การพัฒนาโปรแกรมเกมสามมิติโดยการควบคุมผ่าน Wiimote ด้วย DirectX

DEVELOPMENT OF 3D GAME BY WIIMOTE CONTROLLING USING DIRECTX API

นายดำเนิน เพชรชื่นสกุล
นายนิพิฐพนธ์ วงศ์ศุภรัตน์กุล

โครงงานนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิทยาศาสตรบัณฑิต
สาขาวิชาวิทยาการคอมพิวเตอร์
คณะวิทยาศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2555

การพัฒนาโปรแกรมเกมสามมิติโดยการควบคุมผ่าน Wiimote ด้วย DirectX

DEVELOPMENT OF 3D GAME BY WIIMOTE CONTROLLING USING DIRECTX API

นายดำเนิน เพชรชื่นสกุล

นายนิพิฐพนธ์ วงศ์ศุภรัตน์กุล

โครงงานพิเศษนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิทยาศาสตรบัณฑิต

สาขาวิทยาการคอมพิวเตอร์

คณะวิทยาศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2555

DEVELOPMENT OF 3D GAME BY WIIMOTE CONTROLLING USING DIRECTX API

MR.DAMNERN PETCHUENSAKUL

MR.NIPITPON WONGSUPARATKUL

A SPECIAL PROJECT SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF BACHELOR OF SCIENCE

IN COMPUTER SCIENCE

FACULTY OF SCIENCE

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

ACADEMIC YEAR 2012

หัวข้อปัญหาพิเศษ	การพัฒนาโปรแกรมเกมสามมิติโดยการควบคุมผ่าน Wiimote ด้วย DirectX DEVELOPMENT OF 3D GAME BY WIIMOTE CONTROLLING USING DIRECTX API
ชื่อนักศึกษา	นายดำเนิน เพชรชื่นสกุล 52050707 นายนิพิฐพนธ์ วงศ์ศุภรัตนกุล 52050729
ปริญญา	วิทยาศาสตรบัณฑิต
ภาควิชา	วิทยาการคอมพิวเตอร์
อาจารย์ที่ปรึกษา	ผศ.ดร.กรกช ประชุมรักษ์

คณะวิทยาศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง อนุมัติให้ปัญหาพิเศษนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตร วิทยาศาสตรบัณฑิต สาขาวิชาวิทยาการคอมพิวเตอร์ประจำปีการศึกษา 2555

คณะกรรมการสอบ	ลายมือชื่อ
อาจารย์ธีระ ศิริธีรากุล ประธานกรรมการ	
ดร.วรางคณา กิ้มปาน กรรมการ	
ผศ.ดร.กรกช ประชุมรักษ์ กรรมการและอาจารย์ที่ปรึกษา	

หัวข้อปัญหาพิเศษ	การพัฒนาโปรแกรมเกมสามมิติโดยการควบคุมผ่าน Wiimote ด้วย DirectX DEVELOPMENT OF 3D GAME BY WIIMOTE CONTROLLING USING DIRECTX API		
ชื่อนักศึกษา	นายดำเนิน	เพชรชื่นสกุล	52050707
	นายนิพิฐพนธ์	วงศ์ศุภรัตนกุล	52050729
ปริญญา	วิทยาศาสตรบัณฑิต		
ภาควิชา	วิทยาการคอมพิวเตอร์		
ปีการศึกษา	2555		
อาจารย์ที่ปรึกษา	ผศ.ดร.กรกช ประชุมรักษ์		

บทคัดย่อ

ปัญหาพิเศษชิ้นนี้มุ่งเน้นไปที่การศึกษาและพัฒนาโปรแกรมเกม 3 มิติ ตั้งแต่ขั้นตอนการสร้างระบบของโปรแกรมเกม ออกแบบ และพัฒนาจนได้เป็นเกม Action-RPG 3 มิติ ที่มีคุณสมบัติให้ผู้เล่นสามารถใช้ Wiimote ในการควบคุมเกมได้ โดยมีการประยุกต์ร่วมกันระหว่าง DirectX ซึ่งเป็น API ที่มีประสิทธิภาพและเป็นที่นิยมในการพัฒนาโปรแกรมเกมมาประยุกต์ใช้ในส่วนของการแสดงภาพกราฟิก 3 มิติ พร้อมเทคนิคต่างๆ กับการประยุกต์เอา Wiimote API มาใช้ร่วมกัน ทำให้โปรแกรมเกมสามารถรับข้อมูลนำเข้าจาก Wiimote ได้

คำสำคัญ : การพัฒนาโปรแกรมเกม, DirectX API, Wiimote

Title	DEVELOPMENT OF 3D GAME BY WIIMOTE CONTROLLING USING DIRECTX API
Student	Mr.Damnern Petchuensakul 52050707
	Mr.Nipitpon Wongsuparatkul 52050729
Degree	Bachelor of Science
Major Program	Computer Science
Academic Year	2012
Advisor	Asst. Prof. Dr. Korakot Prachumrak

ABSTRACT

This special project emphasizes on studying and developing 3D game application, including the system creation, the design and the development of this 3D Action-RPG game. This game has ability of playing with Wiimote. This game is developed using DirectX, a powerful and popular graphic API, with Wiimote API, making the game able to receive input from Wiimote.

Keywords: Game Programming, DirectX API, Wiimote

กิตติกรรมประกาศ

ในการจัดทำโครงงานปัญหาพิเศษ เรื่องการพัฒนาโปรแกรมเกมสามมิติโดยการควบคุมผ่าน Wiimote ด้วย DirectX นี้ สามารถสำเร็จลุล่วงได้ คณะผู้จัดทำขอขอบพระคุณ ผศ.ดร.กรกช ประชุมรักษ์ อาจารย์ที่ปรึกษาปัญหาพิเศษ คณะกรรมการสอบปัญหาพิเศษซึ่งประกอบด้วย ดร. วรางคณา กิ้มปาน และ อ.ธีระ ศิริธีรากุล ที่ได้เสียสละเวลาให้คำแนะนำในการปรับปรุงคู่มือการทำโครงงานพิเศษนี้ให้มีความสมบูรณ์มากยิ่งขึ้น ขอบคุณนายเบญจพล ศรีวิสุทธิ์สำหรับคำแนะนำในส่วนเนื้อหาของเกม และบุคคลต่างๆ ที่เกี่ยวข้องที่ได้ให้ความช่วยเหลือในการให้ข้อมูลข้อเสนอแนะในการทำโครงงานพิเศษนี้

ขอขอบคุณอาจารย์สาขาวิชาวิทยาการคอมพิวเตอร์ทุกๆท่านที่ได้ประสิทธิ์ประสาทความรู้ทั้งในภาคทฤษฎีและภาคปฏิบัติแก่ผู้จัดทำตลอดเวลาทั้ง 4 ปี จนกระทั่งโครงงานพิเศษเสร็จสมบูรณ์ได้ด้วยดีทุกประการ

สุดท้ายนี้คณะผู้จัดทำขอขอบพระคุณ บิดา มารดา และครอบครัว ที่ให้ความสนับสนุนด้านกำลังใจและทุนทรัพย์ และเพื่อนๆของคณะผู้จัดทำ ที่คอยให้กำลังใจ ให้ความช่วยเหลือ ทำให้คณะผู้จัดทำมีกำลังใจสามารถดำเนินงานพัฒนาโครงงานพิเศษนี้จนสำเร็จ

คณะผู้จัดทำ

สารบัญ

สารบัญ(ต่อ)

สารบัญ(ต่อ)

สารบัญ(ต่อ)

สารบัญ(ต่อ)

สารบัญ(ต่อ)

สารบัญรูป

สารบัญรูป (ต่อ)

สารบัญรูป (ต่อ)

สารบัญตาราง

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญ

อุปกรณ์นำเข้าที่ใช้ในการควบคุมเกมคอมพิวเตอร์พีซีทั่วไปที่รู้จักกันดีก็คือ เมาส์ แป้นพิมพ์ และจอยสติก ซึ่งอุปกรณ์เหล่านี้ล้วนแล้วผู้เล่นจะต้องนั่งอยู่กับที่ แต่สำหรับ Wiimote ของเครื่องคอนโซล Nintendo Wii แล้วมีคุณสมบัติที่ทำให้ผู้เล่นสามารถเคลื่อนไหวร่างกายไปด้วยได้ขณะที่เล่น จึงเป็นที่มาของโครงงานชิ้นนี้เพื่อค้นหาวิธีการนำเอาคุณสมบัติของ Wiimote มาประยุกต์ใช้กับเครื่องคอมพิวเตอร์พีซี เพื่อเพิ่มทางเลือกในการควบคุมเกมให้แก่ผู้เล่น

สำหรับส่วนแสดงผลกราฟิกของเกมในโครงงานชิ้นนี้นำเอา DirectX มาใช้ในการแสดงผลผ่านหน้าจอ ส่วนเกมที่นำเสนอเป็นเกมในรูปแบบ Action-RPG ที่นำคุณสมบัติของ Wiimote ข้างต้นมาประยุกต์ใช้ เพื่อให้ผู้เล่นเกิดความเพลิดเพลินและสร้างอรรถรสในการเล่นเกมมากยิ่งขึ้น รวมไปถึงเป็นแนวทางให้ผู้สนใจศึกษาการนำเอาคุณสมบัติของ Wiimote มาประยุกต์ใช้ต่อไป

1.2 วัตถุประสงค์ของงานวิจัย

1) เพื่อพัฒนาเกมคอมพิวเตอร์พีซีที่รองรับคุณสมบัติของ Wiimote ได้
2) มีความรู้ความเข้าใจเกี่ยวกับการใช้งาน DirectX API สำหรับสร้างภาพกราฟิก
3) เพื่อเป็นแนวทางแก่ผู้สนใจในการประยุกต์ใช้ Wiimote ในการควบคุมเกม
4) เพื่อเป็นแนวทางแก่ผู้ที่สนใจการพัฒนาและออกแบบเกม 3มิติ

1.3 ทฤษฎีหรือแนวความคิดที่ใช้ในการศึกษา

คณะผู้จัดทำโครงงานได้นำเอาทฤษฎีเกี่ยวกับคอมพิวเตอร์กราฟิกมาประยุกต์ใช้ในการศึกษาการทำงานของ DirectX เพื่อสร้างภาพกราฟิกสำหรับเกม และนำเอาเทคนิคต่างๆ มาประยุกต์ใช้เพิ่มเติมเพื่อให้ได้ภาพกราฟิกที่สวยงามตามต้องการ ซึ่งในที่นี้คือการนำแผนที่เงาแบบคำนวณค่าความคลาดเคลื่อนเอง (Variance Shadow Map) เทคนิคการคัดสรรวัตถุผ่านมุมกล้อง (View Culling) มาใช้กับฉาก 3 มิติเพื่อลดการใช้ทรัพยากรในการสร้างภาพกราฟิกลง และใช้ทฤษฎีเกี่ยวกับการติดต่อสื่อสารระหว่างอุปกรณ์ฮาร์ดแวร์กับคอมพิวเตอร์แบบไร้สาย มาใช้ในส่วนของการประยุกต์ใช้ Wiimote

1.4 ขอบเขตของงานวิจัย

1) การสร้างโปรแกรมเกมด้วยภาษา C++ โดยใช้ DirectX ในการแสดงผลภาพกราฟิก

2) การประยุกต์ใช้ WiiAPI ในโปรแกรมเกมสำหรับการติดต่อระหว่าง Wiimote กับคอมพิวเตอร์

3) พัฒนาโปรแกรมเกมที่สนับสนุนข้อมูลนำเข้าจาก Wiimote และแสดงผลด้วยความเร็วที่เพียงพอต่อการเล่น ในระดับที่ไม่ก่อให้เกิดปัญหาในการควบคุมของผู้เล่น

4) นำเนื้อเรื่องและระบบเกมที่ออกแบบเอาไว้มาใช้ในโปรแกรมเกม โดยสามารถเล่นได้จนจบเนื้อเรื่อง และไม่มีปัญหาทางด้านระบบเกมจนไม่สามารถเล่นจนจบได้

1.5 ขั้นตอนในการดำเนินงาน

1) กำหนดเขต จุดประสงค์ ความต้องการของโครงงาน

2) วางแผนขั้นตอนการทำงาน

3) ศึกษาหลักการเขียนโปรแกรม ได้แก่ภาษา C++ และ ภาษา HLSL ของ DirectX

4) ศึกษาทฤษฎีที่เกี่ยวข้องกับการสร้างภาพกราฟิก และเทคนิคเพิ่มเติม

5) ศึกษาคุณสมบัติของ Wiimote

6) ศึกษาหลักการออกแบบเกม

7) วิเคราะห์ ออกแบบระบบ และสร้างเนื้อเรื่องสำหรับเกม

8) พัฒนาและเขียนโปรแกรมตามที่ออกแบบ

9) ทดสอบโปรแกรมเพื่อหา Bug ที่อาจก่อให้เกิดปัญหาจนโปรแกรมไม่สามารถกลับมาทำงานต่อไปได้

10) จัดทำเอกสารเกี่ยวกับการพัฒนาโปรแกรมเกม และคู่มือเกม

1.6 ประโยชน์ที่คาดว่าจะได้รับ

1) ช่วยเพิ่มทักษะในด้านการเขียนโปรแกรมและความรู้เกี่ยวกับคอมพิวเตอร์กราฟิกส์

2) สามารถเรียนรู้การใช้งาน DirectX API ในการสร้างภาพกราฟิก และสามารถประยุกต์ใช้ Wiimote ในการควบคุมเกมคอมพิวเตอร์พีซี

3) สามารถพัฒนาเกมที่สร้างความเพลิดเพลินและได้อรรถรสในการเล่นเกมแก่ผู้เล่น

บทที่ 2
ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

2.1 หลักการทั่วไปของคอมพิวเตอร์กราฟิกส์ (Computer Graphics)

กราฟิก(Graphic) หมายถึง ศิลปะแขนงหนึ่งซึ่งใช้การสื่อความหมาย ผ่านการใช้สิ่งต่างๆ เช่น เส้น ภาพวาด สัญลักษณ์ ภาพถ่าย กราฟ แผนภูมิ การ์ตูน เป็นต้น เพื่อให้สามารถสื่อความหมายของข้อมูลได้ถูกต้องตรงตามที่ผู้รับสารต้องการ และคอมพิวเตอร์กราฟฟิกส์ (Computer Graphics) ก็คือการสร้าง การตกแต่งแก้ไข หรือการจัดการเกี่ยวกับภาพกราฟิกโดยใช้เครื่องคอมพิวเตอร์ในการจัดการ ตัวอย่างเช่น การสร้างภาพตามจินตนาการ การใช้ภาพกราฟิกในการนำเสนอข้อมูลต่างๆ การจำลองฉากเสมือนจริง เป็นต้น เพื่อให้สามารถสื่อความหมายได้ตรงตามที่ผู้สื่อสารต้องการและมีความน่าสนใจมากยิ่งขึ้น

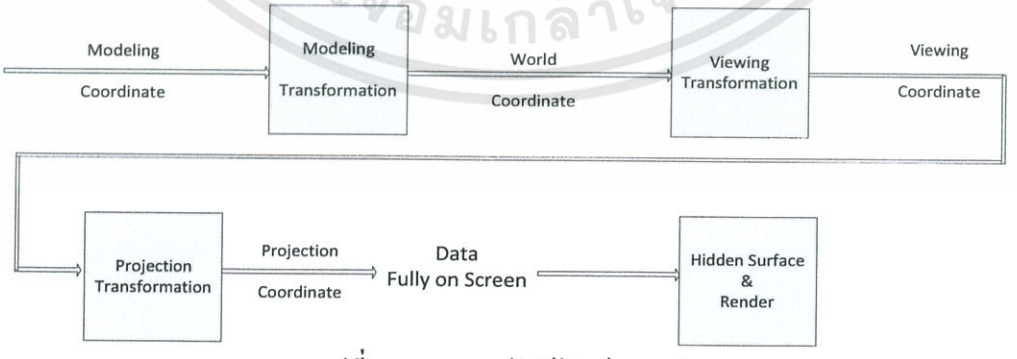
กราฟิกที่ได้จากการนำคอมพิวเตอร์กราฟิกมาใช้นั้น สามารถแบ่งได้เป็น 2 ประเภทหลักๆ ได้แก่ ภาพกราฟิกแบบ 2 มิติ (2D Graphic) ที่นำเสนอออกมาในรูปแบบภาพ 2 มิติที่สร้างจากพิกเซล (Pixel) นำมาเรียงต่อกันในแกน x และ แกน y ให้ได้ภาพออกมา และภาพกราฟิกแบบ 3 มิติ (3D Graphic) ที่นำเสนอออกมาในรูปแบบโมเดล 3 มิติ (3D Model) ในการจำลองภาพของวัตถุต่างๆ ออกมาในลักษณะสมจริงด้วยการเพิ่มแกน z ขึ้นมาให้ภาพมีความลึก

2.2 การสร้างภาพกราฟิก 3 มิติบนหน้าจอ

ในการสร้างภาพกราฟิก 3 มิติบนหน้าจอนั้นเริ่มต้นจากการนำภาพกราฟิกแบบ 3 มิติที่อยู่ในรูปของโมเดลเข้ามาในฉากที่เตรียมไว้ จากนั้นจึงนำไปผ่านไปป์ไลน์การแปลงฉาก (Transformation Pipeline) เพื่อให้หน้าจอที่สามารถแสดงผลได้เฉพาะมุมมอง 2 มิติสามารถสร้างภาพที่มีมุมมอง 3 มิติได้

2.2.1 ไปป์ไลน์การแปลงฉาก (Transformation Pipeline)

ไปป์ไลน์ในการแปลงฉาก แบ่งเป็นขั้นตอนได้ดังแสดงดังรูปที่ 2-1 มีรายละเอียดดังต่อไปนี้



รูปที่ 2-1 แผนภาพไปป์ไลน์การแปลงฉาก

- การแปลงโมเดล (Modeling Transformation) เริ่มจากข้อมูลนำเข้าเป็นพิกัดของโมเดลสามมิติ (Modeling Coordinate) ต่างๆ นำเปลี่ยนตำแหน่งในส่วนนี้ จากพิกัดที่วัตถุสร้าง เอามาไว้ในฉากเดียวกัน แล้วได้เป็นพิกัดของฉาก (World Coordinate) ที่วัตถุนั้นๆ ตั้งอยู่
- การแปลงมุมมอง(Viewing Transformation) จากพิกัดของฉากที่ได้จากขั้นตอนที่แล้วจึงนำมาแปลงให้อยู่ในพิกัดของมุมมอง (View Coordinate) บนหน้าจอซึ่งจะอยู่ในมุมมองตามประเภทของกล้อง (Camera) รายละเอียดอยู่ในหัวข้อที่ 2.2.3 โดยพิกัดเหล่านี้จะถูกเก็บเอาไว้ในเมทริกซ์ขนาด 4x4 เพื่อใช้ในการเปลี่ยนแปลงวัตถุต่อไป รายละเอียดอยู่ในหัวข้อที่ 2.2.2
- การแปลงการฉาย (Projection Transformation) เมื่อได้พิกัดของมุมมองมาแล้วจึงเข้าสู่ขั้นนี้เพื่อแปลงพิกัดที่ได้ให้เป็นวัตถุ 3 มิติที่จำลองอยู่บนหน้าจอ 2 มิติ ได้เป็นพิกัดของการฉายภาพ (Projection Coordinate) ต่อไปเพื่อนำไปแสดงบนหน้าจอ
- การซ่อนและเรนเดอร์ภาพ (Hidden Surface & Render) สำหรับการแสดงบนหน้าจอต้องผ่านกระบวนการนี้ก่อนเพื่อลบส่วนที่ควรมองไม่เห็นของโมเดลสามมิติออกไป จากนั้นจึงทำการเรนเดอร์ (Render) วัตถุเพื่อให้ได้ภาพวัตถุเป็นโมเดล 3 มิติที่ต้องการ

2.2.2 มุมกล้องของการฉายภาพ (Camera Projection)

มุมกล้องของการฉายภาพนำมาใช้ในการคำนวณค่าในขั้นตอนของการสร้างพิกัดมุมมอง สามารถแบ่งได้เป็น 2 แบบดังต่อไปนี้

- มุมกล้องเชิงตั้งฉาก (Orthogonal Projection) เป็นการแสดงฉากที่เหมาะสำหรับออกแบบฉาก ในการแสดงผลแบบนี้ วัตถุที่อยู่ในฉาก จะไม่เล็กลงตามระยะทางอย่างที่ควรจะเป็นในความเป็นจริง แต่จะมีขนาดเท่ากันหมด ไม่ว่าจะห่างจากกล้องขนาดไหน ทำให้เหมาะสำหรับออกแบบและตรวจสอบฉากที่ได้สร้างมาว่าตรงตามความต้องการหรือไม่
- มุมกล้องแบบฟรัสตัม (Frustum Projection) เป็นการแสดงฉากแบบสมจริง วัตถุที่อยู่ไกลออกไปจะมีขนาดเล็กกว่าวัตถุที่อยู่ใกล้ตัวกล้อง เราสามารถกำหนดมุมที่เราสามารถมองเห็นได้มากที่สุด 180 องศา หากมากกว่านี้จะเกิดข้อผิดพลาดในการสร้างฉากขึ้น

2.2.3 การเปลี่ยนแปลงวัตถุ (Transform)

การเปลี่ยนแปลงวัตถุเป็นการนำเอาเมทริกซ์ขนาด 4x4 ที่ได้มาหลังจากผ่านการแปลงมุมมองในไปป์ไลน์แล้วนำมาแก้ไขค่า ก่อนที่จะนำไปแสดงบนหน้าจอ สามารถแบ่งได้เป็น 3 ประเภทดังต่อนี้

2.2.2.1 การย้ายตำแหน่งวัตถุ (Translate)

เป็นการเปลี่ยนตำแหน่งของวัตถุในฉากจากที่หนึ่ง ให้ไปอยู่อีกที่หนึ่ง โดยข้อมูลที่ใช้ในการย้ายตำแหน่งวัตถุ จะเก็บอยู่ในเมทริกซ์ที่ระบุตำแหน่งในแกนสามแกน (x, y, z) มีหน่วยที่เก็บเป็นพิกเซล แล้วนำไปบวกกับพิกัดของการฉาย ทำให้วัตถุถูกย้ายตำแหน่ง

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ X & Y & Z & 1 \end{bmatrix} \quad (2\text{-}1)$$

2.2.2.2 การหมุนวัตถุ (Rotate)

วัตถุสามารถหมุนได้ตามแกนสามแกน ข้อมูลการหมุนที่เก็บในเมทริกซ์จะเป็นค่า cos และ sin ของแกนx y และ z ตามลำดับ โดยแต่ละแกนจะเก็บเป็นค่า cos 2 ตำแหน่ง sin 1 ตำแหน่ง และ –sin อีก 1 ตำแหน่ง ที่เก็บเช่นนี้เพื่อทำให้การคำนวณสร้างฉากเรียบง่ายขึ้นและลดความซับซ้อนให้หน่วยประมวลผล

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2\text{-}2)$$

ตำแหน่งที่เก็บค่าการหมุนตามแกน X

$$\begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2\text{-}3)$$

ตำแหน่งที่เก็บค่าการหมุนตามแกน Y

$$\begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2\text{-}4)$$

ตำแหน่งที่เก็บค่าการหมุนตามแกน Z

2.2.2.3 การปรับขนาดวัตถุ (Scale)

การปรับขนาดวัตถุสามารถทำได้ตามแกนสามแกนเช่นกัน โดยค่าที่เก็บในเมทริกซ์จะเป็นจำนวนเท่าของขนาดวัตถุที่มีต่อขนาดดั้งเดิม หากเป็นค่าที่อยู่ในช่วง [0,1] มากกว่า 1 ก็จะเป็นการขยายวัตถุ หาก หากเป็นค่าที่อยู่ในช่วง [0,1] ก็จะเป็นการย่อขนาดวัตถุ

$$\begin{bmatrix} X & 0 & 0 & 0 \\ 0 & Y & 0 & 0 \\ 0 & 0 & Z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2\text{-}5)$$

2.3 การคัดสรรวัตถุผ่านมุมกล้อง (View Culling)

การคัดสรรวัตถุผ่านมุมกล้อง (View Culling) เป็นเทคนิคที่ใช้กับมุมกล้องแบบฟรัสตัม ที่ใช้แสดงฉากแบบ 3 มิติ ซึ่งเป็นวัตถุหรือโมเดลที่จำเป็นต้องมีการเรนเดอร์วัตถุก่อนนำเข้ามาในฉาก เทคนิคนี้มีขึ้นเพื่อตรวจสอบว่า สิ่งที่อยู่ในฉากนั้นๆ อยู่ภายในจอแสดงผลหรือไม่ เพื่อเพิ่มประสิทธิภาพในการเรนเดอร์วัตถุที่อยู่ในฉากให้เร็วขึ้น อันเป็นผลมาจากการลดจำนวนงานในการเรนเดอร์วัตถุที่ไม่จำเป็นออกหรืออยู่นอกฉากออกไปได้

ขั้นตอนวิธีของการคัดสรรวัตถุนี้ เป็นการนำเมทริกซ์ที่ได้จากการฉายภาพมาใช้ แล้วจึงทำการสร้างเมทริกซ์จำลองมารองรับเมทริกซ์ข้างต้นเอาไว้ จากนั้นนำค่าที่ได้มาคำนวณ ซึ่งจะแบ่งได้เป็นระนาบต่างๆ ได้แก่ ด้านซ้าย ด้านขวา ด้านบน ด้านล่าง ความใกล้ และความไกล ในสมการระนาบทั่วไปแสดงดังในสมการที่ 2-6 กำหนดให้ a b c เป็นค่าของระนาบทั่วไป และ d เป็นค่าระยะห่างจากจุดกำเนิด

$$ax + by + cz + dw = 0 \quad (2\text{-}6)$$

ในทุกๆ เฟรม เมทริกซ์ที่ได้จากการฉายภาพจะทำการปรับปรุงค่าที่ได้จากกล้องเพื่อสร้างค่าขึ้นมาใหม่ให้ตรงกับตำแหน่งปัจจุบัน ใช้ในการตัดการสร้างพื้นผิวที่เคยแสดงไปแล้วออกไป จากข้อมูลข้างต้นที่ได้ นำไปทดสอบว่าวัตถุอยู่ในมุมมองหรือไม่ โดยวิธีที่จะนำมาใช้ในที่นี้คือ การทดสอบเพื่อหาขอบของทรงกลม (Bounding Sphere Test) โดยจะใช้ตำแหน่งของทรงกลมกับรัศมีที่เป็นตัวบอกพื้นที่ของวัตถุ แล้วนำมาตรวจสอบว่าวัตถุนั้นๆ อยู่ในมุมมองหรือไม่ ซึ่งหาได้จาก Dot Product ระหว่างระนาบปกติกับระยะห่างของระนาบ เมื่อเทียบกับจุดกำเนิด

จากนั้นก่อนที่จะเรนเดอร์วัตถุที่อยู่ในฉาก ให้ทำการตรวจสอบตำแหน่งของวัตถุกับเมทริกซ์จำลองก่อนว่าวัตถุนั้นอยู่ในฉากหรือไม่ ก็จะสามารถคัดสรรเฉพาะวัตถุที่อยู่ในฉากมาทำการเรนเดอร์เพียงส่วนที่ปรากฏบนหน้าจอ โดยเมทริกซ์จำลองนี้จะถูกปรับปรุงค่าใหม่ทุกครั้งที่กล้องถูกปรับปรุงค่าตามไปด้วย เพื่อให้ได้รับข้อมูลจากเมทริกซ์ที่ได้จากการฉายภาพที่เป็นปัจจุบัน

2.4 การเลือกวัตถุผ่านหน้าจอ (Picking)

เทคนิคอีกอย่างหนึ่งที่มักต้องใช้ในเกมก็คือการเลือกวัตถุผ่านหน้าจอ หรือก็คือการหยิบจับสิ่งของที่อยู่ในจอในรูปแบบ 3 มิติ ด้วย Pointer บนหน้าจอ ซึ่งมีอยู่หลายเหตุการณ์ที่ต้องการทำในเกม เป็นเทคนิคที่

เป็นการนำเอาไปป์ไลน์ของการแปลงฉากมาทำย้อนกลับ เพื่อแปลงจาก Pointer ที่อยู่บนหน้าจอเป็นตำแหน่งในรูปแบบ 2 มิติกลับเข้าไปเป็นมุมมอง 3 มิติเพื่อตรวจสอบการชนกัน (Collision) ระหว่าง Pointer และวัตถุภายในฉาก

2.4.1 การสร้างลำแสง (Ray)

ลำแสงในที่นี้ก็คือ การจำลองเส้นทางของ Pointer ที่ลากพุ่งเข้าไปในฉากสามมิติขึ้นมา ในขั้นตอนแรกนี้เป็นการแปลงจากตำแหน่งในมุมมอง 2มิติ ของPointer ให้เป็นมุมมอง 3มิติ โดยตำแหน่งของ Pointer จากหน้าจอแสดงผล ซึ่งคำนวณจากจุด (0,0) สำหรับระบบปฏิบัติการ Windows จะอยู่ที่ตำแหน่งล่างขวาของจอภาพ จากนั้นจึงนำมาปรับค่าให้เป็นตำแหน่งที่ใช้ในหน้าจอเกม จากนั้นนำตำแหน่งที่ได้ไปสร้างลำแสงโดยใช้ตำแหน่งที่ได้ไปสร้าง

2.4.2 ตรวจสอบการตัดกันด้วยลำแสง

เมื่อได้ลำแสงมาแล้ว ขั้นตอนต่อไปก็คือทำการแปลงลำแสงที่ได้เข้าสู่ขั้นตอนการแปลงพิกัดของวัตถุ เพื่อหาว่าลำแสงพุ่งไปตกกระทบที่โมเดล 3มิติ ชิ้นใดๆ บนหน้าจอหรือไม่ แล้วนำผลที่ได้ซึ่งเป็นค่าแบบ Boolean คือ True ลำแสงชนกับวัตถุ และ False คือไม่ชนกับวัตถุ จากนั้นนำค่าที่ได้ไปประกอบกับเหตุการณ์ที่จะเกิดขึ้นภายในเกมต่อไป

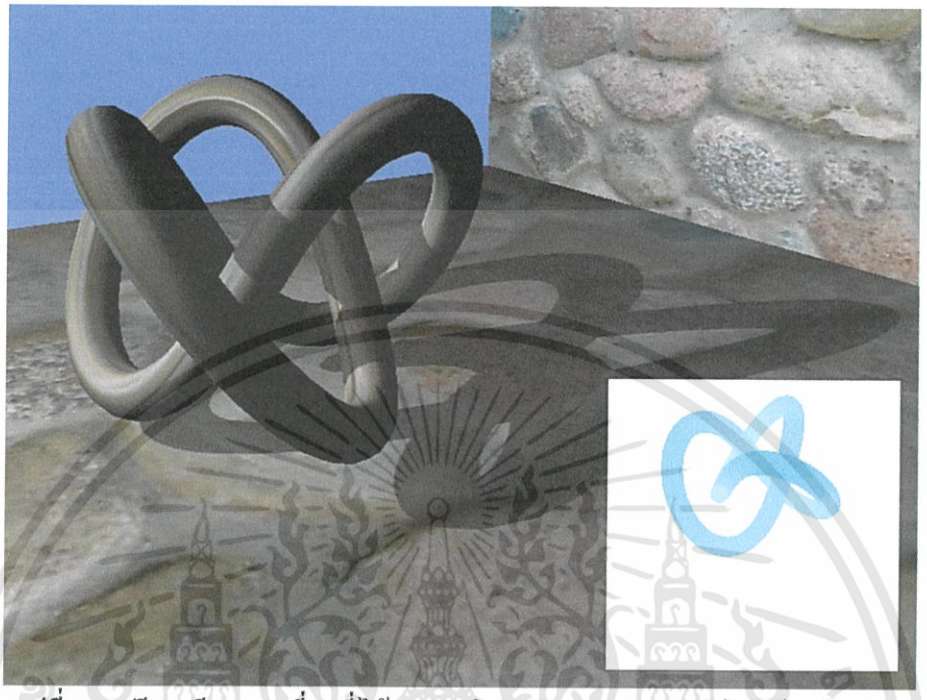
2.5 แผนที่เงา (Shadow Map)

