Senior Project Writeup

Written By: J. Reece Engle

Advised By: Professor Z. Wood

Additional Group Members:

Alan Delonga

Evan Kleist

Jordan Gacsh

I. Introduction

With the recent boom in the video game industry, the design and development of games has become an extremely competitive and sought-after job market for software developers. In 2009 the video game industry generated over 19.6 billion dollars, this overwhelming revenue surpassed both music and movies.

Video games immerse the player into visually complex and captivating worlds that are represented using complex algorithms. This is done by creating realistic effects that model the real world such as dynamic shadows, detailed shading, and vivid particle effects. Additionally, utilizing lifelike models with fluid animations in true to life environments helps a player feel oriented in this representation of a virtual world.

Despite the leaps and bounds in hardware performance representing all these realistic effects mentioned above is computationally expensive. This makes conventional programming design specifications, structured iterative development processes, and constant testing critical aspects of video game development. Without concern for efficiency these complex effects will easily hurt performance and has the ability to make a graphically beautiful game completely unplayable.

Dr. Wood's '476++' course presented a unique opportunity for students to take a 'head-first dive' into the world of game development, while learning advanced graphics techniques, and completing the senior project requirement all in 20 weeks. Personally, I have always been very fascinated by video games. I could argue that my interest in video games may have been most influential factor in my choice in going into computer science at Cal Poly. Therefore, I chose to implement, *Zombs*, a real-time 3D video game with classmates Evan Kleist, Alan Delonga and Jordan Gasch.

The *Zombs* project was an appealing choice for our group with the present popularity of video games and the demand the presence of many essential elements of Computer Science such as design, teamwork, and implementation. Zombies have been increasingly appearing in all forms of popular culture. Regularly depicted in horror and fantasy based entertainment, Zombies have captured the interest of millions of people plus our group of four.

Software projects have grown in size significantly and as a result applications are rarely worked on individually in practice. This makes being productive in groups a critical skill exercised regularly while working in industry as a software developer. With our ideas inspired by popular video games and other forms entertainment as a reference for the look and feel we wanted for our game, we gave ourselves a lot of work from the start. The overwhelming workload made task prioritization and assignment an essential dynamic that continually evolved over the 20 week implementation to suit the needs of a particular milestone.

II. Project Overview

The *Zombs* project was initially conceived as a proposal in our CPE476++ class, of a game that would be a mix between bomber-man and Diablo II, using cell shading. As the project progressed we chose to follow a more ominous ambiance with our models, lighting, camera angle and background sounds. By having a democratic atmosphere, where all members gave input, our project became a cohesive culmination of our ideas. Our team was initially 5 members but due to the inabilities of one of the members we only had 4 for the second phase of implementation. Our members are

Alan DeLonga, Evan Kleist, Jordan Gasch, and Reece Engle. As we started the project we were given a list of technologies that were mandatory to incorporate into the game.

By the end of development we were able to complete all of the following technologies and fully integrated them into our game:

Real-time movement/update (All)	Level Editor (Reece Engle)
View Frustum Culling(Evan K., Reece E.)	Level Import/Export Functionality (Reece E.)
Particle generation (Alan DeLonga)	HUD (Alan D., Reece E., Evan K.)
Spatial data structures (Evan Kliest)	Inventory (Alan D., Reece E.)
Per pixel shading (Jordan Gasch)	Per pixel shading (Jordan Gasch)
Spatial data structures (Evan K.)	Collision detection (Evan K.)
Wall Transparency (Evan K.)	Sounds (Alan DeLonga, Evan K.)
AI (Jordan Gasch, Reece E.)	Shadows (Jordan Gasch)
Models (Alan DeLonga)	Animations (Alan DeLonga)
Smart Camera (Evan K.)	

Along with these technologies we had to create a cohesive gaming experience. Since each member incorporated different aspects I will only go into detail on the parts I worked on.

