2007

Video game development with 3D Studio Max and the XNA framework

Cole Mahoukau Koffi

Follow this and additional works at: https://scholarworks.lib.csusb.edu/etd-project

Part of the Game Design Commons, and the Software Engineering Commons

Recommended Citation
https://scholarworks.lib.csusb.edu/etd-project/3228

This Project is brought to you for free and open access by the John M. Pfau Library at CSUSB ScholarWorks. It has been accepted for inclusion in Theses Digitization Project by an authorized administrator of CSUSB ScholarWorks. For more information, please contact scholarworks@csusb.edu.
VIDEO GAME DEVELOPMENT WITH 3D STUDIO MAX
AND THE XNA FRAMEWORK

A Project
Presented to the
Faculty of
California State University,
San Bernardino

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
in
Computer Science

by
Cole Mahoukau Koffi

December 2007
VIDEO GAME DEVELOPMENT WITH 3D STUDIO MAX
AND THE XNA FRAMEWORK

A Project
Presented to the
Faculty of
California State University,
San Bernardino

by
Cole Mahoukau Koffi

December 2007

Approved by:

Dr. David Turner, Chair, Computer Science

Dr. Kerstin Voigt

Dr. Ernesto Gomez

11/8/2007
ABSTRACT

Developing a video game is a team work that requires time, determination, good discipline and various talents. With each person involved in the game development having their role defined, a game could be developed in a limited amount of time.

The research areas carried in this project include learning to develop three-dimensional models using specific tools such as 3D Studio Max and integrating those models in the game engine, which is written in C# programming language.

CyberShot is a top down shooter game that has been developed using the XNA framework which is available for free and provides a platform for game development.
ACKNOWLEDGEMENTS

First of all, thanks to my advisor Dr. David Turner for all his support, his encouragement and his constructive input to the project. Special thanks to Dr. Kerstin Voigt and Dr. Ernesto Gomez for accepting to be members of my committee.

Thanks to Dr Conception for allowing me to be part of the summer game project team, and James Finley for sharing his knowledge on three-dimensional modeling. Thanks to Marc Chapman for making this project happen and also to the rest of the team for creating a welcoming environment which made the collaboration so much easier that I thought.

Many thanks for the strong support of Lakisha and Eric Tillman, Cynthia and Philippe Boortz, Isabelle and Jean-Pierre Serre, Zachary Howe, Daniel Apodaca, Patricia Aldama and Alex Solano.

My deepest thanks go to my dear and beloved parents, Tiburce and Valerie Koffi for allowing me to come study in the US, and for all their active help that raised confidence in me to achieve my goals. Many thanks to my two younger sisters, Mokan Koffi and Damirifa Brouzro.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>iii</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>iv</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>viii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>ix</td>
</tr>
<tr>
<td>CHAPTER ONE: INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Background</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Significance</td>
<td>3</td>
</tr>
<tr>
<td>1.3 Purpose</td>
<td>3</td>
</tr>
<tr>
<td>1.4 Project Overview</td>
<td>4</td>
</tr>
<tr>
<td>1.4.1 Scope</td>
<td>4</td>
</tr>
<tr>
<td>1.4.2 Definitions, Acronyms and Abbreviations</td>
<td>6</td>
</tr>
<tr>
<td>1.5 Overall Description</td>
<td>7</td>
</tr>
<tr>
<td>1.5.1 Product Perspective</td>
<td>7</td>
</tr>
<tr>
<td>1.5.2 Product Functions</td>
<td>9</td>
</tr>
<tr>
<td>1.5.3 Users</td>
<td>9</td>
</tr>
<tr>
<td>1.5.4 Constraints</td>
<td>10</td>
</tr>
<tr>
<td>1.5.5 Assumptions and Dependencies</td>
<td>10</td>
</tr>
<tr>
<td>CHAPTER TWO: RESEARCH</td>
<td></td>
</tr>
<tr>
<td>2.1 Gameplay</td>
<td>11</td>
</tr>
<tr>
<td>2.1.1 Definition of Gameplay</td>
<td>11</td>
</tr>
<tr>
<td>2.1.2 Key Components of Computer Games</td>
<td>12</td>
</tr>
<tr>
<td>2.2 Game Genres</td>
<td>13</td>
</tr>
<tr>
<td>2.3 Game Engines</td>
<td>13</td>
</tr>
</tbody>
</table>
LIST OF TABLES

