การพัฒนาเกมคอมพิวเตอร์ผจญภัย 3 มิติ

DEVELOPMENT OF 3D ADVENTURE GAME

ไกรวิทย์ ลอออรรถพงศ์
กวินทร์ พิพัฒน์กุล
ยุทธพงษ์ ลอยทอง

ปัญหาพิเศษนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิทยาศาสตรบัณฑิต
ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์
คณะวิทยาศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2548

DEVELOPMENT OF 3D ADVENTURE GAME

KRAIWIT LAORATTHAPONG

KAWIN PIPUTKUL

YUTTAPONG LOYTHONG

A SPECIAL PROJECT SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIRMENT FOR THE DEGREE OF BACHELOR OF SCIENCE

DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE

FACULTY OF SCIENCE

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

ACADEMIC YEAR 2005

หัวข้อปัญหาพิเศษ	การพัฒนาเกมผจญภัยสามมิติ 3D GAME ADVENTURE	
ชื่อนักศึกษา	นายกวินทร์ พิพัฒน์กุล	45050452
	นายกรวิทย์ ลอออรรถพงศ์	45050458
	นายยุทธพงษ์ ลอยทอง	45050509
ภาควิชา	คณิตศาสตร์และวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์	
สาขาวิชา	วิทยาการคอมพิวเตอร์	
ปีการศึกษา	2548	
อาจารย์ที่ปรึกษา	รศ.ธีรวัฒน์ ประกอบผล	

ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง อนุมัติให้นับปัญหาพิเศษนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิทยาศาสตรบัณฑิต สาขาวิชาวิทยาการคอมพิวเตอร์ ประจำปีการศึกษา 2548

คณะกรรมการสอบ		ลายมือชื่อ
ประธานกรรมการ	ผศ.กฤษฎา บุศรา	
กรรมการ	อ.ศังกรศรัณย์ ล่องชูผล	
กรรมการและอาจารย์ที่ปรึกษา	รศ.ธีรวัฒน์ ประกอบผล	

(รองศาสตราจารย์ ดร.วีระ บุญจริง)

หัวหน้าภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์

หัวข้อปัญหาพิเศษ	การพัฒนาเกมผจญภัยสามมิติ	
ชื่อนักศึกษา	นายกวินทร์ พิพัฒน์กุล	45050452
	นายไกรวิทย์ ลอออรรถพงศ์	45050458
	นายยุทธพงษ์ ลอยทอง	45050509
ปริญญา	วิทยาศาสตรบัณฑิต	
ภาควิชา	คณิตศาสตร์และวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์	
สาขาวิชา	วิทยาการคอมพิวเตอร์	
ปีการศึกษา	2548	
อาจารย์ที่ปรึกษา	รศ.ธีรวัฒน์ ประกอบผล	

บทคัดย่อ

เกมที่สนุกคือเกมที่มีการติดต่อกับผู้เล่นโดยใช้รูปลักษณ์ใหม่ เช่น ความแปลกใหม่ ความท้าทาย สำหรับโครงงานนี้คณะผู้ศึกษาได้ทำการศึกษาเทคนิคต่างๆจากโปรแกรมเกมหลายเกม โดยเกมนี้ถูกพัฒนาโดยใช้ภาษา C#, DirectX และ 3D Studio Max โดยสนับสนุนผู้เล่นหลายคนบนระบบเครือข่าย

Special Project Title	DEVELOPMENT OF 3D GAME ADVENTURE	
Students	Mr. Kawin Pipatkul	45050452
	Mr. Kraiwit Laoatthapong	45050458
	Mr. Yuttapong Loythong	45050509
Degree	Bachelor of Science	
Department	Mathematics and Computer Science, Faculty of Science	
Programme	Computer Science	
Academic Year	2005	
Special Project Advisor	Assoc.Prof.Teerawat Prakobphon	

ABSTRACT

An enjoyable game is an interactive game with interesting features such as new styles and challenges. This project delevoped such a game using techniques collecting from various game products. The new game delevoped using C#, DirectX and 3D Stuidio Max. It supports multiplayer mode over a network.

กิตติกรรมประกาศ

ในการทำปัญหาพิเศษเรื่องการพัฒนาเกมคอมพิวเตอร์ผจญภัยสามมิติ สำหรับการเล่นเป็นทีมสามารถสำเร็จลุล่วงได้ด้วยดี ด้วยความช่วยเหลือและความร่วมมือจากหลายๆท่านคณะผู้จัดทำต้องขอขอบพระคุณบุคคลที่มีส่วนช่วยให้การทำงานครั้งนี้เสร็จไปได้ด้วยดี โดยเฉพาะอย่างยิ่งอาจารย์ที่ปรึกษาปัญหาพิเศษฉบับนี้ คือ รศ.ธีรวัฒน์ ประกอบผล ที่กรุณาให้คำแนะนำแนวคิดต่างๆ ในการทำปัญหาพิเศษนี้ ดูแลเอาใจใส่ และให้การสนับสนุนทาง ด้านซอฟท์แวร์ รวมทั้งเป็นผู้ตรวจสอบความถูกต้องของโครงงานปัญหาพิเศษฉบับนี้

นอกจากนี้คณะผู้จัดทำต้องขอขอบบุคคลที่เกี่ยวข้องทุกฝ่าย ที่ทำให้การทำปัญหาพิเศษนี้สำเร็จลุล่วงไปด้วยดี รวมทั้งบิดา มารดา ที่ให้ความ ช่วยเหลือในด้านต่างๆ ไว้ ณ ที่นี้

คณะผู้จัดทำ

มีนาคม 2549

สารบัญ

หน้า

สารบัญ (ต่อ)

สารบัญ (ต่อ)

สารบัญ (ต่อ)

สารบัญตาราง

สารบัญรูปภาพ

สารบัญรูปภาพ (ต่อ)

รูปที่ หน้า

สารบัญรูปภาพ (ต่อ)

รูปที่ หน้า

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มาของปัญหา

ปัจจุบันประเทศไทยต้องนำเข้าซอฟต์แวร์เกมส์จากต่างประเทศเป็นจำนวนมาก ทำให้ประเทศขาดดุลการค้า อีกทั้งเกมส์ที่นำเข้าส่วนใหญ่เป็นแนวต่อสู้ซึ่งทำให้ผู้เล่นที่ส่วนใหญ่เป็นเด็กอยากเลียนแบบซึ่งอาจทำให้เกิดอันตรายได้ สมาชิกในกลุ่มจึงเล็งเห็นที่จะสร้างเกมส์ผจญภัย 3 มิติซึ่งเป็นเกมส์ที่ทำให้ผู้เล่นได้ใช้ความคิดสร้างสรรค์ในการแก้ปัญหาเฉพาะหน้าให้กับตัวละ ครในเกมส์ที่ผู้เล่นได้ทำการเลือกไว้

ลักษณะเกมส์จะเป็นการผจญภัยระหว่างในป่าของมนุษกับป่าของภูติผี ซึ่งผู้เล่นสามารถเลือกตัวละครได้ว่าจะเล่นเป็นฝ่ายใด ซึ่งผู้เล่นแต่ละฝ่ายจะต้องช่วยกันทำลายฐานทัพของฝ่ายตรงข้าม เกมส์จะสิ้นสุดเมื่อมีฝ่ายใดฝ่ายหนึ่งสามารถยึดธงของฝ่ายตรงข้ามได้

1.2 วัตถุประสงค์

1. เพื่อศึกษาทฤษฎีและหลักการเขียนโปรแกรมเกม
2. ผู้เล่นเกมส์ได้ใช้ความคิดสร้างสรรค์ในการแก้ปัญหาเฉพาะหน้า
3. ศึกษาการพัฒนาซอฟต์แวร์ในรูปแบบกราฟิก และเกมคอมพิวเตอร์

1.3 ขอบเขตของการศึกษา

1.3.1 ระบบที่พัฒนาขึ้นจะต้องดำเนิน ภายใต้โปรแกรม Direct X

1.3.2 โปรแกรมเกมที่ทำงานภายใต้ระบบปฏิบัติการ Windows

1.3.3 เป็นเกมสามมิติแนวเดินโจมตีมุมมองบุคคลที่สาม (3D Third Person Shooting)

1.3.4 มีการพัฒนาความสามารถของตัวละครเมื่อสามารถกำจัดฝ่ายตรงข้ามได้

1.3.5 เมื่อตัวละครตายมีการหน่วงเวลาในการเกิด

1.3.6 มีดนตรีและเสียงประกอบในระหว่างเล่นซึ่งจะช่วยเพิ่มความน่าติดตามยิ่งขึ้น

1.4 ประโยชน์ที่ได้รับจากการศึกษา

1.4.1 เพิ่มทักษะในการเขียนโปรแกรมเชิงวัตถุ

1.4.2 เรียนรู้และเข้าใจ การใช้เครื่องมือในการพัฒนาเกม

1.4.4 ได้แนวคิดการออกแบบเกมส์ให้ถูกใจผู้เล่น

1.4.5 ฝึกการทำงานเป็นทีม

1.4.6 เพิ่มทักษะในการใช้คณิตศาสตร์ร่วมในการเขียนโปรแกรม

1.5 ขั้นตอนของการศึกษา

1.5.1 ออกแบบเนื้อหาของเกม

1.5.2 ศึกษา DirectX9 SDK เพื่อนำมาใช้ในการพัฒนาโปรแกรมเกม

1.5.3 จัดการทรัพยากรต่างๆ ไม่ว่าจะเป็นภาพ เสียง ดนตรี เนื้อเรื่องและอื่นๆให้พร้อมเพื่อที่จะ นำมาใช้ในการเขียนโปรแกรม

1.5.4 พัฒนาโปรแกรมเกมตามที่ออกแบบไว้

1.5.5 พัฒนาโปรแกรมเกมเป็นเวอร์ชั่นทดลองก่อน เพื่อให้ผู้ทดสอบได้ลองเล่นเพื่อช่วยหาจุดบกพร่อง ของโปรแกรมเพื่อนำข้อบกพร่องนั้นกลับมาปรับปรุงแก้ไขเกมให้มีความสมบูรณ์มากยิ่งขึ้น

1.5.6 ทำเป็นเวอร์ชั่นสมบูรณ์

1.5.7 จัดทำส่วนของการคิดตั้งโปรแกรม และบันทึกลงสื่อ

1.6 อุปกรณ์ที่ใช้ทำในปัญหาพิเศษ

1.6.1 เครื่องคอมพิวเตอร์ PentiumIV 2.56 GHz

1.6.2 หน่วยความจำ 512 MB

1.6.3 Harddisk 40 GB และ Removable rack

1.6.4 Microsoft Visual C#

1.6.5 DirectX 9 SDK

บทที่ 2
หลักการและทฤษฎีที่เกี่ยวข้อง

2.1 Microsoft Visual C# .NET

ภาษา C# เป็นภาษาที่ถูกออกแบบมาเพื่อรองรับการทำงานในยุค .NET โดยมีแนวของภาษาเป็นแบบของการเขียนโปรแกรมเชิงวัตถุสมัยใหม่ (Modern Object Oriented Programming) โดยที่จุดยืนของภาษา C# จะอยู่ที่การอาศัยไวยากรณ์ที่ปรับปรุงมาจากภาษา C/C++ ร่วมกับความง่ายของภาษา Visual Basic โดยแนวความคิดของการเขียนโปรแกรมแบบ Modern OOP เกิดจากการที่ไมโครซอฟต์พัฒนาคลาส (CLASS) ต้นแบบต่างๆที่เรียกว่า Base Class Library แล้วนำมาจัดหมวดหมู่ให้เป็นระเบียบ เมื่อต้องการเรียกใช้งานคลาสใดก็จะอาศัยเนมสเปซ (Name Spaces System) เข้ามาช่วยในการระบุคลาสต้นแบบต่างๆ เพื่อให้ผู้พัฒนาสามารถนำออบเจ็กต่างๆ ที่อยู่ในคลาสนั้นๆ ออกมาใช้งานได้ นอกจากนั้นภาษา C# ยังสามารถเรียกใช้คอมโพเนนต์ (Components) ซึ่งเป็นชิ้นส่วนทางซอฟต์แวร์มาประกอบเป็นโปรแกรม ทำให้ลดเวลาในการพัฒนาโปรแกรมได้เป็นอย่างดี

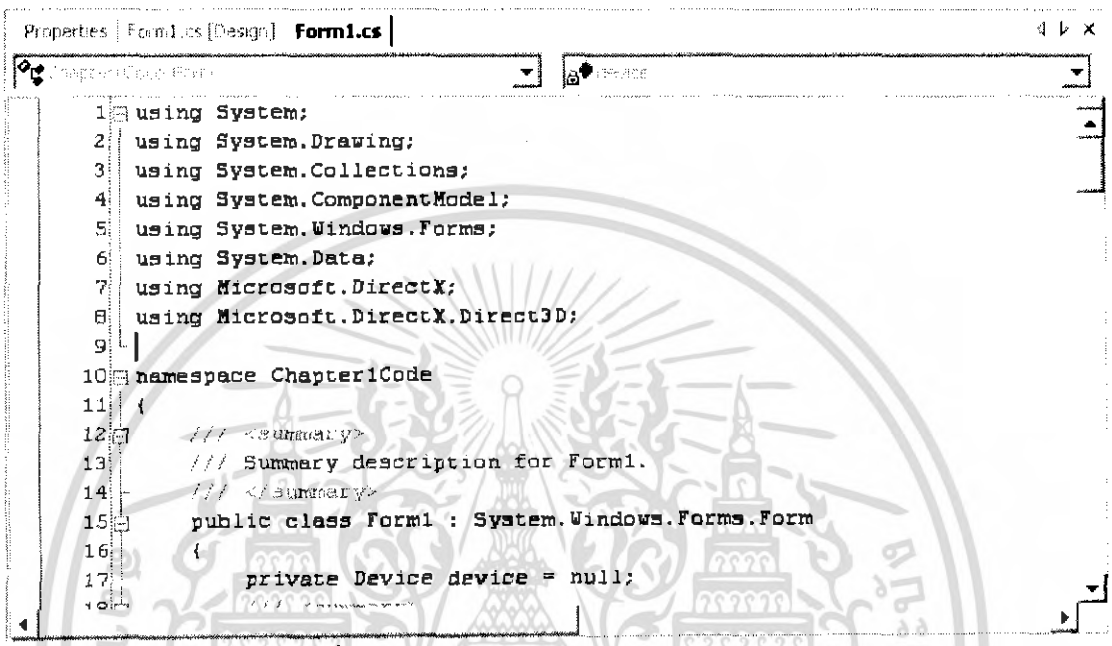
สถาปัตยกรรม .NET (.NET Framework)

คือกรอบการทำงานของการเขียนโปรแกรมที่ไมโครซอฟต์คิดขึ้นมา เพื่อรองรับการติดต่อสื่อสาร เพื่อแลกเปลี่ยนข้อมูลระหว่างกัน หรือแลกเปลี่ยนข้อมูลระหว่างแพลตฟอร์มให้มีความสมบูรณ์ยิ่งขึ้น โดยอาศัยภาษา XML (Extensible Markup Language) ทำหน้าที่เป็นตัวกลางในการแลกเปลี่ยนข้อมูลระหว่างแพลตฟอร์มไฟล์ของฐานข้อมูล

Namespaces

ระบบ Namespaces ทำหน้าที่รวบรวมออบเจ็กต้นแบบต่างๆ เข้าด้วยกันแล้วแบ่งออกเป็นหมวดหมู่เป็นสัดส่วน แล้วกำหนดให้เป็นออบเจ็กหลักของระบบเพื่อให้ผู้พัฒนาโปรแกรมสะดวกในการเรียกใช้ Namespaces ชั้นบนสุดในภาษา C# เรียกว่า System มีเพียงตัวเดียว ประกอบด้วยเนมสเปซย่อยๆ สืบทอดลงมามากมายมหาศาล แยกเป็นกลุ่มๆ เป็นชั้นๆ โดยที่แต่ละลำดับชั้นจะคั่นด้วยเครื่องหมาย . เช่น

System.Windows.Forms , System.Data , System.Web เป็นต้น เมื่อไดก็ตามที่เราต้องการเรียกใช้งานออบเจ็กต้นแบบใดๆ เราต้องระบุชื่อเนมสเปซ เพื่อให้การเรียกใช้งานออบเจ็กดังกล่าว ไม่เกิดข้อผิดพลาดขึ้นมา

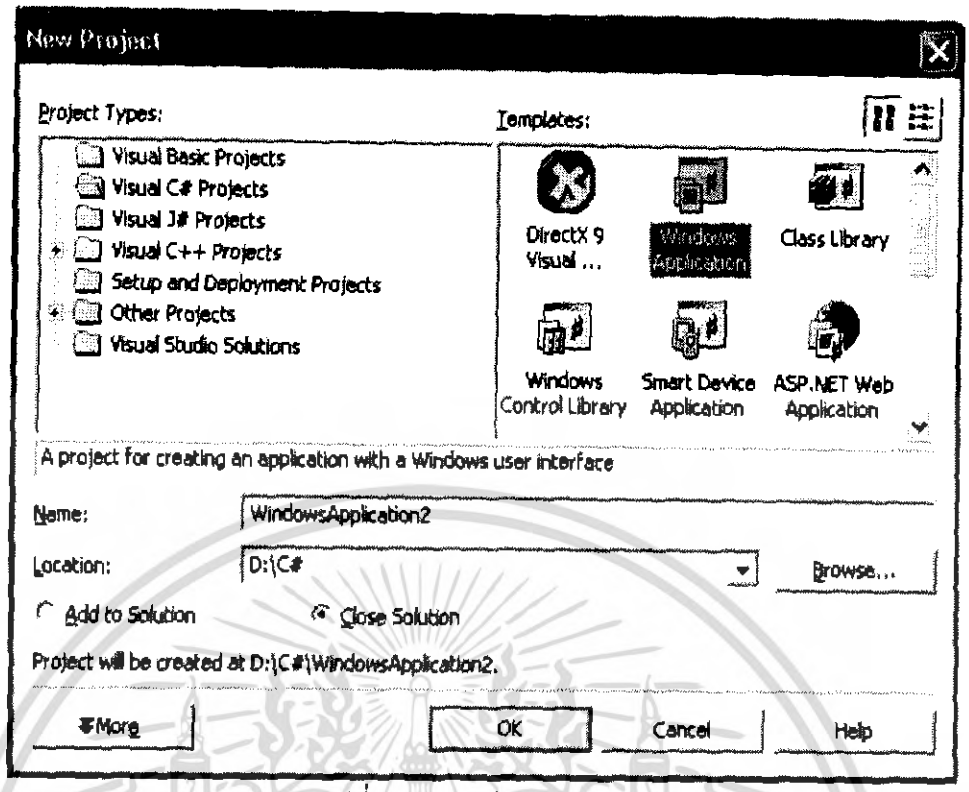


รูปที่ 2.1 การพัฒนา C# แบบ Windows

จากรูปข้างบนเป็นการพัฒนาแอพลิเคชั่นภาษา C# แบบ Windows จะมีเนมสเปซส่วนหนึ่งถูกระบุเข้ามาโดยอัตโนมัติ โดยการใช้คำสั่ง using ซึ่งคล้ายกับการอิมพอร์ต (import) คลาสที่เก็บอยู่ในแพ็คเกจของภาษาจาวา

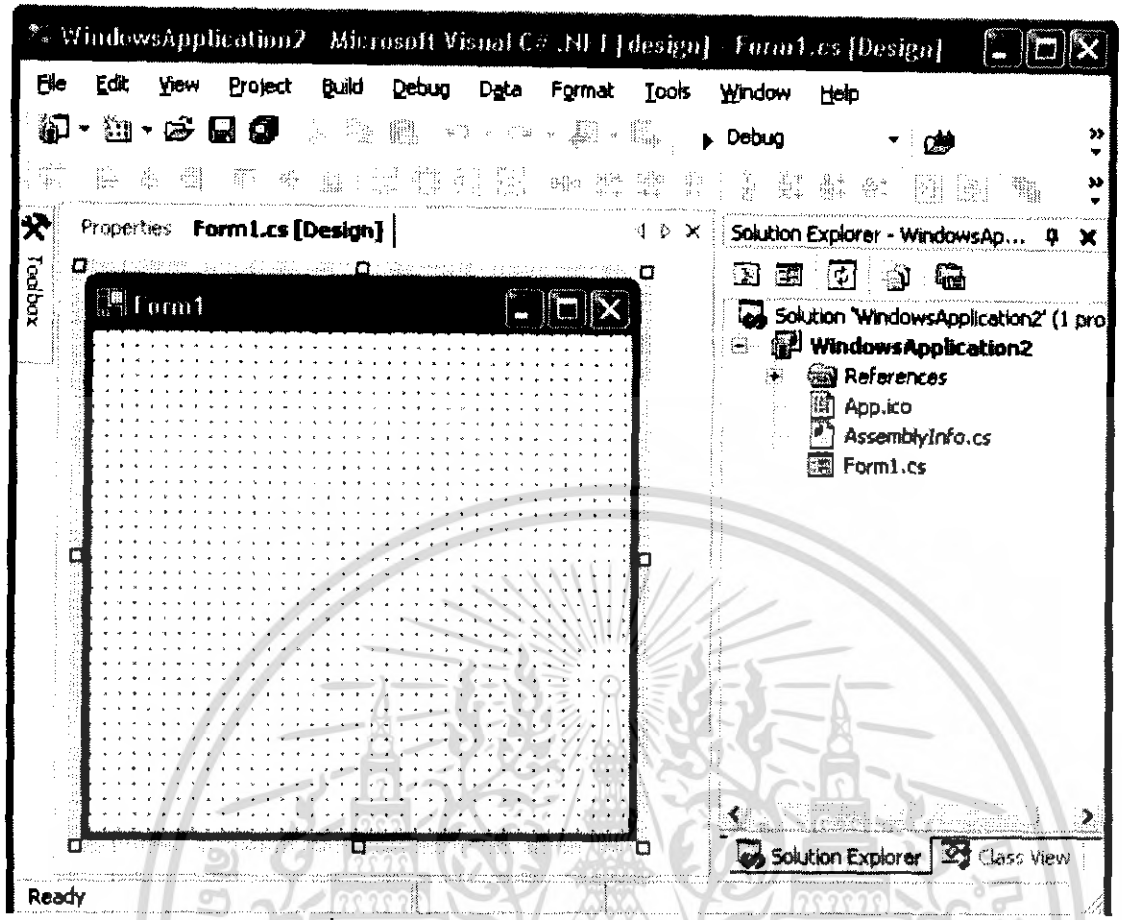
2.2 การเขียนโปรแกรมด้วย Visual C# .NET บน Window

คือโปรแกรมที่แสดงผลแบบในวินโดวส์ หรือแบบวิชวล ซึ่งจะเป็นการประกอบออบเจ็กต์หรือคอนโทรลต่างๆ (เช่น ปุ่ม เท็กซ์บ็อกซ์ รูปภาพ) ลงในฟอร์ม โดยการสร้างโครงงาน (Project) แบบวินโดวส์ฟอร์มแอปพลเคชั่นมีขั้นตอนการทำงานเป็นลำดับขั้นดังต่อไปนี้



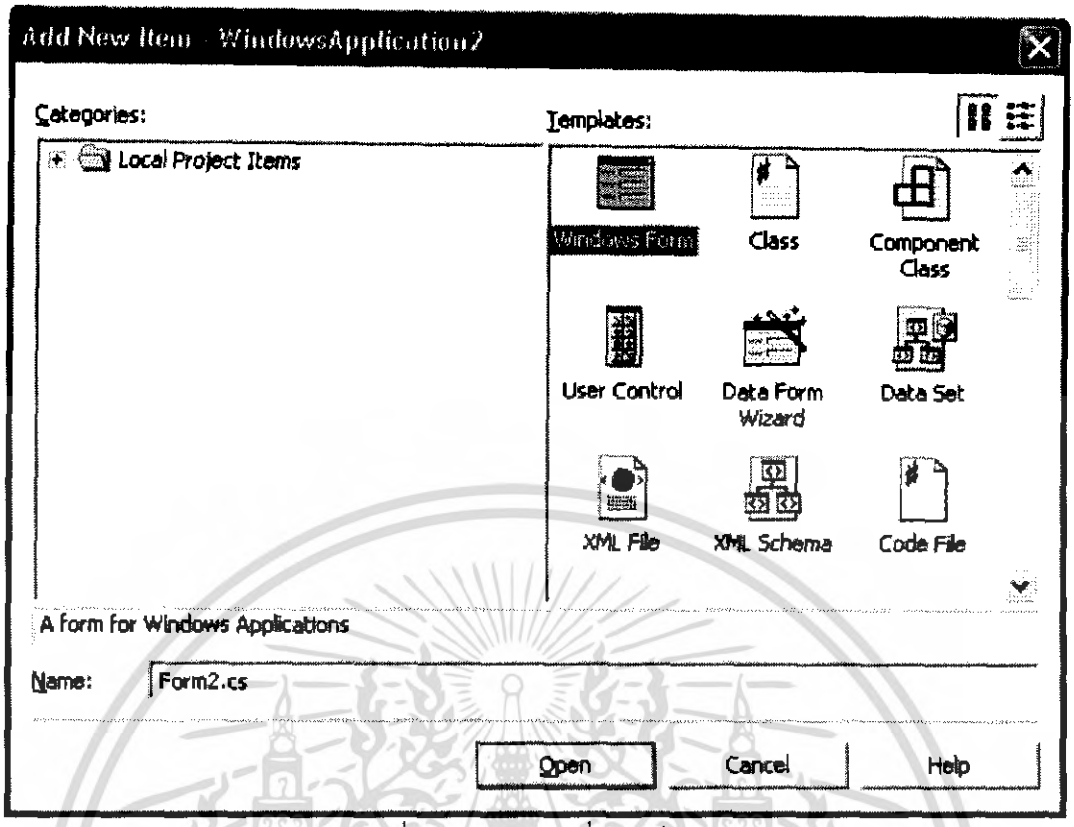
รูปที่ 2.2 การสร้าง Project

1. ที่เมนูบาร์ ให้เลือก File => New => Project จะขึ้นหน้าจอดังภาพข้างบน
2. ที่ Project Types ให้เลือก Visual C# Projects ส่วนที่ Templates ให้เลือก Windows Application
3. ใส่ชื่อโปรเจกต์ และ ตำแหน่งที่เก็บโปรเจกต์ จากนั้นกด OK เราก็จะได้โปรเจกต์พร้อมไฟล์ ภาษา C# ที่ชื่อว่า Form1.cs มาด้วย



รูปที่ 2.3 ผลลัพธ์หลังจากสร้างโปรเจกต์เสร็จแล้ว

4. ที่แทบ Design ของ Form1.cs ให้เราดับเบิ้ลคลิ๊กหรือคลิ๊กขวาแล้วเลือก View Code ก็จะปรากฏโค้ดของ Form1 เราก็สามารถเพิ่มเติมโค้ดที่เราต้องการลงไปได้
5. ในกรณีที่เราต้องการเพิ่มไฟล์ลงไปในโปรเจกต์ของเราให้เราเลือก File => Add New Item จะปรากฏหน้าต่างดังรูปข้างล่าง ให้เราเลือก Templates ที่ต้องการจากนั้นใส่ชื่อไฟล์แล้วกด OK



รูปที่ 2.4 แสดงการเพิ่มไฟล์

2.2.1 ฟังก์ชั่นและวิธีการสร้างฟังก์ชั่น

ฟังก์ชั่นหรือเมธอด คือส่วนของโปรแกรมที่สร้างขึ้นเป็นกระบวนความของโปรแกรมย่อยๆ ที่ประกอบด้วยคำสั่ง ตั้งแต่ 1 คำสั่งขึ้นไป โดยเราสามารถกำหนดค่าเริ่มต้นให้กับเมธอดเพื่อให้ฟังก์ชั่นทำงาน และเราอาจจะรับคืนค่าจากเมธอดก็ได้ โดยในภาษา C# เมธอดจะถูกเขียนอยู่ในคลาส

โครงสร้างพื้นฐานของการกำหนด หรือสร้างฟังก์ชั่นหรือเมธอดขึ้นมาเป็นไปตามรูปแบบนี้

```
Modifiers  returnType  methodName(parameterList)
{
        //statement(s)
}
```

รายละเอียดของแต่ละส่วนมีดังนี้

- Modifiers เป็นส่วนที่จะมีหรือไม่มีก็ได้ ได้แก่ public , protected และ private มีความหมายเหมือนการประกาศตัวแปรข้อมูลต่างๆ public คือที่ไหนก็เรียกได้

protected คือให้คลาสเดียวกันและคลาสย่อยของคลาสที่ประกาศไว้เรียกได้
private คือเฉพาะในคลาสเดียวกันเรียกได้

- returnType คือชนิดของข้อมูลที่เมธอดนั้นๆ จะคืนค่าออกมาเช่น int , Boolean , string เป็นต้น และสามารถคืนค่าได้เพียงค่าเดียวเท่านั้น หรือไม่ต้องการให้คืนค่าก็ได้ซึ่งก็ต้องใช้ void โดยภาษา C# ไม่มีการคืนค่าให้เองอย่างอัตโนมัติ เหมือนภาษา C
- methodName คือชื่อที่ใช้สำหรับอ้างอิงเมธอด ใช้ชื่อตามหลักเกณฑ์ของตัวอ้างอิงตามหลักการทั่วไปมักให้ชื่อขึ้นต้นด้วยตัวเล็ก ยกเว้นเมธอด Main ซึ่งเป็นเมธอดที่เริ่มการทำงานของโปรแกรม เพื่อแยกกับชื่อคลาสที่เราจะให้ขึ้นต้นด้วยตัวอักษรตัวใหญ่
- parameterList ในส่วนนี้ใช้สำหรับการผ่านค่าเข้ามาในเมธอด ถ้ามีหลายตัวอาจใช้ , คั่น อาจจะไม่มีก็ได้
- statement (s) คือคำสั่งต่างๆที่อยู่ในเมธอด จะทำงานเมื่อเมธอดถูกเรียกใช้ ภายใต้คำสั่งอาจมีการประกาศตัวแปรที่ใช้ภายในเมธอดได้ จะเรียกตัวเรียกตัวแปรแบบโกลบอลเหมือนในภาษาอื่นไม่ได้

โดยในการประกาศเมธอดนั้น เราต้องประกาศอยู่ในคลาสเท่านั้น ไม่สามารถประกาศไว้ในเนมสเปซหรือเมธอดด้วยกันเองได้

ตัวอย่างการประกาศและเรียกใช้เมธอด

```
public int sub(int a, int b)
{
        return  a – b;
}
```

เมื่อเราเรียกใช้เมธอดนี้โดยใช้คำสั่ง

```
int  subResult = sub(10, 4);
```

เราจะได้ว่าตัวแปร subResult จะเก็บผลลัพธ์เท่ากับค่า 6
เมื่อใดก็ตามที่เราใช้คำสั่ง return จะเป็นการออกจากเมธอดนั้นทันที แต่เราต้องคืนค่าออกไปด้วยเสมอ ยกเว้นถ้าเราประกาศไว้ว่า void เราสามารถใช้คำสั่ง return; เฉยๆได้เลย

2.3การเขียนโปรแกรมเชิงวัตถุ(Object Oriented Programming)

นิยามของการเขียนโปรแกรมเชิงวัตถุ คือ การมองทุกอย่างในโลกให้เป็นวัตถุ (objects) เรามาทำความเข้าใจกับแนวคิดนี้กันดังนี้ ทุกๆ สิ่งที่อยู่รอบตัวเราก็คือวัตถุ ซึ่งแต่ละชิ้นแต่ละอย่างก็มีชื่อเรียก เช่น

- เครื่องบิน ไม่ว่ายี่ห้อใด แบบใด ประเภทใด ก็เรียกว่าเครื่องบิน เช่น เครื่องบินไอพ่น เครื่องบินใบพัด
- โทรทัศน์ จะยี่ห้อใด แบบไหน เราก็เรียกวัตถุนี้ว่าโทรทัศน์

จากตัวอย่างเป็นออบเจกต์ที่อยู่ในโดเมนที่มีอยู่ในโลกเท่านั้น ถ้าเราต้องการจำลองสิ่งต่างๆ ให้อยู่ในเครื่องคอมพิวเตอร์ เราก็ต้องทำให้เกิดออบเจกต์ขึ้นในเครื่องคอมพิวเตอร์ ในขณะเดียวกันก็ต้องสร้าง Relationships และ Interactions ระหว่างออบเจกต์ประเภทต่างๆ ให้เกิดขึ้นในเครื่องคอมพิวเตอร์ด้วย และด้วยเหตุที่ว่าเราไม่สามารถนำเอา ออบเจกต์ในโลกของความเป็นจริงเข้าไปใส่ในเครื่องคอมพิวเตอร์ได้ สิ่งที่ต้องทำคือการใส่แนวคิด(Concept)ให้กับออบเจกต์แล้วจึงสร้างแบบจำลองของออบเจกต์ในโลกของความเป็นจริงนั้นๆ เพื่อนำไปใส่ไว้ในคอมพิวเตอร์ การให้แนวคิดกับออบเจกต์ต่างๆ นั้นจะถูกกำหนดโดยโดเมน เพราะเราจะให้แนวคิดกับออบเจกต์ในบางส่วนของออบเจกต์ที่เราสนใจเท่านั้น เช่น เมื่อเราสนใจเฉพาะแขน ขา ของคนเท่านั้น ดังนั้นเราจะให้ Concept ของคนว่าเป็นออบเจกต์ที่มี 2 แขน และ 2 ขา โดยเราไม่สนใจ หู ตา หรือ จมูก ของคนซึ่งถือว่าอยู่นอกโดเมน

ผลจากการให้ Concept กับ ออบเจกต์นั้นทำให้เกิดการจัดกลุ่มของออบเจกต์ขึ้น ซึ่งกลุ่มของออบเจกต์ที่ได้จากกระบวนการนี้เรียกว่า Abstract Objects หรือ Class นั่นเอง เช่น

- คลาสของรถถัง
- คลาสของเฮลิคอปเตอร์
- คลาสของบ้าน
- คลาสของเก้าอี้

โดย Class นั้นถือเป็นนามธรรม เราไม่สามารถทำให้คลาสดำเนินกิจกรรมใดๆ ได้เลยเช่น ประโยค รถวิ่งไปบนถนน ในทาง Object Orientation นั้นถือว่าไม่ได้เป็นเหตุการณ์ที่เกิดขึ้นจริงๆ เพราะคำว่า รถ จะหมายถึง แนวความคิดของการรวมเอาตัวถังรถ ล้อ และเครื่องยนต์มารวมกัน และ ถนน ก็คือแนวความคิดของสิ่งหนึ่งซึ่งอยู่บนพื้นโลกที่ทำไว้เพื่อให้ยานพาหนะทางบกวิ่งไปได้ แต่ถ้าพูดว่า รถยนต์ของนาย สุวรรณภูมิ วิ่งไปบนถนนลาดกระบัง นั่นหมายถึง รถยนต์ของนายสุวรรณภูมิซึ่งมีอยู่จริงในโลก และเป็นออบเจกต์ของคลาสรถยนต์ วิ่งไปบนถนนลาดกระบัง ซึ่งเป็นออบเจกต์ของคลาสถนน ดังนั้นถ้าเราต้องการให้เกิดกิจกรรมขึ้นในระบบ

คอมพิวเตอร์ เราจะต้องสร้างออบเจกต์ของคลาสต่างๆ ขึ้นในระบบคอมพิวเตอร์เสียก่อน เพื่อให้ออบเจกต์นั้นสามารถทำงานและดำเนินบทบาทของตัวเองได้

องค์ประกอบของออบเจกต์

จะต้องประกอบด้วย 3 องค์ประกอบคือ Attribute(s), Method(s) และ Unique Identity โดย

- Attribute หมายถึง สิ่งที่ใช้บรรยายคุณลักษณะต่างๆ ของออบเจกต์ซึ่งอยู่ในโดเมนที่เราสนใจ เช่น สีผิวและเพศของคนๆ หนึ่ง
- Method หมายถึงการกระทำที่ออบเจกต์สามารถกระทำได้หรือสามารถถูกร้องขอให้กระทำได้ โดยออบเจกต์ในโลกของ Object Orientation นั้นจะต้องเป็นออบเจกต์ที่ถูกกระทำ(Passive Objects) เท่านั้น เช่น นายจินดาขับรถยนต์วอลโว่เลขทะเบียน 1234 หมายความว่า รถยนต์วอลโว่คันนี้ต้องมีความสามารถในการขับ หรือมีเมธอดขับอยู่ในตัวเอง โดยนายจินดาเป็นเพียงผู้มากระตุ้น ให้รถยนต์คันนี้ขับ
- Unique Identity คือ การที่ออบเจกต์จะต้องมีความเป็นหนึ่งเดียว ไม่สามารถซ้ำกับออบเจกต์ตัวอื่นได้