Figure 2.1 : *Zombs* Video Game

The game was coded the project using C++ utilizing the following libraries: SDL, libSDL_mixer (for sounds), libSDL_ttf (for text), libfreetype, libGLEW(for models, and shading). Our game idea was originally inspired by Diablo II's camera view and model interaction. After receiving negative player feedback about our camera's view we made the decision to alter it by locking the camera behind the player, dropping the angle, and pulling the camera in (instead of above) when the camera collides with walls.

A. Story

The basic idea behind the story is you are a survivor in a zombie apocalypse. With an 'all-to-cliché' start in the zombie genre, your character wakes up in a hospital and the last thing you remember is being in a car accident. Your main objective being to find your wife who was also in the car. Your walking is labored and you can only sprint for short distances. As you progress through the level you realize something has gone horribly wrong and that zombies have over run the hospital. Throughout the game you are updated with the inner thoughts of your character which help lay out this story line and give hints to game play aspects. The game is centered on following the objectives which aid you in finding weapons and items to fight off the zombie hordes. By using ordinary hospital items as weapons and tools to aid your navigation through the floors of the hospital to try to find your wife.

Except there's an interesting twist.... Although, you are having conscious thoughts you begin to notice that your wounds don't look like they came from a car accident alone, could you be infected too?

B. Look and Feel

Breaking Down the Heads-up Display

The heads-up display (HUD) displays all the information a player needs to keep in mind while playing *Zombs*. A major element of the HUD is the mini-map which shows your current position and orientation, the current objective location, and any enemies within a radius of the player. Above the mini map we have the current objective. Oriented to the right of the mini-map are inner thoughts/story line in blue, and hints and game play help appear in yellow.

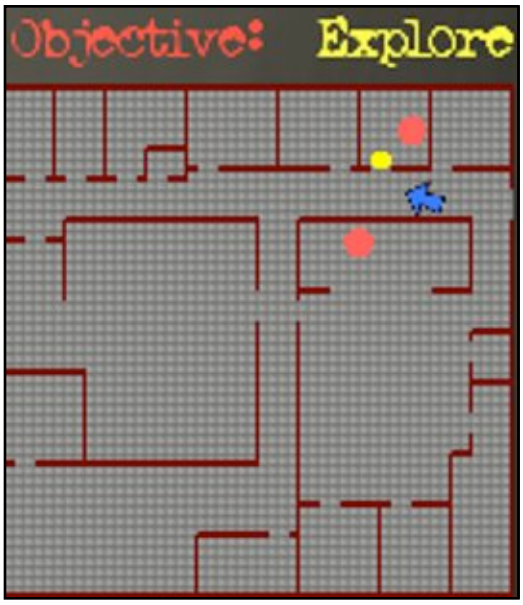

Figure 2.B.1 - Mini-map

Figure 2.B.2 - Story-Line/Hint Text

In the bottom right corner we have a zombie awareness indicator. Currently it is a zombie picture that changes through 6 color sets, portraying the alertness of zombies in the level. The higher the awareness the faster the zombies move, and the longer the path distance for the zombie AI, to get to the player, becomes. This means at the highest awareness level zombies move faster than the player, unless you are sprinting, and all the zombies in the level will be aware of the player and be trying to get to them. In the upper right the collected non-weapons are shown. Currently this contains 3

different types of keys and adrenaline injection. Lastly just to the left of the keys is the player's health and the adrenaline timer bar shows up under the health, when activated.

Keeping Inventory

Figure 2.B.3 - Inventory Display

Our game also includes an inventory that automatically combines, creates, and makes weapons available for use. The system is set up to make a weapons available for selection as soon as all necessary components have been collected. Once the weapons are available they are set to appear in the upper left hand corner. The currently selected weapon is shown in the left corner; all available weapons are show in their upgrading tiers to the right of the current weapon. Below the currently selected weapon its name and associated cool-down are displayed. Each weapon is set up with different use and cool-down weights to reflect the power of the item, and balance them with game play.

Did you hear that?