Table 1. Game Divisions .......................... 14
LIST OF FIGURES

Figure 1. Key Components of Computer Games ............... 12
Figure 2. Screenshot of Independent Study Game ............. 16
Figure 3. Floater Wireframe ............................... 17
Figure 4. 3D Studio Max Interface ........................... 19
Figure 5. Concept Art of Command Core ....................... 21
Figure 6. Adding Background Image
        in 3D Studio Max ........................................ 22
Figure 7. Selected Vertices Turned Red ....................... 23
Figure 8. Deleting Vertices .................................. 24
Figure 9. Using Symmetry Modifier ........................... 25
Figure 10. Using Boolean Operation ......................... 26
Figure 11. Model with Tentacles ............................. 27
Figure 12. Command Core Three-Dimensional Model .......... 28
Figure 13. White Hornet Concept Art ......................... 29
Figure 14. Making White Hornet Using Box with
        Cylinder ................................................. 30
Figure 15. Making of Wings .................................. 31
Figure 16. White Hornet Three-Dimensional Model ........... 32
Figure 17. Bulldog Concept Art .............................. 33
Figure 18. Making Bulldog Using Spheres
        and Cylinders ............................................ 34
Figure 19. Making Bulldog Using Duplicate Method .......... 35
Figure 20. Bulldog Three-Dimensional Model ................. 36
Figure 21. Floater Concept Art .............................. 37
Figure 22. Making Floater with Bevel ......................... 38
| Figure 23. Floater in 3D Studio Max | 39 |
| Figure 24. Floater Three-Dimensional Model | 40 |
| Figure 25. Mound Concept Art | 41 |
| Figure 26. Making Mound with Beveled Cylinder | 42 |
| Figure 27. Making Mound's Body | 43 |
| Figure 28. Mound Three-Dimensional Model | 44 |
| Figure 29. Hive Concept Art | 45 |
| Figure 30. Making of Hive | 46 |
| Figure 31. Hive Three-Dimensional Model | 47 |
| Figure 32. Making of Stinger | 48 |
| Figure 33. Stinger Three-Dimensional Model | 49 |
| Figure 34. Game's World Coordinates | 50 |
| Figure 35. Game's World View | 51 |
| Figure 36. RedSparrow Model at Position (0, 10, 0) | 55 |
| Figure 37. BlackScarab Shooting Laser Missile | 59 |
| Figure 38. Game Sequence | 60 |
| Figure 39. Class Diagram of Game Screen | 62 |
CHAPTER ONE
INTRODUCTION

This project is about understanding game development using 3D Studio Max and the XNA framework.

Developing a video game is a team work and requires various talents. We need 2D artists, 3D artists, game designers, musicians and programmers to work together in order to create a successful game.

The research areas include learning to develop three-dimensional models using specific tools such as 3D Studio Max and integrating those models in the game engine, which is written in C# programming language.

In this project, a top down shooter game called "CyberShot" will be developed using the XNA framework. The XNA framework is available for free and provides a platform for game development.

The core gameplay will be centered around shooting enemies to get the maximum of points. The player will be going through different levels that will be clarified further down in the documentation.
The two different ways of controlling the player will be: the XBOX 360 controller or the computer keyboard. The functions of the different buttons and keyboard keys will be specified as well.

The research carried out throughout this project gathered a lot of useful information about developing video games using 3D Studio Max and XNA framework, and can be used as reference for future students interested in video game development with these tools.

1.1 Background

The idea of developing a video game came to mind after I completing an independent study with Dr. Turner on game programming using Blender. I found out that there are a lot to consider when developing a game such as writing the storyline, creating the concept art, making the 3D models, texturing the models, adding music and special effects to the game, synchronizing the different levels, etc.

Although the independent study lasted only one quarter (Fall 2006), I did learn a lot and decided to expand my knowledge in the game development field by working on a project in order to someday be an experienced, highly-skilled three-dimensional professional game developer.
1.2 Significance

This project will also help me put in practice what I have learned so far and understand what a game development environment is. Besides, collaborating with other game developers will enable me to get an experience working with other developers on a development project and prepare me for a good career start. In addition, I found out that Microsoft enrolled a number of universities in the US (including USC, the University of Southern California and Georgia Tech College of Computing) which all have planned to include console game development, 3D Studio Max and XNA framework in their curricula [6]. So, the realization of this project will help me measure up to the newest trend in the game industry.

1.3 Purpose

The purpose of this project on game design and development is to experiment with actual technology (including professional tools that I am not familiar with) used in computer games and get an experience in a three-dimensional game development project. Most of the project involves research about technologies that I am a little familiar with, such as game development, game genres, and game engines currently used. However, I will need to learn several other tools used in 3-dimensional games development. I decided with my advisor to
collaborate with other students working on a summer game development project under the guidance of Dr. Turner and Dr. Concepcion.

1.4 Project Overview

1.4.1 Scope

A top down shooter game called "CyberShot" will be developed within this project using the XNA framework. By way of the Microsoft XNA connection with Autodesk, the three-dimensional models created can now be easily added into the XNA framework. This is able to be done by using Autodesk's FBX file exchange format [6] which is, by far, the most wide-used and supported 3 dimensional file exchange format on the market. The models are made of several different primitives, and it is possible to add new primitives into the models for further uses.

The core gameplay will be centered on shooting enemies to get the maximum of points. The player will be going through different levels that will be clarified further down in the documentation. The two different ways of controlling the player will be: the XBOX 360 controller or the computer keyboard. The functions of the different buttons and keyboard keys will be specified as well.