นอกเหนือจากนั้นการเขียนโปรแกรมแบบ OOP ยังมีแนวความคิดที่สำคัญ 3 ประการคือ

1. Encapsulation เป็นข้อกำหนดที่ระบุว่า คลาสใดๆ ก็ตามที่สร้างขึ้นมา ผู้ใช้งานไม่จำเป็นต้องรู้ว่า ภายในคลาสมีกลไกการทำงานอย่างไร รู้แต่เพียงว่าคลาสนั้นๆ ทำหน้าที่อะไร มีคุณสมบัติอะไร ประกอบไปด้วยเมธอดอะไรบ้าง ข้อกำหนดนี้มีจุดประสงค์เพื่อปกป้องไม่ให้ผู้ใช้เข้าไปแก้ไข หรือยุ่งเกี่ยวใดๆกับวิธีการทำงานของคลาสต้นแบบนั่นเอง
2. Inheritance เป็นข้อกำหนดที่ระบุว่า คลาสต้นแบบที่ถูกสร้างขึ้นมา ซึ่งเรียกว่าคลาสแม่ สามารถที่จะทำสำเนาตัวมันเองได้ คลาสที่ได้คือคลาสลูกโดยที่การทำสำเนาดังกล่าว จะเรียกว่าการสืบทอดคลาส ส่งผลให้คลาสลูกมีความสามารถทุกอย่างเหมือนกับคลาสแม่ นอกเหนือจากนั้นเรายังสามารถเพิ่มความสามารถให้คลาสลูกเก่งกว่าคลาสแม่
3. Polymorphism เป็นข้อกำหนดที่ระบุว่า คลาสที่ถูกสำเนาขึ้นมาใหม่ ผู้ใช้งานสามารถแก้ไข หรือเปลี่ยนแปลงความสามารถของคลาสนั้นๆได้ด้วย ซึ่งเป็นข้อกำหนดที่ต่อเนื่องมาจากการทำ Inheritance

การออกแบบโปรแกรมเชิงวัตถุ

ในการเขียนโปรแกรมคอมพิวเตอร์ก็เช่นกัน เราต้องมองสิ่งที่อยู่ในโปรแกรมของเราให้เป็นเชิงวัตถุเพื่อที่จะได้ออกแบบวิธีการเก็บและจัดการข้อมูลที่เป็นอิสระต่อกัน แต่ก่อนที่จะเข้าสู่การเขียนโปรแกรมเชิงวัตถุในภาษา C# นั้นคล้ายกับตัวแปรแบบโครงสร้างในภาษา C การประกาศตัวแปรแบบโครงสร้างนี้ เป็นแนวคิดพื้นฐานที่จะนำมาใช้ในการสร้างคลาสต่อไป ลักษณะของตัวแปรแบบโครงสร้างจะช่วยแก้ไขปัญหาในเรื่องความซับซ้อนของตัวแปรได้ เพราะเป็นการเปลี่ยนวิธีการเขียนโปรแกรมโดยอาศัยชุดข้อมูลเป็นหลัก ยกตัวอย่างเช่น เรามีข้อมูลของ พนักงานบริษัทแห่งหนึ่ง ที่ต้องการจัดเก็บดังนี้

1. นายมานะ รหัสพนักงาน 0001 อายุ 32 เงินเดือน 8,900
2. นายปิติ รหัสพนักงาน 0002 อายุ 33 เงินเดือน 8,950
3. นางสาวมานี รหัสพนักงาน 0003 อายุ 27 เงินเดือน 8,500
4. นางสาวชูใจ รหัสพนักงาน 0004 อายุ 27 เงินเดือน 8,600

สมมุติว่าเราต้องการเขียนโปรแกรมให้เก็บข้อมูลของพนักงานเหล่านั้นเอาไว้ ถ้าเราใช้การเขียนโปรแกรมภาษา C ธรรมดา สามารถใช้อาเรย์แบบ 1 มิติ โดยการประกาศตัวแปรตามสิ่งที่เราต้องการจะเก็บ คือ รหัส, ชื่อ, อายุ, เงินเดือน จำนวน 4 คนดังนี้

```
string  name
string  id
int     age[4];
int     salary[4];
```

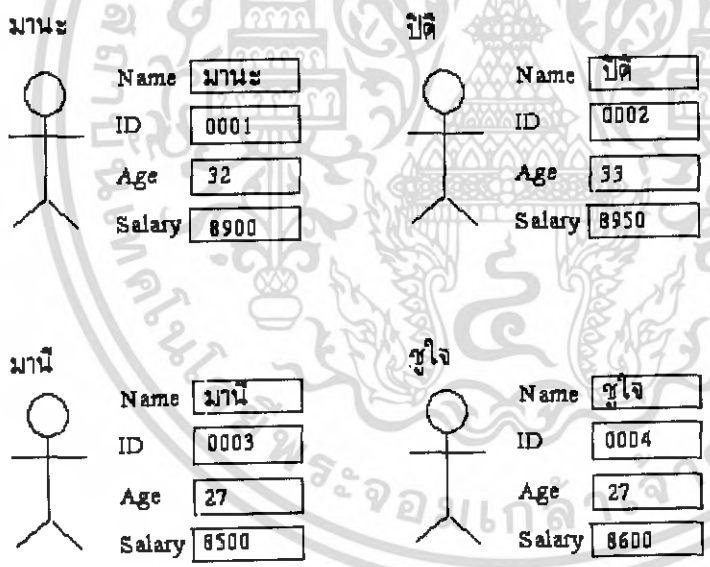
ถ้าเราจะนำเอาข้อมูลมาใส่ลงในตัวแปร name, id, age และ salary เราก็จะอาศัยลำดับในอาเรย์เข้ามาช่วยนั่นคือ คนที่ 1 ก็จะเก็บลงในอาเรย์ช่องที่ 0 ในทุกตัวแปรดังนี้

```
name = "มานะ";
id   = "0001";
age[0]    = 32;
salary[0] = 8900;
```

สำหรับคนที่ 2, 3, 4 ก็จะเก็บลงไปในอาเรย์ช่องที่ 1, 2 และ 3 ของทุกๆ ตัวแปรตามลำดับเช่นกัน จากตัวอย่างนี้เราสามารถสรุปได้ว่า การใช้อาเรย์ในการเก็บข้อมูลที่มีจำนวนมากๆ และมีตัวแปรหลายๆ ตัวค่อนข้างจะยุ่งยากพอสมควร เพราะเราจะใช้ลำดับของอาเรย์เป็นตัวกำหนดลำดับการเก็บข้อมูล ถ้าเราต้องการดูข้อมูลของคนที่ 1 เราก็ต้องแสดงข้อมูลที่อยู่ในช่องที่ 0 ของอาเรย์ทุกตัวออกมา รูปแบบของการเก็บข้อมูลในลักษณะนี้เป็นดังตาราง

	มานะ	ปิติ	มานี	ชูใจ
Name	มานะ	ปิติ	มานี	ชูใจ
ID	0001	0002	0003	0004
Age	32	33	27	27
Salary	8900	8950	8500	8600

จะเห็นว่าอาเรย์ตัวที่ 1 เก็บชื่อก็จะเก็บชื่อทุกๆ ช่อง ตัวที่เก็บรหัสพนักงานก็จะเก็บรหัสพนักงานทั้งหมด วิธีการลักษณะนี้เสี่ยงต่อการผิดพลาดอย่างมากในเรื่องของลำดับ เพราะตัวแปรแต่ละตัวมีความเป็นอิสระต่อกัน การอ้างลำดับผิดในกรณีที่มีปริมาณข้อมูลมากๆ เช่นเก็บข้อมูลเป็นหลักร้อยขึ้นไป การไล่ลำดับจะต้องไล่ให้ตรงกัน ซึ่งจากปัญหาที่ได้กล่าวไปนี้ เราจะต้องใช้ตัวแปรแบบโครงสร้างมาช่วยแก้ปัญหา การใช้งานตัวแปรโครงสร้างนี้จะคล้ายๆ กับการที่เราสร้างตัวแปรชนิดใหม่ขึ้นมาเป็นของเราเอง โดยพิจารณาจากข้อมูลที่เราต้องการจัดเก็บ รูปแบบการเก็บข้อมูลแสดงได้ดังรูป



รูปที่ 2.5 ตัวอย่างของออบเจกต์คน

การประกาศตัวแปรโครงสร้าง เราจะใช้คำสั่ง struct ดังนี้

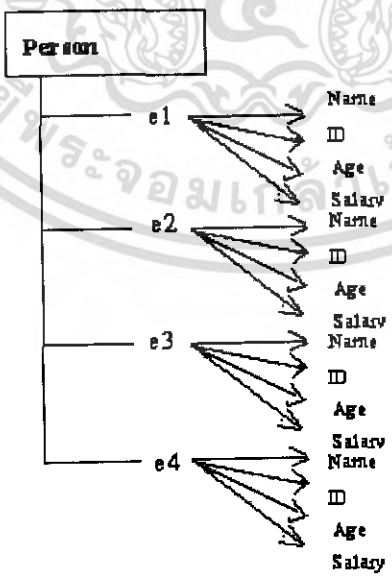
```
public struct  employee
{
        string  name;
        string  id;
        int     age;
        int     salary;
 }
```

จากข้างต้นเป็นการสร้างตัวแปรแบบโครงสร้างชนิดใหม่ขึ้นมา ชื่อว่า employee เราจะถือว่าเป็นการสร้างตัวแปรแบบใหม่ และตัวแปรนี้สามารถเก็บค่าได้ 4 ค่า คือ ชื่อ (name), รหัสพนักงาน (id), อายุ (age) และเงินเดือน (salary) เมื่อเราประกาศตัวแปรแบบนี้เอาไว้ในโปรแกรมของเราแล้ว เราก็สามารถสร้างตัวแปรแบบ employee นี้ได้ทันทีโดยเขียนโปรแกรมได้ดังนี้

employee e1, e2, e3, e4;

ตอนนี้เราได้ตัวแปร e1, e2, e3 และ e4 เป็นตัวแปรแบบ employee แล้ว

สำหรับตัวอย่างนี้ก็เช่นกัน คือ เมื่อเราได้สร้างตัวแปรโครงสร้างชนิดใหม่ขึ้นมาแล้วโดยชื่อว่า employee เราก็จะสามารถสร้างตัวแปรแบบ employee ขึ้นมาได้ และตัวแปรที่ถูกสร้างขึ้นมาได้ จะมีคุณลักษณะการเป็นตัวแปรแบบ employee ทุกอย่าง คือ ภายในตัวมันเองสามารถเก็บค่าย่อยๆ ได้อีก 4 ค่าดังรูป



รูปที่ 2.6 การเก็บข้อมูลแบบสดรัด

ลักษณะของตัวแปรแบบโครงสร้างนี้ จะจัดเก็บข้อมูลเป็นหมวดหมู่ และแบ่งแยกชัดเจน อย่างเห็นได้ชัดว่า name, id, age และ salary ก็เป็นของคนๆ นั้นเลยเพราะฉะนั้นมันจะไม่ปะปนกัน คือ เช่น salary ของ e1จะเป็นของ e1 โดยเฉพาะอาจจะไม่เท่ากับ salary ของ e2 ก็ได้ ดังนั้นด้วย หลักการของข้อมูลแบบโครงสร้างนี้ เมื่อเรานำมาใช้ในการเก็บข้อมูล
ชื่อ, รหัสพนักงาน, อายุ และ เงินเดือน ของพนักงาน 4 คนดังที่ได้ยกตัวอย่างไว้ ถ้าเราเขียนออกมาเป็นโปรแกรมภาษา C จะได้ดังนี้

```
public  struct employee
{
        string  name;
        string  id;
        int     age;
        int     salary;
}

employee e[4];

e[0].name  =  "มานะ";
e[0].id      =  "0001";
 e[0].age     =  "32";
 e[0].salary =  "8900";

e[1].name  =  "ปิติ";
e[1].id      =  "0002";
e[1].age     =  "33";
 e[1].salary =  "8950";

 e[2].name  =  "มานี";
 e[2].id      =  "0003";
 e[2].age     =  "27";
 e[2].salary =  "8500";
```

```
e[3].name = "ชูใจ";
e[3].id   = "0004";
e[3].age  = "27";
e[3].salary = "8600";
```

จากตัวอย่าง จะเหมือนกับว่าเรามีตัวแปรทั้งหมด 16 ตัว และแต่ละตัวก็ถูกจัดหมวดหมู่แบ่งออกเป็นคนๆ เป็นข้อมูลเฉพาะส่วนๆ ไป เราสามารถเข้าถึงข้อมูลที่เป็นเงินเดือนของพนักงานที่ต้องการได้โดยอ้างอิงลำดับของพนักงานนั้นๆ และตามด้วยชื่อตัวแปรที่เราต้องการ เช่นต้องการรู้เงินเดือนของนางสาวมานี ซึ่งอยู่ลำดับที่ 3 ก็คือ e[2].salary เป็นต้น จะเห็นได้ว่าวิธีเป็นการเปลี่ยนแนวคิดและมุมมองจากอาเรย์ให้มาเป็นแนวคิดที่คล้ายกับเชิงวัตถุ คือเรามองเห็นพนักงานแต่ละคนเป็นวัตถุมากขึ้น 1 คนก็จะมี ชื่อ, รหัส, อายุ และ เงินเดือน ทำให้การจัดการข้อมูลทำได้ง่ายกว่าการใช้อาเรย์แบบในวิธีแรกที่ได้นำเสนอไป

2.2.4 การสร้างคลาส

คลาส คือที่รวมของเมธอด ฟิลด์ และคลาสอีกที่หนึ่ง โดยการนิยามคลาสให้ใช้คีย์เวอร์ดว่า class แล้วตามด้วยชื่อคลาส หลักการตั้งชื่อก็เหมือนกับชื่อตัวอ้างอิงทั่วไป แต่สำหรับคลาสนิยมขึ้นต้นตัวแรกเป็นตัวอักษรใหญ่ แต่สำหรับชื่อเมธอดนิยมเขียนขึ้นต้นด้วยด้วยตัวอักษรเล็ก

ตัวอย่างการสร้างคลาสในภาษา C#

โปรแกรม Start.cs

```
class Employee
{
        private string name;
        private string id;
        private int age;
        private int salary;

        public Employee()
        {
            Console.WriteLine("Constructor Employee() is called");
```

```
}

public Employee(string name)
{
        Console.WriteLine("Constructor Employee(string name) is called");
        this.name = name;
}

public void ShowName()
{
        Console.WriteLine(name);
}

public void setName(string  name)
{
        this.name  =  name;
}

public void setID(string  id)
{
        this.id  =  id;
}


public void setAge(int  age)
{
        this.age  =  age;
}

public void setSalary(int  salary)
{
```

```
            this.salary = salary;
        }

        public string getName()
        {
            return name;
        }

        public string getID()
        {
            return id;
        }

        public int getAge()
        {
            return age;
        }

        public int getSalary()
        {
            return salary;
        }

}
```

```
class Start
{
        public static void Main()
        {
                Employee x = new Employee ("Samran");
                x.ShowName();
        }
}
```

โปรแกรม Start.cs ในข้างต้นมีการประกาศคลาส Employee เอาไว้ภายในคลาสมีสมาชิกดังนี้

- แอทริบิวต์ name
- แอทริบิวต์ id
- แอทริบิวต์ age
- แอทริบิวต์ salary
- เมธอด Employee()
- เมธอด Employee(string name)
- เมธอด ShowName()
- เมธอด setName(string name)
- เมธอด setID(string id)
- เมธอด setAge(int age)
- เมธอด setSalary(int salary)
- เมธอด getName()
- เมธอด getID()
- เมธอด getAge()
- เมธอด getSalary()

คลาส Employee ที่ยกตัวอย่างมานี้ แสดงให้เห็นถึงการจัดเก็บข้อมูล ก็คือ ชื่อ, รหัสพนักงาน, อายุ และ เงินเดือน โดยทั้ง 4 ถูกกำหนดให้มีลักษณะการเข้าถึง(visibility) เป็น private มีคอนสตรัคเตอร์ 2 ตัว คือ Employee() และEmployee(string name) โดย คอนสตรัคเตอร์จะมีชื่อเหมือนคลาสจะเรียกใช้ทุกครั้งที่คลาสถูกประกาศ มักใช้เป็นการกำหนดค่าเริ่มต้นให้กับคลาสที่คอนสตรัคเตอร์สร้างขึ้นนั่นเอง ส่วนเมธอด ShowName() ซึ่งจะทำการแสดงชื่อออกมาทางจอภาพ ส่วนเมธอดที่ขึ้นต้นด้วย set นั้นใช้ในการ

กำหนดค่าของแอทริบิวต์ ส่วนเมธอดที่ขึ้นต้นด้วย get นั้นใช้ในการอ่านค่าของแอทริบิวต์ โดยทุกเมธอดยกเว้นคอนสตรัคเตอร์ถูกกำหนดให้มีลักษณะการเข้าถึงเป็น public ซึ่งมีความหมายดังนี้

- private แอทริบิวต์ และ/หรือ เมธอด ที่ถูกกำหนดให้เป็น private จะไม่ถูกเปิดเผยแก่ภายนอก และไม่สามารถเข้าถึงได้โดยตรงจากภายนอก แต่สามารถเข้าถึงได้จากภายในตัวคลาสเอง
- pretected แอทริบิวต์ และ/หรือ เมธอดที่ถูกกำหนดอย่างนี้ จะไม่ถูกเปิดเผยแก่ภายนอกและไม่สามารถเข้าถึงได้โดยตรงจากภายนอก แต่สามารถเข้าถึงได้จากภายในตัวคลาสเอง ซึ่งทั้งแอทริบิวต์และเมธอดที่ถูกกำหนดอย่างนี้จะถูกถ่ายทอดไปให้กับคลาสลูก (subclass) และสามารถเข้าถึงได้จากภายในคลาสลูก
- public แอทริบิวต์ และ/หรือ เมธอดที่ถูกกำหนดอย่างนี้จะถูกเปิดเผยและถูกเข้าถึงได้โดยตรงจากภายนอก รวมทั้งยังสามารถถ่ายทอดไปยังคลาสลูกได้ด้วย นอกจากนี้มันยังทำหน้าที่เป็น public Interface ของ Class

เมื่อเราต้องการใช้งานคลาสเราต้องทำการสร้างออบเจกต์ของคลาสขึ้นมาก่อน เช่นจากโปรแกรมในคลาส Start มีการสร้างออบเจกต์ของคลาส Employee

```
Employee x = new Employee ("Samran");
```

ทำให้เราได้ออบเจกต์ x เป็นของคลาส Employee จากนั้นเราก็สามารถนำ x ไปใช้ดำเนินกิจกรรมแทนคลาส Employee ได้เช่นจากเมธอด Main ในคลาส Start มีคำสั่ง

```
x.ShowName();
```

เป็นการเรียกเมธอดชื่อ ShowName() มันจะทำการแสดงชื่อ Samran ออกทางหน้าจอ ถ้าเราต้องการกำหนดเงินเดือนให้กับออบเจกต์ x เราสามารถใช้คำสั่ง

```
x.setSalary(15000);
```

มันจะทำการกำหนดอาร์กิวเมนต์ 15000 ไปให้กับแอทริบิวต์ salary ของออบเจกต์ x เป็นต้น

2.3 DirectX

2.3.1 DirectX คืออะไร

ก่อนที่จะมี DirectX เกิดขึ้นนั้น นักพัฒนาเกมคอมพิวเตอร์สำหรับ DOS หรือ วินโดวส์ จะต้องเขียนเกม คอมพิวเตอร์ให้รู้จักกับฮาร์ดแวร์ ซึ่งมีอยู่มากมายในท้องตลาด ซึ่งในกรณีที่มีฮาร์ดแวร์ตัวใหม่เกิดขึ้น อาจเกิดปัญหาความไม่สนับสนุนกันระหว่างเกมคอมพิวเตอร์ กับฮาร์ดแวร์ตัวใหม่นั้น นักเล่นเกมจะต้องรอจนกว่านักพัฒนาเกมจะทำการอัพเดตเกมนั้นๆ ให้ใช้ความสามารถของฮาร์ดแวร์ตัวใหม่ได้ เมื่อเป็นเช่นนี้ไมโครซอฟต์จึงได้ทำการพัฒนาเทคโนโลยีที่มีความสามารถในการเป็นสื่อกลางติดต่อระหว่างเกมคอมพิวเตอร์ หรือโปรแกรมมัลติมีเดียต่างๆ กับ ฮาร์ดแวร์ขึ้นมาโดยใช้ชื่อว่า DirectX

DirectX ตามความหมายจะหมายถึง ไลบรารี่คำสั่ง (Run Time Library) ที่ช่วยทำงานด้านมัลติมีเดีย Graphic โดยตัว DirectX Foundation จะมีส่วนประกอบที่เรียกว่า HAL (Hardware Abstraction Layer) จะใช้ซอฟต์แวร์ในการรวจสอบความสามารถของฮาร์ดแวร์ที่อยู่ในเครื่องคอมพิวเตอร์นั้นอย่างอัตโนมัติ แล้วนำมากำหนดพารามิเตอร์ของแอพลิเคชั่นให้ตรงตามความเหมาะสมระหว่างเกมคอมพิวเตอร์ หรือโปรแกรมมัลติมีเดียกับไดร์เวอร์ของฮาร์ดแวร์ที่มีส่วนเกี่ยวข้องกับเกมหรือโปรแกรมนั้น ทำให้การประมวลผลโปรแกรมทำได้เร็วขึ้น เนื่องจากขั้นตอนต่างๆ จะถูกนำไปประมวลผลโดยตรง ไม่ต้องอาศัยตัวกลางอย่างเช่น GDI (Graphic Device Interface) ก่อน ทำให้นักพัฒนาเกมคอมพิวเตอร์สามารถเขียนโปรแกรมให้สื่อสารกับ DirectX เท่านั้นก็เพียงพอ นอกจากนี้ DirectX Foundation ยังมีส่วนประกอบที่เรียกว่า HEL (Hardware emulation Layer) ทำให้สามารถใช้โปรแกรมมัลติมีเดีย หรือโปรแกรมที่เกี่ยวข้องกับ 3D บน Hardware ที่ไม่สนับสนุนการใช้งานทางด้าน 3 มิติ โดยจะทำการจำลองความสามารถบางอย่างที่ฮาร์ดแวร์ตัวนั้นไม่มี ให้สามารถใช้งานได้กับโปรแกรมที่ต้องการ แม้จะมีข้อเสียอยู่บ้างตรงที่ อาจทำให้ช้าลงบางก็ตาม แต่ก็คุ้มค่ากับความสามารถของ DirectX ที่มีอยู่ในปัจจุบัน

DirectX ถูกใส่ไว้ให้เป็นส่วนหนึ่งตั้งแต่ Windows98 และ Windows2000 และส่วนประกอบอื่น ๆ ของ DirectX จะสามารถติดตั้งเพิ่มเติมได้ในภายหลัง ด้วยจุดประสงค์ที่จะให้ผู้พัฒนาโปรแกรม (developers) มีชุดคำสั่ง และส่วนประกอบต่าง ๆ ร่วมกัน เพื่อที่จะ

- ทำให้โปรแกรม multimedia ที่ผู้พัฒนาผลิตขึ้นมา สามารถทำงานได้กับ PC ที่ใช้ windows โดยไม่คำนึงถึงระบบ hardware ที่ใช้ และยืนยันว่า โปรแกรมจะสามารถใช้ประสิทธิภาพสูงสุดที่ hardware มี เพื่อให้ได้ผลงานที่มีคุณภาพสูงสุด
- ทำให้ผู้พัฒนา สามารถใช้อุปกรณ์การพัฒนาช่วยเหลือการเขียนโปรแกรม ทำให้การเขียนโปรแกรม multimedia สามารถทำได้โดยง่าย และยังสามารถผสมสื่อหลากชนิดเข้าไว้ด้วยกันได้

2.3.2 ประวัติความเป็นมาของ DirectX

ในช่วงที่ระบบปฏิบัติการไมโครซอฟต์วินโดวส์เริ่มต้นขึ้นมานั้น สิ่งที่เป็นเอกลักษณ์ของวินโดวส์ ก็คือเรื่องของ GUI(Graphic User Interface) โดยมี GDI (Graphic Device Interface) เป็นเครื่องมือสำหรับจัดการทางด้านภาพ และเกมทั่วไปก็ยังคงพัฒนาภายใต้ระบบปฏิบัติการเดิมอยู่ ซึ่งก็คือระบบปฏิบัติการ "DOS" นั่นเอง

การเริ่มต้นครั้งแรกสำหรับไมโครซอฟต์คือ "เครื่องมือ Win-G" สำหรับ Windows95 และดูเหมือนว่า เครื่องมือตัวนี้จะไม่ได้รับการตอบรับที่ดีจากผู้พัฒนาทั่วโลก ยังผลให้ไมโครซอฟต์เองต้องปรับปรุงรูปแบบใหม่ให้กับเครื่องมือตัวนี้อีกครั้ง

ไมโครซอฟต์ได้ทำการจัดตั้งทีมงานขึ้นมาใหม่เพื่องานกราฟิก งานมัลติมีเดีย งานเน็ตเวิร์ก งานด้านการรับข้อมูล Input และ งานกราฟิก 3 มิติ (ชื่อเทคโนโลยี "RenderGraphic") โดยใช้ชื่อว่า "DirectX"

DirectX 2.0 สามารถเข้ามาจัดการงานด้านเกมได้เป็นอย่างดีการทำงานเร็วขึ้นกว่า ระบบปฏิบัติการ DOS ก็เลยทำให้โปรแกรมเมอร์ทั่วโลกได้เริ่มหันมามองเทคโนโลยีตัวนี้ใหม่อีกครั้ง และจากการพัฒนาอย่างต่อเนื่องของไมโครซอฟต์เพื่อเพิ่มประสิทธิภาพของ DirectX ให้มีความสามารถมากขึ้นทำให้เกิด DirectX 3.0, DirectX 4.0, DirectX5.0, DirectX 6.0, DirectX 7.0, DirectX 8.0 และ DirectX 9.0 ซึ่งปัจจุบันเป็น เวอร์ชัน 9.0C และยังคงมีการพัฒนาเวอร์ชันใหม่ๆ ต่อไป จากเทคโนโลยีนี้เองเป็นผลให้เกมบนระบบปฏิบัติการ DOS ค่อยๆ เลือนหายไปในที่สุด

2.3.3 ความรู้เบื้องตนเกี่ยวกับ DirectX

DirectX เป็น API ของไมโครซอฟท์พัฒนาเพื่อใช้จดเตรียมอินเตอร์เฟส สำหรับควบคุมฮาร์ดแวร์มัลติมีเดียบนระบบ Microsoft Windows ได้อย่างมีประสิทธิภาพ และเป็นเครื่องมือให้โปรแกรมเมอร์ทำงานกับคำสั่ง และโครงสร้างข้อมูลในระดับใกล้ชิดฮาร์ดแวร์ โดยไม่ต้องสร้างโค้ดติดต่อในระดับล่างซึ่งวิธี ติดต่อจะแตกต่างกันไปตามประเภทของอุปกรณ์ การเขียนโค้ดที่เป็นอิสระจากอุปกรณ์ในลักษณะนี้ ช่วยให้โปรแกรมเมอร์สามารสร้างซอฟต์แวร์เพื่อทำงานดังกล่าวได้อย่างดี แม้ผู้ใช้จะปรับเปลี่ยนอุปกรณ์ตัวใหม่ และเพิ่มการ์ดเร่งความเร็วแบบสามมิติ เสียง อุปกรณ์อินพุต และ อื่นๆก็ตาม

DirectX ได้รับการออกแบบให้นักพัฒนามีสภาพแวดล้อมคล้ายคลึงกับสภาพแวดล้อมที่มีประสิทธภาพของ MS-DOS ซึ่งทำงานได้เร็วกว่าโค้ดที่ทำงานบนวินโดวส์ เนื่องจากไม่ต้องสูญเสียประสิทธิภาพจาก API สำหรับจัดการงานมัลติมีเดียของ วินโดวส์รุ่นก่อน แต่อย่างไรก็ตามการ

สนับสนุนความสามารถในการทำงานของฮาร์ดแวร์ที่มีอยู่บนระบบ โค้ดที่เขียนขึ้นด้วย DirectX สามารถรันได้เร็วกว่าแอพพลิเคชั่นของ MS-DOS

เมื่อใดก็ตามที่สร้างอ๊อบเจ็กต์ DirectX ให้กบดีไวซ์นั้น DirectX จะเข้าไปซักถามฮาร์ดแวร์ผ่าน HAI เพื่อดึงเอาข้อมูลเกี่ยวกับดีไวซ์ออกมานี้มีอยู่ในตาราง Cap Bits (Capability Bits) ข้อมูลที่มีอยู่ใน Cap Bits เป็นข้อมูลที่ใช้บอกความสามารถที่ฮาร์ดแวร์สมารถทำได้หรือความสามารถใดที่ HEL ต้องจำลอง

ไมโครซอฟท์จัดเตรียม Cap Bits ให้รับทราบฟีเจอร์ของฮาร์ดแวร์ที่มีอยู่ใน HAL และฟีเจอร์ที่ต้องจำลองขึ้นโดยซอฟต์แวร์ HEL ดังนั้นวิธีที่ดีที่สุดก็คือเขียนแอพพลิเคชั่นโดยใช้ค่าระบบ หรือฟีเจอร์ต่ำสุดที่ยอมรับได้ และ Optimize โค้ดที่เขียนให้รันได้เร็วและมีประสิทธิภาพมากที่สุด อีกทางหนึ่งควรเขียนโค้ดเผื่อการทำงานของ HEL ในอนาคตไว้ด้วย การสนับสนุนการใช้ฟีเจอร์เหล่านี้เป็นฟีเจอร์พิเศษที่จะมีให้เลือกบนเกมได้ เช่น การทำงาน Texture ขั้นสูง การทำโพลีกอนที่มีความซับซ้อนสูง หรือแม้แต่การสร้างแสงแบบไดนามิก เผื่อว่าใครก็ตามที่มีฮาร์ดแวร์ที่มีความสามารถสูงจะสามารถใช้งานฟีเจอร์ดังกล่าวได้ ดังนั้นควรออกแบบให้รองรับฟีเจอร์ต่างๆที่มีได้ทั้งหมด

DirectX จะใช้หลักการของ COM (Component Object Model) ซึ่งเป็นวิธีในการนำเอา component ที่มีประโยชน์ มาใช้ใหม่ใช้ในโปรแกรมต่างๆ โดยส่วนใหญ่แล้ว COM จะถูกสร้างเอาไว้ในไฟล์ .dll ซึ่งการที่เราจะสามารถใช้เมทธอดต่างๆ ที่มีใน COM ตัวนั้นได้ เราจะต้องทำการสร้าง COM object ขึ้นมา และเนื่องจาก COM จะใช้คุณสมบัติ Encapsulation อย่างเข้มงวด ทำให้เราไม่สามารถเรียกใช้เมทธอดต่างๆ จาก COM object ที่สร้างขึ้นมาได้ เราต้องสร้าง Interface จาก COM object อีกทีหนึ่ง เราถึงจะเรียกใช้เมทธอดที่ต้องการจาก Interface ได้ โดยใน Interface หนึ่งๆ ก็จะมีเมทธอดตามประเภทของ Interface ที่สร้างขึ้นเท่านั้น

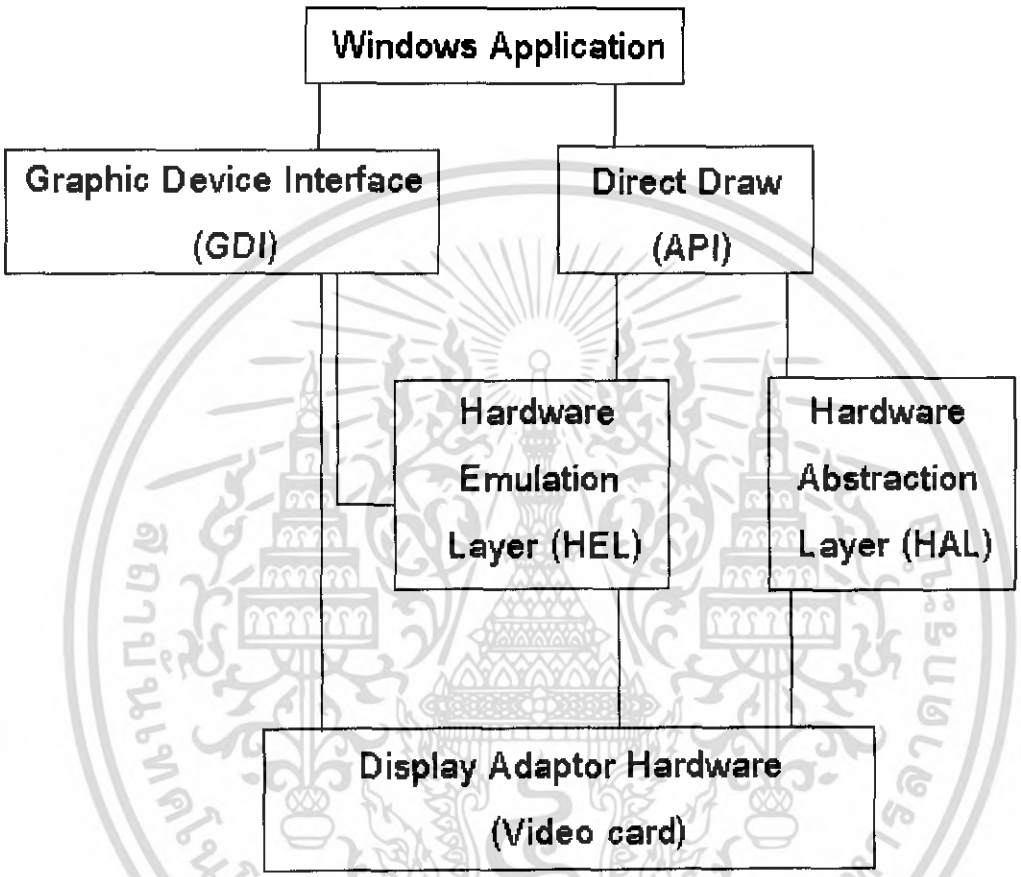
DirectX จะประกอบขึ้นมาจากส่วนหลักๆ 7 ส่วน ตามนี้

- DirectDraw จัดการกับกราฟิก 2 มิติ และการติดต่อกับการ์ดแสดงผล
- Direct3D จัดการกับกราฟิก 3 มิติ และการติดต่อกับฮาร์ดแวร์ 3 มิติ
- DirectSound จัดการกับเสียง และการติดต่อกับการ์ดเสียง
- DirectPlay ควบคุมการติดต่อกับเครือข่ายสำหรับการเล่นเกมแบบ Multi-Player
- DirectInput ควบคุมการรับข้อมูลจากอุปกรณ์นำข้อมูลเข้า (Input Device)
- DirectMusic จัดการกับการเล่นเพลง
- DirectSetup สำหรับการติดตั้งส่วนประกอบต่างๆ ของ DirectX

2.3.4 API ของ DirectX

2.3.4.1 DirectDraw

เป็นชุด API สำหรับใช้จัดการอุปกรณ์แสดงผล ควบคุมข้อมูลบิตแมปหน่วยความจำ ออกนอกพื้นที่สกรีน และสร้างการติดต่อที่รวดเร็วให้กับฟีเจอร์ของฮาร์ดแวร์ เช่น Blitting และ Page Flipping ซึ่งเป็นฟีเจอร์พื้นฐานที่ Direct3D สามารถทำได้



รูปที่ 2.7 แสดงแผนภาพการทำงานของ Direct X

- **Cooperative level และ Display mode**

ก่อนที่จะคำสั่งต่างๆใน DirectX ให้แสดงผลภาพได้นั้น เราจะต้องเซต Cooperative level และ Display ode ก่อน โดยการเซต Cooperative level ก็คือการบอก DirectDraw ว่าเราจะทำงานกับ DirectDraw ในลักษณะใด เช่น window ode หรือ fullscreen mode สามารถ interrupt ด้วยการกดปุ่ม Ctrl-Alt-Del ได้หรือไม่ และสามารถที่จะเปลี่ยนขนาดของ window ได้หรือไม่ (ใน window mode) เป็นต้น หลังจากเซต Cooperative level แล้วจะต้องเซต Display mode เป็นลำดับถัดมา เพื่อระบุ screen Resolution, color depths และ refresh reate ที่ต้องการ โดยจะต้องระบุความกว้างความยาว

ของ resolution เป็นหน่วย pixel และจะใช้สีแบบใด 6 ,8 ,24 หรือ 32 บิต หรือจะใช้ตารางสี (palette) ทั้งนี้ video hardware ที่ใช้จะต้องสนับสนุนกับ mode ที่เลือกด้วย มิฉะนั้นจะเกิด error Display mode

- **Surface**

Surface เป็นพื้นที่บน memory ใช้สำหรับเก็บข้อมูลภาพ bitmap เพื่อเตรียมสำหรับการนำมาแสดงผลบนจอภาพต่อไป โดยพื้นที่นี้อาจอยู่บน video memory หรือ system memory ก็ได้ขึ้นอยู่กับค่า flags ที่ส่งให้ตอนสร้าง โดย surface แบ่งออกเป็น 2 ชนิด คือ Primary Surface และ Off-screen surface

Primary surface นั้นเป็น surface หลักที่จะแสดงผลออกสู่จอภาพ โดยจะต้องมีขนาดเท่ากับ Display mode ที่เลือกไว้ ภาพใดๆ ก็ตามที่ต้องการแสดงออกทางจอภาพต้องนำมาวาดไว้บน Primary surface เสมอ แต่การนำภาพมาวาดลงบน Primary surface โดยตรงจะทำให้ภาพที่ได้กระพริบไม่ราบรื่น จึงต้องใช้การทำ flipping มาช่วย (จะกล่าวรายละเอียดในหัวข้อถัดไป)

Off-screen surface คือ surface ใช้สำหรับเก็บข้อมูลภาพต่างๆ เพื่อ Blitting ลงใน buffer หรือ surface อื่นเพื่อเตรียมแสดงผลต่อไป

- **Blitting และ Flipping**

Blitting คือ การนำภาพต่างๆ จาก source ไปวาดบนบริเวณ destination ที่กำหนด โดยอาจมีการย่อ – ขยายภาพให้พอดีกับขนาด destination ที่กำหนด และสามารถทำการ transparent สีบางสีได้ด้วย การทำ Blitting สามารถทำได้หลายครั้ง โดยภาพที่ทำ Blitting จะเรียงซ้อนทับกันเป็นชั้นๆ