เทคนิคในการสร้างเงานั้น สามารถแบ่งได้ออกเป็น 2 เทคนิคตามรูปแบบการคำนวณได้แก่ การสร้างเงาด้วยปริมาตรเงา (Shadow Volume) และการสร้างเงาด้วยแผนที่เงา (Shadow Map) ซึ่งแผนที่เงานั้นมีข้อดีเหนือการสร้างเงาด้วยปริมาตรเงา คือความซับซ้อนของวัตถุจะมีผลต่อความเร็วในการประมวลผลน้อยกว่า ซึ่งความเร็วในการประมวลผลนั้นเป็นสิ่งสำคัญที่ต้องมีในการแสดงภาพเคลื่อนไหว โดยเฉพาะเกม ที่ต้องรักษาระดับเพื่อรักษาอัตราการแสดงภาพเคลื่อนไหวในหนึ่งวินาที หรือ เฟรมเรท (Frame Rate) ให้อยู่ในระดับที่เพียงพอต่อความต้องการในระดับที่ผู้เล่นไม่รู้สึกว่าภาพไม่ราบรื่นหรือกระตุก ซึ่งจะส่งผลโดยตรงต่อการบังคับควบคุมตัวละคร เป็นปัญหาสำคัญที่ต้องแก้ไขให้ได้

2.5.1 ทฤษฎีทั่วไปของการสร้างเงาด้วยแผนที่เงา

การสร้างเงาโดยใช้แผนที่เงา จะใช้วิธีเก็บระยะห่างระหว่างวัตถุกับแหล่งกำเนิดแสงไว้ในตัวแปรหนึ่ง แล้วนำมาเทียบกับตำแหน่งต่าง ของวัตถุที่อยู่ในฉาก หากวัตถุที่วาดอยู่มีระยะห่างจากแสงมากกว่าค่าที่เก็บไว้ ก็แสดงว่าวัตถุนั้นอยู่ใต้เงา และจะวาดเงาทับลงไปที่ส่วนนั้น

การนำแผนที่เงาไปประยุกต์ใช้นั้น จะต้องทำการวาดฉากด้วยกัน 2 ขั้นตอน ขั้นตอนที่ 1 สำหรับการกรองแผนที่เงา และขั้นตอนที่ 2 คือการนำแผนที่เงาที่ได้มาสร้างฉากพร้อมเงา



รูปที่ 2-2 เปรียบเทียบแผนที่เงาที่ได้จากแหล่งกำเนิดแสงและการนำไปใช้วาดในฉาก

การกรองแผนที่เงาในขั้นตอนที่ 1 ทำโดยการวาดฉากจากมุมกล้องที่ตั้งไว้ในตำแหน่งของจุดกำเนิดแสง (Light Source) ลงมาตามทิศทางของแสงที่ส่องออกมา จากนั้นแทนที่การวาดสีและแสงตามปกติ ด้วยการให้ใส่ค่าแกน Z ของวัตถุที่พบ นำมาเก็บไว้ในตัวแปรค่าหนึ่ง ค่าแกน Z นั้นจะมีที่มาจากของวัตถุที่ผ่านไปป์ไลน์ในการแสดงฉากมาแล้ว ตำแหน่งได้มาจึงเป็นตำแหน่งที่อยู่บนหน้าจอ เพราะฉะนั้น แกน Z จึงเป็นแกนที่พุ่งลึกเข้าไปในหน้าจอ เท่ากับว่าเป็นแกนที่พุ่งออกมาจากมุมกล้อง ซึ่งในที่นี้ คือพุ่งออกมาจากของจุดกำเนิดแสงนั่นเอง ดังนั้นจึงสามารถนำค่าแกน Z ของตำแหน่งนั้นๆ มาแทนระยะห่างระหว่างแสงกับวัตถุได้ แล้วจึงเก็บฉากนี้ไว้ในภาพๆ หนึ่ง แล้วส่งไปคำนวณในฉากจริงในขั้นตอนที่ 2 ต่อไป

ขั้นตอนที่ 2 จะเป็นการวาดฉากพร้อมให้เงาที่ได้มาจากขั้นตอนแรก โดยมุมกล้องจะอยู่ในตำแหน่งที่ต้องการจะฉายภาพ โดยใช้การคำนวณตำแหน่งที่วัตถุควรจะอยู่หากมุมกล้องอยู่ที่ตำแหน่งของแสงด้วย เพราะจะได้เลือกตำแหน่งในภาพที่จะเอามาเปรียบเทียบได้ถูก เมื่อได้ค่าแล้วจึงนำเอาค่า Z ของวัตถุจากขั้นตอนที่แล้ว มาเทียบกับค่าสีที่เก็บไว้ในภาพ หากค่า Z ใหม่นั้นมีค่ามากกว่า แสดงว่าอยู่หลังวัตถุอื่น ทำให้สรุปได้ว่าควรจะมีเงาทับวัตถุนี้ และวาดเงาลงไปที่จุดนั้น ก็จะได้ฉากที่มีเงาเรียบร้อยแล้วออกมา

2.5.2 ปัญหาในการสร้างเงาด้วยแผนที่เงา

แม้ว่าการสร้างด้วยแผนที่เงาจะมีข้อดีที่ใช้การประมวลผลน้อยกว่า แต่ก็ยังมีปัญหาที่ยังต้องคำนึงถึงตามมาอยู่ 2 กรณีหลักๆ ได้แก่ การฉายแผนที่เงาบนฉากที่มีพื้นที่ขนาดใหญ่ (Magnification) และการฉายเงาที่จุดเดียวกันซ้ำกันจากการที่มีแหล่งกำเนิดแสงมากกว่า 1 จุดในฉากเดียวกัน (Minification) ซึ่งส่งผลให้เกิดการใช้เนื้อที่ในการประมวลผลมากจนส่งผลต่อเฟรมเรทไปด้วย นอกจากนี้แล้วในการคำนวณค่าแกน Z เพื่อให้ได้มาซึ่งแผนที่เงานั้นคิดจากการที่ฮาร์ดแวร์ นำค่าความลึกนั้นมาทำการเฉลี่ย จึงได้ค่าไม่แม่นยำเท่าที่ควร จึงมีเทคนิคอื่นๆ เพิ่มเติมเพื่อแก้ปัญหาในจุดนี้ หนึ่งในนั้นก็คือ แผนที่เงาแบบคำนวณค่าความคลาดเคลื่อนเอง (Variance Shadow Map)

2.6 แผนที่เงาแบบคำนวณค่าความคลาดเคลื่อนเอง (Variance Shadow Map)

การคำนวณค่าความคลาดเคลื่อนด้วยเทคนิคนี้ เป็นการแก้ไขโดยการนำค่าความลึกมาขั้นตอนวิธีในการคำนวณค่าตามลำดับ จนได้เป็นค่าความลึกที่แท้จริง นำไปสู่การได้เงาที่สวยสมจริงยิ่งขึ้น

หลักการหลักๆ ของการคำนวณค่าความคาดเคลื่อนนั้นคล้ายกับการสร้างแผนที่เงาแบบทั่วไป ยกเว้นว่าแทนที่จะเขียนค่าความลึกไปในแผนที่เงา จะเขียนค่าความลึกและค่าความลึกยกกำลังสองลงไปเก็บในสองตัวแปรของแผนที่เงา แล้วนำมากรองผ่านพื้นที่หนึ่งๆ

การคำนวณค่า Moment M_1 และ M_2 สามารถหาได้จากสมการที่ 2-7 และ 2-8 ตามลำดับ

$$M_1 = E(x) = \int_{-\infty}^{\infty} xp(x)dx \tag{2-7}$$

$$M_2 = E(x^2) = \int_{-\infty}^{\infty} x^2 p(x)dx \tag{2-8}$$

จากนั้นทำการคำนวณค่าเฉลี่ย m และ s^2 ด้วยสมการที่ 2-9 และ 2-10 ตามลำดับ

$$\mu = E(x) = M_1 \tag{2-9}$$

$$\sigma^2 = E(x^2) - E(x)^2 = M_2 - M_1^2 \tag{2-10}$$

เมื่อได้ค่าความเปลี่ยนแปลงมาแล้ว จึงนำมาคำนวณหาค่าสูงสุดบนความน่าจะเป็นที่พื้นที่ที่ถูกคำนวณอยู่ในขณะนั้น ที่ความลึก t จะถูกบังอยู่หลังเงา ด้วยอสมการของ Chebyshev

$$P(x \geq t) \leq p_{\max}(t) = \frac{\sigma^2}{\sigma^2 + (t-\mu)^2} \tag{2-11}$$

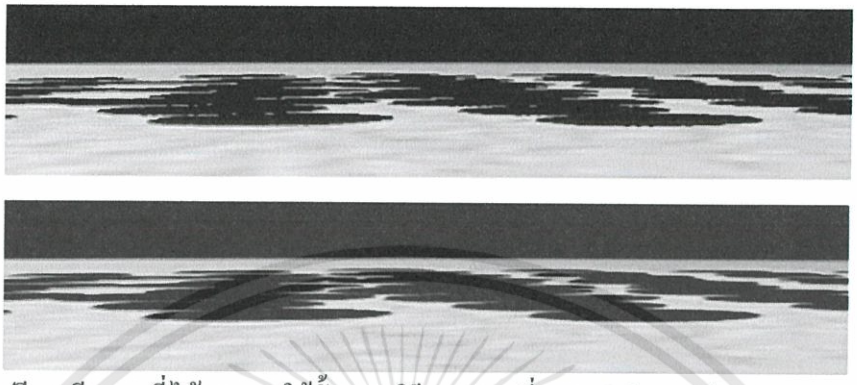
อสมการของ Chebyshev ดังแสดงในอสมการที่ 2-11 นี้จะถูกต้องเมื่อ $t > m$ ถ้า $t \leq m$ p_{max} จะเป็น 1 และฉากได้รับแสงเต็มที่ จากนั้นค่าที่ได้จะเป็นค่าขอบบน (Upper Bound) จากการกรองแบบร้อยละของความใกล้ สำหรับพื้นที่ส่วนหนึ่ง ซึ่งจะมีจุดเด่นที่ทำให้สามารถหลักเลี่ยงการวาดเงาทับซ้อน ในจุดที่ไม่ควรมีเงาได้ด้วย

นอกจากนี้ยังมีผู้ค้นคว้าเพิ่มเติมและพบว่า อสมการนี้ให้ค่าเปรียบเสมือนการให้แสงบนฉากที่มีแหล่งกำเนิดแสงเดียว กับวัตถุชิ้นเดียวในฉากนั้นๆ ซึ่งทำให้ค่า p_{max} ที่คำนวณจากบริเวณนี้มีความถูกต้องสูงตามไปด้วยเมื่อเทียบกับความเป็นจริง ดังนั้นค่า p_{max} ที่ได้จากอสมการนี้งสามารถนำไปใช้ได้ ตัวอย่างที่ 2-1 เป็นการนำไปใช้ในภาษา HLSL (High Level Shader Language) ซึ่งเป็นภาษาที่นำไปใช้กับ DirectX API ดังที่จะกล่าวในส่วนถัดไป

ตัวอย่างที่ 2-1 การคำนวณหาค่า p_{max} ในการสร้างเงา

```
float ChebyshevUpperBound(float2 Moments, float t)
{
        // ค่าความคลาดเคลื่อนถูกต้องเมื่อ t > Moments.x
        float p = (t <= Moments.x);
        //คำนวณค่าความคลาดเคลื่อน
        float Variance = Moments.y – (Moments.x*Moments.x);
        Variance = max(Variance, g_MinVariance);
        // หาค่าขอบบนโดยประมาณ
         float d = t – Moments.x;
         float p_max = Variance / (Variance + d*d);
          return max(p, p_max);
}
float ShadowContribution(float2 LightTexCoord, float DistanceToLight)
{
        //อ่านค่าที่ได้จากแผนที่เงา
        float2 Moments = texShadow.Sample(ShadowSampler, LightTexCoord).xy;
        // คำนวณหาค่าขอบบนด้วยอสมการChebyshev
        return ChebyshevUpperBound(Moments, DistanceToLight);
}
```

นอกเหนือจากแล้วยังมีผู้ค้นคว้าเพิ่มเติม โดยการนำวิธีการเบลอแผนที่เงาก่อนจะนำมาใช้ เพิ่มเติมจากการกรองแบบกรองครั้งเดียว ซึ่งวิธีนี้เทียบเท่ากับการเลือกค่าของแผนที่เงาโดยรอบมาช่วยในการคำนวณด้วย จึงทำให้ใช้ทรัพยากรลดลง และช่วยซ่อนปัญหาที่เกิดจากการฉายแผนที่เงาบนฉากที่มีพื้นที่ขนาดใหญ่ได้อีกด้วย



รูปที่ 2-3 เปรียบเทียบผลที่ได้จากการใช้ขั้นตอนวิธีของแผนที่เงาแบบคำนวณค่าความคลาดเคลื่อนเอง

2.6.1 การแก้ไขปัญหาค่าความเอียงเอน (Bias)

วัตถุที่มีระยะความลึกในแต่ละจุดต่างกันมากจะเป็นปัญหาสำหรับขั้นตอนวิธีในการสร้างแผนที่เงาทั่วไป ซึ่งแผนที่เงาแบบคำนวณค่าความคลาดเคลื่อนเองนี้ได้มีวิธีการแก้ปัญหาโดยการหาค่าจากพิกเซลด้วย Moment ตัวที่ 2 (M_2)

ในขั้นตอนวิธีนี้จะแทนค่าความลึกเปรียบดังมีทั้งพื้นผิวมีค่าความลึก m เพียงค่าเดียว แทนที่การให้ทั้งแผนที่เงามีค่าความลึก m ต่างกันไปในแต่ละพื้นที่ของแผนที่เงา ดังแสดงในสมการที่ 2-12

$$f(x,y) = \mu + x\frac{\partial f}{\partial x} + \frac{\partial f}{\partial y} \tag{2-12}$$

จากนั้นจึงคำนวณค่า M_2 โดยใช้สมการที่ 2-13

$$M_2 = E(f^2) = E(\mu^2) + E(x^2)\left[\frac{\partial f}{\partial x}\right]^2 + E(y^2)\left[\frac{\partial f}{\partial y}\right]^2 \tag{2-13}$$

จากนั้นนำไปแทนค่าพิกเซลด้วยการกระจายแบบ Gaussian ดังสมการที่ 2-14 และ 2-15 ตามลำดับ

$$E(\mu^2) = \mu^2 \tag{2-14}$$

$$E(x^2) = E(y^2) = \sigma^2 = \left(\frac{1}{2}\right)^2 = \frac{1}{4} \tag{2-15}$$

ดังนั้นจะได้ค่า M_2 ดังสมการที่ 2-16

$$M_2 = \mu^2 + \frac{1}{4}\left(\left[\frac{\partial f}{\partial x}\right]^2 + \left[\frac{\partial f}{\partial y}\right]^2\right) \quad (2\text{-}16)$$

จะเห็นได้ว่าขั้นตอนวิธีนี้ช่วยลดค่าที่ใช้คำนวณเหลือเพียงแค่ค่าความลึกยกกำลังสองเมื่ออนุพันธ์ย่อยของฟังก์ชันหลายตัวแปร (Partial Derivative) เป็น 0 ทั้งคู่ หรือก็คือเมื่อพื้นผิวขนานกับเส้นแสง สังเกตอีกทั้งยังไม่มีปัญหาเพิ่มขึ้นอีกหลังจากเพิ่มความกว้างของการกรองเพราะค่าความแปรปรวนจะถูกคำนวณจากทั่วพื้นที่ไปแล้ว สำหรับการนำไปประยุกต์ใช้กับภาษา HLSL แสดงดังตัวอย่างที่ 2-2

ตัวอย่างที่ 2-2 การคำนวณหาค่า p_{max} ในการสร้างเงา

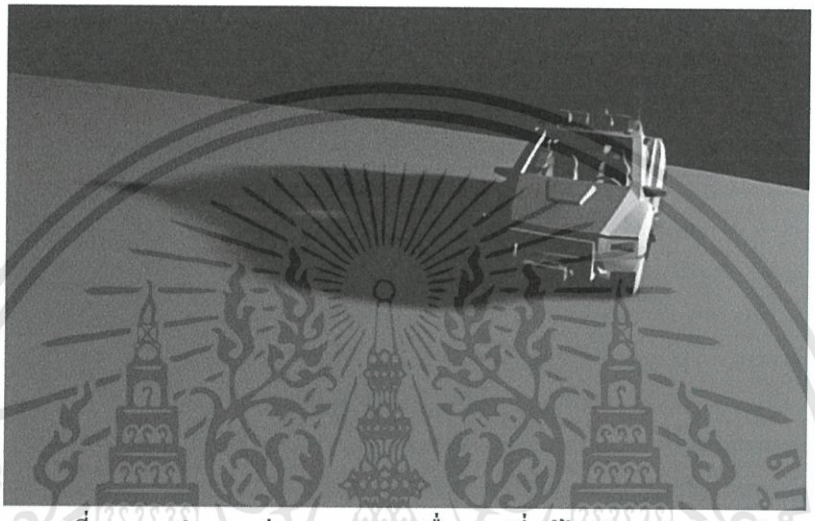
```
float2 ComputeMoments(float Depth)
{
        float2 Moments;
        // ค่า Moment ตัวแรกก็คือความลึกเอง
        Moments.x = Depth;
        // คำนวณหาอนุพันธ์ย่อยของฟังก์ชันหลายตัวแปรของค่าความลึก
        float dx = ddx(Depth);
        float dy = ddy(Depth);
        //// หาค่าจากพิกเซลให้กับ Moment ตัวที่ 2
        Moments.y = Depth*Depth + 0.25*(dx*dx + dy*dy);
        return Moments;
}
```

ปัญหาสำคัญที่ยังคงเหลืออยู่ก็คือ ค่าความผิดผลาดของตัวเลข แต่จะสามารถจัดการกับปัญหานี้ได้ง่ายๆ โดยการจำกัดค่าต่ำสุดของค่าความแปรปรวนให้เป็นค่าที่น้อยมากๆ ก่อนที่จะนำมาคำนวณหาค่า p_{max} จากในตัวอย่างที่ 2-1 เนื่องด้วยค่านี้เป็นอิสระจากรูปทรงในฉากอยู่แล้ว ทำให้มีการตั้งค่าใหม่เพียงครั้งเดียวเท่านั้นก็แก้ปัญหาข้างต้นได้แล้ว

นอกเหนือจากนี้ควรจะจำกัดค่าอนุพันธ์ย่อยของฟังก์ชันหลายตัวแปรของ M_2 เพื่อเลี่ยงกรณีค่าความแปรปรวนที่จะสูงมาก หากวัตถุทำมุมเกือบขนานกับทิศทางของแสง ซึ่งจะเริ่มเกิดความไม่เสถียรของ

ค่า แล้วก่อให้เกิดปัญหาพิกเซลกระพริบถูกแสดงแบบกระพริบขึ้นมาแทนในตำแหน่งที่ควรจะเป็นเงาทั้งหมดตามมา

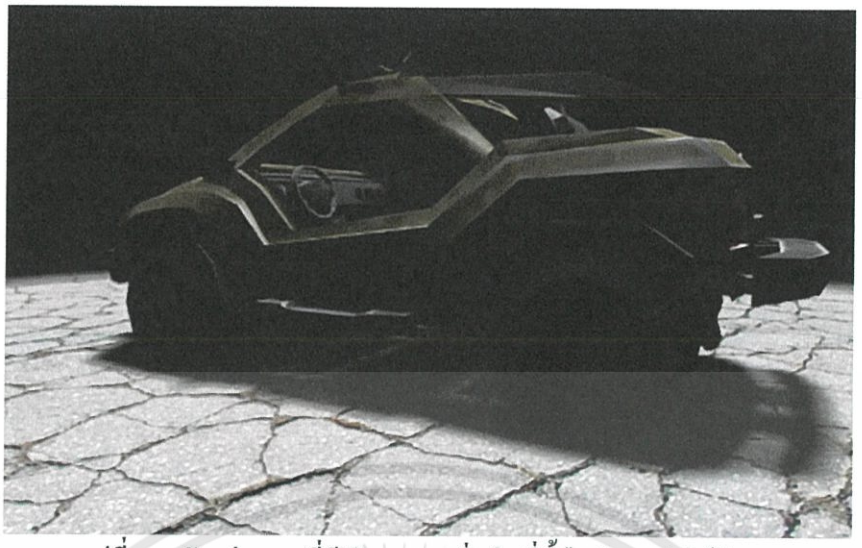
สำหรับในหลายๆ ฉากจะสามารถเลี่ยงความไม่เสถียรนี้ได้ โดยตัดการใช้ค่าความคลาดเคลื่อนจากอนุพันธ์ย่อยของฟังก์ชันถ้าแผนที่เงามีความละเอียดพอ แล้วทำการจำกัดค่าความแปรปรวน ก็อาจจะเพียงพอแล้วต่อการหลีกเลี่ยงสิว (Acne) ซึ่งเป็นปัญหาที่จะเกิดขึ้นบนฉากได้ โดยรูปที่ 2-4 แสดงภาพเงาที่ได้หลังจากแก้ไขปัญหาค่าความเอียงเอนนี้เรียบร้อยแล้ว



รูปที่ 2-4 แผนที่เงาแบบคำนวณค่าความคลาดเคลื่อนเองที่แก้ไขปัญหาค่าความเอียงเอนแล้ว

2.6.2 ปัญหาแสงรั่ว (Light Bleeding)

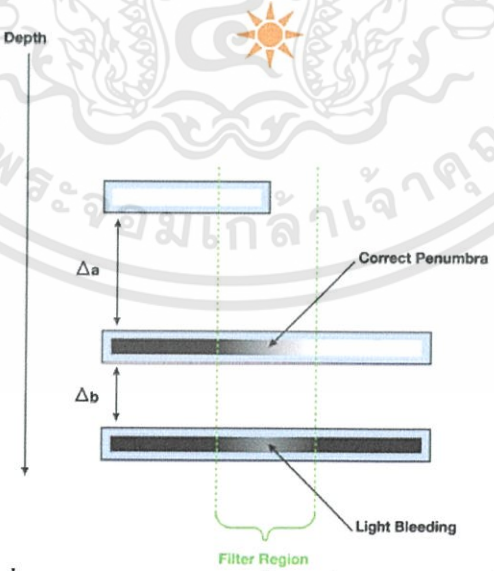
ปัญหาแสงรั่วคือ การที่มีแสงปรากฏในจุดที่ควรจะเต็มไปด้วยเงา ซึ่งเป็นปัญหาที่เกิดขึ้นในเงาที่มีคุณลักษณะแบบ Soft-Edge หรือก็คือมีเงาที่มีรูปแบบจางๆ ในส่วนที่ใกล้เคียงระหว่างแสงกับเงาที่ 2-5 แสดงฉากที่มีปัญหาแสงรั่ว และแผนภาพในรูปที่ 2-6 แสดงจุดที่ก่อให้เกิดปัญหาแสงรั่ว



รูปที่ 2-5 ตัวอย่างฉากที่มีปัญหาแสงรั่ว ในที่นี้คือตำแหน่งใต้ตัวรถ

ปัญหานี้ก็เกิดขึ้นกับแผนที่เงาที่ใช้เทคนิคอื่นเช่นกัน เพราะเกิดจากลักษณะการกระจายของวัตถุที่อยู่ในฉากบางแบบ และถึงแม้ว่าจะมีวิธีแก้ปัญหาโดยการไม่เลือกค่าจำนวนวัตถุตายตัว แต่การกำหนดแบบนี้เป็นไปไม่ได้เพราะจำนวนวัตถุย่อมแต่ต่างกันไปในแต่ละฉากอยู่แล้ว ขั้นตอนวิธีที่ดีคือการคำนวณหาเงาวัตถุทีละชิ้นจนหมดฉาก ปัญหานี้จึงเป็นสิ่งที่แสดงให้เห็นว่าขั้นตอนวิธีการสร้างแผนที่เงานั้นจำเป็นต้องมีการปรับตัวได้เรื่อย ๆ

สำหรับแผนที่เงาแบบคำนวณค่าความคลาดเคลื่อนเองนี้ จะมีวิธีการแก้ปัญหาแสงรั่วโดยจำเป็นต้องเสียค่าความแม่นยำที่คำนวณได้ออกไปเล็กน้อยด้วยการปรับค่า p_{max} ซึ่งจะทำให้จุดที่เป็นเงาแบบมัว จะมีความมัวมากขึ้น รายละเอียดแสดงในหัวข้อถัดไป



รูปที่ 2-6 แผนภาพแสดงจุดที่ทำให้เกิดปัญหาแสงรั่ว

2.6.3 การแก้ไขปัญหาแสงรั่ว

แม้ว่าปัญหาแสงรั่วนี้จะสามารถแก้ไขได้ด้วยการคำนวณซ้ำอีก แต่ก็จะทำให้ขั้นตอนวิธีทั้งหมดข้างต้นที่ผ่านมาใช้ไม่ได้ทันที จึงมีขั้นตอนวิธีที่จะใช้ในการแก้ปัญหาแสงรั่วโดยประมาณขึ้นมาใช้ทดแทนวิธีการคำนวณซ้ำ โดยมีหลักการดังนี้

สำหรับพื้นผิวที่มีความลึกเท่ากับ t จะถูกเงาเข้าไปเติมเต็มด้วยความลึกเฉลี่ยประมาณ m ดังนั้นจะได้ว่า $t > m$ ทำให้ $(t-m)^2 > 0$ จากอสมการของ Chebyshev ที่ค่า $p_{max} < 1$ อันก่อให้เกิดพื้นที่ที่มีเงามัวผิดธรรมชาติขึ้นมา ซึ่งจะเป็นบริเวณที่ก่อให้เกิดปัญหาแสงรั่ว

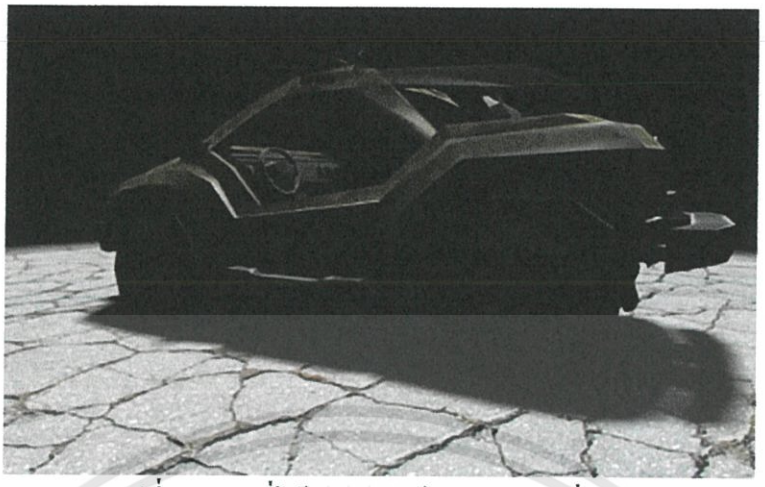
การแก้ปัญหาบริเวณนี้ทำได้โดนการ โดยแก้ p_{max} ให้ค่าทุกๆ ค่าที่น้อยกว่าค่าความเข้มต่ำสุดกลายเป็น 0 ไป และค่าที่เหลือจะถูกปรับใหม่ให้อยู่ในระยะ 0 ถึง 1 โดยฟังก์ชั่นนี้สามารถเขียนด้วยภาษา HLSL ดังแสดงในตัวอย่างที่ 2-3

ตัวอย่างที่ 2-3 การนำวิธีแก้ปัญหาแสงรั่วไปใช้

```
float linstep(float min, float max, float v)
{
    return clamp((v – min) / (max – min), 0, 1);
}
float ReduceLightBleeding(float p_max, float Amount)
{
    // ทำการปรับค่าใหม่
    return linstep(Amount, 1, p_max);
}
```

โดยค่าตัวแปร Amount สามารถเป็นค่าที่ใช้กำหนดระดับความรุนแรงในการลดแสงรั่ว ซึ่งฉากที่มีแสงรั่วมากจะต้องใช้ค่าที่สูงตาม ไม่ได้ขึ้นอยู่กับขนาดของฉาก แต่ขึ้นอยู่กับอัตราส่วนของความลึกระหว่างเงาในจุดนั้นและพื้นผิวที่รับเงาในฉาก ซึ่งค่านี้ควรตั้งให้เหมาะสม เพราะค่าที่สูงขึ้นก็เท่ากับว่าลดความละเอียดของเงาลงไปด้วย เพราะฉะนั้นควรตั้งให้สูงเพียงพอที่จะกำจัดแสงรั่วที่เห็นชัดเจนไปได้เพื่อคงรายรายละเอียดในจุดที่ดีอยู่แล้วไว้ให้มากที่สุด

ในทางปฏิบัติแล้ว ขั้นตอนวิธีนี้แทบไม่ใช้ทรัพยากรเพิ่มขึ้นเลย การลดแสงรั่วจึงเป็นขั้นตอนวิธีที่น่าสนใจและน่านำไปใช้กับโปรแกรมที่ต้องการแสดงภาพที่มีรายละเอียดสูง และยังสามารถใช้ได้ในหลายๆ ฉากอีกด้วย โดยรูปที่ 2-7 แสดงภาพที่ได้หลังการแก้ไขปัญหาแสงรั่วแล้ว



รูปที่ 2-7 ภาพที่ได้หลังจากแก้ไขปัญหาแสงรั่วแล้ว

2.6.4 การแก้ไขปัญหาความเสถียรของตัวเลข

ปัญหาสุดท้ายที่คงเหลืออยู่ก็คือปัญหาความเสถียรของตัวเลข เนื่องจากการคำนวณค่าความแปรปรวนนั้นทำให้ได้เลขที่ไม่เสถียรขึ้นตามมาในกรณีที่นำไปใช้กับค่าที่มีค่ามาก การแก้ไขปัญหานี้ทำได้โดยการสร้างแผนที่เงาขนาด 32 บิทมาใช้ หรือก็คือการนำเอาเมทริกซ์ค่าความลึกเชิงเส้น (Linear Depth Matrix) มาใช้แทนค่า z ในการสร้างแผนที่เงาทั่วไป เพราะค่าจะสูญเสียความแม่นยำเพิ่มขึ้น แปรผันตรงตามระยะวัตถุที่อยู่ไกลแสงมากขึ้นอันเป็นเหตุให้ค่าความแปรปรวนไม่เสถียรมากขึ้นตามไปด้วย การนำเมทริกซ์ค่าความลึกนี้มาใช้แทนจึงมีความแม่นยำกว่า

ตัวอย่างที่ 2-4 การใช้เมทริกซ์ค่าความลึกเชิงเส้นกับสปอตไลท์

```
DepthPSIn Depth_VS(DepthVSIn In)
{
        DepthPSIn Out;
        Out.Position = mul(float4(In.Position, 1), g_WorldViewProjMatrix);
        Out.PosView  = mul(float4(In.Position, 1), g_WorldViewMatrix);
        return Out;
}
```

```
float4 Depth_PS(DepthPSIn In) : SV_Target
{
        float DistToLight = length(In.PosView);
        return ComputeMoments(DistToLight);
}
//Renderและให้เงาจากตำแหน่งมุมกล้องที่กำหนด
float4 Shading_PS(ShadingPSIn In) : SV_Target
{
        // คำนวณระยะห่างจากแสงจนถึงจุดที่วัตถุอยู่
        float SurfaceDistToLight = length(g_LightPosition – In.PosWorld);
        . . .
}
```

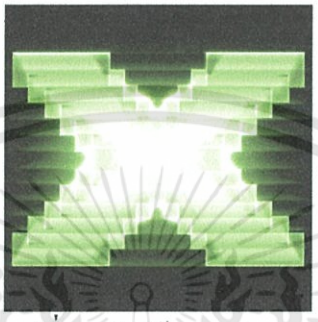
ตัวอย่างที่ 1.4 เป็นการประยุกต์ใช้เมทริกซ์ค่าความลึกกับแสงสปอตไลท์ ซึ่งจะเห็นได้ว่าค่าจะสามารถปรับให้อยู่ในช่วง [0,1] ได้ง่าย หากแผนที่เงาที่อยู่ในตำแหน่งเดิมถูกใช้ แก้ปัญหาความเสถียรของตัวเลขไปได้

