Towards Integrating Role Playing Game Constructs in Real-Time Strategy Games

Brad Towle **Monica Nicolescu** **Sergiu Dascalu**

University of Nevada, Reno, USA
{towle, monica, dascalus}@cse.unr.edu

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: games are played in similar ways. 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. With prospects for making a successful game rewarding a company with perhaps millions of dollars more games are now flooding the shelves and quantity has become more important than quality [2][6]. This paper reviews our work on improving real time strategy (RTSs) games by incorporating aspects from role playing games (RPGs)[1]. In this case, the four major components from the role playing game are: character equipment, character advancement, character customization, and character classification. In addition, this paper describes a successful application of the unified modeling language (UML) to game design. By demonstrating game design through UML, programmers could see potential problems with game play before the entire code was written. This paper 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.

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, surpassing the opening movies that week) [6]. 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 similar mechanics on which the games are built. It would be unfair to say that Real Time Strategy (RTS) games have not evolved, modern RTS games are much more advanced than the first real time strategy game called Herzog Zwei [4]. For simple definition an RTS game allows a player to command a vast army, set up a base, and upgrade units (in a general sense). Although they have some story this is not the main focus of their purpose [7]. This genre evolved from the turn based tactics game which players would take turns moving units to try to gain a strategic balance. A good example would be the board game RISK [11]. This being said there are several reasons that RTS games remain similar:

1. The re-use of code: game engines are usually difficult to write from scratch and therefore, if a company can use the same game engine for multiple games it will save money.
2. 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 proficient in new games as they are produced.
3. 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** [12] in which there were over 100 different subsystems the player needed to control. Therefore, game developers, worried about over-complex game play, continued producing the games based on standard "massive horde" RTS mechanics.
4. 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. This paper proposes to combine five rules of designing RTS games with four concepts taken from the RPGs in order to create a novel and more enjoyable game than the standard games currently being developed. Role playing game generally focus on a small group of people, or a single person, who must progress through a story gaining strength as it progresses. The following section outlines the four components that are necessary

for a role playing game. This genre is derived from the table top versions such as Dungeons and Dragons [9][10]. These were among the first games to be ported over to the earliest computer mainframes [1].

2. Method

The purpose of our project was to combine four components of an RPG into the RTS game play to allow character customization. The four *RPG constructs* are the following:

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

In order to accomplish combining the four components of a role playing game into an RTS game *five rules* were developed:

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.

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 work 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. 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 - 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 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. 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. However a player loosing all his/her advanced units should not limit their chances to victory if they still have an army to command.

3. UML for Game Specification

This section documents an abridged form of the Unified Modeling Language (UML) [3] artifacts that were used to help creating the pilot game. The dynamic nature of computer game design forced us to change the typical UML-based modeling process into a simpler form containing only requirements, use cases, and primary scenarios. These elements were used to define what the game needed and what components were in it. After this was established only a high level diagram and basic flow charts of the modules were necessary in order to organize the design while allowing for the dynamic nature of the game industry.

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, 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 UML. The UML documentation was paramount to ensure that the five final rules were incorporated in the final iteration of the game code.

Table 1 depicts an example of the functional requirements of the game we created, "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 an 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 1: Example of Requirements for the "Fuzz's Revenge" Pilot Game

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.

3.1 Use Case Modeling

Use case modeling is a technique for capturing functional requirements of a system or systems via use cases (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 2) are a sample of what was necessary for the pilot game and 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 2: Sample Use Cases for the Pilot Game

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

Now that the requirements of the game have been defined and the use cases have been derived to meet the requirements the game designer can see what needs to be done or problem areas in the game. At this point it is necessary to figure out which systems are controlled by which actors. Figure 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.