In order to get points, the player will need to destroy
enemies. More points will be allocated to bigger enemies, such as bosses, and fewer points for smaller enemies, such as little spaceships. As you play the game, points will be accumulated to get high scores. The difficulty will focus on the amount of spaceships on the screen and the amount of projectiles they will fire. Sometimes, environmental obstacles and enemy fire will have to be avoided.

A boss battle will be the final step for each level. Players will then need to be allowed to use powerful weapons such as homing missiles, a laser, and a spread shot. They will also be allowed to use two different weapons when needed, firing them one at a time or in tandem. Depending on the combination of weapons used, weapons in tandem will produce different effects.

Players will also have access to a super-weapon they will use to get out of tight situations, such as having a lot of spaceship enemies on the screen at the same time.

Finally, the player will have the option to choose from three different ships before starting the game. Each ship will have their own special effects and super-weapons, but will share the normal weapon effects.

My tasks will include making the three-dimensional models using the concept arts created by Mark Chapman, working on the
game menu, testing the game, making several level designs and the transition scenes.

1.4.2 Definitions, Acronyms and Abbreviations

3D Studio Max - Three-dimensional graphic application by
   Autodesk Media and Entertainment [4].

Autodesk - Provider of three-dimensional software.

Blender - Free open-source software program for three-
   dimensional modeling, animation and rendering [5].

Common Language Runtime - Virtual Machine component of
   Microsoft .NET [4].

C# - Object Oriented programming language.

Direct3D - application programing interface for drawing
   three-dimensional graphics.

FBX - three-dimensional exchange format.

NPCs - Non Player Character controlled by the computer.

OpenGL - (Open Graphics Library) is an application interface
   for programming graphics components of software programs [4].

Primitives - Basic geometric shapes and forms in three-
   dimensional applications such as spheres, cubes, cylinders, pyramids.

Shader Profile - It specifies the assembly shader version to
   use for shader execution.
Shaders - They are algorithms for rendering meshes.

Spread shot - Three-way shots.

Texture - Image put on the faces of a three-dimensional model.

XBOX 360 - Video game console by Microsoft.

XNA Framework - Set of managed libraries based on Microsoft.NET framework [7].

1.5 Overall Description

1.5.1 Product Perspective

The video game will be a top down shooter game called "CyberShot." The level design has been done by Mark Chapman, a student from the CSCI department. It will be able to run on Windows XP, Windows Vista and the Xbox 360.

The two different ways of controlling the player will be: the XBOX 360 controller or the computer keyboard. Here are the functions of the different buttons and keyboard keys:

- In-Game: Xbox 360 Controller
  - Start Button: Menu
  - Left Analog Stick: Movement of the player ship in all directions
  - Left Trigger: Fire the left weapon
  - Right Trigger: Fire the right weapon
• Y button: Fire the superweapon

• In-Game: Keyboard
  
• Left, Right, Up, Down Arrow Key: Movement of the player ship in all directions

• Q: Fire the left weapon

• W: Fire the right weapon

• E: Fire Combo weapon

• S: Fire the superweapon

The core gameplay will be centered around shooting enemies to get the maximum of points. The player will be going through different levels that will be clarified further down in the documentation.

The game will have different levels with increasing difficulty. The difficulty will focus on the amount of spaceships on the screen and the amount of projectiles they will fire. Sometimes, environmental obstacles and enemy fire will have to be avoided.

1.5.2 Product Functions

The game will be designed for only one player. The game will begin with three different ships that the player can have access to. The three spaceships are named the Red Sparrow, the Black Scarab, and the White Hornet.

The Red Sparrow will carry a powerful weapons system that
will contribute in enhancing the power of its normal weapons.

The Black Scarab will carry a plasma force field generator used to absorb smaller projectiles.

The White Hornet will be an experimental craft, acting as a mobile platform for a newly developed beam cannon.

The player will be able to manipulate each one of the three ships similarly. The spaceships' weapons will be combined to generate different effects. Each spaceship will only have a unique super-weapon.

In the first level, the player will be traveling through the span of space separating the satellites and the enemies, intercepting the first waves of enemy attackers.

1.5.3 Users

The target audience is effectively everybody. The simple design and control will suit players of all ages.

1.5.4 Constraints

The program will run on personal computers that are equipped with a high quality graphics card and the .NET libraries. It will be developed using the Microsoft XNA Game Studio Express which is a collection of game development tools based on Microsoft Visual C# 2005 Express Edition [7]. It includes a large class of libraries that are all part to game development. The XNA Game Studio Express combines the XNA framework that runs on a version of the Common Language
Runtime. Any XNA games can run on Windows Xp, Windows Vista and Xbox 360. The XNA framework allows the use of the Xbox 360 controllers.

1.5.5 Assumptions and Dependencies

We assume that the users have a graphics card that supports Microsoft's Shader Profile 2.0 or higher.
CHAPTER TWO
RESEARCH

Doing some intensive research on game design and
development was the core part of this project. This section
below describes the research done.