2.7 DirectX คืออะไร?

DirectX API เป็น API ที่พัฒนาโดย Microsoft สำหรับจัดการงานเกี่ยวกับ Multimedia โดยเฉพาะกับการสร้างเกมและวิดีโอ ประกอบไปด้วยหลายส่วนประกอบสำหรับจัดการด้านต่างๆ หลายส่วน โดยใช้ชื่อที่ขึ้นต้นด้วย Direct และตามด้วยหน้าที่ของส่วนประกอบนั้นๆ ได้แก่

- DirectDraw สำหรับการวาดภาพ
- Direct3D (D3D) สำหรับการสร้างกราฟิกแบบ 3มิติ
- DXGI (DirectX Graphics Infrastructure) สำหรับจัดการทรัพยากรเกี่ยวกับกราฟิกต่าง ๆ
- Direct2D for 2D Graphics สำหรับการสร้างกราฟิกแบบ 2มิติ
- DirectWrite สำหรับจัดการกับ font ของตัวอักษร
- DirectCompute สำหรับจัดการการคำนวณด้วย GPU
- DirectInput สำหรับจัดการนำเข้าข้อมูลรูปแบบต่างๆ ไม่ว่าจะเป็นเม้าส์ คีย์บอร์ด จอยสติก หรือตัวคอนโทรลเลอร์ของเกมใด ๆ
- DirectPlay สำหรับการติดต่อสื่อสารผ่านอินเทอร์เน็ต
- DirectSound สำหรับเล่นและบันทึกเสียงในรูปแบบคลื่นสั้นๆ หรือ Sound Effect ต่าง ๆ

- DirectMusic สำหรับเล่นเพลงบรรเลงเพลง ซึ่งรองรับไฟล์สกุล .wav และ .midi
- DirectX Media สำหรับจัดการการเล่นไฟล์ Multimedia ต่าง ๆ
- DirectX Diagnostics (DxDiag) สำหรับใช้เป็นเครื่องมือสร้างรายงานต่างๆ ที่เกี่ยวข้องกับ DirectX เพื่อแก้ไขปัญหาที่พบ
- DirectX Media Objects สำหรับรองรับวัตถุต่างๆ เช่น encoder decoder และเอฟเฟคต่าง ๆ
- DirectSetup สำหรับช่วยในการติดตั้ง DirectX กับเครื่องคอมพิวเตอร์



รูปที่ 2-8 โลโก้ของ DirectX

สำหรับโครงงานชิ้นนี้จะเน้นไปที่การประยุกต์ใช้ Direct3D สำหรับการแสดงภาพ 3มิติ มีรายละเอียดดังต่อไปนี้

2.7.1 Direct3D คืออะไร?

Direct3D เป็นส่วนหนึ่งของส่วนติดต่อกับโปรแกรมประยุกต์ของ DirectX ใช้ได้ในระบบปฏิบัติการ Windows 95 หรือใหม่กว่าเท่านั้น นอกจากนี้ Direct3D ยังเป็นส่วนติดต่อโปรแกรมประยุกต์ของหน่วยประมวลผลภาพใน Xbox และ Xbox360 อีกด้วย Direct3D ถูกใช้เพื่อสร้างฉากสามมิติในโปรแกรมประยุกต์ที่ต้องการประสิทธิภาพในการทำงานสูง

2.7.2 คุณสมบัติของ Direct3D

Direct3D ทำให้โปรแกรมแสดงผลทั้งหน้าจอแทนที่จะอยู่ในหน้าต่างหนึ่งได้ แต่ก็ยังสามารถแสดงผลผ่านหน้าต่างเดียวได้ตามการตั้งค่าที่ต้องการ นอกจากนี้ยังสามารถเรียกใช้ฮาร์ดแวร์มาช่วยประมวลผลได้ด้วยหากฮาร์ดแวร์ที่มีอยู่สามารถทำได้ ซึ่งความสามารถของฮาร์ดแวร์ในประมวลผลกราฟิกมีหลายประการ เช่น Z-Buffering Spatial Anti-Aliasing Alpha Blending, Mipmapping เพื่อสร้างบรรยากาศให้กับฉาก และการวางภาพ Texture อย่างถูกต้อง ผ่านชุดคำสั่งหลากหลายสำหรับการสร้างฉากสามมิติ

ตั้งแต่ DirectX หลังเวอร์ชั่น 8 เป็นต้นไป Direct3D ได้รวมเอา DirectDraw เข้ามาใช้สร้างฉาก 2มิติด้วย โดยจะมีการทำงานขึ้นอยู่กับฮาร์ดแวร์ ทำให้สิ่งที่ฮาร์ดแวร์ทำไม่ได้ Direct3D ก็จะไม่สามารถทำได้ เช่น ฮาร์ดแวร์ที่ไม่สนับสนุน Pixel Shader Direct3D ก็จะไม่สนับสนุนเช่นกัน ถึงแม้ว่าจะยังแสดงผลได้ แต่

คุณภาพจะลดลงไปมาก ด้วยเหตุนี้ Direct3D จึงมีความสามารถในการจำลองฮาร์ดแวร์ประมวลผลกราฟิกขึ้นมาทดแทนเองได้ แต่จะประสิทธิภาพจะน้อยกว่าและไม่สามารถใช้ได้ในโปรแกรมกราฟิกสามมิติทั่วไป

นอกจากนี้ Direct3D ได้มีการพัฒนานำเอาภาษา HLSL (High Level Shading Language) เพิ่มเติมขึ้นมาใน DirectX เวอร์ชั่น 8 เป็นภาษาสำหรับควบคุมฮาร์ดแวร์ที่เกี่ยวกับการแสดงผลกราฟิกโดยตรง สำหรับ DirectX 9 ที่นำมาประยุกต์ใช้ในโครงงานนี้เป็นเวอร์ชั่นที่มีการปรับปรุงภาษา HLSL ล่าสุดที่ใช้ในปัจจุบันเรียบร้อยแล้ว

2.8 ข้อมูลทั่วไปเกี่ยวกับ Wii Remote

Wii Remote หรือบางครั้งเรียกว่า Wiimote เป็นอุปกรณ์ควบคุมสำหรับเครื่องเล่นเกม Wii ของ Nintendo ซึ่งได้เปิดตัวครั้งแรกในงาน Tokyo Game Show ที่จัดขึ้นในวันที่ 14 กันยายน พ.ศ.2548 และเปิดตัวอย่างเป็นทางการในนาม Wii Remote ในวันที่ 27 เมษายน 2549 เป็นอุปกรณ์นำเข้าข้อมูล (Input) ที่ใช้การติดต่อสื่อสารแบบไร้สายด้วย Bluetooth HID Protocol

(ก) (ข) (ค)

รูปที่ 2-9 Wii Controller ในมุมต่างๆ (ก) เมื่อมองจากด้านล่าง (ข) เมื่อมองจากด้านบน (ค) เมื่อมองจากด้านข้าง

Wiimote จัดอยู่ในอุปกรณ์สำหรับควบคุมประเภทที่มีการตัวตรวจจับความเคลื่อนไหว (Motion Detection) ซึ่งเป็นจุดเด่นที่สำคัญของ Wiimote มีแหล่งพลังงานที่ต้องการคือถ่านขนาด AA จำนวน 2 ก้อน ภายในมีหน่วยความจำ 16.3 kilobytes EEPROM chip มีลำโพงติดตั้งอยู่ภายใน 1 ตัว พร้อมมอเตอร์สำหรับสร้างการสั่น นอกจากนี้ยังมีอุปกรณ์เสริมอื่นๆ ที่สามารถเชื่อมต่อกับ Wiimote ได้เช่น Nunchuck และ Balance Board เป็นต้น

2.8.1 คุณสมบัติของ Wiimote

- ปุ่มกด (Buttons) Wiimote มีคุณลักษณะแบบเดียวกับจอยสติกนั่นก็คือปุ่มสำหรับป้อนข้อมูลง่ายๆ โดย Wiimote มีปุ่มทั้งหมด 11 ปุ่มด้วยกัน ได้แก่ปุ่ม D-Pad 4 ปุ่ม (ขึ้น ลง ซ้าย ขวา) ปุ่ม A ใต้ D-Pad ปุ่ม+ ปุ่ม – ปุ่ม Home ปุ่ม1 ปุ่ม2 ดังรูปที่ 2-9b และปุ่ม B อยู่ด้านหลังของ Wiimote ดังรูปที่ 2-9a

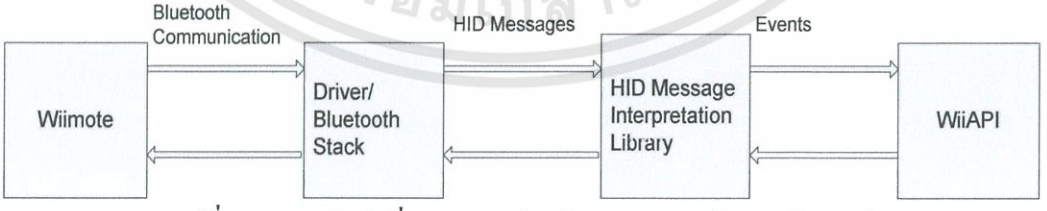
- เซนเซอร์ (Sensor) สามารถตรวจจับ Infrared ได้ ซึ่งมีอุปกรณ์สำหรับส่งแสงอินฟราเรด (Infrared) โดยอุปกรณ์ที่มีมาพร้อมกับเครื่องเล่นเกม Wii ก็คือ IR Sensor Bar ประกอบด้วยหลอดสำหรับเปล่งแสงอินฟราเรด (Infrared LEDs) 10 หลอด แบ่งเป็นด้านซ้าย 5 หลอด และขวา 5 หลอด ความยาวประมาณ 20 เซนติเมตร ให้ Wiimote สามารถประมวลผลตำแหน่งเมื่อเทียบกับหน้าจอได้ในระยะประมาณ 5 เมตร ซึ่งค่าที่ได้จากแสงอินฟราเรดจะเป็นตำแหน่งแกน x y เพื่อนำไปคำนวณตำแหน่งของ Wiimote กับ หน้าจอที่แสดงผลเกม

- ตัววัดอัตราเร่ง (Accelerometer) เป็นอุปกรณ์ที่ติดตั้งอยู่ภายใน ใช้สำหรับจับการเคลื่อนไหวทางแนวขวางได้ด้วยความแม่นยำที่ 10% ระหว่างความเร่งที่ -3g ถึง +3g อีกทั้งตัววัดความเร่งนี้ก็ยังทำหน้าที่เป็นตัววัดองศา (Inclinometer) ได้อีกด้วยในกรณีที่ไม่มีการเคลื่อนไหวเกิดขึ้น

รูปที่ 2-10 IR Sensor bar สำหรับเปล่งแสงอินฟราเรด

2.9 การติดต่อสื่อสารระหว่าง Wii Remote กับคอมพิวเตอร์

วิธีการที่ใช้ในการติดต่อกับคอมพิวเตอร์ ประยุกต์มาวิธีการที่เครื่องเกม Wii ใช้ในการติดต่อและสื่อสารกับ Wii Controller ตัวหนึ่งๆ หรือมากกว่านั้นด้วยการใช้ WiiAPI ซึ่งตัว API นี้จะประกอบไปด้วยฟังก์ชั่นต่างๆ ของ Wiimote ซึ่งโดยมีวิธีการติดต่อสื่อสารดังรูปที่ 2-11



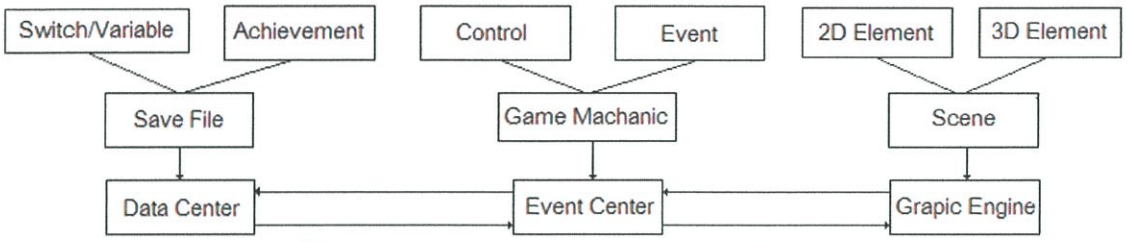
รูปที่ 2-11 การติดต่อสื่อสารระหว่างตัว Controller กับคอมพิวเตอร์

โดยทั่วไปแล้วอาจแบ่งการสื่อสารด้วย Wiimote ได้ด้วย 2 ช่องทางได้แก่ การติดต่อด้วย Bluetooth โดย Wiimote จะทำการส่งสัญญานเมื่อผู้ใช้ Wiimote กดปุ่มใดๆ ของ Wiimote พร้อมๆ กับการข้อมูลองศาที่ได้มาจากตัววัดองศา (Inclinometer) จากนั้นผ่านเข้าไปใน Bluetooth Stack เพื่อแปลงเป็น HID Messages ซึ่งเป็นข้อความสำหรับประมวลผล และนำไปแปลงใน HID Message Interpretation Library ให้ได้เป็นคำสั่งหรือเหตุการณ์ (Event) ที่จะไปเลือกใช้ฟังก์ชั่นที่อยู่ใน WiiAPI และสร้าง Feedback กลับไปที่ Wiimote เช่น ให้ส่งเสียง ทำให้สั่น เป็นต้น ซึ่งช่องทางนี้อาจเปรียบได้กับการสื่อสารกับคอมพิวเตอร์ด้วยแป้นพิมพ์ หรือจอยสติก อีกช่องทางหนึ่งก็คือการติดต่อผ่านแสงอินฟราเรด ซึ่งการที่จะได้มาซึ่งข้อมูลของตำแหน่ง Wiimote กับหน้าจอแสดงผลนั้นต้องใช้แสงอินปราเรดจาก IR Sensor bar ที่ติดตั้งเอาไว้ส่งไปยัง Wiimote อีกทีหนึ่ง แล้วจึงได้ค่าเป็นตำแหน่งของ Wiimote แล้วจึงส่งข้อมูลนี้ไปยัง WiiAPI ได้ ซึ่งช่องทางที่สองนี้จะมีจุดประสงค์คล้ายกับเป็น Pointer บนหน้าจอผ่านเม้าส์ แต่สามารถส่งได้ในระยะห่างระดับหนึ่ง ทำให้ผู้เล่นสามารถออกท่าทางเคลื่อนไหวได้มากขึ้น

สำหรับข้อมูลได้ที่จาก Wiimote เมื่อเชื่อมต่อกับคอมพิวเตอร์แล้วจะประกอบไปด้วย

- Battery Level ระดับพลังงานของถ่าน ให้ค่าระหว่าง 0-100 แทนจำนวนร้อยละของพลังงานคงเหลือ
- LEDs ตำแหน่งและสถานะเปิด/ปิดของหลอด LED ที่ฉายแสงทั้ง 4 ช่องที่ติดตั้งอยู่บนส่วนปลายของ Wiimote
- Rumble สถานะการสั่นของ Wiimote ให้ค่าเป็นเปิดหรือปิด
- ID รหัสของ Wiimote ซึ่ง Wiimote แต่ละตัวจะมีรหัสเป็นของตัวเอง
- Buttons สถานะของปุ่มบน Wiimote ให้ค่าเป็นชื่อปุ่มและสถานะว่าถูกกดหรือไม่
- Accel มุมของ Wiimote ปัจจุบัน ซึ่งจะมีการ Update อยู่ตลอดเวลา และจะมีค่าคงที่เมื่อ Wiimote อยู่กับที่ ให้ค่าเป็นมุม X Y Z
- Orient ความเปลี่ยนแปลงของมุม หากอยู่กับที่จะมีค่าเป็น 0 ให้ค่าเป็นมุม X Y Z
- IR ตำแหน่งของแสงอินฟราเรดที่ได้รับ ให้ค่าเป็นตำแหน่งในแกน X Y Z หากไม่พบจะให้ค่าเป็น 0
- Speakers สถานะและระดับเสียงที่เปล่งออกมาจาก Speaker ของ Wiimote
- Extension ข้อมูลอุปกรณ์เสริมอื่นๆ ที่ติดตั้งคู่กับ Wiimote ซึ่งในโครงงานนี้จะใช้ Nunchuck โดยจะได้ข้อมูลเป็น Accel ค่ามุม X Y Z Button สถานะของปุ่ม C Z และมุมแกนโยกของ Nuncuck เป็นค่าของแกน X Y

2.10 โครงสร้างและการทำงานของโปรแกรมเกม



รูปที่ 2-12 แผนผังการติดต่อระหว่างโปรแกรมทั้ง 3 ส่วน

2.10.1 ส่วนแสดงผลผ่านหน้าจอ (Graphic Engine)

แบ่งได้เป็น 2 ส่วนได้แก่ ส่วนแสดงผลภาพ 3 มิติ ซึ่งใช้ในการแสดงฉากภายในเกมและการออกท่าทางต่างๆ ของตัวละครในรูปแบบ 3 มิติ ซึ่งในส่วนนี้เป็นส่วนที่นำทฤษฎีต่างๆ เข้ามาใช้ในการแสดงผลเพื่อให้ได้ภาพที่สวยงามตามที่ต้องการ และแสดงผลได้ราบรื่น ส่วนถัดมาคือส่วนแสดงภาพ 2 มิติ ซึ่งจะแสดงซ้อนอยู่บนฉาก 2 มิติ โดยส่วนนี้จะเกี่ยวข้องกับ Interface สำหรับติดต่อกับผู้ใช้ ซึ่งออกแบบเป็นปุ่มตัวเลือกให้ผู้เล่นเข้าถึงและจัดการได้ง่าย

ในการทำงานของส่วนแสดงผล จะทำการเรียกไฟล์ภาพและวัตถุตามแต่ละชนิดเข้ามาผ่านส่วนควบคุมนี้ โดยมีโปรแกรมอ่าน Path ของไฟล์ภาพที่ใช้และคุณสมบัติที่จะให้แสดงในฉาก จากไฟล์ที่กำหนดเอาไว้ แล้วนำมาประมวลผลในส่วนโปรแกรมที่เป็น Engine ของเกมเพื่อทำการเรนเดอร์ให้ได้มาซึ่งฉากๆ หนึ่ง และแสดงภาพออกมา

ส่วนของการแสดงผลแบบ 3มิติ จำเป็นต้องใช้เทคนิคต่างๆ เข้ามาสนับสนุน เนื่องด้วยส่วนนี้เป็นส่วนที่ใช้ทรัพยากรของเครื่องมากที่สุดจึงจำเป็นต้องมีเทคนิคมารองรับ

เทคนิคต่างๆ ที่นำมาใช้ในการแสดงผลและรักษาระดับ Frame Rate ได้แก่

- View Culling ช่วยให้การวาดฉาก 3มิติ เป็นไปได้ราบรื่นขึ้นด้วยการวาดฉากและวัตถุภายในเกมเท่าที่พบอยู่ในมุมกล้อง ตัดการเรนเดอร์วัตถุและฉากในส่วนที่ไม่จำเป็นออกไป ลดการใช้ทรัพยากรในการแสดงผลลง รายละเอียดดังแสดงในหัวข้อที่ 2.4
- Variance Shadow Mapping ในการสร้างเงาให้กับตัวละคร วัตถุต่างๆ ในฉาก 3 มิติของเกม รายละเอียดดังแสดงในหัวข้อที่ 2.6
- Instancing ช่วยในส่วนของการแสดงผลของวัตถุภายในฉาก 3มิติ โดยการให้โปรแกรมเรนเดอร์วัตถุภายในฉากที่มีลักษณะเหมือนกันแต่มีหลายชิ้น เช่นหญ้า ต้นไม้ หิน เป็นต้น เพียงครั้งเดียว

2.10.2 ส่วนควบคุมเหตุการณ์ (Event Center)

ในเกมนั้นทุกอย่างล้วนเปลี่ยนแปลงไปตามเหตุการณ์ที่ส่งเข้ามาในเกมผ่านผู้เล่นทั้งหมด ในส่วนนี้จะเป็นส่วนกลางที่ทำให้เกิดการติดต่อกับผู้เล่น ทำหน้าที่ในการรับคำสั่งจากผู้เล่นเข้ามาผ่านส่วนของ Control และนำมาประมวลผลร่วมกับ Event เพื่อส่งคำสั่งไปยังระบบเกม (Game Mechanic) แล้วส่วนควบคุมเหตุการณ์จะติดต่อกับส่วนแสดงผลบนหน้าจอให้เปลี่ยนแปลงวัตถุต่างๆ ในฉาก สร้างเอฟเฟคบนหน้าจอ กำหนดข้อความโต้ตอบกับผู้เล่น สร้างเหตุการณ์ให้ผู้เล่นตัดสินใจ ให้ผู้เล่นได้ปรับเปลี่ยนข้อมูลที่เกี่ยวกับเกมต่างๆ รวมไปถึงรองรับระบบเกมต่างๆ ที่จะแสดงในหัวข้อรายละเอียดการออกแบบตัวเกม

สำหรับการควบคุมด้วย Wiimote นั้น ได้นำไลบรารี่ (Library) ชื่อว่า WiiYourself! ซึ่งถูกออกแบบมารองรับกับภาษา C++ ซึ่งเป็นภาษาที่ใช้ในการสร้างเกมของโครงงานนี้ มาประยุกต์ใช้ในร่วมอยู่ในส่วน Control ให้โปรแกรมเกมรองรับการสื่อสารกับ Wiimote ได้นอกเหนือจากข้อมูลนำเข้าทั่วไปที่ได้จากเมาส์และคีย์บอร์ด

2.10.3 ส่วนควบคุมข้อมูล (Data Center)

เป็นส่วนสำหรับจัดการข้อมูลต่างๆ ที่เปลี่ยนแปลงไปภายในเกม ความคืบหน้าของเกมต่างๆ ที่ผู้เล่นได้ทำลงไป ในส่วนนี้จะเป็นส่วนที่ออกแบบขึ้นมาเพื่อรองรับการกลับมาเล่นต่อจากจุดเดิม โปรแกรมจะนำข้อมูลส่วนนี้มาปรับเปลี่ยนคุณสมบัติต่างๆ ของเหตุการณ์มาสร้างเป็นไฟล์เซฟ สำหรับนำมาใช้อ่านข้อมูลต่างๆ ให้กลับมาสู่สถานะเดิมแล้วนำไปประมวลผลใน ส่วนควบคุมเหตุการณ์ให้ปรับเปลี่ยนสถานะต่างๆ ภายในเกมให้เป็นดังที่บันทึกไว้เพื่อให้ผู้เล่นดำเนินเนื้อเรื่องของเกมต่อไป

นอกจากนี้ยังใช้เป็นส่วนที่ช่วยในการพัฒนาโปรแกรม โดยสามารถปรับเปลี่ยนข้อมูลต่างๆ เพื่อให้ส่วนควบคุมเหตุการณ์นำไปแสดงผลในจุดที่ต้องการจะแก้ไขได้สะดวกขึ้น รวมไปถึงการทดสอบโปรแกรมให้เป็นไปตามรายละเอียดที่ออกแบบเอาไว้ต่อไป

บทที่ 3
วิธีการดำเนินงานวิจัย

3.1 รายละเอียดการออกแบบเกี่ยวกับตัวเกม

3.1.1 ภาพรวมของเกม

เกมที่พัฒนานี้ใช้ชื่อว่า “Reovano” เป็นเกมแนว Action-RPG โดยตัวเอกหรือผู้เล่นจะรับบทเป็นคนในคณะสำรวจที่ได้เดินทางมายัง “ทวีปใหม่” ร่วมกับกองกำลังทหารจากเมืองหลวงที่ห่างออกไป แต่ได้เกิดเหตุการณ์ไม่คาดฝันเมื่อเดินทางมาถึงทวีปใหม่ มีพายุรุนแรงซัดสาดเข้ามาถล่มเรือจนอับปาง เมื่อตัวเอกตื่นขึ้นมาก็พบว่า ตนเองได้กลายร่างเป็นมอนสเตอร์ที่เรียกว่า “สไลม์(Slime)” สิ่งมีชีวิตประหลาดลักษณะคล้ายเยลลี่สีเขียวโปร่งใส ไม่มีแขนไม่มีขาแต่กลับใช้อาวุธได้อย่างคล่องแคล่ว พร้อมกับคู่หูหญิงของเขาที่กลายเป็น “ค้างคาว” ประหลาดตัวสีม่วง ผู้ใช้เวทมนตร์สนับสนุนแก่ตัวเอก ส่วนกองกำลังและคณะสำรวจที่เหลือก็หายสาบสูญไป ทั้งสองจึงไม่มีทางเลือก ต้องออกเดินทาง จนพบกับเมือง “รีโอวาโน” ซึ่งเป็นเมืองของเหล่ามอนสเตอร์ และได้ออกสำรวจ เรียนรู้เรื่องราวความเป็นมาของเมือง สร้างความสัมพันธ์กับชาวเมือง ช่วยแก้ไขปัญหาความวุ่นวายที่เกิดขึ้น จนในที่สุดก็ได้พบกับหลักฐานชิ้นสำคัญในการกลับสู่ร่างเดิม

เนื่องด้วยตัวเกมถูกออกแบบมาเพื่อให้เกิดการโต้ตอบผ่าน Wiimote เกมนี้จึงเน้นจุดเด่นไปที่การเลือกใช้อาวุธของตัวเอก ให้ผู้เล่นได้ออกท่าทางแตกต่างกันไปตามแต่ชนิดอาวุธ และความถนัดของผู้เล่นอย่างอิสระ ไปตามเนื้อเรื่องที่ออกแบบเอาไว้ให้ผู้เล่นได้แก้ปริศนาต่างๆ ตลอดเกม

3.1.2 เนื้อเรื่องภายในเกม

ภายในเกม จะแบ่งเนื้อเรื่องเป็นตอนย่อย ๆด้วยกัน 7 ตอน โดยแต่ละตอนจะมีการดำเนินเนื้อเรื่องเน้นไปที่ฉากหนึ่งๆ โดยมีเหตุการณ์สำคัญต่างๆ ของแต่ละตอนดังต่อไปนี้

3.1.2.1 ตอนที่ 1: ชีวิตใหม่

- หลังจากตัวเอกและคู่หูตื่นขึ้นมา และพบว่าตัวเองกลายร่างเป็นสไลม์ไปแล้ว ผู้เล่นจะได้เข้าสู่ส่วนของการฝึกการควบคุมพื้นฐานของเกม และระบบเครื่องแต่งกาย
- ต่อมาในขณะที่ทั้งสองกำลังสับสนว่าควรจะทำอย่างไรต่อไปดี ก็ถูกสิ่งมีชีวิตประหลาดลักษณะคล้ายกิ้งก่ายักษ์ถือหอกมาพบตัวเข้าและถูกทำร้าย ในส่วนนี้ผู้เล่นก็จะได้เข้าสู่ส่วนฝึกและทำความรู้จักกับระบบสถานะเบื้องต้นภายในเกม และจบลงด้วยการได้รับชัยชนะครั้งแรก
- จากเหตุการณ์นี้ทำให้กิ้งก่ายักษ์ตัดสินใจหนีเข้าไปในป่า เหลือทิ้งไว้เป็นรอยเท้าให้เราได้สะกดรอยตามต่อไป

3.1.2.2 ตอนที่ 2: เส้นทางสู่รีโอวาโน

- หลังจากเหตุการณ์ในตอนแรก ผู้เล่นจะได้เริ่มออกเดินทางในป่า และได้รู้จักกับระบบไอเท็มเบื้องต้น และสู้กับศัตรูที่ปรากฏขึ้นมาเป็นระยะๆ ไปตามทางเดินในป่า จนเมื่อเดินทางได้ระยะหนึ่งรอยเท้าก็ขาดรอยไป
- ขณะที่ทั้งสองกำลังพยายามหารอยเท้าต่อ ก็ได้ยินเสียงประหลาดๆ เหมือนมีใครถูกทำร้ายอยู่ข้างหน้า และเมื่อตามรอยเสียงต่อไป ก็ได้พบกับกิ้งก่ายักษ์ตัวเดิมที่เคยเข้าทำร้ายทั้งสอง ได้รับบาดเจ็บที่ขาจนเดินต่อไม่ได้ ทั้งสองจึงเข้าช่วยเหลือหาทางปฐมพยาบาลให้จนเดินได้อีกครั้ง
- เมื่อเราช่วยเหลือกิ้งก่ายักษ์สำเร็จ เจ้ากิ้งก่าจึงอาสาตอบแทนด้วยการนำทางเราไปที่เมืองที่มันอาศัยอยู่ และสุดทางเราก็จะได้พบกับเมืองรีโอวาโน แต่ด้วยความเหนื่อยล้าจากการเดินทาง ทั้งสองจึงสลบไป

3.1.2.3 ตอนที่ 3: ความวุ่นวายที่ร้านอาหาร

- เมื่อทั้งสองตื่นขึ้นมาก็พบว่าตัวเองพักอยู่ในห้องนอนของบ้านหลังหนึ่งในเมือง และได้พบกับสไลม์สีชมพูที่คอยดูแลพวกเขาเข้ามาถามไถ่อาการบาดเจ็บ และแนะนำเกี่ยวกับการจัดเก็บไอเท็มกับการพักผ่อน จากนั้นก่อนจากไปเธอจึงเพิ่งได้แนะนำตัวว่าเธอชื่อวิคกี้ทำงานอยู่ที่ร้านอาหารของเมืองไม่ไกลจากบ้านหลังนี้นัก
- จากนั้นทั้งสองจึงได้เริ่มสำรวจบ้านพัก ให้ผู้เล่นได้บังคับตัวละคร สำรวจ และทบทวนระบบพื้นฐานต่างๆ ผ่านหนังสือที่จัดเก็บไว้ภายในบ้าน จากนั้นจึงออกจากบ้าน
- ในจุดนี้ ผู้เล่นจะได้บังคับตัวละครอย่างอิสระ ได้สำรวจเมือง และรู้จักกับสถานที่ต่างๆ ภายในเมืองผ่านการพูดคุยกับเหล่ามอนสเตอร์ทั้งหลายที่อาศัยอยู่ในเมือง จนเมื่อเวลาผ่านไปจนเกือบเย็นค้างคาวคู่หูของเราจึงเสนอให้ไปแวะที่ร้านอาหารก่อนจะไปพักผ่อนที่บ้าน
- ขณะที่รับประทานอาหารอยู่นั้น ร้านอาหารก็ตกอยู่ในสถานการณ์วุ่นวาย ถูกกลุ่มก็อบลินนักเลงจากต่างถิ่นเข้ามาขโมยอาหารที่หลังครัวและกำลังจะหลบหนี ทั้งสองจึงตัดสินใจเข้าไปขวาง แต่ถูกหัวหน้าก็อบลินชนจนต้องผละหลีกทางให้มันหลบหนีไปได้สำเร็จ
- จากเหตุการณ์ความวุ่นวายทำให้ทั้งสองรู้สึกไม่พอใจอย่างมาก แต่ยังทำอะไรไม่ได้ ก่อนที่จะกลับบ้านและพักผ่อนด้วยความกังวลใจ

3.1.2.4 ตอนที่ 4: การต่อสู้ที่โรงงานร้าง

- เช้าวันรุ่งขึ้น ทั้งสองจึงตัดสินใจแวะที่ร้านอาหารเพื่อถามไถ่เกี่ยวกับก็อบลินเมื่อวานนี้ ก็ได้พบว่าพวกก็อบลินหลบหนีหายไป ไม่มีใครตามตัวเจอ และนี่เป็นครั้งที่สองแล้ว ทั้งสองจึงอาสาจะช่วยจัดการเรื่องนี้ โดยจะหาทางสะกดรอยตามหาพวกมันให้ได้