Flipping เป็นการสลับที่กันของ buffer โดยก่อนการทำ Flipping จะต้องมีการสร้าง bufer ขึ้นมาอย่างน้อย 2 buffer คือ Front buffer กับ Back buffer เพิ่มขึ้นมาก็ได้ โดย Front buffer คือ buffer ที่แสดงภาพออกที่หน้าจอ หรือ Primary surface นั่นเอง ส่วน Back buffer กับ Third buffer เป็นที่พักของภาพที่จะรอแสดงผลบนหน้าจอ buffer ทั้ง 3 ชนิดนี้จะมีขนาดเท่ากับ Display mode ที่เลือกไว้

การวาดหรือ Blitting ลงบน Front buffer โดยตรงจะให้ภาพกระพริบ ควรเตรียมภาพที่จะแสดงผลไว้ให้พร้อมบน Back buffer แล้วทำการ Flipping โดย DirectDraw จะสลับภาพระหว่าง Front buffer กับ Back buffer เพื่อนำภาพบน Back buffer มาแสดงบนจอภาพ โดยการสลับในที่นี้ไม่ได้ หมายถึง การสลับที่ข้อมูลภาพแต่เป็นการสลับเฉพาะ

pointer ที่ชี้พื้นที่ที่เก็บข้อมูลอยู่เท่านั้น ซึ่งทำได้อย่างรวดเร็ว และไม่ทำให้เกิดการกระพริบหรือการกระตุกของภาพ

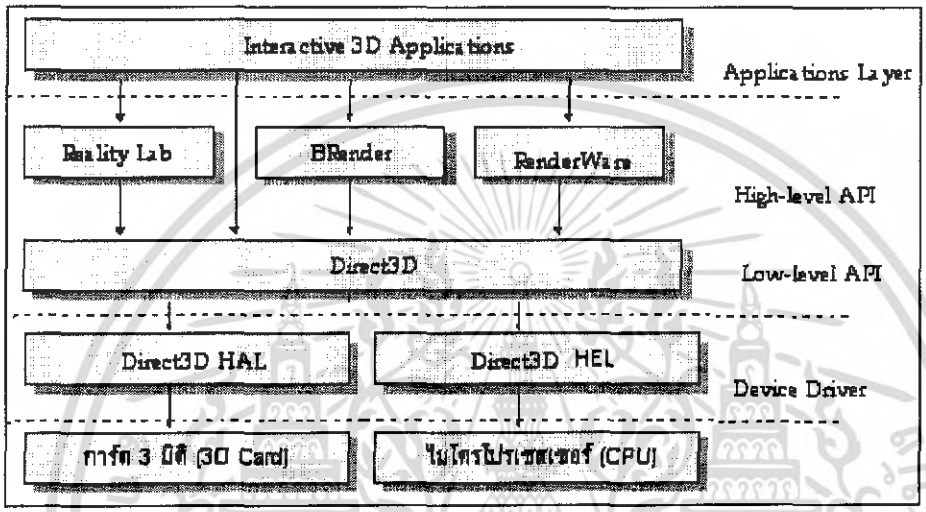
2.3.4.2 Direct3D

Direct3D คือ API ที่สามารถใช้เขียนโปรแกรมการฟิก 3 มิติ และติดต่องานฮาร์ดแวร์แต่งความเร็วสามมิติ การ์ดแสดงผลส่วนใหญ่ที่มีขายในท้องตลาดสนับสนุนความสามารถในการเร่งการแสดงผลสามมิติ และเกมสามมิติส่วนใหญ่ที่มีอยู่ในปัจจุบันก็มักจะรันบนวินโดว์ซึ่งใช้ Direct3D

Direct3D ในยุคแตกมี API อยู่สองโหมดคือ โหมด Immediate (IM) และ โหมด Relation (RM) ในโหมด IM (การเรนเดอร์อ็อบเจ็กต์ทำได้โดยฉับพลันได้ตามความต้องการของโปรแกรมเมอร์) เป็นโหมดที่ใช้งานยากแต่มีความยืดหยุ่นสูงเป็น API ในระดับล่างสำหรับใช้เขียนเกมที่ทำงานได้เร็ว และมีประสิทธิภาพเท่าที่จะเป็นไปได้บนระบบ ในโหมด RM (API จะเก็บซีนลงในฐานข้อมูล แล้วนำมาเรนเดอร์ทั้งหมดในคราวเดียวกัน) เป็นโหมดที่สร้างขึ้นมาเป็นเลเยอร์ที่อยู่บนสุดของโหมด IM โดยโหมดนี้จะจัดเตรียมบริการต่างๆ เช่น การจัดการ Texture, การโหลด Object File, การจัดลำดับเฟรม และการทำ อ็อบเจ็กต์เคลื่อนไหว การศึกษาและใช้งานโหมด RM นั้นง่ายกว่าเมื่อเทียบกับโหมด IM แต่ในโหมด IM เป็นโหมดที่มีประสิทธิภาพและความยืดหยุ่นสูงกว่า ดังนั้นการพัฒนาการทำงานในโหมด RM จึงได้หยุดลงใน DirectX 6.0 และมุ่งพัฒนาการทำงานในโหมด IM ให้มีความสามารถและใช้งานง่ายในเวอร์ชันต่อมา ด้วยเหตุนี้เองการทำงานในโหมด RM จึงไม่สนับสนุนเทคโนโลยีใหม่เช่น Multitexturing ,Bump Mapping, Hardware Transformation และ Linghting ดังนั้นความสามารถทั้งหมดของโปรแกรม 3 มิติควรเขียนขึ้นมาด้วยการใช้โหมด *IM*

หลักการทำงานของ Driect3Dคือ คอยเป็นตัวกลางประสานงานระหว่างซอฟต์แวร์ 3 มิติ จำพวกเกมส์และ Application (ที่ออกแบบให้ใช้งานบน *Windows 95*) และการ์ดแสดงผลแบบ 3 มิติ ซึ่งเป็นฮาร์ดแวร์ โดยการที่โปรแกรมเมอร์ที่เขียนโปรแกรมนั้นสามารถเขียน Code เพื่อเรียกใช้งานการแสตงผล 3 มิติ ที่ระดับใดๆ ก็ได้ และเนื่องจาก Direct3D นั้นเป็น API ที่ทำงานในระดับของ OS โดยไม่ได้ยุ่งเกี่ยวกับฮาร์ดแวร์ ดังนั้นการเขียนโปรแกรมจึงยืดหยุ่น สามารถกำหนดทางเลือกให้กับโปรแกรมเมื่อให้การประมวลผลภาพ 3 มิติบนคอมพิวเตอร์ได้อย่างมีประสิทธิภาพมากที่สุด

สำหรับการ์ด 3 มิตินั้น จะเห็นได้ว่าเทคนิคและกลไกในการแสดงภาพ 3 มิติ ก็อย่าง ที่กล่าวไปแล้วว่ามีหลากหลายมาก (Texturing, Fog, Alpha Bending, Z Buffering ฯลฯ) และผู้ผลิตการ์ดแต่ละราย ต่างก็มีมาตรฐานในการผลิตและการออกแบบชิปสำหรับการ์ด 3 มิติแตกต่างกันออกไป เพราะการ์ด 3 มิติเป็นเทคโนโลยีใหม่ในคอมพิวเตอร์ ในมุมของผู้ผลิตแล้วการออกแบบการ์ด 3 มิติให้ทำงานกับ Direct3D ไม่ใช่เรื่องง่ายนัก เป็นที่คาดการณ์ว่าประสิทธิภาพการทำงานของการ์ด 3 มิติ แต่ละยี่ห้อนั้นจะแตกต่างกันออกไป



รูปที่ 2.8 ภาพแสดงสรุปการทำงานของ Direct3D

เมื่อ Direct3D นั้นมีอัลกอริทึมในการควบคุมฟังก์ชั่น 3 มิติ ออกเป็น 2 ระดับใหญ่ คือ HEL (Hardware Emulation Layer) ซึ่งจะถูกเรียกใช้ในกรณีที่ตรวจพบว่าไม่มีการ์ดแสดงผล 3 มิติในคอมพิวเตอร์ เพื่อให้คอมพิวเตอร์สามารถทำงานกับกราฟิก 3 มิติได้ก็จะต้องจำลองฮาร์ดแวร์ 3 มิติ ซึ่งปกติตัวที่ถูกนำมาจำลองเป็นผู้ประมวลผลกราฟิกก็คือ ซีพียู ภาระหนักจะตกอยู่ที่ซีพียูซึ่งผลที่ได้จากการประมวลผลจะถูกส่งไปที่ Frame Buffer ของ DirectDraw แทน และระดับที่สองคือ HAL (Hardware Abstraction Layer) Direct3D จะควบคุมการแสดงภาพแบบ 3 มิติโดยผ่านฮาร์ดแวร์ที่มีความสามารถในระบบ 3 มิติ ซึ่งในที่นี้คือ การ์ด 3 มิติ Direct3D โดยหลักการแล้ว เมื่อมีการรันซอฟต์แวร์ที่ต้องการใช้งานฟังก์ชั่น 3 มิติ Direct3D ก็จะมีอัลกอริทึมในการตรวจสอบหาช่องทางหรือเครื่องมือที่จะให้การแสดงผล 3 มิติ ในคอมพิวเตอร์นั้นๆ ดีที่สุด ซึ่งในที่สุดก็จะเป็นตัวบ่งว่า Application นั้นต้องใช้ HEL หรือ HAL ในการควบคุมการแสดงผล 3 มิติ นั่นเอง

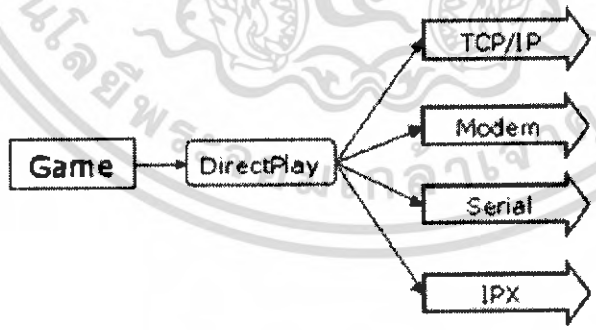
Direct3D ไม่ได้ทำงานเป็นตัวเชื่อมประสานระหว่าง 3D Application อย่างเดียวแต่ยังเปิดช่องให้ผู้ผลิต 3D Application API รายอื่นๆ ได้พัฒนา Application หรือฟังก์ชั่นใน

การเรียกใช้ Direct3D ด้วย ซึ่งการเปิดกว้างนี้เป็นช่องทางหนึ่งที่ทำให้ Direct3D กลายเป็นเครื่องมือสอนคอมพิวเตอร์ให้รู้จักกราฟิก 3 มิติ โดยแท้จริง

ถึงแม้ว่าประสิทธิภาพของ Direct3D ในส่วนของซอฟต์แวร์เพียงอย่างเดียวจะไม่เลวร้ายมากนัก แต่ DirectX นั้นถูกออกแบบมาให้มีความสามารถของฮาร์ดแวร์เร่งความเร็ว ซึ่งหมายถึง ผู้ผลิตการ์ดแสดงผลสามารถที่จะสร้างการ์ดที่สนับสนุนฟังก์ชันของ DirectDraw และ Direct3D ได้จากตัวฮาร์ดแวร์โดยตรง พร้อมกับให้ไดร์เวอร์สำหรับการ์ดของตนที่สนับสนุน DirectX มาด้วยกัน เมื่อมีการเรียกใช้ฟังก์ชันที่การ์ดสนับสนุน การทำงานดังกล่าวก็จะถูกส่งไปให้การ์ดแสดงผลเป็นผู้จัดการแทน ซึ่งจะเพิ่มเวลาว่างให้กับซีพียูในการประมวลผลงานอื่นๆ ได้มากขึ้น ปกติแล้ว ฮาร์ดแวร์สำหรับ Direct3D โดยทั่วไปจะมีหน่วยความจำบนการ์ดอย่างน้อย 2 MB (แต่ควรจะมี 4 MB หรือมากกว่า) สำหรับใช้ในการเก็บ Bitmap (ภาพที่ประกอบขึ้นจากจุดเล็กๆ ที่เรียกว่า "Pixel"), Texture, Sprite (วัตถุที่มีการเคลื่อนที่บนจอภาพ), Overlay และอื่นๆ

2.3.4.3 DirectPaly

DirectPlay คือ ส่วนสำหรับการทำงานสำหรับผู้เล่นหลายคนผ่านระบบเครือข่ายโดยมีพื้นฐานอยู่บน User Datagram Protocol (UDP) DirectPlay ขึ้นมา เพื่อช่วยให้เราสามารถพัฒนาเกมที่เล่นผ่านเน็ตเวิร์คได้ง่ายขึ้น ซึ่งผู้เขียนโปรแกรมไม่จำเป็นต้องเขียนเกี่ยวกับ Socket ผู้เขียนโปรแกรมจะใส่ใจเพียงแค่ว่าจะเขียนอย่างไรให้โปรแกรมเกมของเรามีความสวยงาม และไม่กระตุก มากกว่าจะไปใส่ใจในเรื่องการส่งข้อมูล จะส่งไปกี่บิตดี



รูปที่ 2.9 โครงสร้างของ DirectPlay

ชุดคำสั่ง DirectPlay รองรับการเชื่อมต่อ 4 แบบด้วยกัน คือ TCP/IP , IPX , Serial และแบบ Modem ใน DirectPlay นี้เรียกว่า Service Provider โดยชุดคำสั่ง DirectPlay นั้นได้จัดเตรียมตัวแปรโครงสร้างต่าง ๆ และฟังก์ชั่นที่ใช้ในการปรับแต่ง โดยที่ผู้ใช้ไม่

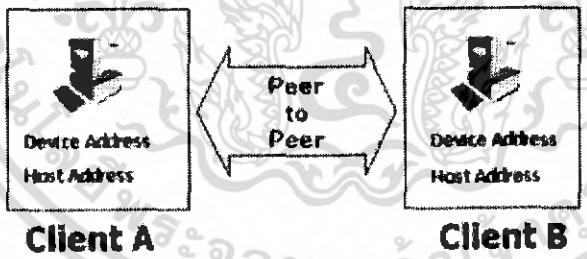
จำเป็นต้องศึกษาให้ลึกถึงเรื่องของ Socket เพียงแค่ใช้ DirectPlay ก็สามารถสร้างเกมแบบเล่นผ่านเน็ตเวิร์คได้แล้ว

นอกจาก IP Address ที่จำเป็นต้องใช้ในการเชื่อมต่อแล้ว เราจะต้องทำความเข้าใจกับ DirectPlay Address ซึ่งเป็นหมายเลขที่ระบุถึงเครื่องคอมพิวเตอร์ที่เชื่อมต่ออยู่ในระบบ เพื่อบอกถึงที่อยู่ของโฮสต์, IP Address หรือชื่อโฮสต์ หรือหมายเลขพอร์ท รวมถึงค่าต่าง ๆ ที่ใช้ในการเชื่อมต่อในแต่ละแบบ เช่น ถ้าเชื่อมต่อแบบโมเด็ม ใน DirectPlay Address นี้ก็จะมี Buad Rate ของการเชื่อมต่อว่ามีความเร็วเท่าไหร่

ในการสร้างเกมเน็ตเวิร์คโดยใช้ DirectPlay นั้น เราจะต้องสร้าง DirectPlay Address 2 ตัวด้วยกันคือ

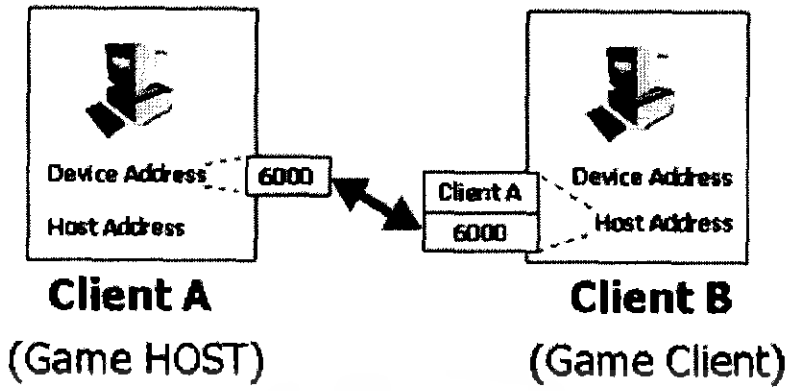
- Device Address
- Host Address

เหตุที่จะต้องสร้างทั้งสอง Address นั้นก็เพราะว่า Device Address เป็น Address ที่บอกถึงตำแหน่งเครื่องของเราเองที่กำลังรันโปรแกรมเกมนี้อยู่ แต่ Host Address นี้คือ Address ของเครื่องที่เราจะเชื่อมต่อเข้าไป โปรแกรมเกมที่ใช้ DirectPlay ทุก ๆ โปรแกรมจะต้องสร้างทั้ง Device Address และ Host Address นี้ แต่ว่าจะใช้ Device Address หรือตัว Host Address นั้นก็ขึ้นอยู่กับว่าเครื่องเราจะทำหน้าที่เป็นโฮสต์ หรือ Server ที่จะให้เครื่องอื่นเข้ามา Join



รูปที่ 2.10 การเชื่อมต่อแบบ Peer to Peer

จากรูป ถ้าเครื่องสองเครื่องนี้จะเชื่อมต่อกันแบบ Peer to Peer โดยเครื่อง A เป็นโฮสต์ของเกม และเครื่อง B เป็นเครื่องที่จะต้องเข้ามา Join ในเกม ในกรณีนี้ เราจะต้องกำหนดให้ Device Address ของเครื่อง A นั้นเปิดพอร์ทเอาไว้เพื่อรอการเชื่อมต่อ เช่น พอร์ท 6000 เพราะฉะนั้นเราจะกำหนด 6000 นี้ไปที่ Device Address ของเครื่อง A และสำหรับเครื่อง B ที่จะเข้ามา Join ก็จะชี้มาที่เครื่อง A ดังรูป



รูปที่ 2.11 การเชื่อมต่อแบบ Client/Server

จากรูปจะเห็นว่าเมื่อเครื่อง A เป็นโฮสต์ก็จะเปิดพอร์ท 6000 เอาไว้ เพื่อรอการเชื่อมต่อ ส่วนเครื่อง B ที่จะเข้ามา Join เกมนั้น นั้นก็จะกำหนดชื่อ Host ชี้ไปที่ IP ของเครื่อง A และกำหนดหมายเลขพอร์ทไว้ที่ 6000 โดยจะกำหนดให้กับ Host Address นั่นเอง จากตรงนี้สรุปได้ว่าเครื่องที่เป็นโฮสต์ จะใช้แค่ Device Address และเครื่องที่จะเข้ามา Join จะใช้แค่ Host Address

2.3.5 การใช้งานฟังก์ชั่นเบื้องต้นของ Direct3D

การจะใช้งานฟังก์ชั่นของ DirectX ได้นั้นจำเป็นที่จะต้องนำเข้าไลบรารีต่างๆดังนี้

```
using System;
using System.Drawing;
using System.Windows.Forms;
using System.ComponentModel;
using Microsoft.DirectX;
using Microsoft.DirectX.Direct3D;
```

การทำงานทุกอย่างจะเริ่มต้นจากฟังก์ชั่น Main ดังตัวอย่าง

```
static void Main()
{
        using (myForm = new WinForm())
        {
                Application.Run(myForm);
        }
}
```

โดยที่เราสามารถกำหนดค่าเริ่มต้นให้กับฟอร์มของเราได้ที่ Constructer ของคลาส โดยในตัวอย่างนี้จะกำหนดให้หน้าต่างมีขนาด 500 x 500 พิกเซล และมีหัวข้อของหน้าต่างว่า DirectX Tutorial สุดท้ายเป็นการกำหนด Style ให้กับหน้าต่างของเรา

```
public class WinForm : System.Windows.Forms.Form
{
        private System.ComponentModel.Container components = null;
        public WinForm()
        {
                this.components = new System.ComponentModel.Container();
                this.Size = new System.Drawing.Size(500,500);
                this.Text = "DirectX Tutorial";
                this.SetStyle(ControlStyles.AllPaintingInWmPaint |
                                ControlStyles.Opaque, true);
        }
}
```

สิ่งที่ต้องทำต่อมาคือการกำหนดค่าให้ Device Device คือตัวติดต่อที่สามารถให้เข้าถึงการใช้งานการ์ดจอได้โดยตรงเริ่มต้นต้องประกาศแอททริบิว Device ก่อนจากนั้นจึงกำหนดค่าให้มันซึ่งอาจสร้างเป็นฟังก์ชั่นแล้วเรียกใช้โดย Main ได้ดังนี้

```
public class WinForm : System.Windows.Forms.Form
{
        private Device device;
        public void InitializeDevice()
        {
            PresentParameters presentParams = new PresentParameters();
            presentParams.Windowed = true;
            presentParams.SwapEffect = SwapEffect.Discard;
            device = new Device(0, DeviceType.Hardware, this,
                    CreateFlags.SoftwareVertexProcessing, presentParams);
        }

}
```

จากฟังก์ชั่นข้างบนนี้จะเห็นได้ว่ามีตัวแปร PresentParameters ตัวแปรนี้จะเป็นตัวแปรไว้สำหรับกำหนดค่าให้ Device ว่าจะมีลักษณะอย่างไรในตัวอย่างนี้จะกำหนดให้การแสดงผลเป็นแบบหน้าต่างไม่ใช่เต็มจอ SwapEffect.Discard หมายถึงการที่กำหนดให้วาดภาพเลยโดยที่ไม่ต้องมี Back Buffer คอยสับเปลี่ยนภาพในขณะที่แสดงผล

ในบรรทัดสุดท้ายจะทำการสร้าง Device ของเราขึ้นมาโดยที่พารามิเตอร์ตัวแรกนั้นหมายถึงอุปกรณ์การ์ดจอที่ต้องการใช้ (การ์ดจอโดยปกติจะมีค่า 0) พารามิเตอร์ต่อมา DeviceType.Hardware จะหมายถึงเราจะทำงานบน Hardware (การ์ดจอ) แต่ถ้าหากเราไม่มีการ์ดจอ เราสามารถใช้ DeviceType.Reference ในการแสดงผลได้แต่ความสามารถในการประมวลผลช้ากว่า พารามิเตอร์ถัดมา this หมายถึงว่าให้ใช้การแสดงผลในคลาสนี้ พารามิเตอร์ถัดมาคือ Flags ของ Device ซึ่งสามารถเปลี่ยนไปได้หลายอย่างมากจะไม่อธิบายในที่นี้ พารามิเตอร์ตัวสุดท้าย คือให้ค่า PresentParameter กับ Device ที่เราได้กำหนดไว้แล้วในตอนแรก

จนถึงขั้นตอนนี้เราสามารถสร้างฟอร์มหน้าต่างได้ หนึ่ง หน้าต่างแล้วแต่จะยังคงเป็นหน้าต่างที่ว่างเปล่าอยู่เราสามารถวาดภาพต่างๆในหน้าต่างนี้ได้โดยเรียกใช้ฟังก์ชั่น OnPaint ซึ่งฟังก์ชั่นนี้จะถูกเรียกใช้อัตโนมัติเมื่อมีการเรียกใช้คำสั่ง Application.Run

```
protected override void OnPaint(System.Windows.Forms.PaintEventArgs e)
{
   device.Clear(ClearFlags.Target, System.Drawing.Color.CornflowerBlue, 1.0f, 0);

   device.BeginScene();
   ...
   device.EndScene();

   device.Invalidate();
   device.Present();
}
```

ในฟังก์ชั่นนี้เราจะวาดสิ่งที่ต้องการลงในระหว่าง device.BeginScene กับ device.EndScene เมื่อทำการวาดเสร็จแล้วจะใช้คำสั่ง device.Invalidate ในการกำหนดให้เราสามารถย่อหรือขยายหน้าต่างของเราได้ สุดท้ายจะใช้คำสั่ง device.Present เพื่อทำการวาดภาพของเราออกมา

เมื่อนำทุกอย่างมารวมกันสามารถแสดงได้ดังนี้

```
using System;
using System.Drawing;
using System.ComponentModel;
using System.Windows.Forms;
using Microsoft.DirectX;
using Microsoft.DirectX.Direct3D;

namespace DirectX_Tutorial
{

        public class WinForm : System.Windows.Forms.Form
        {
                private Device device;
                private System.ComponentModel.Container components = null;

                public WinForm()
                {
                        InitializeComponent();
                        this.SetStyle(ControlStyles.AllPaintingInWmPaint |
                                ControlStyles.Opaque, true);
                }

                public void InitializeDevice()
                {
                        PresentParameters presentParams = new PresentParameters();
                        presentParams.Windowed = true;
                        presentParams.SwapEffect = SwapEffect.Discard;
                        device = new Device(0, DeviceType.Hardware, this,
                                CreateFlags.SoftwareVertexProcessing, presentParams);
                }

                protected override void OnPaint(System.Windows.Forms.PaintEventArgs e)
                {

                        device.Clear(ClearFlags.Target, Color.DarkSlateBlue , 1.0f, 0);

                        device.BeginScene();
                        // draw something here
                        device.EndScene();

                        device.Present();
```

```
                this.Invalidate();
        }


        private void InitializeComponent()
        {
                this.components = new System.ComponentModel.Container();
                this.Size = new System.Drawing.Size(500,500);
                this.Text = "DirectX Tutorial";
        }

        static void Main()
        {
                using (myForm = new WinForm())
                {
                        myForm.InitializeDevice();
                        Application.Run(myForm);
                }
        }
    }
}
```

2.3.6 การใช้งานฟังก์ชั่นเบื้องต้นของ Direct Play

อย่างที่ได้กล่าวไปแล้ว DirectPlay สามารถทำงานได้ 2 แบบ คือ Peer to Peer และ Client/Server ในที่นี้จะขอกล่าวถึงเฉพาะการทำงานแบบClient/Server ในที่นี้จะพูดถึงการทำงานในการรับส่งข้อมูลเมื่อมีการเชื่อมต่อกันเรียบร้อยแล้ว ซึ่งการตั้ง Server และการเชื่อมต่อ ของ Client จะกล่าวไว้ในบทที่ 3

ในขั้นตอนแรกเราจะประกาศแอททริบิวไว้ 2 ตัวคือ

```
private Server ServerConnection = null;
private Client ClientConnection = null;
```

เมื่อการเชื่อมต่อของทั้งสองฝั่งเสร็จสิ้นแล้วเราจะได้ นำแอทริบิว 2 ตัวนี้มาใช้การจะส่งข้อมูลเราสามารถส่งได้ 3 แบบคือ ส่ง Primitive type , Array และ Structure ที่เรากำหนดไว้ในการส่งข้อมูลเราสามารถเขียนชุดของข้อมูลได้ดังนี้

```
NetworkPacket packet = new NetworkPacket();
packet.Write("test send some data");
```

โดยที่หากฝั่ง Server ต้องการที่จะส่งข้อมูลจะใช้ ฟังก์ชั่นดังนี้

```
ServerConnection.SendTo((int)PlayerID.AllPlayers,packet,0
,SendFlags.Guaranteed | SendFlags.NoLoopback);
```

พารามิเตอร์ตัวแรกจะเป็นตัวกำหนดว่าจะส่งข้อมูลไปให้ใครในที่นี้กำหนดให้ส่งไปให้กับ Client ทุกๆ ตัวพารามิเตอร์ต่อมาคือการบอกว่าจะส่งอะไรไปในที่นี้เราจะส่งชุดของข้อมูลที่เราได้เขียนเอาไว้แล้ว พารามิเตอร์ต่อมาคือกำหนด time out หากเป็น 0 คือไม่มี time out สุดท้ายจะเป็นการกำหนด flags ให้กับการส่งข้อมูลซึ่งจะไม่ขออธิบายในที่นี้

หากฝั่ง Client ต้องการจะส่งข้อมูลจะใช้ฟังก์ชั่นดังนี้

```
ClientConnection.Send(packet, 0, SendFlags.Guaranteed);
```

ซึ่งจะคล้ายกับ การส่งข้อมูลของ Server จะแตกต่างกันตรงที่ Client สามารถส่งให้กับ Server ได้เพียงอย่างเดียว เพราะฉะนั้นจึงไม่มีพารามิเตอร์ที่บอกว่าจะส่งไปหาใคร

ในการรับข้อมูลที่มาจากเครือข่ายทั้งฝั่ง Client และ Server จะทำงานเหมือนกันคือ สร้างตัวแปรที่จะรับข้อมูลให้ตรงกับข้อมูลที่มาจากเครือข่ายและเมื่อรับแล้ว ต้องบอกด้วยว่าจะรับข้อมูลประเภทใด ถ้าประเภทของข้อมูลไม่ตรงกันจะไม่รับข้อมูลนั้นๆ

```
float[,] attr = new float[6,8];
int numPlayer;
string text;

attr = (float[,])e.Message.ReceiveData.Read(typeof(float),6,8);
numPlayer = (int)e.Message.ReceiveData.Read(typeof(int));
text = (string)e.Message.ReceiveData.ReadString();
```

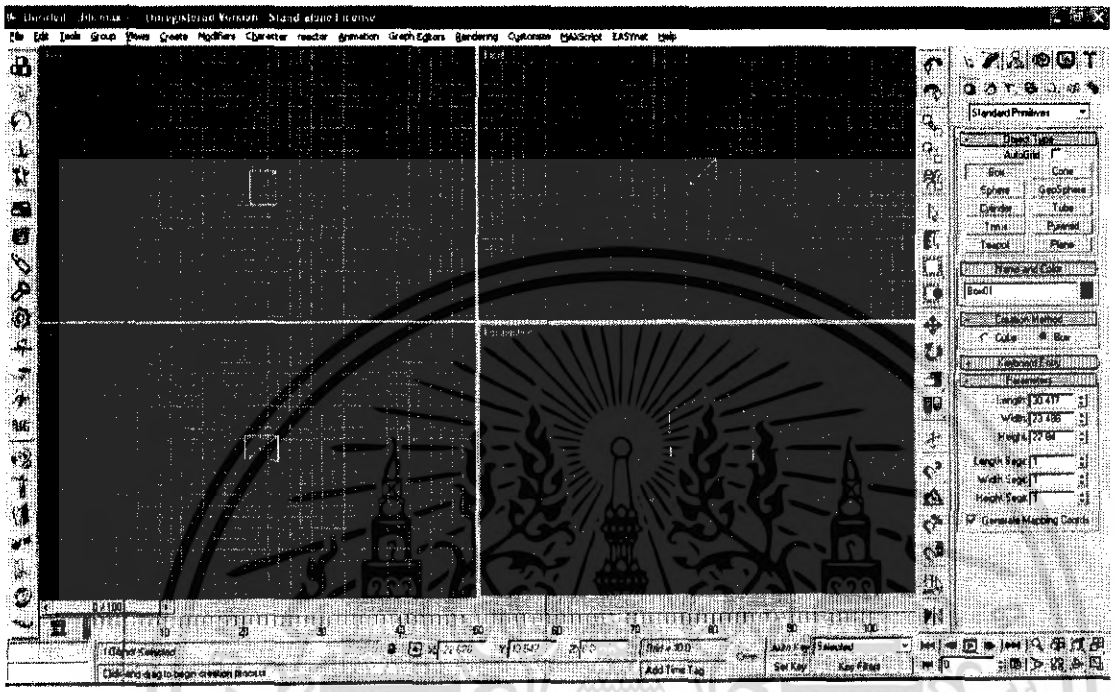
2.4 การใช้งานเกี่ยวกับ 3D Studio Max7

2.4.1 การกำหนดการเคลื่อนไหว

1. สร้างโมเดลที่ใช้ในเกม โดยเปิดโปรแกรม 3d studio max 7 ขึ้นมาแล้วที่ FILE -> New จะได้หน้าต่างดังรูป

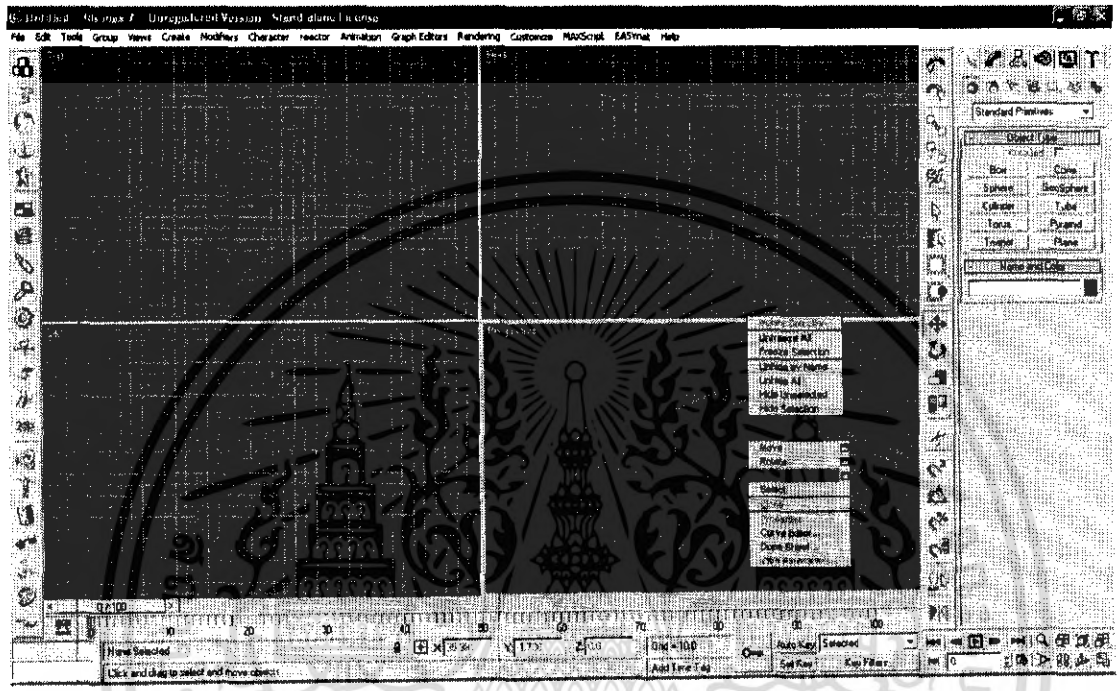
รูปที่ 2.12 แสดงโปรแกรม3D Studio Max

2. เลือกกลุ่มของวัตถุที่ต้องการจะสร้าง ในที่นี้ทำการเลือกชุดวัตถุมาตรฐาน หลังจากนั้นเราก็สามารถเลือกวัตถุต่างๆของชุดวัตถุมาตรฐานมาสร้างเป็นโมเดลที่ต้องการได้



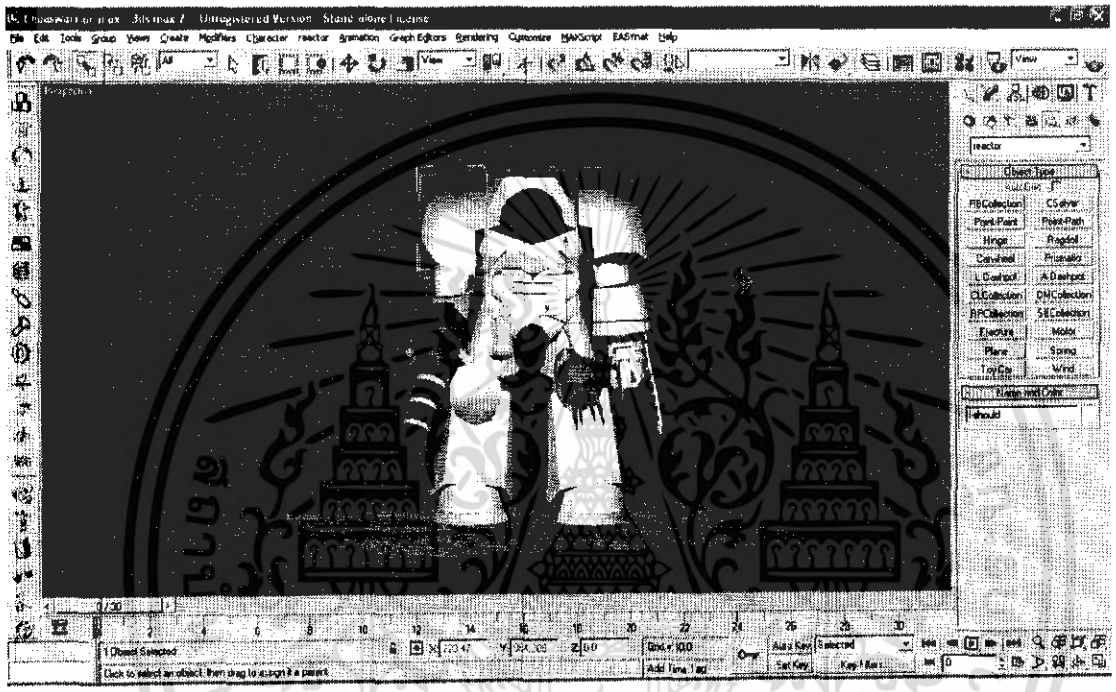
รูปที่ 2.13 การสร้างวัตถุ

3. เราสามารถดำเนินการย่อขยาย เลื่อนพิกัด หรือหมุนวัตถุได้ตามต้องการ โดยการเลือกที่แถบเครื่องมือ หรือการคลิ๊กขวาที่ตัวออบเจ๊กแล้วเลือกสิ่งที่เราต้องการแก้ไข



