University of Nevada, Reno

Combining Role Playing Game Constructs Toward Real Time Strategy Games

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Engineering

By:
Bradford Allen Towle Jr.

Dr. Monica Nicolescu/Thesis Advisor

December 2007

University of
Nevada, Reno

THE GRADUATE SCHOOL

We recommend that the thesis
Prepared under our supervision by

BRADFORD ALLEN TOWLE JR.
Entitled

Combining Role Playing Game Constructs Toward Real Time Strategy Games

Be accepted in partial fulfillment of the
Requirements for the degree of
MASTER OF SCIENCE

Monica Nicolescu, Ph.D., Advisor

_____
Sergiu Dascalu, Ph.D., Committee Member

_____
James Henson, Ph.D., Graduate School Representative

_____
Marsha H. Read, Ph.D., Associate Dean, Graduate School

_____

December 2007

Abstract

The computer game industry has grown to a million dollar industry with new titles coming out every month. However, with all these great achievements the video game industry does have one significant problem: the ways in which the games are played are very similar. One particular genre for which this is true is the group of real time strategy games. Almost all of these games have the same structure, in which players first build and upgrade a base, with the property that the more upgraded the base the more powerful the units that can be built, and the more powerful the units the better chance of winning. The prospects for making a successful game reward a company with perhaps millions of dollars more games are now flooding the shelves and quantity has become more important then quality. This thesis presents a new way to look at game design, by analyzing different computer game genres and incorporating new attributes to the genre which needs improvement. In this case four major components (character equipment, character advancement, character customization, and character classification) were taken from the role playing genre and combined to enhance the real time strategy game using five rules. In addition, this thesis demonstrates a successful implementation of the unified modeling language (UML) in game design. By demonstrating game design through UML programmers could see potential problems with game play before the entire code was written. This thesis also presents a pilot game which was developed in order to prove that real time strategy games could in fact be enhanced with role playing constructs.

Dedicated to Albert Towle a loving grandfather and great inspiration, your memories and kindness will never be forgotten. Thank you for showing the value of education and allowing me the chance to pursue graduate school.

Special thanks to:

Dr. Monica Nicolescu – for being there for me when no one else could help.

Dr. Sergiu Dascalu – helping me even when you had no time to spare.

My father, Brad Towle - for encouraging me through the process.

The Carson City Highway Patrol – for inspiring the name of the pilot game.

Table of Content:

1 Chapter 1: Introduction and Background 1
1.1 Introduction 1
1.2 Background of Role Playing Games (RPG) 3
1.3 Background of Real Time Strategy Games (RTS) 5
1.4 Previous Work 8
1.5 Present Work 10
1.6 Overview of the Thesis 11
2 Chapter 2: Method 12
2.1 The Initial Seven Rules 13
2.2 The revised Set of Rules 15
2.3 Prototype Game Construction using Game Maker 17
2.4 Game Maker Limitations 17
2.5 Method used and an Overview of the Pilot Game “Fuzz’s Revenge” 18
3 Chapter 3: Software Requirements for “Fuzz’s Revenge” 21
3.1 The Initial Requirements 21
3.2 Use Case 25
Use case modeling 25
3.3 Primary Scenarios 28
3.3.1 Select Unit 28
3.3.2 Move Unit 29
3.3.3 Unit Attack 29
3.3.4 Unit Victory 30
3.3.5 Unit Defeat 31
3.3.6 Unit Statistics 31
3.3.7 Unit Attribute Upgrade 35
3.3.8 Unit Class Upgrade 37
3.3.9 Building Construction 39
3.3.10 Dungeon Exploration 42
3.3.11 World Start 43
3.3.12 Build Unit 44
3.3.13 Build Weapon 45
3.3.14 Equip Weapon 47

3.3.15 Manage Resource 48
3.3.16 Game Management 49
3.3.17 Special Abilities 49
3.4 Requirement Fulfillment 52
3.5 Modified Requirements 54
3.6 Modified Use Cases 55
3.6.1 Group Assignment 55
3.6.2 Group Selection 56
3.6.3 Group Movement 56
3.6.4 Group De-Selection 57
4 Chapter 4: High Level Design of Fuzz's Revenge 58
4.1 Mouse Input 59
4.2 Display System 65
4.3 Combat System 67
4.4 System Management 69
5 Chapter 5: Fuzz's Revenge Illustrated 72
5.1 Units 72
5.2 Statistic Window 73
5.3 Weapon Building 74
5.4 Weapon Equipping 75
5.5 Group System 77
6 Chapter 6: Future Work 82
6.1 Future Work 82
6.1.1 Multiplayer 82
6.1.2 New Units 83
6.1.3 Balancing 83
6.1.4 New Combinations of Genre Characteristics 84
6.1.5 Improved Software Engineering Practices and Notations for Game Development 84
7 Chapter 7: Conclusion 85
8 Reference 87

List of Figures:

Figure 3-1:Break down of Actors in "Fuzz's Revenge" 27
Figure 4-1: High Level Diagram of Fuzz’s Revenge 58
Figure 4-2: Modules Dedicated to Mouse Control 59
Figure 4-3: flowchart of the Left Mouse Button’s Multiple Functions 61
Figure 4-4: Unit Movement Controlled by Right Mouse Button 62
Figure 4-5: Equipping Fuzz Units With Weapons 63
Figure 4-6: Attribute Upgrade Process 64
Figure 4-7: Class Upgrade Process 64
Figure 4-8: Display System Process 66
Figure 4-9: Basic Combat System Process 67
Figure 4-10: Fuzz Unit Victory Process 68
Figure 4-11: Fuzz Unit Defeat Process 69
Figure 4-12: Resource Management Process 70
Figure 4-13: Enemy Unit Process Manager 71
Figure 5-1: The Enemy Unit (Left) The Friendly unit (right). 72
Figure 5-2: The Unit's Statistics Window 73
Figure 5-3: Available Weapons in the Magic Shop 74
Figure 5-4:Weapon Selection Menu 75
Figure 5-5: Equipping the Magic Hat 76
Figure 5-6: Assigning a Unit to a Group 77
Figure 5-7: A Selected Group 78
Figure 5-8: The Opening Screen 78
Figure 5-9: After construction of Home Base occurs new buildings are available 79
Figure 5-10: Fuzz unit entering Dungeon 80
Figure 5-11:Fuzz unit inside a Dungeon 81

List of Tables:

Table 2-1: Calculating Level Thresholds 19
Table 3-1: Initial Requirements for the Pilot Game “Fuzz’s Revenge” 22
Table 3-2: Use Cases Designed to Implement Category One Requirements 25
Table 3-3: Selecting a Unit 28
Table 3-4: Selecting a Building 28
Table 3-5: Moving a Unit 29
Table 3-6: Moving Unit when a Collision Occurs 29
Table 3-7: Attacking an Enemy with a Unit 30
Table 3-8: Defeating the Enemy 30
Table 3-9: Friendly Unit Defeated 31
Table 3-10: Statistics Window Display without Skill Points 32
Table 3-11: Statistics Window Display with Skill Points and No class upgrades 32
Table 3-12: Statistics Window Display with Skill Points and First Class Upgrades 33
Table 3-13: Statistics Window Display With Skill Points and Wizard Class Upgrades 34
Table 3-14: Upgrading Unit Defense 35
Table 3-15: Upgrading Unit Attack Range 36
Table 3-16: Upgrading Unit Movement Speed 36
Table 3-17: Upgrading Unit Rate of Fire 36
Table 3-18: Upgrading a Unit's Attack Power 37
Table 3-19: Upgrading a Unit's Regeneration 37
Table 3-20: Upgrading a Unit from "NEWB" to Soldier 38
Table 3-21: Upgrading a unit from "NEWB" to Wizard 38
Table 3-22: Upgrading a unit from "NEWB" to Priest 38
Table 3-23: Upgrading a unit from Wizard to Fire Sorcerer 39
Table 3-24: Upgrading a unit from Wizard to Ice Mage 39
Table 3-25: Constructing the Home Base 40
Table 3-26: Constructing the Marketplace 41
Table 3-27: Constructing the Bio-Dome 41
Table 3-28: Constructing the Gold Mine 41

Table 3-29: Constructing the Factory 42
Table 3-30: Constructing Magic Shop 42
Table 3-31: Friendly Unit Enters a Dungeon 43
Table 3-32: Unit Leaves a Dungeon 43
Table 3-33: Game Initiation 44
Table 3-34: Building a Unit 44
Table 3-35: Building the Holy Stave 45
Table 3-36: Building the Assault Rifle 46
Table 3-37: Building the Bazooka 46
Table 3-38: Building the Magic Hat 46
Table 3-39: Building the Fire Wand 47
Table 3-40: Building the Ice Staff 47
Table 3-41: Equip Weapon Correctly 48
Table 3-42: Attempting to Equip an Invalid Weapon 48
Table 3-43: Resource Management System 49
Table 3-44: Game Management System 49
Table 3-45: Soldier Special Dash Ability 50
Table 3-46: Priest Special Heal Ability 51
Table 3-47: Wizard Special Shield Ability 51
Table 3-48: Fire Sorcerer Special Fire Elemental Ability 51
Table 3-49: Ice Mage Special Frost Nova Ability 52
Table 3-50: Requirement Fulfillment Table 52
Table 3-51: Additional Requirements 54
Table 3-52: Additional Use Cases 55
Table 3-53: Group Assignment for Individual Units 56
Table 3-54: Group Selection 56
Table 3-55: Group Movement 57
Table 3-56: Group De-Selection (automatic) 57

1 Chapter 1: Introduction and Background

1.1 Introduction

With the recent interest in video games, the cost of creating a best selling title is high and increasing [2] [11] . Many video and computer games have a production price surprisingly close to the cost of producing a Hollywood movie. Even more surprising, the reward for creating a best seller game can rival a blockbuster movie (e.g. The week **Halo 2** was released Microsoft made an estimated 125 Million dollars a day, out grossing the opening movies that week). Because of this, many computer games companies adopt the "stick with what works strategy" and simply add different stories and graphics to games that otherwise have very similar game play strategies. For example, **Dawn of War** and **Company of Heroes** have different themes, but very similar mechanics on which the games are built. Another possible reason for similar game play is the re-use of code: game engines are usually arduous to write from scratch and therefore, if a company can use the same game engine for multiple games it will save money. Historically, there are six main computer game genres each with a variety of different subgenres. These genres are:

First Person Shooter (FPS) - As the name implies, this genre gives the player a look through the character's eyes. This style of game was almost always a violent game in which players would go around with large guns and destroy one another. Hence the name shooter was added.

Role Playing Games (RPG) – This genre focuses on an individual or a small party of individuals. In this genre, a party of characters must progress through the story and grow stronger (level up) in order to achieve victory over whatever foes assail them. Character customization, development and advancement are vital.

Real Time Strategy (RTS) – This genre allows the player to command vast armies, whether in an ancient medieval, fantasy or futuristic setting. This game will almost always incorporate resource management and gathering, base construction, and building armies. The player must then make strategic choices as to where to spend the resources that have been gathered.

Simulation Games – Historically, this genre has included mostly flight simulators or other such programs that give the feel of manipulating a vehicle. More recently, a new type of game has emerged, which simulate a certain situation or place. The player does not have direct control of the characters, but can build or add things to the environment in order to change they way they act. **Roller Coaster Tycoon** and the ever popular **Sims** are good examples of this new style of simulation games.

Sports/Board Games – Perhaps some injustice is done by trying to condense these games into one genre, however, games that have been around before computers were available, and which were transferred onto this medium, fall under this category. The sports games have been increasing in popularity for a long time. However, the board games and some of the early puzzle games seem to have a small, but stable popularity.

Adventure/Platform Games - Many of the older "platform" games, where a character must jump from platform to platform, fall under this category. Good examples are the original **Super Mario Brothers** and **Donkey Kong**. With improvements on graphics, the genre has changed from platforms to a three dimensional (3D) setting, where the player can move characters around and try to accomplish a goal. Good examples of these are **Super Mario 64** and **Zelda 64 "Ocarina of Time"**.

The pilot game created for this project combines various aspects of RPG and RTS games by applying individual character development to characters in an RTS setting. Therefore, the project must incorporate the individuality found in the RPG games while maintaining its army command structure. To allow this, four different role playing game constructs

have been programmed into this project that enable the player to equip items, strengthen power, customize attributes, and change classes of individual characters. With these added capabilities, RTS games become more dynamic and player skill plays a larger role in the game outcome.

The next two sections will give background for both role playing games and real time strategy games. This will familiarize the reader with the information to understand various elements necessary of a RPG's and RTS's game play and where these elements were derived from.

1.2 Background of Role Playing Games (RPG)

The author of the book Dungeon and Dreamers [1] explains that computer games were first born on college campus mainframes. Of course, there were no displays on these early computers and everything was text based and controlled by keyboard input. The author describes one of the earliest games he had played which was very limited; a character in a dungeon was controlled by typing commands on the keyboard only. The moves were generally limited to moving North, South, East, West, attack and run. However, this marked the beginning of the computer role playing games that have become very advanced and popular today. Role playing games had been around before, but were limited to table top rules where dice, pencil and paper were used for the mechanics. An example of this type of game is **Dungeons and Dragons** [14] , which allowed the player to create characters in a medieval fantasy realm. According to the official Dungeons and Dragons website this is how the game is described:

"This is the Dungeons & Dragons Role-playing Game, the game that defines the genre and has set the standard for fantasy role-playing for more than 30 years.

D&D is an imaginative, social experience that engages players in a rich fantasy world filled with larger-than-life heroes, deadly monsters, and diverse settings. As a hobby game, D&D is an ongoing activity to which players might devote hours of their time -- much like a weekly poker game -- getting together with friends on a regular basis for weeks, months, or even years.

Players create heroic fantasy characters -- mighty warriors, stealthy rogues, or powerful wizards -- which they guide through an ongoing series of adventures, working together to defeat monsters and other challenges and growing in power, glory, and achievement. The game offers endless possibilities and a multitude of

choices . . . more choices than even the most sophisticated computer game, because you can do whatever you can imagine!" [15] .

Before personal computers were readily available in people's homes, the console industry (i.e., Nintendo and Sega) capitalized on RPGs and introduced graphics in addition to text and allowed the user to equip their characters with various items. As early console games had less than 1 megabyte of memory, repetitious use of background scenery called "tiles" were used to minimize the amount of graphics and memory. With the evolution of these early console (role-playing) games, three major constructs emerged. First, each character could be ***equipped*** with specific weapons and armor. Second, each character had a different ***class*** such as being a fighter, healer, or magician. Equipment items that were available to characters were limited by class. Third, a ***leveling*** system was developed that allowed the game character to become more powerful based on experience and success. For example, if a character survived and managed to defeat a certain number of enemies, the character would reach a new level and acquire new skills. Usually role-playing games were and are driven by a dramatic story line to maintain player interest throughout the game. Essentially, these games are interactive story plots in which the player can develop and improve the character's skill level, thus influencing the game outcome.

With the availability of the personal computer, companies began creating PC versions of role-playing games. The rule sets for these early computer games mimicked the table-top versions. However, the player now had the ability to ***customize*** characters before starting the game by defining their attributes and appearance, thus taking the game a step farther than the console versions. Good examples of this new style RPG are the **Baldur's Gate** [18] series, which debuted in 1998, with an earlier version **Might and Magic** [19] **,** which debuted first on Apple in 1986 and finally made it to PC in 1987. In both of these games the player would create either one character or a small group of characters and define their abilities, appearances and equipment, allowing a full range of customization. These characters venture through the story, gain levels, and eventually become powerful enough or of a high enough level to be able to win the game. Although

this customization became very important to the role-playing community, these games were still designed for a single-player.

Some of these early games allowed for local area network (LAN) connections, but the game play was still single-player. This single-player focus changed with the introduction of **Ultima Online** [1] released in 1997. This was the first game with a "*persistent world"* in which a user could log on and off while maintaining progress in the game and allow other players to keep playing. The game play was very similar to other fantasy based RPG's, except that now players could battle each other as well as computer controlled monsters. **Ultima Online** is the first example of a massive multiplayer online game, such as those that dominate the role-playing game industry today. **Everquest** came out soon after **Ultima Online** in 1999 and gained widespread popularity due to the fact that it added a 3D environment. Today, **World of Warcraft**, which debuted in 2004, has by far surpassed the popularity of its predecessors and it is estimated that 6.5 million players subscribed to this game[13] . Ironically, the settings are still the same for all three of these games, which is in a fantasy realm, where knights and monsters battle each other. The player can choose to be a wizard, or warrior or some other fantasy occupation. However, each massively multiplayer game has used previous massively multiplayer game to evolve, becoming more immersing and addicting to the player. In fact, many critics complain it is becoming too addicting. These Massive Multiplayer Online Role-Playing Games (MMORPG) are the most advanced and innovative form of the role-playing genre and provide social interaction that was only available in the early table-top versions.

1.3 Background of Real Time Strategy Games (RTS)

The history of the real time strategy game genre is not as complex as that of the RPGs. Some early text-based strategy games allowed the creation and deployment of armies, which would be displayed by numbers in a grid on the screen, providing a game atmosphere similar to the board game **Risk**. The game of **Risk** is defined as the following:

"**Risk** is a game of world domination, where the object is to conquer the world. To win, you must attack and defend – attacking to acquire territory, and defending to keep it from your opponents.

The game board features a map of six continents divided into 42 territories. It's a game of strategy as you battle to win by launching daring attacks, defending your territory, and moving across continents with your cunning plan! Play three variations: World Domination, Capital Risk, and Secret Mission Risk. This game will engage and challenge any player to join the ranks of world leaders!

The standard Risk board game includes dice, Risk cards, and six sets of miniature armies. The newer editions include new scenarios and objectives, and are stand-alone games (not expansions).

The software version features cutting-edge artificial intelligence and stunning 3-D graphics, as well as excellent multiplayer options." [16]

These early tactic games were like the game of chess (based on turn taking) and were not RTS games in the modern sense. These early text-based games were supplanted by console games where a graphical environment was introduced, providing an easier interface for character control. The consoles once again used tiles to create a background and allowed the users to move each character and possibly attack with it before their turn ended. These early console strategy games were based on turns and players took turns moving, attacking, or retreating, depending on their individual character's strength.

Military-based strategy games, where victory depended on merely making a more powerful character, employed strategies and limited user options. Game designers began incorporating "*rock, paper, and scissor*" rules of encounter to increase user involvement. For example, one character could be effective against two different types of characters but weak against a third. Therefore, the player should try to find the best combination of characters to handle the widest variety of tactics that the other players may use. This concept still permeates real time strategy games to this day. Although turn-based games were entertaining, they ultimately reduced to a complicated version of chess.

Herzog Zwei [9] , which was released in 1989, changed the way strategy games were played. This strategy game, which was designed for Sega, did not employ turn taking. When units came within a certain range of an enemy, an attack was automatically initiated and units did not wait for user input. This game only allowed the user to give a certain set of basic commands to each of the characters. These commands were more like

goals for the characters. Once the player gave the command, the character would then try to perform it as best as it could without any involvement from the player. This game also employed an interesting mechanism for selecting characters. The player's only controllable character was a jet airplane that could transform into a robot. If the Jet was over a friendly character it could pick it up. This fact allowed the player to move the character and give it a new goal or commands to perform. Such novelty in character selection has not been surpassed since. RTS games have not met with very much success in the console world most likely due to the difficulty of selecting input with a console controller and not a mouse. The PC realm quickly saw the advantage of units not waiting for turns. Thus, games like **Command and Conquer** [20] and **Warcraft I** [21] had a much faster game play and the player had less time to ponder the next move. More emphasis was placed on battle preparations where players accumulate resources, construct proper buildings, acquire wealth, upgrade characters and finally build armies for the upcoming battle. The player now had the capabilities to choose priorities for battle preparation.

From games such as **Warcraft I**, four basic themes evolved that are infused into RTS games today. Players must:

- **Build a base** - base complexity varies from game to game
- **Manage resources -** this includes acquiring and managing resources
- **Decide the types of characters to create** - appropriate character combinations influence battle outcomes
- **Choose the appropriate character upgrades -** character capabilities also influence battle outcomes

Although RTS games use rules and strategies to produce armies, the final battle is mostly controlled by the game and not the user. Although the player decides what units to send into battle, it is almost impossible for a human player to control all units simultaneously once the fighting begins. This battle often degenerates into a brawl between massive numbers of troops where attrition influences the outcome more than the strategy. RTS games today have not evolved much past this character preparation and giant end-battle scenario.

1.4 Previous Work

Several past attempts were made to improve RTS games by incorporating RPG game play elements. The first attempts were turn-taking tactic games that were made popular by the console game industry, but PC versions never achieved wide popularity. These games allowed the player to select specific characters for a mission and move them to locations of choice. These characters fulfilled specific combat roles during battles. In these games the user and computer alternate taking turns, and once armies are in place the battle begins. An example of this type of game is **Final Fantasy Tactics,** produced for the Game Boy Advance. In this game, the characters were stored and made available for future use if the player so wished. These characters could change jobs, learn skills, level up (increase in ability) and be equipped with various weapons and armor depending on their current job. Once the battles started, each character could move and attack once per turn, therefore the user had complete control of each character and the computer did not automatically force the player's character to attack. The more victories a character won, the more powerful it became. With these properties, the game did in fact successfully incorporate the aforementioned RPG elements into a strategic game setting.

Although **Final Fantasy Tactics** incorporated these elements, it was still not a true RTS game. This game allowed the player to customize and deploy characters for specific missions, but no bases or buildings were constructed. Characters were randomly added to the army over time, with the player having no direct control of when that would occur. As previously mentioned, turn based games such as **Final Fantasy Tactics** allowed for thought and careful planning, but lacked the fast paced environment found in current RTS games. These games also lacked the ability for the user to manage and select resources, which were merely gained by completing missions. Therefore, the players' economical involvement in the game was limited to the purchase of equipment for their characters with no ability to build bases, upgrade, or acquire more characters.