- จากนั้นทั้งสองจึงได้เริ่มออกตามหาเบาะแสจากเหล่ามอนสเตอร์ภายในเมืองเกี่ยวกับเหตุการณ์ในร้านอาหาร จนได้พบว่าพวกก็อบลินแอบหลบหนีหายไปในแถบโรงงานทางทิศตะวันออกของเมืองที่ปล่อยร้างไว้ ไม่มีใครไปคอยสอดส่องดูแลเพราะเป็นที่อยู่ของมอนสเตอร์เจ้าถิ่นที่ไม่ค่อยเป็นมิตรกับชาวเมือง
- หลังจากนั้นทั้งสองจึงได้ไปสำรวจโรงงานร้าง และได้พบกับร่องรอยอะไรบางอย่างภายใน พร้อมๆกับต้องต่อสู้กับมอนสเตอร์เจ้าถิ่นที่อาศัยอยู่ภายใน จนในที่สุดก็พบโพรงลับภายในโรงงานร้าง แต่ยังไม่ทันจะลองเข้าไปสำรวจ พวกก็อบลินก็กำลังหลบหนีเข้ามาพอดี จึงเกิดการต่อสู้กันขึ้น ทั้งสองไม่ยอมแพ้และเอาชนะได้ในที่สุด
- พวกก็อบลินได้รับบาดเจ็บจนไม่สามารถต่อสู้หลบหนีได้อีก เจ้าหัวหน้าจึงเปลี่ยนท่าที่ออดอ้อนขอให้ไว้ชีวิต จนพวกตัวเอกเผลอ พวกลูกน้องจึงหนีเข้าไปในโพรงได้สำเร็จ แต่ขณะที่หัวหน้าจะหนีตามไป โพรงก็มีเสียงระเบิดเกิดขึ้นทำให้โพรงถล่ม ทำให้หัวหน้าก็อบลินจนมุมอย่างหลีกเลี่ยงไม่ได้ และถูกจับไปลงโทษในที่สุด

3.1.2.5 ตอนที่ 5: เบาะแสชิ้นสำคัญ

- หลังจากเหตุการณ์จัดการกับกลุ่มก็อบลินต่างถิ่นจบลง ทั้งสองจึงได้รับคำยกย่องจากชาวเมือง ในวันรุ่งขึ้นจึงทำให้ทั้งสองได้รับคำร้องขอให้ช่วยเหลือจากชาวเมืองมากขึ้น ซึ่งในจุดนี้ผู้เล่นสามารถตัดสินใจจะไม่ให้ความช่วยเหลือก็ได้
- วันรุ่งขึ้นอีกวันหนึ่ง ค้างคาวคู่หูของเรามีท่าทีไม่ค่อยสบายใจ จึงตัดสินใจตามหาเบาะแสที่จะกลับคืนร่างเดิมให้ได้ และวางแผนกับตัวเอกให้ไปสอบถามกับปู่สไลม์เจ้าของร้านอาหารเกี่ยวกับมนุษย์ด้วยกัน และพบว่าเมืองนี้ก็เคยมีมนุษย์อาศัยอยู่มาก่อน โดยเฉพาะสุสานเคยเป็นที่พักอาศัยของมนุษย์มาก่อนในอดีตกาล
- จากเบาะแสที่ได้รับทั้งสองจึงไปที่สุสานและพยายามสำรวจอย่างละเอียดแต่ก็ไม่พบอะไรเลยจนตกเย็นและเริ่มท้อแท้ แต่ก็เหมือนฟ้างเส้นสุดท้าย ทั้งสองจึงตัดสินใจสำรวจต่อจนตกดึกก็พบกับคฤหาสน์ปริศนาตั้งตระหง่านอยู่กลางสุสานอย่างไม่น่าเชื่อ แต่ด้วยความเหนื่อยล้าทั้งสองจึงฟุบไปอย่างหลีกเลี่ยงไม่ได้

3.1.2.6 ตอนที่ 6: ภาพลวงตา

- ทั้งสองตื่นขึ้นในบ้านพักของตนในวันรุ่งขึ้น และรู้สึกเหมือนพบกับเหตุการณ์ย้อนกลับมาคือ สไลม์สีชมพูเข้ามาในห้องนอน พูดในที่สิ่งเคยพูดกับเราไปแล้วเหมือนเมื่อครั้งแรกที่มาถึงเมืองแล้วจากไป ทั้งสองรู้สึกแปลกใจอย่างมากแล้วคิดว่าเหตุการณ์ทุกอย่างเป็นความฝัน แต่เมื่อกำลังจะเปิดประตูห้องนอนออกไปนั้นก็มีแสงสว่างสาดส่องเข้ามาจนทั้งสองตาพร่ามัวไปหมด

- คราวนี้กลายเป็นฉากที่ชายหาด ตัวเอกอยู่คนเดียวและค้างคาวก็เข้ามาทักราวกับเหตุการณ์เรื่องอับปางเพิ่งเกิดขึ้นเมื่อครู่ ขณะที่ตัวเอกสับสนอยู่ก็ถูกค้างคาวคู่หูทำร้ายอย่างไม่คาดฝัน และจากท่าทีบางอย่าง ทำให้ตัวเอกเริ่มรู้ว่าค้างคาวตัวนี้ไม่ใช่คู่หูของตน จนในที่สุดก็เกิดการต่อสู้กันอย่างหลีกเลี่ยงไม่ได้
- การต่อสู้จบลง ฉากทั้งหมดบิดเบี้ยว และตัวเอกก็รู้สึกตัวตื่นขึ้นมา พบว่าตัวเองอยู่ที่หน้าคฤหาสน์พร้อมกับค้างคาวที่พยายามปลุกเราที่อยู่ๆ ก็หมดสติไป ตัวเอกเล่าเหตุการณ์ที่ตนประสบมาให้คู่หูฟัง จนทราบความจริงว่าตัวเอกคงถูกมนตร์บางบทเข้าเล่นงานตอนที่ตนกำลังปลดผนึกลวงตาที่ทำให้ไม่เห็นคฤหาสน์หลังนี้เข้า ค้างคาวจึงตัดสินใจพาตัวเอกกลับบ้านพักก่อน แล้วค่อยสำรวจคฤหาสน์ในวันรุ่งขึ้นเพื่อเตรียมการ

3.1.2.6 ตอนที่ 7: การเดินทางครั้งใหม่

- วันรุ่งขึ้นมาถึง ผู้เล่นจะยังคงสามารถทำเหตุการณ์อื่นๆ ได้ตามปกติก่อนเดินทางไปยังคฤหาสน์ได้
- ณ สุสาน ค้างคาวคู่หูของเราร่ายเวทย์ป้องกันมนตร์ให้ก่อนจะคลายมนตร์ที่กำบังคฤหาสน์เอาไว้ ทั้งสองจึงได้เข้าไปสำรวจคฤหาสน์ และพบกับมอนสเตอร์ดุร้ายอาศัยอยู่มากมาย ทั้งสองฝ่าฟันไปจนถึงชั้นในสุดของคฤหาสน์แล้วพบกับสไลม์ผีตัวใหญ่ยักษ์และทำการต่อสู้ด้วยจนสไลม์ยักษ์สลายตัวไป และทิ้งหนังสือปริศนาเอาไว้ให้ทั้งสองเก็บไป
- เมื่อทั้งสองกลับมาถึงบ้าน ค้างคาวจึงได้ลองแกะความหมายหนังสือเล่มนั้น และพบกับเบาะแสเกี่ยวกับการกลับสู่ร่างเดิม ทั้งสองจึงตัดสินใจบอกลาชาวเมืองและออกเดินทางต่อไป ทิ้งเอาไว้เป็นความทรงจำแก่ชาวเมือง

3.1.3 ตัวละครหลักที่ปรากฏภายในเกม

3.1.3.1 สไลม์ตัวเอก

ผู้เล่นจะสวมบทเป็นตัวเอกที่เดินทางมายังทวีปใหม่ แล้วกลายเป็นสิ่งมีชีวิตประหลาดรูปร่างกลม สีเขียวโปร่งใส ไม่มีแขนไม่มีขา แต่มีจุดเด่นที่สามารถใช้อาวุธได้อย่างคล่องแคล่ว สามารถพัฒนาฝีมือ เรียนรู้ทักษะใหม่ๆ และมีความสามารถที่เพิ่มขึ้นเรื่อย ๆ

รูปที่ 3-1 สไลม์ตัวเอก

3.1.3.2 ค้างคาวคู่หู

คู่หูของตัวเอกหรือตัวละครที่จะอยู่ช่วยเหลือเราตลอดเกม เป็นนักเวทย์หญิงนิสัยห้าว รักการสำรวจและค้นคว้าเรื่องราวปริศนา แต่ได้กลายร่างเป็นค้างคาวประหลาดสีม่วง เคราะห์ดีที่ยังสามารถใช้พลังเวทมนตร์ได้ แต่ก็ไม่กลับมาทั้งหมด จึงเป็นตัวละครที่จะค่อยๆ เติบโตไปพร้อมกับตัวเอก และคอยช่วยเหลือในการต่อสู้ รวมถึงการเดินเนื้อเรื่องตลอดเกม

รูปที่ 3-2 ค้างคาวคู่หู

3.1.3.3 กิ้งก่ายักษ์

ตัวละครหลักตัวแรกที่พบเมื่อเดินทางมาถึงทวีปใหม่ เป็นตัวละครที่มีรูปร่างลักษณะคล้ายกิ้งก่ายืนสองขาตัวสูงใหญ่ถือหอก นิสัยขี้ขลาดและค่อนข้างจะซุ่มซ่าม ทำงานเป็นทหารรักษาการอยู่ภายในเมือง ที่กำลังออกเวรมาสำรวจชายหาด จนได้พบกับตัวเอกและค้างคาว

รูปที่ 3-3 กิ้งก่ายักษ์

3.1.3.4 สไลม์สาว: วิคกี้ (Vitoria Klyne)

หนึ่งในตัวละครหลักที่เป็นสไลม์ มีสีชมพู สวมชุดสาวเสิร์ฟขนาดจิ๋ว อาศัยอยู่ในเมืองกับในฐานะลูกสาวของ “ปู่สไลม์” นิสัยดี น่ารัก และเป็นตัวละครที่จะคอยช่วยเหลือและแนะนำเราในช่วงที่เพิ่งเดินทางมาถึง

รูปที่ 3-4 สไลม์สาววิคกี้

3.1.3.5 เจ้าของร้านอาหาร: ไกลน์ (Grandpa Klyne)

เจ้าของบาร์ประจำเมืองเผ่าพันธุ์สไลม์ เป็นผู้อาวุโสรอบรู้เรื่องราวเก่าๆ ของเมืองรีโอวาโน เรียกกันโดยทั่วไปว่า “ปู่สไลม์” เป็นสไลม์สีเทา มีคิ้วและหนวดยาวปกคลุมใบหน้า ถึงแม้จะสูงอายุแล้วแต่ก็ยังคงมีฝีมือการรับลูกค้าและปรุงอาหารที่ไม่น้อยหน้าใคร เป็นตัวละครที่จะมีบทบาทเป็นคนที่ให้ข้อมูลสำคัญๆ กับเราอีกตัวละครหนึ่ง

รูปที่ 3-5 ปู่สไลม์เจ้าของร้านอาหาร

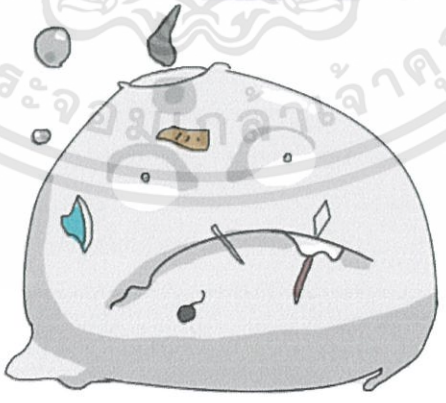
3.1.3.6 หัวหน้ากลุ่มก็อบลิน: แครงค์ (Bozz Crank)

ก็อบลิน (Goblin) เป็นสิ่งมีชีวิตรูปร่างคล้ายคนตัวเตี้ย หัวโต จมูกยาว หูแหลม ผิวสีเขียว ถนัดเรื่องการลักขโมยและหลบหนี แครงค์เป็นหัวหน้ากลุ่มก็อบลิน คาดแว่นตากลม (Goggle) เป็นสัญลักษณ์ รูปร่างใหญ่กว่าก็อบลินทั่วไป และมีระดับฝีมือสูงกว่า ชอบยิ้มยิงฟัน ฉลาดเจ้าเล่ห์

รูปที่ 3-6 ก็อบลินแครงค์

3.1.3.7 สไลม์ยักษ์เก่าแก่ (Great Old Slime)

เป็นสไลม์ผีตัวสีเทาที่สิงสถิตอยู่ในคฤหาสน์ผีสิงกลางสุสาน ภายในร่างมีสิ่งของลอยเคว้งคว้างและสามารถเอามาขว้างปาได้อย่างอิสระ จำอะไรไม่ได้นอกจากหน้าที่ของตนที่ต้องปกป้องคฤหาสน์หลังนี้จากผู้บุกรุก จนเมื่อตัวเอกและคู่หูได้บุกเข้ามาและถูกล้มได้ มันก็นึกอะไรบางอย่างออกและไปสู่สุขติ ก่อนที่จะทิ้งหนังสือที่เป็นเบาะแสสำคัญแก่ตัวเอกและคู่หูไว้



รูปที่ 3-7 สไลม์ยักษ์เก่าแก่

3.1.3.8 เผ่าพันธุ์แมว

สิ่งมีชีวิตรูปร่างคล้ายแมว เป็นประชากรที่ทำหน้าที่เป็นพ่อค้าแม่ค้าของเมืองรีโอวาโนอยู่ที่ตลาด ไม่มีบทบาทสำคัญต่อเนื้อเรื่องมากนักแต่เราจะได้ใช้บริการจากพวกเขาในการซื้อขายไอเท็ม บางครั้งก็มีข่าวลือและคำแนะนำบางอย่างให้กับตัวเอกและคู่หู

รูปที่ 3-8 เผ่าพันธุ์แมว

3.1.3.9 เพนกวิ้นไปรษณีย์ (Postguin)

โพสท์กวิ้น หรือ เพนกวิ้นโพสท์ เป็นอีกหนึ่งในเผ่าพันธุ์ที่อาศัยอยู่ในรีโอวาโน ไม่มีบทบาทสำคัญในเนื้อเรื่องเท่าใดนัก แต่เราจะได้ใช้บริการกับเจ้าโพสท์กวิ้นอยู่บ่อยๆ เนื่องจากเป็นเผ่าพันธุ์ที่ทำงานเป็นบุรุษไปรษณีย์ของเมือง จึงค่อนข้างรู้จักที่ทางในเมืองเป็นอย่างดี

รูปที่ 3-9 เพนกวิ้นไปรษณีย์

3.1.4 สถานที่หลักต่างๆ ภายในเกม

3.1.4.1 ทวีปใหม่

เรื่องราวภายในเกมจะเกิดขึ้นในส่วนตะวันออกของทวีปใหม่แห่งนี้ ซึ่งเป็นทวีปที่อดีตเคยมีการบุกเบิกสำรวจ แต่ถูกปล่อยให้ถูกลืมไป เนื่องจากเป็นทวีปที่มีมอนสเตอร์หรือสิ่งมีชีวิตประหลาดต่างๆ อาศัยอยู่มาก และมีบรรยากาศแปรปรวนรอบทวีป

รูปที่ 3-10 ส่วนของทวีปใหม่

3.1.4.2 รีโอวาโน

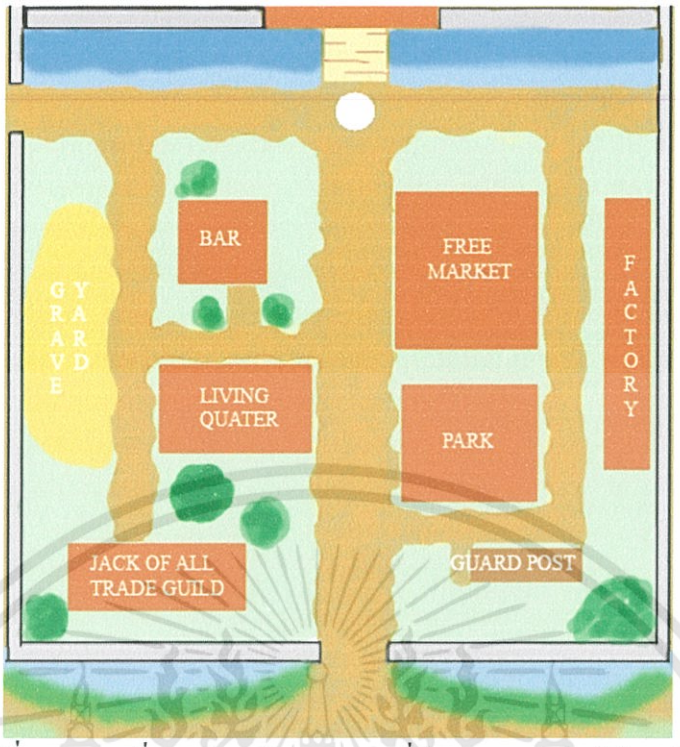
เมืองใน “ทวีปใหม่” ที่ตัวเอกได้อยู่พักอาศัย เป็นเมืองใหญ่ที่มีความสามารถในการปกครองด้วยตนเอง ตั้งอยู่ทางทิศตะวันตกของ “ทวีปใหม่” ภายในเมืองจะถูกแบ่งเป็นส่วนๆ ได้แก่ ส่วนที่พักอาศัย บาร์ ตลาดขายของ สวนสาธารณะ ป้อมทหาร โรงงานร้าง สุสาน และปราสาทดังรูปที่ 3-10 ซึ่งแต่ละสถานที่จะมีความสำคัญแตกต่างกันไปในเนื้อเรื่อง

3.1.4.3 ป่าหน้าเมืองรีโอวาโน

สุดเขตป่าทางใต้จะเป็นจุดที่เรือของคณะสำรวจอับปางลง ซึ่งถือเป็นจุดเริ่มต้นของการเดินทาง ภายในป่าจะมีเส้นทางที่ไม่ซับซ้อนมุ่งไปสู่ทางใต้ของเมืองรีโอวาโน เป็นสถานที่ที่ไม่อันตราย สำหรับให้ผู้เล่นได้คุ้นเคยกับการควบคุมและระบบเกมเบื้องต้น โดยเนื้อเรื่องในตอนที่ 2 จะอยู่ที่จุดนี้

3.1.4.4 บ้านของตัวเอก

ตั้งอยู่ในส่วนพักอาศัย (Living Quarter) กลางเมือง เป็นบ้านสองชั้นโดยชั้นบนจะเป็นที่นอน และชั้นล่างเป็นส่วนของห้องเก็บของและมุมอ่านหนังสือสำหรับทบทวนระบบเกมเบื้องต้น



รูปที่ 3-11 แผนที่และตำแหน่งของสถานที่โดยรวมของเมืองรีโอวาโน

3.1.4.5 ร้านอาหารประจำเมือง

ตั้งอยู่ทางเหนือของเมือง ถัดจากส่วนพักอาศัยและบ้านพักของตัวเอก ภายในจะแบ่งเป็นส่วนนั่งรับประทานอาหารที่ผู้เล่นจะเข้ามาพูดคุยกับคนในร้านได้ และส่วนของหน้าครัวและห้องครัวที่มีปู่สไลม์เป็นผู้ปรุงอาหาร และวิคกี้เป็นพนักงานเสิร์ฟประจำร้าน

3.1.4.6 ตลาดสด

ตั้งอยู่ทางทิศตะวันออกเฉียงเหนือของเมือง เป็นจุดศูนย์กลางการค้าภายในเมืองที่จะให้ผู้เล่นได้เข้ามาจับจ่ายใช้สอยไอเท็ม ซื้อเครื่องแต่งกายและอาวุธใหม่ๆ และรับฟังข่าวลือต่างๆ จากเหล่ามอนสเตอร์เช่น เผ่าแมว ที่ซื้อขายกันอยู่ในตลาด มีสวนสาธารณะทางใต้ของตลาดไว้เป็นที่พักผ่อนหย่อนใจของชาวเมือง

3.1.4.7 โรงงานร้าง

โรงงานร้างตั้งอยู่ชิดกับกำแพงเมืองรีโอวาโนทางตะวันตก เป็นที่พักพิงของมอนสเตอร์ที่ไม่เป็นมิตรกับชาวเมือง แต่ไม่ได้ก่อความเดือดร้อนอะไรให้ ยกเว้นจะบุกรุกล้ำเข้าไป ภายในโรงงานจะมีกลไกพื้นฐานให้ผู้เล่นได้แก้ปริศนา เป็นจุดที่เริ่มมีศัตรูระดับกลางที่มีพฤติกรรมมากขึ้น โดยเนื้อเรื่องในตอนที่ 4 จะอยู่ที่จุดนี้

3.1.4.8 สุสาน

สุสานเป็นพื้นที่ว่างทางตะวันตกของเมือง กินพื้นที่กว้างขวางไปจนสุดกำแพงเมืองทางตะวันตก ซึ่งเป็นส่วนที่จะได้รับการสำรวจอย่างละเอียดอีกครั้งเมื่อเนื้อเรื่องมาถึงในตอนที่ 5 โดยผู้เล่นจะสามารถเข้ามาที่สุสานได้เสมอ ซึ่งจะมีมอนสเตอร์บางชนิดออกมาในตอนกลางคืนให้ผู้เล่นได้ฝึกฝนฝีมือบ้าง

3.1.4.9 คฤหาสน์ผีสิง

ตั้งอยู่ใจกลางสุสานของเมือง แต่ถูกมนตร์บังเอาไว้ จึงไม่มีชาวเมืองคนไหนพบเห็นมาเป็นระยะเวลายาวนานแล้ว เป็นสถานที่ลึกลับซับซ้อน และมีกลไกประหลาดต่างๆ มากมาย เป็นสถานที่สุดท้ายที่ผู้เล่นจะได้เข้าถึงก่อนจบเกม

3.1.5 ระบบต่างๆ ภายในเกม

3.1.5.1 ระบบสถานะของตัวละคร (Status)

ระบบสถานะของตัวละครถือเป็นสิ่งที่ขาดไม่ได้สำหรับเกม RPG ซึ่งจะเป็นส่วนที่แสดงถึงความสามารถของตัวละครในเกม โดยจะแบ่งเป็นค่าพลังและเงื่อนไขต่างๆ ดังต่อไปนี้

- **HP (Health Point)** ค่าพลังชีวิตของตัวละคร ซึ่งหากผู้เล่นถูกโจมตีจากศัตรู ค่านี้จะลดลงตามสัดส่วนที่กำหนด และเมื่อลดลงถึง 0 จะหมายถึงตัวละครนั้นตาย และต้องกลับไปยังจุดที่กำหนด
- **ATK (Attack)** ค่าพลังโจมตี มีผลกับค่าความเสียหาย (Damage) ที่ตัวละครนั้นๆ จะสร้างได้ ยิ่งมีค่ามากก็จะสามารถสร้างความเสียหายเพื่อลด HP ของศัตรูได้มากขึ้น
- **DEF (Defense)** ค่าพลังป้องกัน มีผลในการลดความเสียหายที่ได้รับ ซึ่งหากมีค่าพลังนี้เพิ่มขึ้น ก็จะได้รับความเสียหายลดลง ส่งผลให้ HP ลดน้อยลงไปด้วย มีผลต่อความอยู่รอดของตัวละคร
- **AGI (Agility)** ค่าความว่องไว หากผู้เล่นมีค่านี้มากขึ้น ก็จะส่งผลให้สามารถใช้ทักษะ (Skill) ได้บ่อยขึ้น ซึ่งแต่ละทักษะของตัวละครนั้นจะมีค่าที่เรียกว่าคูลดาวน์ (Cooldown) เป็นเลขนับจำนวนหนึ่งที่จะเริ่มนับเมื่อใช้ทักษะออกไปแล้ว ต้องรอให้เลขจำนวนนี้นับถอยหลังจนเป็น 0 จึงจะใช้ทักษะนั้นๆ ใหม่ได้ โดยค่านี้จะช่วยลดอัตราการคูลดาวน์ลง ทำให้สามารถใช้ทักษะได้บ่อยขึ้น
- **SP (Skill Power)** ค่าทักษะ ซึ่งจะนำมาใช้ในการใช้ทักษะต่างๆ ของตัวละคร ซึ่งทักษะแต่ละประเภทจะมีความต้องการ SP อยู่ หากมีไม่ถึงระดับที่ต้องการจะไม่สามารถใช้ได้ ซึ่งค่านี้จะเพิ่มขึ้นทีละ 1 หน่วยเมื่อโจมตีโดนศัตรูหรือได้รับมาจากทางอื่น โดยจะสะสมได้ถึง 100 แต้ม และจะลดลงเรื่อยๆ เมื่อหยุดใช้อาวุธระยะหนึ่ง และจะถูกล้างเป็น 0 หากเปลี่ยนเป็นอาวุธต่างชนิดจากเดิม
- **MP (Magic Point)** ค่าพลังเวทมนตร์ ซึ่งเป็นค่าพลังที่เป็นของคู่หูซึ่งเป็นนักเวทย์ ใช้ในการร่ายมนตร์ (Cast) บทต่างๆ ซึ่งเปรียบดั่งเป็นทักษะของตัวละครนักเวทย์ จะมีลักษณะพิเศษคือต้องใช้ค่า MP ที่มีอยู่ เมื่อร่ายแล้วค่านี้จะลดลงไป หากมีไม่เพียงพอต่อความต้องการในการร่ายก็จะไม่สามารถใช้เวทย์นั้นๆ ได้จนไปถึงไม่สามารถร่ายเวทย์ได้

- **INT (Intelligence)** ค่าความฉลาด มีผลต่ออัตราความเร็วในการร่ายมนตร์ หรืออัตราความเร็วตั้งแต่เริ่มร่ายมนตร์จนเกิดผลของเวทมนตร์บทนั้นๆ ออกมา รวมไปถึงโอกาสที่เวทมนตร์จะสัมฤทธิ์ผลในกรณีที่เป็นเวทมนตร์ที่เกี่ยวข้องกับสถานะใด ๆ
- **TAL (Talent)** ค่าพลังพรสวรรค์ หากมีมาก เวทมนตร์ที่ร่ายออกมาจะได้ผลรุนแรงขึ้น สร้างความเสียหายเพิ่มขึ้นในกรณีที่เป็นเวทมนตร์ที่สร้างความเสียหายต่อศัตรู
- **EXP (Experience)** ค่าประสบการณ์ เป็นอีกหนึ่งค่าที่เกม RPG ส่วนใหญ่มี เป็นค่าที่เราจะได้รับเมื่อสามารถปราบศัตรูตัวหนึ่งๆ ลงได้ เป็นตัวแทนถึงความชำนาญของตัวละคร และมีผลต่อสถานะของตัวละครเมื่อสูงถึงระดับหนึ่ง นอกจากนี้ยังมีค่าประสบการณ์ของอาวุธแยกออกมา จะมีผลต่อทักษะของตัวละคร ซึ่งจะกล่าวถึงในส่วนถัดไป
- **LV (Level)** ค่าระดับของตัวละคร เป็นค่าที่บ่งบอกถึงความสามารถโดยรวมของตัวละคร ซึ่งจะเพิ่มขึ้นเมื่อค่าประสบการณ์ถึงระดับหนึ่งแล้ว
- ระดับของตัวละครเริ่มต้นที่ 1 (LV. 1) และสูงสุดที่ 50 (LV. 50)
- ระดับความต้องการค่าประสบการณ์เพื่อเพิ่มไปสู่ระดับต่อไปเป็นไปตามสมการที่ 3-1 โดย X_n คือค่าประสบการณ์ทั้งหมดที่ระดับ n และ n คือระดับของผู้เล่นในปัจจุบัน

$$X_n = 10(n-1) + 1.16(X_{n-1}) \tag{3-1}$$

- ค่า HP และ MP ของตัวละครหลัก จะเพิ่มขึ้นเมื่อมีระดับสูงขึ้น สามารถการฟื้นฟูกลับมาได้เมื่อระยะเวลาที่ผ่านไปชั่วขณะหนึ่ง หรือหยุดพัก
- ค่า ATK DEF เป็นสถานะของมอนสเตอร์ทั่วไปและตัวเอก
- ค่า INT TAL เป็นสถานะพิเศษของตัวละครคู่หู
- ค่าพลัง ATK DEF TEC INT TAL ของตัวละครหลักจะเพิ่มขึ้นอย่างละ 1 แต้มเมื่อระดับเพิ่มขึ้น 1 ระดับ และแต้มพิเศษ (Bonus Point) 3 แต้มให้ผู้เล่นได้เลือกเพิ่มเข้าไปด้วยตัวเองตามต้องการ โดยจะเพิ่มขึ้นสูงสุดได้ที่ 255
- การคำนวณค่าความเสียหาย ใช้สูตรที่ 3-2 และ 3-3 ตามลำดับ โดย D ความเสียหายเริ่มต้น a แทนค่าพลังโจมตี d แทนค่าพลังป้องกัน และ A คือความเสียหายที่ทำได้

$$D = 2.5a - \frac{d}{1.5} \tag{3-2}$$

$$A = D(0.85) \pm 15\% \tag{3-3}$$

- การโจมตีโดนจุดตาย (Critical) จะทำให้สร้างความเสียหายได้เพิ่มขึ้นซึ่งจะคิดจากตัวเลขสุ่มที่ทำได้เมื่อกดโจมตีในอัตราพื้นฐาน 3% หรือมากกว่านั้นตามแต่ผลของอาวุธหรือทักษะบางประเภท จะได้ค่าความเสียหายดังสมการที่ 3-4 โดยที่ C คือความเสียหายที่ได้จากการโจมตีโดนจุดตาย

$$C = 2A + a \tag{3-4}$$

3.1.5.2 ระบบไอเท็มและการจัดเก็บ (Item And Inventory)

ไอเท็ม หรือของภายในเกม เป็นตัวช่วยและส่วนประกอบที่สำคัญอย่างหนึ่งในเกมทั่วไป รวมถึงเกม RPG นี้ด้วย สำหรับเกมนี้ จะแบ่งไอเท็มออกเป็นชนิดต่างๆ ได้แก่

- Useable Item ไอเท็มประเภทที่สามารถกดใช้ได้ เมื่อกดใช้แล้วให้ผลต่างกันไป เช่น ฟื้นฟูค่า HP เพิ่มค่าสถานะของตัวละครชั่วคราว เป็นต้น
- Equipment แบ่งย่อยออกเป็น หมวกและอาวุธ โดยจะอธิบายเพิ่มเติมในส่วนของระบบเครื่องแต่งกายต่อไป โดยหลักจะส่งผลในการส่งเสริมค่าสถานะของตัวละคร
- Miscellaneous ไอเท็มประเภทจิปาถะ อาจใช้ในการขายอย่างเดียว หรือนำมาขว้างปาได้
- Unknown ไอเท็มปริศนา ซึ่งจำเป็นต้องนำไปให้ตัวละครบางตัวจำแนกคุณลักษณะก่อน จึงจะนำมาใช้ได้
- Key Item ไอเท็มที่ไม่สามารถทิ้งได้ อาจด้วยไม่สามารถหาได้อีกเมื่อทิ้งไปแล้ว หรือมีความสำคัญต่อเนื้อเรื่อง
- Money หรือเงินตรา สามารถสะสมได้จากการเอาชนะศัตรูทั่วไป หรือนำไอเท็มชนิดอื่นไปขายให้กับร้านค้าในเกม เพื่อนำไปซื้อไอเท็มชนิดอื่นจากร้านค้าภายในเกม

ในการจัดเก็บไอเท็มภายในเกม แบ่งได้เป็นสองส่วนได้แก่ ส่วนที่ติดอยู่กับตัว โดยจะสามารถจัดเก็บได้ทั้งหมด 25 ช่องโดยไอเท็มประเภท Useable Item และ Miscellaneous จะสามารถ Stack ในช่องหนึ่งๆ ได้จำนวนหนึ่ง และส่วนคลังเก็บของที่บ้านของตัวเอก จะสามารถสะสมของได้ไม่จำกัดชิ้นตามต้องการ แต่จะไม่สามารถใช้ได้จนกว่าจะนำออกมาติดอยู่กับตัว

3.1.5.3 ระบบเครื่องแต่งกาย (Equipment)

ระบบเครื่องแต่งกายจะแบ่งตามประเภทของอาวุธ กับหมวก โดยจะแบ่งได้เป็นสามส่วนได้แก่ ส่วนหัว มือซ้าย และมือขวา โดยส่วนหัวจะสวมใส่ด้วยหมวกหรือเครื่องป้องกันพื้นฐาน มือซ้ายและมือขวา หรือในที่นี้คือสิ่งที่ตัวเอกสามารถควบคุมได้ด้วยมือที่มองไม่เห็นด้านซ้ายและขวา จำแนกตามแต่ชนิดของอาวุธที่ใช้ โดยมือขวานั้น หากนำไอเท็มที่ไม่ใช่อาวุธมาสวมใส่ จะแสดงผลเป็นการโยนของออกไปแทน