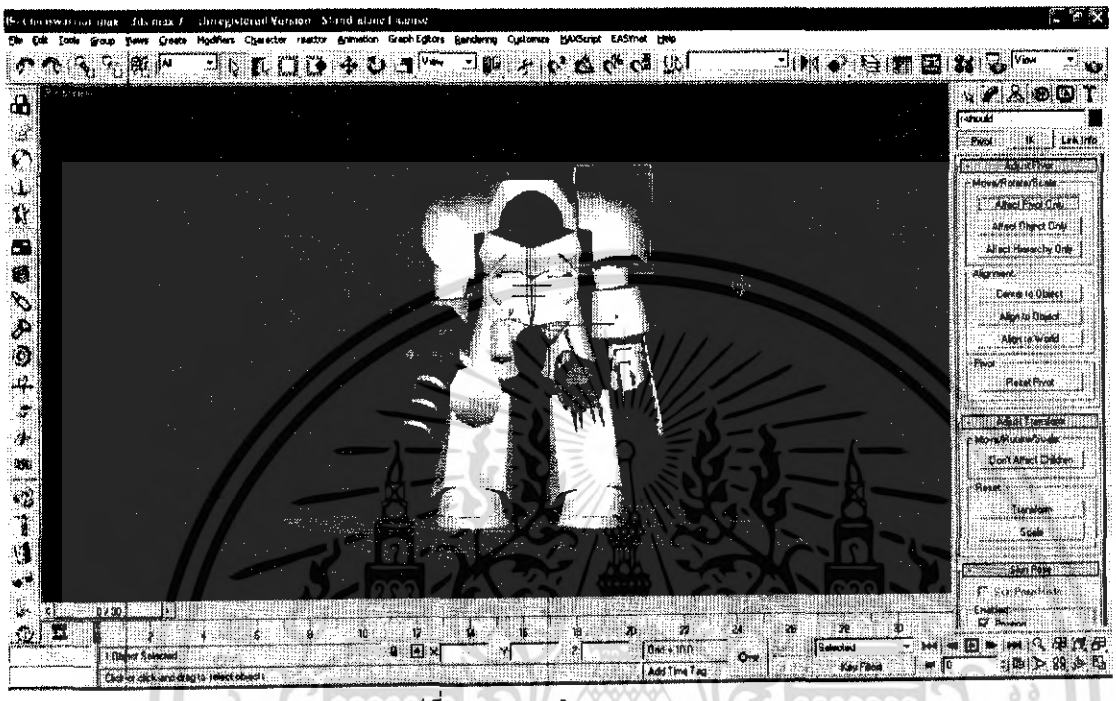
รูปที่ 2.14 การขยายขนาดวัตถุ

4. หลังจากทำการสร้างโมเดลได้ตามต้องการแล้ว เราต้องทำการเชื่อมต่อชิ้นส่วนแต่ละชิ้นส่วนเข้าด้วยกันเพื่อให้ชิ้นส่วนแต่ละชิ้นมีความสัมพันธ์กัน โดยทำการเลือกวัตถุลูกที่จะทำการเชื่อมต่อหลังจากนั้นให้กดปุ่มเลือกและเชื่อมในแถบเครื่องมือ แล้วทำการลากจากวัตถุลูกไปยังวัตถุแม่ที่เราต้องการจะทำให้เราได้ต้นไม้ที่แสดงลำดับชั้นของวัตถุที่เราต้องการ



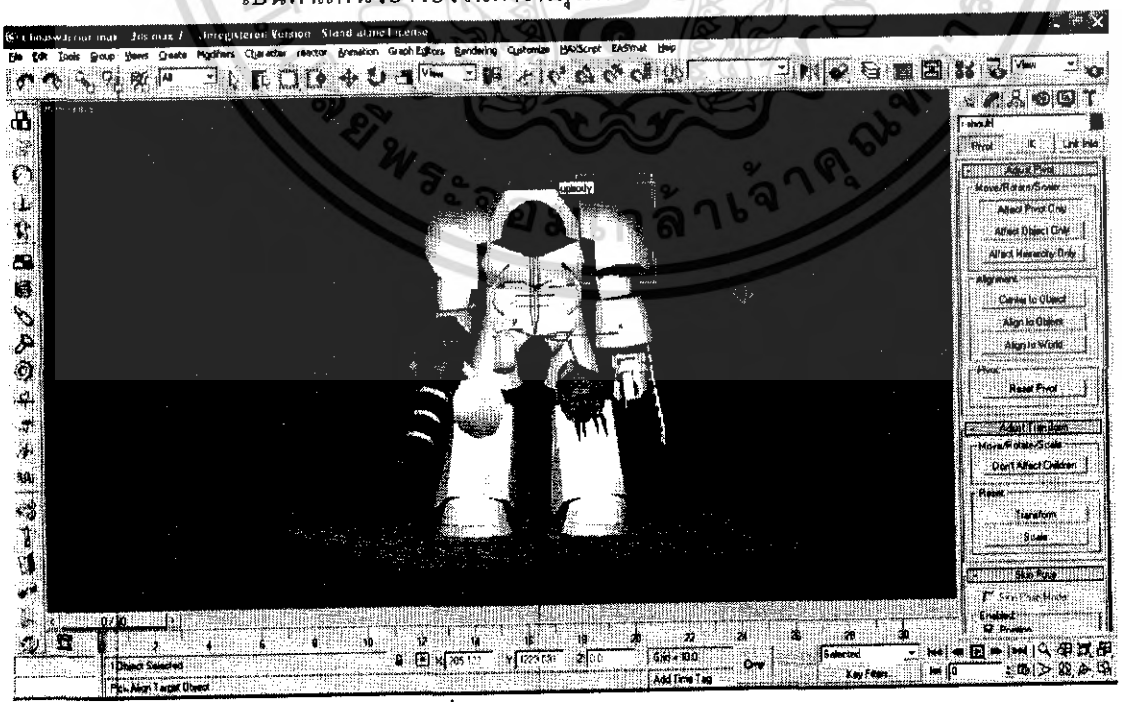
รูปที่ 2.15 การเชื่อมต่อวัตถุ

5. เมื่อเชื่อมวัตถุได้ตามต้องการแล้วเราต้องกำหนดจุดหมุนของแต่ละโมเดลให้ถูกต้อง โดยเลือกวัตถุที่ต้องการให้หมุนได้ก่อน จากนั้นให้ไปที่หน้าต่างลำดับชั้นต่อจากนั้นกดปุ่ม Affect Pivot Only เพื่อเลือกทำงานกับจุด Pivot เท่านั้น



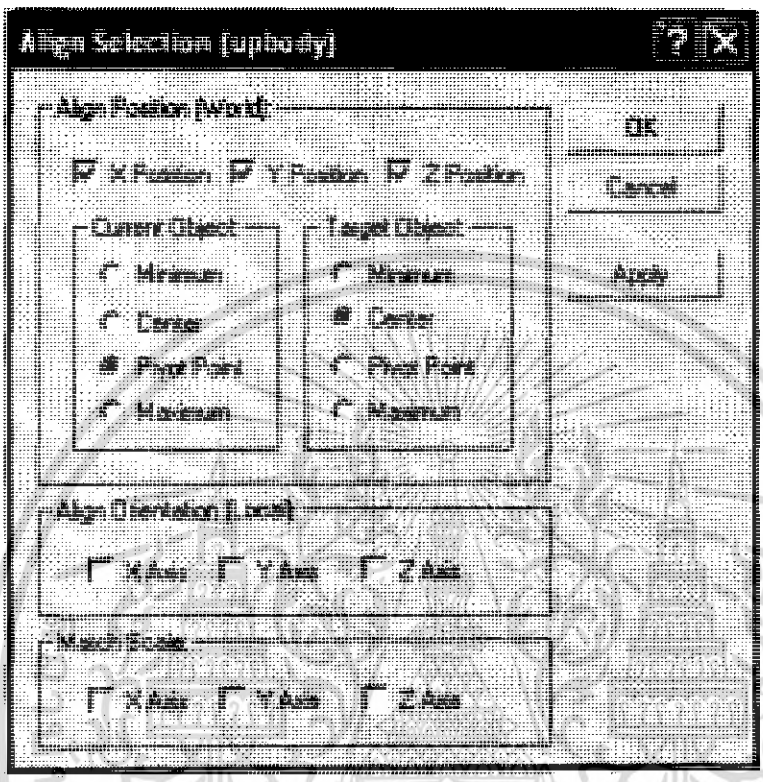
รูปที่ 2.16 การกำหนดจุดหมุน

6. เลือกวัตถุที่ต้องการ Align ในที่นี้ให้เลือกส่วนหลังของตัวโมเดลที่เราจะใช้เป็นตำแหน่งอ้างอิงในการหมุนหัวใหล่ที่เราเลือกไว้



รูปที่ 2.17 การอ้างอิงจุดหมุน

7. หลังจากเลือกวัตถุที่เป็นฐานในการหมุนแล้ว จะปรากฏหน้าต่างดังภาพให้เราทำการเลือกว่าจะวางจุด Pivot ไว้ตรงส่วนไหนที่ฐาน ในที่นี้เลือกวางเข้าไปที่ศูนย์กลางของฐาน

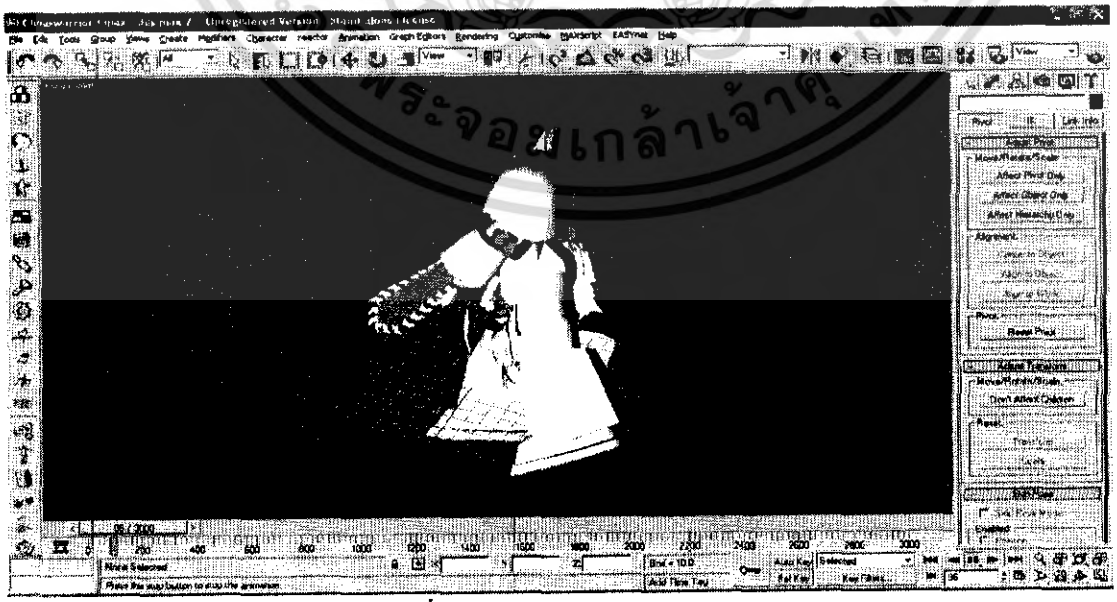


รูปที่ 2.18 กำหนดจุดที่ใช้หมุน

8. เมื่อเราทดลองหมุนหัวใหล่จะทำให้โนดที่มีออบเจ็กหัวใหล่เป็นโนดแม่หมุนตามไปด้วย โดยมีจุดศูนย์กลางในการหมุนอยู่ที่ตัวส่วนบน

รูปที่ 2.19 แสดงการหมุน

9. ทำการกำหนดจุดหมุนให้กับข้อต่อทั้งหมดเราก็จะได้โมเดลที่มีการเคลื่อนไหวตามต้องการ



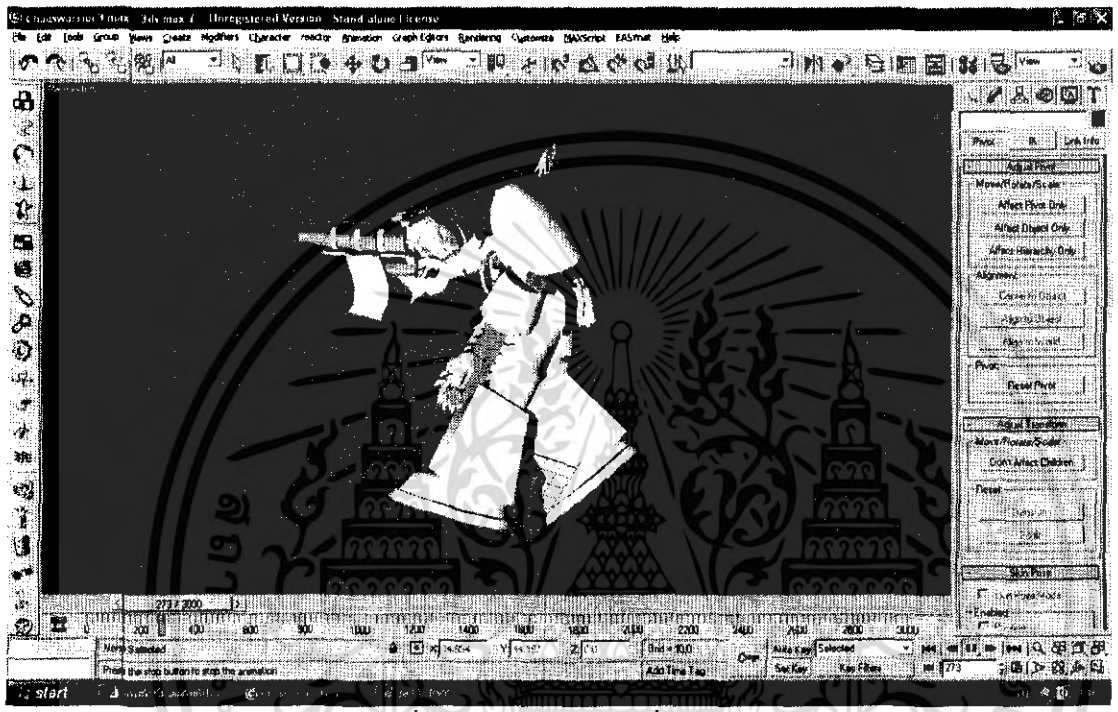
รูปที่ 2.20 ทดสอบการหมุน

10. ทำการสร้างแอนิเมชั่นให้กับโมเดลที่มีจุดหมุนครบแล้ว จากการควบคุมการเคลื่อนแบบกลไกการควบคุมไปข้างหน้า โดยเป็นการควบคุมแบบเรียงลำดับจากวัตถุที่อยู่ลำดับสูงสุดลงไปสู่วัตถุที่อยู่ลำดับล่างต่อๆไป โดยทำการเลือกที่หัวใหล่ของหุ่นยนต์ จากนั้นให้กดปุ่มออโต้คีย์เพื่อเข้าสู่โหมดกำหนดการเคลื่อนไหว หลังจากนั้นให้ลากเมาส์เลื่อนตัวเลื่อนเวลาไปยังเฟรมที่ต้องการ หลังจากนั้นให้ทำการกำหนดการเคลื่อนไหวให้กับหัวใหล่ของหุ่นยนต์ตามต้องการ 3dMax จะทำการกำหนดคีย์เฟรมให้เราอัตโนมัติ

รูปที่ 2.21 การกำหนดคีย์เฟรม

11. ทำการกำหนดคีย์เฟรมให้กับหัวไหล่จนครบ หลังจากนั้นให้ออกจากโหมดกำหนดการเคลื่อนไหวด้วยการคลิ๊กปุ่มออโต้คีย์ให้เป็นปกติ
12. ทำการกำหนดคีย์เฟรมให้กับส่วนอื่นๆของโมเดลให้ครบ

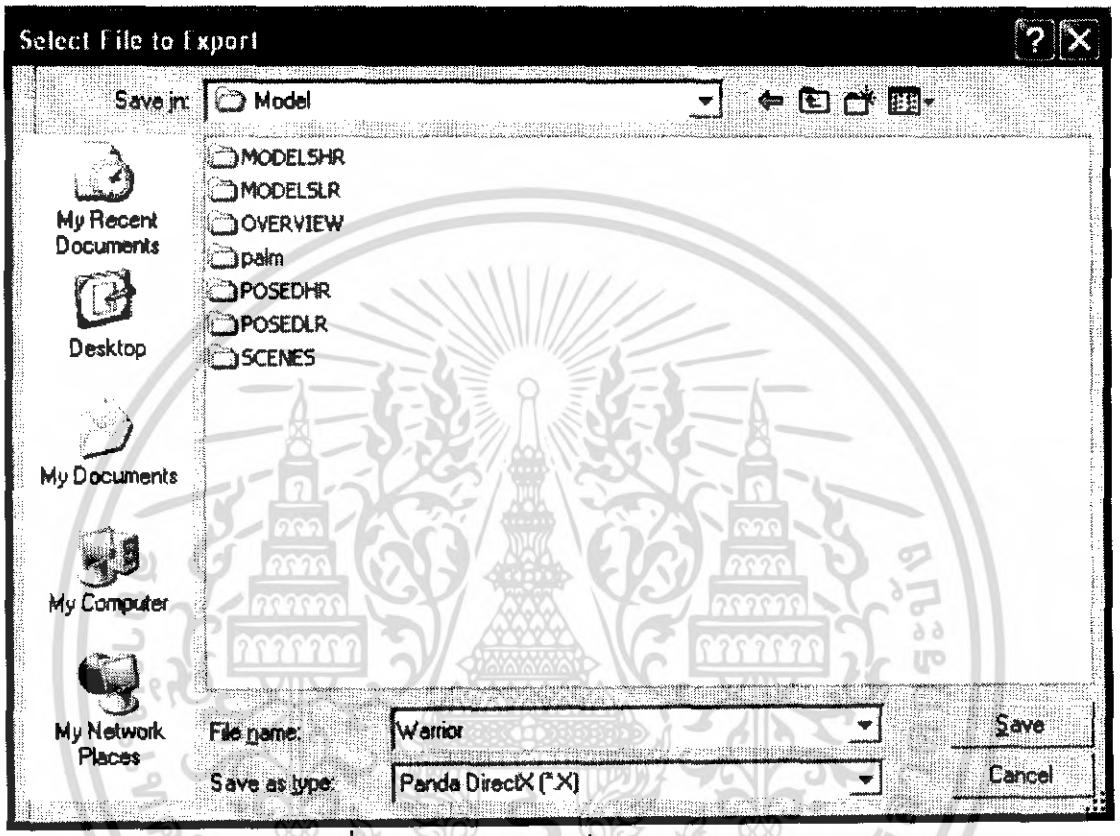
13. ทดลองดูผลลัพธ์จากการเคลื่อนที่จะพบว่าเราได้งานแอนิเมชั่นจากการเซ็ตคีย์ให้ข้อต่อต่างๆ จากการทำงานในระบบกลไกการควบคุมไปข้างหน้าตามที่ต้องการเรียบร้อยแล้ว



รูปที่ 2.22 การเล่นภาพเคลื่อนไหว

2.4.2 การส่งออกโมเดล เป็นไฟล์ .X โดยใช้โปรแกรม Panda DirectX

1. ทำการก็อปปี้ไฟล์ PandaDXExport6.dle ไปไว้ที่โฟล์ดเดอร์ 3dMax/lib
2. จากเมนูบาร์ทำการเลือก File->Export จะได้ดังภาพ ให้ทำการเลือกชนิดของไฟล์ที่จะทำการบันทึกเป็น Panda DirectX(*.X)



รูปที่ 2.23 การบันทึกไฟล์ .X

3. ทำการกำหนดค่าให้กับโมเดลก่อนที่จะทำการส่งออกไป โดยในกรณีโมเดลที่จะส่งออกไปเป็นแอนิเมชั่นสิ่งสำคัญที่จะต้องกำหนดให้กับโมเดลเสมอคือเช็คบอกซ์ Include Animation(requires frames)ที่อยู่ในแทบ 3DS Max Object ส่วนรายละเอียดของเช็คบอกซ์ต่างๆมีดังนี้

Mesh definition เป็นการปรับกรอบให้กับตัวเลือกเมช

Material เป็นการกำหนดแมททีเรียลให้กับโมเดล

Include Animation (requires frames) กรณีส่งออกไฟล์แบบมีเลือกไว้ ถ้าไม่มีเฟรมหรือแอนิเมชันไม่ต้องเลือก

Bones ทำการเลือกถ้าเราใช้วัตถุกระดูกที่ใช้ modifier Skin หรือ Physique

Optimize mesh ทำการเลือก None หรือ Normal ไว้กรณี Optimized แล้วใช้ไม่ได้

Geometric เป็นการกำหนดให้กับโมเดลที่เป็นทรงเรขาคณิต

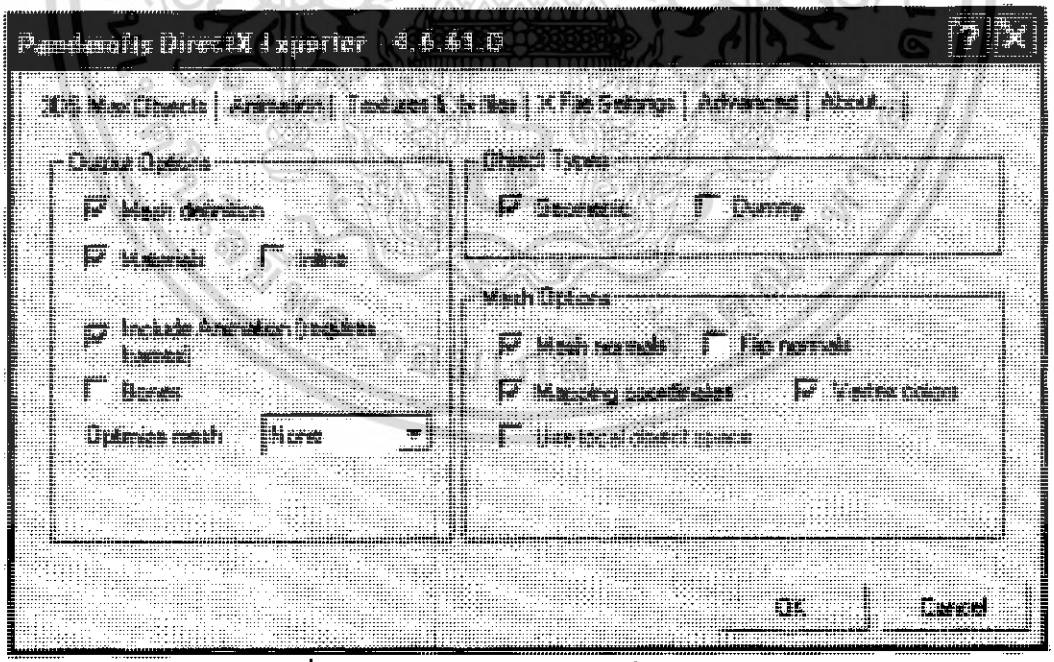
Dummy เป็นการกำหนดสำหรับโมเดลที่มีการใช้ดัมมี่

Mesh normals เป็นการกำหนดเมชให้เป็นแบบปกติ

Mapping coordinates เป็นการกำหนดค่า UV เพื่อไม่ให้ลายเสีย

Flip normals ไว้กลับด้านนอมอล

Vertex colors กำหนดแพนดาลงสีตามจุดยอด



รูปที่ 2.24 กำหนดคุณสมบัติไฟล์ .X

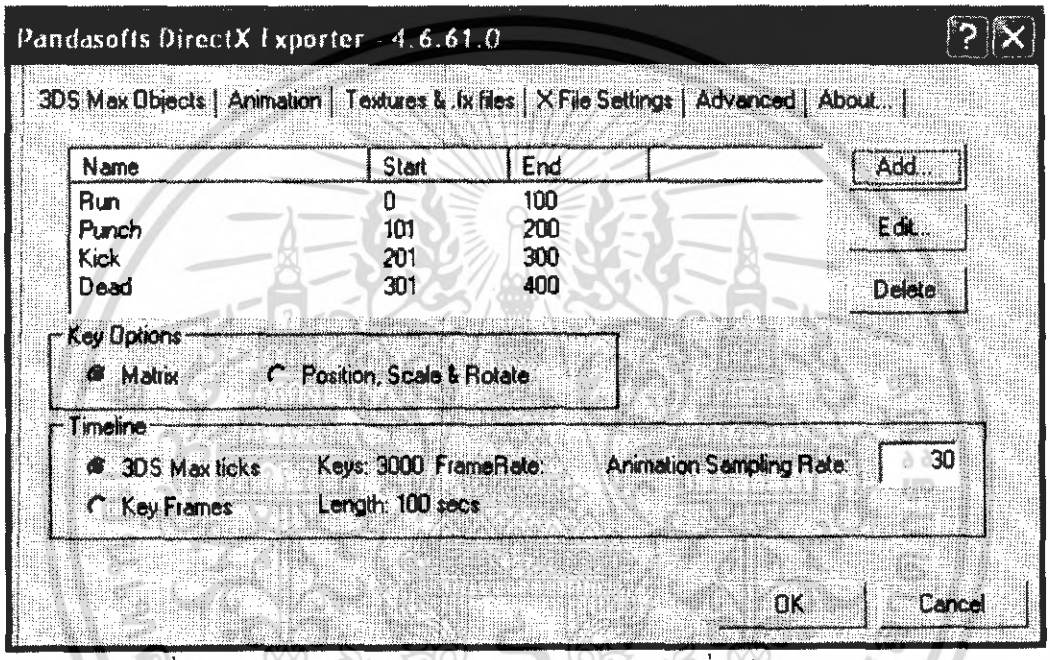
4. ที่แทบ Animation ทำการกำหนดจำนวนแทร็คและกำหนดจำนวนเฟรมให้กับแต่ละแทร็คตามต้องการ โดยจำนวนแทร็คมักเท่ากับจำนวนพฤติกรรมที่โมเดลนั้นสามารถกระทำได้ ส่วนรายละเอียดของตัวเลือกต่างๆมีดังนี้

Matrix กำหนดให้คีย์รวมกันเป็นเมตริกซ์

Position Scale Rotate กำหนดให้คีย์ต่างๆแยกออกจากกัน

3DS Max ticks กำหนดให้ช่วงเวลาเป็นแบบเฟรม

Key Frames กำหนดช่วงเวลาเป็นหน่วยวินาที



รูปที่ 2.25 การกำหนด เฟรม ในการเล่นแต่ละการเคลื่อนไหว

5. ที่แทบ X Files Setting ในส่วนของเฟรม X Files Animation Option ให้ทำการเช็คถูกที่เช็คบอกซ์ Include Animation Option เพื่อเป็นการบอกว่าจะมีการรวมคุณลักษณะของ แอนิเมชั่นเข้าไปกับไฟล์ที่ทำการส่งออก จากนั้นให้กดตกลงเราก็จะได้ไฟล์ .X ตามต้องการ ส่วนรายละเอียดของตัวเลือกต่างๆมีดังนี้

Text ทำการส่งออก .x เป็นข้อความที่อ่านได้ขนาดใหญ่

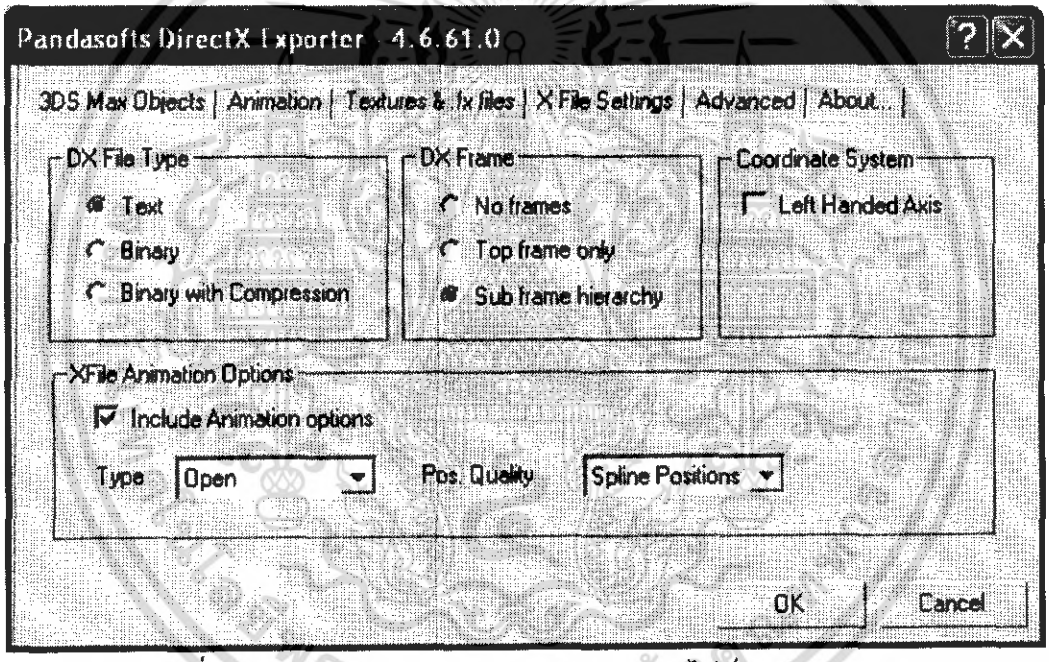
Binary ทำการส่งออก .x เป็นขนาดธรรมดาทำให้อ่านได้เร็ว

Binary with Compression ทำการส่งออก .x เป็นขนาดเล็กทำให้อ่านได้ช้า

No Frames เลือกไว้ถ้าไม่มีโมเดลไม่มีแอนิเมชั่น

Top Frame only เลือกถ้าโมเดลมีแอนิเมชั่นแบบไม่ซับซ้อน

Sub frame hierarchy ถ้าโมเดลมีการใช้กระดูกให้เลือก



รูปที่ 2.26 การกำหนดคุณสมบัติของการส่งออกไฟล์ .X

5. ทดสอบผลลัพธ์ของไฟล์ .X โดยทำการเปิดโปรแกรม Mesh Viewer ที่มากับ DirectX SDK จากนั้นทำการเลือกไฟล์ที่ต้องการทดสอบจะได้ผลลัพธ์ดังภาพ

รูปที่ 2.27 การทดสอบภาพเคลื่อนไหว

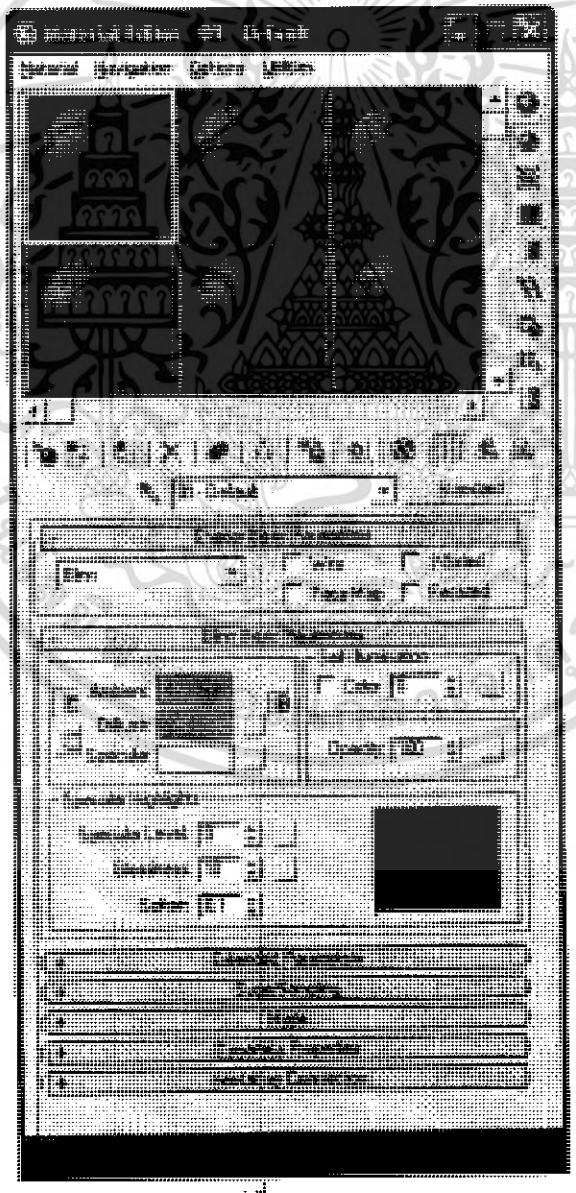
6. เราสามารถทดสอบดูผลของแต่ละแทร็คได้โดยไปที่เมนู Animation->Animations

2.4.3 การสร้างและควบคุมพื้นผิวให้กับโมเดล

การกำหนดพื้นผิวให้โมเดลที่สร้างขึ้นจะถูกแบ่งออกเป็นเรื่องใหญ่ๆ 2 เรื่องคือ Shading และ Texture

- **Shading** คือการกำหนดรายละเอียดของตัวพื้นผิว เช่น ความมันวาว หรือการสะท้อนของผิว
- **Texture** คือการกำหนดลวดลายของพื้นผิด เช่น พื้นมีลายเป็นไม้ รวมทั้งการแปะรูปภาพหรือลวดลายที่สร้างขึ้นเองลงบนพื้นผิว

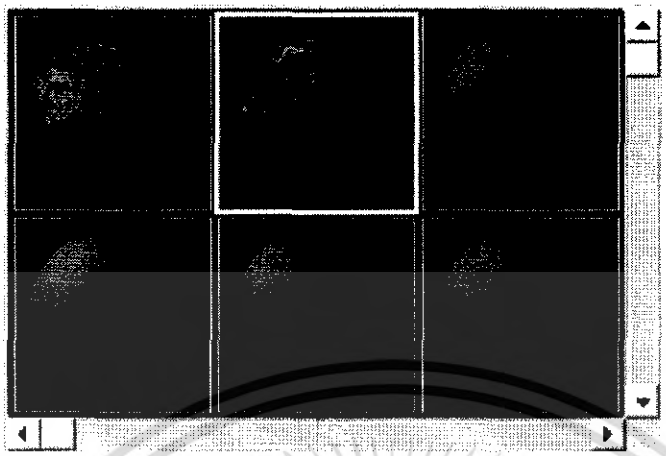
สำหรับ 3d studio max 7 การควบคุมรายละเอียดทั้งสองข้อที่กล่าวมานี้สามารถทำได้ด้วย การกำหนดค่าต่างๆ จากหน้าต่าง Material Editor โดยไปที่ Rendering -> Material Editor จากโปรแกรม 3d studio max 7 ก็จะขึ้นหน้าต่าง Material Editor มาให้ดังรูป



รูปที่ 2.28 Material Editor

ส่วนประกอบของหน้าต่างควบคุมการสร้างพื้นผิด (Material Editor)

- **Material Slot** ช่องแสดงตัวอย่างของ Material ที่สร้างขึ้น



รูปที่ 2.29 Material Slot

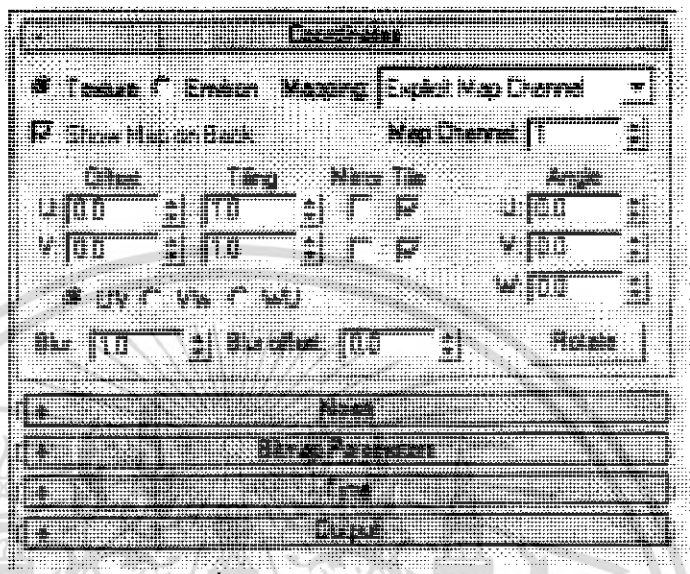
- **Material Toolbar** ชุดเครื่องมือสำหรับควบคุมหน้าต่าง Material Editors ประกอบไปด้วยเครื่องมือสำหรับเลือก ลบ และเรียก Material ที่เก็บไว้ในโปรแกรมขึ้นมาใช้งาน

รูปที่ 2.30 Material Toolbar

- **Material Editors Options** ชุดเครื่องมือควบคุม Option การแสดงผลในหน้าต่าง Material Editors

รูปที่ 2.31 Material Editors Options

- **Rollouts** เป็นส่วนที่เราจะเข้าไปกำหนดค่าต่างๆ เพื่อเปลี่ยนแปลงรายละเอียดของพื้นผิวให้เป็นไปตามที่เราต้องการ ในการทำงานการกำหนดค่าต่างๆ



รูปที่ 2.32 Rollouts

การปรับคุณสมบัติของพื้นผิวให้กับโมเดล

- เปิดหน้าต่าง Material Editors ขึ้นมา
- แดรกเมาส์ลาก Material จาก Slot มายังโมเดลที่ต้องการ
- ภายในหน้าต่าง Material Editors > Blinn Basic Parameters Rollout คลิกที่ช่องสีหลัง Diffuse เพื่อเปลี่ยนสีให้กับ Material