We also incorporated over 30 different sounds, including 7 sounds looping for background ambiance. There are sound cues for various actions and situations including but not exclusive to: dead bodies, doors, crawlers, zombie pain and groan, non-playable character hiding and found reactions, player pain, melee, heartbeat, use of

adrenaline, and sprint. We used the SDL library for it's API on point-based sound and text display management. We did run into issues with the limitation of the number of available channels. Since we have so many sounds going at any given time some get kicked out of their channel before finishing. We tried to solve this issue by allocating more channels, but we seemed to be constricted to 8 running channels at any one time.

I hate hospitals....

As many fans of the zombie genre know a hospital is probably one of the last places you would want to be during a zombie apocalypse. Capturing the "look and feel" of a hospital was very important for the story line to be reflected in our gameplay. By using actual floor plans of hospital to start level design and then adding hospital-like furniture and textures we were able to emulate the feeling of being in a hospital. Using low ambient lighting and shaders we were able to give a darker feeling to the hospital environment.

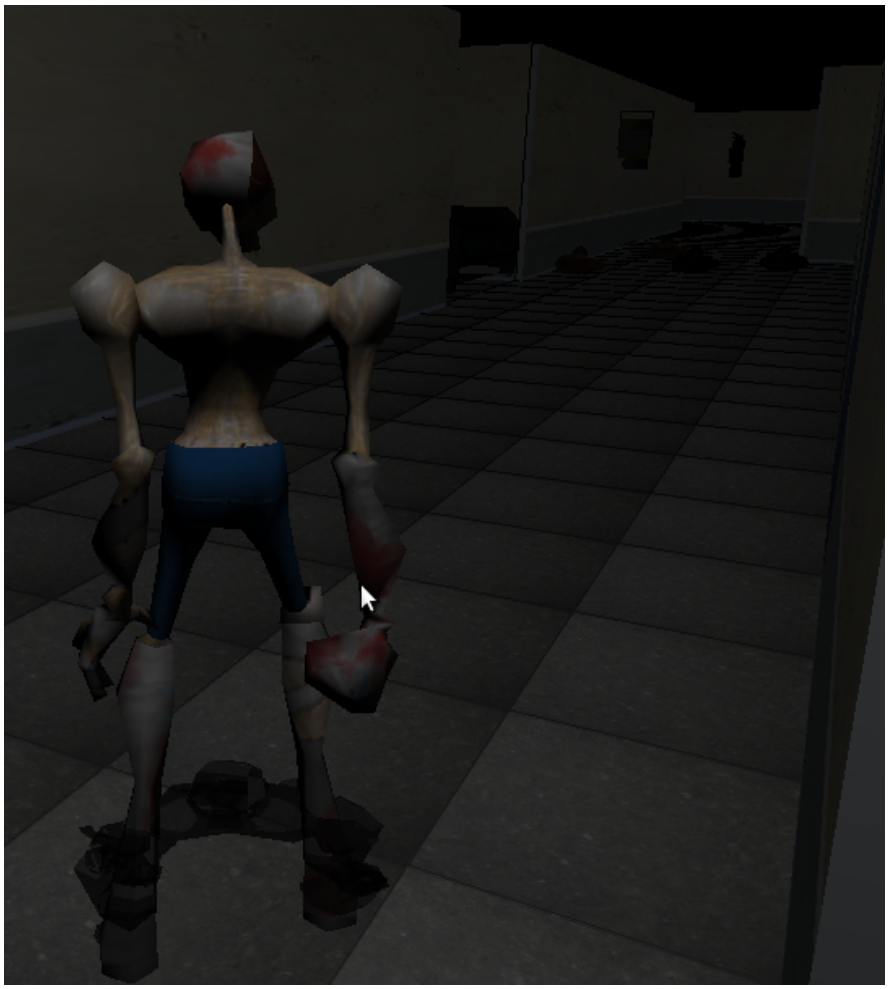

Figure 2.B.4 - Hospital Hallway

C. Gameplay

Zombs is an objective-based game that guides the player through a series of way-points in a level to lead the player to the ultimate objective, finding his wife. The HUD combines the 2D level representation and objectives helps guide the player through the level's objective. This helps with teaching the player critical gameplay aspects and develops the storyline simultaneously.