ประเภทของอาวุธ จำแนกได้เป็น 3 ประเภท ได้แก่

- อาวุธมือเดียวและโล่ (One Hand And Shield) โดยมือขวาจะเป็นอาวุธประเภทที่สามารถถือด้วยมือเดียวได้เช่นดาบ มีดสั้น และมือขวาสวมโล่หรือเครื่องป้องกันอื่นได้ ซึ่งอาวุธประเภทนี้จะเด่นในด้านการป้องกัน แต่สร้างความเสียหายได้ไม่มากนัก
- อาวุธหนัก (Heavy Weapon) ต้องใช้ทั้งมือซ้าย และมือขวา ในการสวมใส่ อาวุธประเภทนี้จะเน้นไปที่การโจมตีที่หนักหน่วง แต่มีข้อด้อยที่น้ำหนักมาก ทำให้ผู้เล่นขยับได้ช้าลง
- อาวุธไกล (Range Weapon) มือขวาจะถืออาวุธ เช่นธนู หรือปืน และมือซ้ายในการถือสิ่งที่ใช้สำหรับยิง ในที่นี้คือลูกธนู หรือกระสุน ซึ่งอาวุธชนิดนี้จำเป็นต้องมีสิ่งที่ใช้สำหรับยิงจึงจะสามารถโจมตีออกไปได้ อาวุธประเภทนี้จะสามารถใช้โจมตีได้ในระยะไกล แต่มีข้อด้อยที่การใช้งานในระยะใกล้

3.1.5.4 ระบบการเรียนรู้ทักษะ (Skill Learning)

ระบบการเรียนรู้ทักษะภายในเกมนี้ จะจำลองโดยใช้หลักในการฝึกอย่างเช่นในชีวิตจริงเข้ามาร่วมด้วย กล่าวคือตัวละครจะมีค่าความชำนาญจำแนกไปตามอาวุธแต่ละประเภท หากใช้อาวุธประเภทใดมากก็จะชำนาญในอาวุธชิ้นนั้นๆ เพิ่มขึ้น ส่งผลให้สามารถเรียนรู้ทักษะในการใช้อาวุธชนิดนั้นๆ ได้เพิ่มขึ้น

สำหรับค่าความชำนาญ จะคิดเป็นระดับ (Level) และมีค่าประสบการณ์ (Experience) คล้ายระบบสถานะ แต่แยกออกมา ซึ่งเมื่อสะสมค่าประสบการณ์ที่ได้จากการใช้อาวุธชนิดนั้นๆ ถึงจุดหนึ่ง ระดับความชำนาญในการใช้อาวุธประเภทนั้นๆ ก็จะเพิ่มขึ้น และได้แต้มสำหรับการเรียนรู้ทักษะของอาวุธชนิดนั้นๆ 1 แต้มต่อระดับ รวมถึงนำไปสู่การเรียนรู้ทักษะในระดับสูงขึ้นไปอีกด้วย โดยมีระดับสูงสุดที่ระดับ 15

ค่าความชำนาญของอาวุธ คิดตามสถานการณ์ต่างๆ ดังต่อไปนี้

- ได้รับ 2 แต้ม หากโจมตีไม่ถูกตัวศัตรู หรือการโจมตีที่ไม่มีประสิทธิภาพ สร้างความเสียหายได้น้อย
- ได้รับ 5 แต้มหากโจมตีโดนศัตรูที่มีระดับต่ำกว่าผู้เล่น 3 ระดับลงไปหรือใช้ทักษะไม่ถูกเป้าหมาย
- ได้รับ 10 แต้ม หากโจมตีโดนศัตรูที่มีระดับใกล้เคียงกับต่ำกว่าไม่เกิน 3 ระดับหรือเทียบเท่า
- ได้รับ 15 แต้ม หากโจมตีโดนศัตรูที่มีระดับเทียบเท่ากับกับผู้เล่น
- ได้รับแต้มเพิ่มคิดจากส่วนต่างของระดับที่มากกว่าของศัตรูกับระดับของผู้เล่นเป็นจำนวนเท่ากับกับความต่างของระดับผู้เล่นกับศัตรู *2
- ได้รับแต้มเพิ่ม 10 แต้ม หากใช้ทักษะโดนเป้าหมาย

โดยจำนวนแต้มที่ต้องการเพื่อเพิ่มไปสู่ระดับต่อไป คิดตามสมการที่ 3-5 โดย X_s คือค่าประสบการณ์ที่ต้องการในระดับ s โดย s คือระดับของทักษะในขณะนั้น

$$X_s = 150X_s + 1.1X_{s-1} \quad (3\text{-}5)$$

ทักษะของอาวุธแต่ละชนิดนั้น จะแบ่งได้เป็น 5 แถว โดยแถวที่ 1 – 3 ผู้เล่นจะสามารถเลือกเรียนรู้ได้โดยแบ่งเป็นทักษะประเภท Active 2 ระดับ และ Passive 3 ระดับเมื่อ

- ตัวละครมี Level 10 จะสามารถเรียนรู้ทักษะในแถวที่ 2 ได้หากมีทักษะในแถวที่ 1 ถูกเรียนรู้ไปแล้ว 3 แต้ม
- ตัวละครมี Level 15 จะสามารถเรียนรู้ทักษะในแถวที่ 3 ได้หากมีทักษะในแถวที่ 2 ถูกเรียนรู้ไปแล้ว 3 แต้ม

นอกเหนือจากทักษะอาวุธแล้วยังมีทักษะในส่วนของการโยนไอเท็ม ซึ่งจะเป็นทักษะที่เรียนรู้เพิ่มเติมได้เองเมื่อระดับของผู้เล่นสูงขึ้น เนื่องจากการโจมตีด้วยการโยนไอเท็มนั้นจะคิดความเสียหายตามน้ำหนักของไอเท็ม ซึ่งจะเป็นการสิ้นเปลืองค่าใช้จ่ายมากเกินไปในการฝึก ผนวกกับเนื้อเรื่องซึ่งกำหนดให้สิ่งมีชีวิตเผ่าสไลม์ เป็นเผ่าพันธุ์ที่มีความสามารถในการจัดการกับสิ่งของอยู่แล้ว

ทักษะอีกส่วนหนึ่งคือเวทมนตร์ของคู่หู จะใช้ระบบเช่นเดียวกับการโยนไอเท็ม ส่วนของเวทมนตร์ออกแบบมาเพื่อสนับสนุนผู้เล่นอีกทางหนึ่ง เพื่อให้สามารถกำจัดศัตรูได้ง่ายขึ้น และเพิ่มโอกาสอยู่รอดของตัวละครให้มากขึ้น เนื่องจากทักษะส่วนใหญ่ของตัวเอกจะเป็นทักษะด้านการโจมตี อีกทั้งศัตรูมักมีความสามารถสูงกว่าเรา หรือจำนวนมากกว่าเรา โดยจะแบ่งทักษะเป็น 4 สาย ได้แก่

- Destruction สายเวทมนตร์ทำลายล้าง เวทมนตร์ชนิดนี้จะเน้นไปที่การช่วยทำความเสียหายเพิ่มเติมให้กับผู้เล่น เพื่อให้ปราบศัตรูได้ไวขึ้น
- Support สายเวทมนตร์สนับสนุน เวทมนตร์ชนิดนี้จะช่วยส่งเสริมค่าสถานะของผู้เล่นขึ้นชั่วคราว ใช้พลิกแพลงได้ตามสถานการณ์
- Confusion สายเวทมนตร์ปั่นป่วน เวทมนตร์ชนิดนี้จะใช้ในการลดความสามารถของศัตรูลง
- Life สายเวทมนตร์ชีวิต เวทมนตร์ชนิดนี้จะช่วยฟื้นฟูค่าพลังต่างๆ ของผู้เล่นให้กลับคืนมา โดยเฉพาะค่า HP

3.1.5.5 ระบบเวลา (Time)

ภายในเกมใช้ระบบเวลาแบบ Real Time โดยมีการแสดงผลเป็นตอนกลางวัน – กลางคืน 24 ชั่วโมง โดยศัตรูที่ปรากฏในฉากบางจุดจะมีการเปลี่ยนแปลงไปตามช่วงเวลานั้นๆ ร้านค้าภายในเมืองที่เปิดปิดตามเวลา และตัวละครภายในเมืองที่จะเปลี่ยนตำแหน่งที่อยู่ของตนในแต่ละช่วงเวลาของแต่ละวันไป รวมไปถึงเหตุการณ์เฉพาะอื่นๆ

นอกจากนี้ยังใช้กับพักผ่อนของตัวละครที่บ้าน เป็นตัวช่วยในการข้ามช่วงเวลาที่เราต้องการข้ามไป โดยกำหนดให้การพักผ่อนแต่ละครั้งจะใช้เวลา 6 ชั่วโมงนับจากเวลาที่เริ่มพักผ่อน

3.1.5.6 ระบบภารกิจ (Mission and Quest)

ระบบภารกิจ หรือเรียกกันโดยทั่วไปว่า ระบบเควสท์ เป็นระบบหนึ่งที่ช่วยให้ผู้เล่นได้รับจุดหมายในการเล่นเพิ่มเติม เป็นตัวช่วยเพิ่มเติมให้ได้มาซึ่งรางวัลสนับสนุนเป็นการตอบแทน หลักๆ คือไอเท็มและเงิน ที่เป็นปัจจัยหลักที่ช่วยให้ผู้เล่นเล่นได้ง่ายขึ้น รวมไปถึงเพิ่มเนื้อหาเชิงลึกของตัวละครในโลกของเกมอีกด้วย

โดยทั่วไป ผู้เล่นจะได้รับภารกิจเมื่อได้สื่อสารกับตัวละครที่กำลังมีปัญหาหรือต้องการความช่วยเหลือจากผู้เล่น และให้ผู้เล่นได้เลือกว่าจะรับผิดชอบเควสท์นั้นๆ หรือไม่ตามความต้องการ โดยตัดสินใจจากเงื่อนไขที่ตัวละครต้องการได้รับความช่วยเหลือ ในที่นี้อาจเป็น ระยะเวลาที่จำกัด ระดับของศัตรู ความพร้อมของผู้เล่น เมื่อผู้เล่นรับผิดชอบเควสท์ได้ผ่านตามเงื่อนไขแล้วก็จะได้รับรางวัลตอบแทน

ลักษณะเงื่อนไขของเควสท์โดยพื้นฐาน จะแบ่งเป็นประเภทได้ดังนี้

- ความต้องการไอเท็ม ตัวละครจะมีเหตุที่ต้องการไอเท็มชนิดใดชนิดหนึ่ง จำนวนหนึ่ง ให้ผู้เล่นตามหาไอเท็มชนิดนั้นมาให้ ก็จะได้รางวัลตอบแทน ซึ่งอาจเป็นไอเท็มที่ผู้เล่นมอยู่แล้วไม่ต้องการใช้
- กำจัดศัตรู ตัวละครจะมีเหตุให้ผู้เล่นช่วยเหลือโดยการไปกำจัดศัตรูชนิดที่กำหนด ในจำนวนที่กำหนด แล้วจึงกลับมาแจ้งตัวละครนั้นๆ อาจจะพร้อมหลักฐานเป็นไอเท็มพิเศษที่จะทิ้งไม่ได้ (Key Item) เพื่อขอรับรางวัล
- ระยะเวลา ภารกิจบางชนิดอาจมีเงื่อนไขในด้านของเวลา เป็นข้อจำกัดให้ผู้เล่นทำภารกิจนั้นๆ ให้สำเร็จตามระยะเวลาที่กำหนด หากไม่สามารถทำได้จะถูกยกเลิก ซึ่งอาจจะรับภารกิจนั้นเพื่อแก้ตัวใหม่ได้ในบางกรณี

3.1.5.7 ระบบเซฟ (Save System)

เป็นระบบสำคัญของเกม RPG ทั่วไปคือต้องสามารถบันทึกความคืบหน้าได้ โดยจะใช้บันทึกเป็นไฟล์ สำหรับอ่านเข้ามาในโปรแกรม แล้วเข้าปรับเปลี่ยนค่าต่างๆ ให้กลับมาสู่สถานะเดิมก่อนที่ผู้เล่นจะหยุดเล่นไปในช่วงก่อนหน้านี้ โดยส่วนนี้จะเป็นการทำงานร่วมกันระหว่างส่วนควบคุมข้อมูลและส่วนความคุมเหตุการณ์ของโปรแกรม

โดยการเซฟภายในเกม จะสามารถทำได้ 3 ช่องทาง ได้แก่ บ้านของตัวเอก จุด Checkpoit ภายในสถานที่ และAuto Save เมื่อจบแต่ละตอน โดยเซฟแบบสุดท้ายนี้มีขึ้นเพื่อให้ผู้เล่นย้อนกลับมาเล่นได้ในกรณีที่ต้องการเก็บรายละเอียดเล็กๆ น้อยๆ เช่นภารกิจ เป็นต้น

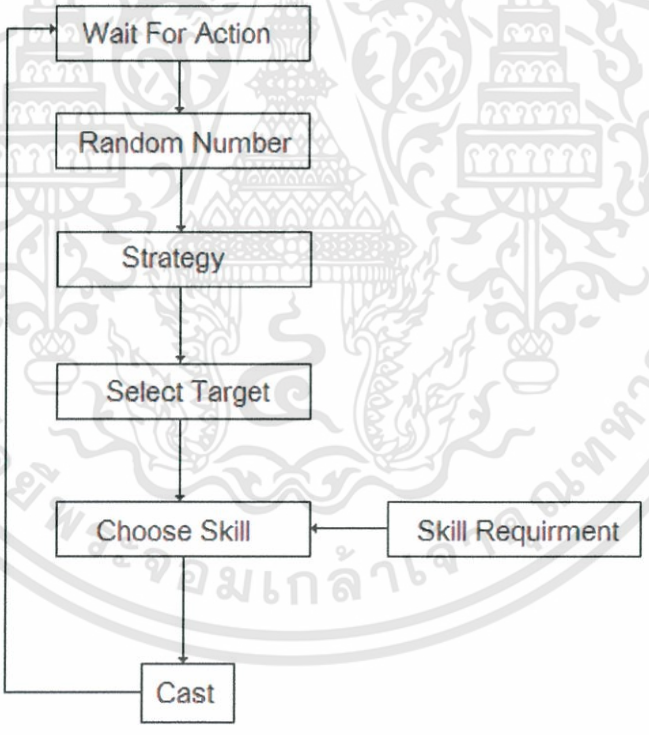
ภายในไฟล์เซฟ จะประกอบไปด้วยข้อมูลต่างๆ ดังต่อไปนี้

- Status สถานะของตัวละครทั้งหมด ตั้งแต่สถานะพื้นฐาน ไปจนถึงทักษะที่มีอยู่
- Switch And Variable ตัวแปรต่างๆ ที่ใช้เป็นเงื่อนไขเหตุการณ์ภายในเกม โดยเก็บเป็นค่าตัวแปรชุดหนึ่ง เพื่ออ้างอิงความคืบหน้า และนำมาใช้ประกอบในการโหลดกลับมาที่สถานะเดิม
- Item ไอเท็มที่ผู้เล่นพกติดตัว และไอเท็มที่เก็บเอาไว้ที่บ้านพักทั้งหมด
- Time เวลาปัจจุบัน

3.1.6 ปัญญาประดิษฐ์ภายในเกม (Artificial Intelligence)

3.1.6.1 ปัญญาประดิษฐ์ของคู่หู (Partner AI)

ค้าวคาวซึ่งเป็นคู่หูของตัวเอก จะเป็นตัวช่วยเราระหว่างการต่อสู้ ซึ่งจะมีลักษณะเป็นตัวละครที่ช่วยสนับสนุนผู้เล่น ด้วยการเลือกใช้เวทมนตร์ชนิดต่างๆ จากในส่วนของการเรียนรู้ทักษะซึ่งแบ่งเป็น 4 สาย แต่ละสายจะมีความโดดเด่นแตกต่างกันไป และเหมาะสมกันสถานการณ์ต่างกันไป อีกทั้งจำเป็นต้องใช้อย่างคุ้มค่า จากค่า MP ที่มีจำกัดในการร่าย ระยะคูลดาวน์ของการร่าย



รูปที่ 3-12 แผนภาพแสดงลำดับพฤติกรรมของ Partner AI

สำหรับวิธีการเลือกใช้เวทมนตร์ของ AI จะใช้การวางแผน (Strategy) กล่าวคือให้ผู้เล่นเลือกแผนให้กับคู่หูตามสถานการณ์ แล้ว AI จะทำการเลือกเวทมนตร์ที่อยู่ในแผนการนั้นๆ มาใช้ โดยคำนึงถึงเงื่อนไขต่างๆ ดังต่อไปนี้

- การสนับสนุนจะไม่เริ่ม จนกว่าผู้เล่นจะเริ่มการโจมตีศัตรูตัวใดตัวหนึ่ง
- เวทมนตร์ต้องมีเป้าหมายที่ชัดเจนในการร่าย ในกรณีเป็นเวทมนตร์ประเภทสนับสนุนและเวทมนตร์ชีวิตจะใช้ให้กับผู้เล่น ส่วนเวทมนตร์ประเภททำลายล้างและปั่นป่วนจะใช้กับศัตรู ไม่ใช่กระทำในทางตรงกันข้าม ซึ่งจะก่อให้เกิดผลเสียตามมา
- เวทมนตร์ประเภทสร้างความเสียหาย จะมีการคำนวณค่าความเสียหายของเวทมนตร์ที่จะทำได้ล่วงหน้ากับเป้าหมายหนึ่งๆ แล้วจึงเลือกเวทมนตร์ที่มีความรุนแรงในระดับที่พอดีต่อการกำจัดศัตรูตัวนั้นๆ ไม่เบาจนไม่เห็นผล หรือรุนแรงเกินไปและสิ้นเปลืองพลัง MP เกินความเหมาะสม ในทางกลับกันก็จะใช้กับส่วนของเวทมนตร์ชีวิต ที่จะมีการคำนวณเพื่อเลือกเวทมนตร์ที่ฟื้นฟูพลังได้พอเหมาะ
- การวางแผน จะเป็นการแบ่งสัดส่วนร้อยละของประเภทเวทมนตร์ว่า จะใช้เวทมนตร์ใด ในโอกาสเท่าใด โดยใช้ค่าสุ่มขึ้นมาค่าหนึ่งระหว่าง 0 – 100 ในการตัดสิน แต่ถ้าหากเลขสุ่มตกในส่วนของเวทมนตร์ที่ไม่ผ่านเงื่อนไขใดๆ เช่น ตกในส่วนของเวทมนตร์รักษาแต่ HP ของผู้เล่นยังเต็มอยู่ เวทมนตร์ประเภทนี้ก็จะไม่มีประโยชน์ ก็จะถูกปรับเปลี่ยนไปเป็นเวทมนตร์ประเภทอื่นที่อยู่ในช่วงที่กำหนดไว้ หรือหยุดพักไปชั่วคราวเพื่อฟื้นฟูพลัง MP แทน
- จากเงื่อนไขข้างต้นสามารถสร้างแผนภาพได้ดังรูปที่ 3-12

3.1.6.2 ปัญญาประดิษฐ์ของศัตรู (Enemy AI)

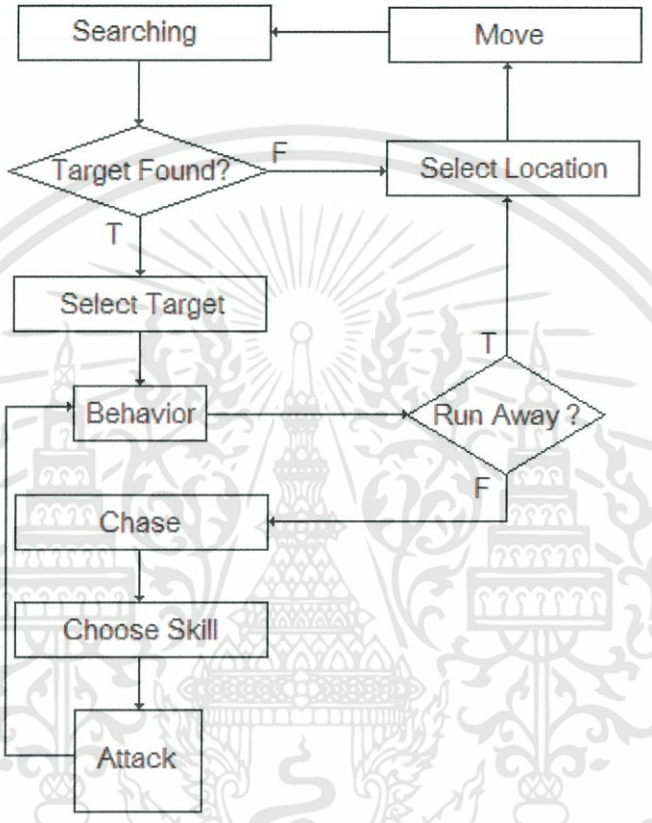
ศัตรูภายในเกม จะมีอยู่ตามส่วนของแผนที่ต่างๆ ที่ไม่ใช่ตัวเมือง มีเป้าหมายในทางกลับกันกับส่วนของคู่หู นั่นก็คือ ทำทุกวิธีทางที่สามารถทำได้ เพื่อทำให้ HP ของผู้เล่นลดลงเหลือ 0 หรือทำให้การเล่นเป็นไปอย่างยากลำบากขึ้น

ศัตรูแต่ละชนิด จะมีความสามารถ และสถานะแตกต่างกันไป แต่โดยส่วนใหญ่แล้วจะมีเงื่อนไขของพฤติกรรมเป็นไปตามแผนที่กำหนดไว้ ซึ่งจะคล้ายกับในส่วน AI ของคู่หู แต่เปลี่ยนจากประเภทของเวทมนตร์ เป็นลักษณะของพฤติกรรมที่ศัตรูตัวนั้นๆ สามารถทำได้แทนแบ่งได้เป็นเงื่อนไขหลักๆ ดังต่อไปนี้

- ศัตรูทั่วไปที่อยู่บนฉากแผนที่จะยังไม่แสดงพฤติกรรมที่เป็นภัยต่อผู้เล่น ซึ่งศัตรูทั่วไปจะมีการเปลี่ยนตำแหน่งของตนเป็นระยะสั้นๆ ในทิศทางที่กำหนดเอาไว้ และหยุดชั่วระยะเวลาหนึ่ง ก่อนที่จะเคลื่อนที่อีกไปเรื่อยๆ เพื่อหาผู้เล่น หรือรอรับการโจมตี เพื่อจำลองนิสัยและพฤติกรรมของศัตรูตัวนั้นๆ ออกมา
- เมื่อศัตรูถูกผู้เล่นโจมตี หรือพบเห็นผู้เล่น จึงจะเริ่มเคลื่อนที่เปลี่ยนไป ซึ่งส่วนใหญ่จะเป็นการเคลื่อนที่เข้าหาผู้เล่นเพื่อให้สามารถโจมตีโดนตัวผู้เล่นได้ในระยะที่เหมาะสม และโดนเป้าหมายที่

เป็นตัวเรา หรืออาจจะเป็นเคลื่อนที่ออกห่างผู้เล่นเพื่อหนี หรือสร้างความปั่นป่วนให้ผู้เล่นตามพฤติกรรมที่กำหนดเอาไว้

- พฤติกรรมพิเศษของศัตรู ศัตรูบางตัวที่เป็นหัวหน้าหรือตัวละครหลัก มักจะมีพฤติกรรมเปลี่ยนไปตามระดับพลังชีวิตที่ลดลงได้ด้วย และอาจรวมไปถึงคำพูดประกอบที่จะแสดงออกมา
- จากเงื่อนไขข้างต้นสามารถสร้างแผนภาพได้ดังรูปที่ 3-13

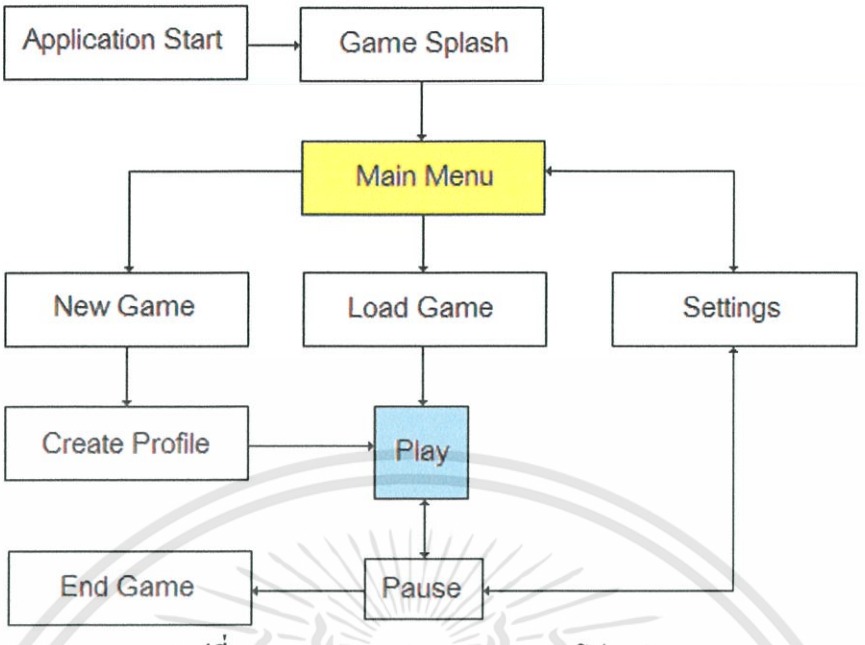


รูปที่ 3-13 แผนภาพแสดงลำดับพฤติกรรมของ Enemy AI

3.1.7 ส่วนติดต่อกับผู้ใช้ (User Interface)

3.1.7.1 ภาพรวมของส่วนติดต่อผู้ใช้

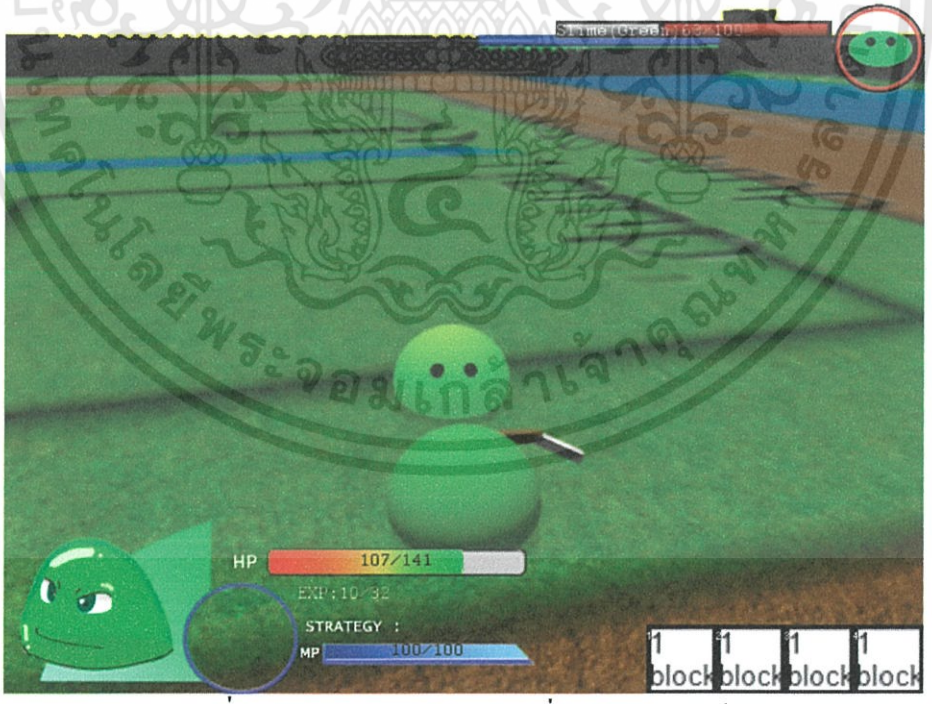
เป็นไปตาม Front End Flow Chart ดังรูปที่ 3-14 เมื่อเปิดโปรแกรมเกมขึ้นมาแล้วจะเข้าสู่หน้า Game Splash ก่อนเข้าสู่หน้าจอเมนูหลัก (Main Menu) เลือกเริ่มเกมใหม่ (New Game) โหลดเกมจากเซฟไฟล์ที่มี (Load Game) หรือปรับแต่งค่าของโปรแกรม (Settings) หากเริ่มเกมใหม่จะเข้าสู่หน้าจอสร้าง Profile ในที่นี้คือสร้าง Save File ขึ้นมาใหม่ แล้วจึงเข้าสู่เกม หากโหลดเกม ก็จะทำการอ่านข้อมูลจากไฟล์เซฟที่เลือก ปรับแต่งสภาพแวดล้อมและสถานะของตัวละคร แล้วจึงเข้าสู่เกม ระหว่างเกมเมื่อเรียกเมนูภายในเกมจะเป็นการหยุดเกมชั่วคราว สามารถเข้าสู่ส่วนปรับแต่งค่าแล้วกลับมาเล่นต่อ หรือจบเกมและปิดโปรแกรม



รูปที่ 3-14 Front-End Flow Chart ของโปรแกรม

3.1.7.2 Head-Up Display (HUD Interface)

เมื่อผู้เล่นเข้าสู่ตัวเกม จะแสดงหน้าจอหลักที่เป็นฉาก 3 มิติสำหรับมองการเคลื่อนไหวของตัวละคร และมี HUD เป็นภาพ 2 มิติแสดงรายละเอียดพื้นฐานของตัวละคร



รูปที่ 3-15 ภาพตัวอย่างภายในเกมที่ใส่ Interface แล้ว

ส่วนแสดงสถานะของผู้เล่น ประกอบไปด้วย

- Player Display ส่วนแสดงรูปตัวละครของผู้เล่น
- Status Bar ประกอบไปด้วย HP แสดงระดับพลังชีวิตปัจจุบัน EXP แสดงค่าประสบการณ์ปัจจุบัน STRATEGY แสดงแผนการปัจจุบันที่มอบหมายให้แก่คู่หูของเรา MP แสดงระดับพลังเวทมนตร์คงเหลือที่คู่หูเหลืออยู่ และ CD ระดับคูลดาวน์ของคู่หูซึ่งจะลดลงเรื่อยๆ จนหมดแล้วจึงจะสามารถร่ายเวทถัดไปได้
- Weapon แสดงอาวุธที่ผู้เล่นถืออยู่ปัจจุบัน ซึ่งจะมีการแสดงเป็นจำควนคงเหลือของกระสุนในกรณีที่ใช้อาวุธปืน และระดับ SP ปัจจุบัน
- Skill Shortcut ปุ่มลัดสำหรับเลือกใช้ทักษะหรือไอเท็มที่ระบุไว้ เพื่อความสะดวกในการใช้ทักษะที่ใช้ได้ระหว่างการต่อสู้ และแสดงเป็นตัวเลขถอยหลังแทนระยะเวลาคูลดาวน์ของท่า
- Enemy Status แสดงข้อมูลพลังชีวิตของศัตรู จะปรากฏเมื่อทำการล็อคเป้าเลือกศัตรูตัวใดๆ ในฉาก

3.1.7.3 เมนูภายในเกม (Menus)

เมื่อเรียกเมนูภายในเกม ตัวเกมจะหยุด และแสดงรายการเพิ่มเติมให้ผู้เล่นได้จัดแต่งและตรวจสอบสถานการณ์ปัจจุบันของผู้เล่นดังต่อไปนี้

- Time แสดงเวลาปัจจุบันในโลกเกม
- Item เมนูสำหรับจัดการไอเท็มที่ผู้เล่นพกติดตัว และการสวมใส่อาวุธ
- Map เมนูแสดงแผนที่คร่าวๆ และตำแหน่งปัจจุบันของผู้เล่น
- Statistic And Attribute เมนูแสดงสถานะเพิ่มเติมของตัวละคร รวมถึงจัดการกับแต้มโบนัสที่ได้รับ และปรับแต่งปุ่มลัดสำหรับทักษะ
- Strategy เมนูปรับแต่งแผนการสำหรับคู่หู
- Quest เมนูแสดงรายละเอียดภารกิจที่ผู้เล่นรับผิดชอบอยู่
- Quit ออกจากเกม