2.1 Gameplay

2.1.1 Definition of Gameplay

Gameplay [3] consists of:

• The challenges that a player must face to get to the
  object of the game, and
• The actions that the player is allowed to take to face
  those challenges.
2.1.2 Key Components of Computer Games

There are 2 conceptual components: the core mechanics and the user interface (see figure 1).

The core mechanics produce gameplay. They define the challenges and the actions taken by the player in order to meet those challenges. They also determine the result of those actions on the game world.

The user interface is between the core mechanics of the game and the player. It usually displays the outputs and
receives the inputs. It has two essential features: perspective and interaction model.

Perspective refers to a certain camera angle (point of view).

Interaction model represents the relationship between the buttons pressed by the player and their resulting actions.

2.2 Game Genres

A game genre is a type of game. Andrew Rollings, author of "Game Design and Development" describes game genre as a category of games characterized by a particular set of challenges, in spite of settings or game-world contents [3].

The purpose of genres is to assign each game to a particular category.

2.3 Game Engines

A game engine is the part of the game's software that runs the core mechanics. With the game engine, the game can run on several different platforms such as video game consoles and desktop operating systems [4].

Theses are the core functionalities provided by a game engine: a rendering engine, a physics engine, networking, animation, scripting, sound.
2.4 Game Divisions

Developing a game is a team work. While doing some research, I found out that there are five main personnel divisions with each containing a number of roles [3].

Table 1. Game Divisions

<table>
<thead>
<tr>
<th>Division</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Management and Design</td>
<td>Software planner</td>
</tr>
<tr>
<td></td>
<td>Lead architect</td>
</tr>
<tr>
<td></td>
<td>Project manager</td>
</tr>
<tr>
<td></td>
<td>Game designer</td>
</tr>
<tr>
<td>Programming</td>
<td>Lead programmer</td>
</tr>
<tr>
<td></td>
<td>Programmer</td>
</tr>
<tr>
<td>Art</td>
<td>Lead artist</td>
</tr>
<tr>
<td></td>
<td>Artist</td>
</tr>
<tr>
<td>Music and Miscellaneous</td>
<td>Musician</td>
</tr>
<tr>
<td></td>
<td>Sound effect technicians</td>
</tr>
<tr>
<td>Support and Quality Assurance</td>
<td>Technical support</td>
</tr>
<tr>
<td></td>
<td>QA lead</td>
</tr>
<tr>
<td></td>
<td>QA technician</td>
</tr>
<tr>
<td></td>
<td>Playtester</td>
</tr>
<tr>
<td></td>
<td>Support technician</td>
</tr>
</tbody>
</table>
CHAPTER THREE
THREE-DIMENSIONAL MODELING

Making a computer game for my Masters Project comes from the amazing experience that I had when doing an Independent Study on Game programming with my advisor Dr Turner. I had a chance to learn about blender which is a free open source three-dimensional creation tool used for modeling three-dimensional objects, applying texture, rendering and animating them.

The game (see figure 2) that I designed and developed is called "Pumpkin Blast."

3.1 Independent Study Using Blender
Blender was the three-dimensional modeling software used for the independent study. It is a free open source software. It is able to run on several different operation systems, including Microsoft Windows, GNU/Linux, Mac OS X [2].
Here is a screenshot of the game that I developed during the Fall quarter of 2006. The Game is called "Pumpkin Blast." The basic gameplay involves rolling a pumpkin from the start pad to the end pad around a platform (usually several platforms), while avoiding the hazards that are placed along the way.

![Game Menu](image)

**Figure 2. Screenshot of Independent Study Game**
3.2 Three-Dimensional Modeling

A three-dimensional model is the product resulting from three-dimensional modeling. Three-dimensional modeling is a series of actions for making a wireframe representation of three-dimensional objects. A three-dimensional model is rendered using wireframe representation.

Figure 3. Floater Wireframe

Nowadays, three-dimensional models are used in quite a number of industries. For instance, in the medical field, they are used for organ models. In the movies, they are used for
making real-life motion pictures. In the architecture domain, they are used for developing proposed buildings.

3.3 3D Studio Max

3D Studio Max is a three-dimensional software application that is used worldwide. Strongly present among video game developers and television commercial studios, 3D Studio Max is known for its capabilities of doing excellent modeling and texturing, animation, specialized visual effects, rendering, movie effects. 3D Studio Max is an expensive software that runs on the Microsoft Windows operating system (Win32 and Win64 platform).
In contrast to Blender, 3D Studio Max has a much easier interface (see figure 4). It has a useful number of primitives with parameters. It also has the polygonal editing tool (probably the most powerful modeling tool). There are already four different views that are preset: top view, front view, left view and perspective view. Besides, the four preset views, there are five additional views: bottom view, back view, right view, isometric user view and camera view. Users are free to change the preset views according to their preferences and adjust the size of each window.
The newest version of 3D Studio max is 3D Studio Max 9, released on July 31, 2006.