Another attempt to combine RPG constructs with RTS games was seen in **Warcraft II**. A Hero character was added who needed to remain alive in order for the player to progress to the next level. This hero had more capabilities (more strength and health) than other non-special counterparts, but the player could still not directly develop

or equip this hero character. This same concept was applied in **Starcraft** as well. In this game, the hero would have more health and greater attack power but would have to remain alive on certain missions if the player wanted to advance in the single player game.

Relic's **Dawn of War** provided a different approach that allowed the hero to increase other character capabilities such as attack power, defense and morale, which could be the defining factor between victory and defeat. However, there still was no procedure for allowing characters to become stronger from experience (leveling up) nor was there any focus on equipment or individual character customization.

The concept of combining role-playing game elements to RTS games by allowing characters to develop and gain power has not been satisfactorily accomplished. Some possible reasons for the lack of incentive to take this innovative step could be the fact that:

1. Fans of RTS games have become accustomed to massive numbers of characters with limited control. Game skills required are similar among various RTS games, allowing the player to easily become accomplished in new games as they are produced.
2. During the 1990's, some extremely complex RTS games were produced. Much of the complexity detracted from their entertainment value. A good example of this is **Star Wars Rebellion** in which there were over 100 different subsystems the player needed to control [17] . Therefore, game developers worried about over-complex game plays, continued producing the games based on standard "massive horde" RTS mechanics.
3. The standard RTS game maintained a delicate balance between characters, in order to allow equal chance for victory. Incorporating aspects from the RPG games complicates this balance due to the fact the characters must be balanced for all levels they could potentially reach. This requires greater game development time.

Regardless of these complications, the survival of the RTS game industry is dependent on fresh ideas. To date, much effort in RTS game development has been to

apply the same mechanics, or processing that is done behind the scenes, to programs with better graphics and more exotic settings. For example, Relic's **Dawn of War** and **Company of Heroes** have similar mechanics and game play, but one is futuristic while the other takes place in a World War II setting. Relic did not merely reuse their code in these two games. The work that was put into the **Company of Heroes** graphic system was quite impressive and much more complex then that for **Dawn of War**. The user could assign squadrons to various buildings and, once inside, the characters would shoot enemies from the windowsill. During game play the user commands a wide variety of camera controls, one of which was the zoom feature. Upon zooming in on the character firing from the window the player can see the facial expressions of that character. This amount of detail proves that the graphics are evolving as technology becomes better and game companies are pushing the envelope for better games.

However, better graphics and visual effects can only improve the game up to a point. Regardless of this impressive graphical feat, the game play, or rules of play, are almost the same for both of these games. The following chapters will expand on how it is possible to change the very mechanics of the game and add an entirely new dimension to real-time-strategy games thus providing a new dimension to enhance this genre.

1.5 Present Work

Warrcraft III was one of the first RTS games with character development. This game allowed a player to have up to three special units (heroes) who could carry items and increase in strength as the game progressed. Each hero unit would gain a new ability or upgrade an existing ability as they increased in levels. In fact, the mechanics of heroes in this game met all five rules proposed by this thesis. In **Warcraft III** the player could only develop three hero units in each game, but in **Fuzz's Revenge** every unit may be customized and developed. However, the ability to develop three characters in **Warcraft III** was a marked improvement over **Warcraft II.** This changed the game play dramatically and made for a much more intriguing RTS game**.** **Warcraft II** was limited to building large numbers of units and engaging in massive battles to wear down the enemy and consume his resources with few strategic options, outside of building massive amounts of the most powerful unit to crush your enemy by force. In addition to

developing heroes, **Warcraft III** provided incentive for players to maintain smaller armies by introducing "upkeep" as a form of a tax which would be increased if armies became too large. Blizzard, the company who produced the **Warcraft** series, also balanced **Warcraft III** by making a level ten unit more valuable than ten or fifteen normal units. In this way, new strategies involving unit combinations were introduced that could affect game outcome.

By creating a game with a maximum of three heroes for customization, Blizzard did not fully develop this idea to full potential. For example, Footmen (the grunt unit for humans) surviving multiple battles could not gain more power. Regardless of this, just allowing the development of three hero units in **Warcraft III** increased strategic options and added more entertainment value to the game.

1.6 Overview of the Thesis

Chapter 2 introduces the test bed software, called Game Maker, used to make our pilot game and defines the method and approach for this project. Chapter 3 lists the initial and final requirements of the prototype, use cases, and primary scenarios. Chapter 4 expands on the high level design and modules of this project. The final pilot game results are discussed in Chapter 5. Chapter 6 details similar and future work regarding the concept of combining RPG elements to RTS genres. Chapter 7 concludes the thesis with an overview of the work done in this project.

2 Chapter 2: Method

This chapter will present the idea behind this project and introduce the rule system behind the pilot games development. This chapter will also introduce the software tool called Game Maker and will then detail various components of the pilot game, “Fuzz’s Revenge”.

As stated in the previous chapter, the purpose of this project is to combine the four components of a RPG into the RTS game play, in order to allow character customization. The four RPG constructs are the following:

- Each character can be ***equipped*** with specific weapons and armor.
- Each character can have the ability to become more powerful, based on experience and success which is called ***leveling*** up.
- Each character can be ***customized*** by defining its attributes and, if graphics allow, its appearance.
- Each character can have the ability to upgrade its ***class*** status when certain levels have been attained.

The approach to this project was quite linear in design. An undesirable aspect of a common game genre (RTS) was noted. Regardless of what battles a unit survived it could not grow based on experience. Therefore, to fix this problem a hypothesis of combining RPG aspects to an RTS game was proposed. After a prototype game was developed the hypothesis was modified to improve upon the undesirable problems in the original proposal. The final design was then documented using uniform modeling language (UML) and the prototype or pilot game was modified to reflect these changes.

Initially, this project had a hypothesis that if seven rules were followed then the two game genres, RPG and RTS, could be combined. This would be accomplished by adding the four components of an RPG to be combined with the RTS game. The next section will look at each rule with detail and then give explanation to why two of the rules were considered unnecessary. From this, the five final rules have been defined to successfully allow character development in an RTS setting.

2.1 The Initial Seven Rules

To preserve the unit control that is characteristic of RTS games and to provide a sense of character development found in role playing games, the prototype game, **Fuzz's Revenge**, was based on character individuality and not on large numbers of undifferentiated identical characters. The following seven rules were applied in the prototype game.

1) Every unit should have some degree of customization.
2) Every unit should have the ability to increase in level or rank.
3) Unit capability should be limited by its skills and rank.
4) Game mechanics should encourage the use of minimal numbers of units.
5) Every unit should have an equipment and statistics panel.
6) The game should have specific goals and objectives for units or groups of units to accomplish and not merely the sole goal of dispatching the enemy.
7) Increasing the character's strength should be much more valuable then building a more expensive unit.

Rule 1 - Every unit should have some degree of customization.

Every unit needs customization capabilities. At a minimum, skills and equipment should be customizable. The prototype (pilot game) for this project allows the player to name the unit, change the unit's equipment, and select the unit's capability when leveling up. With more advanced graphics, the player could also have the ability to change the outward character appearance. The potential for innovation in this area is quite large and could enhance the game play substantially.

Rule 2 - Every unit should have the ability to increase in level or rank.

Each player's unit should develop as the game progresses, based on success and experience. Without this capability, the game would merely be a basic real time strategy game (RTS). Not only should each unit have an XP (experience point) counter variable that allows the characters to increase in level, the game play should facilitate this aspect of strengthening the unit. For example, due to the high death rate experienced by lower

level characters in an RTS game, the units should be allowed to rest after battle to regain strength. Without this capability most units would not survive to grow and would perish after a few battles.

Rule 3 - Unit capability should be limited by its skill or level.

RPG's have evolved two styles of character advancement. One is the skill advancement method with which after the character has successfully accomplished something for a certain amount of time its ability in that task, or skill, will improve. The other method is leveling, where after a character has successfully beaten a certain number of enemies it will advance to a new rank or level. When this happens, all of the unit's attributes increase. Because the pilot game used the level system this paper will refer to this system instead of the skill advancement system.

Unit performance should be limited by level because the higher the level is, the greater the unit's potential performance is. For example, a basic space marine private should not be allowed to command a battle cruiser. Characters must evolve to higher levels in order to use more advanced equipment, and further, certain skill levels must be reached to complete the mission. As previously mentioned, this places the game focus on individual unit development and not on massive armies collectively.

Rule 4 - Game mechanics should encourage the use of minimal numbers of units.

The game mechanic, such as resource management, base management and construction, upgrades, and unit production, should encourage a limited number of units, thus facilitating the use of fewer units versus large armies of anonymous soldiers. In other words, the player may utilize four to eight units in battle during several minutes of game play, where as in a basic RTS game an army could be built during this time period, where the player would not pay as much attention to single units. Just because this hybrid game is built on individual customization, it does not mean it needs to be overly complex. For example, in the mid 1990's a group of games were produced that were so complex it required an economist and a military advisor to play them (e.g., **Star Wars**

Rebellion). The game design for this hybrid game may be complex, but the game play, from the user's perspective, should be relatively simple and easy to interact with.

Rule 5 - Every unit should have an equipment and statistics panel.

For the sake of utility, every unit should have a status window. When this window is opened, it should display important statistics, current equipment, acquired attributes, and optional upgrades.

Rule 6 - The game should have specific goals and objectives for the player to accomplish, and not merely the goal to dispatch the enemy.

A vital part to an RPG is a strong story line that will maintain a player's interest, in contrast to games centered on dispatching the enemy, which soon lose their luster. **Starcraft** and **Warcraft III** are good examples of the latter. The player would use built-in tools, such as a map editor, to add diversity to the game. Without this ability to customize the game mechanics, players might lose interest once the game or the main campaign with a story, was conquered. In the case of **Starcraft** and **Warcraft III** the game makers also depended on renewed interest due to the ability to play with or against friends over a network.

Rule 7 - Increasing a character's strength should be much more valuable then building a more expensive type of unit.

Increasing a unit's level adds a new dimension to the game. The longer surviving units become more powerful and exert more influence on the game outcome. Preserving units and proper utilization of specific units becomes a major game strategy. Players can still build an army, but they will be an army of individuals and not "cookie cutter" units. This rule captures the essence of the major thrust of this project; to incorporate unit development into an RTS game.

2.2 The Revised Set of Rules

Once work began on the initial prototype, it became apparent that rule number four was unnecessary. This rule was initially included to ensure that the game play would allow

small groups of units to gain experience and the opportunity to level up. However, the pilot game seemed to lose the RTS feel because only a small number of units could be used at one time. This seemed more akin to an early RPG game where small groups of units would try to complete a goal. Therefore, it was concluded that small numbers were not essential for combining RPG and RTS game strategies as larger troop numbers could be used while allowing individuality among units.

Rule number six requires a strong story line in order to incorporate all aspects of a RPG into an RTS game. However, the essence of this project is to add unit development to a RTS game, and by enforcing rule six the prototype exhibited more RPG qualities than RTS. RTS games rely on massive battles and large numbers of units to provide their entertainment value and not on story lines. Although RTS games have a story, the addition of a strong story line and alternate focus other then conquest, detracted from the RTS feel of conflict [12] .

Likewise, it was originally thought in rule seven that leveling up was the most important part of this hybrid game development. However, if game success weighs too heavily on leveling individual units, the RTS feel is also lost. For this reason, rule seven was revised. Although unit strength should be more important than building expensive units, building large armies should also exert a strong influence on game outcome.

In summary, the first seven rules were designed to place the game focus on individual units, but when this was carried too far, the RTS game feel was lost. The following list details the final five rules utilized for this project.

1) Every unit should have some degree of customization.
2) Every unit should have the ability to increase in level or rank.
3) Unit capability should be limited by skills and rank.
4) Every unit should have an equipment and statistics panel.
5) Increasing the strength of a character should be more valuable then building a more expensive unit. However, losing their most powerful character should not end the player's chances of winning.

2.3 Prototype Game Construction using Game Maker

Game Maker is a high level program that allows the user to create almost any game desired, while keeping difficult issues such as graphics input and sound abstracted enough to allow easier use. Therefore, the programmer does not have to worry about coding drivers and graphics but merely calls the appropriate function. Game Maker was used to construct the prototype, allowing for easy creation of graphics, sounds, and input thus more time was available for game play development. Game Maker is a reliable test bench for analyzing game design and facilitates the easy use of graphics.

Game maker utilizes object oriented programming for object oriented algorithms. These objects are created before the game is compiled into an executable. The object can then be modified and can either be i) an invisible objects, ii) purely code, or iii) an actual entities in the game. These objects are recipes to create various entities, and once an object is created in Game Maker it is easy to reproduce objects in as many instances as desired. Therefore, only one adjustment is needed to change all instances of that object in the game before the game is compiled.

Various sub-modules, such as mouse, keyboard, and collision, could be defined for each object. These sub modules determined how objects react when colliding with one another. These sub modules also handled keyboard and mouse inputs and with various options the user could define mouse functions (for example, left click, left press, left released, left drag and the equivalent for the other buttons). These sub modules acted as events that can trigger the object to run a piece of code. Special care must be taken that only the desired instances execute the piece of code and not all instances of that object type. This was a large problem with unit movement and the graphical user interface.

2.4 Game Maker Limitations

Game Maker is a great test bench, however because of its ease of use it has functional limitations. Game Maker was designed for non-programmers, therefore functionality was so simplified that it had a lot of overhead that caused limitations. Some of these limitations noted during this project were:

1. A large degree of overhead, or more processes running in the background, was included with the program (especially timer functionality). C or C++ code would have been a lot more efficient then the Game Maker script.
2. Game Maker it is not designed for three-dimensional graphics but produces two-dimensional graphics with sprites very well.
3. A full version of Game Maker is required for network gaming.
4. Each object required more processing time than C++, and the more objects that were used such as walls and characters, the slower the system would run. It took careful programming in order not to over use the timer interrupt functionality.

2.5 Method used and an Overview of the Pilot Game "Fuzz's Revenge"

The pilot game developed in this thesis is called "Fuzz's Revenge". This game allows the player to build a base and there is a hierarchy of buildings that must be built in the correct order. This simulates many of the RTS games on the market today. For simplicity, the player can create one type of unit, a blue fur ball called "Fuzz". These units must search for three keys and in the process fight various monsters, which as they defeat, the fuzz units gain experience points. As the game proceeds and the "Fuzz" units journey farther from the starting point, they encounter different monsters that progressively get harder to defeat. As the monsters become more difficult to defeat, more experience points are awarded to the victorious units. From 3 to 40 experience points are awarded to the units, depending upon which monster they defeat. All units that have the monster within range gain experience points when the monster is dispatched.

A combination of experience points, a variable called "delay constant", and the experience points for next level determines when a "Fuzz" unit is eligible level up. As noted in Equation 2.1, as the delay constant increases more points are needed to level up. This constant could be reduced when testing the game, thus speeding up the leveling process. The following equation is used to determine this eligibility:

$$ExperienceNeededForNextLevel = PreviousExpereienceNeeded + DelayFactor*NextLevel \quad (2.1)$$

Initially, each unit begins at the first level (level 0), the delay constant is 10, and the next level is 1. Therefore, from Equation 2.1 ten points are needed to proceed from level 0 to level 1 (Points needed = 0 + 10 * 1 = 10). As shown in Table 2.1 the number of points required for leveling up increase at a exponential rate with each level. Technically there are a maximum of 99 levels that could be obtained in this game, but in practicality this level cap will most likely never be reached because the player would have to spend a day or so to gain the necessary experience points.

Table 2-1: Calculating Level Thresholds

Level	Minimum Experience Points For Required for Current Level	Delay Factor	Next Level	Point for Next Level
0	0	10	1	10
1	10	10	2	30
2	30	10	3	60
3	60	10	4	100

Every time a unit reaches a new level it gains one skill point. With each skill point the player can open a statistic panel and upgrade one of six attributes: rate of fire, regeneration rate, attack, defense, speed, and range. This process allows for the customization of each unit as the game proceeds and fulfills rule number four. As the unit level increases, class options also increase. For example, at level five the unit may be upgraded from "newb" to soldier. This ability to increase the skill and choose the class of each unit, produces individuality among characters and allows the player to develop them as desired, thus fulfilling rule one.

The game requires the player to go into three dungeons and find three keys. As the units progress farther and farther away from the starting location, the enemies become stronger. In fact, a newly created unit could not survive in the final area. Therefore, it is necessary for the "Fuzz" units to gain levels in order to conquer the game. Thus, rule three is fulfilled, requiring that each unit's capability be limited by rank. Because of this,

unit development is more important than building massive armies and rule five is fulfilled.

As explained above, this project underwent two phases. Initially, seven rules were used to design the first prototype game. Later it became apparent that only five of these rules were necessary, thus the second iteration of the prototype ensued. Along with these changes, group movement was added to maintain the RTS game play. After this was achieved, the game design began. Chapter 3 will give the details of the requirement and high level methods used in **Fuzz's Revenge**.

3 Chapter 3: Software Requirements for “Fuzz’s Revenge”

This chapter details the initial requirements of the game prototype, as well as various modifications implemented during the course of the project. This chapter also presents a thorough view of the use cases and primary scenarios of the pilot game “Fuzz’s Revenge”.

3.1 The Initial Requirements

There are several important reasons for creating game requirements for integrating RPG components into the RTS game structure. Little success has currently been demonstrated in incorporating software engineering into game design. By designing a simple prototype to test a theory before designing a formal design the freedom to make modifications and analyze the results was now possible without rewriting the entire design. It was then possible to document the game design with Unified Modeling Language (UML)[8] . The UML documentation was paramount to ensure that the five final rules were incorporated in the final iteration of the game code.

Table 3-1 details the various functional requirements of "Fuzz's Revenge". Requirements are broken down into three different categories which are:

- R#[1] Required components that **MUST** be in the game.
- R#[2] Components that should be incorporated into a RTS game but are not vital to this project.
- R#[3] Components that would enhance the game, but require a more advanced platform and time to develop. If this were a commercial project these components would be necessary.

Table 3-1: Initial Requirements for the Pilot Game "Fuzz's Revenge"

Requirement	Descriptions
R01 [1]	The SYSTEM shall allow for mouse input to select and construct buildings.
R02 [1]	The SYSTEM shall allow the base camp building to be built upon start.
R03 [1]	The SYSTEM shall select a fuzz unit when the mouse is clicked on it.
R04 [1]	The SYSTEM shall have an interface window at the bottom, which contains available buildings that may be constructed.
R05 [1]	The SYSTEM shall have an interface window at the bottom, which displays the food and gold the player has acquired.
R06 [1]	The SYSTEM shall have an interface window at the bottom, which displays the option of the selected building.
R07 [1]	The SYSTEM shall allow the user to construct a unit if the gold and food requirements are met and the home base is selected.
R08 [1]	The SYSTEM shall allow the user to construct an assault rifle if the gold requirement is met and the fair ground is selected.
R09 [1]	The SYSTEM shall allow the user to create a holy stave if the gold requirement is met and the bio dome is selected.
R10 [1]	The SYSTEM shall allow the user to create a bazooka if the gold requirement is met and the factory is selected.
R11 [1]	The SYSTEM shall allow the user to create a magic hat if the gold requirement is met and the magic shop is selected.
R12 [1]	The SYSTEM shall allow the user to create a fire wand if the gold requirement is met and the magic shop is selected.
R13 [1]	The SYSTEM shall allow the user to create an ice staff if the gold requirement is met and the magic shop is selected.
R14 [1]	The SYSTEM shall require a base camp in order to build a fair ground.
R15 [1]	The SYSTEM shall require a base camp in order to build a bio-dome.
R16 [1]	The SYSTEM shall require a bio-dome in order to build a gold mine.
R17 [1]	The SYSTEM shall require a gold mine in order to build a factory.
R18 [1]	The SYSTEM shall require a fair grounds and a factory in order to build a magic shop.
R19 [1]	The SYSTEM shall, upon start up, define an array of values that shall represent the experience needed for the appropriate levels.
R20 [1]	The SYSTEM shall, if a fuzz unit is selected, open a sub window when the space bar is hit.
R21 [1]	The SYSTEM shall display the Health Points of the unit if the sub-window is open.
R22 [1]	The SYSTEM shall display the total Health Points of the unit if the sub-window is open.
R23 [1]	The SYSTEM shall display the experience points of the unit if the sub-window is open.
R24 [1]	The SYSTEM shall display the next level requirement of experience points if the sub window is open.
R25 [1]	The SYSTEM shall display the attack rating of the unit if the sub window is open.
R26 [1]	The SYSTEM shall display the defense rating of the unit if the sub window is open.
R27 [1]	The SYSTEM shall display the class (or job) of the unit if the sub window is open.
R28 [1]	The SYSTEM shall display the numeric level of the unit if the sub window is open.
R29 [1]	The SYSTEM shall display the speed of the unit if the sub window is open.
R30 [1]	The SYSTEM shall display the weapon the unit is using if the sub window is open.
R31 [1]	The SYSTEM shall display the regeneration rate of the unit if the sub window is open.

