**Online Roleplaying Game System in Silverlight**

Ben Hall

CS470 Project Write-up

April 28, 2011

**Table of Contents**  1

Abstract 2

1. Introduction 2

2. Project Overview 2

2.1 Game Details 2

2.2 Proposed Features 4

3. Project Requirements 5

4. System Design 5

4.1 Loading Game Data 5

4.2 Player Character 6

4.3 Map 7

4.4 Combat 7

4.4.1 AI Characters 8

4.4.2 Rewards 9

5. Software Development Process 9

5.1 Developing the System and Interface 9

5.2 Testing 10

6. Results 11

6.1 Future Plans 11

7. Summary and Conclusion 12

Appendix A: User Manual 13

For Players 13

For Game Designer 19

Appendix B: Code 23

**Online Roleplaying Game System in Silverlight**

**Ben Hall**

**Abstract**

This project was done to develop a system to be used in an online roleplaying game, developed in Visual Studio using Silverlight and C#. It was developed in a way that the client can add data to it to develop it into a full game without needing experience writing code in C#. The program is also intended to be extended from its current state to add additional features not currently implemented.

**1. Introduction**

The client that the project was developed for was a friend who wanted to see his original game world turned into an online roleplaying game. He has been developing ideas for this world off and on for around 10 years now and wants to eventually share them across different mediums like online games, tabletop roleplaying games and books. The client initially had another friend who planned on developing the game, since the client has no programming experience of his own, however the game never made it past basic planning stages. Being interested in developing a game myself, I decided to create the system for this game as my project.

**2. Project Overview**

The goal of this project was to create a base system to handle the internal game features like combat, movement and inventory handling, while also being able to have game content like maps, items and opponents added without much difficulty, to extend the game. The overall goal, outside the scope of this project, is to have the system used in a full game, complete with a storyline and full graphics. The plan is to eventually do this in the future, as well as possibly creating “expansion packs” as well. The client wanted to have the game able to run in a web browser, so the decision to develop the system in Silverlight and C# was made with this in mind.

*2.1 Game Details*

As an online roleplaying game, the player will create a character to interact with the game world, and be able to improve the character as they play. The game uses a character class system, currently including Fighter, Rogue, Cleric and Wizard, to determine how the character develops as it gains experience and levels. The player may also improve their character by finding and using better items and equipment.



*Figure 1: Map Mode Interface*

The basic interface has two modes which allow the player to interact with the game. The map mode (Figure 1) is used to move the character across a map where the player can trigger events such as random combat encounters, movements between maps, and interactions with NPCs. This mode also allows the player to interact with their character’s inventory, to use recovery items or change equipment out of combat.



*Figure 2: Combat Mode Interface*

The combat mode (Figure 2) is used to interact with the combat features of the system. When a random combat encounter is triggered, the interface switches to this mode, allowing the player to select combat actions such as Fight, Use Skill or Run. Combat is turn-based, where turn order is based on the agility stats of the players and opponents involved. Combat will continue until one side, either the player or the opponents, are defeated, or until the player successfully runs away. If the player successfully defeats the opposing side, they are rewarded with experience points used to increase their level, as well as gold. If the player is defeated, combat ends and the player will lose half their current experience points, but be recovered to full health when returned to the map mode.



*Figure 3: NPC Shop introduction*

Gold earned in combat can be used when interacting with shop NPCs (Figure 3). Players can purchase items to be used in combat to create effects such as restoring health, removing or inflicting status effects, and dealing damage to opponents. Players may also purchase new equipment that can be used to increase the character’s damage or armor values. Shops also allow the player to sell items and equipment they have currently to receive gold.

*2.2 Proposed Features*

There are plans to implement an item creation system as well. This would allow the character to learn special crafting skills and combine materials to create items. Item creation would have a random chance to create special, exceptional versions of items that can’t be obtained in other ways.

The system was also intended to include a chat feature to communicate with other players, as well as features to interact with others like trading items and forming adventuring groups. The game system currently only supports a single player however, so these features will need to be implemented in the future. Since the system is currently functioning as a single-player game, the ability to save a character on the user’s machine between sessions was added. Saving is done whenever a change to the character is made, for example, using items or equipment, finishing combat, or moving across the map. This also allows the system to be used in a large-scale single-player game if the client decides to go in that direction instead.