3.4 Construction of the Models

3.4.1 Command Core

This model represents the second stage boss. It consists of three units, each one of them directing the ship. They are located inside a large, armored node. Two mechanical tentacles are attached to that large node and are responsible to protect it from any threat. Each section of the tentacle houses 9 cannons, including the powerful beam projector used to destroy any threat.
The first thing to do is to get the concept art of the Command Core (see figure 5). This concept art has been done by Mark Chapman (Art Team Coordinator).

Figure 5. Concept Art of Command Core
Then, we need to add it as a background in one of the windows (see figure 6). In Views tab, go to Viewport Background and locate the picture from the Background Source Files.

Figure 6. Adding Background Image in 3D Studio Max
Select a sphere shape to edit a sphere primitive that is converted to an editable polygon of 12 segments and 227'9 of radius. Then, in the Edit Polygons rollout, select vertex. The vertices selected turned red (see figure 7).

Figure 7. Selected Vertices Turned Red
Delete the selected vertices in order to get half of the large armored node (see figure 8).

Figure 8. Deleting Vertices
Use the Symmetry modifier from the Modifier List tab in order to get the entire armored shape. Make sure that you choose the X mirror axis and check the flip box (see figure 9).

Figure 9. Using Symmetry Modifier

Then, use one of the most interesting features that 3D Studio Max offers. It is the 3-dimensional boolean operations, available from the geometry tab.
Create a new sphere and click the boolean Compound Object button. The armored node is now Operand A. Click pick Operand B, and choose the new object created. What we want to do, is to combine both geometries in order to get only one object. Figure 10 shows the resulting object from using the boolean operation, subtraction (A-B).

Figure 10. Using Boolean Operation
Afterwards, use the same procedure to create the tentacles along with the Bend modifier, set the angle to 93.5 degrees and bend the tentacles on the X-axis (see figure 11).

Figure 11. Model with Tentacles
This is the final 3-dimensional model of the Command Core.

Figure 12. Command Core Three-Dimensional Model
3.4.2 White Hornet

This playership consists of a big cannon which makes up most of the ship able to fire huge bursts of destructive energy, connected to a mobile body. The ship has two cockpits. The concept art (see figure 13) has been made by Mark Chapman.

Figure 13. White Hornet Concept Art
This model has been created using many box shapes. Select a box modeling to edit a box primitive that is converted to an editable polygon. In the Edit Polygons rollout, click the Settings button for Bevel. Enter height equals 9.1 and the outline amount equals -7. The Bevel on the surface catches the light to enhance the look of the object (see figure 14).

Figure 14. Making White Hornet Using Box with Cylinder
Create an oil tank from the Extended Primitives tab using the following attributes: Radius=Height=18, sides=9 and height Segments=1. Convert it to an Editable Polygon and duplicate it. To make the ship's wings, use a box shape from the Standard Primitives tab, move the vertex until you get the desired shape, duplicate it, and use the Mirror option on the X-axis so that both object be symmetrical (see figure 15).

Figure 15. Making of Wings
Finally, use the 3-dimensional Boolean operations, available from the geometry tab in order to attach all primitives together. Here is a picture (figure 16) showing our White Hornet.

Figure 16. White Hornet Three-Dimensional Model
3.4.3 Bulldog

The Bulldog is an enemy ship. It is a pretty complex model. Figure 17 shows the concept art for the Bulldog done by Juan Moreno (Concept Artist).

Figure 17. Bulldog Concept Art
Create a Sphere of 10 segments and set the radius to 84 to make the body. Convert the sphere shape to an Editable Polygon and from the top view, delete the right part by selecting the corresponding vertices. Use the Symmetry Modifier on the X-axis. Apply the same procedure to create the bulldog's arms and head (see figure 18).

Figure 18. Making Bulldog Using Spheres and Cylinders
Duplicate every additional element by holding down the Shift button and dragging the mouse. Then, use the Mirror function to make sure that they are all symmetrical to one another (see figure 19).

Figure 19. Making Bulldog Using Duplicate Method
Use the Boolean Operations to get only one object. Figure 20 shows the final model of the bulldog.

Figure 20. Bulldog Three-Dimensional Model
3.4.4 Floater

It is a small, fixed-winged spaceship that is always flying within a pretty tight formation with the same type of ship. There is a cannon located on top of the main body (see figure 21). The concept art has been done by Marc Chapman.

Figure 21. Floater Concept Art
Create a box with the following attributes: Length=78, Width=264 and Height=28. Convert it to an Editable Polygon and select the polygon on the top. Apply the Bevel function to it with Height=5 and Outline Amount= -3. Figure 22 shows the box beveled.

Figure 22. Making Floater with Bevel
Then, create another box, convert it to an Editable Polygon and move the vertices (on the left) down. Duplicate that shape (see figure 23).

Next, create half of a sphere on top of the body with Radius=13, Segments=8, and Hemisphere=0.50. Make a bigger sphere with Radius=43, Segments=10, Hemisphere=0.505 and attach it to the body of the Floater. The last shape is an 8-side cylinder, with Radius=2, and Height=24.