บทที่3
ขั้นตอนการดำเนินงานวิจัย

ลำดับขึ้นตอนในการดำเนินงานวิจัยทั้งหมดแบ่งออกเป็น 3 ส่วน ได้แก่ ส่วนของการวิเคราะห์และออกแบบเกม ส่วนของการลงมือปฏิบัติจริง และส่วนของการทดสอบเกม โดยแต่ละขั้นตอนจะประกอบด้วยขั้นตอนย่อยๆดังนี้

1. **ขั้นตอนการวิเคราะห์และออกแบบเกม**
   - การวิเคราะห์ความต้องการว่างต้องการจะสร้างเกมในรูปแบบใดไม่ว่าจะเป็นเนื้อหาภายในเกม กติกาการเล่น ระบบการเล่น แนวทางการเล่น รูปแบบตัวละคร ฉาก เสียง เนื้อหาของเกม ว่าควรมีรูปแบบใด
   - การออกแบบหน้าจอที่ใช้ติดต่อกับผู้เล่น ต้องเป็นรูปแบบที่ไม่ซับซ้อนสามารถเข้าใจได้ทันที และดึงดูดความสนใจ
   - การออกแบบวิธีการจัดเก็บข้อมูลที่จำเป็นต้องใช้ภายในเกม โดยที่ต้องใช้พื้นที่ในการจัดเก็บน้อยเป็นระเบียบ และสามารถนำมาใช้ได้ง่ายเพื่อเพิ่มประสิทธิภาพการทำงานให้กับเกม
   - ออกแบบวิธีการเขียนโปรแกรม และเลือกใช้ฮาร์ดแวร์ , ซอฟต์แวร์ รวมถึง คอมโพเนนต์ และ เครื่องมือต่างๆที่นำมาใช้ในการสร้างเกม
2. **ขั้นตอนการลงมือปฏิบัติการจริง**
   - ด้านภาพและกราฟิก การทำงานในส่วนนี้จะใช้โปรแกรมทางด้านกราฟิกเข้ามาช่วยในการสร้างฉากที่ได้ทำการออกแบบไว้แล้ว
   - ด้านซาวน์เอฟเฟกต์ การทำงานในส่วนนี้จะใช้โปรแกรมการสร้างเสียงต่างๆมาช่วยในการสร้างเสียงประกอบสำหรับเนื้อหาภายในเกม
   - ด้านโปรแกรมมิ่ง การทำงานส่วนนี้จะเกี่ยวข้องกับการเขียนคำสั่งต่างๆ โดยมีการนำ ฉาก ตัวละคร และเสียงประกอบ ที่ได้สร้างไว้มาใช้ในส่วนนี้ ซึ่งมีการจัดลำดับขั้นตอนการทำงานดังนี้
     - ใช้โปรแกรม Visual C# ในส่วนการเขียนและแก้ไขซอร์สโค้ดของเกมส์
     - ใช้ DirectX9 SDK ในการช่วยพัฒโปรแกรมเกม
     - นำฉาก ตัวละคร และเสียง มาประกอบเข้าด้วยกัน

3. ขั้นตอนการทดสอบเกม

การทำงานในส่วนนี้จะเป็นขั้นตอนในการทดสอบเกมที่สร้างขึ้นว่ามีความสมบูรณ์มากน้อยเพียงใด เพื่อหาข้อผิดพลาดและประเมินประสิทธิภาพของเกม เมื่อพบข้อผิดพลาดก็จะทำการแก้ไขโดยทันที

3.1 ขั้นตอนการวิเคราะห์และออกแบบเกม

3.1.1 การออกแบบรูปแบบ กติกาการเล่นและระบบเกม

3D Game Adventure เป็นเกมแนวเดินตีมุมมองบุคคลที่ 3 โดยจะมีลักษณะเป็นทีมแบ่ง เป็น 2 ฝ่าย โดยมีจุดประสงค์ คือการทำลาย สัญลักษณ์ของฝ่ายตรงข้ามให้ได้ซึ่งจะแตกต่างจากเกมลักษณะเดียวกันที่รู้จักกันดี คือ warcraft ตรงที่การบังคับ ยูนิตฮีโร่ จะเป็นการบังคับโดยใช้ คีย์บอร์ด ผสมกับเมาส์ เพื่อให้เป็นเกมแนว แอคชั่น สมจริงมากขึ้น โดยเมื่อเข้าสู่เกมแล้วจะมีสิ่งที่ต้องทำต่างๆดังนี้

- การเลือกทีม
- การเลือกตัวละคร
- สู้กับศัตรูเพื่อสะสมประสบการณ์
- ทำลายสัญลักษณ์ฝ่ายตรงข้าม
- สรุปผล

3.1.1.1 การเลือกทีม

การเลือกทีมจะทำการกำหนดว่าเราจะอยู่ฝ่ายใด จะอยู่กับใครเมื่อทำการเลือกแล้วจะไม่สามารถเปลี่ยนทีมได้จนกว่าจะจบเกม โดยโปรแกรมจะทำการเลือกทีมให้โดยอัตโนมัติเมื่อเราเข้าสู่เกมโดยจะมีการลำดับทีมดังนี้

- ผู้เล่นที่เป็นผู้ให้บริการ จะอยู่ทีม 1
- ผู้เล่นที่เข้ามาใช้บริการคนที่ 1 จะอยู่ทีม 2
- ผู้เล่นที่เข้ามาใช้บริการคนที่ 2 จะอยู่ทีม 1
- ผู้เล่นที่เข้ามาใช้บริการคนที่ 3 จะอยู่ทีม 2

3.1.1.2 การสู้กับศัตรูเพื่อสะสมและประสบการณ์

เมื่อเลือกตัวละครแล้วเวลาผ่านไประยะหนึ่งจะมีลุกน้องออกมาเพื่อให้เราสะสมประสบการณ์และเงินเมื่อเราได้ประสบการมาขึ้นตัวละครเราจะพัฒนา

3.1.1.3 การทำลายสัญลักษณ์ของฝ่ายตรงข้าม

เมื่อทำการต่อสู้ไปเรื่อยๆจนพลังหมดแล้วตาย ผู้เล่นสามารถเกิดใหม่ได้ โดยมีการหน่วงเวลาไว้ตามความสามารถของตัวละคร (ความสามารถสูง จะมีการหน่วงเวลานาน

3.1.1.4 การเกิดใหม่

เมื่อฝ่ายใดเพลี่ยงพล้ำล้มตาย ก็จะถูกฝ่ายตรงข้ามบุกเข้ามาทำลายฐาน และเมื่อใดที่สัญลักษณ์ของฝ่ายตัวเองถูกทำลายก็จะถือว่าพ่ายแพ้ไปในเกมนั้น

3.1.1.5 การสรุปผล

เมื่อเล่นจบเกมก็จะมีการสรุปว่า ใครเป็นผู้แพ้หรือเป็นผู้ชนะ

3.1.2 การออกแบบภาพและกราฟิก

รูปแบบของภาพในเกม มีลักษณะเป็น 3 มิติ โดยจะพยายามใช้แสงเงาและ เอฟเฟกต์ต่างๆที่ Direct X สามารถทำได้ เพื่อให้ภาพที่ออกมานั้นสมจริงที่สุด ซึ่งตัวละครอาวุธ แผนที่ และองค์ประกอบอื่นๆจะใช้โปรแกรม 3ds max 7 ช่วยในการออกแบบ แต่ในส่วนการออกแบบหน้าจอจะใช้โปรแกรม Adobe Photoshop CS เข้าช่วย

3.1.3 การออกแบบเสียงและซาวน์เอฟเฟกต์ประกอบ

ไฟล์เสียงที่ใช้เป็นเอฟเฟกต์ในเกมมี ประเภทเดียวคือไฟล์ WAVE โดยเสียงเอฟเฟกต์จะเสียงประกอบภายเกม เช่น เสียงย่ำเท้า เป็นต้น ส่วนเสียงประกอบจะเป็นทำนองเพลงประกอบขณะเล่นเกมเพื่อเพิ่มอรรถรสในการเล่นเกม โดยโปรแกรมที่นำมาใช้สำหรับการสร้างและตัดต่อไฟล์เสียงและซาวน์เอฟเฟกต์ ได้แก่ Nero Wave Editor และ Adobe Audition

3.2 ขั้นตอนการทำงานจริง

3.2.1 สร้างโมเดลที่ใช้ในเกมทั้งหมด

ส่วนประกอบต่างๆภายในเกมล้วนแต่โมเดลที่สร้างขึ้นทั้งสิ้น ซึ่งโมเดลเหล่านี้ส่วนใหญ่สร้างมาจากโปรแกรม 3ds max 7 ประกอบกับการนำโมเดลที่มีอยู่แล้วมาประยุกต์ใช้ โดยโมเดลภายในเกมจะประกอบด้วย 2 ส่วนหลักด้วยกัน ได้แก่ โมเดลตัวละครในเกม โมเดลแผนที่

3.2.1.1 โมเดลตัวละครในเกม (Character)

ตัวละครภายในเกมเหมือนอย่างที่กล่าวเอาไว้ ซึ่งโมเดลที่จะนำเข้ามาใช้ในเกมนี้จะใช้โมเดลที่สร้างด้วยโปรแกรม 3D Studio Max 7 (ไฟล์ .max) แล้วค่อยส่งออกไฟล์เป็น .x โดยใช้โปรแกรม PandaExport

รูปที่ 3.1 ตัวละคร Tiny

รูปที่ 3.2 ตัวละคร Robot

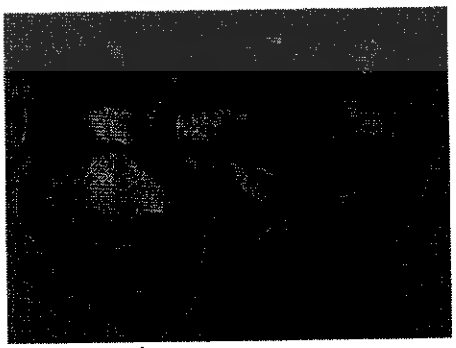
รูปที่ 3.3 ตัวละคร TrollTooth

3.2.1.2 โมเดลแผนที่ (Map)

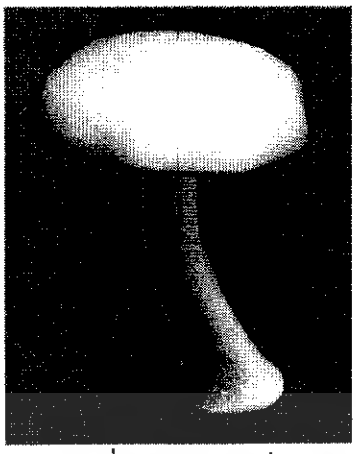
เราจะทำการนำตัวสิ่งของต่างๆเช่น ต้นไม้ กำแพง มาประกอบรวมกันเป็น ฉากๆหนึ่งโดยโมเดลที่เรานำมาใช้ ส่วนหนึ่งจะเป็นการสร้างขึ้นเองอีกส่วนหนึ่งจะนำโมเดลสำเร็จรูปมาใช้ ตัวอย่างโมเดลที่ใช้ทำ ฉากมีดังนี้

รูปที่ 3.4 โมเดลต้นไม้

รูปที่ 3.5 โมเดล ประตู



รูปที่ 3.6โมเดลก้อนหิน

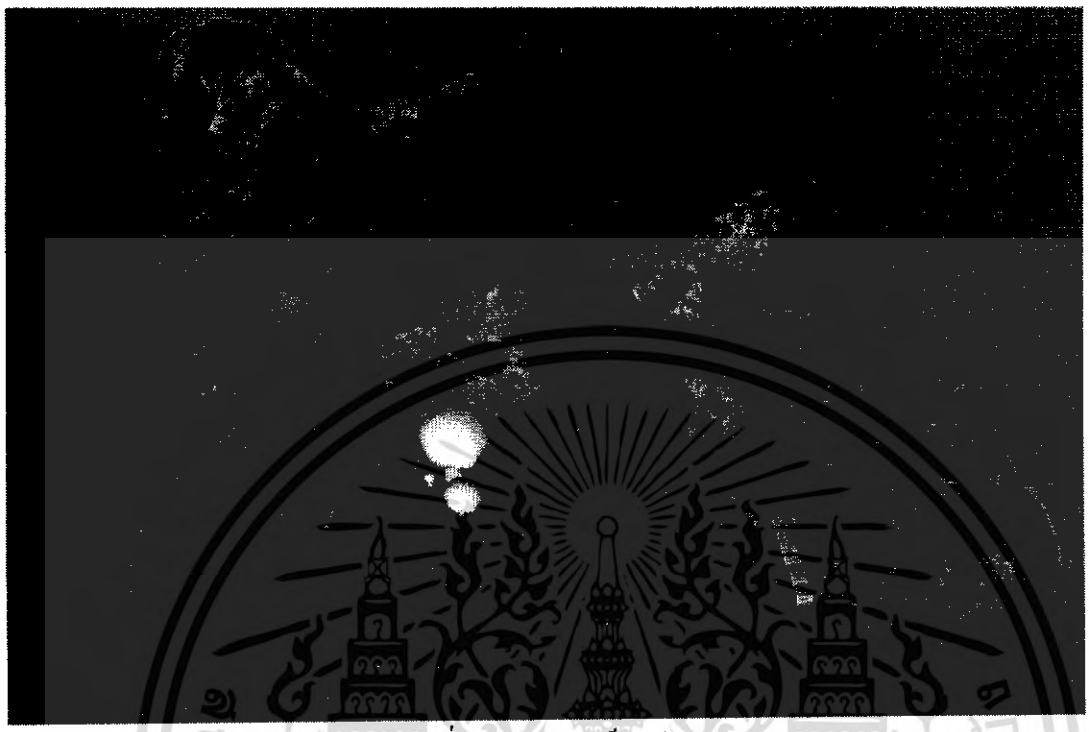


รูปที่ 3.7 โมเดล เห็ด

รูปที่ 3.8 โมเดล ต้นไม้

รูปที่ 3.9 โมเดล ป้อมปราการ

เมื่อนำโมเดลทุกอย่างมารวมกันและใส่ texture จะได้ฉากดังนี้



รูปที่ 3.10 ฉากสำเร็จรูป

3.2.2 หลักการโดยรวมของโปรแกรม

3.2.2.1 หลักการของ Client/Server

เนื่องจากเกมเป็นลักษณะของเกมหลายผู้เล่นจึงต้องมีการติดต่อระหว่างเครื่องคอมพิวเตอร์หลายเครื่องโดยการรับส่งข้อมูลระหว่างกันจะใช้ตัวแปร อเรย์สองมิติมีลักษณะดังนี้

หมายเลขตัวละคร	ค่าพลังชีวิต	ค่าพิกัดแกนX	ค่าพิกัดแกนY	ค่าพิกัดแกนZ	ลักษณะ การกระทำ
0	100	0	0	0	1
1	...	...	...	...	...

มี

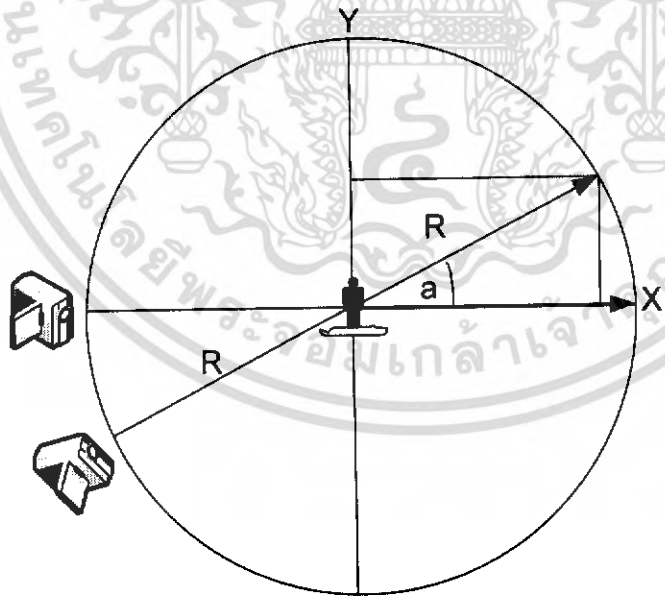
เมื่อเริ่มเกมแล้วจะมีคนที่ตั้งตัวเป็น Server (ผู้ให้บริการ) เมื่อมีการติดต่อจาก Client (ผู้ใช้บริการ) เข้ามาผู้ให้บริการจะทำการบอกกับผู้ใช้บริการที่เข้ามานั้นได้เป็นผู้ใช้บริการหมายเลขใดและจะกำหนดค่าเริ่มต้นต่างๆให้กับผู้ใช้บริการตามตาราง

เมื่อเริ่มเกมแล้วหากตัวละครมีการเคลื่อนไหว หรือ แสดงท่าทางต่างๆ จะมีการกระทำดังนี้

- ในกรณีที่เป็นผู้ให้บริการ เมื่อมีการเลื่อนไหวหรือแสดงท่าทางต่างๆ จะส่งข้อมูลต่างๆ ไปให้กับผู้ใช้บริการทุกๆคนแล้วจึงแสดงท่าทางต่างๆนั้น
- ในกรณีที่เป็นผู้ใช้บริการ เมื่อมีการเคลื่อนไหว หรือ แสดงท่าทางต่างๆจะส่งข้อมูลไปยังผู้ให้บริการเมื่อผู้ให้บริการได้รับข้อมูลแล้วจะทำการตรวจจับดูว่า ข้อมูลของพลังชีวิตที่ส่งมาน้อยกว่า หรือเท่ากับศูนย์หรือเปล่าหากใช่ก็จะเปลี่ยนตำแหน่งตัวละครกลับไปยังจุดเกิดหากไม่ใช่ก็จะ ทำการส่งข้อมูลต่อไปให้กับทุกๆเครื่องรวมถึงเครื่องที่สงมาด้วย เมื่อผู้ใช้บริการได้รับข้อมูลกลับมาแล้ว จึงทำการเคลื่อนไหวตัวละคร

3.2.2.2 หลักการของการเลื่อนที่และมุมกล้อง

เมื่อมีการเลื่อนที่ในแนวแกนต่างๆหากไม่มีการหมุนของตัวละครจะมีการเคลื่อนที่ๆถูกต้อง แต่เมื่อตัวละครมีการหมุนไปจำนวน a องศาแลเคลื่อนทีในแนวแกน X เป็นระยะทาง R จะทำให้ตัวละครเคลื่อนไปในแกนต่างๆดังนี้



รูป 3.11 ตำแหน่งคนและกล้อง

- แกน X จะเคลื่อนที่ไปในระยะ R*cos(a)
- แกน Y จะเคลื่อนที่ไปในระยะ R*sin(a)

ในส่วนของกล้องจะมีตำแหน่งอยู่ข้างหลังคนห่างออกไปในระยะหนึ่ง สูงขึ้นไปกว่าตำแหน่งของตัวละคร และทิศทางของกล้องจะมองมาที่ตัวละครตลอด เมื่อตัวละครหมุนแล้วกล้องจะหมุนตามตัวละครโดยตำแหน่งของกล้องจะคำนวณได้ดังสูตรนี้

- แกน X กล้องจะเคลื่อนที่ไปในระยะ R*cos(-a)
- แกน Y กล้องจะเคลื่อนที่ไปในระยะ R*sin(-a)

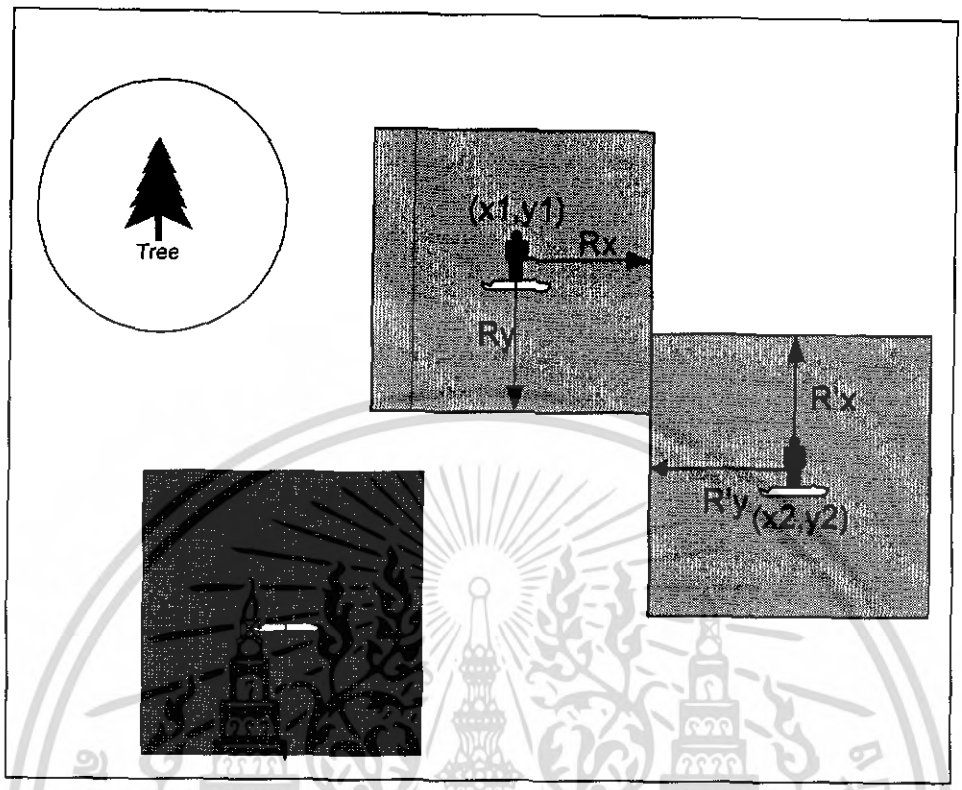
แต่เมื่อมีการหมุนแล้วจะทำให้กล้องมีการหมุนรอบตัวเองด้วยจึงต้องกำหนดค่าการหมุนรอบตัวเองเป็นดังนี้

- ค่าความหมุนรอบตัวเองของแกน X = -R*cos(-a)
- ค่าความหมุนรอบตัวเองของแกน Y = -R*sin(-a)

3.2.2.3 หลักการของตรวจจับการชน

เมื่อตัวละครเดินมาอยู่ในรัศมีของตัวละครอีกตัวหนึ่ง (ความกว้างและยาวของตัวละคร) หรือดินไปอยู่ในรัศมีของสิ่งกีดขวางในฉากจะต้องตรวจจับให้ได้และจะต้องทำให้ไม่สามารถเดินไปยัง ณ จุดๆนั้นได้ โดยสามารถแบ่งการชนออกเป็น 2 แบบ คือการชนกับวัตถุที่มีขนาดเล็ก กับ การชนกับวับวัตถุที่มีขนาดใหญ่

ในการตรวจจับการชนวัตถุที่มีขนาดเล็กจะสมมติว่า วัตถุที่ถูกชนมีรัศมีเป็นสี่เหลี่ยมเหมือนกันหมดทุกวัตถุเพราะจะทำให้การคำนวณ คำนวณได้ง่ายและรวดเร็วและมองเห็นความผิดพลาดได้น้อยเนื่องจากวัดถุมีขนาดเล็กโดยมีหลักการการตวจจับการชนดังนี้



รูปที่ 3.12 การตรวจจับการชนวัตถุขนาดเล็ก

ตัวละครที่ต้องการจะเคลื่อนที่จะตรวจสอบก่อนว่ารอบๆของตัวละครมีผู้เล่นอยู่ในตำแหน่งใดบ้าง และ มีสิ่งกีดขวางอยู่ในตำแหน่งใดบ้างโดยจะคิดเป็นการชนของแกน 2 แกนคือ

- หากมีผู้เล่นอื่นหรือสิ่งกีดขวางมีตำแหน่งในแนวแกน X มากกว่าหรือเท่ากับตำแหน่งในแนวแกน X ของตัวผู้เล่น บวกด้วย รัศมีของเรา บวกด้วยรัศมีของผู้เล่นคนอื่น จะถือว่าเป็นการชนในแนวแกน X หรืออาจ แสดงเป็นสูตรได้ดังนี้

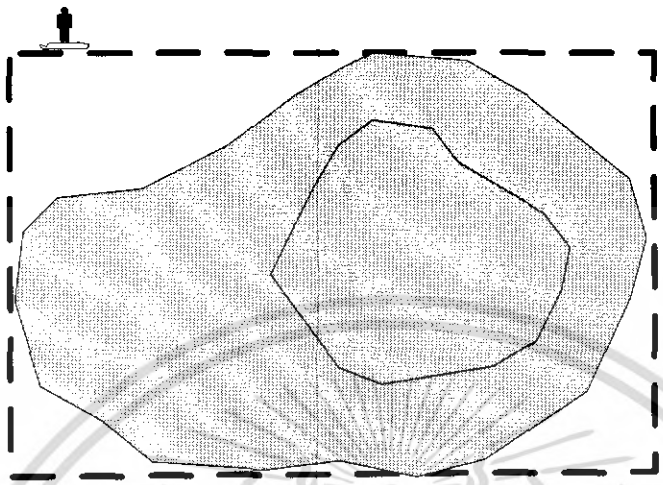
$$X2 >= X1+Rx+Ry$$

- หากมีผู้เล่นอื่นหรือสิ่งกีดขวางมีตำแหน่งในแนวแกนY มากกว่าหรือเท่ากับตำแหน่งในแนวแกน Y ของตัวผู้เล่น บวกด้วย รัศมีของเรา บวกด้วยรัศมีของผู้เล่นคนอื่น จะถือว่าเป็นการชนในแนวแกน Y หรืออาจ แสดงเป็นสูตรได้ดังนี้

$$Y2 >= Y1+R'x+R'y$$

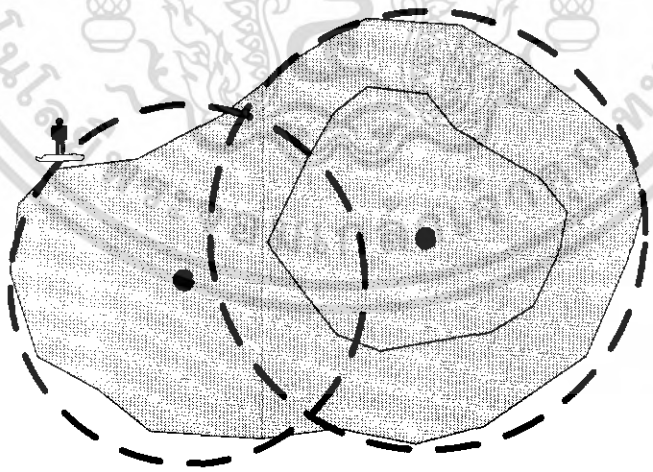
ถ้าหากการชนกันทั้งสองแกนเป็นจริงจะถือว่าวัตถุสองชิ้นนั้นมีการชนกัน

ในการตรวจจับการชนของวัตถุที่มีขนาดใหญ่ในบางกรณีไม่สามารถทำแบบการตรวจจับการชนขนาดเล็กได้เนื่องจากเราสามารถมองเห็นความผิดพลาดในการชนได้ง่ายตัวอย่างเช่นดังรูป



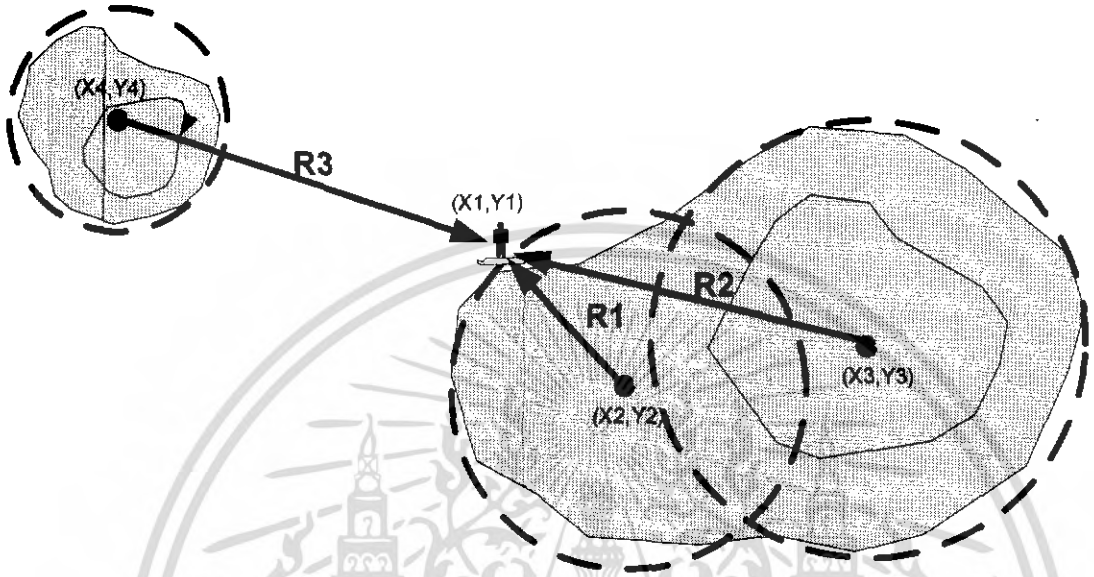
รูปที่ 3.13 รัศมีของการตรวจจับการชนขนาดเล็ก

เมื่อตัวละครเดินมายังจุดดังกล่าวหากเราใช้วิธีการตรวจจับการชนเหมือนกับที่วัตถุมีขนาดเล็กเราจะไม่สามารถเดินเข้าไปยังจุดดังกล่าวได้ (เนื่องจากวัตถุมีขนาดใหญ่เราจึงเห็นส่วนของความผิดพลาดได้มาก ถ้าหากวัตถุมีขนาดเล็กส่วนของความผิดพลาดอาจมองไม่เห็นหรือเห็นได้น้อยมาก) เราจึงต้องเปลี่ยนวิธีตรวจจับการชนให้มีความสมจริงมากยิ่งขึ้นโดยมีวิธีดังนี้



รูปที่ 3.14 รัศมีของการตรวจจับการชนขนาดใหญ่

จากรูปเราจะเห็นได้ว่าเราเปลี่ยนจากการใช้รัศมี สี่เหลี่ยม เป็น วงกลม 2 วงแทนทำให้วัตถุมีความผิดพลาดในการชนน้อยลง โดยมีการคำนวณในการชนดังนี้



รูปที่ 3.15 การตรวจจับการชนวัตถุขนาดใหญ่

ตัวละครที่ต้องการจะเคลื่อนที่จะตรวจสอบก่อนว่ารอบๆของตัวละครมีสิ่งกีดขวางขนาดใหญ่อยู่ที่ตำแหน่งใดบ้างแล้วจะทำการหาระยะห่างระหว่างจุด 2 จุดโดยใช้สูตร

$$R = sqrt((x2-x1)^2 + (y2-y1)^2)$$

เพื่อหาระยะห่างระหว่างตัวละครกับทุกๆจุด เมื่อตัวละครเดินไปแล้วมีระยะห่างจากจุดใดน้อยกว่า หรือเท่ากับรัศมีของวัตถุนั้นๆจะถือว่าเป็นการชนตัวละครจะไม่สามารถเดินไปยังทิศทางนั้นๆ ได้อีก

ตัวละครที่ต้องการจะเคลื่อนที่ก่อนการเคลื่อนที่จะมีการตรวจสอบการชนก่อนว่าชนกับสิ่งกีดขวางหรือตัวละครอื่นๆหรือไม่หากมีการชนจะไม่ส่งข้อมูลไปยัง ผู้ให้บริการในกรณีที่เป็นผู้รับบริการ หรือจะไม่เคลื่อนที่และไม่ส่งต่อไปให้กับผู้รับบริการ

3.2.3 ขั้นตอนการเขียนโปรแกรม

การเขียนโปรแกรมจะแบ่ง คลาส ออกเป็นสอง คลาส คือ คลาส เกม และ ตัวละคร

3.2.3.1 คลาส ตัวละคร

ในส่วนของ คลาส นี้จะทำการเก็บข้อมูลต่างๆของตัวละครว่า ขณะนี้อยู่ตำแหน่งใดของฉาก มีพลังชีวิตเท่าไหร่ ตัวละครที่ใช้มีลักษณะอย่างไร

แอททริบิว

- rootFrame เป็นตัวเก็บรูปแบบของตัวละครและ การเลื่อนไหวทั้งหมดของตัวละคร
- device เก็บเกี่ยวกับ hardware ที่ใช้แสดงผล
- angleX เก็บมุมที่หมุนในแนวแกน X
- angleY เก็บมุมที่หมุนในแนวแกน Y
- angleZ เก็บมุมที่หมุนในแนวแกน Z
- moveX เก็บตำแหน่งที่เคลื่อนที่ในแนวแกน X
- moveY เก็บตำแหน่งที่เคลื่อนที่ในแนวแกน Y
- moveZ เก็บตำแหน่งที่เคลื่อนทีในแนวแกน Z

เมทธอด

- CreateAnimetion เป็นเมทธอดหลักในคลาสนี้ จะทำการโหลดทุกๆอย่างจากไฟล์ มาเก็บไว้ใน แอททริบิว rootFrame และจะไปเรียกใช้ เมทธอด SetupBoneMatrix เพื่อกำหนดกระดูกให้กับ rootFrame
- SetupBoneMatrix เป็นเมทธอดในการเก็บกระดูกทุกๆส่วนให้เข้ากับ ตัวละคร
- DrawFrame เป็นเมทธอดในการ วาดตัวละครทุกๆส่วนในฉาก โดยจะเรียกใช้เมธอด DrawMeshContainer
- DrawMeshContainer เป็นเมธอดในการวาดตัวละครทีละส่วน

3.2.3.2 คลาสเกม

ในคลาสนี้จะเป็นคลาสหลักของเกมทำการ กำหนดค่าของ hardware ในเครื่อง วาดภาพออกทางจอภาพ ควบคุมการรับส่งของข้อมูล และ ดูแลการกระทำต่างๆจากอุปกรณ์ภายนอก

แอททริบิว

- **device** เก็บเกี่ยวกับ hardware ของเครื่อง ที่ใช้แสดงผล
- **anime** เก็บรายระเอียดของตัวละครทุกๆตัว
- **mapMesh** เก็บข้อมูลของแผนที่
- **mapMaterail** เก็บ Materail ที่อยู่ในแผนที่
- **mapTexture** เก็บ texture ที่อยู่ในแผนที่
- **camX** เก็บค่าเริ่มต้นตำแหน่งของกล้องในแนวแกน X
- **camY** เก็บค่าเริ่มต้นตำแหน่งของกล้องในแนวแกน Y
- **camZ** เก็บค่าเริ่มต้นตำแหน่งของกล้องในแนวแกน Z
- **vEye** เก็บเป็น เวกเตอร์ตำแหน่งของกล้องในปัจจุบัน
- **vUp** เก็บเป็นเวกเตอร์การหมุนรอบตัวเองของกล้อง
- **attr** เก็บข้อมูลทุกๆอย่างที่ server ส่งมา และ ข้อมูลที่จะส่งไปให้ Server
- **myChar** เก็บว่าเราเป็นตัวละครที่เท่าไหร่ใน Server
- **numPlayer** เก็บว่ามีผู้เล่นทั้งหมดเท่าไหร่ใน Server
- **canUp** เป็นตัวแปร Boolean ตรวจสอบการชนว่าเราสามารถเดินไปข้างหน้าได้หรือไม่
- **canDown** เป็นตัวแปร Boolean ตรวจสอบการชนว่าเราสามารถเดินถอยหลังได้หรือไม่
- **canLeft** เป็นตัวแปร Boolean ตรวจสอบการชนว่าเราสามารถเดินไปทางซ้ายได้หรือไม่
- **canRight** เป็นตัวแปร Boolean ตรวจสอบการชนว่าเราสามารถเดินไปทางขวาได้หรือไม่
- **serverConnection**เก็บการติดต่อในขณะที่ตัวเองเป็นผู้ให้บริการ
- **clientConnection** เก็บการติดต่อในขณะที่ตัวเองเป็นผู้ใช้บริการ
- **host** เป็นตัวแปร Boolean ตรวจสอบว่า เป็นผู้ให้บริการหรือไม่

- **connected** เป็นตัวแปร Boolean ตรวจสอบว่า ขณะนี้มีการติดต่อกับ ผู้ให้บริการหรือไม่

เมทธอด

- **Draw2DText** เป็นเมทธอดในการวาดภาพ 2 มิติ(พลังชีวิต)
- **DrawMap** เป็นเมทธอดในการวาดแผนที่ที่เราได้โหลดไว้แล้ว
- **LoadMesh** เป็นเมทธอดโหลดค่าของวัตถุที่ไม่ได้เคลื่อนไหว (แผนที่)มาเก็บไว้ใน แอททริบิว
- **InitializeClient** เป็นเมทธอดในการกำหนดค่าเริ่มต้นเกี่ยวกับระบบเครือข่ายให้กับผู้เล่นหากผู้เล่นเป็นผู้ขอใช้บริการ
- **InitializeServer** เป็นเมธอดในการกำหนดค่าเริ่มต้นเกี่ยวกับระบบเครือข่ายให้กับผู้เล่นหากผู้เล่นเป็นผู้ให้บริการ
- **InitializeGraphic** เป็นเมธอดในการกำหนดค่าการแสดงผลเริ่มต้นให้กับผู้เล่น
- **IsCrash** เป็นเมธอดในการตรวจจับการชนกันของ วัตถุ
- **OnConnectComplete** เป็น เมธอดของผู้ใช้บริการ เป็นการการกำหนดว่า เมื่อติดต่อผู้ให้บริการได้แล้วจะทำอย่างไร
- **OnFindHost** เป็นเมธอดของผู้ใช้บริการ ในการหาผู้ให้บริการ
- **OnDataReceive** เป็นเมธอดที่คอยตรวจสอบ เมื่อมีข้อมูลจากเครือข่ายเข้ามาจะแยกประเภทของข้อมูลและจะทำสิ่งใดกับข้อมูลนั้น
- **OnPlayerCreate** เป็นเมธอดของผู้ให้บริการ ในการกำหนดว่าเมื่อมีผู้ใช้บริการเข้ามาแล้วจะให้ทำอย่างไร
- **OnFrameUpdate** เป็นการกำหนดว่า เมื่อมีการเปลี่ยนFrame ไปแล้วให้ทำอย่างไรบ้าง
- **OnKeyUp** เป็นเมธอดในการควบคุมการเคลื่อนไหวของตัวละคร และส่งข้อมูลไปยังผู้ให้บริการว่าเราเคลื่อนไหว
- **OnKeyDown** เป็นเมธอดในการควบคุมการเคลื่อนไหวของตัวละคร และส่งข้อมูลไปยังผู้ให้บริการว่าเราเคลื่อนไหว
- **OnKeyLeft** เป็นเมธอดในการควบคุมการเคลื่อนไหวของตัวละคร และส่งข้อมูลไปยังผู้ให้บริการว่าเราเคลื่อนไหว