**3. Project Requirements**

The main requirement that was given was that the online game should run in a web browser rather than its own client. The reason behind this was that it would be more accessible to players regardless of their location if it can be played in any web browser. The decision to use Silverlight was made with this in mind, since Silverlight is supported in most newer versions of the commonly used web browsers like Internet Explorer, Mozilla Firefox and Google Chrome.

Before beginning the project, the client gave an overview of how the interface should look and how the system should work, but gave few specific details to follow, so there were no other large requirements. Because of this, the client was consulted occasionally during the development process when clarification was needed.

**4. System Design**

Being a Silverlight program, it is designed to run on the user’s machine, and depends heavily on user interaction, such as with selections or button clicks. Most of the algorithms used are fairly simple, with the decision trees used to determine AI opponent combat moves being the most complicated.

*4.1 Loading Game Data*

When the Silverlight program first loads it initializes arrays that hold required game data like information about maps, opponents, skills and items. This is done by including resource text files in the Silverlight project, which contain lines of data that are used to create objects used by the program, such as GameMap or AICharacter objects. For example, in the test data used by the program currently, the AICharacter.txt file includes the line:

 *Slime,5,5,5,5,5,5,BalancedAI,1,4,0.05,5,testcomp.jpg,Poison Strike*

The program reads this file and creates a BalancedAI object in the opponentList array that has the name “Slime,” has strength, stamina, agility, intelligence, faith and persona values all equal to 5, has a minimum damage of 1 and maximum damage of 4, has an armor value of 0.05, gives 5 gold as a reward for defeating it, uses the “testcomp.jpg” file in the Images/ directory in the project as a portrait, and knows the skill “Poison Strike.” The object can then be accessed by the program whenever the player enters combat that includes a Slime opponent.

The resource files that are read can be extended to include any number of lines, as long as the data contained in the lines is valid. If, for example, the line above was missing the armor value and had nothing in its place, the “Slime” opponent would not be initialized properly, however any lines in the file following that line could still be initialized if they are formatted properly.

The text files for Skills, Items, Equipment are similar to the example given above from the AICharacters.txt file, however the files used for maps differ slightly. Maps require multiple files to create GameMap objects properly. The program first reads the Maps.txt file, which contains lines for each map used in the game, and references other files to create those maps.

 *0001,Map0001.jpg,Map0001.txt,0.01,Map0001-Encounters.txt*

This example line shows a map that has the MapID 0001, uses the Map0001.jpg file as a background and must use the Map0001.txt file to initialize the Boolean grid showing movable spaces, as well as any events located on the map. This file has a 15x10 symbol grid that is used to create the movable spaces array, which is followed by each event on the map, if any exist.

The example line also shows the map has a random encounter rate of 0.01 (1%), and uses the Map0001-Encounters.txt file to create an Encounter associated with the GameMap, which is referenced when the player triggers a random encounter on the map. The encounter text files have lines which contain the rate that the specific enemy group is encountered, as well as which opponents are found in that group.

*4.2 Player Character*

When the player decides to create a new character, they are prompted to choose a character class, which determines the character’s initial statistics, skills and equipment, as well as how the character develops as it levels. When a character earns enough experience points to gain a level, the program uses a resource text file based on the character’s class to determine which stats increase at that level and by how much. It also determines if a character learns a new skill. For example, take the following line from WizardLevelTemplate.txt:

 *4,0,0,1,1,1,1,Poison*

This tells the program that when a Wizard player character reaches level 4, his strength and stamina do not increase, and his agility, intelligence, faith and persona values each increase by 1. The character also learns the skill “Poison” at this level. When a character attains a new level, values based on the character’s level like maximum health and magic values, are recalculated based on the appropriate formulas in the GameCharacter class.

A player character’s attack and defense values are based on the character’s equipped Weapon and Armor objects, respectively. Both the Weapon and Armor classes extend the Equipment class. A player may select which pieces of equipment it wishes to equip, but may only have at most a single weapon, shield and body armor equipped at one time.

When the player equips an item, if the PlayerCharacter already has values in equippedWeapon, equippedBodyArmor or equippedShield, they are replaced with the newly equipped item. In the case of equipping two-handed weapons while wearing a shield, or vice versa, the system will unequip the appropriate equipment automatically to make sure the character is only using what equipment it can hold in two hands.