Requirement	Descriptions
R32 [1]	The SYSTEM shall display the rate of fire of the unit if the sub window is open.
R33 [1]	The SYSTEM shall display the name of the unit if the sub window is open.
R34 [1]	The SYSTEM shall display the range the unit can fire at if the sub window is open.
R35 [1]	The SYSTEM shall display the icon (which changes with class upgrades) of the unit if the sub window is open.
R36 [1]	The SYSTEM shall display the speed upgrade icon when the sub window is open, where the skill points of the unit are greater than 0 and if the units speed is less then 2.
R37 [1]	The SYSTEM shall display a rate of fire upgrade icon when the sub window is open and if the selected unit's skill points are greater than 0 and if the rate of fire is greater than or equal to 30.
R38 [1]	The SYSTEM shall display the defense upgrade icon when the sub window is open and the selected unit's skill points are greater than 0.
R39 [1]	The SYSTEM shall display the range upgrade icon when the sub window is open and the selected unit's skill points are greater than 0.
R40 [1]	The SYSTEM shall display the attack upgrade icon when the sub window is open, if the selected unit's skill points are greater than 0.
R41 [1]	The SYSTEM shall display the regeneration upgrade when the sub window is open, if the selected unit's skill points are greater than 0.
R42 [1]	The SYSTEM shall add one to the regeneration rate of the unit when the regenerations upgrade icon is selected and shall subtract 1 from the unit's skill points.
R43 [1]	The SYSTEM shall add 2+round(units levels/2) to the attack power of the unit when the attack upgrade icon is pushed and shall subtract 1 from the unit's skill points.
R44 [1]	The SYSTEM shall add 1+round(units level/2) to the defense rate of the unit when the defense upgrade icon is selected and shall subtract 1 from the unit's skill points.
R45 [1]	The SYSTEM shall subtract 10 from the rate of fire of the unit when the rate of fire upgrade is selected and shall subtract 1 from the unit's skill points.
R46 [1]	The SYSTEM shall add 1 to the speed of the unit when the speed upgrade icon is pressed and shall subtract 1 from the unit's skill points.
R47 [1]	The SYSTEM shall add 10 to the range of the unit when the range upgrade icon is pressed and it shall subtract 1 from the unit's skill points.
R48 [1]	The SYSTEM shall display upgrade to soldier icon when the unit's level is greater than or equal to five, the current class of the unit is "newb", the skill points are greater than 0, and the sub window is open.
R49 [1]	The SYSTEM shall display upgrade to wizard icon when the unit's level is greater than or equal to ten, the current class of the unit is "newb", the skill points are greater than 0 and the sub window is open.
R50 [1]	The SYSTEM shall display upgrade to priest icon when the unit's level is greater than or equal to fifteen, the current class of the unit is "newb", the skill points are greater than 0, and the sub window is open.
R51 [1]	The SYSTEM shall display upgrade to fire sorcerer icon when the unit's level is greater than or equal to fifteen, the current class of the unit is "wizard", the skill points are greater than 0, and the sub window is open.
R52 [1]	The SYSTEM shall display upgrade to ice mage icon when the unit's level is greater than or equal to twenty, the current class of the unit is "wizard", the skill points are greater than 0 and the sub window is open.
R53 [1]	The SYSTEM shall, if the soldier upgrade icon is hit, change the unit's class to soldier, changing the unit's appearance and subtracting 1 from the unit's skill points.
R54 [1]	The SYSTEM shall, if the wizard upgrade icon is hit, change the unit's class to wizard, changing the unit's appearance and subtracting 1 from the unit's skill points.

Requirement	Descriptions
R55 [1]	The SYSTEM shall, if the priest upgrade icon is hit, change the unit's class to priest, changing the unit's appearance and subtracting 1 from the unit's skill points.
R56 [1]	The SYSTEM shall, if the fire sorcerer upgrade icon is hit, change the unit's class to fire sorcerer, changing the unit's appearance and subtracting 1 from the unit's skill points.
R57 [1]	The SYSTEM shall, if the ice mage upgrade icon is hit, change the unit's class to ice mage, changing the unit's appearance and subtracting 1 from the unit's skill points.
R58 [1]	The SYSTEM shall open a dungeon window on the bottom left of the screen if any units have come in contact with a dungeon on the world map.
R59 [1]	The SYSTEM shall close the dungeon window when all units have either left the dungeon or died.
R60 [1]	The SYSTEM shall add 1 gold to the player after every 100 cycles per fair grounds that are built.
R61 [1]	The SYSTEM shall add 3 food to the max food of the player per bio dome build.
R62 [1]	The SYSTEM shall not allow a fuzz unit to be constructed unless the food used is greater than or equal to total food.
R63 [1]	The SYSTEM shall subtract 1 from "food used" when a fuzz unit dies.
R64 [1]	The SYSTEM shall add 1 to "food used" when a fuzz unit is created.
R65 [1]	The SYSTEM shall allow the fuzz unit to fire on an enemy when it is inside a certain range.
R66 [1]	The SYSTEM shall create the appropriate attack based on what weapon the fuzz is using.
R67 [1]	The SYSTEM shall allow enemies to fire on the fuzz units once they are inside the enemies range.
R68 [1]	The SYSTEM shall reward experience points for any monster who dies within that unit's range value.
R69 [1]	The SYSTEM shall calculate damage, whether to a monster or fuzz unit, if and only if the attack actually collides with the fuzz unit.
R70 [1]	The SYSTEM shall, upon initialization, create all monsters on the map.
R71 [1]	The SYSTEM shall make the fuzz unit start walking toward a point where the right mouse button was clicked, if the fuzz is selected.
R72 [1]	The fuzz units will stop should it hit a blocking terrain, such as trees, cactus, walls, etc.
R73 [1]	The player shall gain 3 gold per gold mine every 100 ticks.
R74 [1]	The SYSTEM shall teleport the fuzz unit to the dungeon area if it should collide with a dungeon icon.
R75 [1]	The SYSTEM shall teleport the fuzz outside the dungeon if it should collide with the exit door icon inside the dungeon.
R76 [1]	The SYSTEM shall, if the status window is open, close this window if the space bar is pressed.
R77 [1]	The SYSTEM shall allow the soldier to increase speed if it senses an enemy nearby. This shall cost magic points.
R79 [1]	The SYSTEM shall allow the wizard to cast a protection spell if it senses an enemy nearby. This shall cost magic points.
R80 [1]	The SYSTEM shall allow the priest to heal any friendly units in its vicinity if it detects they are low on health. This shall cost magic points.
R81 [1]	The SYSTEM shall allow the fire sorcerer to summon a fire pet (walking flame) if it detects an enemy within its range. This shall cost magic points.
R82 [1]	The SYSTEM shall allow an ice mage to cast ice attacks in 8 directions if it detects an enemy presence. This shall cost magic points.

Requirement	Descriptions
R83 [1]	The SYSTEM shall give a predefined amount of gold to the player for each monster that is killed.
R84 [1]	The SYSTEM shall not allow the "newb" class to receive any special bonus skills. Therefore, "newb" units can only attack and move.
R85 [1]	The SYSTEM shall allow the equip menu to pop up if the middle mouse button is clicked on the unit.
R86 [1]	The SYSTEM shall allow the equip menu to come up for whichever unit is selected if the left mouse button is pressed on the "E-quip" icon in the bottom right screen.
R87 [2]	The SYSTEM shall be implemented to work in real time.
R88 [2]*	The SYSTEM shall allow for group movement and with possible multiple unit selection.
R89 [2]	The SYSTEM shall include different types of units, performance based upon cost and style.
R90 [2]	The SYSTEM shall allow for certain weapons to destroy certain terrain (trees, rocks, etc).
R91 [2]	The SYSTEM shall include sound effects for different attacks and actions.
R92 [3]	The SYSTEM shall be implemented in a 3D environment.
R93 [3]	The SYSTEM shall include multiple races (not just fuzz units) in which the player can choose to be a member of.
R94 [3]	The SYSTEM shall allow for multi-player LAN games.
R95 [3]	The SYSTEM shall include background music that changes with certain conditions (peaceful, hostile, victorious, etc).

*It is important to note that R88 [2] was modified, and became a mandatory requirement.

3.2 Use Case

Use case modeling is a technique for capturing functional requirements of a system or systems via use cases(Defined by Google Dictionary). Use cases describe the interactions between the users and the system as seen from the outside of the system. The following use cases (table 3-2) walk through various modules or smaller units of the program. Each use case has a title and a brief description that explains what each use case performs.

Table 3-2: Use Cases Designed to Implement Category One Requirements

Use Case Number	Use Case Title	Use Case Descriptions
UC01	Select Unit	If the left mouse button is clicked while the mouse is on a specific unit, that unit's id becomes the global value selected.
UC02	Move Unit	If a unit is selected it will try to move toward a point that the mouse was pointing at the time the right mouse button was clicked.
UC03	Unit Attack	Regardless of whether the unit is moving or not, it will attack if an enemy gets within its range variable.
UC04	Unit Victory	If an enemy is killed, the player receives gold and any friendly units that the enemy was in range of receive

Use Case Number	Use Case Title	Use Case Descriptions
		experience.
UC05	Unit Defeat	If a unit's health falls to 0 or below, then that unit dies. All gear and skills on that unit are lost. However, the food used is decreased by one.
UC06	Unit Statistics	When a unit is selected and the space bar is pushed, the system opens a window on the bottom right screen, displaying Name, Class, HP, max HP, Attack, Defense, Range, Rate of fire, Regeneration, Speed and an icon displaying its appropriate class. If the unit has leveled, skill points are greater than 0, it will display the six attributes of upgrades. Also if the unit reached appropriate levels it will display the appropriate class upgrades.
UC07	Unit Attribute Upgrade	If an attribute upgrade is selected, the system will upgrade the corresponding attribute with a calculated amount. This will subtract 1 from the skill points of the unit.
UC08	Unit Class Upgrade	The unit's class or job will change to the corresponding icon that was pressed. NOTE: all class upgrades become available at different times or conditions.
UC09	Construct Buildings	On the bottom of the screen a bar will list the available buildings. The buildings that cannot be built are scaffold icons. Once the mouse's left button is pressed on non-scaffold building, the mouse will display a scaffold next to the pointer. The player can then move the mouse to a specific area and left click in order to build the building.
UC10	Unit Enter Dungeon	When a fuzz unit touches a dungeon icon, it is teleported to the entrance of that dungeon and a window is opened in the bottom left of the screen.
UC11	Unit Leave Dungeon	In all dungeons there is an exit door, usually very close to where the fuzz came in. If a fuzz comes in contact with that icon then the fuzz unit will be teleported back to the world map outside the dungeon. If all fuzzes have exited or died inside the dungeon the dungeon window will close.
UC12	World Start	Upon startup a script will be run to initialize values such as level placements, enemies, and other entities. One fuzz unit will be created, which the user must name. The map will become visible. The starting fuzz unit will have one skill point to spend.
UC13	Build Unit	If the gold amount is acceptable and if there is enough food, when the mouse arrow is on the create fuzz icon and the left mouse button is pressed, a fuzz unit will be created. The user will be asked to enter its name and then the unit will appear inside the Base Camp icon (the building that must have been selected in order to build the unit). Each fuzz starts with 1 skill point the player can use.
UC14	Build Weapon	If the player has selected one of these buildings: Bio-Dome, Marketplace, Factory or Magic Ship, the appropriate weapon icon/s will be displayed in the bottom interface window. Left clicking on the weapon icon will add that weapon to the weapon list assuming the gold requirement is met.
UC15	Equip Weapon	If the player clicks the middle mouse button on a selected unit or clicks the "E-quip" icon in the bottom right, a weapon

Use Case Number	Use Case Title	Use Case Descriptions
		list of all the weapons the player has built will pop up. If the player selects a weapon from that list and the selected fuzz unit can use that weapon then that weapon will be given to that specific unit. (NOTE: the starting weapon is pistol. If the fuzz unit has not equipped anything before a pistol, it will replace the weapon in the list. (You cannot build pistols)
UC16	Manage Resources	Every marketplace will add 1 gold every 100 game cycles. Every Gold mine will add 3 gold every 100 game cycles. Every Bio-Dome will add 3 food to the max food used upon construction. This sub-module will display gold and food used/total food on the status bar.
UC17	Manage Game	The system will move bad guys (also check for collision of bad guys against trees, rocks, etc). It will also handle bad guys attacking player units and re-spawning the bad guys at the appropriate time.
UC18	Special Abilities	Depending on its class, the fuzz unit should be able to do some specific ability if it has enough magic points and is threatened by an enemy. The “newb” class does not have any special abilities.

Figure 3-1 presents the relationship between the actors involved in this system and the system’s functionality as described in use cases. The fuzz unit handles various functions on its own therefore the user really only needs to move and upgrade the unit. Resources are managed as a function of time and will accrue automatically. Finally, the system, or the Background Process, manages the game and judges which units are victorious in combat. The System also manages enemy units.

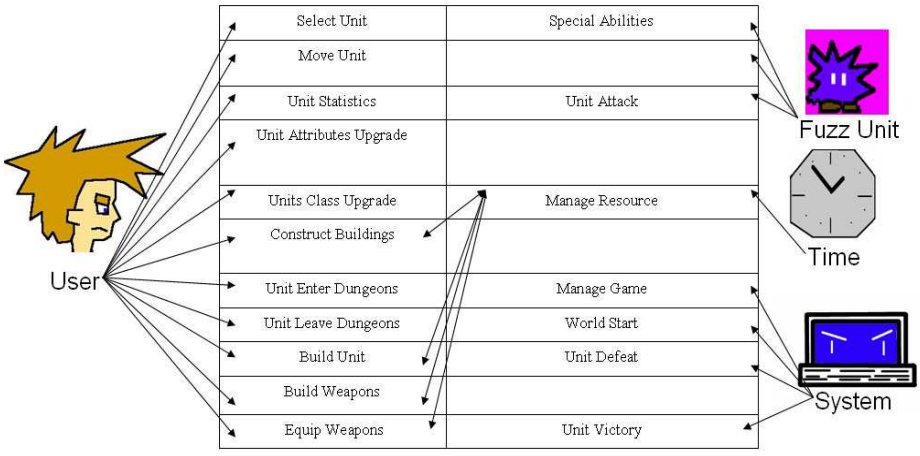

Figure 3-1:Break down of Actors in "Fuzz's Revenge"

3.3 Primary Scenarios

In computing, a scenario is a narrative describing foreseeable interactions between the users (actors) and the system, or between two software components [8] . Due to the large amount of possible scenarios only the most important and prominent scenarios are detailed in the following scenario tables.

3.3.1 Select Unit

A unit or building is highlighted by a white box when the player clicks on it with the mouse's left button. The object's identification number (ID) is stored in the global variable named "selected" (Table 3-3 and 3-4).

Table 3-3: Selecting a Unit

Use Case: Select Unit	
ID:	UC01.1
Actor:	User
Pre-Conditions:	1. World has started. 2. Fuzz unit does exist.
Scenario	1. User moves mouse pointer onto unit. 2. User left clicks mouse button. 3. A box is drawn around the unit.
Post-Conditions:	1. Unit can be moved. 2. Unit statistics can be displayed.

Table 3-4: Selecting a Building

Use Case: Select Unit (Building)	
ID:	UC01.2
Actor:	User
Pre-Conditions:	1. World has started. 2. Building does exist.
Scenario	1. User moves mouse to point at building. 2. User left clicks the mouse button. 3. A box is drawn around the unit.
Post-Conditions:	1. Player can now build any items or characters the building offers.

3.3.2 Move Unit

Once the movable unit is selected and the user right clicks on the location to which the user desires to move, the unit will begin moving. Should the unit encounter any obstacles or other fuzz units, it will stop (Table 3-5 and 3-6).

Table 3-5: Moving a Unit

Use Case: Move Unit (Success)	
ID:	UC02.1
Actor:	User
Pre-Conditions:	1. A fuzz unit is selected.
Scenario	1. The user moves the mouse to the desired location. 2. The user right clicks the mouse. 3. The fuzz unit will attempt to walk toward that point. 4. Once the fuzz unit has arrived within a certain area the unit will stop moving.
Post-Conditions:	1. Unit can be moved again. 2. Unit statistics can be viewed.

Table 3-6: Moving Unit when a Collision Occurs

Use Case Move: Unit (Failure)	
ID:	UC02.2
Actor:	User
Pre-Conditions:	1. A fuzz unit must be selected.
Scenario	1. The user moves the mouse to the desired location. 2. The user right clicks the mouse. 3. The fuzz unit will attempt to walk toward that point. 4. Before the unit reaches the destination it encounters an obstacle. 5. The unit stops moving. 6. The walking animation will continue to specify that the unit has gotten stuck.
Post-Conditions:	1. The unit can move again. 2. The unit statistics can be viewed.

3.3.3 Unit Attack

The fuzz unit will automatically attack when an enemy enters the range designated to that particular unit. The range to the enemy is calculated by a distance formula between the two entities. The more powerful the monster, the greater their attack

ranges. Once initiated, the unit will launch its weapons (each unit may be equipped with different weapons) toward the spot where the monster is. If the attack fails to strike the monster or it strikes a blocking terrain, the attack will be terminated. Some weapons can penetrate interfering objects and still collide with the monster (Table 3-7).

Table 3-7: Attacking an Enemy with a Unit

Use Case: Unit Attack	
ID:	UC03.1
Actor:	Players Unit
Pre-Conditions:	1. Unit exists 2. Enemy comes within range.
Scenario	1. For enemies within range, their id is copied to units target variable. 2. Target variable x and y coordinates are stored. 3. Unit creates an appropriate attack. 4. Appropriate attributes are copied from unit to that attack (attack power, destination, parent unit, etc). 5. Attack will move toward unit at a much faster rate than units move. (It is possible for the attack to miss).
Post-Conditions:	

3.3.4 Unit Victory

Once the enemy's health drops to zero as a result of an attack, it is destroyed. Every unit that had the enemy within range receives a certain number of experience points depending on the monster. This allows for the advancement of multiple units (Table 3-8).

Table 3-8: Defeating the Enemy

Use Case: Unit Victory	
ID:	UC04.1
Actor:	System
Pre-Conditions:	1. Unit fired on enemy. 2. Enemy was hit by unit's attack.
Scenario	1. Enemy takes necessary damage calculated by the system. 2. Enemy's health falls below or equal to 0. 3. Enemy icon is erased from world. 4. All units that had the enemy within their range receive experience bonus. 5. A gold reward is given to the player for killing the enemy. 6. Enemy is destroyed.
Post-Conditions:	1. Unit can now re-acquire a new target.

3.3.5 Unit Defeat

If an enemy hits and destroys a friendly unit, the unit is deleted from the game. With respects of implementation, once the instance of the unit is deleted from the program the array is then consolidated so that all empty spots reside at the end of the array holding the enemies (Table 3-9).

Table 3-9: Friendly Unit Defeated

Use Case: Unit Defeat	
ID:	UC05.1
Actor:	System
Pre-Conditions:	1. Enemy has fired. 2. Enemy's attack hit unit.
Scenario	1. Unit looses necessary hit points. 2. Unit's health points fall below 1. 3. Unit sprite is deleted from world. 4. One is subtracted from food used. 5. Unit is deleted from array.
Post-Conditions:	1. Special weapons equipped to unit are also deleted.

3.3.6 Unit Statistics

The following four scenarios are examples of many unit-statistic combinations available to each unit:

- The statistics, name, and class icon will be displayed in the status window for the unit with no skill (Table 3-10).
- The statistics, name, class icon, and attributes will be displayed in the status window for a unit with skill points, but not eligible to change class (Table 3-11).
- If the unit has not been designated a class by level 15 (it is still a "newb") the statistics window will additionally display three class levels that may be selected if one or more skill points are available, also in the status window. These three options are soldier, wizard, and priest (Table 3-12).
- A unit that has been designated a wizard will have two more secondary classes visible at level 20 that allow the selection of either fire sorcerer or ice mage if the required skill points are available. This shall be displayed in the status window (Table 3-13).

Table 3-10: Statistics Window Display without Skill Points

Use Case: Unit Statistics (No Skill Points)	
ID:	UC06.1
Actor:	User
Pre-Conditions:	1. Unit is selected.
Scenario	1. User pushes space. 2. Sub window pops up. 3. Sub window displays unit name in window. 4. Sub window displays unit class in window. 5. Sub window displays unit icon (representing Class) in window. 6. Sub window displays experience and necessary experience for next level in window. 7. Sub window displays current health out of max health in window. 8. Sub window displays current level in window. 9. Sub window displays attack power in window. 10. Sub window displays defense power in window. 11. Sub window displays speed rating in window. 12. Sub window displays regeneration rating in window. 13. Sub window displays range in window. 14. Sub window displays rate of fire in window. 15. Sub window displays weapon currently used in window.
Post-Conditions:	1. User can close statistics window.

Table 3-11: Statistics Window Display with Skill Points and No class upgrades

Use Case: Unit Statistics (Skill Points)	
ID:	UC06.2
Actor:	User
Pre-Conditions:	1. Unit is selected. 2. Unit has at least one skill point. 3. Unit's speed is only 1. 4. Unit's rate of fire is above or equal to 30.
Scenario	1. User pushes space. 2. Sub window pops up. 3. Sub window displays unit name in window. 4. Sub window displays unit class in window. 5. Sub window displays unit icon (representing Class) in window. 6. Sub window displays experience and necessary experience for next level in window. 7. Sub window displays current health out of max health in window. 8. Sub window displays current level in window. 9. Sub window displays attack power in window. 10. Sub window displays defense power in window. 11. Sub window displays speed rating in window. 12. Sub window displays regeneration rating in window. 13. Sub window displays range in window. 14. Sub window displays rate of fire in window.

	15. Sub window displays weapon currently used in window. 16. Sub window displays upgrade for defense in window. 17. Sub window displays upgrade for range in window. 18. Sub window displays upgrade for speed in window (may not always be available). 19. Sub window displays upgrade for Rate of Fire in window (may not always be available). 20. Sub window displays upgrade for attack in window. 21. Sub window displays upgrade for regeneration in window.
Post-Conditions:	1. User can close statistics window. 2. User can now upgrade the unit.

