Call of Duty: Modification

By

Mitch Albrecht

Submitted to the Faculty of the Information Engineering Technology Program
in Partial Fulfillment of the Requirements for
the Degree of Bachelor of Science
in Information Engineering Technology

University of Cincinnati
College of Applied Science

March 2006

Call of Duty: Modification

By

Mitch Albrecht

Submitted to the Faculty of the Information Engineering Technology Program
in Partial Fulfillment of the Requirements for
the Degree of Bachelor of Science
in Information Engineering Technology

Mitch Albrecht	03/09/06
Dr. Sam Geonetta, Advisor	03/09/06
Patrick C. Kumpf, Ed.D. Interim Department Head	03/09/06

Acknowledgements

I would like to give a special thanks to Activision the creators of the original Call of Duty game. Thanks to ID software gaming company who created the Quake 3 engine for the Call of Duty game to run off. Lastly, I would like to thank IW Nation for creating the tools to help map modification experts to create great MoDs for the community.

Table of Contents

List of Figures

Abstract

Call of Duty: Modification is a “first person shooter” PC game. This game is based on World War II action of Allied vs. Axis scenario. The multiplayer version needs more maps to put into rotation. Once a gamer beats the single player game he/she may want more levels to play. First I created the multiplayer map. Leaving it open for snipers and long range guns. I also programmed two explosion effects on two bombsites. It takes the Allied team five seconds to plant and Axis team ten seconds to defuse. The bomb will explode within one minute of the planted time. When creating the single player map I had to add a lot of walls, AI, environmental effects, and a realistic feel. I added talking team AI along with scripted scenes like Axis AI driving a truck full of Axis enemies. Mortars, crashed planes, flames, destroyed buildings, destroyed walls, bunkers, sniping towers, explosion effects, axis spawn flooders, realistic terrain, and all the above when it comes to combat of war.

1. Statement of Problem:

Statistics show around 200,000 people are playing the top 40 first person shooters. Call of Duty is nineth out of 40. Reasons for this include a lack of teamwork modes; the need for a fast/powerful computer, and no new weapons, models, or maps. This is nothing like Half-Life (96,000 players) or Call of Duty 2(7,000 players). I can't change the requirement for using a fast computer but I can change the game physics, modes, weapons, models, and maps.

2. Description of the Solution

COD Radiant, Photoshop, and 3d Studio Max are all professional programs to help create this modification.

By providing a centralized server people all over the world can access, play, and download the modification. Pings would vary depending upon where the server is hosted.

Familiarity with the game is what can draw people towards a MoD. Since Call of Duty is ranked number six for top played game of 2005, it will be easy for users to understand the controls and console commands.

2.1 User Profile

There are two categories of application users: Gamers and server administrators.

2.1.1 Gamers

Gamers are the primary users of this MoD. They are responsible for playing and critiquing a newly developed modification. They also help distribute the MoD by communicating with other gamers.

The gamers are occasional, moderate, or hardcore/career driven users. They can range from the age of 12-55, although ages 13-28 is the most common range. Gamers will always want to play another game created by the same author if they liked the previous one. If the first game was popular the second game will get double the attention due to technology changes such as new engines and graphics.

2.1.2 Server Administrators

Administrators are the second group of users for this MoD. Administrators are responsible for maintaining the server settings. These include all the variables of the type of game play, restrictions to any type of weapon, and monitoring the game for unfair or rude players. Administrators usually play on the server; this helps them regulate it and create a good atmosphere for all gamers. They have the power to ban unwanted players. They can promote dedicated friends/players to administrator status.

Administrators may be any type of gamer. Administrators don't have to be computer geniuses to administer a server. When they purchase the server to run the MoD it is set up with default options. To maximize game play the administrator must

know about configuration files and script commands. This all can be done with Wordpad, so no special tools are needed. The in game commands can be processed through the game console to relay the administrators' messages to the server.

2.2 Modification Specific Solutions

2.2.1 Game Physics and Modes

The game physics and modes are coded in Visual C++ (VC++) and must be changed in that language. The software development kit (SDK) can be downloaded on many web sites including http://www.gamershell.com/news/17958.html.