The key game controls are "wasd" to move, moving the mouse controls the camera's rotation, left click is melee-attack and space bar is designated for weapon use. Secondary game controls include: 'g' uses any adrenaline shots available, 'p' pauses gameplay, and the tab key allows the player to cycle through weapons in their inventory.

Figure 2.C.1 - Player Fighting Zombie

The difficulty we encountered in developing an objective-based game is the balance forcing the player to follow our planned routes while at the same time allowing the player to enjoy the experience. When a player misses an objective they miss crucial story updates, hints, and description of the controls.

III. Level Editor Tool

The level editor was a straight-forward idea but an important tool that was created to aid in producing levels for the *Zombs* projects. Manually creating levels can be a long and tedious project that can take several hours and when things are placed incorrectly it can be a nightmare to correctly identify and replace an object. The level editing tool for this game went through two main stages of development. Due to an initial urgency to produce a level editing for our game, I built a completely standalone tool. However, during the second quarter I realized that the gap between the tool and our game had grown to large to effectively expand functionality for both the game and level editor functionality.

Since nobody in our group has prior experience developing tools for games I felt this would be a good area for me to step in and fill an important role for our group. Yet in the weeks to come I never imagined how much experience I would receive in tool development as I accepted responsibility for this assignment.

Since our gameplay is largely based on level progression, our entire group realized the importance for a tool that would effectively create levels early in development. The initial push for this led to a rushed development and an extremely

simple standalone level editor that created walls. Then, I needed to design a solution to save all this data since a program is unable to retain memory after it has stopped running. There are numerous innovative methods of storing this level descriptions such as representing your level's walls as pixels in a bitmap image. However, instead of implementing an image parser I decided to take the more direct information and save information on the start and end points of each wall. Below you can see a figure of the first level editor adding walls to a level.