Table 3-12: Statistics Window Display with Skill Points and First Class Upgrades

Use Case: Unit Statistics (Skill Points)	
ID:	UC06.5
Actor:	User
Pre-Conditions:	1. Unit is selected. 2. Unit has at least one skill point. 3. Unit's speed is only 1. 4. Unit's rate of fire is above or equal to 30. 5. Unit's level is 15 or above. 6. Unit's class is "newb".
Scenario	1. User pushes space in window. 2. Sub window pops up in window. 3. Sub window displays unit name in window. 4. Sub window displays unit class in window. 5. Sub window displays unit icon (representing Class) in window. 6. Sub window displays experience and necessary experience for next level in window. 7. Sub window displays current health out of max health in window. 8. Sub window displays current level in window. 9. Sub window displays attack power in window. 10. Sub window displays defense power in window. 11. Sub window displays speed rating in window. 12. Sub window displays regeneration rating in window. 13. Sub window displays range in window. 14. Sub window displays rate of fire in window. 15. Sub window displays weapon currently used in window. 16. Sub window displays upgrade for defense in window. 17. Sub window displays upgrade for range in window. 18. Sub window displays upgrade for speed in window (may not always be available). 19. Sub window displays upgrade for rate of fire in window (may not always be available). 20. Sub window displays upgrade for attack in window. 21. Sub window displays upgrade for regeneration in window. 22. Sub window displays upgrade for priest in window. 23. Sub window displays upgrade for wizard in window.

	23. Sub window displays upgrade for soldier in window.
Post-Conditions:	1. User can close statistics window. 2. User can now upgrade the unit. 3. User can now change class to priest, wizard, soldier.

Table 3-13: Statistics Window Display With Skill Points and Wizard Class Upgrades

Use Case: Unit Statistics (Skill Points)	
ID:	UC06.7
Actor:	User
Pre-Conditions:	1. Unit is selected. 2. Unit has at least one skill point. 3. Unit's speed is only 1. 3. Unit's rate of fire is above or equal to 30. 5. Unit's level is 20 or above. 6. Unit's class is wizard.
Scenario	1. User pushes space. 2. Sub window pops up. 3. Sub window displays unit name in window. 4. Sub window displays unit class in window. 5. Sub window displays unit icon(representing class) in window. 6. Sub window displays experience and necessary experience for next level in window. 7. Sub window displays current health out of max health in window. 8. Sub window displays current level in window. 9. Sub window displays attack power in window. 10. Sub window displays defense power in window. 11. Sub window displays speed rating in window. 12. Sub window displays regeneration rating in window. 13. Sub window displays range in window. 14. Sub window displays rate of fire in window. 15. Sub window displays weapon currently used in window. 16. Sub window displays upgrade for defense in window. 17. Sub window displays upgrade for range in window. 18. Sub window displays upgrade for speed in window. (may not always be available) 19. Sub window displays upgrade for rate of fire in window. (may not always be available) 20. Sub window displays upgrade for attack in window. 21. Sub window displays upgrade for regeneration in window. 22. Sub window displays upgrade for fire sorcerer in window. 23. Sub window displays upgrade for frost mage in window.
Post-Conditions:	1. If user pushes space again statistic window will close. 2. User can now upgrade the unit. 3. User can now change class to fire sorcerer or frost mage.

3.3.7 Unit Attribute Upgrade

Attribute upgrades require 1 skill point that is awarded to the unit at each level boundary. The following attribute upgrades are detailed in the six subsequent tables:

- When the defense upgrade icon (shield) is visible in the statistics window the unit's defense can be increased by a factor corresponding to that particular level (the defense level is determined by the equation included in Table 3-14).
- When the range icon (arrow and target) is visible, the unit's range may be increased by a factor of ten (Table 3-15).
- When the speed icon (boot) is visible, the unit's speed may be upgraded by one. Each unit can only have a speed upgrade once during its lifetime (Table 3-16).
- When the rate-of-fire icon (gun and target) is visible, the unit's fire delay may be decreased by ten. Once the delay value reaches 20, this option is no longer available (Table 3-17).
- When the attack icon (gun and wand) is visible, the unit's attack ability may be increased to the corresponding level (Table 3-18).
- When the regeneration icon (red cross) is visible, the rate at which the unit increases health points is increased by one (Table 3-19).

Table 3-14: Upgrading Unit Defense

USE CASE: Attribute Upgrade (Defense)	
ID:	UC07.1
Actor:	User
Pre-Conditions:	1. Skill points must greater than 0. 2. Skill upgrades must be visible.
Scenario	1. Defense upgrade selected by user with left mouse click. 2. Defense increases by 1+round(level/2). 3. One is subtracted from skill points.
Post-Conditions:	1. Defense is now higher.

Table 3-15: Upgrading Unit Attack Range

USE CASE: Attribute Upgrade (Range)	
ID:	UC07.2
Actor:	User
Pre-Conditions:	1. Skill points must greater than 0. 2. Skill upgrades must be visible.
Scenario	1. Range upgrade selected by user with left mouse click. 2. Range is increases by 10. 3. One's subtracted from skill points.
Post-Conditions:	3. Range is now higher.

Table 3-16: Upgrading Unit Movement Speed

USE CASE: Attribute Upgrade (Speed)	
ID:	UC07.3
Actor:	User
Pre-Conditions:	1. Skill points must be greater than 0. 2. Skill upgrade must be visible. 3. Speed must only be 1.
Scenario	1. Speed upgrade selected by user with left mouse click. 2. Speed increases by 1. 3. One is subtracted from skill points.
Post-Conditions:	1. Speed is now higher.

Table 3-17: Upgrading Unit Rate of Fire

USE CASE: Attribute Upgrade (Rate of Fire)	
ID:	UC07.4
Actor:	User
Pre-Conditions:	1. Skill points must be greater than 0. 2. Skill upgrade must be visible. 3. Rate of fire must be 30 or greater.
Scenario	1. Rate of fire upgrade selected by user with left mouse click. 2. Rate of fire decreases by 10. 3. One is subtracted from skill points.
Post-Conditions:	1. Rate of fire is now faster.

Table 3-18: Upgrading a Unit's Attack Power

USE CASE: Attribute Upgrade (Attack)	
ID:	UC07.5
Actor:	User
Pre-Conditions:	1. Skill points must be greater than 0. 2. Skill upgrade must be visible.
Scenario	1. Attack upgrade selected by user with left mouse click. 2. Attack increases by 2 + round(level/2). 3. One is subtracted from skill points.
Post-Conditions:	1. Attack is now higher.

Table 3-19: Upgrading a Unit's Regeneration

USE CASE: Attribute Upgrade (Regeneration)	
ID:	UC07.6
Actor:	User
Pre-Conditions:	1. Skill points must be greater than 0. 2. Skill upgrade must be visible.
Scenario	1. Regeneration upgrade selected by user with left mouse click. 2. Regeneration increases by 1. 3. One is subtracted from skill points
Post-Conditions:	1. The regeneration rate is now higher.

3.3.8 Unit Class Upgrade

The following class level upgrades are only possible at certain levels when the level requirements are met:

- If the unit's level is 5 or greater and current class is “newb”, its classification can be upgraded to soldier at the player's discretion (Table 3-20).
- If the unit's level is 10 or greater and current class is “newb”, its classification can be upgraded to wizard at the player's discretion (Table 3-21).
- If the unit's level is 15 or greater and current class is “newb”, its classification can be upgraded to priest at the player's discretion (Table 3-22).
- If the unit's level is 15 or greater and current class is wizard, it may be given a secondary classification of fire sorcerer at the player's discretion (Table 3-23).

- If the unit's level is 20 or greater and current class is wizard, it may be given a secondary classification of ice mage at the player's discretion (Table 3-24).

Table 3-20: Upgrading a Unit from "NEWB" to Soldier

Use Case: Unit Class Upgrade (Soldier)	
ID:	UC08.1
Actor:	User
Pre-Conditions:	1. Skill points must be greater than one. 2. Class upgrade must be visible.
Scenario	1. Soldier class upgrade is selected with left mouse click. 2. Sprite changes to soldier graphic. 3. Icon changes to soldier graphic (if status window is open). 4. Unit will now receive soldier attribute increases when it levels up.
Post-Conditions:	1. Unit is now a soldier.

Table 3-21: Upgrading a unit from "NEWB" to Wizard

Use Case: Unit Class Upgrade (Wizard)	
ID:	UC08.2
Actor:	User
Pre-Conditions:	1. Skill points must be greater than one. 2. Class upgrade must be visible.
Scenario	1. Wizard class upgrade is selected with left mouse click. 2. Sprite changes to wizard graphic. 3. Icon changes to wizard graphic (if status window is open). 4. Unit will now receive wizard attribute increases when it levels up.
Post-Conditions:	1. Unit is now a wizard.

Table 3-22: Upgrading a unit from "NEWB" to Priest

Use Case: Unit Class Upgrade (Priest)	
ID:	UC08.3
Actor:	User
Pre-Conditions:	1. Skill points must be greater than one. 2. Class upgrade must be visible.
Scenario	1. Priest class upgrade is selected with left mouse click. 2. Sprite changes to priest graphic. 3. Icon changes to priest graphic (if status window is open). 4. Unit will now receive priest attribute increases when it levels up.
Post-Conditions:	1. Unit is now a priest.

Table 3-23: Upgrading a unit from Wizard to Fire Sorcerer

Use Case: Unit Class Upgrade (Fire Sorcerer)	
ID:	UC08.4
Actor:	User
Pre-Conditions:	1. Skill points must be greater than one. 2. Class upgrade must be visible.
Scenario	1. Fire sorcerer class upgrade is selected with left mouse click. 2. Sprite changes to fire sorcerer graphic. 3. Icon changes to fire sorcerer graphic (if status window is open). 4. Unit will now receive fire sorcerer attribute increases when it levels up.
Post-Conditions:	1. Unit is now a fire sorcerer.

Table 3-24: Upgrading a unit from Wizard to Ice Mage

Use Case: Unit Class Upgrade (Ice Mage)	
ID:	UC08.5
Actor:	User
Pre-Conditions:	1. Skill points must be greater than one. 2. Class upgrade must be visible.
Scenario	1. Frost mage class upgrade selected with left mouse click. 2. Sprite changes to frost mage graphic. 3. Icon changes to frost mage graphic (if status window is open). 4. Unit will now receive frost mage attribute increases when it levels up.
Post-Conditions:	1. Unit is now a frost mage.

3.3.9 Building Construction

The management of structures is vital to RTS games. It is a prerequisite for constructing more powerful weapons. The following list details the existing building types and functions:

- The home base must be constructed before any other structures can be built because it is necessary for building the fuzz unit (Table 3-25).
- The market place provides one gold piece per 100 game ticks. This building is also necessary for creating assault rifles for soldier class units and for the construction of a magic store (Table 3-26).

- The bio-dome provides a one-time increase of three additional food points. In addition, this structure is required to build a holy stave for priest class units, and must be constructed before the player can develop gold mines (Table 3-27).
- The gold mine provides the income source for the player and adds three gold pieces to the gold stores for each 100 game ticks. The gold mine is a prerequisite for factory construction (Table 3-28).
- The factory allows the player to build a more powerful weapon for the soldier (a bazooka). In addition to a market place, a factory is also needed as a prerequisite for building a magic shop (Table 3-29).
- The most advanced building is a magic shop, which is required to build three powerful weapons for wizards: the magic hat, fire wand, and ice staff (Table 3-30).

Table 3-25: Constructing the Home Base

Use Case: Construct Building (Home Base)	
ID:	UC09.1
Actor:	User
Pre-Conditions:	1. Game started. 2. Gold requirement is met.
Scenario	1. User selects home base icon on bottom interface bar. 2. Mouse turns to scaffold allowing player to place where he/she whishes the building to be built. 3. Appropriate value is subtracted from gold. 4. Scaffold appears and after build time home base will appear.
Post-Conditions:	1. Player can now build fuzz units from this building.

Table 3-26: Constructing the Marketplace

Use Case: Construct Building (Marketplace)	
ID:	UC09.1
Actor:	User
Pre-Conditions:	1. Home base constructed. 2. Gold requirement is met.
Scenario	1. User selects marketplace icon on bottom interface. 2. Mouse turns to scaffold allowing player to place where he/she wishes the building to be built. 3. Appropriate value is subtracted from gold. 4. Scaffold will appear in location and after build time marketplace will appear.
Post-Conditions:	1. Player can now build assault rifles. 2. Player now receives gold every 100 ticks.

Table 3-27: Constructing the Bio-Dome

Use Case: Construct Building (Bio-Dome)	
ID:	UC09.3
Actor:	User
Pre-Conditions:	1. Home base has been built. 2. Gold requirement is met.
Scenario	1. User selects bio-dome icon in bottom interface window. 2. Mouse turns to scaffold allowing player to place where he/she wishes the building to be built. 3. Appropriate value is subtracted from gold. 4. Scaffold appears in location 5. After build time bio-dome replaces scaffold.
Post-Conditions:	1. Player now gets a one time +3 to max food. 2. Player can now construct holy staves.

Table 3-28: Constructing the Gold Mine

Use Case: Construct Building (Gold Mine)	
ID:	UC09.4
Actor:	User
Pre-Conditions:	1. Bio-dome constructed. 2. Gold requirement is met.
Scenario	1. User selects gold mine icon in the bottom interface window. 2. Mouse pointer turns to scaffold allowing player to place building where they want. 3. Appropriate value is subtracted from gold. 4. Scaffold appears in location 5. After build time gold mine replaces scaffold.
Post-Conditions:	1. Player now receives 3 gold per 100 ticks.

Table 3-29: Constructing the Factory

Use Case: Construct Building (Factory)	
ID:	UC09.5
Actor:	User
Pre-Conditions:	1. Gold mine constructed. 2. Gold requirement is met for construction.
Scenario	1. User selects factory icon in the bottom interface window. 2. Mouse pointer turns to scaffold allowing player to place building where they want. 3. Appropriate value is subtracted from gold. 4. Scaffold appears in location. 5. After build time factory replaces scaffold.
Post-Conditions:	1. User can now build bazookas.

Table 3-30: Constructing Magic Shop

Use Case: Construct Building (Magic Shop)	
ID:	UC09.6
Actor:	User
Pre-Conditions:	1. Factory is built. 2. Marketplace is built. 3. Gold requirement for construction is met.
Scenario	1. User selects magic shop icon in the bottom interface window. 2. Mouse pointer turns to scaffold allowing player to place building where they want. 3. Appropriate value is subtracted from gold. 4. Scaffold appears in location. 5. After build time magic shop replaces scaffold.
Post-Conditions:	1. Player can now build magic hat, fire wands, and ice staves.

3.3.10 Dungeon Exploration

There are three dungeons associated with the final pilot game. The player must explore these dungeons and collect a key from each. This required designing a way for units to enter and leave the dungeons as follows:

- When a unit touches a dungeon tile, it is transported inside. A window at the bottom left of the screen will open and allow the user to explore the dungeon (Table 3-31).

- When a unit touches the door icon inside the dungeon, it is teleported back to world map next to the corresponding dungeon tile. The dungeon window will close when there are no units remaining in the dungeon (Table 3-32).

Table 3-31: Friendly Unit Enters a Dungeon

Use Case: Unit Enter Dungeon	
ID:	UC10.1
Actor:	User
Pre-Conditions:	1. Unit moves toward dungeon.
Scenario	1. Unit collides with dungeon tile. 2. Unit is teleported to beginning area of dungeon. 3. Dungeon window is opened at the bottom left of screen. 4. Global dungeon count +1 (Note that only one dungeon at a time can be open at the same time).
Post-Conditions:	1. Unit can now move or attack in dungeon. 2. Unit can move toward door tile and exit.

Table 3-32: Unit Leaves a Dungeon

Use Case: Unit Leaves Dungeon	
ID:	UC11.1
Actor:	User
Pre-Conditions:	1. Unit moving toward door icon.
Scenario	1. Unit collides with door icon. 2. Unit teleported to outside, by dungeon icon. 3. Global dungeon count -1. 4. If global dungeon count = 0 then close window.
Post-Conditions:	1. Unit can now attack or move. 2. Unit can re-enter dungeon.

3.3.11 World Start

The world start script is called when a map is first loaded. Upon game initialization, the enemy is placed at various positions of the game world. In addition, a starting amount of gold is assigned to the player and level thresholds are calculated according to the assigned delay factor (Equation 2.1). Table 3-33 details this process.

Table 3-33: Game Initiation

Use Case: World Start	
ID:	UC12.1
Actor:	System
Pre-Conditions:	1. Game has just started.
Scenario	1. Player array is created. 2. Enemy array is created. 3. Global parameters are created. 4. Player gold is set. 5. Fuzz unit created (initial one). 6. Enemies created statically and placed in “bad-guy” array. 7. Building parameters are set to 0 (no buildings built). 8. Food set to 1. 9. Food used set to 0. 10. Number of keys set to 0.
Post-Conditions:	1. Game can be played.

3.3.12 Build Unit

Each fuzz unit is built as a level one class “newb” and requires 25 gold pieces and one food point. Therefore, in order to create new units, the bottom tool bar must indicate that the available/max food exceeds the food used, and that enough gold pieces are available (Table 3-34).

Table 3-34: Building a Unit

Use Case: Build Unit	
ID:	UC13.1
Actor:	User
Pre-Conditions:	1. Gold requirements are met. 2. Food used is at least one less than total food.
Scenario	1. User selects base camp. 2. User selects build fuzz icon in interface window. 3. Appropriate amount of gold is taken out of users gold resource (25). 4. Food used is incremented by 1. 5. Unit is created and added to players array (50 max). 6. Default attributes are given to the unit. 7. Unit is classified as “newb” and given a pistol. 8. Unit is level 1 and has 1 skill point to spend. 9. Unit appears inside the base camp icon.
Post-Conditions:	1. Unit can now move. 2. Unit can now attack. 3. Unit can be upgraded.

3.3.13 Build Weapon

Providing that the player has enough gold, weapons can be created and stored for use when units meet the requirements to use them. Weapons are assembled in four different structures:

- Holy staves, used by priests, are constructed in the bio-dome (Table 3-35).
- Assault rifles are constructed in the market place (Table 3-36).
- Bazookas are built in the factory and can penetrate numerable obstacles inflicting much more damage than assault rifles (Table 3-37).
- Magic hats are created in the magic shop and increase the wizard's attack power (Table 3-38).
- Fire wands are built in the magic shop and increase the fire sorcerer's attack power more than the magic hat (Table 3-39).
- Ice staffs are also produced in the magic shop and increase the ice mage's attack power more than the magic hat. These are the most powerful weapons of the game (Table 3-40).

Table 3-35: Building the Holy Stave

Use Case: Build Weapon (Holy Stave)	
ID:	UC14.1
Actor:	User
Pre-Conditions:	1. Bio-dome is selected. 2. Gold requirement is adequate.
Scenario	1. User selects build “holy stave” icon. 2. Appropriate money is withdrawn from the gold resource. 3. Holy stave is added to the weapons list.
Post-Conditions:	1. Holy stave can now be equipped by a priest.

Table 3-36: Building the Assault Rifle

Use Case: Build Weapon (Assault Rifle)	
ID:	UC14.2
Actor:	User
Pre-Conditions:	1. Market place is selected. 2. Gold requirement is adequate.
Scenario	1. User selects build “assault rifle” icon. 2. Appropriate money is withdrawn from the gold resource. 3. Assault rifle is added to the weapons list.
Post-Conditions:	1. Assault rifle can now be equipped by a soldier.

Table 3-37: Building the Bazooka

Use Case: Build Weapon (Bazooka)	
ID:	UC14.3
Actor:	User
Pre-Conditions:	1. Factory is selected. 2. Gold requirement is adequate.
Scenario	1. User selects build “bazooka” icon. 2. Appropriate money is withdrawn from the gold resource. 3. Bazooka is added to the weapons list.
Post-Conditions:	1. Bazooka can now be equipped by a soldier.

Table 3-38: Building the Magic Hat

Use Case: Build Weapon (Magic Hat)	
ID:	UC14.4
Actor:	User
Pre-Conditions:	1. Magic shop is selected. 2. Gold requirement is adequate.
Scenario	1. User selects build “magic hat” icon. 2. Appropriate money is withdrawn from the gold resource. 3. Magic hat is added to the weapons list.
Post-Conditions:	1. Magic hat can now be equipped by a wizard, fire sorcerer, and ice mage.

Table 3-39: Building the Fire Wand

Use Case: Build Weapon (Fire Wand)	
ID:	UC14.5
Actor:	User
Pre-Conditions:	1. Magic shop is selected. 2. Gold requirement is adequate.
Scenario	1. User selects build “fire wand” icon. 2. Appropriate money is withdrawn from the gold resource. 3. Fire wand is added to the weapons list.
Post-Conditions:	1. Fire wand can now be equipped by a fire sorcerer.

Table 3-40: Building the Ice Staff

Use Case: Build Weapon (Ice Stave)	
ID:	UC14.6
Actor:	User
Pre-Conditions:	2. Magic shop is selected. 3. Gold requirement is adequate.
Scenario	1. User selects build “ice stave” icon. 2. Appropriate money is withdrawn from the gold resource. 3. Ice stave is added to the weapons list.
Post-Conditions:	1. Ice stave can now be equipped by a ice mage.