There are absolutely no physics codes in the core engine itself, the Quake 3 engine. The physics code is massive. An idPhysics object is a tool to manipulate the position and orientation of an entity. The physics object is a container for idClipModels used for collision detection. The physics system deals with moving these collision models through the world according to the laws of physics or other rules. Every idEntity has a pointer to an idPhysics object, the object can be moved around using idForces, though there are other ways to move it. Every frame the idEntity updates its visual model from the origin and axis information in the physics system. The render system has a visual model of every entity, and the physics system has a clip model of every entity, sometimes called "combat model". A collision model can be loaded from the map model using any surfaces marked as clip, or it can be generated from the render model by default it uses all surfaces in the render model, but a simplified object can be created with the

textures/common/collision material applied to it, in which case it uses that simplified model instead), or it can be loaded from a .dm file created using CODEdit. There are multiple subclasses of idPhysics which all implement specialized types of physics. For example, doors use idPhysics_Parametric and rag dolls uses idPhysics_AF. Most of the physics types are self explanatory, and all of them have a brief description at the top of the .h (header) file.

2.2.2 Weapons

An idWeapon is an entity the player holds in front of him as he runs around in the game. It may or may not shoot projectiles, but most of the time it does. The interesting thing about idWeapon is there is one per player rather than one per weapon. When the player switches weapons, it simply loads a new weapon script into the existing idWeapon. Scripting can be done easily through .NET 2005.

2.2.3 Models

This is script for the Axis character model animation:

model axis_mp40

```
{
   mesh         models/axismp40.md5mesh
   channel torso ( *Waist )
   channel legs  ( *Hips Body origin ROT -*Waist)

   anim af_pose  models/md5/af_pose.md5anim
```

```
    anim ik_pose  models/md5/ik_pose.md5anim

    anim stand    models/md5/idle.md5anim
    anim idle     models/md5/idle.md5anim
    anim sight1   models/md5/ sight.md5anim {
        frame 16.5  sound_voice    snd_sight1
    }
    anim walk     models/md5/ walk3.md5anim {
        frame 17    sound_body snd_footstep
        frame 17    triggerSmokeParticle cyber_rfoot_dust
        frame 37    sound_body snd_footstep
        frame 37    triggerSmokeParticle cyber_lfoot_dust
    }
    anim pain     models/md5/ pain_big1.md5anim {
        frame 1    call overrideLegs
        frame 1    sound_voice snd_pain
        frame 16   sound_body  snd_footstep
        frame 49   sound_body  snd_footstep
    }
} Figure 1 (7)
```

The mesh and the animations are defined in separate files and glued together with this axis script. That means I can use the same animation file with multiple meshes. I can also reference the same animation file multiple times in the same model for example, notice the 'idle' animation and the 'stand' animation both use idle.md5anim'. Apart from tying animations to meshes, this declaration type allows me to specify events that happen on certain frames. This makes it trivial to sync sounds and special effects to things like

footsteps, weapons fire, and any other sounds. This is extremely useful in multiplayer coding to make my effects more realistic. To create new models I used 3d studio max.

2.2.4 Maps

Maps in CoD are defined by four different files. For a map to work properly, all four files must be included (the only exception is .aas files are not needed for multiplayer maps).

.map: The .map file is the main file that is created when I edit a file; it defines all the entities and brushes in the map. The other three files are all generated from the .map file with the Dmap command. The format hasn't changed much from the Quake series.

.cm: The .cm file defines the collision geometry in the map. It is used by the physics system for collision detection.

.proc: The .proc contains all the pre-processed geometry in the map. It stores all the visible triangles, batched up in to surfaces. It also stores all the portal information and any pre-calculated shadow volumes. If a light doesn't move, and a brush doesn't move, the shadow volume can be pre-calculated.

.aas: The .aas files contain the 'area awareness' data for the AI to navigate through the level. A separate aas file is generated for each size monster. Generally an aas48 and an aas96 file are generated for most monsters sizes. If a map has a special

monster in it, then it will generate a special aas file for them. CoDRadiant was used to graphically make a new map (10).