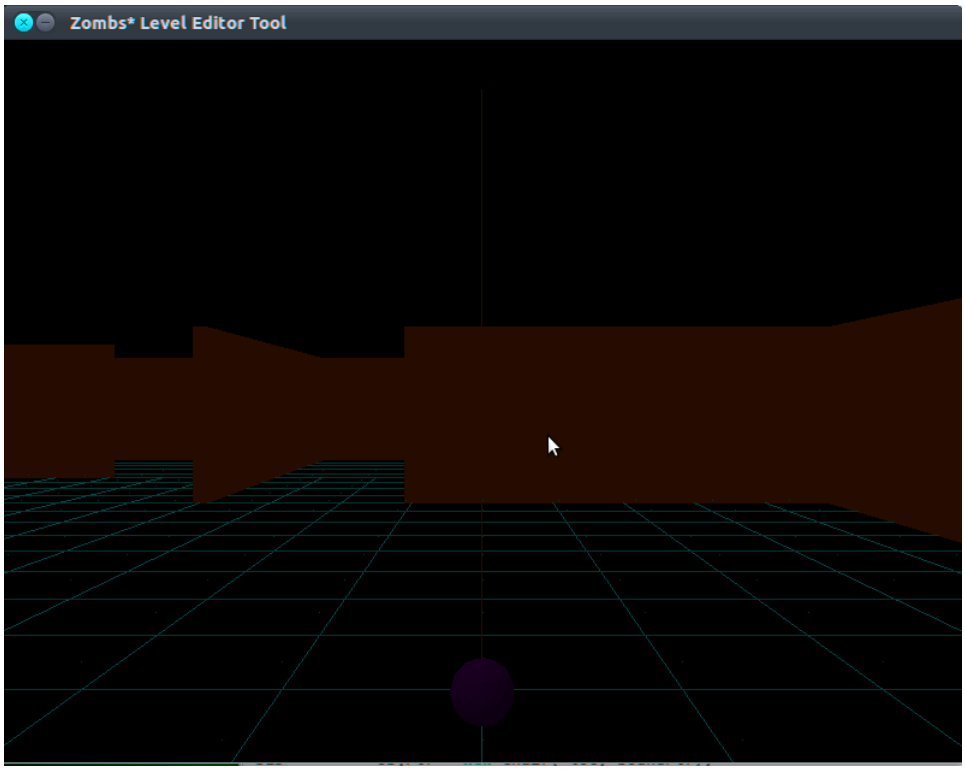

Figure 3.1 - Level Editor - v1.0

As depicted above, the first version of the level editor was a basic grid-based editor. After this initial development, the amount of objects and information our levels vastly expanded and so did the technologies in our game. By the end of the first quarter the shortcomings of the tool were becoming unbearable. The growing number of

objects that occupied our levels led to very confusing controls. Also, the grid-based movement made it very difficult to accurately place items and the fact that there wasn't a method to undo previous placement made mistakes potentially disastrous if you did not frequently use the export functionality frequently. Hence, the birth of the *Zombs* Level Editor 2.0 illustrated in the figure below.

Figure 3.2 - *Zombs* Level Editor v2.0

As previously stated, the main short-coming of the previous level editor had been the growing gap between the technology of our tool and the constantly advancing technologies of our game. To solve this critical issue I decided to inject the level editor code into our game and essentially combine their functionality. With the proper

commands running our game *Zombs* will load into ‘Edit Mode.’ By doing this I was able to see how object’s would look inside our game instantly. This upgraded feature alone made the redesign process worthwhile because previously you would have to shut down the level editor and start up the game to see how changes would look after they had been added. This also allowed the level editor to change as the game did and rarely made it necessary to upgrade for new technologies that were implemented.

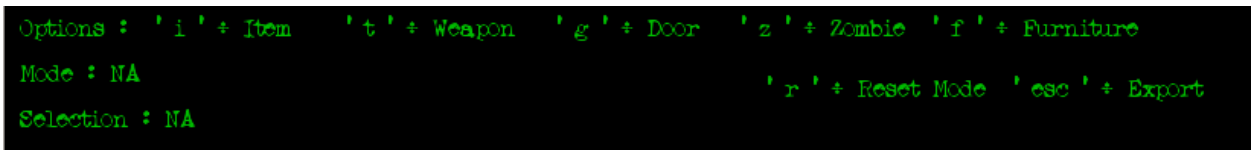

Figure 3.3 - Level Editor v2.0 - Menu

The addition of a ‘Edit Menu’ shown in the figure above allowed for the simplification of controls. This was essential to support the steady increase of unique objects during implementation. This allowed the user to select different items separated into broad categories in the menu. To cite a use-case as an example, when a user wants to add a chair to a level they would first press ‘f’ to access the furniture menu. After this the user is presented with all of the different furniture objects available to the level editor. By pressing ‘c’ a user will select chair mode and will be able to drop a chair into a level with whatever orientation he indicates with the tab toggle. In addition, if a user is unsatisfied by their previous placement by pressing ‘r’ the user reset’s all edits after the last export. This ability to undo and export object placements without quitting the program allowed for a user to create levels significantly faster by granting the user the ability to preview items in the level.

Shown below is a user adding two chairs with different rotation to block off the exit and force the player to choose another route.