3.3.14 Equip Weapon

To equip a unit, the user selects the appropriate fuzz unit and clicks the middle mouse button or the "E-quip" icon on the lower right of the screen. A list of all weapons available to the player is displayed and the desired weapon may be selected. If the unit class is appropriate, it will be equipped with the selected weapon (Table 3-41). If the unit does not fulfill class requirements for the weapon, nothing changes and a message box is displayed that says "This class cannot use this weapon" (Table 3-42).

Table 3-41: Equip Weapon Correctly

Use Case: Equip Weapon (Success)	
ID:	UC15.1
Actor:	User
Pre-Conditions:	1. Fuzz unit selected. 2. Either middle mouse button pressed or “E-quip” icon in bottom right is pressed.
Scenario	1. Once mouse button or icon pressed list opens. 2. User selects desired weapon. 3. Class can use desired weapon. 4. The weapon string is placed into unit’s weapon. 5. The old weapon the unit was using is placed into available weapons list.
Post-Conditions:	1. Next time unit fires weapon string will be compared and new attack will be created.

Table 3-42: Attempting to Equip an Invalid Weapon

Use Case: Equip Weapon(Failure)	
ID:	UC15.2
Actor:	User
Pre-Conditions:	1. Fuzz unit selected. 2. Either middle mouse button pressed or “E-quip” icon in bottom right pressed.
Scenario	1. Once mouse button or icon pressed weapons list opens. 2. User selects desired weapons. 3. Class cannot use weapon. 4. Error message pops up.
Post-Conditions:	1. Nothing has changed the user can try to equip a different weapon or go on using the same weapon.

3.3.15 Manage Resource

Resources are managed automatically as a background process. Therefore, market places and gold mines provide the appropriate amount of gold every 100 cycles in the game, automatically. Available food is also managed in the background processes (Table 3-43).

Table 3-43: Resource Management System

Use Case: Manage Resource	
ID:	UC16.1
Actor:	Time
Pre-Conditions:	1. Gold mine, marketplace, and/or bio-dome built.
Scenario	1. For every 100 game cycles user gets 1 gold per market place. 2. Fore every 100 cycles user gets 3 gold per gold mine. 3. For ever bio-Dome user gets +3 to max food once (Not time dependant).
Post-Conditions:	

3.3.16 Game Management

The Manage Game Use Case provides the enemy artificial intelligence. This function controls the monster's movements, attacks, and occasionally produces more monsters to provide opportunity for additional units to level up (Table 3-44).

Table 3-44: Game Management System

Use Case: Manage Game	
ID:	UC17.1
Actor:	System
Pre-Conditions:	1. Game has started.
Scenario	1. System checks to see if enemies collide with walls or blocking terrain. 2. When units collide they will be assigned a random direction. 3. When enemies are within range they will stop and attack players units. 4. The system will randomly choose array locations of enemies. 5. When the array locations are empty (enemy has been defeated) it will create a new enemy using that array locations. 6. System checks to see if all three keys are taken by player, the game ends.
Post-Conditions:	

3.3.17 Special Abilities

Except for the “NEWB” class, every class has a special ability associated with it described as follows:

- The soldier units have the ability to increase speed for a short time, when attacked, to better avoid the enemy. This action requires magic points (not

replenished very quickly in the soldier unit) and is not always available (Table 3-45).

- A priest unit has the ability to heal friendly units who are short on health points. Magic points must be available to perform this function (Table 3-46).
- The wizard can create a defense shield around itself making it more difficult to be hit. Magic points are also required to perform this function (Table 3-47).
- The fire sorcerer has the ability to summon a "fire elemental" into the world. This "fire elemental" is another unit that can move at a speed of one and fire a bazooka. In addition, the “fire elemental" can absorb the enemy attack, preserving the fire sorcerer's life. Magic points are also required for this function (Table 3-48).
- The ice mage has the ability to fire a "frost nova" in eight directions. This is the most powerful special ability in the game and requires a considerable number of magic points, limiting the number of times it can be used (Table 3-49).

Table 3-45: Soldier Special Dash Ability

Use Case: Special Abilities (Soldiers DASH)	
ID:	UC18.1
Actor:	Fuzz
Pre-Conditions:	1. Unit must be class soldier. 2. Unit must have enemy within range. 3. Unit must have enough magic points for the desired special ability.
Scenario	1. Soldier unit recognizes threat. 2. Soldier unit checks a random percent to see if it will perform speed boost. 3. Soldier unit passes percent and increases its speed temporarily. 4. Magic points subtracted.
Post-Conditions:	1. Soldier unit can now move faster than other units. 2. Magic points begin to go up again until they hit the maximum value.

Table 3-46: Priest Special Heal Ability

Use Case: Special Abilities (Priest Heal)	
ID:	UC18.2
Actor:	Fuzz
Pre-Conditions:	1. Unit must be class priest. 2. Other units must be less than full health. 3. Enemy must be within range. 4. Magic points must be high enough to perform heal.
Scenario	1. Soldier unit recognizes threat. 2. Priest checks to see if any friendly units are in range. 3. Priest checks a random percent whether to heal or not. 4. 5. When the priest pass the check priest will heal all friendly units within its range. 6. Magic points are reduced.
Post-Conditions:	1. All fuzz units within the range are now at higher health points (cannot exceed max hp). 2. Priest will begin to regain magic points.

Table 3-47: Wizard Special Shield Ability

Use Case: Special Ability (Wizard Defense)	
ID:	UC18.3
Actor:	Fuzz
Pre-Conditions:	1. Unit must be class wizard. 2. Enemy units must be within range. 3. Magic points must be high enough to perform defense shield.
Scenario	1. Soldier unit recognizes threat. 2. Wizard checks a random percent to see if it performs shield or not. 3. When wizard passes the check and increases its defense temporarily. 4. Magic points are reduced.
Post-Conditions:	1. Wizard now has a temporary higher defense rating. 2. Wizard begins to regain magic points.

Table 3-48: Fire Sorcerer Special Fire Elemental Ability

Use Case: Special Ability (Fire Elemental)	
ID:	UC18.4
Actor:	Fuzz
Pre-Conditions:	1. Unit must be class fire sorcerer. 2. Enemy unit must be within range. 3. Magic points must be high enough to summon elemental.
Scenario	1. Soldier unit recognizes threat. 2. Fire sorcerer checks a random percent to see if it will summon elemental. 3. When fire sorcerer passes it creates a new entity that resembles a walking

	flame. 4. After a while the entity will disappear. 5. Magic points are reduced from fire sorcerer.
Post-Conditions:	1. Player can move fire elemental. 2. Fire elemental will attack enemies. 3. Fire Sorcerer will begin to regain magic points.

Table 3-49: Ice Mage Special Frost Nova Ability

Use Case: Special Ability (Frost Nova)	
ID:	UC18.5
Actor:	Fuzz
Pre-Conditions:	1. Unit must be class ice mage. 2. Enemy must be within range. 3. Magic points must be high enough to perform attack.
Scenario	1. Soldier unit recognizes threat. 2. Ice mage checks a random percent to see if it will use its ability. 3. When the ice mage passes it will shoot ice blast in every direction (only eight directions). 4. Magic points are reduced from unit.
Post-Conditions:	1. Ice blast deal powerful damage and can kill most units in one shot. 2. Ice mage will begin to regain magic points.

3.4 Requirement Fulfillment

Table 3-50 maps game requirements to use cases revealing that all level 1 requirements were met. These are requirements labeled R1 to R86.

Table 3-50: Requirement Fulfillment Table

[Continued on next page]

R#\UC	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18
R01									√									
R02									√									
R03				√	√													
R04									√									
R05																√		
R06												√	√					
R07													√					
R08														√				
R09														√				
R10														√				
R11														√				
R12														√				
R13														√				
R14									√									

R#\UC	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18
R15									√									
R16									√									
R17									√									
R18									√									
R19												√						
R20						√												
R21						√												
R22						√												
R23						√												
R24						√												
R25						√												
R26						√												
R27						√												
R28						√												
R29						√												
R30						√												
R31						√												
R32						√												
R33						√												
R34						√												
R35						√												
R36						√												
R37						√												
R38						√												
R39						√												
R40						√												
R41						√												
R42						√												
R43							√											
R44							√											
R45							√											
R46							√											
R47							√											
R48							√											
R49						√												
R50						√												
R51						√												
R52						√												
R53						√												
R54								√										
R55								√										
R56								√										
R57								√										
R58										√								
R59											√							
R60																	√	
R61																	√	
R62													√					
R63					√													
R64													√					

R#\UC	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18
R65			√															
R66			√															
R67			√															
R68																	√	
R69				√														
R70												√					√	
R71	√																	
R72		√																
R73		√																
R74																√		
R75										√								
R76											√							
R77						√												
R78																		√
R79																		√
R80																		√
R81																		√
R82																		√
R83				√														
R84																		√
R85															√			
R86															√			

3.5 Modified Requirements

The player can create a maximum number of ten groups. To assign a unit to a group, the user selects the desired unit and then presses the control key. A list of zero to nine appears and the user selects the group number this unit will be associated with. In order to re-assign a unit to another group this process is repeated. When the user selects a number key (0 to 9) the corresponding group of units will highlight in blue, and this group can then be moved as a unit. Additional requirements for group selection are detailed in Table 3-51.

Table 3-51: Additional Requirements

AR01[1]	The SYSTEM shall have groups from 0 – 9.
AR02[1]	The SYSTEM shall, by default, assign every new fuzz unit to group 10 (no group).
AR03[1]	The SYSTEM shall, if control is pressed on the key board, assign the selected fuzz to a group depending on which number is selected in a menu.
AR04[1]	The SYSTEM shall enable all units within a group to move if the user has pushed the corresponding group number without control being pressed.
AR05[1]	The SYSTEM shall make the fuzz units stop when they collide with one another. Otherwise they would merely stand on top of each other ruining the effect.

AR06[1]	The SYSTEM shall, once a group has been selected, allow the user to right click on a point all fuzz units within that group will attempt to move to that point.
AR07[1]	The SYSTEM shall clear the group move flag if the user selects an individual unit with the mouse (thus, individual movement is still possible).
AR08[1]	A limitation with this implementation is that once a unit has been re-assigned to a group it will not be able to return to group 10 (no assignment group). Therefore, once a unit is assigned it will always be part of a group regardless if it is re-assigned or not.

3.6 Modified Use Cases

Four new use cases were added to incorporate the group movement function. These use cases are detailed in table 3-5.2.

Table 3-52: Additional Use Cases

Additional Use Cases		
AUC01	Group assignment	When control is pushed a menu will appear allowing the user to assign the specific fuzz unit to a group 0-9.
AUC02	Group Selection	If the user pushes a numeric key the corresponding group will be selected for movement. (Note: this is not the global select variable for the status window. This is a separate variable).
AUC03	Group Movement	After a group is selected, the user can right click with the mouse and the fuzz units will try to move toward that point. They will stop if they hit each other or blocking terrain.
AUC04	Group De-Selection	If the user selects an individual unit. The group movement flag is turned off. Therefore, the user only controls one unit.

3.6.1 Group Assignment

Table 3-53 details the process the player must perform in order to assign a fuzz unit to a group.

Table 3-53: Group Assignment for Individual Units

Use Case: Group Assignment	
ID:	AUC01.1
Actor:	User
Pre-Conditions:	1. Game started. 2. Fuzz selected.
Scenario	1. User pushes control. 2. Menu pops up 0-9. 3. User selects a group. 4. The selected fuzz changes its group number to the corresponding number.
Post-Conditions:	1. Fuzz can now be moved in a group. 2. Player can select individual fuzz again.

3.6.2 Group Selection

Table 3-54 details the process the player must perform in order to select a group that fuzz units have been assigned to.

Table 3-54: Group Selection

Use Case: Group Selection	
ID:	AUC02.1
Actor:	User
Pre-Conditions:	1. Game started. 2. Fuzz units have been assigned to that group.
Scenario	1. User pushes digit 0 – 9. 2. Global variable set to that group's number.
Post-Conditions:	1. Units can now move as a group. 2. User can select new unit or group.

3.6.3 Group Movement

Table 3-55 details the process the player must perform in order to move a group that is currently selected.

Table 3-55: Group Movement

Use Case: Group Move	
ID:	AUC03.1
Actor:	User
Pre-Conditions:	1. Game started. 2. Group selected.
Scenario	1. User right clicks the mouse at desired location. 2. Fuzz checks to see if global group variable is equal to its group variable. 3. If so fuzz begins to move toward point. 4. Fuzz will stop if it comes in contact with another fuzz unit or blocking terrain.
Post-Conditions:	1. Fuzz units can be moved again. 2. New group or unit can be selected.

3.6.4 Group De-Selection

Table 3-56 details the process the player must perform in order to de-select a group. This process is done automatically when the player selects another entity with the mouse.

Table 3-56: Group De-Selection (automatic)

Use Case: Group De-select	
ID:	AUC04.1
Actor:	User
Pre-Conditions:	1. Game started. 2. Group selected.
Scenario	1. User will left click the mouse on a unit or a building. 2. Global group selected will be set to either “noone” (null) or group11 (unusable group).
Post-Conditions:	1. Player can now move single unit. 2. Player can now select others unit, buildings or groups.

Now that the requirements have been listed out with great detail, the complexity of this program can be appreciated. This process of creating requirements allows the programmer to identify tangible goals to achieve. This process would be helpful to game designers who were interested in the game play of a game.

4 Chapter 4: High Level Design of Fuzz's Revenge

This chapter addresses the high level design of the pilot game. Due to space limitations not all of the UML (Unified Modeling Language) design models used for the game are outlined in this chapter. The design diagram shown in Figure 4-1presents the placement of modules in the program. Subsequent flow charts outline each component of the prototype game. Modules can fall under one of the following four categories:

- Mouse/Keyboard Input
- Display
- Combat
- Resource and System Management

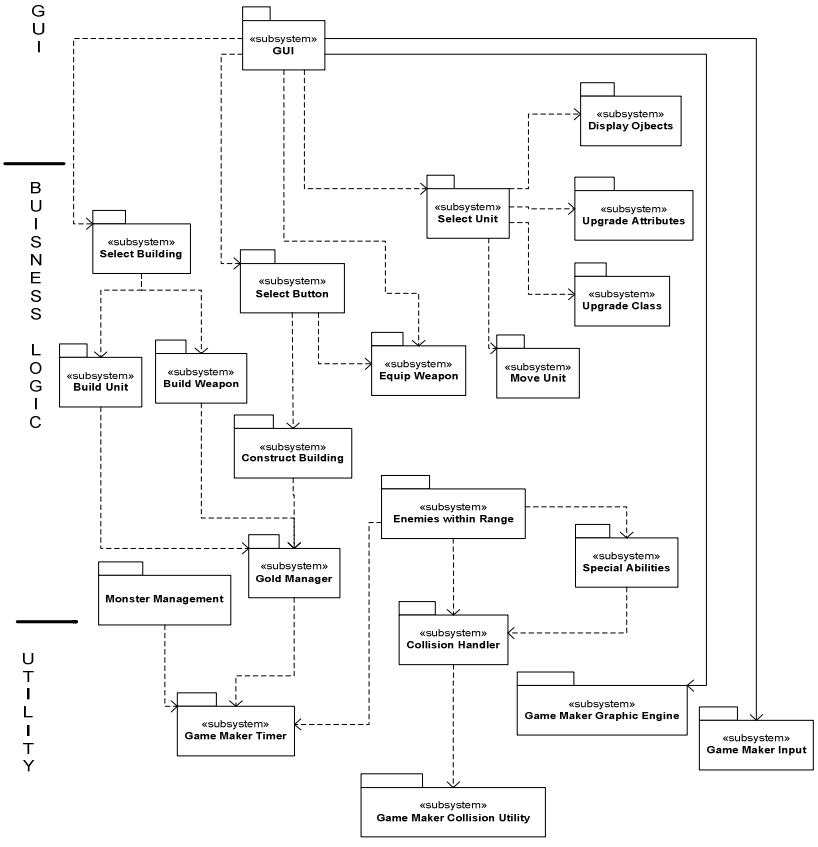

Figure 4-1: High Level Diagram of Fuzz's Revenge

4.1 Mouse Input

Due to the interactive style in which this game is played, many modules provide mouse functionality. When an object is selected, these modules determine the appropriate outcome for that particular object. Clicking on different objects has different consequences. For example, clicking on a building may bring up buttons for weapon or unit construction, but clicking on a fuzz unit merely selects that unit. Figure 4.2 details the modules that are dedicated to mouse control.

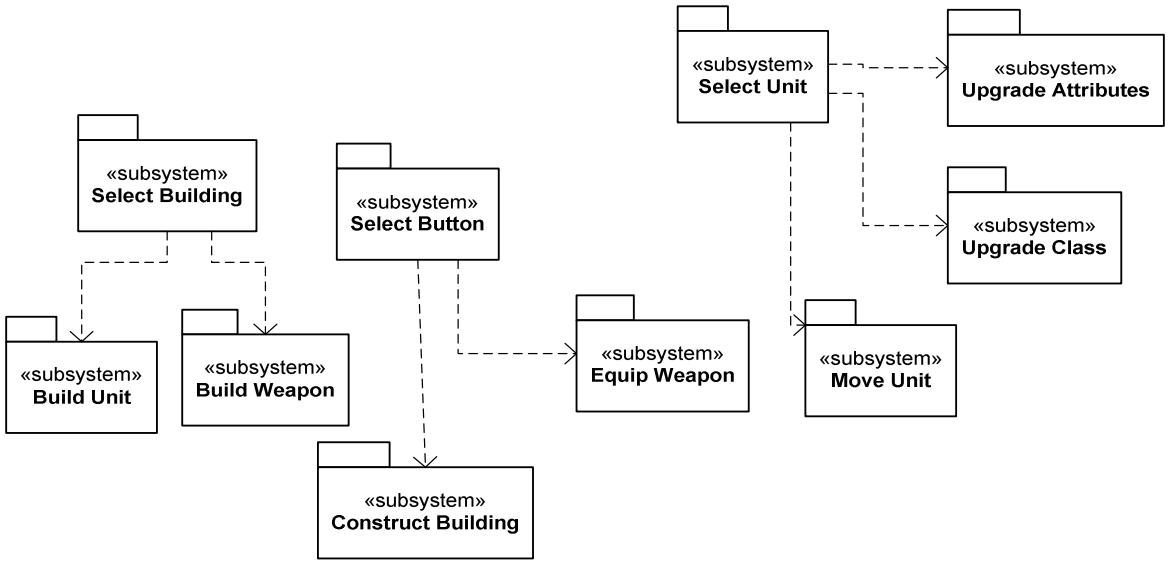

Figure 4-2: Modules Dedicated to Mouse Control

The four main scenarios for the left mouse click are i) upgrading unit statistics, ii) building weapons or units, iii) selecting buildings or units, and iv)construction of new buildings. These functionalities are explained below and illustrated graphically in Figure 4-3.

- If the mouse's left button is pushed while the cursor is pointing to a unit upgrade, that upgrade is applied to the selected unit.
- If the mouse's left button is pressed on a fuzz unit, the unit will be selected and can be moved or have its statistics viewed by pressing the space bar. Building-specific functions appear at the bottom of the screen when buildings are selected with the left mouse button.

- When a “Build Weapons” button is pressed, the player’s gold will be reduced by the appropriate amount and the list of available weapons will increase.
- When a “Build Fuzz Unit” button is pressed, a new fuzz dialogue window will appear asking for a name for the new unit.
- Multiple events occur when the “Construct Building” button is selected. First, if the player has the necessary gold, a scaffold will appear on the screen following the mouse cursor. Second, when the player clicks the left mouse button on the desired location for the new building, construction will begin if no other object is present at the selected location. The player cannot select another object until a suitable location has been found for the new building.

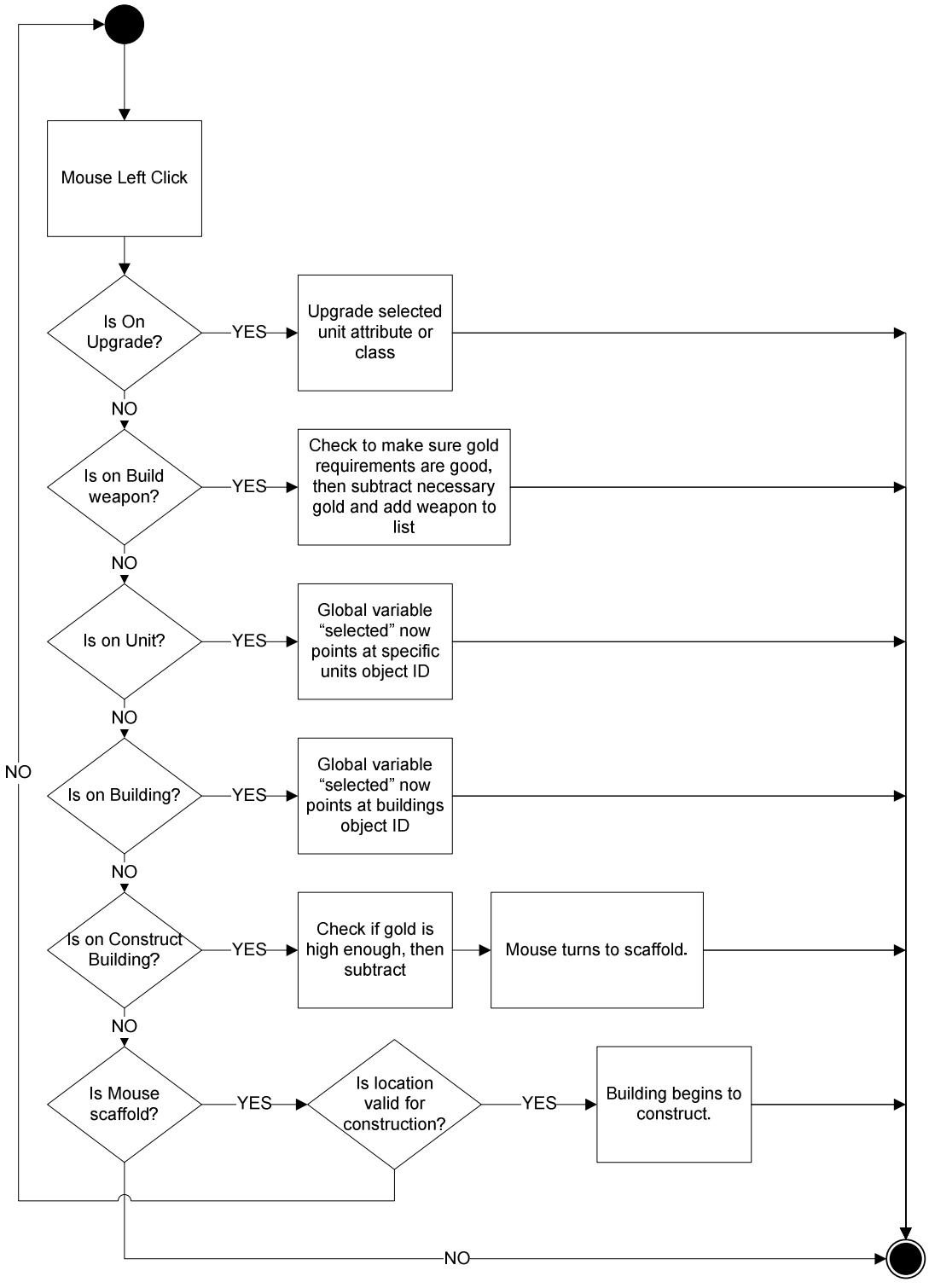

Figure 4-3: flowchart of the Left Mouse Button’s Multiple Functions

Unit movement requires that the user has selected a fuzz unit. This is controlled exclusively with the right mouse button. There is no *"path finding"* functionality in Fuzz's Revenge. Therefore, if a unit collides with a blocking object it will stop. As Figure 4-4 reveals, even with this simplified approach, the process is still complex.