- **OnKeyRight** เป็นเมธอดในการควบคุมการเคลื่อนไหวของตัวละคร และส่งข้อมูลไปยังผู้ให้บริการว่าเราเคลื่อนไหว
- **OnMouseUp** เป็นเมธอดในการควบคุมท่าทางของตัวละครเมื่อปล่อยการคลิกเมาส์
- **OnMouseDown** เป็นเมธอดในการควบคุมท่าทางของตัวละครเมื่อมีการคลิกเมาส์
- **OnMouseMove** เป็นเมธอดในการควบคุมมุมกล้องของเกม เมื่อมีการเลื่อนไหวเมาส์
- **OnPaint** เป็นเมธอดในการวาดทุกๆอย่างในเกม

3.2.3.3 การวาดภาพ 3 มิติ

การวาดภาพ 3 มิติในที่นี้จะเป็นการนำวัตถุ 3 มิติจากภายนอกเข้ามาวาดในฉากของเราซึ่งสามารถแบ่งออกได้เป็น 2 ประเภทคือ วัตถุที่ไม่มีการเคลื่อนไหว เช่น ฉากต่างๆในเกม และวัตถุที่มีการเคลื่อนไหว เช่นตัวละครต่างๆ

3.2.3.3.1 การวาดภาพ 3 มิติที่ไม่มีการเคลื่อนไหว

- ขั้นแรกจะประกาศตัวแปร Mesh ,Texture[] ,Material[]เอาไว้ก่อน

```
private Mesh mapMesh = null;
private Material[] mapMaterials = null;
private Texture[] mapTextures = null;
```

- ต่อมาจะทำการโหลด Mesh ,Texture[] ,Material[] โดยฟังก์ชั่น LoadMesh

```
public bool InitializeGraphics()
{
      ...
      mapMesh = LoadMesh(dDevice, @"..\..\map.x",
                  ref mapMaterials, ref mapTextures);
      ...
}
```

- ขั้นตอนต่อมาคือการเขียนฟังก์ชั่นในการวาด Mesh ของเราดังนี้

```
private void DrawRoad(float x, float y, float z)
{
        dDevice.Transform.World = Matrix.Translation(x, y, z);
        for (int i = 0; i < roadMaterials.Length; i++)
        {
                dDevice.Material = roadMaterials[i];
                dDevice.SetTexture(0, roadTextures[i]);
                roadMesh.DrawSubset(i);
        }
}
```

- ขั้นตอนสุดท้ายคือการวาดวัตถุของเราในฉากจริง

```
protected override void OnPaint(System.Windows.Forms.PaintEventArgs e)
{
        ...
        DrawRoad(0.0f, 0.0f, 0);
        ...
}
```

3.2.3.3.2 ขั้นตอนการวาดภาพ 3 มิติที่มีการเคลื่อนไหว

- ขั้นแรกทำการสร้างคลาสตัวละครก่อนโดยกำหนดค่าแอททริบิวต่างๆตามที่เราต้องการ

```
public class Anime : System.Windows.Forms.Form
{
    ...
}
```

- ต่อมาทำการกำหนดฟังก์ชั่นต่างๆในคลาสนี้ โดยที่หากเราต้องการวาดวัตถุของเราในตำแหน่งใดในฉากสามารถกำหนดได้ดังนี้

```
private void DrawMeshContainer(MeshContainerDerived mesh, FrameDerived frame)
{
    ...
Matrix tempMatrix = offsetMatrices[matrixIndex]
*frameMatrices[matrixIndex].CombinedTransformationMatrix
*Matrix.Scaling(0.08f,0.08f,0.08f)
*Matrix.RotationYawPitchRoll(angleY,angleX,angleZ)
*Matrix.Translation(moveX,moveY,moveZ);
    ...
}
```

- ต่อมาทำการประกาศแอททริบิวในการเก็บข้อมูลไว้ในคลาสหลัก

```
private Anime[] anime = null;
private AnimationRootFrame[] tempRootFrame;
```

- ต่อมาทำการกำหนดค่าเริ่มต้นให้กับแอททริบิว

```
public bool InitializeGraphics()
{
        ...
        anime = new Anime[4];
        anime[0] = new Anime(dDevice,@"..\..\tiny_4anim.x",presentParams);
        anime[1] = new Anime(dDevice,@"..\..\tiny_4anim1.x",presentParams);
        anime[2] = new Anime(dDevice,@"..\..\tiny_4anim.x",presentParams);
        anime[3] = new Anime(dDevice,@"..\..\tiny_4anim.x",presentParams);

        tempRootFrame = new AnimationRootFrame[4];
        tempRootFrame[0] = anime[0].rootFrame;
        tempRootFrame[1] = anime[1].rootFrame;
        tempRootFrame[2] = anime[2].rootFrame;
        tempRootFrame[3] = anime[3].rootFrame;
        ...
}
```

- ขั้นตอนสุดท้ายเป็นการนำวัตถุที่โหลดมาแล้วมาวาดในฉาก

```
protected override void OnPaint(System.Windows.Forms.PaintEventArgs e)
{
        ...
        anime[0].DrawFrame((FrameDerived)tempRootFrame[0].FrameHierarchy);
        ...
}
```

- ใน Code ตัวอย่างนี้ จะแสดงท่าทางได้เพียงท่าทางเดียวใน Track ที่ 0 เราสามารถเปลี่ยนท่าทางของตัวละครได้โดยการเปลี่ยน Track ที่ 0 ในตัวอย่างนี้จะเปลี่ยนจาก Track ที่ 0 เป็น Track ที่2

```
tempTinyRootFrame.AnimationController.SetTrackAnimationSet(0
        ,tempTinyRootFrame.AnimationController.GetTrackAnimationSet(2));
```

3.2.3.4 การจัดการระบบ ผู้ให้บริการ

ในหน้าเริ่มต้นเกมถ้าหากผู้เล่นเลือก Create Game ผู้เล่นคนนั้นจะเป็นผู้ให้บริการซึ่งจะทำการเรียกใช้ฟังก์ชั่น IntitializeServer() ในการจัดการค่าเริ่มต้นให้กับผู้ให้บริการเมื่อฟังก์ชั่นนี้ถูกเรียกใช้จะมีการตรวจจับ เหตุการณ์ที่เกิดขึ้นคือ เมื่อมีผู้ใช้บริการส่งข้อมูลเข้ามา ,เมื่อผู้มีผู้ใช้บริการเข้ามาขอใช้บริการ ,เมื่อมีผู้ใช้บริการออกจากระบบไป (การทำงานจะกล่าวในส่วนต่อไป) สามารถแสดงได้ดังนี้

```
public void InitializeServer()
{
        ServerConnection = new Server();
        // Hook the events we want to listen for
        ServerConnection.Receive += new
                                ReceiveEventHandler(OnDataReceive);

        ServerConnection.PlayerCreated += new
                                PlayerCreatedEventHandler(OnPlayerCreated);

        ServerConnection.PlayerDestroyed += new
                                PlayerDestroyedEventHandler(OnPlayerDestroyed);
        ...
}
```

การส่งข้อมูลของผู้ให้บริการไปยังผู้ใช้บริการทุกๆคนจะสามารถส่งได้ด้วยคำสั่งดังนี้

```
        ...
        NetworkPacket packet = new NetworkPacket();
        packet.Write(numPlayer);
        packet.Write(attr);
        ServerConnection.SendTo((int)PlayerID.AllPlayers,packet,0,SendFlags.Gua
ranteed | SendFlags.NoLoopback);
        ...
```

3.2.3.5 การจัดการระบบ ผู้ใช้บริการ

ในหน้าเริ่มต้นเกมถ้าหากผู้เล่นเลือก Join Game ผู้เล่นคนนั้นจะเป็นผู้ใช้บริการซึ่งจะทำการเรียกใช้ฟังก์ชั่น IntitializeClient() ในการจัดการค่าเริ่มต้นให้กับผู้ให้บริการเมื่อฟังก์ชั่นนี้ถูกเรียกใช้จะมีการตรวจจับ เหตุการณ์ที่เกิดขึ้นคือ เมื่อมีการเชื่อมต่อสำเร็จ ,เมื่อมีการหาผู้ให้บริการ ,เมื่อได้รับข้อมูลจากผู้ให้บิรการ, เมื่อมีการยกเลิกใช้บริการ (การทำงานจะกล่าวในส่วนต่อไป) สามารถแสดงได้ดังนี้

```
public void InitializeClient()
{
        // Create our client object
        ClientConnection = new Client();
        // Hook the events we want to listen for
        ClientConnection.ConnectComplete += new
                ConnectCompleteEventHandler(OnConnectComplete);

        ClientConnection.FindHostResponse += new
                FindHostResponseEventHandler(OnFindHost);

        ClientConnection.Receive += new
                ReceiveEventHandler(OnDataReceive);

        ClientConnection.SessionTerminated += new
                SessionTerminatedEventHandler(OnSessionTerminate);
        ...
}
```

การส่งข้อมูลของผู้ใช้บริการไปยังผู้ให้บริการจะสามารถส่งได้ด้วยคำสั่งดังนี้

```
        ...
        NetworkPacket packet = new NetworkPacket();
        packet.Write(attr);
        ClientConnection.Send(packet,0,SendFlags.Guaranteed);
        ...
```

3.2.3.6 การจัดการอีเวนต์ ต่างๆ

ในส่วนนี้จะกล่าวถึงการที่มีการกระทำต่างๆเข้ามาจากทั้งภายนอกและภายในเกมทำให้ตัวละครมีการเปลี่ยนแปลงไปในรูปแบบต่างๆเช่น การเดิน การหมุนรอบตัวเอง ซึ่งมีฟังก์ชั่นต่างๆ ที่เกี่ยวข้องดังนี้

- OnFrameUpdate เป็นการจัดการของตัวละครเมื่อหน้าจอมีการวาดใหม่ทุกๆครั้ง ฟังก์ชั่นนี้จะอยู่ในฟังก์ชั่น OnPaint ซึ่งในฟังก์ชั่น OnFrameUpdate นี้จะสามารถแบ่งได้เป็นสามส่วนคือ การจัดการของมุมกล้อง, การหน่วงเวลาในการตาย และการดูแลเรื่องการโจมตีของป้อมในกรณีของผู้ให้บริการ
  - ➢ ส่วนการจัดการของมุมกล้อง ในส่วนแรกจะเป็นการกำหนดการหมุนรอบตัวเองของกล้อง (vUp) เพื่อให้สอดคล้องกับการเปลี่ยนตำแหน่งของกล้อง (vEye)

```
private void OnFrameUpdate()
{
        ...
        vUp.X   = -(float)Math.Sin(-anime[myChar].angleZ)*camY;
        vUp.Y   = -(float)Math.Cos(-anime[myChar].angleZ)*camY;

        vEye = new Vector3((float)Math.Sin(-anime[myChar].angleZ)
                           *camY+attr[myChar,2],
                           (float)Math.Cos(-anime[myChar].angleZ)
                           *camY+attr[myChar,3],
                           camZ+attr[myChar,4]);

        dDevice.Transform.View = Matrix.LookAtLH(vEye,
                                  new Vector3(
                                 attr[myChar,2],
                                 attr[myChar,3],
                                 attr[myChar,4]+50),
                                 vUp);

        ...
}
```

- *ส่วนการหน่วงเวลาการตาย* เมื่อได้รับชุดข้อมูลจากผู้ให้บริการว่าเรามีพลังชีวิตน้อยกว่า ศูนย์ จะทำการบันทึกเวลาที่ได้รับข้อมูลนั้น จากนั้นในฟังก์ชั่น OnFrameUpdate ตรวจสอบเรื่อยๆในทุกเฟรมว่าถึงเวลาที่กำหนดหรือยังเมื่อถึงเวลาที่กำหนดแล้วจะทำการส่งตัวละครลงมาข้างล่างตามพิกัดที่กำหนดในแต่ละตัวละคร และทำการเพิ่มพลังชีวิตให้เต็ม สุดท้ายจะทำการบอกไปยังผู้ให้บริการว่า ตัวละครได้เกิดแล้ว

```
private void OnFrameUpdate()
{
  ...
if(!firstTimeReceiveDeath)
{
  Console.WriteLine(DXUtil.Timer(DirectXTimer.GetApplicationTime) - times);
  if(DXUtil.Timer(DirectXTimer.GetApplicationTime) - times >= 10.0f)
  {
        NetworkPacket packet = new NetworkPacket();
        if(myChar == 0)//host
        {
              attr[myChar,2] = 404.0f;
              attr[myChar,3] = 468.0f;
              attr[myChar,4] = 0.0f;
              attr[myChar,1] = 100 + attr[0,6]*20;
              anime[0].moveX = 404.0f;
              anime[0].moveY = 468.0f;
              anime[0].moveZ = 0.0f;
        }

        else if(attr[myChar,0] == 1)
        {
              attr[myChar,2] = -414.0f;
              attr[myChar,3] = -366.0f;
              attr[myChar,4] = 0.0f;
              attr[myChar,1] = 100 + attr[myChar,6]*20;

        }
        //do same thing to all player
        ...

        //send packet to tell my position
        packet.Write(attr);
        if(host)
              ServerConnection.SendTo((int)PlayerID.AllPlayers,
                    packet,0,SendFlags.Guaranteed | SendFlags.NoLoopback);
        else
              ClientConnection.Send(packet,0,SendFlags.Guaranteed);
              firstTimeReceiveDeath = true;
  }
}
```

- การดูแลเรื่องการโจมตีของป้อมในกรณีของผู้ให้บริการจะทำการตรวจสอบทุกๆ 1 วินาทีว่ามีศัตรูเข้ามาในรัศมีหรือไม่(ใช้วิธีเดียวกับการตรวจจับการชนแบบวัตถุขนาดเล็ก)ถ้าหากมีผู้ให้บริการจะทำการบอกไปยัง ผู้รับบริการทุกๆตัวว่า มีตัวละครที่มีพลังชีวิตลดลงแล้ว และถ้าหากตายจะทำการส่งตัวละครนั้นกลับไปยังจุดเกิด และบอกว่าตัวละครนั้นมีพลังชีวิตเท่ากับ ศูนย์

```
// get application tiime (time always go on)
serverTimes = DXUtil.Timer(DirectXTimer.GetApplicationTime);
// if time pass to 1 sec
if(Math.Round(serverTimes,0)> tempTimes)
{
        //get rounded time
        tempTimes = (float)Math.Round(serverTimes,0);
        //detect crash in the area
        temp = IsCrash(479,479,77,-1);
        //if crash and not my team
        if((temp >=0)&&(attr[temp,7] !=0))
        {
        //reduce hp and check what is playernumber then send him to born place
                attr[temp,1] -= 40;
                if(attr[temp,1] <= 0)
                {
                        if(temp == 1)
                        {
                                attr[temp,2] = -414.0f;
                                attr[temp,3] = -366.0f;
                                attr[temp,4] = 80.0f;
                                attr[temp,1] = 0f;
                        }

                        if(temp == 3)
                        {
                                attr[temp,2] = -362.0f;
                                attr[temp,3] = -414.0f;
                                attr[temp,4] = 80.0f;
                                attr[temp,1] = 0f;
                        }
                }

                NetworkPacket packet = new NetworkPacket();
                packet.Write(attr);
                //tell to all people to know who is attacked
                ServerConnection.SendTo((int)PlayerID.AllPlayers,packet,0
                                ,SendFlags.Guaranteed | SendFlags.NoLoopback);
  }
//do like this in another team
}
```

- OnKeyDown เป็นการตรวจจับการรับข้อมูลทาง Keyboard หากในที่นี่จะตรวจจับเฉพาะการกดปุ่ม a, s, d, w, Enter และ Esc ในฟังก์ชั่นนี้จะทำการตรวจสอบดูว่าในขณะนี้อยู่ในหน้าจอใด (State) และเมื่อกดปุ่มแล้วจะให้ทำอะไร
  - การทำงานในขณะที่อยู่หน้าจอเริ่มต้น หากกดลูกศรขึ้นหรือลงจะเป็นการเลื่อนตัวเลือก กด Enter เพื่อเลือกตัวเลือกนั้นๆ กด Esc เพื่อออกจากโปรแกรมเกม

```
protected override void OnKeyDown(System.Windows.Forms.KeyEventArgs e)
{
        ...
        if (e.KeyCode == Keys.Escape)
        {
                // Close the form and return
                this.Close();
                return;
        }
        if (e.KeyCode == Keys.Up)
        {
                if( selectPosition.Y > 350)
                {
                        selectPosition.Y -= 50;
                        substate -= 1;
                }
                else
                {
                        selectPosition.Y = 450;
                        substate = 3;
                }
                chooseSound.Play(0, BufferPlayFlags.Default);
        }
        if (e.KeyCode == Keys.Down)
        {
                if( selectPosition.Y < 450)
                {
                        selectPosition.Y += 50;
                        substate += 1;
                }
                else
                {
                        selectPosition.Y = 350;
                        substate = 1;
                }
                chooseSound.Play(0, BufferPlayFlags.Default);
        }
        if (e.KeyCode == Keys.Enter)
        {
                if(substate == 1 )
                {
                        //stop old sound
                        bgSound.Stop();
                        //start new sound
                        backgroundSound.Play(0,BufferPlayFlags.Looping);
                        //go to server
                        state = 0;
                        host = false;
                        this.InitializeServer();
                }
                else if(substate == 2)
                {
                        bgSound.Stop();
                        backgroundSound.Play(0,BufferPlayFlags.Looping);
                        state = 0;
                        host = false;
                        this.InitializeClient();
                }
                else if(substate == 3)
                {
```

```
                    bgSound.Stop();
                    craditSound.Play(0,BufferPlayFlags.Looping);
                    state = 2;
                }
        }
        ...
}
```

- การทำงานในขณะอยู่หน้าจอเล่นเกม เมื่อกดปุ่ม w จะเป็นการเปลี่ยนให้เป็นสถานะ การเดินขึ้น เมื่อกดปุ่ม s จะเป็นการเปลี่ยนให้เป็นสถานะ การเดินลง เมื่อกดปุ่ม a จะเป็นการเปลี่ยนให้เป็นสถานะ การเดินซ้าย เมื่อกดปุ่ม d จะเป็นการเปลี่ยนให้เป็นสถานะ การเดินขวา เมื่อกดปุ่ม Esc จะเป็นการออกจากโปรแกรมเกม

```
protected override void OnKeyDown(System.Windows.Forms.KeyEventArgs e)
{
        ...
        if (e.KeyCode == Keys.Escape)
        {
                // Close the form and return
                this.Close();
                return;
        }
        if ((e.KeyCode == Keys.W) && firstTimeReceiveDeath)
        {
                moveUp = true;

        }
        if ((e.KeyCode == Keys.S) && firstTimeReceiveDeath)
        {
                moveDown = true;
        }
        if ((e.KeyCode == Keys.A) && firstTimeReceiveDeath)
        {
                moveLeft = true;
        }
        if ((e.KeyCode == Keys.D) && firstTimeReceiveDeath)
        {
        moveRight = true;
        }


        // send data to another people when change position
        NetworkPacket packet = new NetworkPacket();
        if(host)
        {
                packet.Write(attr);
                ServerConnection.SendTo((int)PlayerID.AllPlayers,packet,0
                                ,SendFlags.Guaranteed | SendFlags.NoLoopback);
        }
        else
        {
                packet.Write(attr);
                ClientConnection.Send(packet, 0, SendFlags.Guaranteed);
        }
        ...
}
```

➢ การทำงานในขณะอยู่ในหน้าจอ Credit เมื่อกดปุ่ม Esc จะกลับมายังหน้าจอเริ่มเกม

```
protected override void OnKeyDown(System.Windows.Forms.KeyEventArgs e)
{
        ...
        if (e.KeyCode == Keys.Escape)
        {
                craditSound.Stop();
                bgSound.Play(0,BufferPlayFlags.Looping);
                state =1;
        }
        ...
}
```

- OnKeyUp จะทำงานเหมือนกับ OnKeyDown แต่จะต่างกันตรงที่จะตรวจจับเหตุการณ์ ในเฉพาะหน้าจอเล่นเกมเท่านั้น
- OnMouseDown จะทำงานเมื่อมีการกดเม้า ในที่นี้จะตรวจจับเฉพาะการกดเม้าข้างซ้ายเท่านั้น ฟังก์ชั่นนี้จะทำการตรวจดูว่าเมื่อมีการกดเม้าซ้าย มีศัตรูอยู่ในบริเวณที่กำหนดหรือไม่ถ้าหากมีจะทำการลดพลังชีวิตของศัตรู ถ้าหากและถ้าศัตรูตายจะทำการเพิ่มพลังชีวิตให้กับตัวละครที่กดคลิกซ้าย ในกรณีที่เป็นผู้ให้บริการจะทำการส่งตัวละครไปยังจุดเกิด สุดท้ายจะทำการเปลี่ยนท่าทางของตัวละครเป็นการโจมตี

```
protected override void OnMouseDown(MouseEventArgs e)
{
  NetworkPacket packet = new NetworkPacket();
  int temp = -1;
  if (e.Button == MouseButtons.Left)
  {
        if(host)
        {
                temp = IsCrash(attr[0,2],attr[0,3],10.0f,true);
                if((temp >= 0)&&(attr[0,7] != attr[temp,7]))
                {
                        attackSound.Play(0,BufferPlayFlags.Default);
                        attr[temp,1] -= 10;
                        if (attr[4,1] <= 0)
                        {
                                end = true;
                                // you lose
                        }
                        if (attr[5,1] <= 0)
                        {
                                // you win
                                end = true;
                        }
                        if(attr[temp,1] <= 0)
                        {
                                if(attr[temp,0] == 1)
                                {
                                        attr[temp,2] = 442.0f;
                                        attr[temp,3] = 430.0f;
                                        attr[temp,1] = 0.0f;
                                        attr[temp,4] = 80.0f;
                                }
                                // do same thing to all player
                                ...

                                // can kill other player u level up
                                attr[0,6] +=1;
                        }
```

```
                }
                attr[0,5] = 2;
                packet.Write(attr);

                ServerConnection.SendTo((int)PlayerID.AllPlayers,packet,0
                                ,SendFlags.Guaranteed | SendFlags.NoLoopback);

                tempRootFrame[myChar].AnimationController.SetTrackAnimationSet(0
                ,tempRootFrame[myChar].AnimationController.GetTrackAnimationSet(0));


        }
        else //not host
        {
                temp = IsCrash(attr[myChar,2],attr[myChar,3],10.0f);
                if((temp >= 0)&&(attr[myChar,7] != attr[temp,7]))
                {
                        attackSound.Play(0,BufferPlayFlags.Default);
                        attr[temp,1] -= 10;
                        if(attr[temp,1] <= 0)
                                attr[myChar,6] += 1;
                }
                attr[myChar,5] = 2;
                packet.Write(attr);
                ClientConnection.Send(packet, 0, SendFlags.Guaranteed);
                tempTinyRootFrame.AnimationController.SetTrackAnimationSet(0
                ,tempTinyRootFrame.AnimationController.GetTrackAnimationSet(0));
        }

  }
}
```

- OnMouseUp ในฟังก์ชั่นนี้จะทำงานคล้ายกับ OnMouseDown แต่ทำงานเพียงแค่เปลี่ยนท่าทางตัวละครให้กลับเป็นปกติ แล้วส่งไปบอกกับผู้เล่นทุกๆคน

- OnMouseMove ในฟังก์ชั่นนี้จะทำการเพิ่มองศาในการหมุนมุมกล้องและตัวละครถ้าหากเม้าขยับไปทางขวาและจะทำการลบองศาในการหมุนมุมกล้องและตัวละครหากเม้าขยับไปทางซ้าย โดยจะมีตัวแปรเก็บตำแหน่งเดิมไว้ก่อนเมื่อมีการขยับ เม้า จะตรวจสอบว่าตำแหน่งในแกน x มีค่ามากกว่า หรือน้อยกว่า ในตัวแปรที่กำหนดไว้

```
protected override void OnMouseMove(MouseEventArgs e)
{
        if(e.X > beforeMoveX)
        {
                anime[myChar].angleZ += (float)5/180* (float)Math.PI;

        }
        else
        {
                anime[myChar].angleZ -= (float)3/180* (float)Math.PI;
        }
        beforeMoveX = e.X;
}
```

- OnDataReceive ในส่วนของฟังก์ชั่นนี้จะคอยตรวจจับเหตุการณ์ว่ามีชุดของข้อมูลมาจากทางเครือข่ายหรือไม่ โดยจะสามารถแยกเป็นสองส่วน คือส่วนของผู้ให้บริการและส่วนของผู้ใช้บริการ
  - ➢ ในส่วนของผู้ให้บริการเมื่อมีข้อมูลจากเครือข่ายเข้ามาจะทำการรับข้อมูลนั้นแล้วทำการตรวจสอบดูว่ามีพลังชีวิตของตัวละครใดมีค่าน้อยกว่า ศูนย์ หรือไม่ถ้าหากมีจะถือว่าตัวละครนั้นตายและจะทำการส่งตัวละครตัวนั้นไปยังจุดเกิดและส่งข้อมูลนั้นไปให้กับทุกๆคน แต่ถ้าหากข้อมูลที่ได้รับมาเป็นผู้ให้บริการตาย จะทำการกำหนดค่าให้กับตัวแปรเพื่อบอกว่า ตัวเองตายแล้วเพื่อที่จะได้ไปหน่วงเวลาในฟังก์ชั่น *OnFrameUpdate()* เพื่อรอการเกิดใหม่ส่วนต่อมาจะทำการตรวจสอบดูป้อมปราการว่า มีพลังชีวิตน้อยกว่า ศูนย์หรือไม่ถ้าหากมีจะแสดงตัวหนังสือว่าแพ้หรือชนะตามแต่ว่าป้อมปราการของฝ่ายใดถูกทำลาย เมื่อผู้ให้บริการได้รับขอมูลจากผู้ใช้บริการข้อมูลทุกๆอย่างจะถูกส่งต่อไปให้กับผู้ใช้บริการ และในขั้นตอนสุดท้ายจะทำการเปลี่ยนตำแหน่งหรือท่าทางของตัวละครตามที่ได้รับข้อมูลมา

```
private void OnDataReceive(object sender, ReceiveEventArgs e)
{
  NetworkPacket packet = new NetworkPacket();
  if(host)
  {
    attr = (float[,])e.Message.ReceiveData.Read(typeof(float),6,8);
    attr[0,0] = numPlayer;
    for(int i = 0;i<numPlayer;i++)
    {
        if(attr[i,1] <=0)
        {

                //host die
                if(i == 0)
                {
                        if((attr[myChar,1]<= 0) && (firstTimeReceiveDeath))
                        {
                                times = DXUtil.Timer(DirectXTimer.GetApplicationTime);
                                firstTimeReceiveDeath = false;

                                attr[i,2] = 404.0f;
                                attr[i,3] = 468.0f;
                                attr[i,4] = 80.0f;
                                attr[i,1] = 0f;
                                anime[0].moveX = 404.0f;
                                anime[0].moveY = 468.0f;
                                anime[0].moveZ = 80.0f;
                                packet.Write(attr);

                        ServerConnection.SendTo((int)PlayerID.AllPlayers,packet
                                ,0,SendFlags.Guaranteed | SendFlags.NoLoopback);
                        }
                }
                //other people die
                else if(attr[i,0] == 1)
                {
                        //send to born place
                }
                //do same thing to all player
                ...
```

```
    .   }
    }
    // if tower is broken
    if (attr[4,1] <= 0)
    {
        headerFont.DrawText(null, "YOU LOSE",
                new Rectangle(315, 380+m, this.Width , this.Height ),
                DrawTextFormat.NoClip | DrawTextFormat.ExpandTabs |
                DrawTextFormat.WordBreak , Color.WhiteSmoke);
    }
    if (attr[5,1] <= 0)
    {
        headerFont.DrawText(null, "YOU WIN"
                , new Rectangle(315, 380+m, this.Width , this.Height ),
                DrawTextFormat.NoClip | DrawTextFormat.ExpandTabs |
                DrawTextFormat.WordBreak , Color.WhiteSmoke);
    }

    // send packet to all player
    packet.Write(attr);
    ServerConnection.SendTo((int)PlayerID.AllPlayers,packet
                ,0,SendFlags.Guaranteed | SendFlags.NoLoopback);

    //change everything that u received
    for(int i=1;i<=numPlayer;i++)
    {
        //change position
        anime[i].moveX = attr[i,2];
        anime[i].moveY = attr[i,3];
        anime[i].moveZ = attr[i,4];

        //change animation
        if(attr[i,5] == 0)
                tempRootFrame[i].AnimationController.SetTrackAnimationSet(0
                        ,tempRootFrame[i].AnimationController.GetTrackAnimationSet(3));
        else if(attr[i,5] ==1)
                tempRootFrame[i].AnimationController.SetTrackAnimationSet(0
                        ,tempRootFrame[i].AnimationController.GetTrackAnimationSet(1));
        else if(attr[i,5] ==2)
                tempRootFrame[i].AnimationController.SetTrackAnimationSet(0
                        ,tempRootFrame[i].AnimationController.GetTrackAnimationSet(0));
    }
  }
  ...
}
```

- ในส่วนของผู้ใช้บริการจะตรวจสอบดูก่อนว่าเป็นการรับข้อมูลครั้งแรกหรือไม่ (ติดต่อผู้ให้บริการได้สำเร็จ) ถ้าใช่จะทำการรับข้อมูลมาด้วยว่าเราเป็นผู้เล่นลำดับที่เท่าไหร่ในผู้ให้บริการ ถ้าไม่ใช่จะตรวจสอบดูว่าข้อมูลที่ได้รับมา ป้อมปราการมีพลังน้อยกว่าหรือเท่ากับ 0 หรือไม่ถ้าใช่จะทำการแสดงตัวอักษรว่าแพ้หรือชนะ ขึ้นอยู่กับฝ่ายที่ผู้เล่นอยู่ต่อมาจะทำการตรวจสอบดูว่า พลังชีวิตของตัวละครมีค่าน้อยกว่าหรือเท่ากับ 0 หรือไม่ ถ้าใช่จะทำการกำหนดค่าให้กับตัวแปรเพื่อบอกว่า ตัวเองตายแล้วเพื่อที่จะได้ไปหน่วงเวลาในฟังก์ชั่น OnFrameUpdate() เพื่อรอการเกิดใหม่ สุดท้ายจะทำการเปลี่ยนตำแหน่งตัวละครและท่าทางตามข้อมูลที่ได้รับมา

```
private void OnDataReceive(object sender, ReceiveEventArgs e)
{
  ...
  NetworkPacket packet = new NetworkPacket();
  if(myChar < 0)
  {
        numPlayer = (int)e.Message.ReceiveData.Read(typeof(int));
        attr = (float[,])e.Message.ReceiveData.Read(typeof(float),6,8);
        anime[numPlayer].moveX = attr[numPlayer,2];
        anime[numPlayer].moveY = attr[numPlayer,3];
        myChar = numPlayer;
  }
  else
  {

        attr = (float[,])e.Message.ReceiveData.Read(typeof(float),6,8);

        //tower broken
        if (attr[4,1] <= 0)
        {
                ...
                //do same thing with host
                ...
        }

        //my character die
        if((attr[myChar,1] <= 0) && (firstTimeReceiveDeath))
        {
                times = DXUtil.Timer(DirectXTimer.GetApplicationTime);
                firstTimeReceiveDeath = false;
        }
        //change everything that u received
        for(int i=1;i<=numPlayer;i++)
        {
                //change position
                anime[i].moveX = attr[i,2];
                anime[i].moveY = attr[i,3];
                anime[i].moveZ = attr[i,4];

                //change animation
                if(attr[i,5] == 0)
                tempRootFrame[i].AnimationController.SetTrackAnimationSet(0
                        ,tempRootFrame[i].AnimationController.GetTrackAnimationSet(3));

                else if(attr[i,5] ==1)
                tempRootFrame[i].AnimationController.SetTrackAnimationSet(0
                        ,tempRootFrame[i].AnimationController.GetTrackAnimationSet(1));

                else if(attr[i,5] ==2)
                tempRootFrame[i].AnimationController.SetTrackAnimationSet(0
                        ,tempRootFrame[i].AnimationController.GetTrackAnimationSet(0));
        }

  }
}
```

- OnPlayerCreate ในฟังก์ชั่นนี้จะเป็น การทำงานในส่วนของผู้ให้บริการ จะทำการตรวจจับเหตุการณ์เมื่อมีผู้ใช้บริการเข้ามาของใช้บริการ เมื่อมีผู้ใช้บริการเข้ามาแล้วจะทำการกำหนดค่าเริ่มต้นต่างๆ ให้แก่ผู้ใช้บริการเช่น เป็นตัวละครที่เท่าไหร่ในผู้ให้บริการ,ตำแหน่งในแนวแกนต่างๆ สุดท้ายจะทำการส่งข้อมูลไปยังผู้ใช้บริการทุกคน

```
private void OnPlayerCreated(object sender, PlayerCreatedEventArgs e)
{
        try
        {
                NetworkPacket packet = new NetworkPacket();
                string playerName = ((Server)sender).GetClientInformation
                (e.Message.PlayerID).Name;

                string newtext = string.Format
                        ("Accepted new connection from {0}, UserID: 0x{1}",
                        playerName, e.Message.PlayerID.ToString("x"));
                AddText(newtext);
                numPlayer++;
                if(numPlayer == 1)
                {
                        attr[0,0] = numPlayer;
                        attr[numPlayer,0] = numPlayer;
                        attr[numPlayer,1] = 120.0f;
                        attr[numPlayer,2] = -414.0f;
                        attr[numPlayer,3] = -366.0f;
                        attr[numPlayer,4] = 0f;
                        attr[numPlayer,5] = 0;
                        attr[numPlayer,6] = 1;
                        attr[numPlayer,7] = 1;
                }
                // do same thing to all player
                ...
                packet.Write(numPlayer);
                packet.Write(attr);

                ServerConnection.SendTo((int)PlayerID.AllPlayers,packet
                        ,0,SendFlags.Guaranteed | SendFlags.NoLoopback);

                anime[numPlayer].moveX = attr[numPlayer,2];
                anime[numPlayer].moveY = attr[numPlayer,3];
        }
        catch { /* Ignore this, probably the server */ }
}
```

- OnFindHost ในฟังก์ชั่นนี้จะเป็น การทำงานในส่วนของผู้ใช้บริการ จะทำการหาผู้ให้บริการในที่นี้เมื่อหาผู้ให้บริการเจอแล้วจะทำการแสดงข้อความออกมาทาง console

```
private void OnFindHost(object sender, FindHostResponseEventArgs e)
{
        lock(this)
        {
                // Do nothing if we're connected already
                if (connected)
                        return;

                connected = true;
                string foundSession = string.Format
                        ("Found session ({0}), trying to connect.",
                        e.Message.ApplicationDescription.SessionName);

                this.BeginInvoke(new AddTextCallback(AddText),
                         new object[] { foundSession });

                // Connect to the first one
                ((Client)sender).Connect(e.Message.ApplicationDescription
                        , e.Message.AddressSender,e.Message.AddressDevice
                        , null, ConnectFlags.OkToQueryForAddressing);
        }
}
```

- OnConnectComplete ในฟังก์ชั่นนี้จะเป็น การทำงานในส่วนของผู้ใช้บริการ จะทำการตอบกลับผู้ใช้บริการว่าติดต่อกับ ผู้ให้บริการเรียบร้อยแล้ว

```
private void OnConnectComplete(object sender, ConnectCompleteEventArgs e)
{
        // Check to see if we connected properly
        if (e.Message.ResultCode == Microsoft.DirectX.DirectPlay.ResultCode.Success)
        {
                this.BeginInvoke(new AddTextCallback(AddText),
                        new object[] { "Connect Success." });

                connected = true;
        }
        else
        {
                this.BeginInvoke(new AddTextCallback(AddText), new object[] {
                        string.Format("Connect Failure: {0}", e.Message.ResultCode) });

                connected = false;
        }
}
```

3.2.3.7 การนำภาพ 2มิติ เข้ามาสู่โลก 3มิติ

เนื่องจาก DirectX ซึ่งเป็น API ที่เรานำมาใช้ในการพัฒนาเกมนี้ ได้ทำการแยกส่วนของการแสดงผลกราฟิก 2มิติ และ 3มิติ ออกจากกัน ซึ่งก็คือ DirectDraw และ Direct3D ดังนั้นเราจะทำอย่างไรเมื่อเราต้องการให้เกมของเรา ซึ่งให้ Direct3D ในการพัฒนาจึงจะสามารถนำ ภาพ 2มิติมาใช้งานในเกมของเราได้ ทาง Microsoft จึงได้มี Class ซึ่งใช้ช่วยในการนำภาพ 2มิติ มาใช้ในเกม 3มิติ คือ Class Sprite ซึ่งจะอธิบายในลายละเอียดต่อไป