3. Call of Duty: Deliverables:

1. A multimedia-based project modifying the first person shooter game, Call of Duty.
    - The project will be created with CoDRadiant, VC++ scripting, PakScape, and Codpiler.
2. The project will provide a new multiplayer map for Senior Design II.
3. The project will provide all aspects of a modified multiplayer map, including the following:
    - Scripting/Programming
    - Search and Destroy
    - Map design will conform with the Game Theory of CoD
4. The project will provide a multiplayer interface to choose between two teams.
    - Allies(American) and Axis(German)
    - Once a side is chosen a choice of multiple guns will be displayed.
    - Guns differ depending upon which team is chosen.
5. The project will provide a fully functional Single Player Map for Senior Design III.
    - The AI will be created with VC++ programming.
6. The project will include three objectives for the player to complete throughout the MoD map.

- The AI will try to stop the player from completing his/her objectives.

4. Product Descriptions and Intended Uses:

Modification (MoD) of Call of Duty uses the Quake 3 engine as well as aspects of the original game. The mechanics and engine for the game will have set rules for new characters, guns, game mode, and a new map. When I distribute it to the public I must have it running on a centralized server. When a client attempts to connect to the server it will then send the downloadable files to the client. Once the client has finished his/her download he/she can play on any server running this MoD. Users also can load the files onto their server to host the MoD. www.codfiles.com is an example of the fan based website with the most current MoD's. Uploading the MoD onto this site will give it more exposure.

The main elements in the creation of this MoD are Visual C++, visual user interface, animation, and digital artwork. The character, weapons, and mode of the game must be programmed properly to create a crash free and error free environment. As a character moves it talks to the server about the specific actions the user wants performed. This is done instantaneously by responding to the client. This user will play with and against other users to accomplish game objectives. If the animation isn't done properly, the user could possibly crash with an error and/or see visual spikes. The digital artwork is for the user's viewing pleasure.

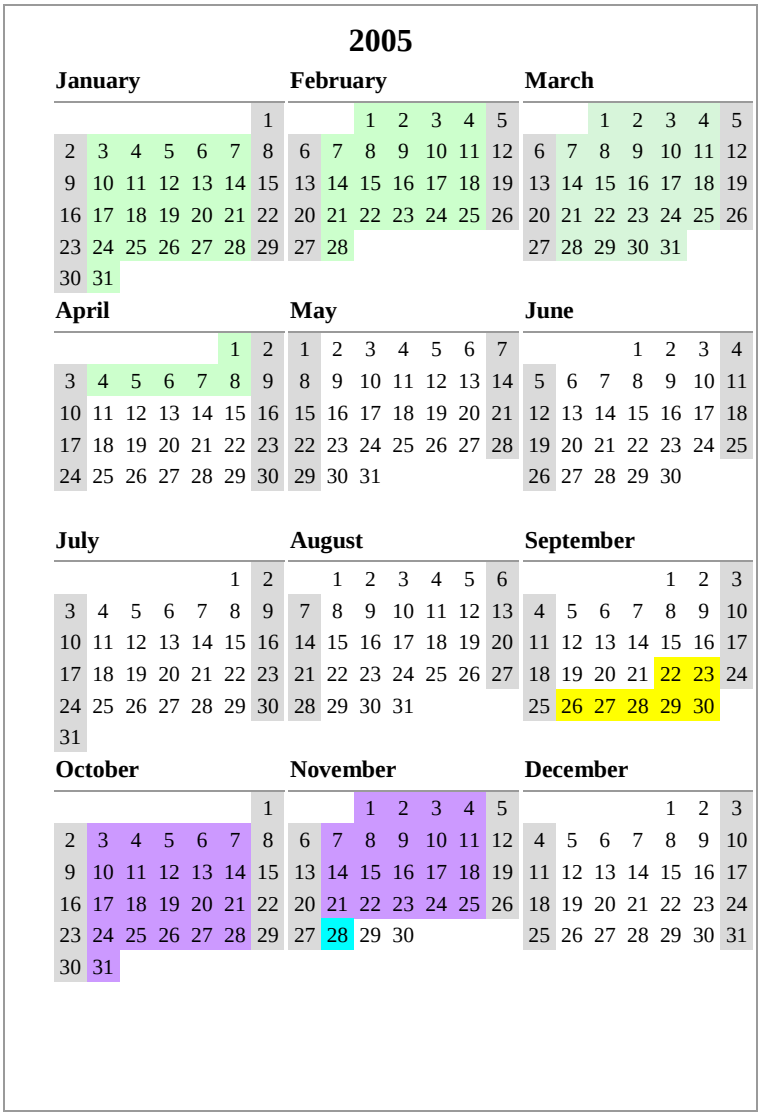
4.1 Timeline:

Time LINE

Senior Design: Call of Duty Modification
Mitch Albrecht

Design Freeze Week of October 31, 2005.
Mapping through out the whole quarter.
Presentation preparation: 11/15/05 – 11/28/05
Testing map: 11/19/05
Bombsite design: 11/21/05
Effects design: 11/23/05
Texture design: 11/24/05

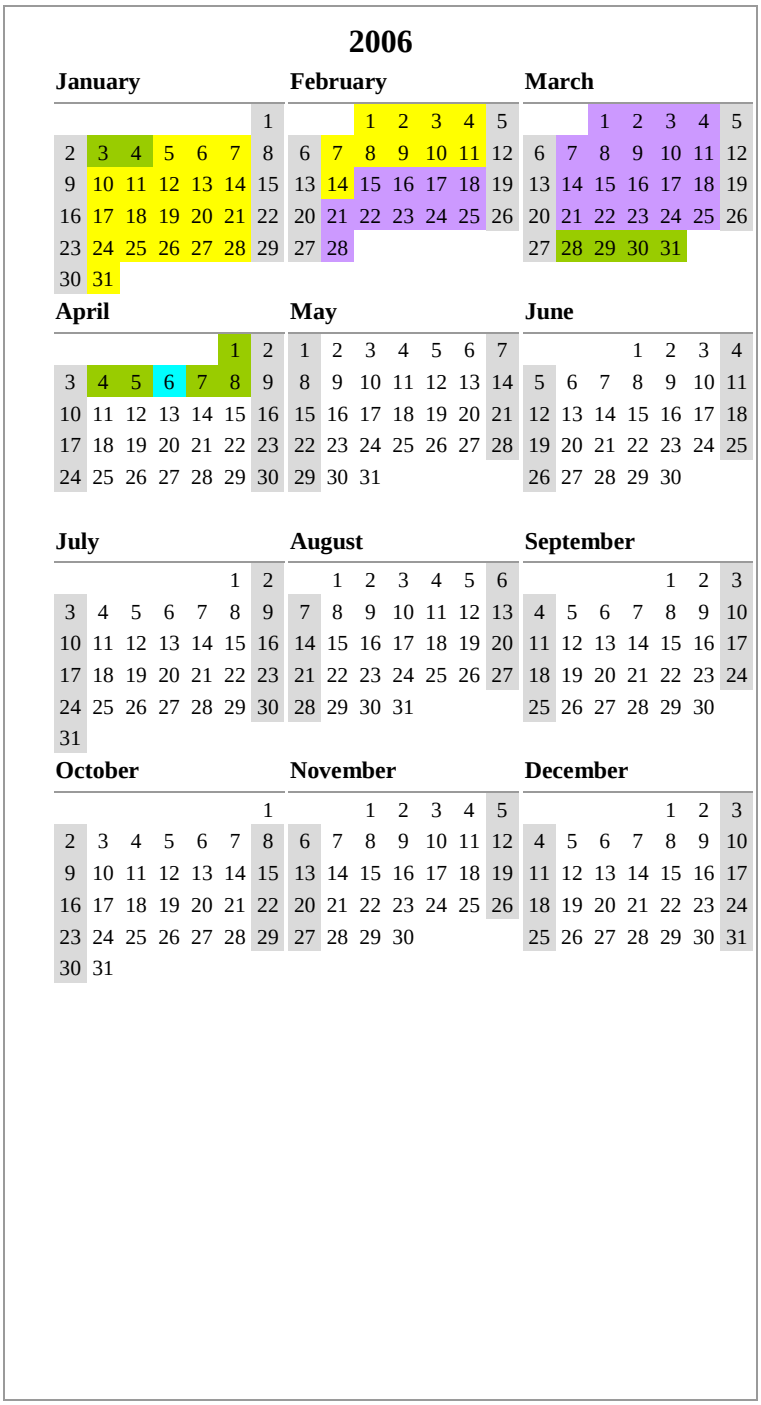
PROJECT PHASE	STARTING	ENDING
Senior Design 1	01/03/05	04/08/05
Senior Design 2	09/22/05	11/28/05
Prototype Mapping	10/03/05	11/27/05
Presentation	11/28/05	