Once a piece of equipment is worn or taken off by the character, its damage and armor values are immediately updated. Equipment may not be changed during combat, however, and if a piece of equipped equipment is sold in a store, it is automatically unequipped.

*4.3 Map*

The map interface uses GameMap objects to handle where the player can move, as well as how events are handled. A two-dimensional Boolean array the size of the map (15x10) is checked whenever a player tries to move into a space. If the value in the array is false at that position, no move is made.

If a player does move into a new space, the program first checks to see if an event exists in that space and triggers the event if it does. There are two main types of events, MapMoveEvents and NPCEvents. MapMoveEvents contain the ID of the map and the position on that map that the character moves to when the event is triggered. NPCEvents will open a dialog box over the map view where the player can interact with NPCs. Figure 3, above, is an example of an NPCEvent. If no event exists on a square, the program then checks the random encounter rate of the map and randomly determines if a combat encounter is triggered. If so, it determines which enemy group is encountered and initiates combat.

*4.4 Combat*

When combat starts, the view switches into combat mode (Figure 2), changing the map view area into a combat view. It starts a separate thread to handle the main combat loop, where opponents will take their turns and wait for player input during the player’s turn. Combat continues as long as both sides have at least one participant still alive, or until the player runs away successfully. These conditions are checked at the beginning of each turn in combat, breaking the loop when combat should end.

*4.4.1 AI Characters*

AI Characters use decision trees to determine which moves they will make on their turn based on the current state of combat. There are currently four types of AIs, Attacker, Balanced, Defender, and Healer. Each type is a separate class that extends the AICharacter class and overrides the MakeMove() method, which is called on an opponent’s turn during combat to decide which move should be made.

Along with using decision trees, at each branch the AI randomly decides if it will take the “best” move or skip to the next branch. This random chance is based on the AI character’s intelligence stat with more intelligent opponents being more likely to use the best move available. If the AI does skip its “best” move at a branch, the chance of it skipping its next best move is decreased, until it either takes its “best” move 100% of the time, or it reaches the end of a branch.

The AttackerAI prioritizes using strong offensive moves and typically ignores the state of its allies and is more likely to attempt to do damage than improve its own condition.

BalancedAIs randomly decide, with a 50/50 chance, whether they will choose an offensive or defensive action. If an offensive action is taken, it typically uses the most powerful skill it knows, or uses a standard Fight action. If a defensive action is taken, it is more likely to try to help itself than its allies, but may help allies that are in a worse condition than it is in.

DefenderAIs prioritize boosting the defense of themselves and their allies, as well as protecting injured allies with skills like Cover, which takes the physical damage of a selected ally. DefenderAIs with many defensive skills are more likely to use those skills than attack if allies are present, however if the DefenderAI is alone, it will be more likely to attack than usual, since it can no longer rely on allies to deal damage.

HealerAIs try to use healing skills on allies when appropriate, to heal damage or remove negative status. After that, HealerAIs will attempt to boost allies with positive status effects, like increasing defense values, or hinder opponents with negative status effects.

At certain branches of the decision trees, AICharacters use methods like GetCompHealSkill() and GetCompAttackSkill() to find skills of a certain type (healing, attack, or defensive) that can be used. These methods will either return the skill ID of a skill that can be used, or return -1 to show that no relevant skill is available, which tells the AICharacter to skip that branch.

At the ends of the decision trees, AICharacters may call methods like UseBestAttack() or UseBestDefense() which will find the best move available of that type and execute it, defaulting to a standard physical attack if no attack skills are available, or the Defend command if no defense skills are available.

*4.4.2 Rewards*

When the player successfully defeats all opponents in battle, the character receives experience points based on the difference between the opponent’s level and player character’s level, for each opponent in combat. The base experience given for each opponent defeated is 100 points if the player character is the same level as the opponent. For each level the player is below the opponent defeated, 10 experience points are added up to a maximum of 200 experience points. For each level the player is above the opponent, 10 experience points are deducted, to a minimum of zero points awarded.