3.2.3.7.1 ขั้นตอนการนำภาพ 2มิติ มาใช้ใน Direct3D

ในการนำภาพ2มิติมาใช้ใน3มิตินั้น เราจะต้องทำการสร้าง Surface ขึ้นมารองรับภาพ 2 มิตินั้นก่อน เพื่อที่จะให้ภาพ 2มิติ นั้นมีตัวตนอยู่ในโลก 3มิติได้ (เหมือนกับการทำ Texture) ซึ่งต่างกันตรงที่ ภาพ 2 มิติ (Sprite)ที่เราจะนำมาแสดงนั้นจะไม่ขึ้นอยู่กับ โลก 3มิติ ซึ่งก็คือ Sprite จะมีตำแหน่งอยู่บนจอภาพ ไม่ได้อยู่ในโลก 3 มิติที่เราสร้างขึ้น

- สร้างตัวแปร Sprite และ Texture เพื่อใช้ในการแสดงผล

```
private Sprite sprite;
private Texture spriteTexture;
```

- สร้าง Surface เพื่อมารองรับกับ Texture ที่สร้างขึ้น และทำการกำหนดค่า เริ่มต้นให้กับ surface ให้มีความสอดคล้องกับภาพ

```
public void InitializeGraphics()
{
        …
        using (Surface s = spriteTexture.GetSurfaceLevel(0))
        {
                SurfaceDescription desc = s.Description;
                textureSize = new Rectangle(0, 0,
                desc.Width,desc.Height);
        }
        …
}
```

- การโหลดภาพ2มิติ มาไว้ใน Texture

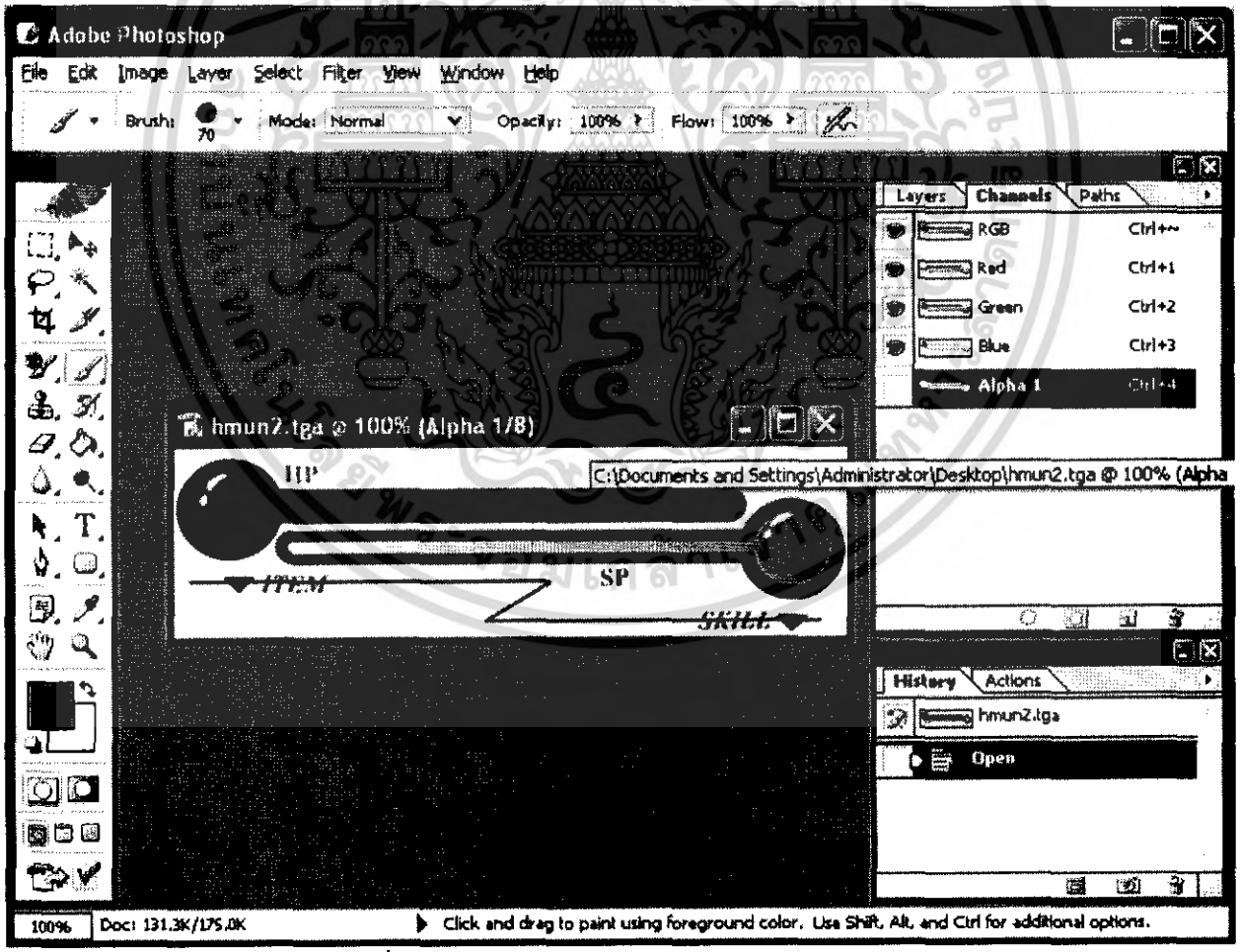
```
public void InitializeGraphics()
{
        …
        spriteTexture = TextureLoader.FromFile(device, @"image file");      …
}
```

- การนำภาพมาแสดงบนจอภาพ

```
protected override void OnPaint(Windows.Forms.PaintEventArgs e)
{
        ...
        device.BeginScene();

        // Begin drawing our sprites with alpha blend
        sprite.Begin(SpriteFlags.AlphaBlend);
        sprite.Draw(texture,textureSize,Center,position,Color);
        sprite.End();

        device.EndScene();

        device.Present();

        this.Invalidate();
        ...
}
```

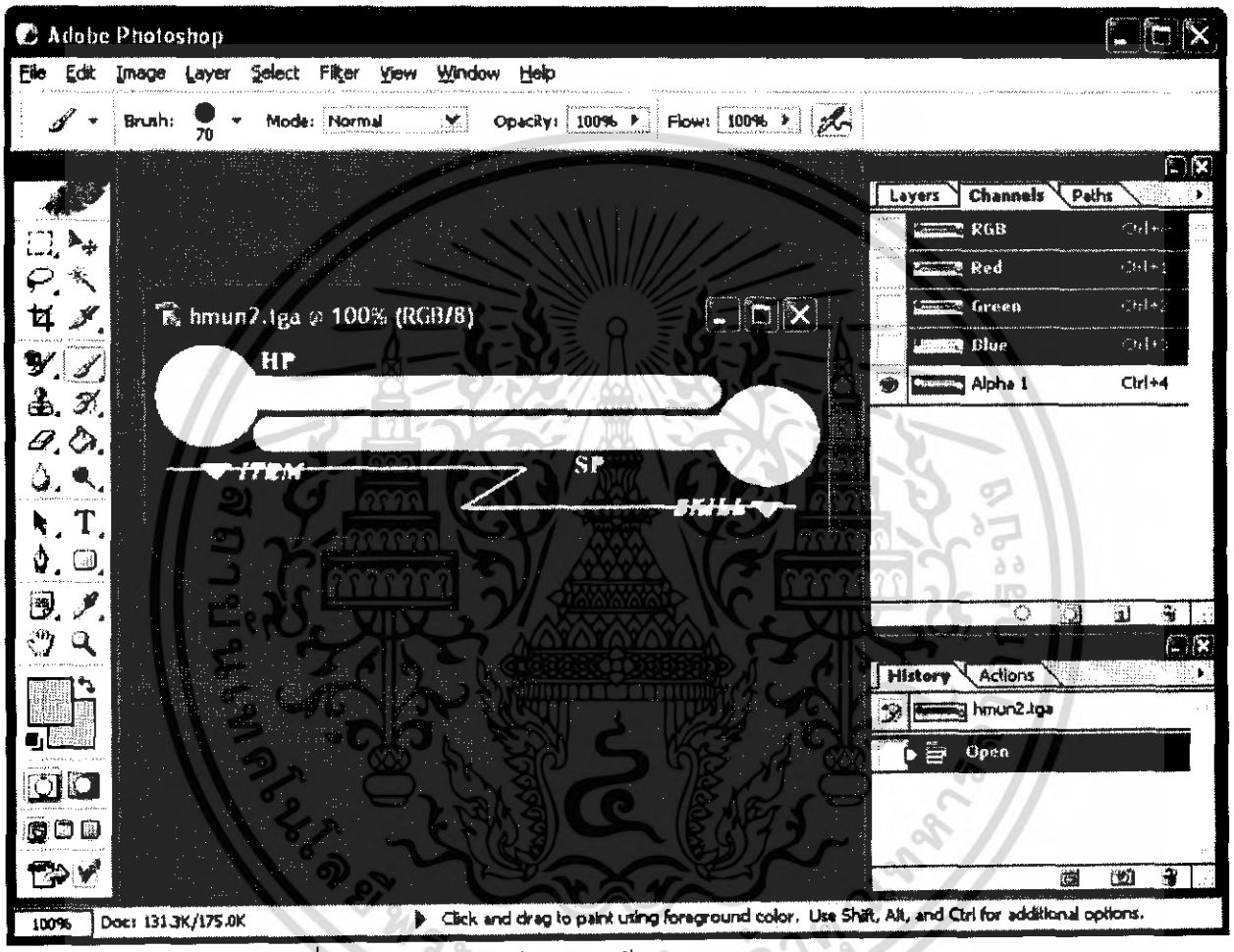
3.2.3.7.2 การทำภาพ 2มิติ ให้มีความโปร่งใสด้วย PhotoshopCS

- รูปแบบของ file รูปภาพที่จะสามารถทำให้โปร่งใสใน DirectX ได้นั้นจะต้องสนับสนุนการใช้งาน Alpha ซึ่งในที่นี้แนะนำให้ใช้ file ประเภท Targa(.tga file) ดังรูป

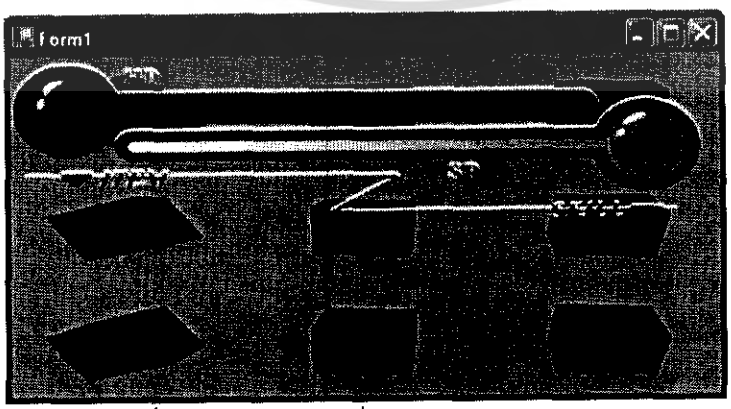


รูปที่ 3.16 แสดง channel ของรูป .tga

- กำหนดส่วนที่ต้องการจะแสดงและส่วนที่ต้องการจะให้โปร่งใส ในการสร้างภาพที่โปร่งใสได้นั้นขั้นแรกให้ทำการวาดภาพไปตามปรกติ เมื่อทำการวาดเรียบร้อยแล้วให้ใช้ Channels Toolbar ทำการเลือกไปที่ Alpha Channel เพื่อทำการปรับแต่งส่วนที่สามารถมองเห็นและส่วนที่จะให้โปร่งใสโดยบริเวณที่ต้องการจะให้โปร่งใสให้ใช้สีดำ ส่วนที่ต้องการจะให้มีการแสดงผลให้ลงสีขาวลงไป ดังรูป



รูปที่ 3.17 การทำรูปให้โปร่งใสด้วย alpha channel



รูปที่ 3.18 การนำภาพที่ปรับแต่งแล้วไปใช้

3.2.3.7.3 การเล่น Animation 2มิติ

ในการทำ Animation ในภาพ 2มิติ สามารถทำได้โดยการแสดงภาพที่มี ที่มีความต่อเนื่องกันทำให้ดูเหมือนว่าภาพนั้นกำลังเคลื่อนไหวอยู่ ซึ่งเป็นวิธีการเดียวกับการทำ Flip Book หรือการที่ทำการวาดภาพลงในกระดาษแต่ละแผ่น กระดาษแต่ละแผ่นถือว่าเป็น 1 frame แล้วทำการคลี่กระดาษ เราก็จะได้ภาพที่สามารถเคลื่อนไหวได้ ในการทำ Computer Animation เป็นใช้วิธีการเดียวกัน

- การเล่น Animation จาก file ภาพเพียง file เดียวเนื่องจากการfile ภาพแต่ละ file จะมี Header ซึ่งแสดงคุณสมบัติของ file ภาพนั้นๆ เช่น ขนาดของภาพ ความละเอียดของภาพ ฯลฯ ซึ่งในการแสดง animation file แต่ละ file โดยมากมักจะมี header ที่เหมือนกัน ดังนั้นการที่จะทำการเล่น 1 frame โดยต้องใช้ file 1 file จะทำให้ในการโหลดภาพแต่ละภาพออกมาจะเกิดงานที่ไม่จำเป็น (overhead) ซึ่งเกิดจากการโหลด file ในส่วนของ header ซึ่งจะเหมือนๆกันในแต่ละรูป ดังนั้นหากเราทำการรวมรูปภาพที่จะใช้ในการทำ animation เหล่านั้นไว้ในรูปๆเดียว ก็จะลดงานที่ไม่จำเป็นได้ดังรูป

รูปที่ 3.19 ภาพที่ใช้ในการเคลื่อนไหว

- การนำไฟล์ภาพเพียงไฟล์เดียวมาทำ Animation จะใช้วิธีการแสดงภาพแต่ละ frame โดยการแสดงเพียงบางส่วนของภาพใหญ่เพื่อทำหน้าที่เป็นแต่ละ frame ใน Animation

```
...
private Texture animationTexture;
private Rectangle animeTextureSize;
private Vector3 animationPosition = new Vector3(int x,int y,int z);
private int column = 0;
private int row = 0;
...

public bool InitializeGraphics()
{
        ...
        animation = TextureLoader.FromFile(device,
                @"..\..\image file");
        using (Surface s = animation.GetSurfaceLevel(0))
        {
                SurfaceDescription desc = s.Description;
                animeTextureSize = new Rectangle(0, 0,
                desc.Width, desc.Height);
        }
        ...
}
protected override void OnPaint(Windows.Forms.PaintEventArgs e)
{
        ...
        sprite.Begin(SpriteFlags.AlphaBlend);

        sprite.Draw(animation,
        new Rectangle(column * SpriteSizeWidth,
        row * SpriteSizeHeight,
        SpriteSizeWidth, SpriteSizeHeight),
        Center, animationPosition, Color.White);
        column++;
        if (column >= NumberSpritesCol)
        {
                row++;
                column = 0;
        }
        if (row >= NumberSpritesRow)
        {
                row = 0;
        }
        ...
}
```

3.2.3.8 การนำเสียงมาใช้ในเกม

DirectX ได้สร้างคลาสที่ไว้จัดการกระบวนการต่างๆเกี่ยวกับเสียง คือ คลาสDirectSound ซึ่งการนำคลาสนี้มาใช้งานจะต้องทำการ "Add Reference" คอมโพเนนที่ชื่อ DirectX.DirectSound เข้ากับโปรเจค และทำการเรียกคอมโพเนนท์มาใช้งานโดยคำสั่ง

```
using Microsoft.DirectX.DirectSound;
```

3.2.3.8.1 ทำการติดต่อกับการ์ดเสียง

ก่อนจะใช้งาน DirectSound ซึ่งเป็นคลาสที่ควบคุมการทำงานของการ์ดเสียง จะต้องทำการติดต่อกับคลาสเสียงโดยใช้ คอนสตรัคเตอร์ของ DirectX.DirectSound.Device()

```
private Microsoft.DirectX.DirectSound Device device = null;
...
public void InitializeSound()
{
        device = new Microsoft.DirectX.DirectSound.Device();
        device.SetCooperativeLevel(this,CooperativeLevel.Normal);
        ...
}
```

3.2.3.8.2 การโหลดเสียง

```
...
private SecondaryBuffer sound = null;
...
public void InitializeSound()
{
    ...
    sound = new SecondaryBuffer(@"..\..\drumpad-crash.wav",device);
    sound.Play(0, BufferPlayFlags.Default);
    ...
}
```

3.2.3.8.3 การเล่นเสียง

โดยในส่วนของ BufferPlayFlags จะเป็นส่วนที่บอกว่าเสียงที่ทำการเล่นนั้นจะทำการเล่นอย่างไร โดย flag ที่ใช้กันมากจะใช้ Default ซึ่งเป็นการเล่นเสียง 1 รอบ และ Looping จะเป็นการบอกว่าทำการเล่นเสียงนั้นเรื่อยๆ

```
sound.Play(0, BufferPlayFlags.Default);
```

บทที่4
ผลการทดลองและการวิเคราะห์ปัญหา

ในบทนี้ จะเป็นขั้นตอนของการทดสอบและผลที่ได้จากการทดสอบในแต่ละส่วนของเกมทั้งหมด โดยผลการทดสอบที่ได้นี้ จะถูกนำไปวิเคราะห์ถึงปัญหาและแนวทางในการแก้ปัญหาเพื่อการพัฒนาต่อไปในอนาคต โดยทางผู้จัดทำหวังเป็นอย่างยิ่งว่าผลการทดลองและแนวทางในการแก้ปัญหาที่ได้จากการทดสอบในครั้งนี้ จะเป็นประโยชน์และแนวทางต่อผู้ที่มีความสนใจจะศึกษาและค้นคว้าเพื่อจะนำไปศึกษาต่อ ให้ได้เห็นถึงปัญหา และข้อดีเสียของปัญหาโดยผู้ศึกษาไม่จำเป็นต้องทำการทดสอบเพื่อหาปัญหาด้วยตนเองอีกครั้ง

คุณสมบัติของระบบที่จะนำมาทดสอบ

1. ระบบปฏิบัติการ Microsoft Windows XP
2. หน่วยประมวลผลกลางที่มีความเร็ว 1.5 Ghz
4. หน่วยความจำหลักขนาด 512 MB
5. VGA Card ของ ATI RADEON 9550 หน่วยความจำบนการ์ดขนาด 128 MB
6. ระบบเสียง Sound Card ของ Creative Live 5.1

ขั้นตอนการดำเนินการทดสอบ

- ทำการรันและประมวลผลภายใต้ระบบที่กำหนด
- ตรวจสอบการใช้คำสั่งของปุ่มกดต่างๆและการทำงานของปุ่มกดต่างๆ
- ตรวจสอบการทำงานของโปรแกรมโดยการใช้เมาส์และคีย์บอร์ด
- ตรวจสอบการเชื่อมต่อผ่านระบบเครือข่าย
- ตรวจสอบการหา Error

จุดประสงค์ของการดำเนินการทดสอบ

- หลังจากติดตั้งตัวโปรแกรมแล้วสามารถใช้โปรแกรมได้
- การประมวลผลของโปรแกรมจะต้องทำความเร็วในระดับที่ยอมรับได้
- การตอบโต้ของคอมพิวเตอร์จะต้องเป็นไปตามที่กำหนดไว้
- การคลิกเมาส์และการกดแป้นของคีย์บอร์ดถูกต้องตามที่กำหนดไว้
- การใช้คำสั่งของปุ่มกดต่างๆเป็นไปตามรูปแบบ
- การเชื่อมต่อผ่านระบบเครือข่ายต้องสามารถทำได้
- จำนวน Error ที่เกิดขึ้นต้องอยู่ในเกณฑ์ที่ยอมรับได้

4.1 ขั้นตอนการดำเนินการทดสอบการประมวลผลภายใต้ระบบที่กำหนด

รันโปรแกรมภายใต้เครื่องคอมพิวเตอร์ที่กำหนด โดยดับเบิ้ลคลิกที่ไฟล์ "DotA.EXE"

รูปที่ 4.1 หน้าจอเริ่มต้น

ตาราง 4.1 ขั้นตอนการรันโปรแกรมภายใต้เครื่องคอมพิวเตอร์ที่กำหนด

การทดสอบ	ขั้นตอนการทดสอบ	ผลการทดสอบ
รันโปรแกรมภายใต้ระบบเครื่องคอมพิวเตอร์ที่กำหนด	ทำการรันโปรแกรมโดยดับเบิ้ลคลิกที่ไฟล์ "DotA.exe" ในโฟลเดอร์ปลายทางที่ได้ทำการติดตั้ง	โปรแกรมสามารถทำงานได้
การเลือกการทำงานของโปรแกรมเกมว่าจะเป็นผู้ให้บริการหรือผู้ใช้บริการ	กดลูกศรขึ้นหรือลงเพื่อเลือกตัวเลือกต่างๆ	สัญลักษณ์เลื่อนไปตามที่กำหนดและมีเสียงประกอบ

4.2 ขั้นตอนการดำเนินการทดสอบความสมบูรณ์ของเกม

รูปที่ 4.2 หน้าจอ Credit

ตารางที่ 4.2.1 การทดสอบหน้าจอ Credit

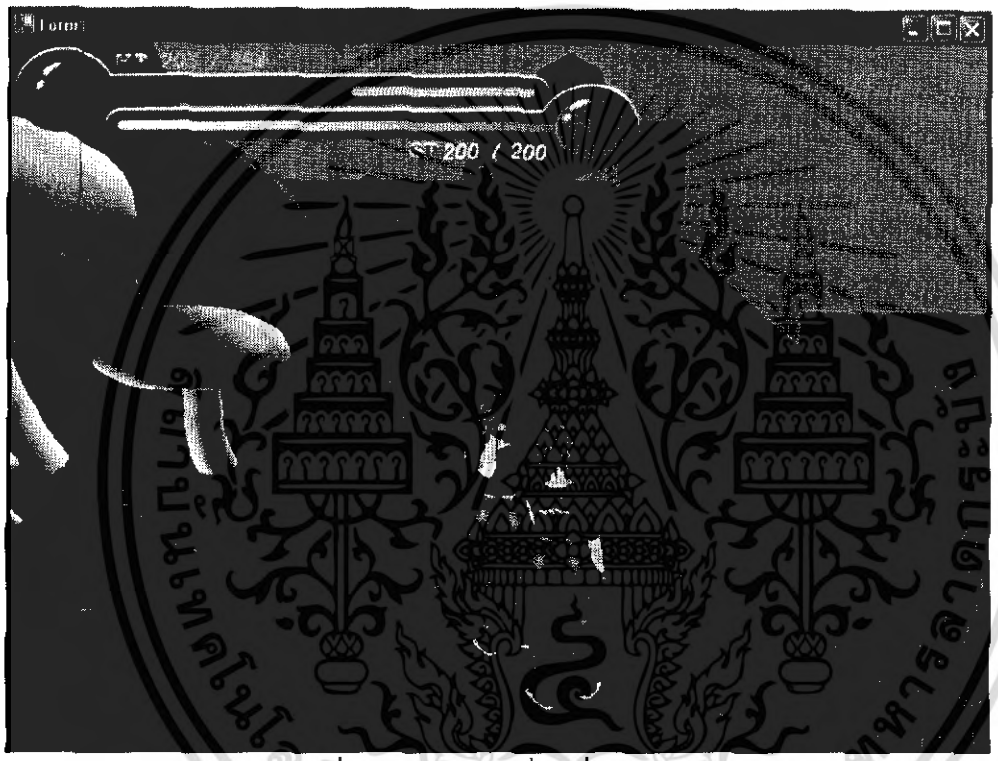
การทดสอบ	ขั้นตอนการทดสอบ	ผลการทดสอบ
การแสดงผลหน้าจอ Credit	เลื่อนสัญลักษณ์มายังคับว่า Credit แล้วกด Enter	หน้าจอเปลี่ยนไปยังหน้าจอ Credit และเสียงเพลงเปลี่ยนไป
การออกจากหน้าจอ Credit	กดปุ่ม Esc	กลับมาสู่หน้าจอเริ่มเกมอีกครั้ง

ตารางที่ 4.2.2 การทดสอบการเข้าเกม

การทดสอบ	ขั้นตอนการทดสอบ	ผลการทดสอบ
เข้าเกมในส่วนของผู้ให้บริการ	เลื่อนสัญลักษณ์มายังคับว่า Create Game แล้วกด Enter	หน้าจอเปลี่ยนไปยังหน้าจอเล่นเกม
เข้าเกมในส่วนของผู้ใช้บริการ	เลื่อนสัญลักษณ์มายังคับว่า Join Game แล้วกด Enter	หน้าจอเปลี่ยนไปยังหน้าจอเล่นเกม

ตารางที่ 4.2.3 ขั้นตอนการทดสอบการควบคุมตัวละคร

การทดสอบ	วิธีการทดสอบ	ผลการทดสอบ
การเดินหน้า	กดปุ่ม w	เกมแสดงผลของการเดินหน้า
การถอยหลัง	กดปุ่ม s	เกมแสดงผลของการถอยหลัง
การสไลด์ไปทางซ้าย	กดปุ่ม a	โมเดลสไลด์ไปทางซ้าย
การสไลด์ไปทางขวา	กดปุ่ม d	โมเดลสไลด์ไปทางขวา
การตี	คลิ๊กเมาท์ซ้าย	ตัวโมเดลจะยกมือขึ้นโจมตี



รูปที่ 4.3แสดงการเคลื่อนที่และการลดของเลือด

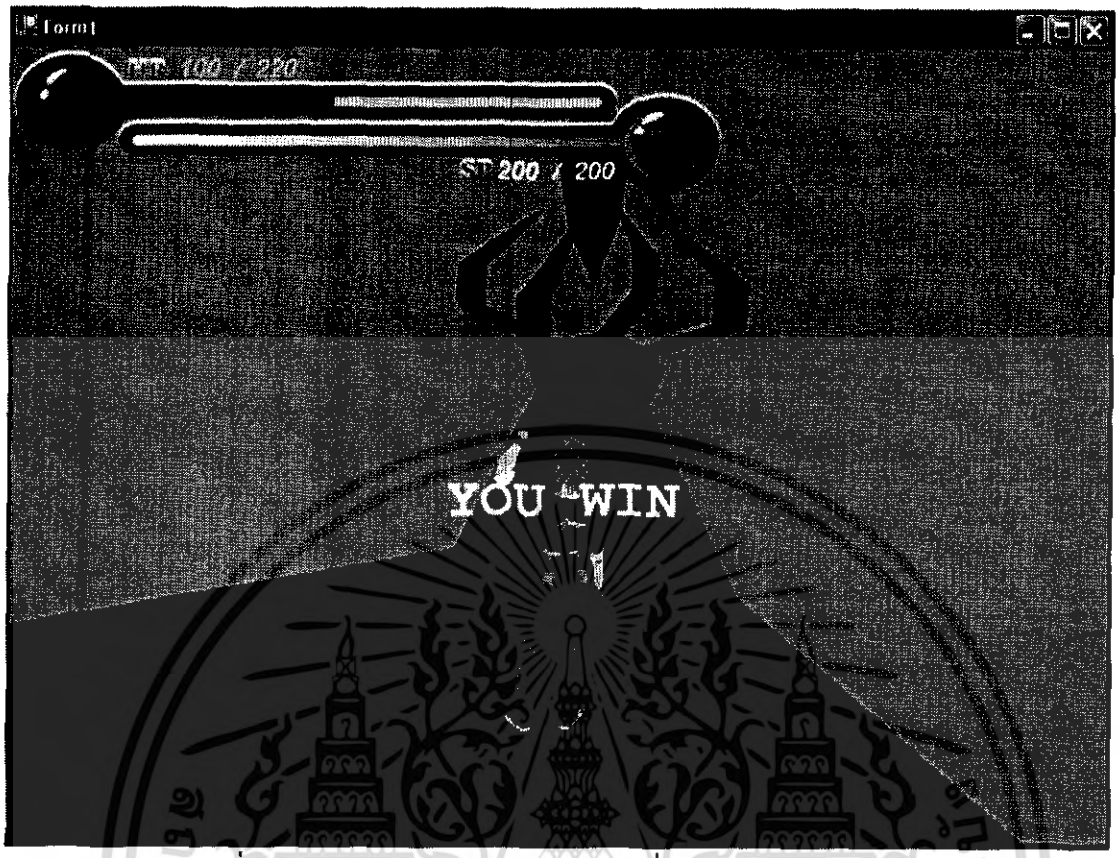
ตารางที่ 4.2.4 การโจมตีและการเคลื่อนไหว

การทดสอบ	**ขั้นตอนการทดสอบ**	**ผลการทดสอบ**
เคลื่นอนไหวผู้เล่นในเกม	กด w,s,a หรือ d เพื่อนเลื่อนตัว *ละคร*	ผู้เล่นอีกฝ่ายเห็นการเคลื่อนที่
การโจมตี	กดปุ่มคลิ๊กซ้ายในระยะประชิดตัว	ผู้เล่นฝ่ายที่ถูกโจมตีพลังชีวิต ลดลง

รูปที่ 4.4 แสดงการตายโดยถูกหน่วงเวลาเกิด

ตารางที่ 4.2.5 การทดสอบการตาย

การทดสอบ	**ขั้นตอนการทดสอบ**	**ผลการทดสอบ**
การตาย	ถูกผู้เล่นอีกฝ่ายโจมตีจนมีพลังชีวิตน้อยกว่าหรือเท่ากับ 0	ผู้เล่น ตาย โดยการถูกส่งกลับมายังจุดเกิดและลอยอยู่ข้างบนเมื่อครบเวลาที่กำหนดจะลงมาอยู่ข้างล่าง



รูปที่ 4.5 แสดงการชนะและพลังชีวิตลดเมื่อถูกป้อมปราการโจมตี

ตารางที่ 4.2.6 การทดสอบป้อมปราการ

การทดสอบ	**ขั้นตอนการทดสอบ**	**ผลการทดสอบ**
การถูกป้อมปราการโจมตี	เดินเข้าไปในระยะโจมตีของป้อมปราการฝ่ายตรงข้าม	พลังชีวิตลดลง
การโจมตีป้อมปราการ	เดินเข้าไปในระยะโจมตีแล้วกดคลิ๊กซ้ายเพื่อโจมตีป้อม	เมื่อโจมตีป้อมไประยะหนึ่งพลังชีวิตของป้อมหมดจะแสดงข้อความการจบเกมว่าฝ่ายเราแพ้หรือชนะ

4.3 ขั้นตอนการทดสอบการหาข้อผิดพลาดของโปรแกรมเกม

ในขั้นตอนการทดสอบเพื่อหาข้อผิดพลาดจะทำการทดสอบโดยการเล่นเกมหลายๆรอบ เพื่อหาข้อผิดพลาดและจากการทดสอบพบว่า ข้อผิดพลาดที่พบจะได้แก่

- การโจมตียังดูไม่สมจริง
- รัศมีการชนยังไม่สมบูรณ์
- ไม่สามารถรองรับผู้เล่นจำนวนมากๆได้

4.4 ปัญหาที่พบในขั้นตอนการทำปัญหาพิเศษและวิธีการแก้ไข

4.4.1 ความต่อเนื่องในการแสดงผล เมื่อทำการเล่นแบบหลายผู้เล่น

เริ่มแรกในการพัฒนาเกม ได้ทำการพัฒนาและทดลองอยู่บนเครื่องที่ทำการพัฒนา การเคลื่อนไหวต่างๆในเกมเป็นไปด้วยความต่อเนื่อง ไม่มีการสะดุดในการแสดงผลทางจอภาพ หลังจากนั้นได้ทำการพัฒนาต่อในส่วนของระบบเนตเวิร์ค โดยที่การเคลื่อนไหวต่างๆของแต่ละผู้เล่น จะกระทำผ่านทางเครื่องที่ทำหน้าที่เป็นเซอเวอร์ เช่น การย้ายตำแหน่ง เมื่อผู้เล่นต้องการจะทำการย้ายตำแหน่ง เครื่องของผู้เล่นจะทำการส่งข้อความมาบอกกับทางเครื่องเซอเวอร์ว่า จะขอทำการย้ายตำแหน่ง จากนั้นเครื่องเซอเวอร์จะตรวจสอบว่าการร้องขอนั้นสามารถทำได้หรือไม่ หากว่าสามารถทำได้เซอร์เวอร์ก็จะทำการย้ายตำแหน่งให้ โดยการส่งข้อความไปบอกกับผู้เล่นทุกคนว่าขณะนี้ ผู้เล่นแต่ละคนอยู่ที่ตำแหน่งไหนบ้าง และมีสถานะเป็นอย่างไร(ยืน , วิ่ง และโจมตี) เมื่อเครื่องของผู้เล่นได้รับข้อความแล้วก็จะทำการวาดสิ่งต่างๆให้ตรงกับทางเซอเวอร์

ดังนั้นเราจึงได้ทำการเขียนโคด ส่งข้อมูลของตัวผู้เล่นแต่ละคนให้กับทางเซอเวอร์ อยู่ในส่วนของฟังค์ชัน OnPaint() ซึ่งเป็นฟังค์ชันที่ใช้ในการวาด คือในการวาดเฟรมทุกๆ เฟรมทำภายในฟังค์ชันนี้ ซึ่งก็หมายความว่าทุกๆครั้งที่มีการวาดเฟรมแต่ละเฟรม จะมีการส่งข้อมูลต่างๆให้กับทางเซอเวอร์ แต่เมื่อทำการทดลองก็พบปัญหาว่าการทำเช่นนี้ทำให้เกิดการส่งข้อมูลไปยังเซอเวอร์มากเกินไป ทำให้เซอเวอร์ทำงานหนักเกินไป ซึ่งเซอเวอร์มีหน้าที่ในการกำหนดตำแหน่งของทุกๆผู้เล่นเมื่อเซอเวอร์ทำงานไม่ทัน จึงทำให้ระบบโดยรวมเกิดความล่าช้า ทำให้เสียอรรถรสในการเล่น

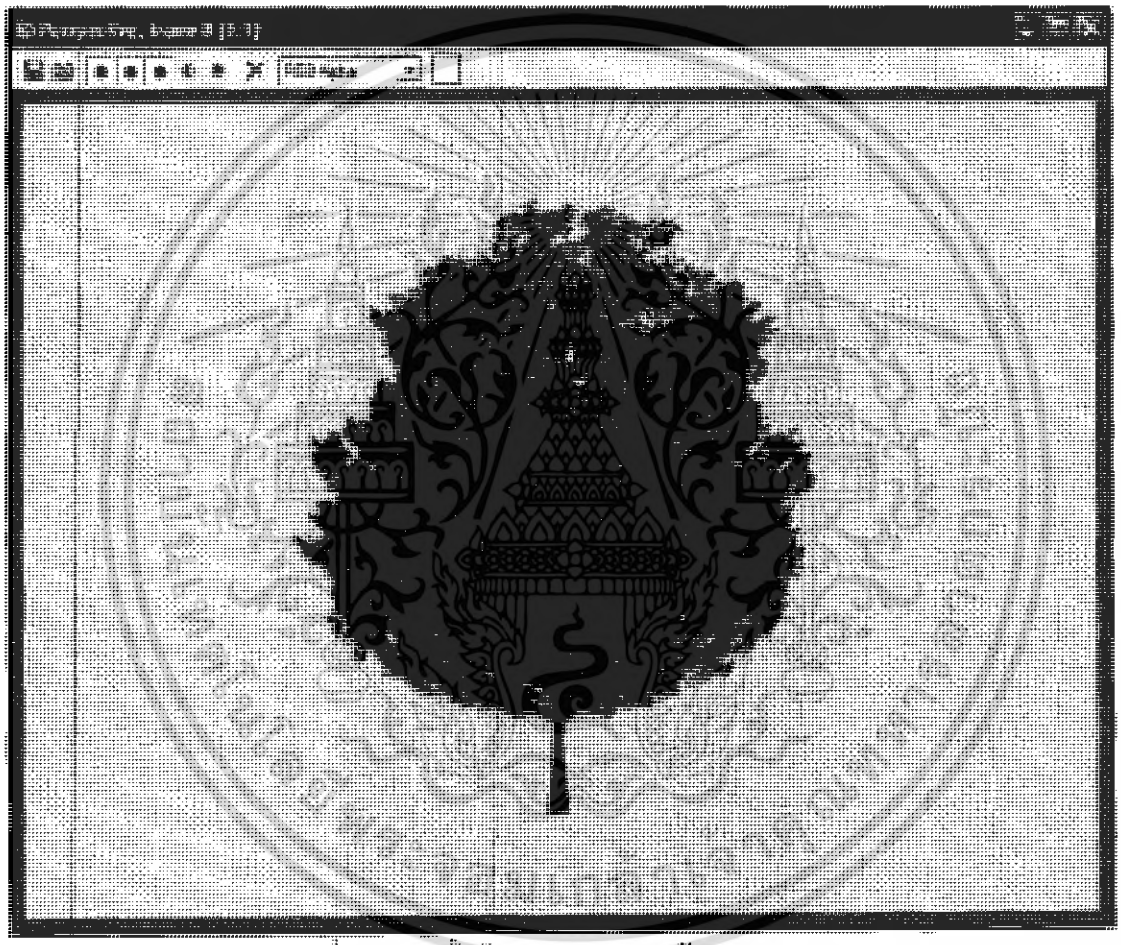
จะเห็นได้ว่าวิธีการข้างต้นมีการส่งข้อมูลให้กับทางเซอเวอร์มากเกินไป ทางผู้พัฒนาจึงได้เปลี่ยนจังหวะการส่งข้อมูลจากทำการส่งทุกเฟรม เป็นการส่งเฉพาะเมื่อทางผู้เล่นมีการป้อนอินพุตเข้ามา (ในฟังค์ชัน OnKeyDown() ,OnKeyUp() ,OnMoseDown() OnMouseUp()) ซึ่งมีผลทำให้เซอเวอร์ทำงานน้อยลง แต่ก็ทำให้ความลื่นไหลในการแสดงผล และความสมจริงลดลงไปด้วย