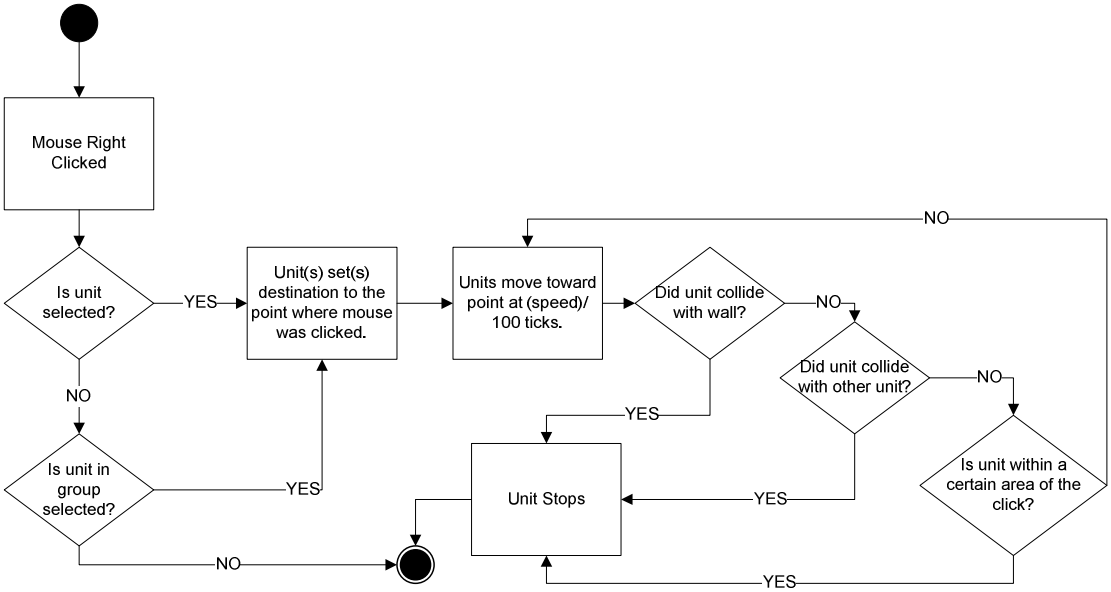

Figure 4-4: Unit Movement Controlled by Right Mouse Button

The player presses the middle mouse button or the scroll wheel to equip fuzz units with weapons (Figure 4-5). Players using two-button mice can select the "E-quip" icon to accomplish this task. After pressing the middle mouse button on the appropriate fuzz unit, a list of all available weapons will appear and a new weapon can be selected. The previous weapon associated with the fuzz unit will then be placed in the list of available weapons. Therefore, weapons are not lost but moved from the unit to the existing weapons list.

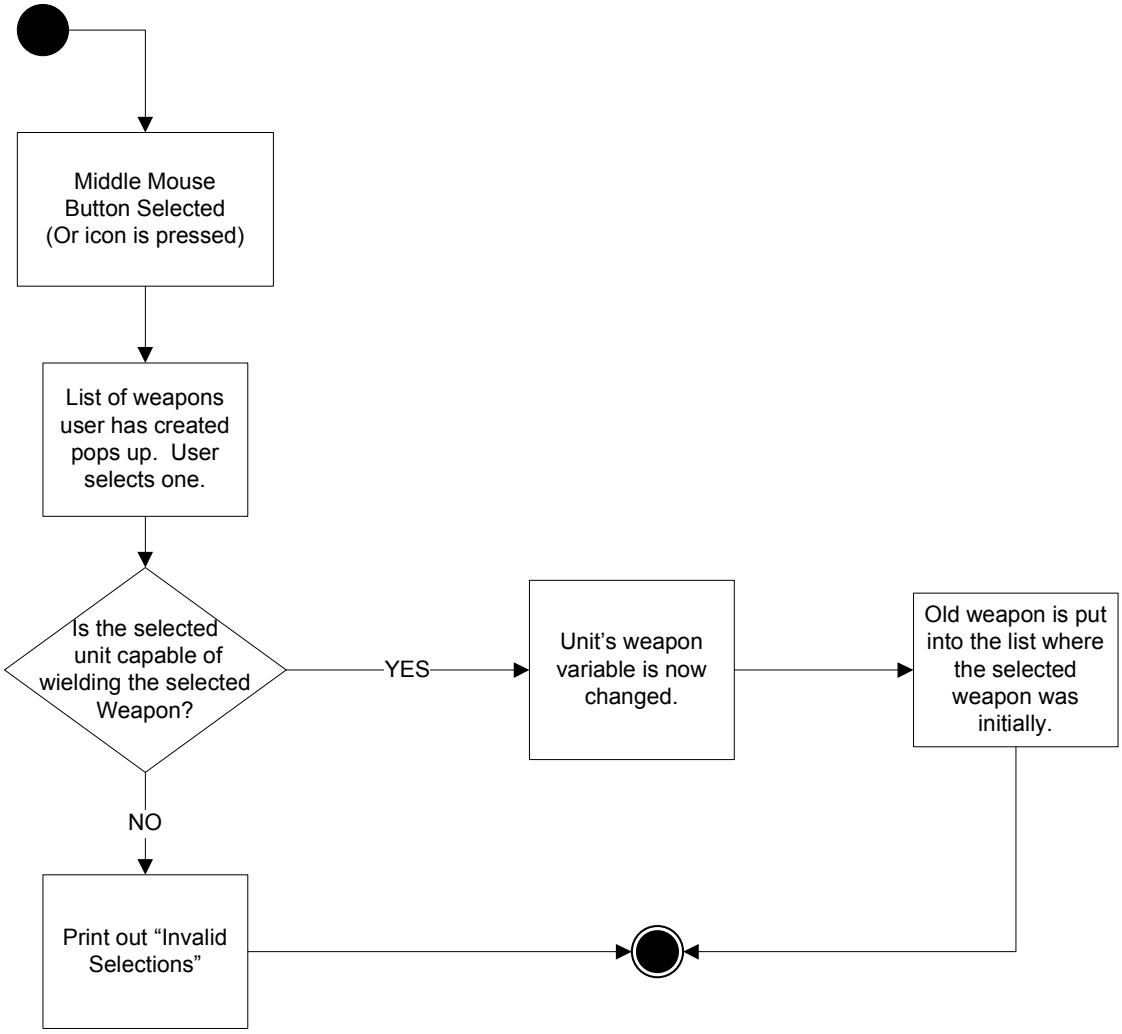

Figure 4-5: Equipping Fuzz Units With Weapons

Figures 4-6 and 4-7 detail attribute and class enhancements available for fuzz unit upgrades. Upgrade buttons appear only when the selected fuzz unit qualifies for the upgrade.

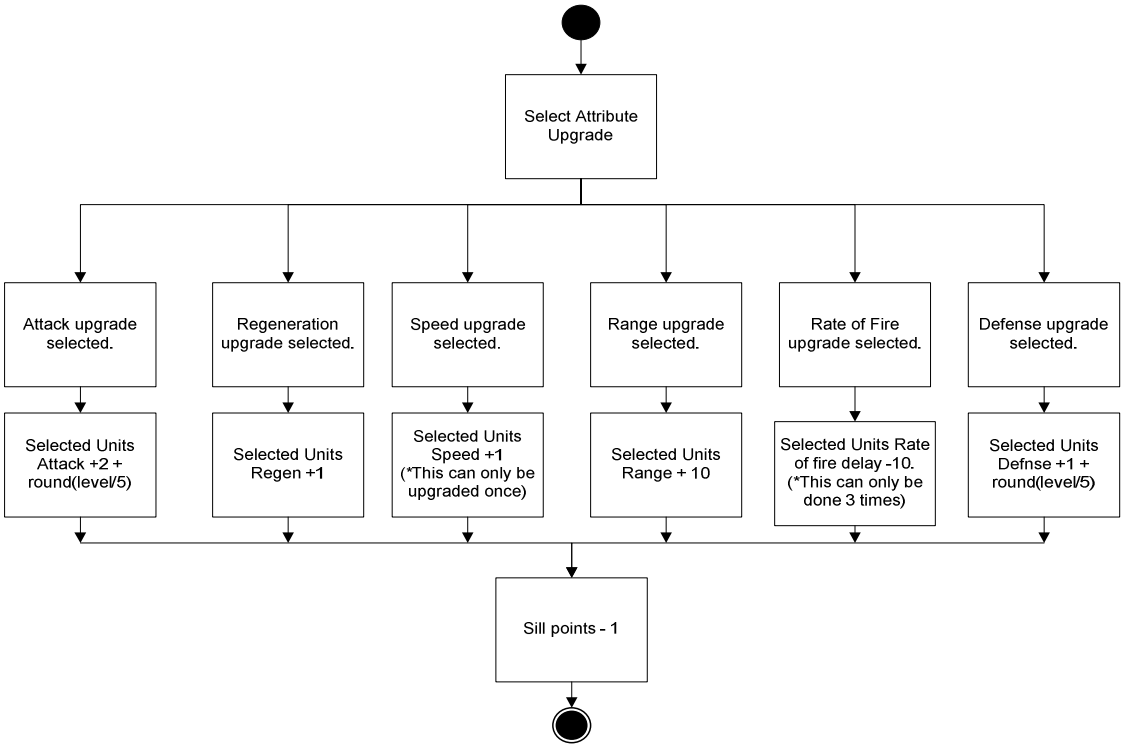

Figure 4-6: Attribute Upgrade Process

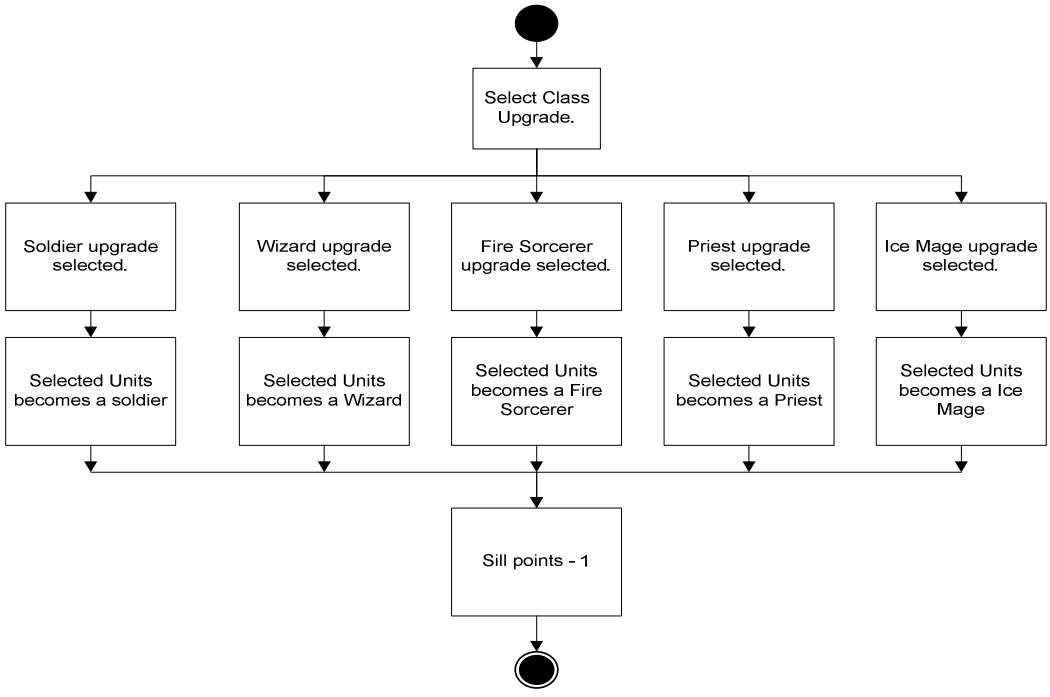

Figure 4-7: Class Upgrade Process

4.2 Display System

The display system is encapsulated in the Game Maker's graphic engine and each object can have a sub-routine called *draw()*. During screen refresh, the Game Maker will call each object's *draw* function and perform the code that is written there. If there is no draw routine, such as in the case of terrain objects, the default sprite will be drawn. The default sprite is a setting that can be changed before the game is compiled. It is a static picture of some entity that can either have transparent or filled background. The enemy's draw routine adds their names, health bar, and animated sprite or moving picture, to the video buffer. The building and fuzz unit draw routines are more complicated than the enemy's (Figure 4-8). A white frame will always be drawn around the selected unit. A selected building will display available options (i.e., to build weapons or fuzz units). The fuzz unit's name and health bar is always displayed on the screen. When the fuzz unit is selected and the space bar depressed a sub-window appears showing the appropriate unit statistics. In addition, upgrade buttons will appear if the unit has skill points and new class upgrade options will appear if the unit level/class requirement is met.

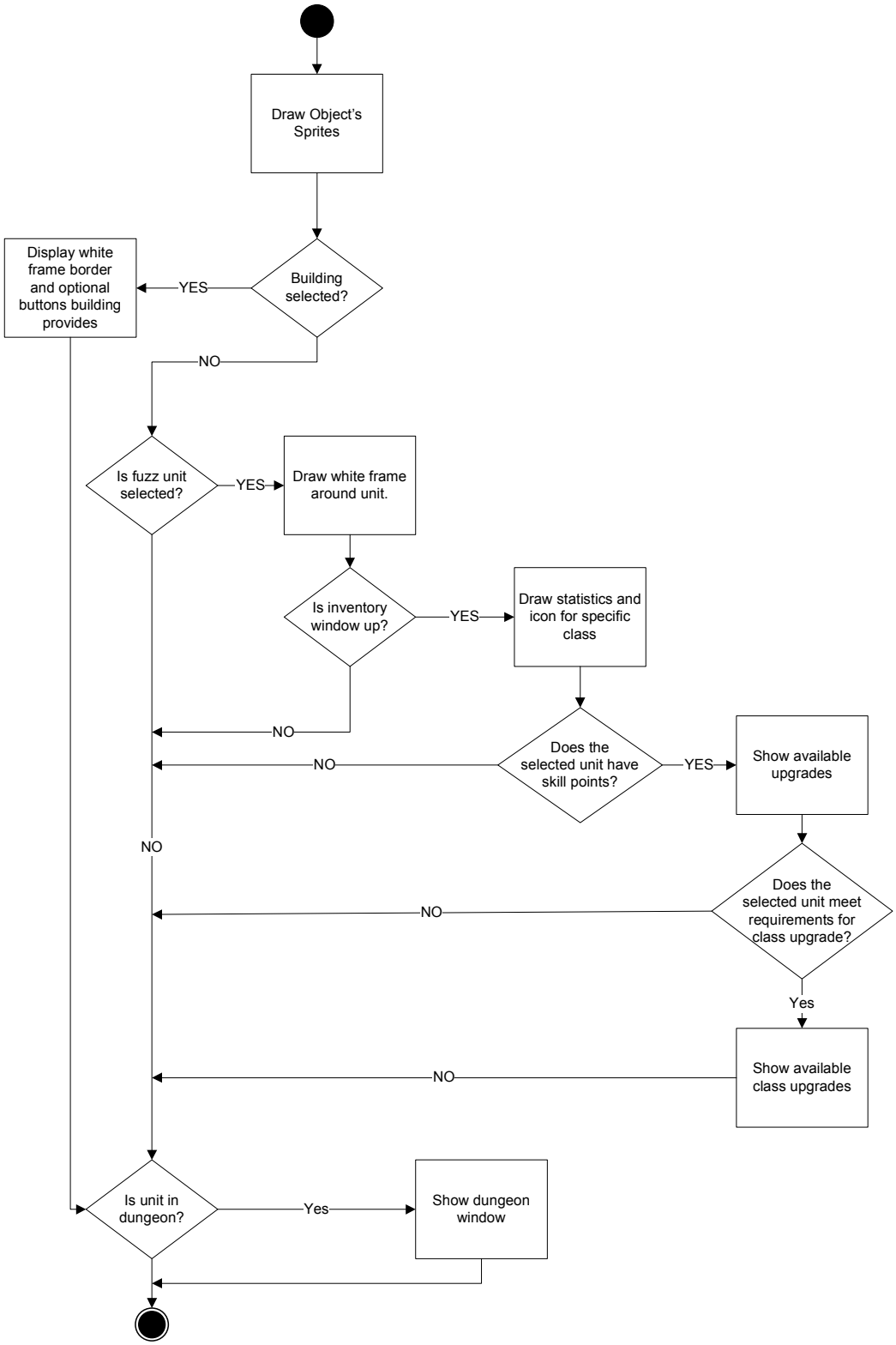

Figure 4-8: Display System Process

4.3 Combat System

As this is a pilot game designed to test a theory of combining game genres, a simple battle sub-module was implemented, in which both enemy and fuzz units automatically attack each other when they come within range(Figure 4-9). Also, a delay variable controls the rate of fire. Once the fuzz unit becomes a class other than "NEWB", it will gain a special ability. If the character has enough magic points, a random number is generated. If this generated number is less than a pre-defined unit value, the fuzz unit will perform the special ability associated with its class. Therefore chance is introduced into the game that can influence outcome. Enemy units do not have special abilities.

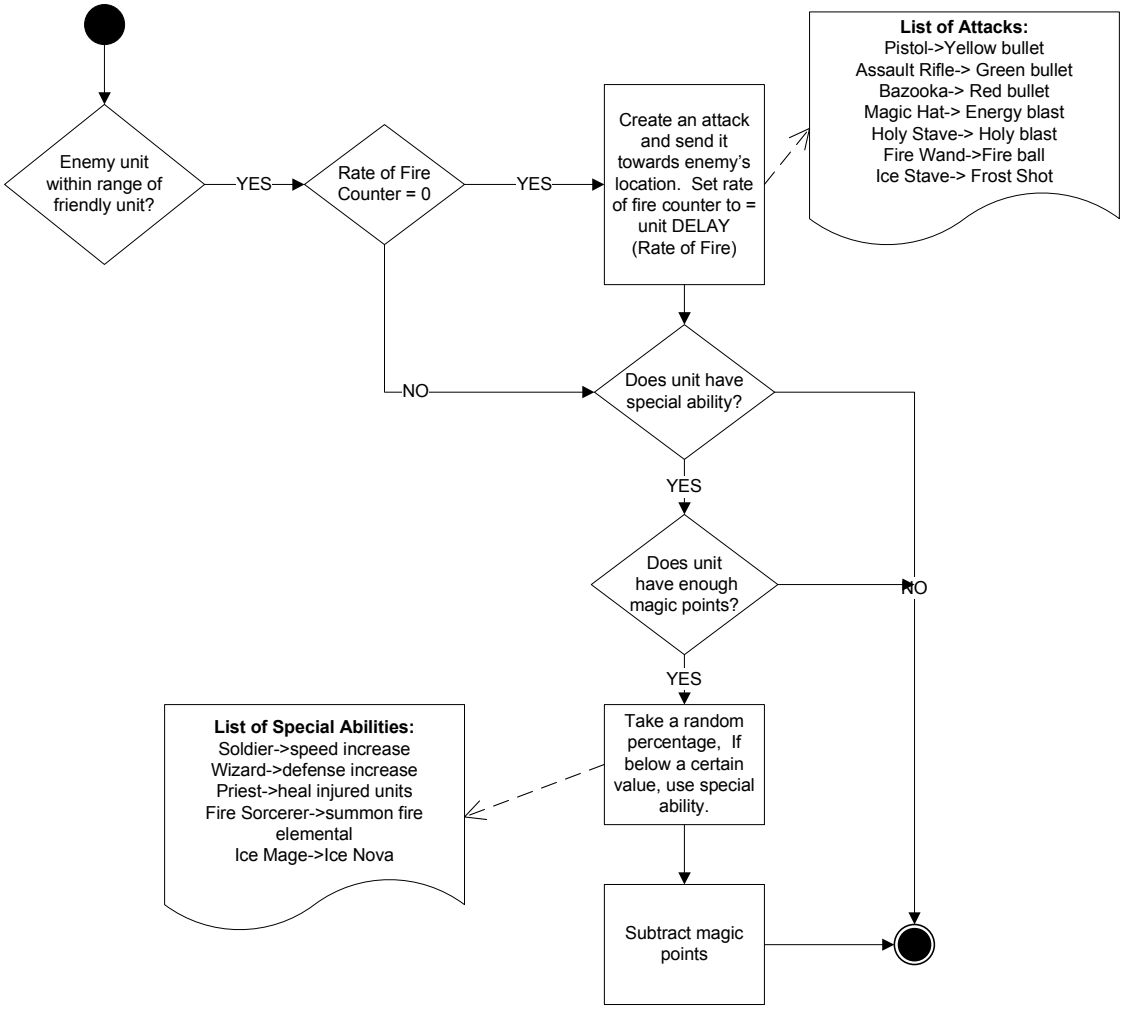

Figure 4-9: Basic Combat System Process

When the attack initiated by the fuzz unit reaches the enemy, the program will compare the attack with the enemy's defenses to determine the effect. If the attack is greater than the defense, the enemy unit takes damage, and if the enemy's health falls to zero it is destroyed. When the enemy is destroyed, fuzz units within range receive experience points and the player receives the appropriate amount of gold. This process is depicted in Figure 4-10.