Senior Design: Call of Duty Modification
Mitch Albrecht

Final Report Week of February 06, 2006.
Mapping through out the whole quarter.
Presentation preparation: 03/01/06 – 03/06/06
Multiplayer map: Completed
Manipulate MP map: 01/05/06
Flow Design: 01/17/06
SP map without AI: 02/15/06
Axis AI: 02/15/06
Allies AI: 02/25/06
Bugs + Testing 03/01/06
Presentation: 03/06/06

PROJECT PHASE	STARTING	ENDING
Senior Design 3	01/03/06	04/08/06
Single Player Map Creation	01/05/06	02/15/06
Artificial Intelligence Creations	02/15/06	02/25/06
Presentation	03/06/06	



4.1.1 Senior Design I Accomplishments:

During Senior Design I, I accomplished the following:

- Analyzed the most popular First Person Shooters
- Performed research of gamers needs
- Considered the specific needs of the targeted audience
- Increased knowledge of game design tools
- Structured modification design and layout
- Began the development process
- Developed my proposal and oral presentation

I began researching other popular modifications.

4.1.2 Senior Design II Accomplishments:

During Senior Design II, I accomplished the following:

- Designed the Multiplayer Map
- Created special effects that explode after a bomb has been planted
- Started development of Friendly and Enemy Artificial Intelligence (AI) for the single player (SP) Map
- Prepared Design Freeze documentation and completed the oral presentation

Designing the Multiplayer Map, making sure the map conforms to game theory of CoD, making effects, and explosions realistic. Senior Design II I completed a fully functional Multiplayer map for the community to enjoy. I began planning ahead for the single player map. I had to change the design to conform to SP. AI takes a lot of programming and work. I began planning for Senior Design III.

4.1.3 Senior Design III Accomplishments:

During Senior Design III I accomplished the following:

- Completed game programming
- Tested game design
- Modified project as needed
- Completed documentation for the project
- Presented the final project

Creating a single player is a lot more involved than creating a MP map. It consisted of a lot more programming and planning. Just placing the models in the map was difficult. Then I had to program them to have triggers so they would do actions at certain times depending upon how the player would react.

4.2. Budget:

Hardware: This project requires a fast and powerful computer.

Item	Cost	Brand
Hard Drive	$129.00	Hitachi Serial ATA 250 GB
CPU	$145.00	AMD 64bit 3000+
Motherboard	$129.00	MSI K8T Neo
Video Card	$197.00	Geforce FX5900 Ultra
Ram	$146.00	Kingston HyperX 1GB
Case	$100.00	Acrylic
PSU	$50.00	Demon 650watt
Map Editor	$0.00	QERadiant
Model Editor	$2395.95	3D Studio Max 7
Code/Script Editor	$699.00	.NET 2005
Call of Duty SDK	$0.00	Software Development Kit
Call of Duty Game	$50.00	Call of Duty
FINAL PRICE:	$4090.95	
Budget	$1000	
Minus 3d Studio Max	-$2395.95	
Minus .NET 2005	-$699.00	
Final Budget Price	$996.00	

5. Proof of Design

The next section shows in detail how deliverables of the project were fulfilled and what challenges I encountered.

5.1 Allied Introduction

In the Allies versus Axis scenario the allied team is supposed to plant one of the two bombs within a set amount of time. If the Axis defends it in that amount of time they are deemed the winner. If all the players are killed before the bomb is planted the team with players left is the victor.