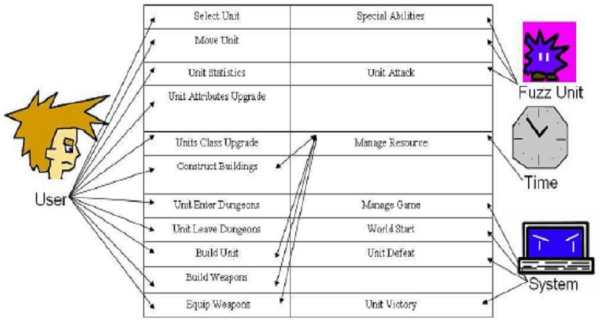

Figure 1: Breakdown of Actors and Use Cases in "Fuzz's Revenge"

3.2 Primary Scenarios

In computing, a scenario is a narrative describing foreseeable interactions between the users (actors) and the software system, or between two software components [13]. This allows the game designer to determine what the game play is going to be like. This was a great help in determining problems such as multiple functions requiring the same key or mouse press, situations that would be boring for the user, and situations that may have been overlooked. This allows the programmer to perceive the entire game as if it was completed thus reducing time because expensive functional prototypes did not have to be developed. Due to the large amount of possible scenarios only a few of the prominent scenarios are detailed in the following scenario tables. For a more exhaustive list of scenarios please refer to [14].

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).

Table 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.

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 4).

Table 4: 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.

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 5).

Table 5: 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:	

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 6).

Table 6: Defeating the Enemy: Unit Victory

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.

After listing out the requirements and the use cases the program could then be designed. Using high level flow charts to describe the systems that went into the game it became apparent that this game was a much larger undertaking than first thought. However with careful planning from the UML requirements when the game was finished there were no unexpected problems and the game played exactly as anticipated. This proves the need to make sure you spend time listing out the requirements and basic scenarios for any game design.

4 High Level Design of Fuzz's Revenge

Typically, most UML designs will start at a high level of abstraction and then continue to get more and more detailed and precise. The problem here is that game design requires more freedom. If a game designer had to re-write every flow chart, every class diagram, every sub-system graph every time a change was made it would decrease the efficiency of the game design. Once again only the high-level design diagram and basic flow charts were used. Several problems arose when we tried to generalize game design at lower levels of abstraction. First, different games have different engines and not all game engines fit nicely into a certain generic sense. Second, the development platform for Fuzz's Revenge did not fit exactly into regular UML class structures, thus trying to write out

class diagrams was difficult. Figure 2 details the high level design diagram for Fuzz's Revenge.

Once this was established the game designer could use simple flow charts to design each module. After this coding could then take place. After following this procedure the game turned out exactly as was expected. Through this project, we were able to prove that it is possible to use UML to aid game design especially when a new style game, such as the one we were attempting to make, is being designed. This could prove very useful to game companies so to minimize the risk of designing new games.

4. The Game

In order for this research to be considered successful a pilot game had to be created. We created Fuzz's Revenge, a game in which the player commands a group of blue fur balls that can learn different skills. Each fuzz can gain levels after reaching a predefined threshold for each level. Each level gives the fuzz one skill point in which the player can upgrade speed, rate of fire, attack, defense, range and regeneration. Along with this if the fuzz reaches a certain level, the user can change the class (job) of the fuzz unit. At level 5 the fuzz unit can become a soldier. At level 10 the fuzz can become a wizard and at level 15 the fuzz can become a priest. The wizard also has two subclasses, fire sorcerer and frost mage, which can be upgraded at levels 15 and 20 respectively.

The goal of the game is to capture 4 keys, which involves facing progressively more difficult enemies. Along the way the user can change their own unit's: abilities, classes, and even equipment. Along with this, the game follows the five basic rules outlined in the previous section. This game was implemented on Game Maker [5] development platform. It provided enough functionality to allow for proof of concept.