Figure 23. Floater in 3D Studio Max
Connect all the different shapes together using the Boolean Operations. Figure 24 shows the final model.

Figure 24. Floater Three-Dimensional Model
3.4.5 Mound

This spaceship houses two rocket pods on its back. Very easy to maneuver, it can fly from one side to another while loading its rocket pod (see figure 25). The concept art has been made by Mark Chapman.

Figure 25. Mound Concept Art
Create a cylinder and change the number of sides to 8, the radius to 72, and the Height to 10. Figure 26 shows the cylinder shape with the Bevel function applied to it.

Figure 26. Making Mound with Beveled Cylinder
Next, add a box (Length=32, Width=105, Height=28) and cylinder (Radius=17, Height=92 and with 8 sides). Convert them to Editable Polygons. Modify the vertices to get the desired shape. Apply Bevel to cylinder. Figure 27 shows the model of the cylinder and box on top of the Mound's body.

Figure 27. Making Mound's Body
Figure 28 shows the final three-dimensional model of the Mound.

Figure 28. Mound Three-Dimensional Model
3.4.6 Hive

This spaceship is a carrier primarily used for defense between ships. Its outstanding octagonal body houses smaller other small spaceships (Floater or Gnat) for quick deployment (see figure 29). The concept art has been made by Mark Chapman.
It is made of 4 cylinders connected together using the Boolean Operations. The Bevel function is applied on each one of the cylinders (see figure 30).

Figure 30. Making of Hive
Figure 31 shows a 3-dimensional view of the Hive.

Figure 31. Hive Three-Dimensional Model
3.4.7 Stinger

It is very small. It has a cannon fixed in place so that its base can rotate around easily. Create a sphere and change the Radius to 83, Segments to 12 and Hemisphere to 0.5. Create three identical 8-side cylinders with Radius=7, Height=163 and a bigger cylinder with Bevel applied to it (see figure 32).

Figure 32. Making of Stinger
After using the Boolean Operations, you obtain the following model (see figure 33).

Figure 33. Stinger Three-Dimensional Model
Each level is constrained to the following world coordinates. All time units are in seconds, and distance units are in world units. Figure 34 shows an example of it.

Figure 34. Game's World Coordinates
The default position of everything has z coordinate equal to 0. The camera position has z value set so that the screen includes 100 world units (-50 to 50) of width and 75 world units of height in the z equals 0 plane.

Figure 35 shows a general view of the game’s world from the origin to the end of level including the camera and the player positions that are relative to the gamePlayCenterPoint.

Figure 35. Game’s World View
4.1 Level Script

The level script will specify the location of game objects in the world. Lua is the scripting language used to structure the game runtime behavior. There are lots of tutorials available online. Lua can be easily incorporated into other applications. The following examples show and describe some of the Lua scripts used in the game.

4.1.1 EndOfLevel(x, y, z)

The EndOfLevel command is used in the level script to define the position to the world coordinates when the level ends; when the gamePlayCenterPoint arrives at the end of level point, the level will end. The end of level command corresponds to an end of level command in the C# code. The following shows the C# implementation that is invoked when the EndOfLevel function runs in the level script.

```csharp
public static void EndOfLevel(float x, float y, float z)
{
    Vector3 position = Adjust(x, y, z);
    EndOfLevel endOfLevel = new EndOfLevel(position);
    LevelManager.AddPendingGameObject(endOfLevel);
}
```
4.1.2 Music("01.mid")

Another example of a level script command is the music command. The following example shows how this command will be used to make the game play a music file called 01.mid.

The music file must be under the music folder in the content folder. The following are the music files used by the game. The mid extension means these files are in midi format.

- 01.mid
- 02.mid
- magic.mid
- try2.mid
- try3.mid
- try5.mid

Here is the code that shows how the music command has been implemented. When the command in the level files executes, they cause corresponding methods in the c# sharp program to run. The music command causes the running of the method called LuaMethods.
public static void Music(string filename)
{
    LevelManager.Music = 
    SoundEngine.CreateLoopedSound 
    ("Content/Music/" + filename);
}

4.1.3 Playership(x, y, z)

One more example of a level script command is the playership command. The following example shows how this command will be used to create an instance of the playership.

A player could be either RedSparrow, BlackScarab or WhiteHornet. RedSparrow(0, 10, 0) creates instance of Red Sparrow at the specified positions. No behaviors are assigned through this.

Here is the code that shows how the playership command "RedSparrow" has been implemented. When the command in the level files execute, they cause the method RedSparrow() in the LuaMethods class to run.

public static void RedSparrow(float x, float y, float z)
{
    Debug.Assert(LevelManager.Playership == null);
    Vector3 position = Adjust(x, y, z);
    RedSparrow redSparrow = new RedSparrow(position);
ApplyCurrentSettings(redSparrow);
LevelManager.AddPendingGameObject(redSparrow);
LevelManager.PlayerShip = redSparrow;
}

The following figure shows the RedSparrow model when the game is running.