Figure 2. Allied Spawn Point

5.2 Axis Introduction

Axis spawn is on the other side of the map and closer to the bombs so they have the advantages of defense.

Figure 3. Axis Spawn Point

5.3 Allied Run for Bomb A

Allies need to develop strategies based on the map. They can decide to go A or B. When deciding which map they must have set grenades and specific spots to watch. They can make quick or slow selections. There are many spots in which to hide. There are many spots to get shot from an Axis player. The next sequences of

screenshots show the allied player going to A bomb site. I also show the opposite view where the Axis defends.

Figure 4. Allied Bunkers

An Allied player can get billed easily with the fire power of a MG42.

Figure 5. Axis MG Defending

Figure 6. Allied Rush Warehouse

This Allied player must be careful entering the warehouse because he doesn't know where the Axis player is.

Figure 7. Axis Defend Warehouse

This Axis player must be careful because there can be an Allied player lurking above him.

Figure 8. Allied Above Warehouse

5.4 Successful Explosion of Bomb A

My favorite part of this modification was programming the bomb. The screenshots demonstrate the bomb icon coming up when in range to plant. The planting bar and message that comes up after an Allied has planted. Lastly, I will show the explosion of Bombsite A.

Figure 9. Before Planting

It is set to take five seconds to get the plant off. The bar indicates how long the player has left before the plant is set.

Figure 10. While Planting

Once the plant is down it will indicate the action was completed.

Figure 11. Done Planting

If the bomb isn't defused by an Axis player within sixty second the bomb will explode. The Allies win if that's the case.

Figure 12. Bomb A Explosion

5.5 Allied Run for Bomb B

This is another path for the Allies. There are many ways to get to a destination, but it can be difficult with Axis lurking around every corner. Figures show the underground path, the bombed house, and upper path to B site.

Figure 13. Allied Run to B site

This is house is good to sniper from and not be MG42'd from the warehouse.

Figure 14. Allied Run to B site

Allies going this way may take longer, but it is safer.

Figure 15. Allied Underground Road

There's a lot of cover underneath for a stealthy attack.

Figure 16. Allied Underground Road

Figure 17. Allied in front of B

5.6 Allied Successfully Planting of Bomb B

Figures demonstrate the planting, timing, and explosion graphics of the B bomb site.

Figure 18. Allied At B

Figure 19. Allied At B: Middle of Plant

Figure 20. Allied At B: Bomb Planted

After sixty seconds the bomb explodes and the Allied player is awarded one match point.

Figure 21. Bomb site B explosion

5.7 Axis Defusing Bomb B

Figures are a series of screenshots showing the Axis player from spawn point running to defuse Bomb B. He successfully defuses it within the sixty seconds. Technically an Axis player only has fifty seconds to defuse since it takes ten seconds to completely defuse.

Figure 22. Axis Run to B

There's a lot of detail within the smallest areas of the map.

Figure 23. Axis Run to B

There are many rooms for Axis to camp in for better defensive positions.

Figure 24. Axis Run to B

Figure 24 shows the Axis five seconds into diffusion.

Figure 25. Axis Bomb B

Once the action is completed the players get a message saying the Bomb has been diffused.

Figure 26. Axis Defused

6. Testing Procedures

Due to the compile times I really had to watch what I added to the main program. Compile times could range from three hours to seven hours. I couldn't make little changes without losing a lot of time. . I had to plan it so each time I compiled it; I would know what worked and may have caused an error. I usually tested everything on a small scale. Putting the new material created in a small sky box with a generic ground. Testing it in this scenario wouldn't guarantee functionality but have a high percentage of success rates. I also could fix any type of graphical bug using the small map scenario and do the same for the big map.

7. Conclusion

This product was created in response to a need for good game modifications. I designed the project for any type of gamer to enjoy. I learned how to use many tools like codradiant, pakscape, codpiler, and .net/scripter. The project was completed over three quarters of senior design plus a summer. The budget of $1000 would be a realistic estimate for this type of mod. Testing ensured that the design freeze deliverables were fulfilled.