackPower -= amount;

}

break;

case Stats.Level:

if (action == StatAction.Add)

{

exp += amount;

if (exp >= totalExp)

{

TotalExp += 10;

exp = 0;

Level += 1;

TotalHealth += 10;

TotalMana += 10;

Health = TotalHealth;

Mana = TotalMana;

}

}

break;

}

}

public void UpdateWeapon(String itemName, Int32 itemValue, InventoryTyp itemType)

{

switch (itemType)

{

case InventoryTyp.Sword:

weapon = itemName;

weaponDamage = itemValue;

break;

case InventoryTyp.Ranged:

ranged = itemName;

rangedDamage = itemValue;

break;

case InventoryTyp.Shield:

shield = itemName;

shieldValue = itemValue;

break;

}

}

public void RemoveItem(String itemName, InventoryTyp itemType)

{

int index = inventory.IndexOf(inventory.First(s => s.Contains(itemName)));

inventory.Remove(inventory.First(s => s.Contains(itemName)).ToString());

inventoryValue.RemoveAt(index);

inventoryType.RemoveAt(index);

}

public override void Move(object combatMove)

{

throw new NotImplementedException();

}

}

}

**Delegate & Event**

This code can be found in Form3.cs

//Custom KeyPressEvent for the TreeView Control

public class InventoryViewEventArgs : KeyPressEventArgs

{

public Char Name;

public InventoryViewEventArgs(Char name): base(name)

{

Name = name;

}

}

private void KeyClick(KeyPressEventArgs e)

{

OnKey(new InventoryViewEventArgs(e.KeyChar));

}

public delegate void KeyHandler(object sender, KeyPressEventArgs e);

protected event KeyHandler Key;

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

{

KeyClick(e);

}