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

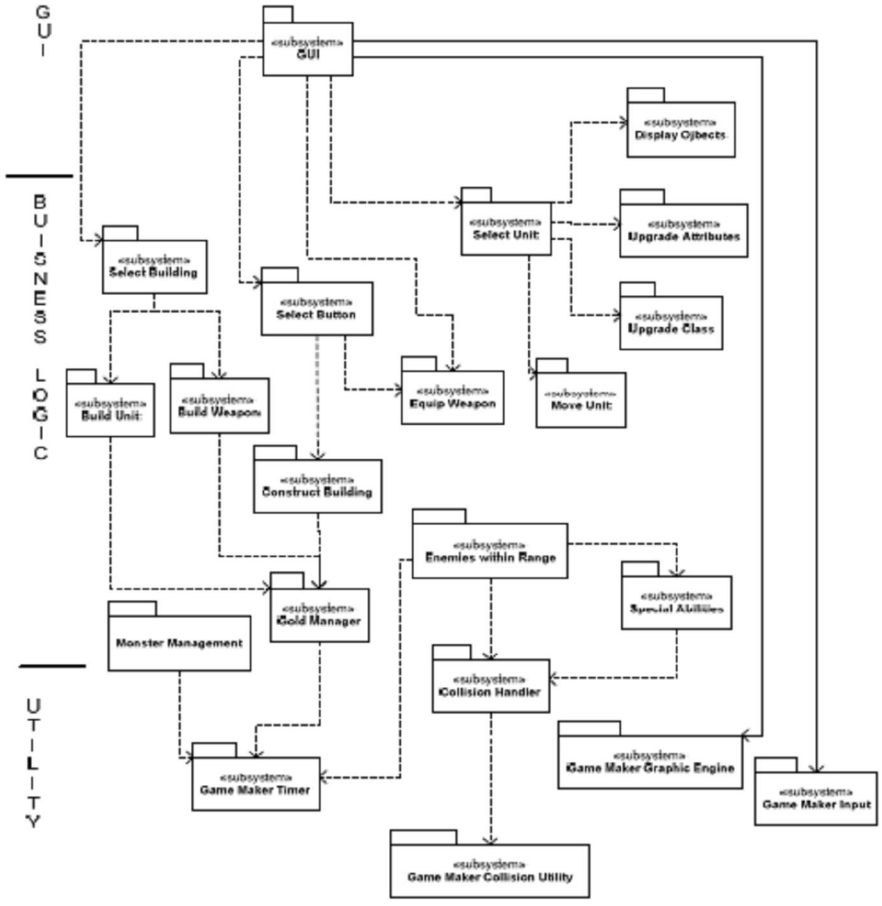

Figure 2: High Level Diagram for Fuzz's Revenge

4.1 Units

Figure 3 (top) shows the standard display of an enemy unit including name, health bar, and sprite (image). Figure 3 (bottom) shows the friendly unit surrounded by the white selection box and includes its name, health bar, and appropriate class sprite.

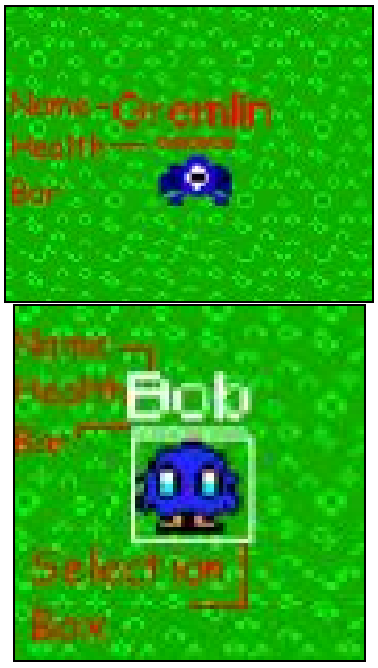

Figure 3: Enemy Unit (top); Friendly Unit (bottom)

4.2 Statistics Window

Figure 4 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", essentially a novice unit, 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 all 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 game cycles.

4.3 Weapon Building

Figure 5 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.