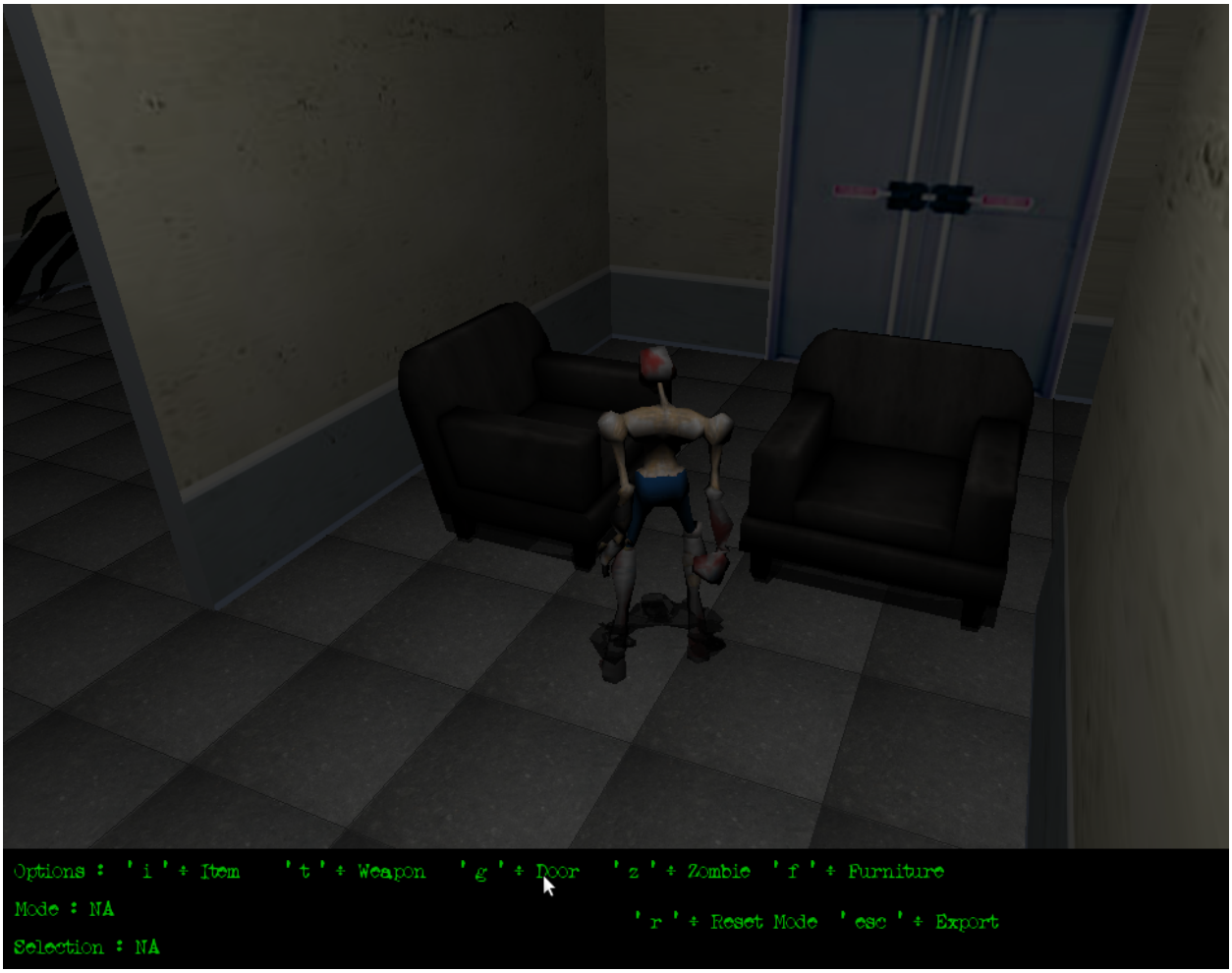

Figure 3.3 - Level Editor in Action

Zombs used a series of files to represent the various objects that make up a level. The different categories of objects represented in these files include weapons/ power-up items, static furniture models, zombies, non-playable characters, walls and doors. These files kept information pertaining to an object's particular type, rotation and position within our virtual world. By giving the level editor the capability to export all the

information in these various files *Zombs* was able import this information and give the same representation of our levels as presented in the level editing tool.

One particularly interesting file that has not been mentioned was our transformation of the level's spatial data for the "artificial intelligence" that controls the movement of all the zombies. This file is a representation of our map as a 50 x 50 grid of zero and one values. One values represent where walls exist and the zeros represent empty space in which zombies can move freely. Using this representation of our level a particular zombie in a level can determine the best possible path to the player. The artificial intelligence is explained in more detail later but to put it simply by calculating the distance of the best path to the player we determine whether a zombie is within range to move towards the player and if so, we make a 'step' towards the player along the path.