![RedSparrow Model at Position (0, 10, 0)](image)

Figure 36. RedSparrow Model at Position (0, 10, 0)

Each model used in the game needs to be processed. This code shows how a mesh is processed to a model content that is optimal for runtime in C#. The model can be scaled and rotated.
class RedSparrowModelProcessor : ModelProcessor
{

public override ModelContent Process(NodeContent input, ContentProcessorContext context)
{

    MeshHelper.TransformScene(
        input,
        Matrix.CreateScale(0.04f) * 
        Matrix.CreateTranslation(0, 0, 0) * 
        Matrix.CreateRotationX(0) * 
        Matrix.CreateRotationY(0) * 
        Matrix.CreateRotationZ(0)
    );

    return base.Process(input, context);
}

Then, we need to load the model in the game. This code shows how the loading of the model has been implemented in C# using the ContentManager.

public class RedSparrow : PlayerShip
{

    public RedSparrow(Vector3 position)
    : base(position)
One more example of a level script command is the UserControlledBehavior() command. The following example shows how this command will be used to let the user control the selected player ship using the X-BOX 360 controller or the keyboard. Before creating an instance of the player ship, UserControlledBehavior() has to be called.

When the command in the level files execute, they cause the method UserControlledBehavior() in the LuaMethods class to run.
public static void UserControlledBehavior()
{
    behaviorFactoryList.Add((new UserControlledBehaviorFactory());
}

4.1.5 ClearBehaviors()

Declaring a behavior adds it to the back of a queue. The behavior in the front of the queue is the only one performed. ClearBehaviors() is another example of level script command that is used in the game. When the command in the level files execute, they cause the method ClearBehaviors() in the LuaMethods class to run.

ClearBehaviors() is used to clear all behaviors from the current behavior queue. It should be used after assigning UserControlledBehavior() to the playership or after some other behavior to the enemy group so only that particular enemy group does a specific action.

public static void ClearBehaviors()
{
    behaviorFactoryList.Clear();
}
4.1.6 PowerUp(x, y, z, dx, dy, dz)

PowerUp(x, y, z, dx, dy, dz) is another example of level script command used in the game. When the command in the level files execute, they cause the method PowerUp() in the LuaMethods class to run. The playership will collect different power-ups, allowing the use of several different weapons. The three main weapons are homing missiles, a laser, and a spread shot.

The following figure (figure 37) shows the blackscarab shooting the laser after picking up the power up (four little squares on the center of the screen).

Figure 37. BlackScarab Shooting Laser Missile
4.2 Game Menu Content

4.2.1 Game Menu Sequence

The game menu is the first screen displayed after starting the game. This figure below shows the sequence of the game. The circle represents the screen options and the box represents the screen. We have three self-explanatory options displayed: Play Game, Credits, and Exit.

Figure 38. Game Sequence
4.2.2 Game Menu UML Diagram

Here is a UML diagram (class diagram) that describes the structure of the game by showing the game's classes, their attributes, and the way the different classes are connected. Attributes and operations have the following representation:

- "+" for public
- "#" for protected
- "-' for private.
Each attribute is shown with its name, type, initial value and other properties. Similarly, a class’s operation is shown with its name, and other parameters (see figure 39).

![Class Diagram of Game Screen]

Figure 39. Class Diagram of Game Screen
4.2.3 Game Menu Code Sample

The following passage describes the implementation of some of the game menu classes done in C#.

4.2.3.1 Main Menu Screen. The main menu screen is first displayed when the game starts up. It offers three options: Play Game, Credits and Exit. These options are self-explanatory. Here is the code that shows the implementation of the main menu screen.

```csharp
public class MainMenuScreen : MenuScreen
{

    public MainMenuScreen()
    {

        MenuEntries.Add("Play Game");
        MenuEntries.Add("Credits");
        MenuEntries.Add("Exit");
    }
}
```
This code handles the response to the user menu selections. We have three choices:

- If the menuEntryIndex is 0, we go to the player selection
- If the menuEntryIndex is 1, we go to the credits screen.
- If the menuEntryIndex is 2, we exit the game.

protected override void
ProcessMenuSelect(int menuEntryIndex)
{
    if (menuEntryIndex == 0)
    {
        StartTransitionOff();
        PlayerShipSelectionScreen.Instance.StartTransitionOn();
    }
    else if (menuEntryIndex == 1)
    {
        StartTransitionOff();
        CreditsScreen.Instance.StartTransitionOn();
    }
    else if (menuEntryIndex == 2)
    {

GameManager.ExitGame();
}
}

MainMenuSceen is a derived class that inherits from MenuScreen class.