An AICharacter’s level is determined by its stats and skills known. It works on a point system, where each point in a stat is worth a single character point, each offensive skill is worth 15 points and each defensive skill is worth 7 points. Currently, an AICharacter has one level for every 30 points it possesses, however this may change in the future for better game balance.

**5. Software Development Process**

Using the information gathered from the client about the look and feel the system should have, I decided to develop the program by starting as far into the code as possible and work my way out, extending from what I had. I made a simple interface initially using Visual Studio, however the actual code development started with the combat system.

*5.1 Developing the System and Interface*

Initially I developed a text-based combat system in order to get the features of the system to work properly, which I could then link into a combat GUI interface. This made it much easier to find and fix bugs in the combat system, as well as develop additional features, without worrying about GUI-related problems as well.

I started by implementing the turn-based part of the combat and each of the combat commands (Fight, Defend, Use Skill, Use Item and Run) from a player perspective, with the AI opponents doing nothing. As I was developing these commands, I would have to extend the program to include things such as the Skill and Item classes that were needed for using the related commands.

Once I had all of the player’s commands working properly I was able to focus on developing the different AICharacter subclasses with each of their specific decision trees. This required adding methods for the AI character to select an appropriate skill of a certain time (ex: GetCompAttackSkill()), but actually using skills or commands like Fight and Defend could be done using the methods used by the player character.

When I felt the combat system was bug-free, as far as I could tell, I created the interface needed to display the combat and allow the player to interact with it. Some of the combat code needed to be trimmed, as it was no longer needed in the GUI version of combat.

From this point, moving outward, I developed the map portion of the interface, to allow the player to move across the current map and between maps, as well as trigger random combat encounters.

Up until this point, all data used by the program was hard-coded in, so I needed to find a way to have the program load data that could be changed without going into the code. I decided to use text files as resources in the Silverlight program that would be added to the .xap file when building the program. The text files require a strict format for entering data, which may not make sense to someone unfamiliar with the format, however it would be easier for someone unfamiliar with C# to learn how to input data into the text files than hard-coding it in. This, however, is also just a temporary solution that I have plans to change in the future as the program grows.

Once I had the basic map movement and combat system working in the GUI interface, I started focusing on a lot of smaller pieces, such as adding functionality for character leveling, equipment use, map event handling and new character creation.

Since the scope of the project was just to develop the system that could be used in the game, it currently only uses data to test and show the different features of the program. That is why it only contains two maps, a handful of AI opponents, items and skills, and why it has very basic graphics. In the plan for the development of the full game, I will not be involved in creating the story, maps or graphics anyways.

During the development process I had a few meetings with the client to show off what I had as large parts were completed, such as the completion of the combat GUI interface, and the inclusion of the map interface with random encounters. Other than that and getting answers to a few quick questions when they came up, I did not meet with the client much about the project.

*5.2 Testing*

All of the testing done for the program was done as I was implementing new features, so I would only have to worry about a few problems at a time. This seems to have worked fairly well, as I would have probably been overwhelmed if I had to deal with bugs in the GUI and the background code at the same time, especially for large things like combat.

*5.3 Proposed vs. Actual Implementation Time*

In my initial proposal, I estimated that I would spend about 5 hours in design and a combined 72 hours in implementation and testing. I believe I should have planned for more time in initial design, since I used more than 5 hours in total with designing each addition as I came to them. I did not track exactly how much time I spent working on the implementation, but I know the initial estimate was much lower. I would guess that I spent nearly that much time developing the combat system alone.

**6. Results**

While my initial thoughts on the project did not include developing a full game during the semester, I feel I did not accomplish as much as I had hoped I would in this time. This is mostly due to underestimating the time that it would take to develop the basic parts of the system like combat and map movement. This is a project I am looking forward to working on more in the future, both in implementing new features that are not currently included, as well as working with the client to develop a full game that can be put online, whether it remains single-player or has multiplayer functionality as well.

*6.1 Future Plans*

There are many things I hope to add to this program in the future, including but not limited to the features that I had originally wanted to include but could not under the time constraints. This includes things like an item creation system, and having a server to run the game on to include the chat system and player interactions for a multiplayer game. I also would like to include things like animations for the combat system and a quest system where NPCs give players tasks to complete to earn special rewards. The system will most likely also require a lot of tweaking of formulas to balance out characters for a more enjoyable experience for players. As it is currently, the system works, but some character classes for example are better choices than others.