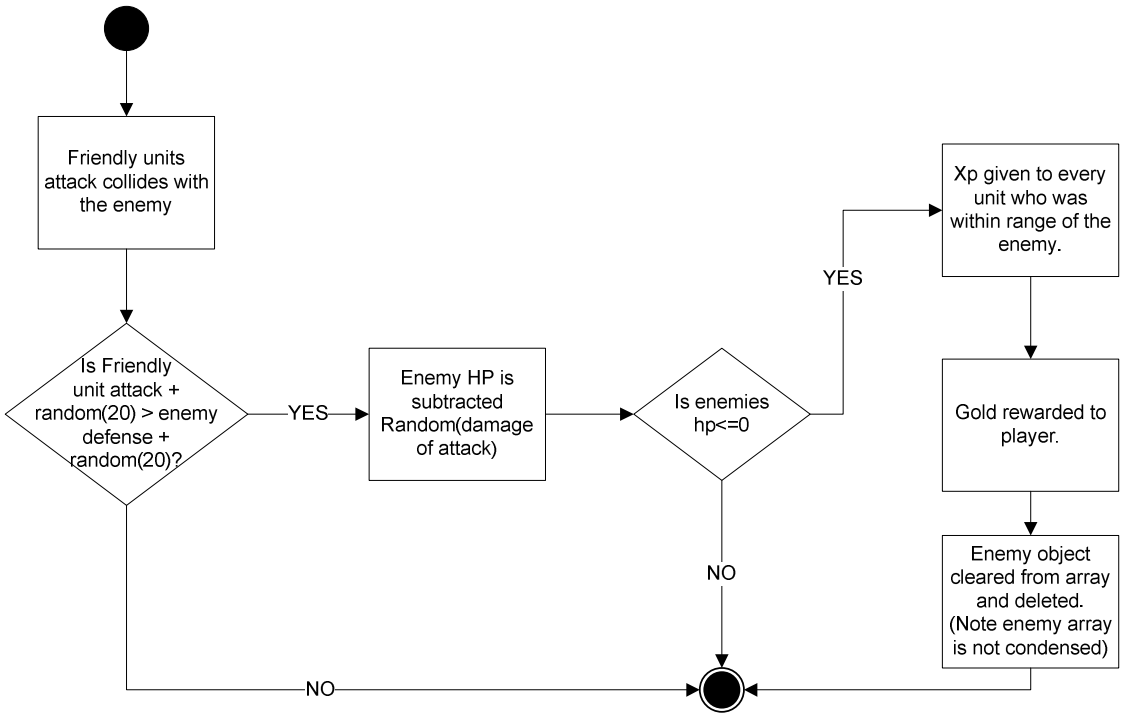

Figure 4-10: Fuzz Unit Victory Process

In contrast with the previous scenario is the situation where the fuzz unit is hit by an enemy's attack and takes damage (Figure 4-11). If the fuzz unit's health falls to zero, it is destroyed and the player's array is adjusted. After this is done, the unit is destroyed and any weapons it was equipped with are lost.

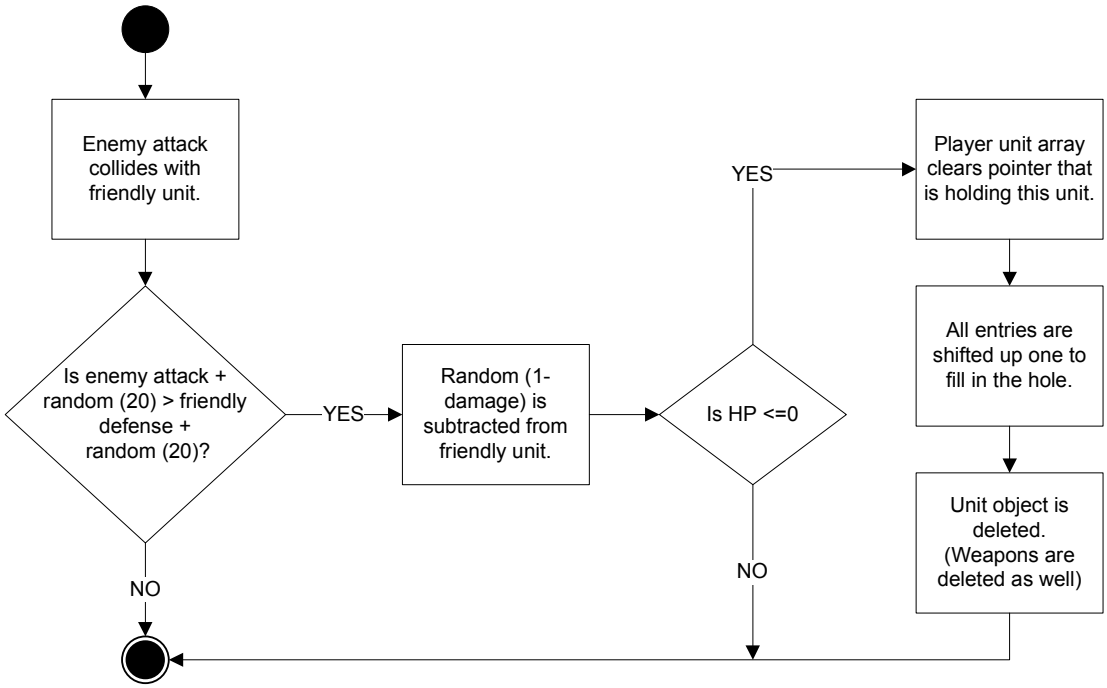

Figure 4-11: Fuzz Unit Defeat Process

4.4 System Management

The System Management has two parts, one the resource management of the game and two, controlling the enemy. The first sub-system takes care of various miscellaneous tasks to keep the game running. The gold and resource system functions are as follows and illustrated in Figure 4-12:

- For every bio-dome constructed three food units are added to the player's resources. The system tracks the numbers of fuzz units constructed and also determines if enough food points are present to create new fuzz units.
- For every market place constructed, a gold piece is added to the player's money every 100 game tics.
- For every gold mine, three gold pieces are added to the players money every 100 game tics.

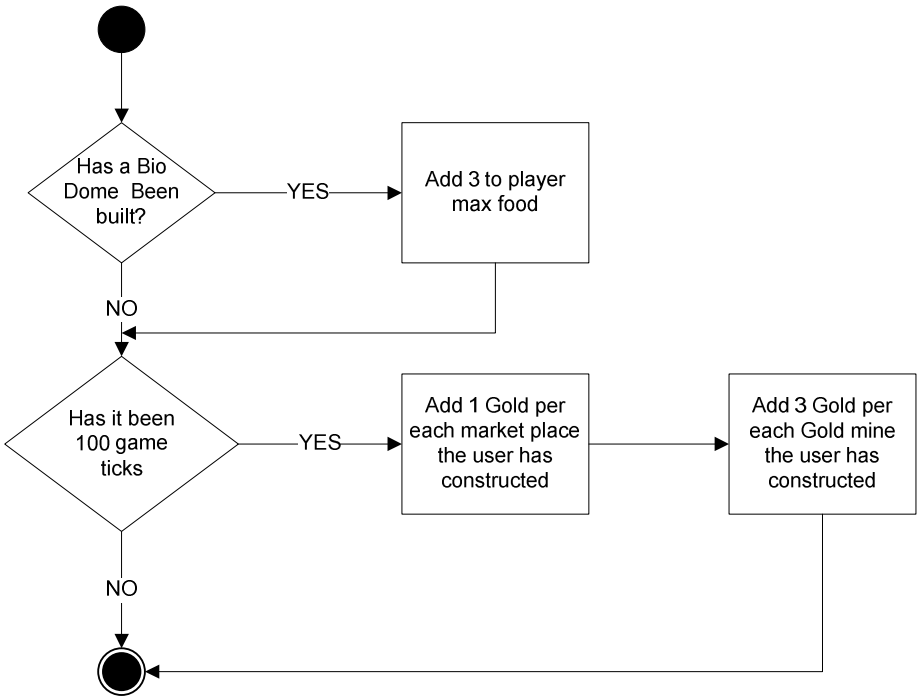

Figure 4-12: Resource Management Process

The enemy manager component, Figure 4-13, keeps the enemy units moving continuously through the world. If the enemies collide with an object, they will proceed in a new random direction. When they come in range of a fuzz unit they will stop and fire. They will only begin moving after the fuzz unit is destroyed or moves out of range.

This module also spawns new enemy units when a certain number has been destroyed. Some of the newly created enemy units are weaker than veteran units allowing for novice fuzz units to test their mettle and gain experience points.

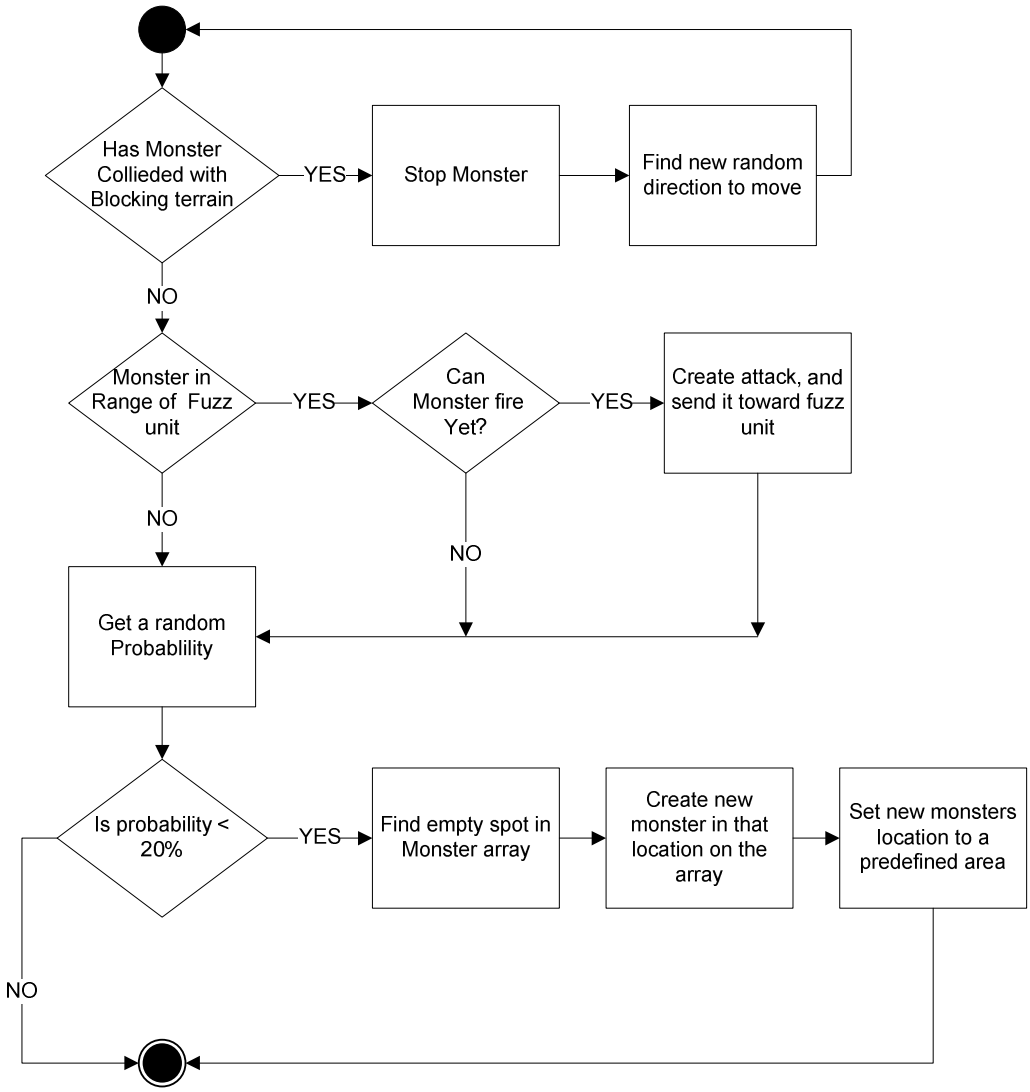

Figure 4-13: Enemy Unit Process Manager

As the flow charts demonstrate, even with simple sub-modules used, such as the attack and move, the game was still quite complex. The process of outlining the code in flow chart form allows the programmers to see what variables, functions, and possibly objects they will need in order to complete the task. Flow charting is also a good way of finding potential flaws or bugs in the code before the code is actually entered.

5 Chapter 5: Fuzz's Revenge Illustrated

This chapter provides screen shots from the pilot game with accompanying explanations which describe the results of the project. These screen shots demonstrate both real time and role playing strategies that have been incorporated into the game.

5.1 Units

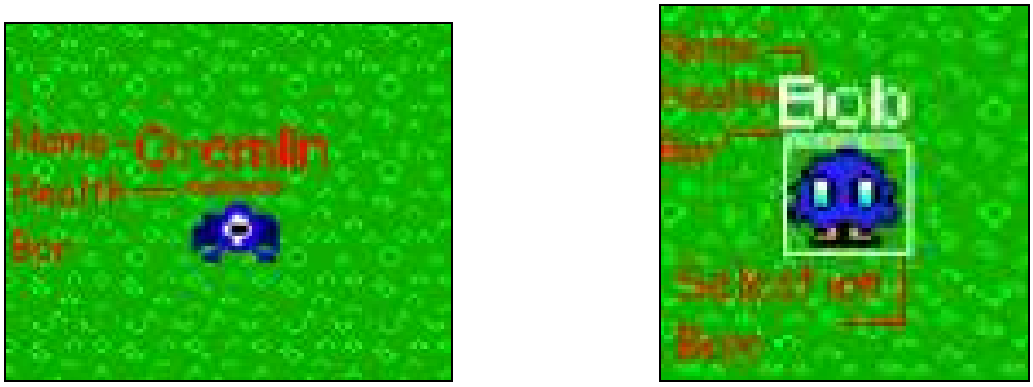

Figure 5-1: The Enemy Unit (Left) The Friendly unit (right).

Figure 5-1 shows the standard display of an enemy unit including name, health bar, and sprite (image). Figure 5-1 shows the friendly unit surrounded by the white selection box and includes its name, health bar, and appropriate class sprite.

5.2 Statistic Window

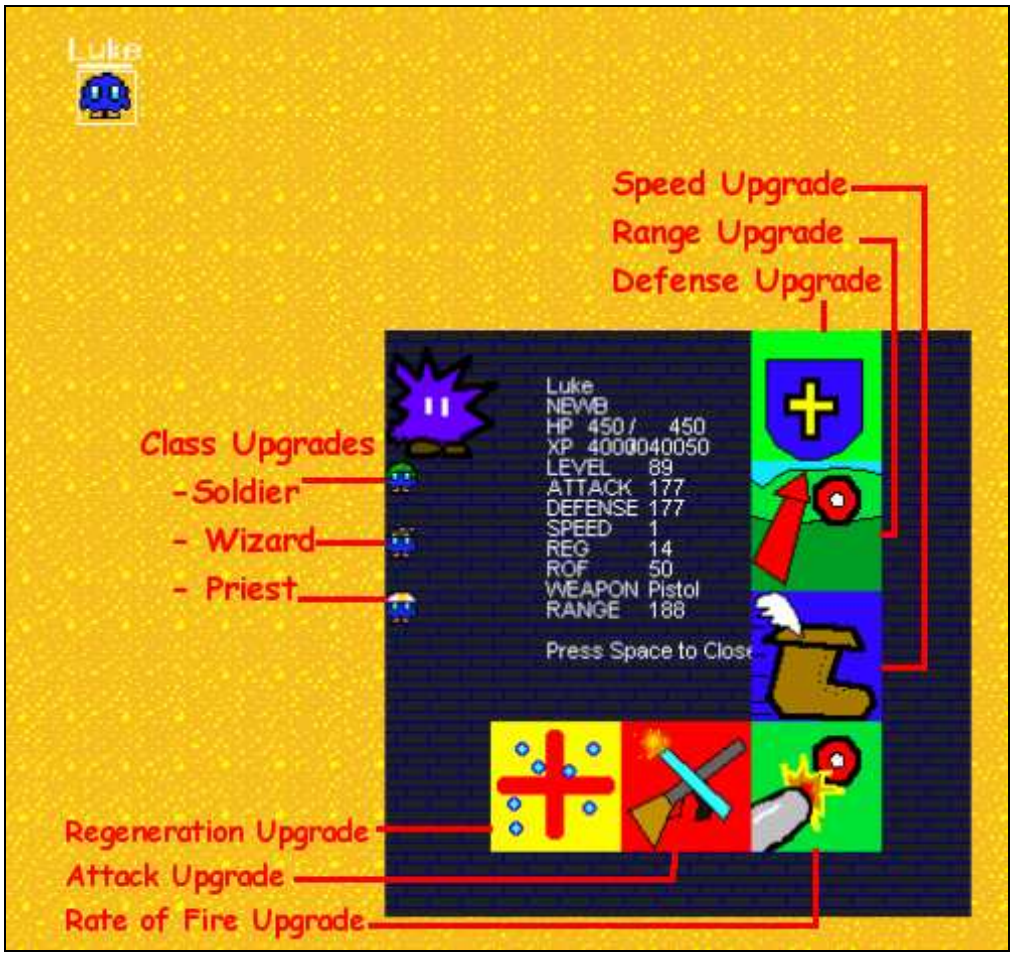

Figure 5-2: The Unit's Statistics Window

Figure 5-2 shows the status box that is associated with each unit. Once a unit is selected and the space bar is pressed, the status box appears showing the attributes of the selected unit. In this example, 40,000 experience points were given to the unit, placing it at level 89. Because the fuzz unit's class is a “NEWB” it can become a solder, wizard, or priest because its level is higher than 5, 10, and 15 respectively. Also note that the player can individually customize six attributes for each unit depending on the unit's skill level. However, certain upgrades have limitations. For example, speed can only be upgraded once, to a rating of 2, and rate of fire can only be upgraded three times, which lowers the delay that is required to attack again to 20 cycles.

5.3 Weapon Building

Figure 5-3 gives one example of building a weapon-note the selected building (white square) and the option tool bar to build weapons. In this example, the magic shop can build three powerful weapons: a magic hat that is used by wizards, fire sorcerers, and ice mages, a fire wand that is used by fire sorcerers, and a frost stave that can only be used by ice mages.