4.4.2 การนำโมเดลจาก 3D MAX มาใช้งาน

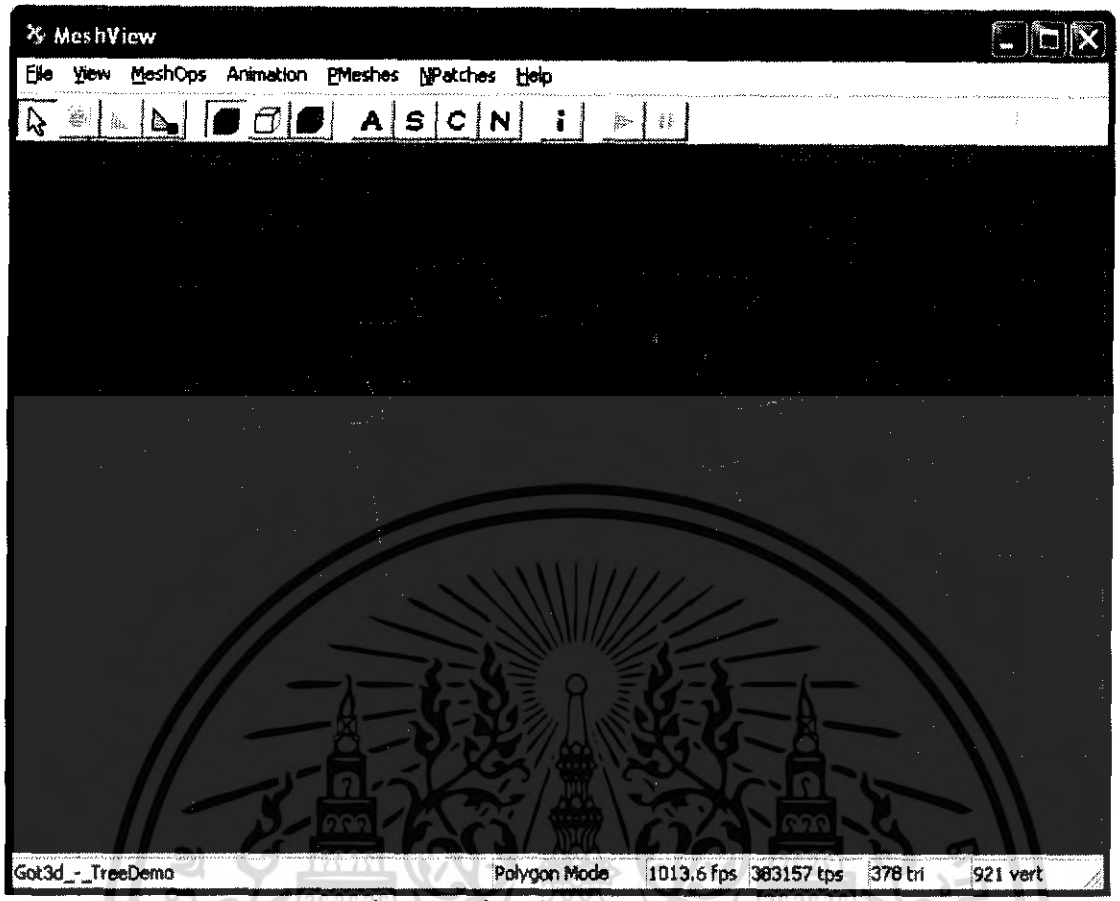
ปัญหาที่เกิดขึ้นในส่วนนี้โดยมากมักจะเกิดจากการเปลี่ยนฟอร์แมต จากฟอร์แมตของ 3d max file มาเป็นส่วนของ .X file ซึ่งเป็นฟอร์แมตที่ DirectX สามารถนำมาใช้งานได้ โดยปัญหาที่เกิดขึ้นจะเกิดขึ้นในหลายลักษณะตามหัวข้อต่อไปนี้

4.4.2.1 โมเดลที่ทำการเปลี่ยนฟอร์แมตไม่เหมือนกับต้นฉบับ

ในบางครั้งจะที่มีการแปลงฟอร์แมตจาก 3d max file มาเป็น .X file ในการแสดงผลโดย DirectX ซึ่งนำ .X file มาใช้งานนั้นผลที่ได้มีความแตกต่างจากต้นฉบับ เช่นในการ Export ไฟล์โมเดลต้นไม้มาใช้งานใน DirectX ดังรูป



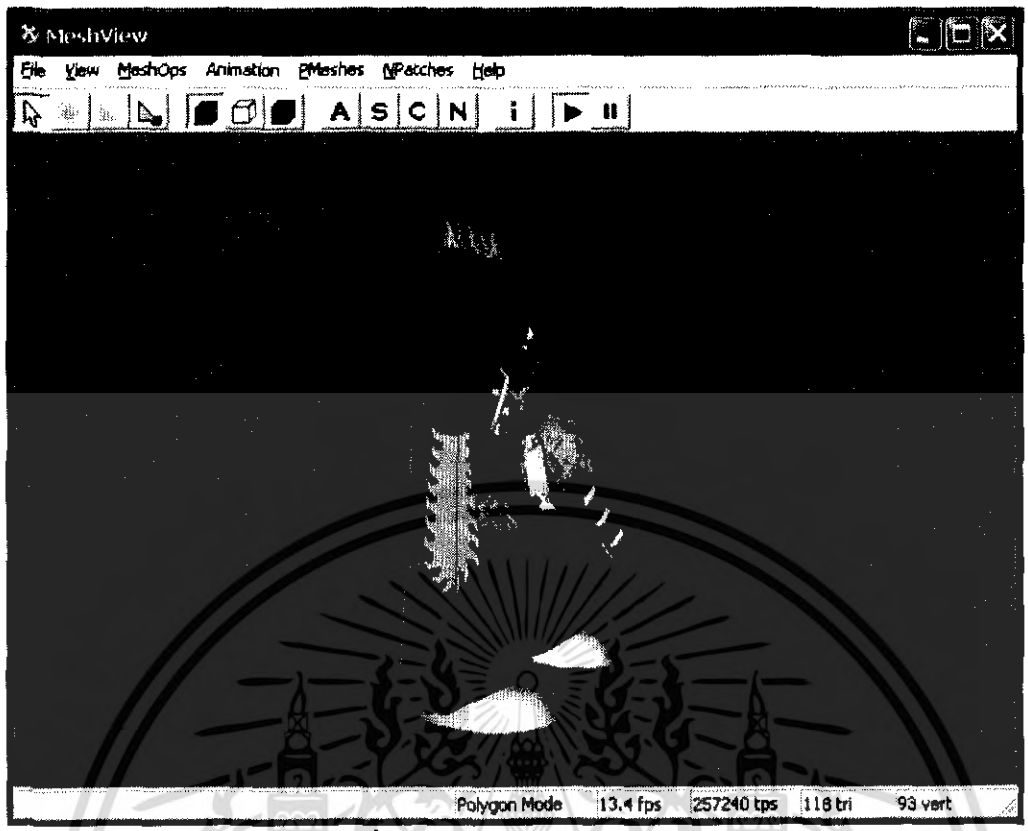
รูปที่ 4.6 ภาพที่ได้จากการ Render ด้วย 3D MAX



รูปที่4.7 ภาพที่ได้เมื่อนำ file .X มาใช้งาน

4.4.2.2 โมเดลที่ทำการแปลงฟอร์แมตไม่สามารถนำมาใช้งานได้

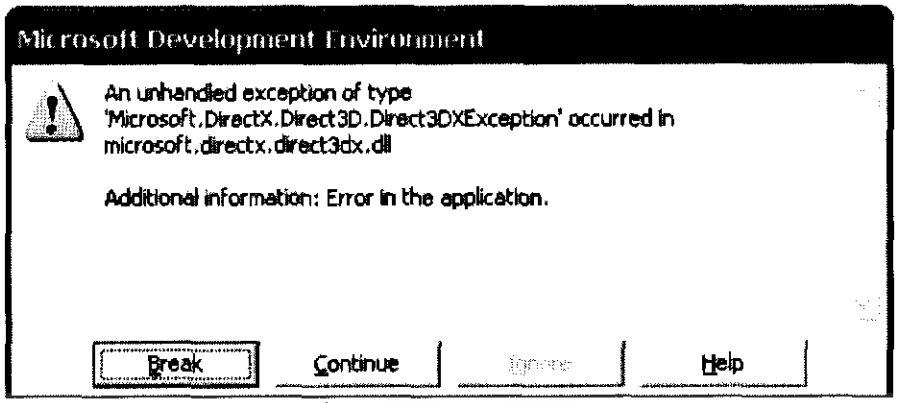
ในระหว่างทำการแปลงไฟล์โดยใช้โปรแกรมแพนด้าเอกซ์พอเตอร์ พบปัญหาเมื่อเรานำไฟล์ .X ที่ได้ไปใช้ในเกมส์ เพราะว่าDirectXไม่สามารถจะวาดไฟล์ .X นั้นได้แต่เราสามารถทดลองดูภาพไฟล์ .X นั้นได้ด้วย Mesh Viewer ซึ่งเป็นโปรแกรมทดสอบการวาดออบเจกต์ที่มีมากับDirectX



รูปที่ 4.8 โมเดลใน Mesh Viewer

เมื่อเราลองนำไฟล์ test.x ที่แปลงได้ไปใช้ในโปรแกรมเกมของเราจะเกิดปัญหา โดยโปรแกรมประมวลผลไม่ผ่าน ดังภาพ

รูปที่ 4.9 ทดสอบไฟล์ .X ในโปรแกรม



รูปที่ 4.10 ผลจากการคอมไพล์

สาเหตุที่โปรแกรมประมวลผลไม่ผ่านอาจเกิดจากปัญหาในตอนส่งออกไฟล์ .MAX เป็น .X โดยใช้โปรแกรมแพนด้าเอกซ์พอร์ตเตอร์ ซึ่งอาจจะทำให้คุณสมบัติบางอย่างของโมเดลที่เราใส่เข้าไปเช่น การกำหนดจุดหมุนของโมเดล จุดเชื่อมต่อ ตลอดจนจำนวนของเฟรมต่อแทร็คขาดหายไปได้ซึ่งอาจจะเป็นที่มาเมื่อเรานำไปใช้ในเกมส์ของเรา ซึ่งผู้ที่สนใจจะพัฒนาเกมส์โดยใช้โมเดลที่เป็นไฟล์ .X อาจลองใช้ซอฟต์แวร์ตัวอื่นในการแปลงไฟล์แทน ด้วยเหตุนี้เราจึงใช้โมเดล.X ที่หาได้จากตัวอย่างโปรแกรมแทนดังภาพ

รูปที่ 4.11 โมเดล .X ที่ใช้ในเกม

4.4.2.3 โมเดลมีขนาดใหญ่เกินไป

ในการทำปัญหาพิเศษมีโมเดลหลายตัวที่มีความสวยงาม แต่ไม่สามารถนำมาใช้งานได้เนื่องจากตัวโมเดลมีความละเอียดมากเกินไปทำให้เมื่อนำเข้ามาใช้ในเกมจะทำให้เครื่องคอมพิวเตอร์ทำงานมากจนเกินไป และทำให้เกมมีขนาดใหญ่จนเกินไป เนื่องจากโมเดลหลายๆตัวไม่ได้ถูกออกแบบมาเพื่อใช้กับการนำมาใช้ในเกม จึงต้องทำการเปลี่ยนโมเดล ให้มีขนาดเล็กลงทำให้ความสวยงามน้อยลงตามไปด้วย

รูปที่4.12 file โมเดลต้นไม้ที่มีขนาด 2.52MB

รูปที่ 4.13 file ฉากที่ใช้ในเกม มีขนาด 2.29MB

4.4.3 ปัญหาการชนแล้วเกิดการติดกัน

ในการชนกันของตัวละครในบางครั้งเมื่อตัวละครชนกันแล้วไม่สามารถเคลื่อนที่ต่อไปไหนได้เนื่องจากเมื่อมีการชนแล้วโปรแกรมเกมจะทำการห้ามเดินไปในทิศทางที่ชน ถึงแม้จะหมุนตัวละครไปในทิศทางอื่นแล้วก็ตามก็ยังไม่สามารถเดินไปในทิศทางที่ต้องการได้อยู่ เราจึงได้หาทางแก้ไขโดยเมื่อมีการเลื่อนเม้าไปทางซ้ายหรือขวาจะยอมให้เดินไปในทิศทางที่ชน 1 ก้าวเพื่อที่จะได้หลุดจากการชน แต่สิ่งนี้ก็ทำให้เกิดปัญหาใหม่ขึ้นมาคือตัวละครจะสามารถทะลุสิ่งกีดขวางได้หากทำการเลื่อนเม้าพร้อมกับการเดินในทิศทางที่ต้องการ

4.5 ประเมินประสิทธิภาพของเกมส์

จากเกมทดสอบที่ผ่านมาทั้งหมดทำให้เราสามารถประเมินประสิทธิภาพโดยรวมของเกมส์ได้ว่า เกมส์นี้สามารถทำงานได้ตามที่ออกแบบไว้พอสมควร โดยมีข้อผิดพลาดพอสมควรที่เกิดขึ้นโดยการทดสอบได้จากระบบที่กำหนดไว้ข้างต้นซึ่งอาจจะสามารถเปลี่ยนแปลง เพิ่มหรือลดประสิทธิภาพได้ ถ้าหากมีการเปลี่ยนระบบที่ใช้ในการประมวลผลของเกมส์ และเนื่องจากเกมส์นี้มีการติดต่อกับเสียงและภาพพอสมควรทำให้ระบบที่จะนำเกมส์นี้ไปประมวลผลต้องมีประสิทธิภาพสูงพอสมควร

บทที่ 5
สรุปผลการดำเนินงานและข้อเสนอแนะ

5.1 สรุปผลการดำเนินงาน

5.1.1 การศึกษาและรวบรวมข้อมูล

การศึกษาและรวบรวมข้อมูลที่จำเป็นต้องใช้ในการพัฒนาเกมทำได้โดยการศึกษาและรวบรวมข้อมูล โดยศึกษาก่อนว่าจะใช้ภาษาใดในการพัฒนาเกมภาษานั้นๆมีข้อดีข้อเสียอย่างไรและจะติดต่อกับตัวควบคุมดัวไหน (DirectX, OpenGL) ต่อมาต้องพิจารณาความเป็นไปได้ที่จะพัฒนาเกมนี้บนเครื่องคอมพิวเตอร์ได้หรือไม่อาจต้องพิจารณาจากหลายๆปัจจัยได้แก่

1. ความเร็วของหน่วยประมวลผลกลาง ต้องศึกษาว่า ความเร็วขั้นต่ำในการใช้พัฒนาเกมนี้ต้องมีความเร็วเท่าใดเพื่อให้เพียงพอในการประมวลผลอย่างราบรื่น
2. หน่วยความจำหลัก ต้องศึกษาว่า หน่วยความจำหลักในการใช้พัฒนาเกมนี้ต้องมีขนาดเท่าใด เพื่อให้เพียงพอในการประมวลผลอย่างราบรื่น
3. หน่วยประมวลผลทางภาพ เนื่องจากเกมที่พัฒนานั้นเป็นเกม 3 มิติ จึงจำเป็นต้องมีหน่วยประมวลผลทางภาพที่เร็วพอเพื่อให้แสดงผลได้อย่างราบรื่น

เมื่อทำการวิเคราะห์และตัดสินใจได้แล้วก็ทำการรวบรวมข้อมูลเพื่อสร้างกฎและกติกาที่จะนำมาเป็นแนวทางในการพัฒนาเกม พร้อมทั้งทำการศึกษาขั้นตอนวิธีการต่างๆในการพัฒนาเกม โดยอาจศึกษาจากตัวเกมต่างๆ ที่มีอยู่จากนั้นรวบรวมความรู้ทั้งหมดเพื่อนำไปใช้เป็นโครงสร้างและวิธีในการพัฒนาและการเลือกใช้เครื่องมือต่างๆ ที่เหมาะสมและสนับสนุนในการพัฒนา

5.1.2 การวิเคราะห์และออกแบบเกม

การออกแบบเกมต้องทำให้เกมมีความสวยงามตึงดูดผู้เล่น มีกติกาที่ชัดเจน สามารถทำงานได้ง่าย โดยการออกแบบของเกมนี้จะมีพื้นฐานมาจากเกม warcraft ในแผนที่ Defense of the Ancient ซึ่งทำให้การกำหนดกติกาและการทำงานของเกมดำเนินไปอย่างถูกต้อง เมื่อออกแบบเกมเป็นที่เรียบร้อยแล้วก็ทำการวิเคราะห์เกมว่า มีส่วนใดที่ควรจะเพิ่มส่วนใดที่ควรจะตัดออก และ ส่วนใดที่ควรจะเปลี่ยนแปลงเพื่อให้เกมที่พัฒนาขึ้นมานั้นมีความสนุกมากยิ่งขึ้น เมื่อทำการวิเคราะห์เรียบร้อยแล้ว จะนำข้อมูลต่างๆ ที่รวบรวมไว้มาประกอบกับความรู้ความสามารถที่มีอยู่ รวมไปถึงความรู้ที่ได้มาจากการศึกษาค้นคว้าเพิ่มเติมจากแหล่งข้อมูลต่างๆทั้งทางหนังสือ และ เวปไซต์ ที่ให้ความรู้ในการเขียนเกม จากนั้นทำการวางโครงสร้างและออกแบบหน้าตาส่วนติดต่อผู้ใช้ รวมถึงขั้นตอนวิธีการต่างๆ ในการพัฒนาเกมให้เสร็จตามระยะเวลาที่กำหนด แล้วจึงทำการแบ่งหน้าที่และมอบหมายส่วนต่างๆให้กับสมาชิก

5.1.3 การสร้างตัวละคร ภาพและ เสียงต่างๆ

ขั้นตอนนี้สารมารถทำควบคู่ไปได้กับการเขียนโปรแกรม โดยเกมนี้ได้ใช้เครื่องมือหลายอย่างที่ช่วยในการออกแบบและสร้างตัวละคร ภาพ และเสียงต่างๆ รวมทั้งการสร้างภาพเคลื่อนไหวภายในเกม ซึ่งตัวละครที่ใช้ และฉากต่างๆ ในเกมส่วนมาก จะเป็นการดาวโหลดมาจาก อินเตอร์เน็ต แล้วมาทำการดัดแปลง หรือเพิ่มท่าทางเข้าไป โดยเครื่องมือที่ใช้ในการออกแบบและสร้างงานกราฟิกได้แก่ โปรแกรมสร้างตัวละครสามมิติ 3D Studio Max และโปรแกรมตกแต่งรูปภาพ Adobe Photoshop ส่วนเครื่องมือที่ใช้ในการออกแบบและสร้างงานด้านเสียง Effect และเสียงประกอบได้แก่ โปรแกรมสร้าง และตัดต่อเสียง Adobe Audition

5.1.4 การพัฒนาโปรแกรม

ขั้นตอนนี้จะเป็นขั้นตอนที่สำคัญและใช้ระยะเวลายาวนานที่สุดในการพัฒนาเกม เราะว่าขั้นตอนนี้จะเป็นการนำความรู้ที่เรารวบรวมได้ทั้งหมดมารวมกับ โครงสร้าง รูปแบบ และกติกาของเกมที่เราจะพัฒนา รวมไปถึงภาพและเสียงที่เราได้จัดเตรียมไว้และเมื่อเริ่มต้นพัฒนาก็ต้องมีการค้นหาความรู้ใหม่ๆ มากมายนอกเหนือจากที่ค้นคว้าไว้ในตอนแรก มีการวิจัยและพัฒนาอัลกอริธึมที่ใช้ในการควบคุมกระบวนการของเกมให้เป็นไปตามกฎและกติกาที่วางเอาไว้ และทำการทดสอบการทำงานของเกมที่พัฒนาไปเรื่อยๆ จึงต้องใช้ระยะเวลาในขั้นตอนนี้มากพอสมควร ซึ่งขั้นตอนการทำงานในส่วนนี้มีซอร์ฟแวร์ และเครื่องมือที่ใช้ในการพัฒนา ดังนี้ *Microsoft Visual C#* ,DirectX SDK

5.2 ข้อจำกัดของโปรแกรม

5.2.1 ข้อจำกัดทาง ระบบปฏิบัติการ

เกมที่เราได้พัฒนานี้เป็นเกมที่พัฒนาบนพื้นฐานโครงสร้างของ ระบบปฏิบัติการ Microsoft Windows *เท่านั้น*

5.2.2 ข้อจำกัดการแสดงผล

เกมนี้จะทำงานที่ความละเอียด 800 x 600 pixels เนื่องจากเกมนี้เป็นเกม 3 มิติ จึงต้องมีการประมวลผลทางภาพเป็นอย่างมากและจะต้องคอยตรวจสองเงื่อนไขต่างๆอยู่ตลอดเวลาจึงต้องใช้หน่วยประมวลผลกลางและ หน่วยประมวลผลภาพที่มีประสิทธิภาพสูงพอสมควร เพื่อที่จะทำให้เกมดำเนินไปอย่างราบรื่น

5.3.3 ข้อจำกัดทางเครือข่าย

การรับส่งข้อมูลในเกมมีการรับส่งข้อมูลที่มากเกินไปทำให้ถ้าหากมีผู้เล่นเข้ามามากอาจทำให้เกิดการกระตุกได้

5.3 ข้อเสนอแนะ

เนื่องจากเกมที่พัฒนานี้เป็นเกมที่มีกฎและกติกาค่อนข้างซับซ้อน และปลีกย่อยมากมาย จึงทำให้ในการพัฒนาจำเป็นต้องมีอัลกอริทึมในการควบคุมการระบวนการของเกมที่ดี และต้องมีการพัฒนาโปรแกรมที่มีโครงสร้างซับซ้อนมากขึ้น ด้วยเหตุนี้ทำให้การออกแบบอัลกอริธึมของเกมต้องทำด้วยความรอบคอบ และต้องคอยตรวจสอบผลการทำงานของเกมไปทุกระยะ เพื่อตรวจหาข้อผิดพลาดที่อาจเกิดขึ้น นอกจากนี้ยังสามารถนำเกมนี้ไปพัฒนาโดยการสร้างกฎและกติกาใหม่ๆ หรือความพิเศษใหม่ๆ เพื่อให้ดูมีเหมาะสมมากขึ้นได้ตามต้องการ

บรรณานุกรม

[1] ณัช ภู่วรวรรณ, การเขียนโปรแกรมภาษา C# , บริษัท ซีเอ็ด จำกัด

[2] ปิยะบุตร สุทธิดารา, 3ds max6 basic 2004 ,บริษัท อินโฟเพรส บุ๊คส จำกัด

[3] Tom Miller, Managed DirectX® 9 Kick Start: Graphics and Game Programming, Sams Publishing

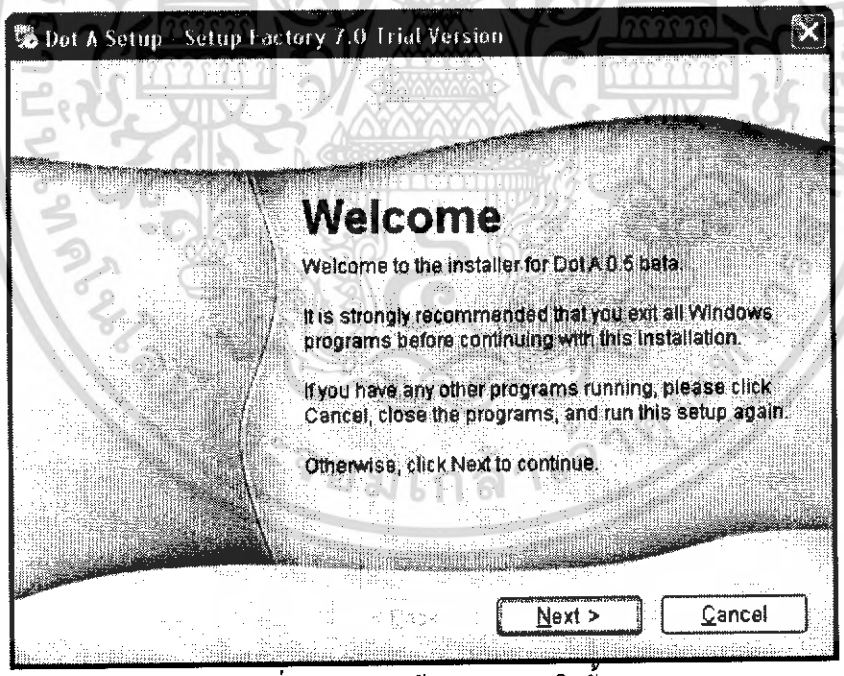
ภาคผนวก ก.
คู่มือการติดตั้ง

คุณสมบัติขั้นต่ำของเครื่องคอมพิวเตอร์ที่ต้องการ

1. ระบบปฏิบัติการ Microsoft Windows XP Professional
2. หน่วยประมวล CPU 2.4 GHz
3. หน่วยความจำหลักขนาด 512 MB
4. หน่วยความจำบนการ์ดจอขนาด 128 MB
5. หน่วยความสำรองต้องมีพื้นที่ว่างเหลือไม่ต่ำกว่า 130 MB
6. ต้องสนับสนุน DirectX 9.0 ขึ้นไป
7. มี Network การ์ด

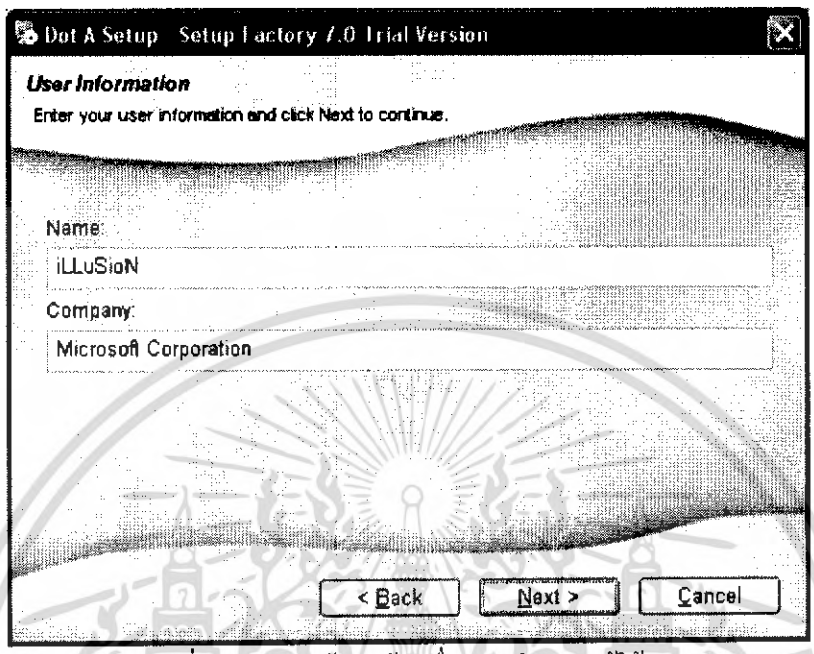
วิธีการติดตั้งโปรแกรมเกม Dot A มีดังต่อไปนี้

1. รันไฟล์ Setup.exe จะปรากฏหน้าจอดังภาพ



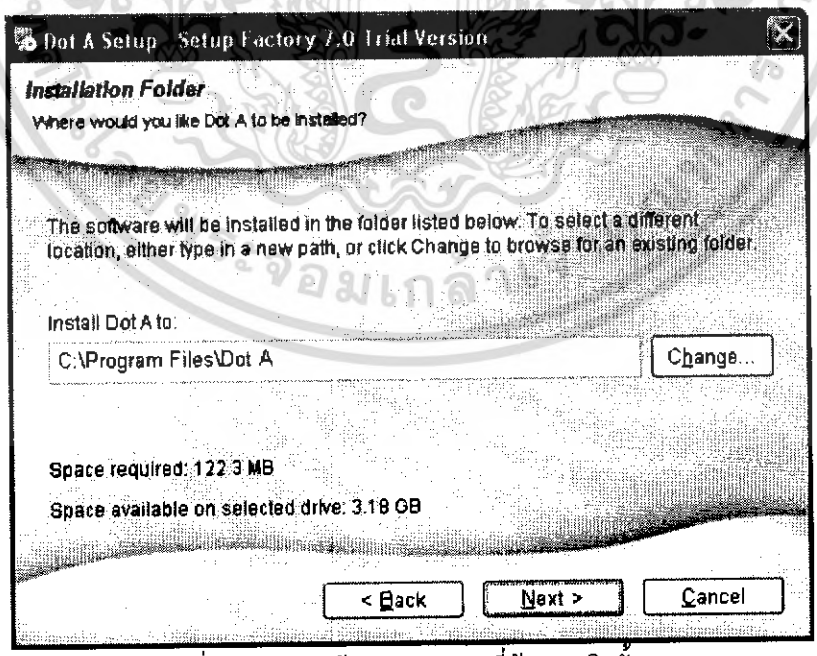
รูปที่ ก.1 แสดงหน้าแรกของการติดตั้ง

2. เมื่อกด Next แล้วจะเข้าสู่ขั้นตอนถัดไปคือการป้อน ชื่อผู้ใช้ และชื่อบริษัท



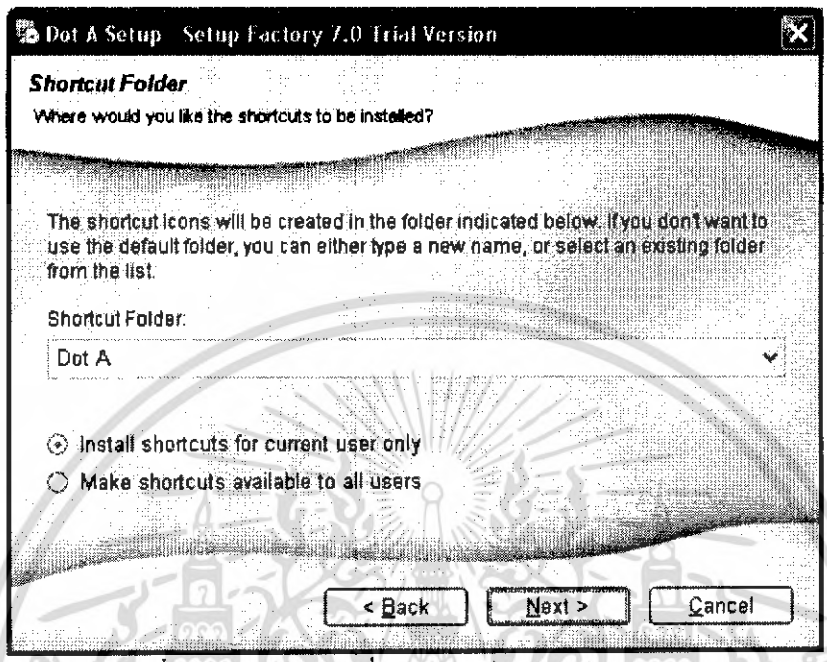
รูปที่ ก.2 แสดงหน้าการป้อนชื่อและบริษัทของผู้ใช้

3. เมื่อป้อนข้อมูลเสร็จแล้วกด Next จะเป็นขั้นตอนการเลือก Path ที่ต้องการลงโปรแกรม โดยเนื้อที่ของโปรแกรมที่ใช้ คือ 122 MB



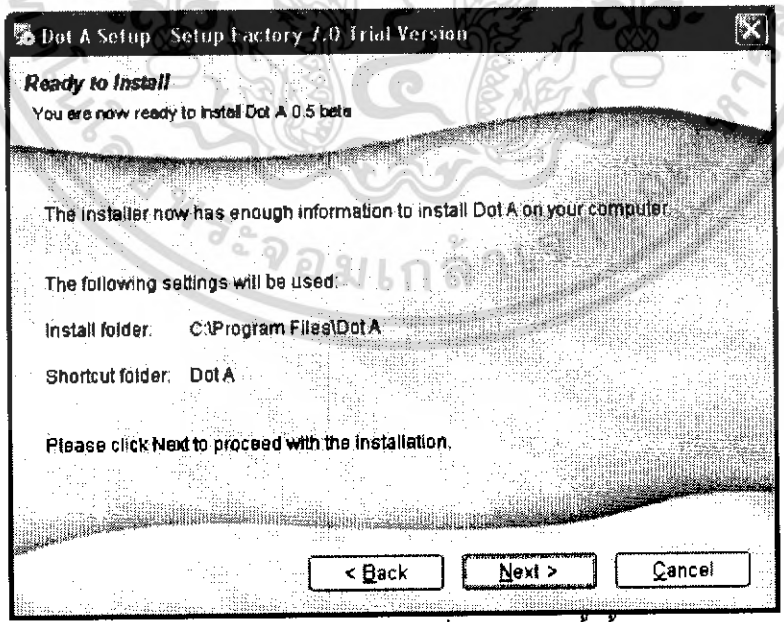
รูปที่ ก.3 แสดงหน้าการระบุ Path ที่ต้องการติดตั้ง

4. ขั้นต่อไปเป็นการเลือกที่ใส่โฟลเดอร์ลัด (Shortcut Folder) และเลือกผู้ใช้ที่สามารถใช้ได้



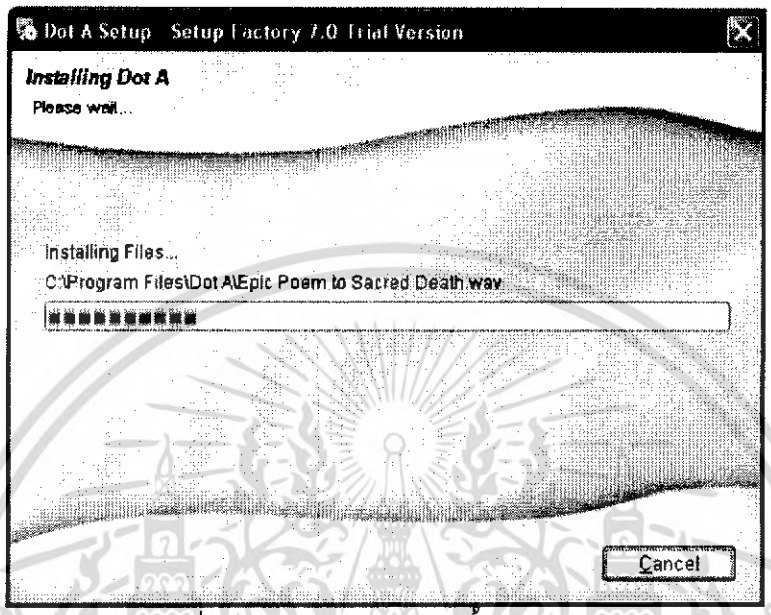
รูปที่ ก.4 แสดงการเลือกที่ใส่โฟลเดอร์ลัด (Shortcut Folder)

5. ขั้นตอนต่อไปเป็นการแสดงรายละเอียดข้อมูลที่จะทำการติดตั้งทั้งหมด ดังภาพ



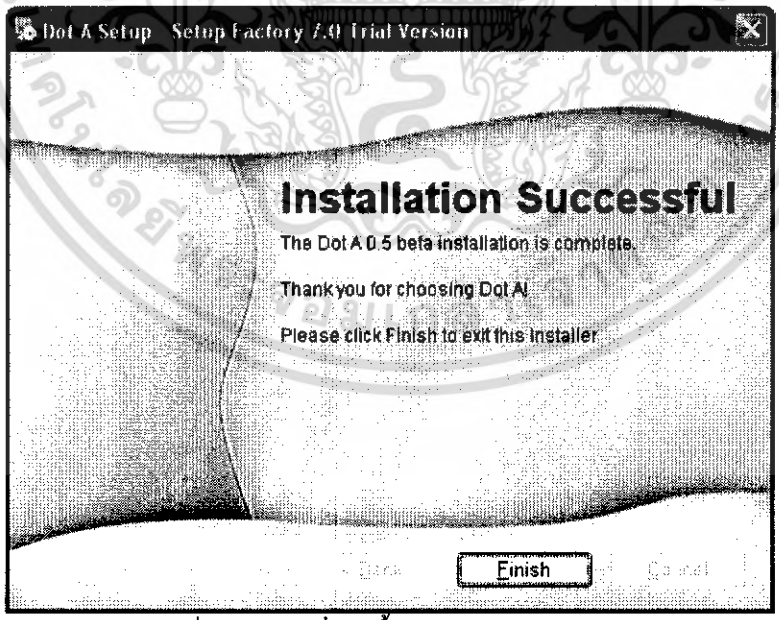
รูปที่ ก.5 แสดงรายละเอียดข้อมูลที่จะทำการติดตั้งทั้งหมด

6. หลังจากนั้นจะเป็นการเริ่มการติดตั้ง



รูปที่ ก.6 แสดงกระบวนการติดตั้งโปรแกรม

7. เมื่อติดตั้งเสร็จสมบูรณ์จะปรากฏหน้าจอดังภาพ



รูปที่ ก.7 แสดงเมื่อติดตั้งโปรแกรมเสร็จสมบูรณ์