One large thing I would like to add, that I would have considered doing in this project if I had much more time, is to develop a separate program that could be used by the client to add content to the game. For example, it would have a mode for creating maps, where the user would import an image to be used for the background and select the squares on the 15x10 grid where a player can move. The user would also select where events occur and give the details of those events. This program would then add that content to the overall game project, to completely remove the need to edit or add text resource files. With a program like this, I would also likely remove the use of text resource files and use a different filetype to store data.

**7. Summary and Conclusions**

This project was developed in Visual Studio 2010, using Silverlight and C# with the goal of creating a system that can be extended and used for an online, browser-based roleplaying game. I was not able to complete all of the features proposed initially at the beginning of the semester due to underestimating the time needed for some of the features, however I am satisfied with the current state of the program and look forward to continuing work on it.

Overall, I enjoyed working on this project. It was full of firsts for me, as I had never used Silverlight, C# or Visual Studio before. I am very glad to have had Visual Studio 2010 to work with because of the built-in GUI designer, which made working with the Silverlight components a lot easier. Having experience programming in Java made adapting to C# a lot easier, since the two languages are really similar with most of the basic uses I needed. I am still warming up to Silverlight, however. Using Silverlight required doing a few things in different ways than I would have done had I just used C# or Java to develop the game system, such as not being able to rely on I/O and needing to instead add resource files to the program. I really did enjoy watching the program develop, step by step, from its initial basic combat system to the product it is now.

I am also glad to have gained experience working on a larger project, since most of the experience I have had up to this point was working with small projects that could be completed in just a few days, at most. By working on this project I have realized that I tend to underestimate the amount of time it takes to develop programs. I also believe that I would benefit from and enjoy working with at least one other person on a program of this scale or larger. I think it would be beneficial to have other ideas on how to implement things, as well as having someone who may have more experience in some areas than I do that I could learn from.

**Appendix A: User Manual**

**Minimum Requirements**

Silverlight 4 plug-in in a compatible web browser

(Compatible browsers include: Internet Explorer 7+, Mozilla Firefox 3+, or Google Chrome)

**For Players**

**Starting the Program**

Since the program is not hosted on a server currently, it can be accessed by opening the included OnlineGame.html file, which will load the Silverlight program. If Silverlight is not installed on your machine, you will most likely be prompted to download the plug-in for your current browser.

**Playing the Game**

When the program loads, you will see the following Welcome screen:



If this is the first time loading the game on this machine, the Load Character button will not be available until a new character is created, since no save data exists. Note that saving a character is done automatically whenever a change is made to the character.

Clicking the New Character button will bring up the character creation screen:



You will be prompted to select a name for your new character, as well as selecting a class and portrait. When you select a class, the starting stats for that class will be displayed on the right, and when selecting a portrait, a preview of the portrait will be shown below. One you have given your character a name, class and portrait, hitting Create Character will create the new character and move you to the first game map.



In the upper left is the map view. Here is where you will interact with the world by moving your character. Movement is done using the buttons on the right, labeled North, East, West and South. You will also see the buttons Search, Camp and Harvest. Search is used to scout an area before camping, and may result in triggering a random encounter.

If no encounter is found, the Camp button becomes enabled until you camp, or move to another space. Camping will restore your character’s health and magic values to full.

The Harvest button is disabled and currently has no function, but in the future, searching an area may result in finding raw materials that can be harvested and used in item creation.

Below the map view is the chat system. While there is currently no chat system to communicate with other players, the game will use the box to display information, such as skill usage or displaying combat logs.

On the upper right of this view is your character’s information. It shows the character’s current level, stats, health and magic values, as well as damage and armor values (based on currently equipped equipment) and experience and gold. Below those values is a drop-down box that can be used to use skills that are available outside of combat, like Heal.

The lower right corner is for handling your character’s inventory. Clicking the different tabs will let you explore the different types of items in your inventory: usable items, equipment and materials. Selecting an item or piece of equipment in this view will display a description of the item below the box.



Equipped items will display an (E) next to the quantity in the box, and when selected, the Equip button will instead display Unequip. Equipping items will change your damage values (for weapons) or armor values (for body armor and shields), and will also unequip any equipment using the same equipment slot. There are two-handed weapons, which may not be wielded at the same time as a shield and will unequip any equipped shield when used.