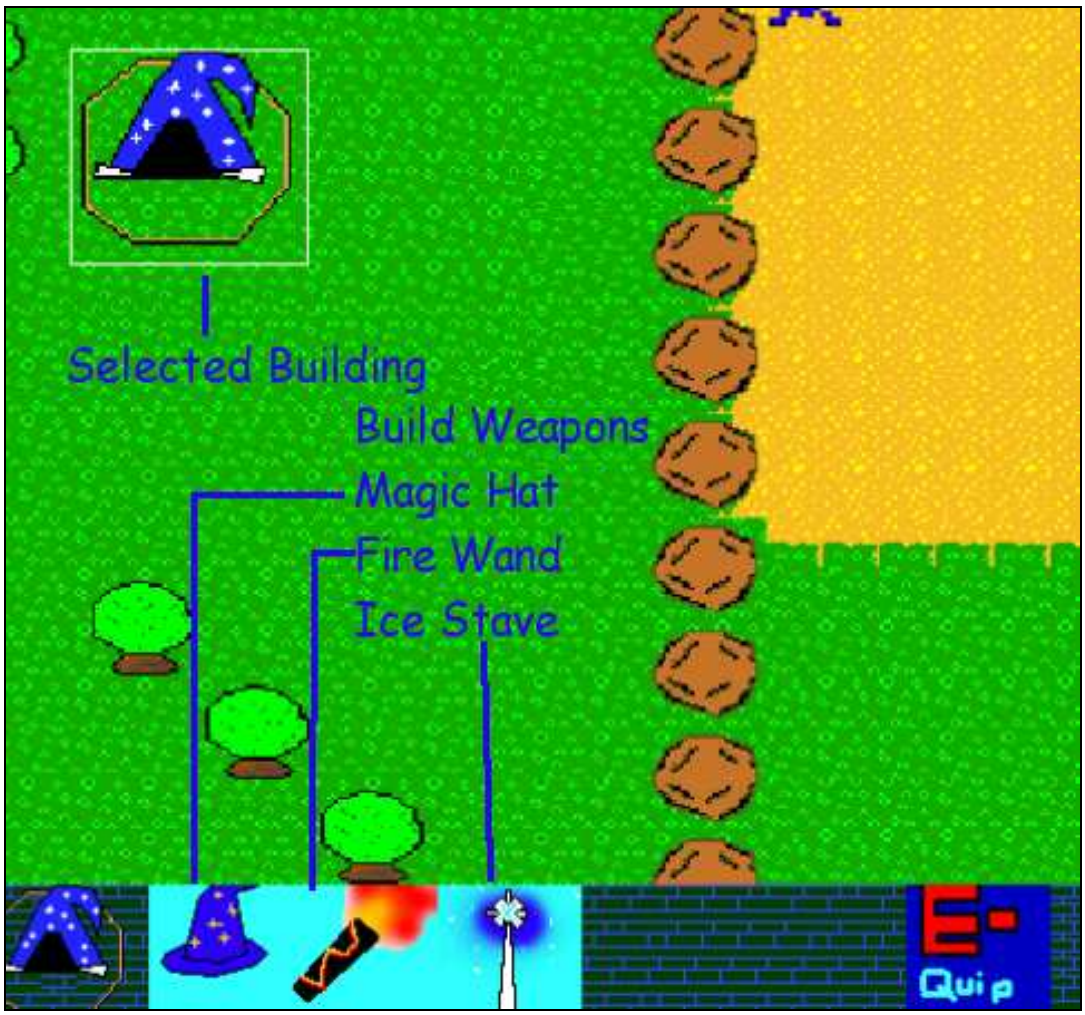

Figure 5-3: Available Weapons in the Magic Shop

5.4 Weapon Equipping

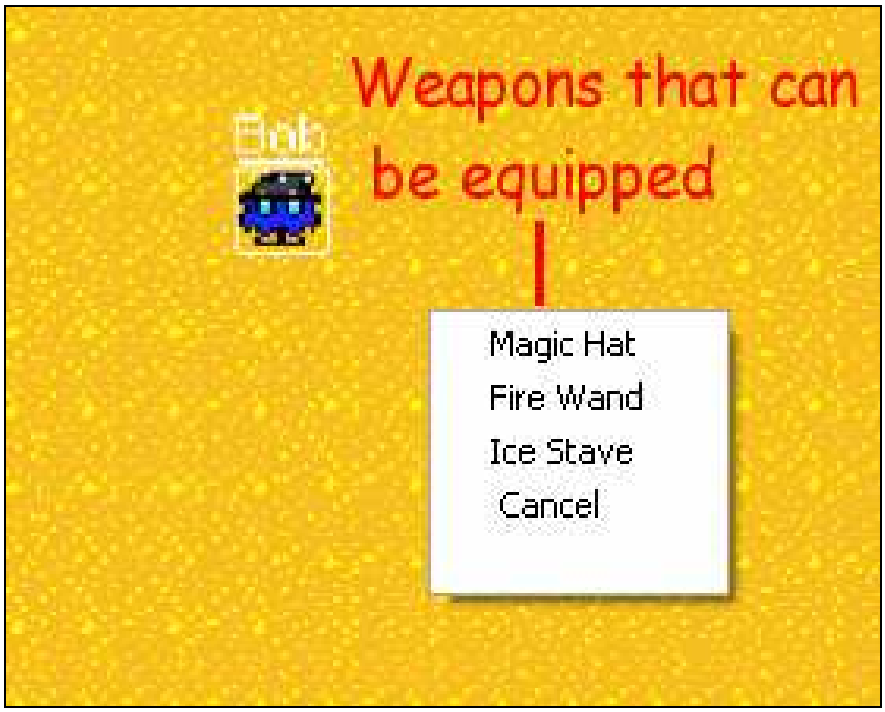

Figure 5-4:Weapon Selection Menu

Figure 5-4 shows a selected fuzz unit (wizard class) and the equipment menu for this unit that appears when the middle mouse button is clicked. Only wizards, fire sorcerers and ice mages can use the magic hat weapon. Although all weapons are visible in the menu, only fire sorcerers can use a fire wand and only ice mages can use an ice stave. The system will not allow a wrong weapon to be selected for a unit.

Once the fuzz unit reaches the appropriate class the player can trade the current weapon used by the unit to a more powerful one. Figure 5-5 shows a fuzz unit that has been elevated to a wizard class with the magic hat.

Figure 5-5: Equipping the Magic Hat

Figure 5-5 shows the two secondary classes available to the wizard: the fire sorcerer and the ice mage. On the condition that the fuzz unit is a wizard class, it can be upgraded to fire sorcerer or ice mage, providing it has reached level 15 or 20 respectively.

5.5 Group System

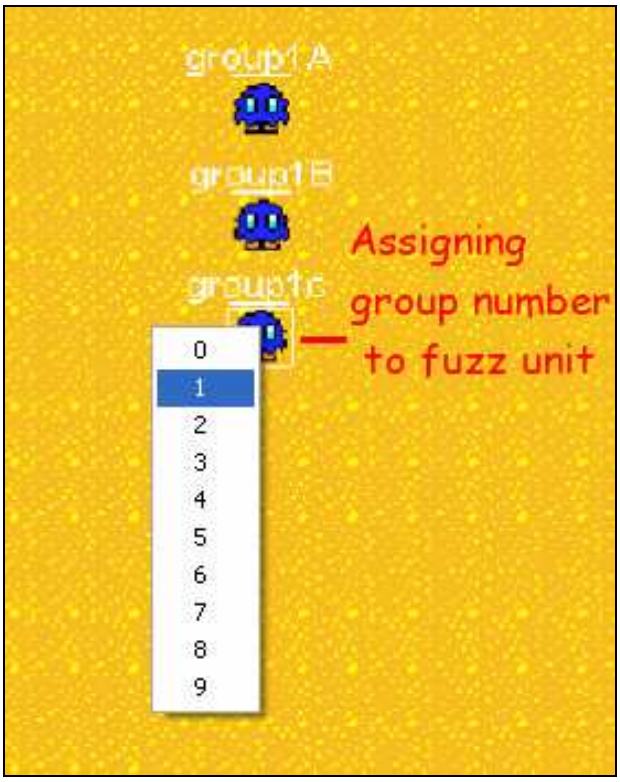

Figure 5-6: Assigning a Unit to a Group

In order to regain the *army* feel that is found in RTS strategy games, group movement was introduced into the pilot game. Figure 5-6 illustrates the procedure for group assignment. Once the user selects the appropriate fuzz unit and then presses the control key, a menu with the list of existing group numbers will appear. The user then clicks the appropriate group number with the mouse. The group number is stored in the unit's group variable, and when the appropriate group number key is pressed, units assigned to that group will be highlighted in a blue box (Figure 5-7).

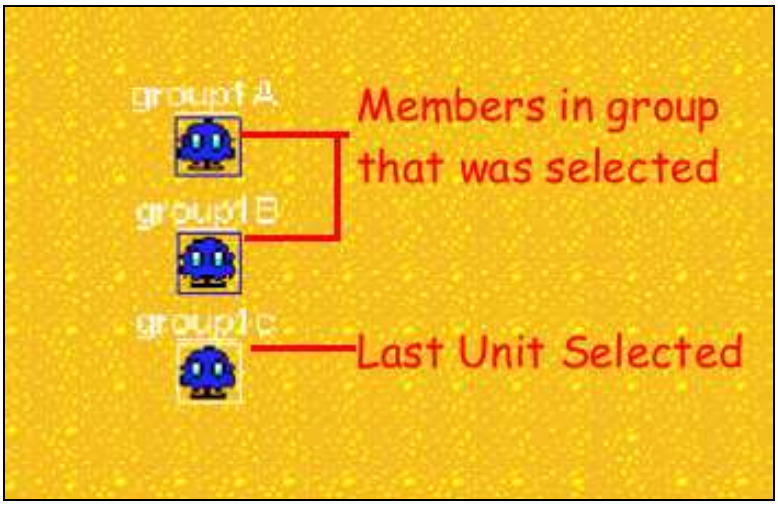

Figure 5-7: A Selected Group

There is still only one fuzz unit stored in the global "selected" variable, highlighted in white. Therefore, the status window will still work, even if a group is selected. If the user selects another unit or group, the current group will be de-selected automatically. Once a unit or group is selected, the player can right click the mouse on any location in the world and the group or unit will move to that particular location unless they collide with blocking terrain.

Figure 5-8: The Opening Screen

The opening screen provides users with money and food counters. The "E-quip" button allows players without a middle mouse button to equip units with weapons (Figure 5-8). All, but one, of the buildings in the tool bar are scaffolds. Therefore, the home base must be built first. Once the home base is constructed, other buildings will become available (Figure 5-9).

Figure 5-9: After construction of Home Base occurs new buildings are available

The final, novel component in this RTS game is the dungeon screen which opens and closes when a fuzz unit comes in contact with a dungeon (Figure 5-10).

Figure 5-10: Fuzz unit entering Dungeon

Once the fuzz unit comes in contact with the dungeon (Figure 5-10), it will be transported into another area of the map and a secondary window will open (Figure 5-11). Note the white, door-shaped image: this is the exit. When a fuzz unit collides with the door, the fuzz unit leaves the dungeon.

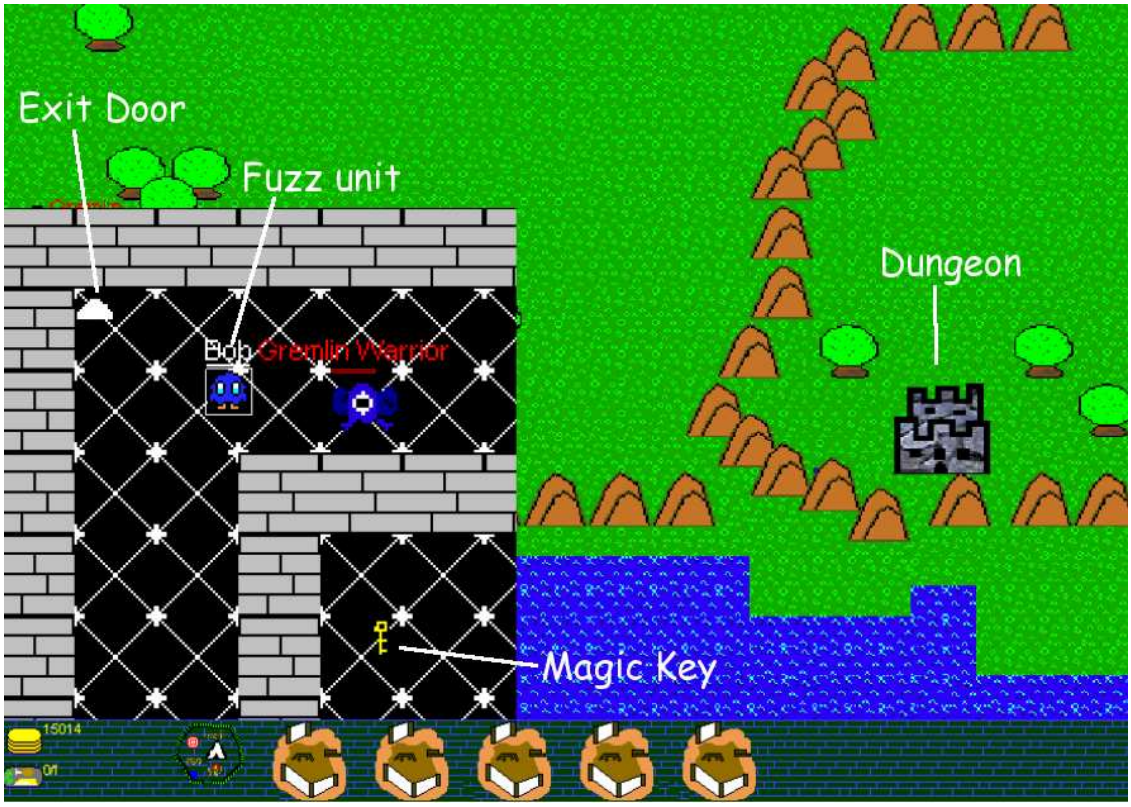

Figure 5-11:Fuzz unit inside a Dungeon

Recalling chapter two, the pilot game required the user to get three keys. Figure 5-11 shows the location of the key in the first dungeon. Two more keys are located in two other dungeons found in the world.

This chapter has covered the major aspects of the pilot game "Fuzz's Revenge". Although difficult to demonstrate on paper, the screen shots display an RTS setting where the player controls multiple units, constructs buildings, builds weapons to equip to fuzz units and can move units in groups. Along with these RTS factors this chapter also demonstrated various RPG concepts such as: leveling, customization, weapon equipping, and "dungeon crawling".

6 Chapter 6: Future Work

This chapter discusses further research possibilities for the concept of RPG/RTS combination. Using the ideas discussed in the previous chapters as a spring board, various new options become available. This chapter discusses some of the options which have the largest potential.

6.1 Future Work

Although the pilot game presented in this thesis contains the desired functionality, there are still more areas in which the game could be improved. These areas fall into three categories:

- Multiplayer
- New units
- Balancing (Game mechanics)

In addition, further research and development can be done to investigate new combinations of genre characteristics as well as improved approaches and notations (in particular, UML) in game development.

6.1.1 Multiplayer

Preliminary programming was done to allow two players to compete against each other on the same computer (using the same keyboard) in **Fuzz's Revenge**. Those who played the game found it entertaining and developed an interest in individual characters. Therefore, the players were much more careful in attempting to keep their characters alive. Upgrading unit attributes provided a sense of accomplishment and induced players to set goals for leveling up, which is what drives RPG game participation. Therefore, incorporating LAN capabilities and allowing for multiple players to compete on individual computers would make it much more marketable. This feature was not a scope in the original problem; however, Game Maker does allow for networking. This would be a logical next step for this research.

This game could be further developed by applying massive multiplayer online (MMO) capabilities. With support for MMO, a player can log out of the game and log back in to find the server still running; this is called a "persistent world". To accomplish this, several design problems would need to be overcome such as home base design and the problem of an individual building a base in the same location as another who is logged off the system at the time. Never-the-less, by allowing characters to advance in rank, a game such as this would allow for many more combinations and increase game longevity, which in turn would help maintain the player's interest.

6.1.2 New Units

This pilot game could also be further developed to include multiple types of characters. The pilot game has one type of unit with different classes, but other units could be introduced with their own class system. These new units could develop and change within their own class structure. Thus, capabilities and power would also vary between character types as well as within unit classes. RTS games depend largely on the "rock, paper, and scissor" construct, where different types of units are weak against other units. Developing several types of characters with their own class system would introduce this concept and allow for more strategic variation in the game play.

6.1.3 Balancing

Further work could also be done to assure unit balance and allow equal chances to win. Now that individual units can increase in strength a new dimension has been added to the balancing equation. Therefore, if game companies implement the five rules described in this project, balancing will become much more difficult. For example, if one player has an army of humans and he is playing against an army of orcs, each with their own class structure, then the power and unit cost of these two different characters must be balanced to assure equal chance of winning. After playing this game and seeing the beneficial effects that unit development had on **Warcraft III** (see Chapter 1), the pros outweigh the extra time needed in balancing. In the case of the pilot game, the value of unit development is exaggerated and the loss of high-level units results in game loss. In a fully developed game a player should be able to recover from unit loss. The development

of a generic algorithm to evaluate the degree of balance between a unit and its cost versus a different unit and its cost would be very beneficial to the game industry and expedite this balancing process.

Hopefully one of these ideas proposed in this chapter will be used to further develop the RTS genre. Once the player is allowed to create units to their specifications, the options for RTS games are expanded greatly.

6.1.4 New Combinations of Genre Characteristics

A promising direction of related research and development is offered by the possibility of combining new genre characteristics to develop hybrid games and combine paradigms. For example, future work could explore extending the MMO (Massively Multi-Player) paradigm to incorporate RTS type of games.

6.1.5 Improved Software Engineering Practices and Notations for Game Development

Connected with one of this thesis' main contributions, that of applying software engineering to game development, new research could be performed to define enhanced processes and specific modeling notations customized for the realm of game construction. For example, a specialized UML-based notation could be developed for game specifications, design and deployment.

7 Chapter 7: Conclusion

This thesis has presented an approach of combining four necessary components of an RPG, *equipment*, *leveling*, *customization*, and *classification*, into the realm of RTS games by obeying the following five rules:

1. Every unit should have some degree of customization.
2. Every unit should have the ability to increase in level or rank.
3. Unit capability should be limited in skills and rank.
4. Every unit should have an equipment and statistics panel.
5. Increasing the strength of a character should be much more valuable then building a more expensive unit. However, loosing your most powerful character should not end the player's chances of winning.

With these new constructs the Real Time Strategy genre can expand in new dimensions and possibilities for the players. This will increase the number of strategies used and bring the player's own creativity into the game. In addition, the characters will have more worth and the player will have a greater need to preserve the units instead of merely sending them off to battle without a thought.

Another contribution of this thesis was the novel approach to applying more extensively software engineering aspects to game development. Building and evaluating a prototype greatly assisted with the innovative process of game design. Once the game requirements (the five rules) were established and tested, then the well-defined model elements of UML were used to specify the program, make it more efficient, and thoroughly check for errors. This process of game design left room for innovation and maintained design standards through UML. The aforementioned steps used for the prototype design are as follows:

- Program a prototype with the desired testing features.
- Evaluate the prototype and design a final list of specifications.
- Use UML to design the final software package.
- Write the final software.

These steps would allow the computer game industry to make their design process more efficient and still easily add or change game mechanics. Also, based on the modeling power of UML, the design team could detect possible flaws or errors in game algorithms before the code was written, thus reducing the time and money needed for correcting mistakes.

Computer games have become compartmentalized into their specific types (genres) and a good degree of repetition can be found from one game to another. A good example of this was the similarity between **Dawn of War** and **Company of Heroes**: the setting was different, the graphics were improved, however the resource management and game play were almost the same. This thesis focused on a new method for computer game design in general and illustrated it on a particular prototype game developed, “The Fuzz’s Revenge”. The thesis performed an analysis of game patterns (genres) and explored the combination of RPG and RTS game mechanics as a means to enhance the entertainment value of video games. In addition to just improving graphics, this approach could assist game manufacturers to create novel game play mechanics that would hold consumer interest for longer periods of time.

8 Reference

[1] King, Brad., John Borland. Dungeon and Dreamers The Rise of Computer Game Culture from Geek to Chic. Emmeryville: McGraw-Hill, 2003.

[2] Crawford, Chris. Compiled by Prof. Sue Peabody. The Art of Computer Game Design. Vancouver: Washington State University, 1997.

[3] Paul Gee, James. What Video Games Have to Teach Us About Learning and Literacy. New York: Palgrave Macmillan, 2003.

[4] Wirman, Hanna., Rika Nakamura. "Counter-Playing Tactics of Female Players." The International Journal of Computer Game Research, Volume 6, Issue 1 (2006).

[5] Ermi, Laura., Frans Mayra. "Player-Centered Game Design: Experience in Using Scenario Study to Inform Mobile Game Design." The International Journal of Computer Game Research, Volume 5, Issue 1 (2005).

[6] Nintendo Wii Zone. "Wii Development Kits to Cost $1,700." 12-17-06. http://www.nwiizone.com/nintendo-wii/nwii/wii-development-kit-to-cost-1700/

[7] Kolo, Castulus., Timo Baur. "Living a Virtual Life: Social Dynamics of Online Gaming." The Journal of Computer Game Research, Volume 4, Issue 1 (2004).

[8] Arlow, Jim. Ila Neustadt. UML and the Unified Process. London, Addison-Wesley.

[9] Imagine Publishing, "Why you Must Play Herzog Zwei." The Essential Guide to Classic Games, Retro Gamer, Volume 28, Pages 34-37.

[10] Habgood, Jacob. Mark Overmars. The Game Maker's Apprentice, China, Apress (2006).

[11] GameSpot. "Microsoft Raises Estimated First-Day Halo 2 Sales to $125 Million-Plus". http://www.gamespot.com/news/2004/11/10/news_6112915.html Retrieved on 2006-03-15.

[12] Salen, Katie. Eric Zimmerman. Rules of Play, Game Design Fundamentals. Cambridge, MS: MIT Press 2004.

[13] Khazan, Olga. “Lost in an Online Fantasy World”, Washington Post, 8-18-2006. http://www.washingtonpost.com/wpdyn/content/article/2006/08/17/AR2006081700625_pf.html, Retrieved on 2006-5-25.

[14] Tweet, Jonathan. Monte Cook. Skip Williams. Dungeons and Dragons Players Handbook, Wizards of the Coast (2003).

[15] D&D Home “What is D&D”. http://www.wizards.com/default.asp?x=dnd/whatisdnd, Retrieved on 2006-5-25.

[16] Board Game Central. “The Game of Risk”. http://boardgamecentral.com/games/risk.html, Retrieved on 2006-5-25.

[17] E. Ryan, Michael. “Star Wars Rebellion”. http://www.gamespot.com/features/rebellion/, retrieved on 2006-5-25.

[18] Game FAQS. “Game Company Information”. http://www.gamefaqs.com/features/company/844.html Retrieved on 2006-5-25.

[19] Game FAQS. “Game Company Information”. http://www.gamefaqs.com/features/company/72467.html.Retrieved on 2006-5-25.

[20] Wikipedia. “Command and Conquer”. http://en.wikipedia.org/wiki/Command_and_Conquer Retrieved on 2006-5-25. Retrieved on 2006-5-25.

[21] Wikipedia. “Warcraft, Orcs and Humans”. http://en.wikipedia.org/wiki/Warcraft Retrieved on 2006-5-25.