3.1.8 เพลงประกอบและเสียง (Music And Sound)

เพลงประกอบฉาก (Background Music) และเสียงเอฟเฟค (Sound Effect) ใช้ไฟล์สกุล .wav เนื่องจาก DirectX รองรับไฟล์สกุล .mp3 ไม่ได้ โดยในส่วนนี้โปรแกรมสามรถโหลดไฟล์เสียงเข้ามาเล่นได้พร้อมกัน 100 เสียง ซึ่งเพียงพอต่อการผสมเสียงเอฟเฟคต่างๆ ภายในฉาก

เพลงประกอบฉากที่ใช้โดยประมาณ แบ่งตามสถานที่สำคัญภายในเกม และเพลงประกอบพิเศษสำหรับบางสถานการณ์ดังต่อไปนี้

- Main Menu เพลงเปิดเกม
- Forest เพลงในฉากป่าหน้าเมืองรีโอวาโน
- City (Reovano Theme) เพลงภายในเมืองรีโอวาโน
- Factory เพลงในฉากโรงงานร้าง
- Haunted Mansion เพลงในฉากคฤหาสน์ผีสิง
- Bandit Theme เพลงประกอบสถานการณ์ตึงเครียดและเพลงของตัวร้าย
- Boss Theme เพลงประกอบระหว่างต่อสู้กับศัตรูระดับหัวหน้า
- Final Battle Theme เพลงประกอบการต่อสู้กับสไลม์ยักษ์เก่าแก่
- Ending Theme เพลงจบเกม

เสียงเอฟเฟคและสภาพแวดล้อม แบ่งได้ดังต่อไปนี้

- Environment เสียงสภาพแวดล้อม
- Trigger เสียงประกอบปุ่มและลูกเล่น
- Weapon เสียงประกอบต่างๆ สำหรับอาวุธแต่ละประเภท
- Emotion เสียงประกอบอารมณ์

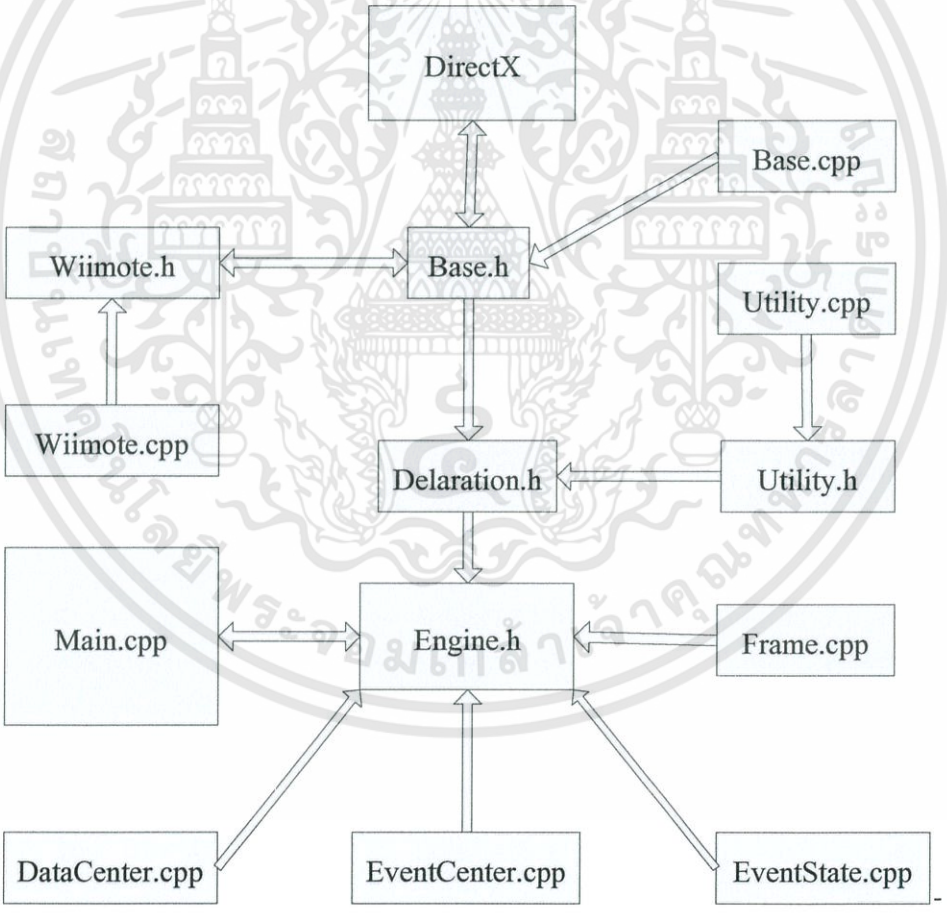
บทที่ 4
ผลการดำเนินงาน

4.1 การพัฒนาเกมรีโอวาโน

โปรแกรมเกมรีโอวาโน พัฒนาขึ้นด้วยภาษา C++ โดยใช้โปรแกรม Microsoft Visual C++ 2010 ในการพัฒนา สำหรับหรับทรัพยากรเครื่องที่ต้องการ การติดตั้งโปรแกรมและการตั้งค่าสำหรับการพัฒนาโปรแกรม รายละเอียดอยู่ในส่วนของภาคผนวก

หัวข้อนี้จะเป็นการอธิบายถึงความสำคัญของไฟล์ที่บรรจุโค้ดเอาไว้ ซึ่งแต่ละไฟล์จะมีหน้าที่แบ่งแยกการทำงานกันชัดเจน เพื่อให้สามารถทำการแก้ไขส่วนประกอบของโปรแกรมได้โดยง่ายขึ้น โดยไม่ทำให้เกิดความสับสน หรือมีการทำงานทับซ้อนกัน

สำหรับโครงสร้างโค้ดที่ใช้ สามารถอธิบายได้ดังรูปที่ 4-1



รูปที่ 4-1 แผนภาพแสดงโครงสร้างของไฟล์โค้ด

โดยแต่ละไฟล์ มีหน้าที่และความสำคัญดังต่อไปนี้

- main.cpp โค้ดส่วนหลักที่ทำหน้าที่เรียกเมธอดต่างๆ จากไฟล์อื่น ๆ
- base.cpp สำหรับติดต่อกับ DirectX โดยมีการเรียกไฟล์ base.h ซึ่งเป็น Header File สำหรับติดต่อ Interface กับDirectX ทั้งหมด
- EventCenter.cpp สำหรับปรับแต่ง เปลี่ยนแปลงข้อมูลระบบเกม
- EventState.cpp สำหรับรองรับค่าจากอุปกรณ์นำเข้าต่างๆ และพฤติกรรมของวัตถุต่างๆในจอ
- Frame.cpp สำหรับติดต่อกับระบบเกี่ยวกับการวาด มีทำหน้าที่ในการสร้างฉาก
- DataCenter.cpp สำหรับรวมรวมข้อมูลจากไฟล์ภายนอก
- Utility.cpp โค้ดสำหรับจัดการเกี่ยวกับตัวช่วยต่างๆ ในขั้นตอนวิธี
- Wiimote.cpp สำหรับการจัดการกับการเชื่อมต่อและรับค่าจาก Wiimote ทำให้โปรแกรมสามารถรับค่าที่ได้จาก Wiimote นอกเหนือจากเมาส์และคีย์บอร์ดที่เป็นอุปกรณ์นำเข้าพื้นฐาน

4.2 การสร้างฉาก 3 มิติ

ในส่วนนี้ จะเป็นการอธิบายวิธีการสร้างฉาก 3 มิติด้วย Direct3D เพื่อให้ได้มาซึ่งฉากของเกม แบ่งเป็นหัวข้อย่อยได้ดังต่อไปนี้

4.2.1 การสร้างพื้นดิน (Terrain) และวาด Texture

พื้นดิน (Terrain) เป็นชื่อเรียกของแผ่นฐานทรงสี่เหลี่ยม 3มิติ ที่มีลักษณะนูนหรือเป็นหลุมลงไปเพื่อแสดงลักษณะภูมิประเทศของฉาก ใช้เป็นพื้นฐานสำหรับวางวัตถุสามมิติชิ้นอื่นๆ ลงไป

ส่วนแรกที่จะต้องสร้างคือทำขึ้นแผ่นพื้นดินธรรมดาขึ้นมาก่อน จากนั้นจึงนำแผ่นพื้นดินที่ได้ไปใส่ Texture ซึ่งสร้างขึ้นโดยการนำเอารูปภาพ 2 มิติ แปะทาบลงไปบนหน้าของพื้นดินเปล่าที่สร้างขึ้นมา เพื่อให้ความรู้สึกว่าพื้นดินส่วนนั้นมีคุณลักษณะอย่างไร เช่น พื้นหญ้า พื้นดิน เป็นต้น

ในการวาด Texture บนฉากนั้น ใช้วิธีการอ่านจากภาพที่มีคุณสมบัติพิเศษเรียกว่า Texture Map เพื่อให้สามารถตกแต่งได้ง่ายผ่านโปรแกรมรูปแต่งรูป 2 มิติทั่วไปอย่าง Microsoft Paint

โดยโปรแกรมจะทำการอ่านข้อมูลจากไฟล์ภาพดังกล่าว ซึ่งจะแบ่งเป็นขนาดของไฟล์ภาพ ข้อมูลช่องสี 3 สี ประกอบไปด้วยสีแดง เหลือง เขียว และตำแหน่ง Pixel ของจุดสีนั้นๆ ซึ่งเมื่อโปรแกรมสร้าง Terrain ตามขนาดที่กำหนดแล้วจะนำมาคำนวณอัตราส่วนระหว่างแผ่นพื้นดินกับรูปภาพ Texture Map ว่ามีร้อยละเท่าใด จากนั้นจึงทำการทาบ Texture Map ลงไป และตรวจสอบว่าช่องสีใดมีความเข้มเท่าไหร่ โดยกำหนดอ้างอิงว่า จะให้สี่ช่องใด แทน Texture ประเภทใด

ตัวอย่างเช่น กำหนดให้มีแผ่นพื้นดินขนาด 100x100 และมี Texture Map ขนาด 20x20 ก็จะได้ว่า Texture Map 1 พิกเซล จะวาด Texture ลงไปบนแผ่นพื้นดินเป็นพื้นที่ 5x5

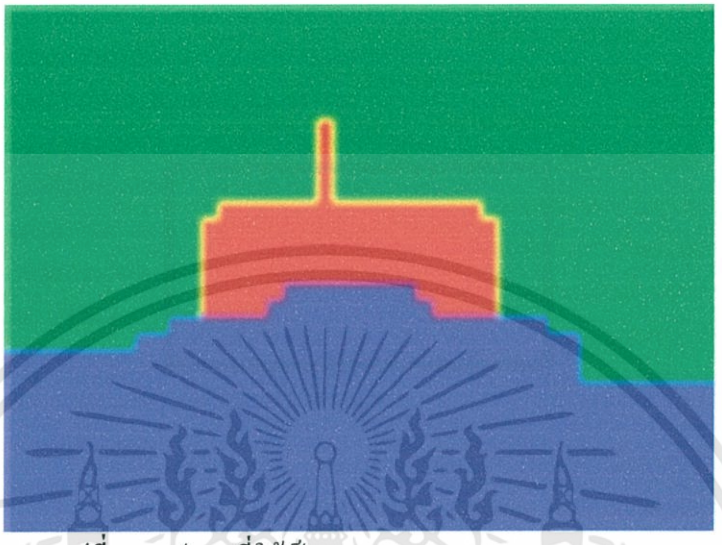
จากนั้นกำหนดให้ ช่องสีแดงให้วาด Texture พื้นทราย ช่องสีเหลืองให้วาด Texture พื้นดิน และช่องสีเขียวให้วาด Texture พื้นหญ้า หากที่จุดที่ตำแหน่ง 1,1 มีช่องสีแดงมีความเข้ม 255 ก็จะวาด Texture พื้นทรายให้มีความคมชัด (Alpha) 100 % ลงบนแผ่นพื้นดินเป็นบริเวณ 5x5 ที่มุมซ้ายบนสุดของแผ่นพื้นดิน

ในอีกกรณีเช่น หากช่องสีดังกล่าวสีแดงความเข้ม 255 และสีเขียวความเข้ม 128 ก็จะวาด Texture ช่องพื้นทรายด้วยความคมชัด 100% และวาด Texture พื้นหญ้าความคมชัด 50% ทำให้ได้พื้นดินที่มีลักษณะเป็นพื้นทรายมีสีเขียวของหญ้าแซมอยู่

โดยส่วนของโค้ดที่ใช้งานนี้จะอยู่ในไฟล์ชื่อว่า Light.fx ซึ่งเป็นไฟล์ที่เรียกใช้โดยการ์ดจอ ผ่านภาษา HLSL ดังที่นำเสนอไปแล้วโดยจะมี Pixel Shader ทำหน้าที่วาด Texture ลองไปบนแผ่นพื้นดิน ซึ่ง Pixel Shader นี้ถูกเรียกครั้งหนึ่งต่อการวาดหนึ่ง Pixel ทำให้สามารถเปลี่ยนสีระดับพิกเซลได้

```
//tMapSampler คือ Terrain Map
//tTextureSampler1, tTextureSampler2, tTextureSampler3 คือ Texture ของทั้งสามช่องสี
//incoming.texCoord คือพิกัด u, v ของพิกเซลที่เรนเดอร์อยู่
//tex2D เป็นเมธอดของ Shader ที่คืนค่ามาเป็นค่าสีของ Texture ในตำแหน่ง u, v ที่ส่งไป

float2 co = float2(incoming.texCoord.x / terrainMaxCoord.x, incoming.texCoord.y / terrainMaxCoord.y);
float4 mapCoord = tex2D(tMapSampler, co);
float tColor = mapCoord.r + mapCoord.g + mapCoord.b;
 float4 color = float4(0, 0, 0, 1);
if(tColor > 0)
        {
        color = tex2D(tTextureSampler1, incoming.texCoord) * mapCoord.r * mapCoord.r / tColor;
        color += tex2D(tTextureSampler2, incoming.texCoord) * mapCoord.g * mapCoord.g / tColor;
         color += tex2D(tTextureSampler3, incoming.texCoord) * mapCoord.b * mapCoord.b / tColor;
        }
if(hasTexture)
        {
        float4 fTemp – tex2D(ColoredTextureSampler, co);
         color = color * (1-fTemp.a) + fTemp * fTemp.a;
        }
outgoing.color *= color;
outgoing.color.a = 1;
```

ซึ่งในจุดนี้ จะได้แผ่นพื้นดินที่ใส่ Texture เรียบร้อยแล้ว รูปที่ 4-2 แสดง Texture Map โดยกำหนดให้สีแดง แทนพื้นทราย สีเขียว แทนพื้นหญ้า สีฟ้า แทนพื้นดินสีเข้ม และรูปที่ 4-3 แสดงแผ่นพื้นดินของฉากชายหาดที่ใส่ Texture เรียบร้อยแล้ว



รูปที่ 4-2 รูปภาพที่ใช้เป็น Texture Map ของฉากชายหาด

รูปที่ 4-3 ภาพพื้นดินที่ใส่ Texture เรียบร้อยแล้ว

4.2.2 การสร้างลักษณะภูมิประเทศ (Height Map)

เมื่อ ได้แผ่นพื้นดินมาแล้ว จึงนำมาเข้ากระบวนการเพื่อให้แผ่นพื้นดินมีลักษณะสูงต่ำต่างกัน ซึ่งแต่ละฉากจะมีลักษณะจุดที่สูงต่ำแตกต่างกันไปเช่น ชายหาดจะมีส่วนที่ลึกลงไปเป็นส่วนของทะเล และอีกส่วนที่เป็นเนินสูงขึ้นมามีต้นหญ้าขึ้น

ในการสร้างลักษณะภูมิประเทศ ใช้หลักการคล้ายกับการวาด Texture ลงบนแผ่นพื้นดิน โดยใช้การอ่านภาพที่บรรจุคุณลักษณะของภูมิประเทศเอาไว้ เรียกว่า Height Map โดยกำหนดให้ค่าสีแดง แทนการเพิ่มความสูง ค่าสีเขียว แทนการลดความสูง หากค่าสีแดงและเขียวมีค่าเป็น 255 จะทำการคำนวณจากจุดแทน

โดยส่วนของโค้ดนี้ เป็นส่วนหนึ่งของ Frame.cpp

```
//อ่านข้อมูลความความสูงต่ำของแต่ละจุดของ Terrain จาก Height Map
  LPDIRECT3DTEXTURE9 tTemp;
//D3DXCreateTextureFromFileEx คือเมธอดของ DirectX ที่จะทำการโหลด Texture มาจากไฟล์ ลงไปใส่
LPDIRECT3DTEXTURE9 ที่เราส่งเข้าไป
  D3DXCreateTextureFromFileEx(base->Device(),
node.GetContentByName(heightmap).content.c_str(), 0, 0, 1, 0, D3DFMT_A8B8G8R8,
    D3DPOOL_MANAGED, D3DX_FILTER_NONE, D3DX_FILTER_NONE, 0, 0, 0,
&tTemp);
  D3DSURFACE_DESC desc;
  tTemp->GetLevelDesc(0, &desc);
  D3DLOCKED_RECT lRect;
  tTemp->LockRect(0, &lRect, 0, 0);
  BYTE *bPointer = (BYTE*)lRect.pBits;
  width = ms[0]-1;
  height = ms[1]-1;
  for(int i = 0; i< ms[0]; i++) for(int j = 0; j< ms[1]; j++)
  {
    DWORD index=(i*4+(j*(lRect.Pitch)));
    int y = bPointer[index+2] - bPointer[index+1];
   // กำหนดว่าจุดหนึ่งๆอยู่ตรงไหนของฉาก
    terrain->setGrid(i, j, 150.0f*(i), (float)y, -150.0f*(j));
  }
```

จากนั้นเมื่อทำการปรับแต่งพื้นดินเรียบร้อยแล้ว จึงทำการ Initialize ให้ได้ออกมาเป็นแผ่นพื้นดิน

```
terrain->initialize(base->Device(), base);
```

ในจุดนี้ ก็จะได้แผ่นพื้นดินที่มีการปรับลักษณะภูมิประเทศเรียบร้อยแล้ว รูปที่ 4-4a แสดง Height Map และรูปที่ 4-4b แสดงแผ่นดินที่ทำการใส่ Texture พร้อมปรับระดับภูมิประเทศเรียบร้อยแล้ว

(ก)

(ข)

รูปที่ 4-4 (ก) รูปภาพที่ใช้เป็น Height Map (ข) แผ่นดินที่ใส่ Texture Map และ Height Map แล้ว

4.2.3 การสร้างพื้นน้ำ

ในฉากบางฉากนั้น จะมีจุดที่เป็นแอ่งลึกลงไปแล้วมักจะเป็นส่วนที่มีน้ำขังอยู่ ในจุดนี้จะเป็นวิธีการนำเอาแผ่นพื้นดินที่มีภูมิประเทศแล้ว มาคำนวณว่าจุดใดควรมีน้ำขังอยู่ ซึ่งในส่วนนี้จะทำโดยการวัดระดับพื้นดิน หากอยู่ในระดับต่ำกว่าที่กำหนดเอาไว้ ส่วนนั้นก็จะถูกปิดด้วยน้ำ จากนั้นจึงนำข้อมูลที่ได้มาสร้าง Texture น้ำปิดครอบลงไป จากนั้นทำการปรับปรุงเพิ่มเติมให้ Texture น้ำเคลื่อนไหวได้เพื่อจำลองลักษณะของน้ำ โดยมีโค้ดด้านล่างนี้เป็นส่วนหนึ่งของ Frame.cpp ทำหน้าที่ในส่วนนี้

```
if(ConvertToFloat(node.GetContentByName(waterheight).content) >= -999)
{
  terrain->SetWater(
    ConvertToFloat(node.GetContentByName(waterheight).content),
    node.GetContentByName(watertexture).content.c_str(),
```

```
            150.0f*width, -150.0f*height, base->Device(),
            D3DXVECTOR2(ConvertToFloat(node.GetContentByName(waterdirx).content),
              ConvertToFloat(node.GetContentByName(waterdirz).content)));
        }
```

เมื่อทำการสร้างผืนนำแล้ว ก็จะได้พื้นดินของฉากชายหาดที่สมบูรณ์ดังรูปที่ 4-5 พร้อมนำไปเข้าสู่ขั้นตอนการเพิ่มรายละเอียดต่อไป

รูปที่ 4-5 แผ่นดินชายหาดที่ปรับแต่งเสร็จสมบูรณ์แล้ว

4.2.4 การนำเข้าวัตถุมาวางในฉาก

เมื่อได้แผ่นพื้นดิน (Terrain) ที่ปรับแต่งเรียบร้อยแล้ว ขั้นตอนถัดไปคือการนำวัตถุเข้ามาวางบนพื้นดินเพื่อประกอบฉาก ในโครงงานชิ้นนี้จะทำการสร้างวัตถุ 3 มิติด้วยโปรแกรม Autodesk 3ds Max แล้วนำมา Export แปลงเป็นไฟล์สกุล .x ผ่านปลั๊กอิน "Panda DirectX" ซึ่งเป็นไฟล์ที่มีคุณสมบัติที่ DirectX สามารถอ่านและนำคุณสมบัติจากโมเดลที่ปั้นในโปรแกรม Autodesk 3ds Max อันได้แก่ลักษณะTexture คุณสมบัติของBone หรือแม้แต่อนิเมชั่นของโมเดลมาใช้ได้ทันทีผ่าน method ที่รองรับ

ดังนั้นในการวางวัตถุ 3 มิติจากภายนอกเข้ามาในฉาก จึงใช้วิธีการอ่านไฟล์ทั่วไป โดยเริ่มต้นจากการกำหนด ID และตำแหน่งของไฟล์ .x ผ่านไฟล์ข้อมูลพิเศษที่สร้างขึ้นมา เรียกว่า ModelInfo.pss ซึ่งเป็นสกุลไฟล์ที่ตั้งขึ้นมาเอง สามารถปรับแต่งได้ผ่าน Notepad ทันที ทำให้ง่ายต่อการเพิ่มโมเดลเข้าไปในตัวโปรแกรมผ่านไฟล์ Object.pss

การอ่านไฟล์ข้อมูลดังกล่าว ใช้เมธอด FileReader::Read ของไฟล์ Frame.cpp

```
FileEntity* FileReader::Read(std::string filename)
{
  const char *cTemp = filename.c_str();
  if(filename == "")
     throw string("empty file path");
  FILE *file = 0;
  int error = fopen_s(&file, cTemp, "r");
  if(error)
  {
     std::string sTemp = "Cannot open file \"" + filename + "\"";
     MessageBox(0, sTemp.c_str(), "Error", MB_OK);
     exit(32);
  }
  ULinkedList<char*> *line = new ULinkedList<char*>(), *pointer = line;
// เมธอด feof เป็นเมธอดใช้ตรวจสอบว่าอ่านไฟล์ที่ส่งเข้ามาทั้งหมดแล้วหรือไม่
  while(!feof(file))
  {
     pointer->data = new char[500];
     fgets(pointer->data, 500, file);
     if(!feof(file))
     {
        int len = strlen(pointer->data);
        pointer->data[len-1] = '\0';
        pointer->next = new ULinkedList<char*>();
        pointer = pointer->next;
        pointer->next = 0;
     }
  }
  pointer = line;
  fclose(file);
  FileEntity *entity = new FileEntity(line, filename);
  return entity;
```

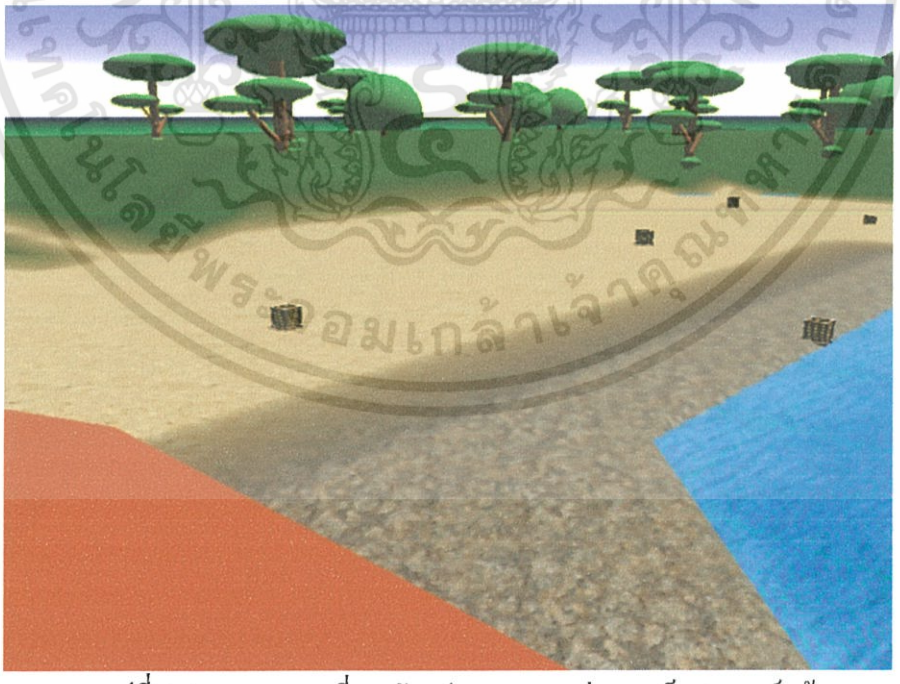
```
}
PSSFileEntity::PSSFileEntity(FileEntity *entity)
{
  name = entity->GetName();
  path = entity->GetPath();
  ULinkedList<char*> *pt = entity->GetContent();
  node = new ULinkedList<PSSNode>();
  ULinkedList<PSSNode> *pointer = node;
  pointer->data.name = "";
  ULinkedList<PSSContent> *content = 0;
  bool underquote = false;
  bool uq = false;
  bool onleft = true;
  unsigned int line = 0;
  while(pt != 0)
  {  line++;
    for(int i = 0; pt->data[i] != '\0'; i++)
    {
      if(!underquote)
      {
        if(pt->data[i] == '{')
          underquote = true;
        else
          pointer->data.name += pt->data[i];
      }
      else
      {
        if(!uq)
        {
          if(pt->data[i] == ';')// if found ; then end statement
          {
            if(onleft)
```

```
        {
            MessageBox(0, "Expect ; before beginning new content", "ERROR", MB_OK);
           exit(833);
        }
        else
        {
            onleft = true;
            content->next = new ULinkedList<PSSContent>();
            content = content->next;
        }
        continue;
    }
    else if(pt->data[i] == '=')
    {
        onleft = false;
        continue;
    }
    else if(pt->data[i] == '}')// if found } then end node
    {
        onleft = true;
        underquote = false;
        if(content)
        {
            content->next = 0;
            ULinkedList<PSSContent> *lTemp = pointer->data.content;
            while(lTemp->next != 0 && lTemp->next != content) lTemp = lTemp->next;
            delete lTemp->next;
            lTemp->next = 0;
            content = 0;
        }
        pointer->next = new ULinkedList<PSSNode>();
        pointer = pointer->next;
```

```
            pointer->next = 0;
            continue;
        }
        else if(pt->data[i] == '{')
        {
            string err = "Unexpected '{' in file " + path + "/" + name + " line " + ToString(line);
            MessageBox(0, err.c_str(),"ERROR", MB_OK);  exit(889);
        }
    }
    if(!pointer->data.content)
    {//initialize data.content
        pointer->data.content = new ULinkedList<PSSContent>();
        content = pointer->data.content;
    }
    if(pt->data[i] == "")
    {//หากพบเครื่องหมาย "
        uq = !uq;
        if(i > 0) if(pt->data[i-1] != '\\') continue;
    }
    if(pt->data[i] != '\\')
    {
        if(onleft)
        {
            if(pt->data[i] != ' ' && pt->data[i] != '	')
                content->data.name += pt->data[i];
        }
        else
        {
            if((pt->data[i] != ' ' && pt->data[i] != '	') || uq)
                content->data.content += pt->data[i];
        }
    }
```

```
        }
    }
    if(uq)
    {
        MessageBox(0, string("Expect \" before end of line in file " + path + "/" + name + " line " +
ToString(line)).c_str(), "ERROR", MB_OK);  exit(788);
    }
    pt = pt->next;
  }
  ULinkedList<PSSNode> *nTemp = node;
  while(nTemp->next != 0 && nTemp->next != pointer) nTemp = nTemp->next;
  delete nTemp->next;
  nTemp->next = 0;
}
```

ซึ่งในจุดนี้ก็จะได้ฉากเสร็จสมบูรณ์แล้วรูปที่ 4-6 แสดงฉากชายหาดที่ตกแต่งเรียบร้อยแล้ว ส่วนถัดไปคือกรรมวิธีในการแต่งฉากให้มีความสวยงามมากขึ้น รวมไปถึงลดการใช้ทรัพยากรในบางจุดลง และเตรียมการสำหรับระบบเกมที่จะเพิ่มเข้ามาต่อไป



รูปที่ 4-6 ฉากชายหาดที่วางวัตถุประกอบฉากต่างๆ เสร็จสมบูรณ์แล้ว

4.2.5 การให้แสงสี (Light)

แสงสีในฉาก 3 มิติทำให้วัตถุต่างๆ มีความสว่างและสีแตกต่างกันไป ซึ่งมีการทำงานอยู่ส่วนของเมธอด Scene3D::GetCurrentSunlightDirection ในไฟล์ Frame.cpp

```
D3DXVECTOR3 Scene3D::GetCurrentSunlightDirection(float currenttime)
{
  Time time = eventcenter->GetGameTime();
  if(time.hour == 5)
  {
    return INTERPOLATE(sunlightvec[3], sunlightvec[0], (currenttime-5));
  }
  else if(time.hour >= 6 && time.hour < 12)
  {
    return INTERPOLATE(sunlightvec[0], sunlightvec[1], (currenttime-6)/6.0f);
  }
  else if(time.hour >= 12 && time.hour < 18)
  {
    return INTERPOLATE(sunlightvec[1], sunlightvec[2], (currenttime-12)/6.0f);
  }
  else if(time.hour == 18)
  {
    return INTERPOLATE(sunlightvec[2], sunlightvec[3], (currenttime-18));
  }
  else
  {
    return sunlightvec[3];
  }
}
D3DXVECTOR4 Scene3D::GetCurrentSunlightColor(float currenttime)
{
  Time time = eventcenter->GetGameTime();
  if(time.hour == 5)
```

```
    {
        return INTERPOLATE(sunlightcol[3], sunlightcol[0], (currenttime-5));
    }
    else if(time.hour >= 6 && time.hour < 9)
    {
        return INTERPOLATE(sunlightcol[0], sunlightcol[1], (currenttime-6)/3.0f);
    }
    else if(time.hour >= 9 && time.hour < 15)
    {
        return sunlightcol[1];
    }
    else if(time.hour >= 15 && time.hour < 18)
    {
        return INTERPOLATE(sunlightcol[1], sunlightcol[2], (currenttime-15)/3.0f);
    }
    else if(time.hour == 18)
    {
        return INTERPOLATE(sunlightcol[2], sunlightcol[3], (currenttime-18));
    }
    else
    {
        return sunlightcol[3];
    }
}
```

4.2.6 การให้เงา (Shadow)

เมื่อกำหนดจุดกำเนิดแสงเรียบร้อยแล้ว จะได้ฉากที่วัตถุมีลักษณะมองดูมีความลึกความนูน แต่วัตถุจะไม่มีเงาพาดตกลงมาให้เห็นบนพื้นดิน โครงงานชิ้นนี้จึงนำเอาการสร้างเงาแบบ Variance Shadow Mapping มาประยุกต์ใช้งานอยู่ในส่วนหนึ่งของไฟล์ Light.fx ซึ่งมีรายละเอียดและวิธีการทำงานดังนำเสนอไปแล้วในบทที่ 2 หัวข้อที่ 2.6 แผนที่เงาแบบคำนวณค่าความคลาดเคลื่อนเอง (Variance Shadow Map)