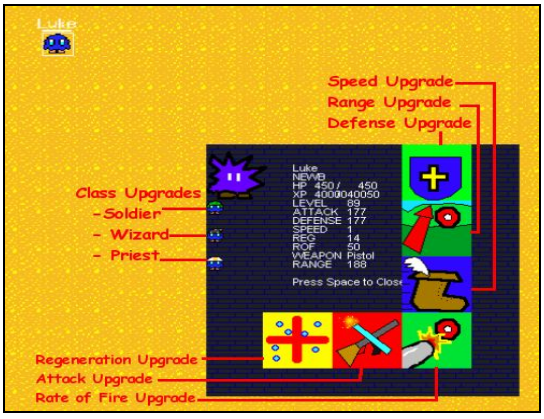

Figure 4: The Unit's Statistics Window

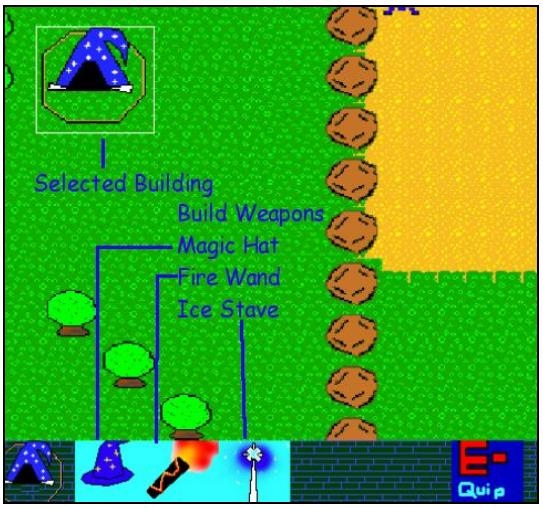

Figure 5: Available Weapons in the Magic Shop

4.4 Weapon Equipping

Figure 6 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 7 shows a fuzz unit that has been elevated to a wizard class with the magic hat.

Figure 8 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.

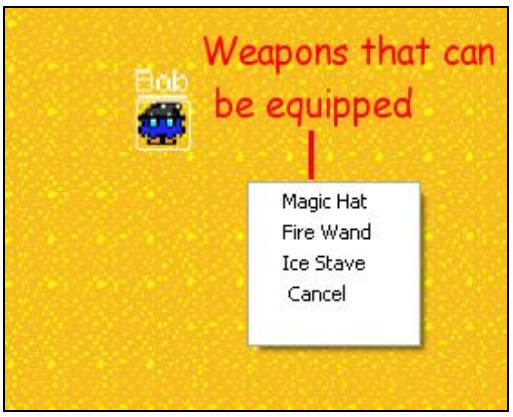

Figure 6: Weapon Selection Menu

Figure 7: Equipping the Magic Hat

4.5 Base Construction

Figure 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 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 9). Once the fuzz unit comes in contact with the dungeon (Figure 10), it will be transported into another area of the map and a secondary window will open (Figure 11).

4.6 Dungeon Exploration

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 10).

Figure 9: After Construction of Home Base Occurs New Buildings Are Available

Figure 10: Fuzz Unit Entering Dungeon

Note the white, door-shaped image: this is the exit. When a fuzz unit collides with the door, the fuzz unit leaves the dungeon. The pilot game required the user to get three keys. Figure 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 paper has covered the major aspects of designing the pilot game "Fuzz's Revenge". The screenshots display an RTS setting where the player controls 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".

5. Future Work

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 in the scope of the original problem. However this would be a logical next step for research and development.

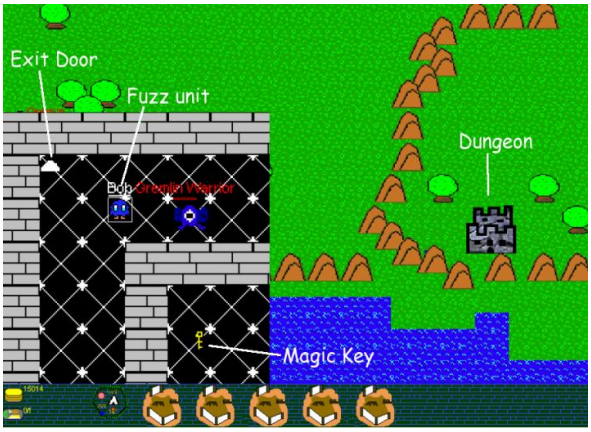

Figure 11: Fuzz Unit Inside a Dungeon

Connected with one of this projects main goals, 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.

6. Conclusion

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 similar. This paper 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 research 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 through the method of combining aspects of different game genres [8].

References

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

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

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

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

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

[8] Khazan, Olga. "Lost in an Online Fantasy World", Washington Post, 8-18-2006. http://www.washingtonpost.com/wpdyn/content/article/2006/08/17/AR20060817 00625_pf.html,

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

[10] D&D Home "What is D&D" http://www.wizards.com/default.asp?x=dnd/whatisdnd

[11] Board Game Central. "The Game of Risk". http://boardgamecentral.com/games/risk.html,

[12] E. Ryan, Michael. "Star Wars Rebellion". http://www.gamespot.com/features/rebellion/

13] Fowler, Martin. UML Distilled Third Edition. Boston. Addison-Wesley. 2006.

[14] Towle, Bradford. Thesis: Combining Role Playing Game Constructs Toward Real Time Strategy Games. Reno. University of Nevada, Reno. 2007. Available at: http://www.cse.unr.edu/~towle/thesis.pdf