To make it possible to use this form of artificial intelligence there had to be certain restrictions placed on our levels and some of the objects within. To symbolize a level with a 50 by 50 grid-like representation we first placed a limitation on the size of our maps to 50 units long and 50 units wide. Additionally, we had to orient our walls on this grid to have concurrency with our abstracted representation used by the zombies. Consequentially since doors are attached to walls this restriction was also forced upon the doors. However, doors are not represented in the artificial intelligence file because we wanted zombies to able to pile up on a door to attack a player if they are playing without any discretion towards their 'awareness' meter.

The level editor made it possible to finish two complete levels in the *Zombs* project. This tool also leaves the door wide open for our group and even other users to

expand upon our game with minimal coding experience or background knowledge by creating new levels.

IV. Results

In the end, we accomplished what our group set out to do; make a game that was fun to play. After the first stage of development we were able to produce an extremely visually stimulating experience. However, in our haste to add content and technologies we neglected to think if the game was fun to play. Since engaging gameplay is arguably the most critical aspect of a game we focused our efforts towards making *Zombs* a fun game to play.

To correct our mistake we literally took our game back to the drawing board and carefully laid out future levels. This allowed us to give our player's direction by giving them a pre-determined route through the level. Determining how the player would progress in a level allowed us to include our storyline and introduce controls as the user played the game. Throughout the development of *Zombs* I frequently had friends play in order to make sure our game was simple to understand and fun to play. This quick and sometimes brutally honest feedback allowed to quickly eliminate aspects that detracted from the player's experience. Following the full development of our first level I had many of the play testers anxiously asking when the second level would be done.

However, not all the feedback was good and we failed to give some of our testers a fun experience. The main cause for negative feedback was when player's chose to ignore our objectives. Without following the objectives the user completely misses out on introductions to essential controls to gameplay, story-line progression, and helpful

hints. Unfortunately, most testers who did not follow the objectives ended up getting confused and generally lost interest with the game very quickly. It was always heart-breaking to see someone struggling to have fun with your game, but identifying aspects that lead to confusion and take away from the experience is critical to making a fun game. Then again, it was also important to keep in mind that it is impossible to truly please *everyone,* but that didn't stop us from trying.

Our team was also able to incorporate complicated algorithms in order to achieve effects such as per-pixel lighting, dynamic shadows, and vibrant particle effects. I am truly proud to say that I took this challenge head-on and was a integral part of the development of *Zombs*.

V. Conclusion

One of the most interesting aspects of working with this game was the seemingly endless amount of time one can spend working on developing a game and it's tools. Making seemingly simple changes to gameplay can sometimes take a fair amount of time, become extremely complicated, and eventually taken out of the game. For instance, I developed an inventory system for keeping a player's items. At first, it was simple and just required keeping a list of references but then extra requirements to make the inventory click-and-drop to enable item combinations made it very difficult. Then the most important lesson of all was after all this development we decided to take all this work out of the game because it made things to complex. At first, I was extremely biased and wanted the product of all my hours of work to be in the game despite how it changed the gameplay. Now, I can see that it was a better decision to try

to simplify the gameplay despite the work I put into development. This is important because with programming requirements change and to be a useful software engineer you have to be ready to adapt to whatever challenges present themselves.

Looking back over the months of development it is amazing to think that *Zombs* started 20 weeks ago with absolutely nothing. *Zombs* evolved from an idea to a set of specifications and then finally became the game it is today. Being apart of the design, development, and testing of *Zombs* is an experience that I will never forget. Learning the in's and out's of game development has been a bumpy road with the constant pressure of adding new technologies to your game. But now that it's over, I cannot stop thinking of new features to add to the game to make it better. This makes me think that future development is almost a certainty.