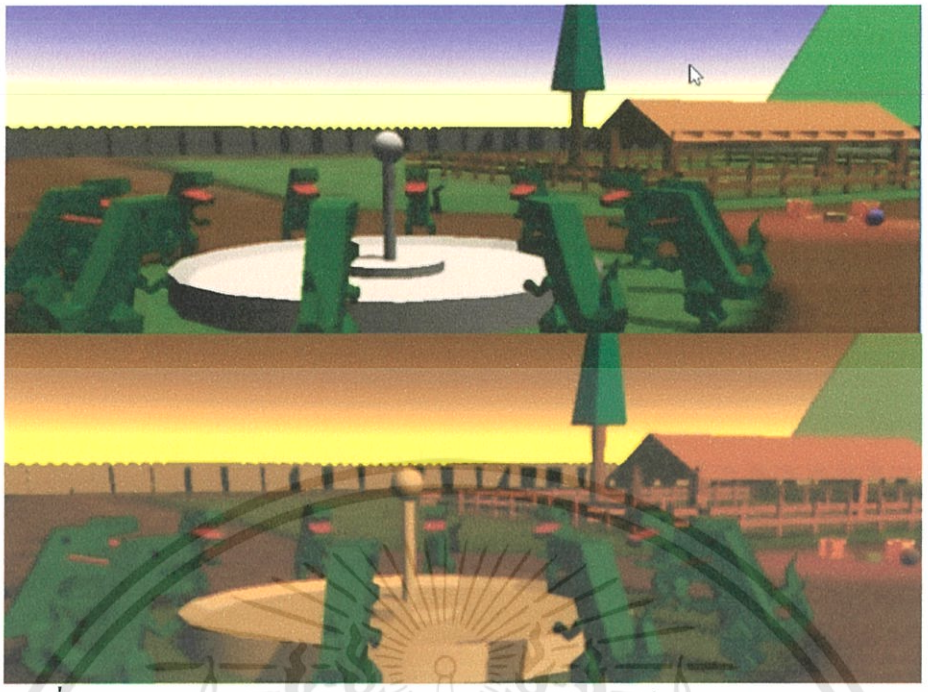
4.2.7 การให้แสงแบบ Real Time

ในจุดนี้จะเป็นส่วนที่นำไปประยุกต์ใช้กับระบบเวลาภายในเกม ทำหน้าที่ในการปรับแต่งระดับความเข้มแสงเมื่อเวลาผ่านไป จากสีหนึ่ง ไปเป็นสีหนึ่ง โดยให้โปรแกรมทำการเกลี่ยสีระหว่างสีเริ่มต้น ไปยังสีปลายทาง ในระยะเวลาที่กำหนดด้วยเมธอด Scene3D::GetCurrentSunlightDirection และ Scene3D::GetCurrentSunlightColor ตามลำดับ ในไฟล์ Frame.cpp

```
D3DXVECTOR3 Scene3D::GetCurrentSunlightDirection(float currenttime)
{
  Time time = eventcenter->GetGameTime();
  if(time.hour == 5)
  {
    return INTERPOLATE(sunlightvec[3], sunlightvec[0], (currenttime-5));
  }
  else if(time.hour >= 6 && time.hour < 12)
  {
    return INTERPOLATE(sunlightvec[0], sunlightvec[1], (currenttime-6)/6.0f);
  }
  else if(time.hour >= 12 && time.hour < 18)
  {
    return INTERPOLATE(sunlightvec[1], sunlightvec[2], (currenttime-12)/6.0f);
  }
  else if(time.hour == 18)
  {
    return INTERPOLATE(sunlightvec[2], sunlightvec[3], (currenttime-18));
  }
  else
  {
    return sunlightvec[3];
  }
}
D3DXVECTOR4 Scene3D::GetCurrentSunlightColor(float currenttime)
{
```

```
Time time = eventcenter->GetGameTime();
if(time.hour == 5)
{
   return INTERPOLATE(sunlightcol[3], sunlightcol[0], (currenttime-5));
}
else if(time.hour >= 6 && time.hour < 9)
{
   return INTERPOLATE(sunlightcol[0], sunlightcol[1], (currenttime-6)/3.0f);
}
else if(time.hour >= 9 && time.hour < 15)
{
   return sunlightcol[1];
}
else if(time.hour >= 15 && time.hour < 18)
{
   return INTERPOLATE(sunlightcol[1], sunlightcol[2], (currenttime-15)/3.0f);
}
else if(time.hour == 18)
{
   return INTERPOLATE(sunlightcol[2], sunlightcol[3], (currenttime-18));
}
else
{
   return sunlightcol[3];
}
}
```

เมื่อทำการปรับแสงสีได้เรียบร้อยแล้ว จะได้ผลแสงสีของฉากแตกต่างกันตามช่วงเวลาที่ผ่านไป ดังแสดงเปรียบเทียบในรูปที่ 4-7



รูปที่ 4-7 เปรียบเทียบแสงสีในตอนกลางวัน (บน) และตอนเย็น (ล่าง) ในฉากทดสอบ

4.2.8 การประยุกต์ใช้การคัดสรรวัตถุผ่านมุมกล้อง (View Culling)

เมื่อได้ฉากที่ปรับแต่งแสงสีและเงาเรียบร้อยแล้ว สิ่งถัดไปที่ต้องตรียมคือการจัดการทรัพยากร ให้โปรแกรมใช้ทรัพยากรที่จะวาดภาพลดลง เพราะฉากแต่ละฉากจะต้องถูกวาดใหม่ทุกเฟรม ซึ่งในจุดนี้จะเป็นการทำ View Culling มาประยุกต์ใช้ โดยมีรายละเอียดส่วนของเมธอด MGE_ViewFrustrumCull::cull จากไฟล์ base.cpp ดังนี้

```
//fPlane เป็นอาร์เรย์ของ D3DXPLANE 6 อัน ซึ่งเป็นด้านหนึ่งๆของขอบด้านต่างๆของกล้อง
bool MGE_ViewFrustrumCull::cull(D3DXPLANE *fPlane, D3DXVECTOR3 sphereCenter, float
sphereRadius)
{
  for(int i = 0; i< 6; i++)
  {
    if(D3DXPlaneDotCoord(&fPlane[i], &sphereCenter) + sphereRadius < 0)
    return false;
  }
  return true;
}
```

```
//viewproj คือ Matrix ผลรวมของ view matrix และ projection matrix ของกล้องปัจจุบัน
void MGE_ViewFrustrumCull::calculateFrustrumPlane(D3DXPLANE *fPlane, D3DXMATRIX
viewproj)
{
  D3DXMATRIX projection = viewproj;
  fPlane[0].a = projection._14 + projection._11;
  fPlane[0].b = projection._24 + projection._21;
  fPlane[0].c = projection._34 + projection._31;
  fPlane[0].d = projection._44 + projection._41;

  fPlane[1].a = projection._14 - projection._11;
  fPlane[1].b = projection._24 - projection._21;
  fPlane[1].c = projection._34 - projection._31;
  fPlane[1].d = projection._44 - projection._41;

  fPlane[2].a = projection._14 - projection._12;
  fPlane[2].b = projection._24 - projection._22;
  fPlane[2].c = projection._34 - projection._32;
  fPlane[2].d = projection._44 - projection._42;

  fPlane[3].a = projection._14 + projection._12;
  fPlane[3].b = projection._24 + projection._22;
  fPlane[3].c = projection._34 + projection._32;
  fPlane[3].d = projection._44 + projection._42;

  fPlane[4].a = projection._13;
  fPlane[4].b = projection._23;
  fPlane[4].c = projection._33;
  fPlane[4].d = projection._43;

  fPlane[5].a = projection._14 - projection._13;
  fPlane[5].b = projection._24 - projection._23;
```

```
  fPlane[5].c = projection._34 - projection._33;
  fPlane[5].d = projection._44 - projection._43;

  for(int i = 0; i< 6; i++) D3DXPlaneNormalize(&fPlane[i], &fPlane[i]);
}
```

4.2.9 การเปลี่ยนฉาก

การเปลี่ยนฉากที่แสดงนั้น ก็เปรียบได้ดังการเปลี่ยนเอาพื้นดินและวัตถุต่างๆ ทั้งหมดในฉากเดิมออกไป และเปลี่ยนเอาฉากใหม่ที่มีคุณลักษณะต่างกันออกมาแสดง โค้ดที่จัดการการเปลี่ยนฉากต้องทำการล้างข้อมูลฉากเดิมออกก่อนด้วยเมธอด Scene3D::Flush ในไฟล์ Frame.cpp

```
void Scene3D::Flush()
{
  loaded = false;
  onenter.clear();
  onleave.clear();
  LinkedList<Object*> *pointer = object;
  while(pointer)
  {
    if(pointer->data)
    {
      pointer->data->FlushModel();
      delete pointer->data;
    }
    pointer = pointer->next;
  }
  if(object) delete object;
  object = null;
  LinkedList<MGE_Model*> *mPointer = instancepool;
  while(mPointer)
  {
    base->FlushModel(mPointer->data);
```

```
    mPointer = mPointer->next;
}
if(instancepool) delete instancepool;
instancepool = null;

LinkedList<Effect*> *ePointer = effect;
while(ePointer)
{
    if(ePointer->data)
        delete ePointer->data;
    ePointer = ePointer->next;
}
if(effect) delete effect;
effect = null;

LinkedList<SpawnPoint*> *sPointer = spawnpoint;
while(sPointer)
{
    delete sPointer->data;
    sPointer = sPointer->next;
}

LinkedList<Effect*> *cPointer = cache;
while(cPointer)
{
    if(cPointer->data)
    {
        Effect *ef = (Effect*)cPointer->data;
        ef->FlushModel();
        delete ef;
        cPointer->data = null;
    }
```

```
    cPointer = cPointer->next;
  }
  if(cache) delete cache;
  cache = null;

  if(spawnpoint) delete spawnpoint;
  spawnpoint = null;

  if(objtodelete) delete objtodelete;
  objtodelete = null;

  if(effecttodelete) delete effecttodelete;
  effecttodelete = null;

  terrain = null;
  base->FlushTerrain();
  base->FlushSkybox();
  base->FlushLight();
}
```

นอกเหนือจากนั้นเป็นการปรับเปลี่ยนข้อมูลเกี่ยวกับระบบเกม ซึ่งจะอธิบายในส่วนของการนำการออกแบบมาประยุกต์ต่อไป

4.3 การบังคับตัวละครและการควบคุมผ่านอุปกรณ์นำเข้า

ในหัวข้อนี้ จะทำการอธิบายในส่วนที่เกี่ยวข้องกับการติดต่อสื่อสารกับผู้เล่น ทำให้ผู้เล่นสามารถป้อนข้อมูลเข้ามาในเกมได้ผ่านอุปกรณ์นำเข้าต่าง ๆ อันประกอบไปด้วยคีย์บอร์ดและเมาส์ กับ Wiimote

4.3.1 การควบคุมตัวละครผ่านคีย์บอร์ดและเมาส์

เมื่อสามารถกำหนดผู้เล่นและมุมกล้องได้แล้ว ถัดมาที่จะต้องทำก็คือ ให้ผู้เล่นสามารถบังคับตัวละครได้ผ่านอุปกรณ์นำเข้า ซึ่งเกมนี้มีคุณสมบัติของการรับข้อมูลนำเข้าแตกต่างกันไป สำหรับในส่วนของการควบคุมด้วยคีย์บอร์ดและเมาส์ จะมีการอ่านข้อมูลการกำหนดปุ่มจากไฟล์ภายนอกโดยใช้ชื่อว่า Control.pss โดยใช้เมธอด ControlCenter::Load ในไฟล์ EventCenter.cpp โดยเมธอดนี้จะมีการจัดการค่าที่ได้จาก Wiimote ด้วย ซึ่งรายละเอียดของการเชื่อมต่อกับ Wiimote นั้นจะแสดงในหัวข้อถัดไป

```
void ControlCenter::Load(PSSFileEntity *entity)
{
  PSSNode node = entity->GetNodeByIndex(0);
  LinkedList<ControlInfo> *pointer = action;
  for(int i = 1; node.name.compare("") != 0; i++)
  {
    if(action)
    {
      pointer->next = new LinkedList<ControlInfo>();
      pointer = pointer->next;
    }
    else
    {
      action = new LinkedList<ControlInfo>();
      pointer = action;
    }
    Wiimote *wii = input->GetWiimote();

    pointer->data.name = node.name;
    if(node.GetContentByName("keyboard").name.compare("") != 0)
    {
      string c = node.GetContentByName("keyboard").content;
      if(c.length() == 1)
      {
        pointer->data.keyboard = c.at(0);
      }
      else
      {
//VK_SHIFT/VK_CONTROL/VK_TAB/VK_ESCAPE/VK_SPACE เป็นค่าแทนปุ่ม Shift, Ctrl ทั้งซ้าย
และขวา, Tab, Esc และ Spacebar ตามลำดับ
        if(c.compare("shift") == 0) pointer->data.keyboard = VK_SHIFT;
        else if(c.compare("ctrl") == 0) pointer->data.keyboard = VK_CONTROL;
```

```
            else if(c.compare("tab") == 0) pointer->data.keyboard = VK_TAB;
            else if(c.compare("esc") == 0) pointer->data.keyboard = VK_ESCAPE;
            else if(c == "spacebar") pointer->data.keyboard = VK_SPACE;
        }
    }
    if(node.GetContentByName("wii").name.compare("") != 0)
    {
        int d = 0;
        string c = node.GetContentByName("wii").content;
        if(!IsConvertibleToInt(c, &d))
        {
            pointer->data.wii = ConvertToWiimoteKey(c);
        }
        else pointer->data.wii = (WiimoteKey)d;
    }
    if(node.GetContentByName("mouse").Exist())
    {
        pointer->data.mouse = ConvertToInt(node.GetContentByName("mouse").content);
    }
    node = entity->GetNodeByIndex(i);
  }
  delete entity;
}
```

จากนั้น เมื่อทำการกำหนดปุ่มเรียบร้อยแล้ว จึงนำมาใช้งาน โดยทำการตรวจสอบสถานะของปุ่มว่าได้ถูกใช้งานแล้วหรือไม่ โดยมีเมธอด JustPressKey ในการตรวจสออบว่า ปุ่มที่กำหนดเพิ่งถูกกดหรือไม่ และเมธอด PressingKey ในการตรวจสอบสถานะปัจจุบันหลังกดไปแล้ว และ JustReleaseKey ตรวจสอบว่าปุ่มถูกปล่อยแล้วหรือไม่ เพื่อยกเลิกการควบคุม

```
bool ControlCenter::JustPressKey(string name)
{
  LinkedList<ControlInfo> *pointer = action;
```

```
//GetWiimote() เป็นเมธอดที่ส่งตัวแทนวีโมทมาให้
  Wiimote *wii = input->GetWiimote();
  while(pointer)
  {
    if(pointer->data.name.compare(name) == 0)
    {
      bool result = false;
      result = input->JustPressKey(pointer->data.keyboard);
      if(input->isJoyStickConnected()) result |= input->isJoyButtonPressed(pointer->data.joystick);
      if(input->GetWiimote()->IsConnected())
      {
        result |= input->JustPressWiimote(pointer->data.wii);
      }
      if(pointer->data.mouse >= 0)
      {
        result |= input->JustPressMouse(pointer->data.mouse);
      }
      return result;
    }
    pointer = pointer->next;
  }
  return false;
}
bool ControlCenter::PressingKey(string name)
{
  LinkedList<ControlInfo> *pointer = action;
  Wiimote *wii = input->GetWiimote();
  while(pointer)
  {
    if(pointer->data.name.compare(name) == 0)
    {
      bool result = false;
```

```
            result = input->isKeyPressed(pointer->data.keyboard);
            if(input->isJoyStickConnected()) result |= input->isJoyButtonPressed(pointer->data.joystick);
            if(input->GetWiimote()->IsConnected())
            {
               result |= input->PressingWiimote(pointer->data.wii);
            }
            if(pointer->data.mouse >= 0)
            {
               result |= input->isMousePressed(pointer->data.mouse);
            }
            return result;
         }
         pointer = pointer->next;
      }
      return false;
}
bool ControlCenter::JustReleaseKey(string name)
{
      LinkedList<ControlInfo> *pointer = action;
      Wiimote *wii = input->GetWiimote();
      while(pointer)
      {
         if(pointer->data.name.compare(name) == 0)
         {
            bool result = false;
            result = input->JustReleaseKey(pointer->data.keyboard);
            if(input->GetWiimote()->IsConnected())
            {
               result |= input->JustReleaseWiimote(pointer->data.wii);
            }
            if(pointer->data.mouse >= 0)
            {
```

```
            result |= input->JustReleaseMouse(pointer->data.mouse);
        }
        return result;
    }
    pointer = pointer->next;
  }
  return false;
}
```

4.3.2 การควบคุมตัวละครผ่าน Wiimote

ในส่วนของการรองรับข้อมูลนำเข้าจาก Wiimote นั้นจำเป็นต้องมีคลาสที่ใช้ในการรองรับโดยเฉพาะ โดยมีวิธีการเริ่มต้นจากการตรวจสอบสถานะการเชื่อมต่อของ Wiimote และนำข้อมูลที่ได้จากตัว Wiimote เข้ามาประมวลผล ซึ่งโค้ดของเมธอด WiiAPI นี้จะคัดมาจาก Wiimote library ชื่อว่า WiiYourself! ส่วนหนึ่ง เนื่องจากโปรแกรมนี้ไม่ได้ใช้ทุกคุณลักษณะของ Wiimote เพราะบางคุณสมบัตินั้นไม่สามารถใช้เมาส์และคีย์บอร์ดในการหาค่าได้ โดยก่อนที่จะสามารถพัฒนาในส่วนนี้ได้ จำเป็นต้องมีการปรับแต่งคุณลักษณะบางอย่างของโปรเจค และเชื่อมต่อกับ Wiimote ให้ได้ก่อน รายละเอียดส่วนนี้จะปรากฏในส่วนของภาคผนวกต่อไป

สำหรับรายละเอียดเมธอดที่ใช้งาน WiiAPI อยู่ในไฟล์ EventCenter.pss ดังนี้

```
void WiiAPI(Wiimote &remote, state_change_flags changed, const Wiimote_state &new_state)
{
  EventCenter *eventcenter = EventCenter::GetInstance();
  MGE *base = eventcenter->GetEngine();
  Frame *frame = Frame::GetInstance(base, eventcenter);
//ถ้าเพิ่งต่อสำเร็จ ให้ติดตามการเปลี่ยนแปลงของ Accelerator และค่าจากหลอด IR ด้วย
  if(changed & CONNECTED)
  {
    if(new_state.ExtensionType != Wiimote::BALANCE_BOARD)
    {
      if(new_state.bExtension)
        remote.SetReportType(Wiimote::IN_BUTTONS_ACCEL_IR_EXT);
      else
```

```
            remote.SetReportType(Wiimote::IN_BUTTONS_ACCEL_IR);
        }
    }
    base->GetInputPointer()->UpdateWiimoteBuffer();
//ถ้าสถานะปุ่มมีการเปลี่ยนแปลงให้ทำการอัพเดทฉากด้วย
    if(changed != NO_CHANGE)
    {
        Wiimote_state::ir::dot d = remote.IR.Dot[0];
        if(d.bVisible)
        {
            int mouseX = (int)(d.X * Global::WindowSize.x) + Global::WindowPosition.x,
                mouseY = (int)(d.Y * Global::WindowSize.y) + Global::WindowPosition.y;
SetCursorPos(mouseX, mouseY);
            base->GetInputPointer()->UpdateMousePosition();
        }
        if(!eventcenter->GetEngine()->BreakingMainLoop())
            eventcenter->GetGameSystem()->UpdateInput(base, frame, base->GetInput());
        frame->UpdateInput(base->GetInput());
    }
}
```

4.3.3 การเปลี่ยนตำแหน่งกล้องในมุมมองบุคคลที่ 3

เมื่อได้สามารถรับข้อมูลนำเข้าจากอุปกรณ์นำได้แล้ว จึงเริ่มต้นในส่วนของผู้เล่น หรือก็คือวัตถุ 3 มิติที่เป็นตัวแทนผู้เล่น ให้สามารถเคลื่อนไหวได้อยู่ภายในฉาก 3 มิติ

มุมมองของเกมแต่ละเกมนั้นจะมีรูปแบบแตกต่างกันไป ในเกมนี้จะใช้มุมมองบุคคลที่ 3 ซึ่งเป็นมุมมองที่มองจากผู้เล่นจากหน้าจอ เปรียบเหมือนกำลังตามหลังตัวละครภายในฉากอยู่ ดังนั้นในการเคลื่อนที่ของตัวละคร นอกจากจะเปลี่ยนตำแหน่งของตัวละครแล้ว ก็จะต้องเปลี่ยนมุมกล้องให้สอดคล้องกันไปด้วย โดยหลักก็คือการปรับมุมกล้องให้ติดตามผู้เล่นจากมุมที่กำหนด โดยมีโค้ดที่ใช้ในการเลื่อนมุมกล้องติดตามผู้เล่นด้วยเมธอด CameraMode_Follow::Update ในไฟล์ EventCenter.cpp

```
bool CameraMode_Follow::Update(Camera *cam, MGE *base, EventCenter *eventcenter)
{
  Frame *frame = eventcenter->GetFrame();
  ControlCenter *control = eventcenter->GetGameSystem()->GetControl();
  if(target)
  {
    base->setCameraAttribute(GAMECAMERA_PIVOTX, target->GetLocation().x, frame-
>GetEffectFileArray(), frame->GetEffectFileArrayLength());

    base->setCameraAttribute(GAMECAMERA_PIVOTY, target->GetLocation().y + 50.f, frame-
>GetEffectFileArray(), frame->GetEffectFileArrayLength());

    base->setCameraAttribute(GAMECAMERA_PIVOTZ, target->GetLocation().z, frame-
>GetEffectFileArray(), frame->GetEffectFileArrayLength());
  }
  float oanglex = (float)base->getCameraAttribute(GAMECAMERA_ANGLEX);
  float oangley = (float)base->getCameraAttribute(GAMECAMERA_ANGLEY);
  float anglex = oanglex;
  float angley = oangley;
  if(control->PressingKey("MoveCamRight")) anglex += 5;
  if(control->PressingKey("MoveCamLeft")) anglex -= 5;
  if(control->PressingKey("MoveCamUp")) angley += 5;
  if(control->PressingKey("MoveCamDown")) angley -= 5;
  if(anglex != oanglex || angley != oangley)
  {
    base->setCameraAttribute(GAMECAMERA_ANGLEX, anglex, frame->GetEffectFileArray(),
frame->GetEffectFileArrayLength());
    base->setCameraAttribute(GAMECAMERA_ANGLEY, angley, frame->GetEffectFileArray(),
frame->GetEffectFileArrayLength());
  }
  if(control->PressingKey("ZoomIn"))
  {
```

```
    float zoom = (float)base->getCameraAttribute(GAMECAMERA_EYELENGTH);
    zoom -= 5.f;
    if(zoom < 130.f) zoom = 130.f;
    base->setCameraAttribute(GAMECAMERA_EYELENGTH, zoom, frame->GetEffectFileArray(),
frame->GetEffectFileArrayLength());
  }
  else if(control->PressingKey("ZoomOut"))
  {
    float zoom = (float)base->getCameraAttribute(GAMECAMERA_EYELENGTH);
    zoom += 5.f;
    if(zoom > 410.f) zoom = 410.f;
    base->setCameraAttribute(GAMECAMERA_EYELENGTH, zoom, frame->GetEffectFileArray(),
frame->GetEffectFileArrayLength());
  }
  base->refreshCamera(eventcenter->GetFrame()->GetEffectFileArray(), eventcenter->GetFrame()-
>GetEffectFileArrayLength());
  return true;
}
```

หลังจากทำการปรับมุมมองแล้ว เมื่อผู้เล่นบังคับตัวละคร กล้องและตำแหน่งของตัวละครจะเคลื่อนที่ไปสอดคล้องกันดังรูปที่ 4-8 เปรียบเทียบภาพการเคลื่อนที่จากมุมทางซ้าย เดินไปทางขวา

รูปที่ 4-8 รูปภาพภายในเกมที่มีตัวละครและมุมมองแบบบุคคลที่ 3 แล้ว

4.3.4 การสร้างพื้นที่ Collision

บนพื้นดินนั้น บางจุดจะมีลักษณะที่เดินเข้าไปไม่ได้ พื้นที่ Collision ก็คือ พื้นที่ไม่สามารถเคลื่อนที่ผ่านเข้าไปได้ ซึ่งถือเป็นจุดเริ่มต้นพื้นฐานอันนำไปสู่การสร้างเนื้อเรื่อง เช่นจากจุดที่ยืนอยู่เป็นพื้นดิน และด้านหน้าเป็นภูเขาที่ชัน ก็จะไม่สามารถเดินทะลุเข้าไปได้ ผู้เล่นก็จะต้องหาทางลัดเลาะไปทางอื่น เป็นต้น ซึ่งในจุดนี้จะเป็นการปรับแต่งครั้งสุดท้ายให้กับฉาก

ในการสร้างพื้นที่ Collision นั้น จะใช้พื้นดินเป็นตัวอ้างอิง ซึ่งจะมีวิธีการสร้างโดยอ้างอิงจากไฟล์ภาพ มีวิธีการทำงานคล้ายกับการสร้าง Texture Map และ Height Map ดังที่กล่าวไปแล้ว โดยภาพที่ใช้นี้จะเรียกว่า Collision Map

รูปที่ 4-9 รูปภาพที่ใช้เป็น Collision Map ของฉากชายหาด

โดยถ้าเป็นสีเหลือง (ค่าสีแดง = 255 และ ค่าสีเหลือง = 255) พื้นที่นั้นจะเดินผ่านได้ หากเป็นสีแดงพื้นที่นั้นจะถูกกำหนดให้ไม่สามารถเดินผ่านได้ สำหรับโค้ดที่จัดการการสร้าง Collision Map อยู่ในไฟล์ Frame.cpp เป็นดังนี้

```
D3DXCreateTextureFromFileEx(base->Device(), node.GetContentByName(colisionmap).content.c_str(),
0, 0, 1, 0, D3DFMT_A8B8G8R8,D3DPOOL_MANAGED,
D3DX_FILTER_NONE,D3DX_FILTER_NONE, 0, 0, 0, &tTemp);
        tTemp->GetLevelDesc(0, &desc);
        tTemp->LockRect(0, &lRect, 0, 0);
        bPointer = (BYTE*)lRect.pBits;
        for(int i = 0; i< ms[0]-1; i++) for(int j = 0; j< ms[1]-1; j++)
        {
      /*
        หนึ่งพิกเซลใช้เนื้อที่สี่ช่องของอาร์เรย์ สำหรับช่องสีแดง เขียว น้ำเงิน และค่าความโปร่งใสของพิกเซลนั้นๆ
      */
          DWORD index=(i*4+(j*(lRect.Pitch)));
          int r = bPointer[index+2];
          int g = bPointer[index+1];
          if(r != 255 || g != 255)
```

```
            terrain->getPlate(i, j)->CustomHeight(r - g);
        }
        tTemp->UnlockRect(0);
        CUtility::DXRelease(tTemp);
```

4.4 การสร้างภาพ 2 มิติ วางบนฉาก 3 มิติ

การนำภาพ 2 มิติมาวางบนฉาก 3 มิตินั้นถือเป็นสิ่งที่ขาดไม่ได้เลยสำหรับเกม เพราะจะเป็นส่วนที่ผู้ใช้จะสามารถใช้ติดต่อและเข้าถึงระบบได้ ผ่านข้อความและภาพลักษณะ 2 มิติ ที่แสดงบนหน้าจอ รวมถึงปุ่มต่างๆ สามารถจัดการได้ง่าย อีกทั้งยังสามารถประมวลผลได้ซับซ้อนตามที่ต้องการ

4.4.1 การแสดงข้อความ 2 มิติ

ในการแสดงข้อความบนหน้าจอนั้น จะใช้เมธอด Text::DrawText ในไฟล์ Frame.cpp ซึ่งมีรายละเอียดดังต่อไปนี้

```
bool Text::DrawText(Frame *frame, MGE *base, void* param)
{
  if(visible)
  {
     int alpha = 255;
     if(scene)
     {
        alpha = (int)(scene->GetAlpha() * 255 * this->alpha);
     }
     DWORD color = D3DCOLOR_ARGB(alpha, red, green, blue);
     RECT r;
     if(rect.bottom == rect.top) ::SetRect(&r, (int)location.x, (int)location.y, 1024, 768);
     else r = rect;
  //สั่งให้ font ที่เก็บไว้ วาดตัวอักษรขึ้นหน้าจอ
     font->draw((char*)text.c_str(), r, color, align);
  }
  return true;
}
```

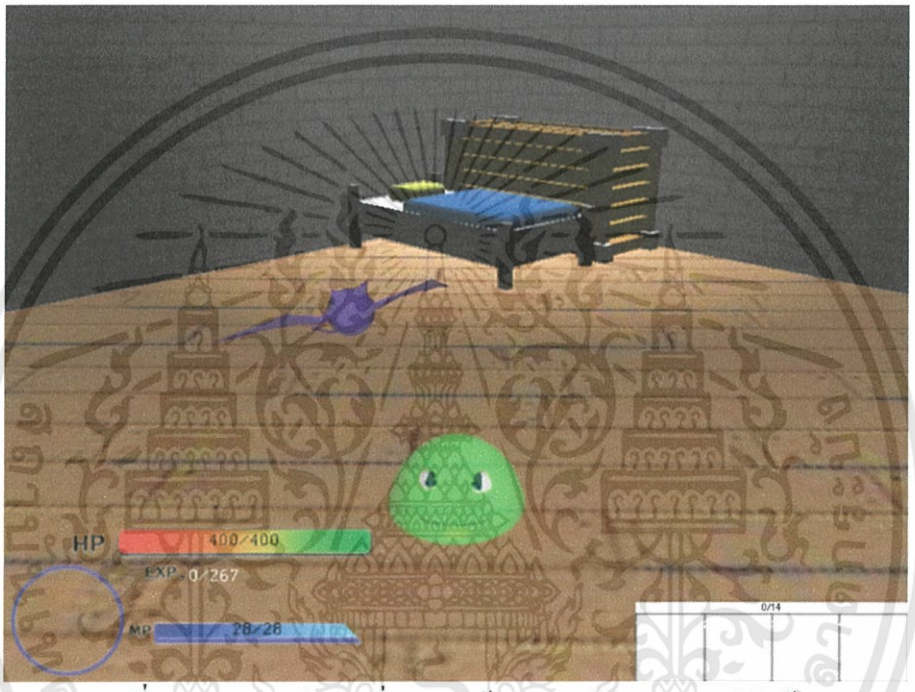
4.4.2 การแสดงภาพ 2 มิติ

ในส่วนนี้จะเป็นการแสดงภาพ 2 มิติจากไฟล์ภายนอก เช่นเดียวกับการเรียกวัตถุ 3 มิติที่ทำการอ่านข้อมูลจากไฟล์ .x เข้ามาในฉาก แต่สำหรับภาพ 2 มิติแล้ว จะสามารถเรียกเข้ามาแสดงผลได้โดยตรง ไม่จำเป็นต้องมีเมธอดหรือ ID ในการเรียก ข้อมูลที่ต้องการมีเพียงตำแหน่งที่จะให้แสดงบนหน้าจอ ขนาด และที่อยู่ของไฟล์ภาพเท่านั้น ซึ่งส่วนนี้จะจัดการผ่านเมธอด Image::Draw ในไฟล์ Frame.cpp โดยมีกระบวนการเพิ่มเติมที่ช่วยให้ใช้ทรัพยากรลดลงด้วย ดังนี้

```
bool Image::Draw(Frame *frame, MGE *base, void* param)
{
  if(visible && this->alpha > 0)
  {
/*แทนที่จะสร้างสี่เหลี่ยมเพื่อวาดภาพหลายๆภาพ จะเพียงใช้สี่เหลี่ยมเดียว และทำการปรับขนาดและย้ายตำแหน่งทุกครั้งที่มีการวาดใหม่*/
    MGE_Primitive3D *background = (MGE_Primitive3D*)param;
    D3DXMATRIX tran, rotx, roty, rotz, scale, world;
//ตำแหน่ง
    D3DXMatrixTranslation(&tran, GetLocation().x, GetLocation().y, GetLocation().z);
//ขนาด
    D3DXMatrixScaling(&scale, GetScale().x, GetScale().y, GetScale().z);
    D3DXMatrixRotationX(&rotx, GetRotation().x);
    D3DXMatrixRotationY(&roty, GetRotation().y);
    D3DXMatrixRotationZ(&rotz, GetRotation().z);

    LPD3DXEFFECT effect = frame->GetEffectFileArray()[0];
//รวมกันได้ world transform
    world = scale * rotx * roty * rotz * tran;
//ใส่ค่าเข้าไปใน Effect File
    effect->SetMatrix(xWorld, &world);
    effect->SetBool(HasTexture, image != null);
    D3DXVECTOR4 vec = D3DXVECTOR4(0, 0, 0, 0);
    if(scene->GetIndex() == 0)
    {
```

```
        if(Highlighted())
        {
            vec.x = 0.5;
            vec.y = 0.5;
            vec.z = 0.5;
        }
        else if(Darken())
        {
            vec.x = -0.5;
            vec.y = -0.5;
            vec.z = -0.5;
        }
    }
//ตั้งค่าต่างๆลงไปใน Shader
    effect->SetVector("additionalColor", &vec);
    effect->SetInt(ObjectType, ObjectType_2D);
    effect->SetFloat("objAlpha", this->alpha);
    if(image != null)
        effect->SetTexture(Texture, image->getTexture());
    effect->SetVector(ObjectMaterial, &D3DXVECTOR4(1, 1, 1, 1));
//การวาดโดยใช้ Effect File สามารถวาดได้หลายๆรอบก่อนที่จะย้ายไปวัตถุต่อไป
//บอก Shader ว่าเราจะใช้ชุดคำสั่งไหนในการประมวลผลพิกเซล
    if(SUCCEEDED(effect->SetTechnique(Shader3)))
    {
        unsigned int pass = 0;
        if(SUCCEEDED(effect->Begin(&pass, 0)))
        {
            for(unsigned int i = 0; i< pass; i++)
            {
                effect->BeginPass(i);
                background->draw(base, world);
                effect->EndPass();
```

```
            }
            effect->End();
        }
    }
  }
  return true;
}
```