You may also drop items you do not wish to keep, although there is no limit to your inventory at the moment. You also will not be allowed to drop the last of a piece of equipment that you have equipped.

**NPC Shops**



There are locations where NPCs will sell you items and equipment, or allow you to sell your own items and equipment. Selecting “Buy” will open up a box with the shopkeeper’s inventory, split into categories like your character inventory, where you can browse through items and purchase them if you have the required amount of coins.



The merchant’s “Sell” option will open a similar box, which will display your own inventory and allow you to sell items at their base price. Be careful when selling equipment though, if you sell the last of a piece of equipment you have equipped, it will be unequipped and you may not be able to purchase a replacement!

**Map Movement**

Moving to specific spaces may trigger a move to another map. In the map you start off initially, if you move to the space on the far right side, you will move to this map:

****

Each map has its own encounter rate and enemy encounter lists, so moving to different maps may allow you to fight different enemies, at a higher or lower rate.

**Random Encounters**

Each time you move to another space, or search an area, you have a chance of triggering a random combat encounter. If this happens, the game will shift to the combat view where you will be able to fight the opponent(s) you encounter.



Combat is turn-based, based on your own agility compared to the agility of your opponents. Combat continues until you defeat your opponents, run away successfully, or you are defeated. You may select an opponent by clicking on its portrait, with the currently selected opponent appearing in the largest section in the middle.

In combat have the option of five commands, at the bottom of the combat view:

*Fight:* Use a physical attack based on your character’s current damage value.

*Defend:* Reduce physical damage taken by 50% until your next turn.

*Use Skill:* Use the skill selected on the left of the button. If it is a beneficial skill, it will be used on your character. If it is an offensive skill, it will be used on the currently selected opponent.

*Use Item:* Like with skills, beneficial items will be used on your character and detrimental items will be used on the currently selected opponent.

*Run:* You attempt to run from combat, with a success chance determined based on your agility and the average agility of your opponents.

If you defeat your opponents, you will earn experience points and gold coins for each opponent that is defeated. You will also gain a level for every 500 experience points you earn.

If you are defeated in combat, you will be returned to your previous map with full health and magic, but will lose half the experience you had earned for your current level.

**Gaining Levels**

When you earn at least 500 experience points, you will gain a level in your character’s class. Gaining a level will increase some of your stats and your character may learn additional skills.



**For Game Designer**

Currently all data used in the game is included in resource files added to the Silverlight .xap file when compiled, so adding new data to the game will require recompiling the project in Visual Studio 2010 after making the required changes to the text files and adding any images that are needed. With all files, the format is very important. Most use commas to separate categories and the program depends on the use of these commas to read the file correctly. There should also be no spaces after character separators.

**Adding AICharacters**

Add lines to the AICharacters.txt file in the following format:

*Name,strength,stamina,agility,intelligence,faith,persona,AIType,minDamage,maxDamage,armorValue,goldReward,portraitNameUsed,SkillsKnown*

Stat values = whole number between 1 and 100

AIType = AttackerAI, BalancedAI, DefenderAI, or HealerAI

minDamage = positive whole number

maxDamage = positive whole number

armorValue = number between 0.0 and 1.0. Armor value is a percentage of how much damage is reduced by, so a value of 0.95 will reduce damage by 95%.

goldReward = positive whole number

portraitName = name of an image file located in the Images/ project directory

SkillsKnow = list of skill names, separated by periods (ex: Hard Strike.Bandage)

**Adding Equipment**

The Equipment.txt file follows the following format for weapons:

*EquipmentType,Name,GoldValue,minDamage,maxDamage,isTwoHanded*

And the following format for armor:

*EquipmentType,Name,GoldValue,armorValue,equipmentSlot.*

EquipmentType = Weapon or Armor

Name = Any string

Gold Value = Any positive whole number

minDamage = any positive whole number

maxDamage = any positive whole number

isTwoHanded = true or false

armorValue = any number from 0.0 to 1.0

equipment Slot = Body or Shield

**Adding Items**

The Items.txt file follows the following format:

Name,goldValue,isMagic,isDamaging,isHealing,numValue,isBeneficial,usableOutOfCombat, poisonEffect,pDefenseEffect,mDefenseEffect

Name = Any string

goldValue = any positive whole number

isMagic = true or false (used in determining modifiers for healing or damage)

isDamaging = true or false

isHealing = true or false

numValue = positive whole number (used in determining strength of the heal or damage effect)

isBeneficial = true or false (affects how targeting is selected when the player uses it)

usableOutOfCombat = true or false (true allows the item to be used out of combat)

poisonEffect = -1, 0 or 1 (-1 means the item removes poison, 1 means it inflicts it, 0 has no effect)

pDefenseEffect = between -1.0 and 1.0 (negative values reduce the target’s physical defense, positive values increase it, and 0 does nothing to defense)

mDefenseEffect = between -1.0 and 1.0 (same as pDefenseEffect, but for magical defense)

**Adding Skills**

Adding lines to the Skills.txt file uses this format:

Name,skillCost,isMagic,isDamaging,isHealing,numValue,isBeneficial,poisonEffect,pDefenseValue,mDefenseValue

The values expected for skills are the same as the values expected for items of the same name (above).

**Adding Character Classes**

First add the name of the new class to PlayerClasses.txt on its own line

Then add a text file to the resources with the name <NewClassName>LevelTemplate.txt

The file should have lines with the following format:

*Level,strength,stamina,agility,intelligence,faith,persona,skillLearned*

The first line is used to determine the starting stats of a character of that class. To balance with existing classes, the stat total should equal 60 and the character should know one skill.

Each following line should have stat values equal to whatever increase is made to that stat at that level. The current classes gain a total of 19 stat points every 4 levels, and learn a new skill every three levels. If no skill is to be learned, instead write NONE at the skill to be learned.

**Adding Character Portraits**

On a new line in the CharacterPortraits.txt resource file, add the name of an image file located in the Images/ directory of the project.

**Adding Maps**

First open the Maps.txt file in the GameData/Maps/directory and add data with the following format:

*MapID,mapBackgroundImage,MapInfoFile,MapEncounterFile*

MapID = unique ID identifier for the new map

mapBackgroundImage = name of an image file located in the Images/Maps/ directory

mapInfoFile = name of a file in the Maps/ directory

mapEncounterFile = name of a file in the Maps/ directory

The new map will also need two additional files in the maps directory.

The first (typically named <MapID>.txt) should have a symbol grid made up of 15 characters each on the first 10 lines to show which spaces of the map can be moved to, with an X representing a square that can’t be moved to and a . representing a square that can be moved to.

Following the symbol map will be any events that are located on the map. MapMove Events follow this format:

*MM,xPosThisMap,yPosThisMap,MapIDToMoveTo,xPosNextMap,yPosNextMap*

MM = shows that the event is a map move event

xPosThisMap = the position on the x-axis of this map the event is located

yPosThisMap = the position on the y-axis of this map the event is located

MapIDToMoveTo = the name of a map ID that the event moves the character to

xPosNextMap = position on the x-axis the character will be moved to

yPosNextMap = position on the y-axis the character will be moved to

A shop event will use this format:

*Shop,xPos,yPos,NPCName,NPCportrait,priceIncrease,ShopFile,NPCMessage*

Shop = shows that the event is an NPC event for creating a store

xPos = the position on the x-axis of this map the event is located

yPos = the position on the y-axis of this map the event is located

NPCName = the name of the NPC to be displayed

NPCPortrait = the name of the file in the Images/ directory to be used as the NPC’s portrait

priceIncrease = this should be above 1.0, this is the amount the NPC multiplies the value of the item by to determine how much it is sold for

ShopFile = the name of a file in the GameData/Shops/ directory listing the NPC’s inventory

NPCMessage = the welcoming message the NPC gives when the event is triggered.

The shop txt file should have lines that follow this format:

*ItemType,ItemList*

ItemType = I or E (for Item or Equipment)

ItemList = a list of all items of the designated type that the NPC sells, separated by commas

This file should not have more than one line for each item type.

**Appendix B: Code**

All code used in the project is attached in a zip file. Also included in the zip file is the most recent .xap file to run the Silverlight program, and an html file that loads the .xap file, which will allow you to run the program.