4.2.3.2 Menu Screen. This code is done in the update method of the MenuScreen class. It shows how the user input is handled. Basically, it responds to the user input, changing the selected entry and accepting or cancelling the menu.

    if (InputState.MenuUp && selectedEntry > 0)
    {
        --selectedEntry;
    }

    if (InputState.MenuDown && selectedEntry < menuEntryList.Count - 1)
    {
        ++selectedEntry;
    }
    // We can accept the menu
    if (InputState.MenuSelect)
    {
        ProcessMenuSelect(selectedEntry);
This code shows how each menu entry is drawn in turn. The selected entry has an aquamarine color with an animating size and the other entries are white.

```csharp
for (int i = 0; i < menuEntryList.Count; i++)
{
    // the other entries are white.
    Color color = Color.White;
    float scale = 1;

    // selected entry : aquamarine + animating size.
    if (itemsAreSelectable && selectedEntry == i)
    {
        color = Color.Aquamarine;
        scale = selectedMenuItemScale;
    }

    // fades text out while transitioning.
    color = new Color(color.R, color.G, color.B, TransitionAlpha);
}
```
// text is centered in the middle

Vector2 origin =

new Vector2(0, GameManager.ArialFont.LineSpacing / 2);
4.2.3.3 Background Screen. The background screen is behind the game menu screen. The ContentManager takes care of loading and unloading the background screen before moving from the selected menu screen to the actual game. The following code shows the implementation of the background screen.

It loads the graphics content for this screen.

```csharp
public static void Initialize(ContentManager contentManager)
{
    backgroundTexture =
    contentManager.Load<Texture2D>("game_menu");
}
```

This code shows the implementation of drawing the background screen in the game.

```csharp
public override void Draw(GameTime gameTime)
{
    Viewport viewport =
    GameManager.GraphicsDeviceManager.
    GraphicsDevice.Viewport;
    Rectangle fullscreen = new
    Rectangle(0, 0, viewport.Width, viewport.Height);
}
```
5.1 Publishing an Application

There are a number of steps to follow in order to publish an application. The best way is to follow the Publish Wizard and specified where the project is to be published. The application could be published either:

- to the Web
- to a file share, or
- to a CD-ROM or DVD-ROM

During the publishing process, it is necessary to sign the application using a public/private key; otherwise the game will not run.

Here are the steps required to publish the application to a CD-ROM or DVD-ROM [7]:

- Choose Project Properties
- Click Publish tab
- Specify how the users will install the application and whether or not it will check for updates.
After that, the finished game obtained is to be shared with any windows user. Normally, XNA Game Studio Express does not need to be installed on their computers to run the game. However, they need to have the following requirements [7]:

- .Net Framework 2.0 v2.0.50727
- video card that supports shader profile PS_2_0 (at least Shader Model 1.1)
- some files from DirectX 9.0c (dsetup.dll, dxsetup.exe, dxupdate.cab, etc)
- at least Windows XP Service Pack 2.

5.2 Problems Running CyberShot on Users' Computers

Below is the summary of the errors that were detected when trying to install CyberShot:

- Could not find a part of the path 'C:\Documents and Settings\ckoffi\Desktop\CyberShot_1_0_0_3\CyberShot.exe.manifest'.
- C:\Documents and Settings\ckoffi\Desktop\CyberShot.application': "The system cannot find the file specified."
- Some applications will not run correctly if you don't have administrative privileges on the computer.
• Downloading file C:/Documents and Settings/ckoffi/Desktop/CyberShot_1_0_0_2/Content/Models/Bat/Bat.xnb.deploy did not succeed.

• Could not find a Direct3D device that has a Direct3D9-level driver and supports pixel shader 1.1 or greater.

5.3 Tentative Solutions

Many approaches have been used in an attempt to solve these problems.

The following list describes the tentative solutions to the problems stated earlier.

• Concerning the missing files, I looked in the designated folders and the files were effectively there. Whenever I created a new CyberShot application, there was always an error message saying that the files were missing. More specifically, there were files with the extensions xnb.

• Concerning the graphics card, I replaced it with the NVIDIA GeForce4 MX 440 which was supposed to solve the problem, but it did not. It seems like XNA requires a good graphics card that can support pixel shaders.
• Concerning the administrative privileges, an account has been setup for me.
• However, I was unable to run CyberShot due to the graphics card problem stated above.
CHAPTER SIX

CONCLUSION

6.1 Conclusion

Being part of this summer project has been such a wonderful and advantageous experience. I gained some skills that will surely help me in the future. I learned to be responsible for working on the assigned tasks, and to collaborate with other team members on a game development project in a safe and pleasant environment.

In order to develop this game, game designer, programmers, musicians, concept artists and three-dimensional modelers had to work together to ensure that the game produced is an application that everyone would enjoy to play. CyberShot was developed in C# using the XNA Framework, and 3D Studio Max.
6.2 Future Directions

CyberShot is a game application that can be further extended easily. So far, we just have the first level done with a certain number of playerships and enemyships. Students interested in learning game development using the same tools (XNA Framework and 3D Studio Max), could improve the game by adding more features into it. For example, incorporating the three-dimensional models of the enemyships for the second and third levels, creating more textures, adding some AI for the enemyships. Since CyberShot has not been tested on the console game, X-BOX 360, it will be a good idea to do some research about the requirements to make it run on the X-BOX 360, and make sure that the vibration feature is activated on the game controller.
REFERENCES