รูปที่ 4-10 ภาพภายในเกมที่ทำการสร้างภาพ 2 มิติบนฉาก 3 มิติแล้ว

4.4.3 การประยุกต์ใช้: แสดงเวลาปัจจุบัน

จากในส่วนของการสร้างฉาก 3 มิติที่สามารถเปลี่ยนแปลงแสงสีตามจำนวนเฟรมที่เปลี่ยนไปได้แล้ว ในฉาก 2 มิติก็สามารถทำการเปลี่ยนแปลงค่าตัวเลขให้เป็นไปตามลักษณะของฉาก เพื่อบอกเวลาปัจจุบันได้เช่นกัน โดยจุดนี้จะถือเป็นตัวอย่างการใช้งานร่วมกันระหว่างฉาก 3 มิติ และข้อความ 2 มิตินั่นเอง

4.5 การนำการออกแบบมาประยุกต์ใช้

จากรายละเอียดการออกแบบรูปแบบเกมในบทที่ 3 ส่วนตั้งแต่หัวข้อที่ 3.1.5 เป็นต้นไป ได้ถูกนำมาพัฒนาเป็นโปรแกรมเกม แบ่งเป็นหัวข้อได้ดังต่อไปนี้

4.5.1 Non-Player Character (NPC)

NPC ในทางเทคนิคแล้ว ก็คือตัวละครหรือวัตถุใดๆ ที่มีความารถในการตอบสนองกับผู้เล่นได้ ในที่นี้ก็คือ วัตถุ 3 มิติภายในฉากนั่นเอง โดย NPC จะมีคุณสมบัติพิเศษต่างจากวัตถุทั่วไปประกอบไปด้วย การตอบสนองด้วยผู้เล่น การเคลื่อนที่ เริ่มบทสนทนา และการทำไฮไลท์เพื่อบ่งบอกระยะ

4.5.1.1 การตอบสนองกับ NPC

คุณสมบัติอย่างแรกที่ขาดไม่ได้เลยของ NPC ก็คือ การทำให้ผู้เล่นสามารถตอบสนองได้ ผ่านการป้อนข้อมูลเข้าไปในเกม สิ่งแรกที่ทำให้ผู้เล่นตอบสนองกับ NPC ได้ คือระยะที่ผู้เล่นสามารถทำการตอบสนองกับ NPC ตัวนั้นๆ สำหรับเกม 2 มิติ ก็คือการตรวจสอบว่า ผู้เล่นอยู่ในช่องที่ข้างเคียงกับผู้เล่นหรือไม่ สำหรับเกม 3 มิติก็คือ ระยะรอบตัว ที่มีจุดศูนย์กลางจากจุด ที่ NPC อยู่นั่นเอง

โดยในส่วนนี้มีเมธอด NPC::IntersectsInteraction ในการตรวจสอบว่าผู้เล่นเข้าใกล้ NPC ในระยะที่เพียงพอต่อการตอบสนองแล้วหรือไม่ รวมไปถึงการไฮไลท์ NPC ที่เกิดขึ้นเมื่อเข้ามาอยู่ในระยะที่กำหนด

```
bool NPC::IntersectsInteraction(D3DXVECTOR3 center, float radius)
{
//D3DXVec3Length เป็นเมธอดของ DirectX ที่จะคืนค่าระยะห่างระหว่างสองจุดของ Vector 3 มิติสองจุด
  if(interactionradius < 0 || !Visible()) return false;
  else return D3DXVec3Length(&(center - GetLocation())) <= interactionradius + radius;
}
void NPC::UpdateMainLoop(MGE *base, float *dist)
{
bool result = IntersectsInteraction(eventcenter->GetPlayer()->GetLocation(), eventcenter->GetPlayer()->GetColisionRadius());
  Highlighted(result);
}
```

4.5.1.2 การเคลื่อนที่ของ NPC

NPC โดยทั่วไปแล้วจะสามารถเคลื่อนที่ได้เพื่อทำให้ตัวละครดูมีชีวิตขึ้นมาได้ควบ คู่ไปกับ Animation ของตัวละครนั้นๆ เช่นการเดิน

โค้ดที่ใช้จัดการการเคลื่อนที่ของ NPC เป็นแบบที่เดินตามเส้นทางที่กำหนดไว้ ซึ่งสามารถต่อยอดไปใช้ในส่วนของศัตรูภายในเกมได้อีกด้วย โดยมีเมธอด NPCStatus_Move::Update ในการจัดการดังนี้

```
bool NPCStatus_Move::Update(Object *object, EventCenter *eventcenter, void* extraparam)
        {
        //Update() เป็นเมธอดที่อัพเดทรูปร่างวัตถุให้เคลื่อนไหวอย่างเหมาะสม
          model->Update();
          NPC *npc = (NPC*)object;
          MGE *base = eventcenter->GetEngine();
          npc->Move(npc->GetDestination(), base);
          if(npc->Intersects(npc->GetDestination()))
          {
            LinkedList<D3DXVECTOR3> *wp = npc->GetWayPoint();
            wp = wp->next;
            npc->SetWayPoint(wp);
            npc->SetStatus(npc->GetStatus(CharacterStatus::Idle));
          }
          return true;
        }
```

รูปที่ 4-11 การไฮไลท์ NPC พร้อมกับแสดงข้อความเกี่ยวกับ NPC เมื่อผู้เล่นเข้าใกล้

4.5.1.3 การแสดงกล่องข้อความและบทสนทนา

รูปแบบในการตอบสนองกลับมาของ NPC โดยทั่วไปแล้วจะแบ่งได้เป็นการแสดงข้อความ และอีกส่วนหนึ่งก็คือเริ่มบทสนทนา ซึ่งในจุดนี้จะสามารถแบ่งได้เป็นอีก 2 รูปแบบคือ การแสดงกล่องข้อความเป็นกล่องที่มีข้อความสั้นๆ และบทสนทนา ซึ่งจะแสดงเป็นข้อความที่สามารถกดเพื่ออ่านต่อไปจนกว่าจะหมดบทที่กำหนด

ในการสร้างกล่องข้อความจะแสดงผลในรูปแบบ 2 มิติ แสดงขึ้นมาเหนือหัวของ NPC ด้วยเมธอด ConversationBox มีรายละเอียดดังนี้

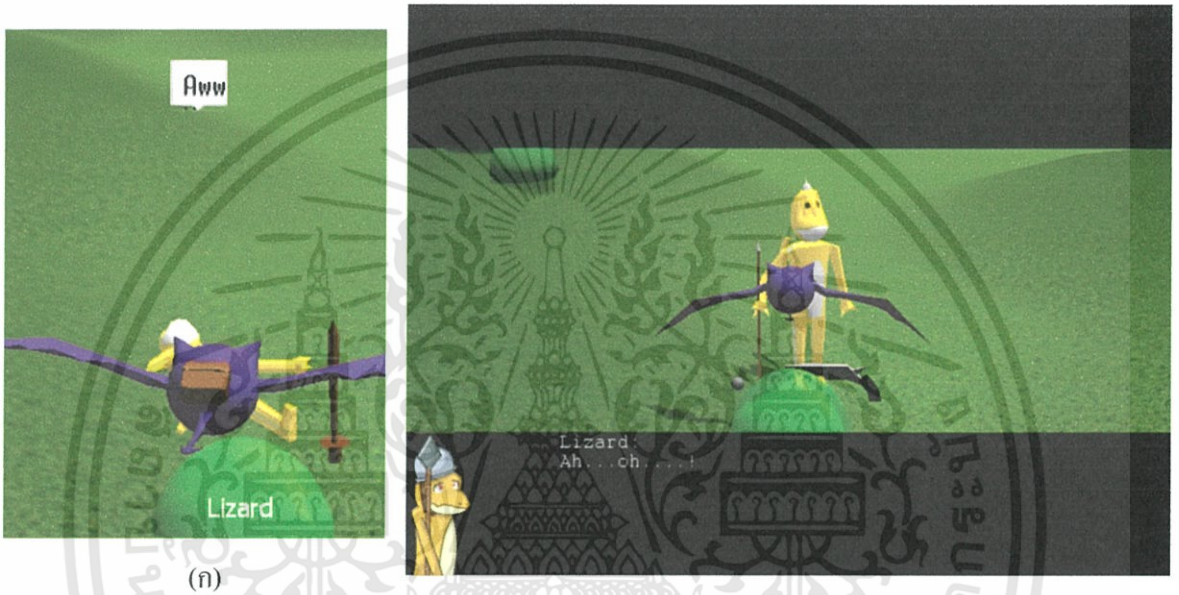
```
ConversationBox::ConversationBox(EventCenter *eventcenter, DataCenter *datacenter, std::string text,
Object *owner) : alpha(0), alphachange(0.1f), textCounter(0),Text(eventcenter)
        {
          type = ObjectType_DialogBox;
          this->text = text;
          this->owner = owner;
          Scene2D *gui = eventcenter->GetFrame()->GetScene2DByName("gui");
          end = eventcenter->GetGameTime() + SecondToFrame(1);
          SetScene(gui);
          SetAlignment(DT_LEFT | DT_WORDBREAK);
          SetFont(datacenter->GetFont(5));
          SetColor(0, 0, 0);
          SetText(text);
          rect = datacenter->GetFont(5)->CalculateRect(text, align);
          rect.right = rect.right - rect.left;
          rect.bottom = (rect.bottom - rect.top) * 2;
          rect.top = 0;
          rect.left = 0;
          background = new Image(eventcenter);
          background->SetScale(D3DXVECTOR3((float)rect.right, (float)rect.bottom, 1));
          background->SetTexture("Image/DialogBox.png", eventcenter->GetEngine());
          background->SetScene(gui);
          background->SetName("ConversationBox_Background");
          mat = owner->GetModel()->GetBoneMatrixByName("head");
```

```
        gui->AddObject(this);
    }
```

สำหรับโค้ดในการแสดงบทสนทนามีรายละเอียดดังนี้

```
    {
        ConversationWindowHandler *handler = (ConversationWindowHandler*)eventcenter-
>GetGameSystem()->GetConversationHandler();
        int id = 0;
    //IsConvertibleToInt เป็นเมธอดที่เช็กว่า String ที่ส่งไปสามารถแปลงเป็น int ได้หรือไม่ ถ้าได้ก็จะ
แปลงและส่งกลับมากับ pointer ที่เราส่งไป ถ้าไม่ก็จะไม่ทำอะไรกับ pointer ที่ส่งไป
        if(!IsConvertibleToInt((char*)command[1].c_str(), &id))
          id = ConvertToInt(HandleExpression(command[1], eventcenter, param1));
        handler->GetHandler()->Show();
        handler->GetHandler()->BringToFront();
        handler->SetConversation(datacenter->GetConversationInfo(id, 0));
    //ContinueScript เป็นเมธอดที่จะทำให้เกิดการ recursive ไปยังสคริปต์คำสั่งถัดไป
        return ContinueScript(eventcenter, command, 1, length, param1, param2);
    }
    void ConversationWindowHandler::SetConversation(ConversationInfo conversation)
    {
      this->conversation = conversation;
      dialog->SetText(conversation.speaker + ":\n");
      D3DXVECTOR3 location = image->GetLocation();
      if(conversation.picPosition.compare("left") == 0)
      {
        location.x = 0;
        image->SetLocation(location);
      }
      else if(conversation.picPosition.compare("right") == 0)
      {
        location.x    (float)dialog->GetRect().right;
```

```
            image->SetLocation(location);
        }
        textCounter = 0;
        eventcenter->RaiseScript(conversation.onEnter, null);
        image->SetTexture(conversation.image);
    }
```



(ก) (ข)

รูปที่ 4-12 (ก) การแสดงกล่องข้อความ (ข) ตัวอย่างหน้าต่างบทสนทนาภายในเกม

4.5.2 การประยุกต์ระบบสถานะ (Status)

ใน ส่วนของระบบสถานะนั้น จะเป็นส่วนที่เกี่ยวข้องโดยตรงกับเกม ถือเป็นระบบที่เป็นพื้นฐาน เพื่อจะนำไปต่อยอดกับระบบอื่นได้ จากรายละเอียดในบทที่ 3 จะใช้โค้ดเหล่านี้ในการสร้างระบบสถานะขึ้นมา แบ่งได้เป็น

การกำหนดค่าพลังเป็นส่วนที่เริ่มนำค่าพลังที่กำหนดจากช่วงที่แล้วมาประยุกต์ เตรียมไว้เพื่อรองรับในส่วนของความสามารถอื่นๆ รวมไปถึงการต่อสู้กับศัตรู ด้วยเมธอด Character::CalculateDamage มีรายละเอียดดังต่อไปนี้

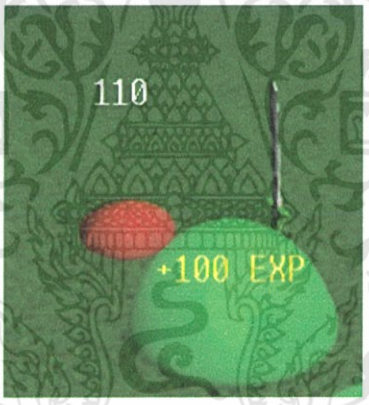
```
float Character::CalculateDamage(float val)
        {
          int atk = this->atk;
          if(atk < 0) atk = 0;

          if(this->GetEquipRight()) atk += this->GetEquipRight()->GetATK();
          if(this->GetEquipHead()) atk += GetEquipHead()->GetATK();
          if(GetEquipLeft()) atk += GetEquipLeft()->GetATK();
          atk = (int)(atk * ConvertToFloat(GetVariable("atkMultiplier")));

          float inidam = atk*2.5f * (1.f + (Random(-15, 30) / 100.0f));
          float mul = ConvertToFloat(GetVariable("damageMultiplier"));
          return mul * inidam + val;
        }
```

จากนั้นเมื่อสามารถคำนวณค่าความเสียหายได้แล้ว จึงนำมาทำการแสดงผลออกมาบนฉาก โดยใช้การแสดงตัวเลขที่มีสีต่างกันตามแต่สถานการณ์ ออกมาในตำแหน่งที่กำหนดเอาไว้ รวมไปถึงปรับปรุงค่าพลัง หรือก็คือ HP ให้ลดลงมาอยู่ในระดับที่ควรจะเป็น ด้วยเมธอด Character::DisplayDamage

```
void Character::DisplayDamage(float value, DWORD dmgcolor, DataCenter *datacenter)
        {
          D3DXVECTOR3 pos3d = GetLocation();
          pos3d.y += 30;
          eventcenter->GetFrame()->GetScene2DByName("gui")->AddObject(
            new DamageNumber(eventcenter, datacenter, (int)value, pos3d, dmgcolor));
        }
        DamageNumber::DamageNumber(EventCenter *eventcenter, DataCenter *datacenter, int
number, D3DXVECTOR3 startpos, DWORD color):
        alpha(1), alphachange(0), Text(eventcenter)
        {
          SetText(ToString<int>(number));
          SetLocation(startpos);
```

```
        end = Global::frame->GetCurrentFrame() + SecondToFrame(1);
        type = ObjectType_DamageNumber;
        SetFont(datacenter->GetFont(5));
        pos3d = startpos;
        int red = color & 0x00ff0000;
        int green = color & 0x0000ff00;
        int blue = color & 0x000000ff;
        SetColor(red, green, blue);
        SetScene(eventcenter->GetFrame()->GetScene2DByName("gui"));
        SetAlignment(DT_CENTER | DT_NOCLIP);
    }
```

นอกเหนือจากนี้จะเป็นส่วนของการประยุกต์ใช้ โดยหลักก็คือใช้งานร่วมกันกับระบบทักษะ ที่มีลักษณะเกี่ยวข้องกับการเปลี่ยนแปลงสถานะต่อไป



รูปที่ 4-13 การแสดงตัวเลขความเสียหาย (เลขสีขาว) ที่เกิดขึ้นเมื่อมีการโจมตีถูกเป้าหมาย พร้อมแสดง EXP ที่ได้จากการปราบศัตรูสำเร็จ (เลขสีเหลือง)

4.5.3 การประยุกต์ระบบไอเท็มและการจัดเก็บ

ไอเท็มเป็นสิ่งที่ช่วยอำนวยความสะดวกให้กับผู้เล่น ซึ่งส่งผลโดยตรงต่อระบบสถานะ ดังเช่น ใช้แล้ว HP จะเพิ่มขึ้น หรืออาจจะส่งผลพิเศษอื่นๆ โดยในการสะสมไอเท็มเข้ามาให้ผู้เล่นเก็บ ก็จะใช้วิธีการคล้างคลึงกับการจัดการกับ NPC คือจะมีสิ่งของที่สามารถตอบสนองได้ แล้วให้ผู้เล่นตอบสนอง ก็จะเป็น

การเก็บไอเท็มชิ้นนั้นเข้ามาในตัว ซึ่งไอเท็มเหล่านั้นจำเป็นต้องมีส่วนที่บันทึกอยู่ และสามารถจัดเรียง ปรับตำแหน่ง รวมไปถึงใช้งานได้ต่อไป

สิ่งแรกที่ต้องคำนึงถึงก็คือ ผู้เล่นมีที่ว่างหรือ Inventory คงเหลือในการเก็บหรือไม่ โดยเมธอด Player::AddItem ทำหน้าที่ในการตรวจสอบและนำมาบันทึกใน Inventory ของผู้เล่นต่อไป

```
bool Player::AddItem(Item *item)
        { bool found = false;
          for(int i = 0; i< 5 && !found; i++) for(int j = 0; j< 5 && !found; j++)
          {
            if(slot[i][j]->stack < ItemStack_Max && slot[i][j]->stack > 0)
            {
              if(slot[i][j]->item[0] != null)
              {
                if(slot[i][j]->item[0]->GetID() == item->GetID())
                {
                  if( slot[i][j]->stack < slot[i][j]->item[0]->GetStack())
                  {
                    slot[i][j]->item[slot[i][j]->stack++] = item;
                    stringstream s;
                    s << slot[i][j]->stack;
                    slot[i][j]->button->SetText(s.str());
                    found = true;
                  }
                }
              }
            }
          }
          if(!found)
          {
            for(int i = 0; i< 5 && !found; i++) for(int j = 0; j< 5 && !found; j++)
            {
              if(slot[i][j]->stack == 0)
```

```
                {
                    slot[i][j]->item[0] = item;
                    for(int k = 0; k< 3; k++)
                    {
                        slot[i][j]->buttondefaultimage[k] = slot[i][j]->button->GetImage(k)->GetTexture();
                        slot[i][j]->button->GetImage(k)->SetTexture(item->GetImage());
                        slot[i][j]->stack = 1;
                        slot[i][j]->button->SetText("1");
                    }
                    found = true;
                }
            }
        }
        return found;
    }
```

เนื่องจากไอเท็มแต่ละชนิดนั้น สามารถเก็บสะสมรวมกันไว้ในช่องเดียวกันได้หลายๆ ชิ้น หรือการสะสม (Stack) ในส่วนนี้จึงมีเมธอด ItemSlot::Merge ทำหน้าที่ในการรวมและจัดการไอเท็ม ให้อยู่ในลักษณะที่ควรจะเป็น เช่น สมุนไพร กำหนดให้สามารถจัดเก็บในเดียวกันช่องได้กัน 30 ชิ้น เมื่อทำการเก็บสมุนไพรชิ้นที่ 31 และ 32 ก็จะต้องถูกแยกเป็น สมุนไพร 30 ชิ้น 1 กอง และ สมุนไพร 2 ชิ้น อีกหนึ่งกอง

```
void ItemSlot::Merge(ItemSlot *destination, ItemSlot *incoming, unsigned int num)
    {
        if(destination == incoming || destination == null || incoming == null) return;
        unsigned int cur = 0;
        for(int i = destination->stack; destination->stack < destination->item[0]->GetStack() &&
incoming->stack > 0 && cur < num; i++)
        {
            destination->item[i] = incoming->item[incoming->stack-1];
            destination->stack++;
            incoming->item[incoming->stack-1] = null;
            incoming->stack--;
```

```
            cur++;
        }
        destination->button->SetText(ToString<int>(destination->stack));
        if(incoming->stack == 0)
        {
            incoming->button->SetText(incoming->defaulttext);
            for(int i = 0; i< 3; i++)
                incoming->button->GetImage(i)->SetTexture(incoming->buttondefaultimage[i]);
        }
    }
```

จากนั้นเมื่อสามารถจัดการรวมไอเท็มเข้ามาได้แล้ว จึงทำให้สามารถสลับเปลี่ยนตำแหน่งของไอเท็มที่อยู่ใน Inventory ได้ เพื่ออำนวยความสะดวกให้แก่ผู้เล่น รวมไปถึงเตรียมการณ์เอาไว้เพื่อรองรับระบบเครื่องแต่งกายต่อไป ด้วยเมธอด ItemSlot::Swap

```
void ItemSlot::Swap(ItemSlot *one, ItemSlot *two)
    {
        if(one == two || one == null || two == null) return;

        int stack = one->stack;
        one->stack = two->stack;
        two->stack = stack;

        for(int k = 0; k< ItemStack_Max; k++)
        {
            Item *it = one->item[k];
            one->item[k] = two->item[k];
            two->item[k] = it;
        }

        one->button->SetText((one->stack > 0)? ToString(one->stack) : one->defaulttext);
        two->button->SetText((stack > 0)? ToString(two->stack) : two->defaulttext);
```

```
    for(int k = 0; k< 3; k++)
    {
      Image *img = one->button->GetImage(k);
      one->button->SetImage(k, two->button->GetImage(k));
      two->button->SetImage(k, img);
    }
  }
```

จากนั้นเมื่อสามารถจัดการไอเท็มภายใน Inventory ได้แล้ว ส่วนถัดไปคือการใช้งานไอเท็ม ซึ่งไอเท็มแต่ละชนิดจะมีคุณสมบัติแตกต่างกันไป รูปแบบแรกก็คือไอเท็มที่สามารถใช้งานได้ หรือก็คือ สามารถลดจำนวนลงได้ แลกกับความช่วยเหลือ เช่น สมุนไพรที่มีผลฟื้นฟู HP 30 หน่วย เมื่อกดใช้สมุนไพรก็จะหายไป 1 ชิ้น และได้รับการฟื้นฟู HP 30 หน่วย โดยมีเมธอด InventoryWindowHandler::UseItem ทำหน้าที่จัดการในส่วนนี้

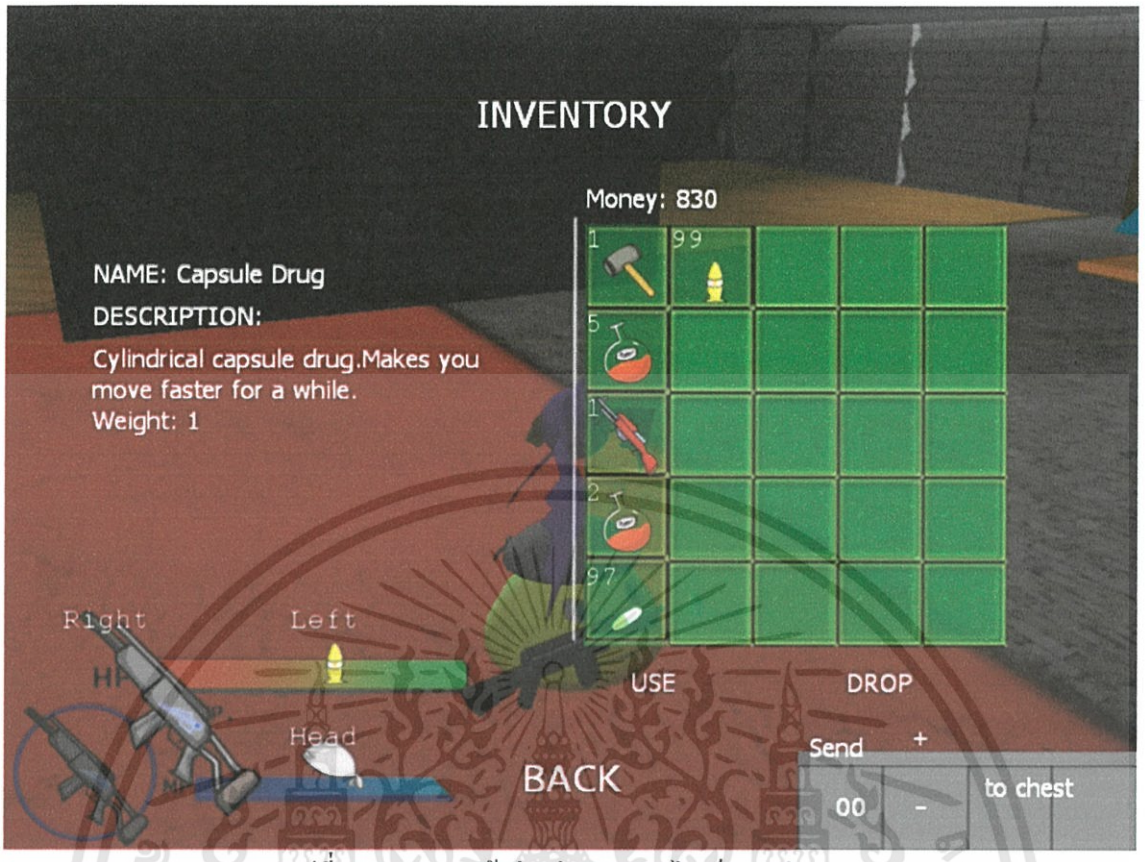
```
void InventoryWindowHandler::UseItem(ItemSlot **highlight)
{
  bool res = (*highlight)->item[(*highlight)->stack-1]->Use(eventcenter->GetPlayer());
  if(res) (*highlight)->stack--;
  if((*highlight)->stack == 0)
  {
    for(int k = 0; k< 3; k++)
      (*highlight)->button->GetImage(k)->SetTexture((*highlight)->buttondefaultimage[k]);
    (*highlight)->button->SetText((*highlight)->defaulttext);
    drop->Visible(false);
    use->Visible(false);
    (*highlight)->button->Highlighted(false);
    *highlight = null;
    this->highlight = null;
    this->cHighlight = null;
  }
  else
```

```
        {
          (*highlight)->button->SetText(ToString((*highlight)->stack));
        }
      }
```

ถัดมาคือการทิ้งไอเท็ม ซึ่งเป็นอีกส่วนหนึ่งที่ออกแบบมาอำนวยความสะดวกให้กับผู้เล่น ในกรณีที่ต้องการนำไอเท็มที่ไม่ได้ใช้งานออกไป ให้มีเนื้อที่ในการจัดเก็บเพิ่มขึ้น เพื่อให้สามารถเก็บไอเท็มชิ้นที่ต้องการได้ โดยมีเมธอด InventoryWindowHandler::DropItem ทำหน้าที่ในส่วนนี้

```
void InventoryWindowHandler::DropItem(ItemSlot **highlight)
      {
        Button *dropitem = drop;
        std::string sTemp = "dropitem";
        eventcenter->RaiseScript(sTemp, (Element*)(*highlight)->item[(*highlight)->stack-1],
(void*)system->GetPlayer());
        if((*highlight)->stack == 0)
        {
          for(int k = 0; k< 3; k++)
            (*highlight)->button->GetImage(k)->SetTexture((*highlight)->buttondefaultimage[k]);
          (*highlight)->button->SetText((*highlight)->defaulttext);
          dropitem->Visible(false);
          (*highlight)->button->Highlighted(false);
          *highlight = null;
          this->highlight = null;
          this->cHighlight = null;
        }
      }
```

เมื่อทำการจัดวางหน้าจอให้รองรับหน้าต่างไอเท็มเรียบร้อยแล้ว จะได้ผลดังแสดงในภาพที่ 4-13



รูปที่ 4-14 แสดงหน้าต่างส่วนจัดการไอเท็มภายในเกม

เมื่อเราสามารถจัดการกับไอเท็มภายในตัว และเก็บไอเท็มทั่วไปได้แล้วสิ่งถัดมาก็คือ การแลกเปลี่ยนสินค้าด้วยเงิน โดยมีหน้าต่างรองรับการแลกเปลี่ยน ซื้อขายสินค้า ในส่วนแรกที่ต้องทำการตรวจสอบก็คือ ไอเท็มชิ้นดังกล่าวนั้นเป็นไอเท็มของเรา หรือไอเท็มของทางร้าน หากเป็นไอเท็มของเราที่อยู่ใน Inventory เมื่ออยู่ในหน้าต่างซื้อขาย ก็จะสามารถขายเพื่อแลกเป็นเงินได้ ในทางกลับกันหากเป็นไอเท็มของทางร้านค้า ก็ระบุราคาขายและแสดงออกมา ให้ผู้เล่นสามารถกดซื้อได้จากรายการที่มีอยู่ มีเมธอด TradeWindowHandler::Action_MouseUp ทำหน้าตัดสินลักษณะไอเท็ม

```
void TradeWindowHandler::Action_MouseUp(Dispatcher_Arg)
        {
         Dispatcher_Arg = (MGE_Event *e, MGE_EventDispatcher *funcOwner, void* param)
         TradeWindowHandler *h = (TradeWindowHandler*)funcOwner;
         string n = h->highlight->GetName();
         int id = ConvertToInt(n.substr(n.length()-2));
         int i = id / 10, j = id % 10;
```

```
        /*string::npos คือตำแหน่งที่มากที่สุดที่ string ตัวหนึ่งจะมีได้ ในที่นี้ ถ้าหาคำที่เราให้หาไม่เจอ จะ
คืนค่านี้มาให้แทน*/
            if(n.find("inventory") != string::npos)
            {
               h->Sell(h->player->slot[i][j], h->num);
               if(h->inventory[i][j]->stack == 0) h->inventory[i][j]->button->Highlighted(false);
            }
            else if(n.find("trade") != string::npos)
            { if(h->player->CanAddItem(h->info.item[i][j], h->num))
               {
                  int money = h->player->GetMoney();
                  int price = h->info.item[i][j].price * h->num;
                  if(money >= price)
                     h->Buy(h->info.item[i][j], h->num);
                  else
                     h->eventcenter->GetFrame()->DisplayMessage("Not enough money");
               }
               else
               {
                  h->eventcenter->GetFrame()->DisplayMessage("Bag is full!");
               }
            }
         }
```

หากไอเท็มที่เลือกเป็นไอเท็มสำหรับซื้อ เมื่อผู้เล่นเลือกที่จะซื้อ ก็ต้องตรวจสอบว่า ไอเท็มชิ้นนั้นสามารถซื้อได้ด้วยจำนวนเงินที่ผู้เล่นมีอยู่หรือไม่ โดยมีเมธอด TradeWindowHandler::Buy ในการตรวจสอบ และปรับปรุงจำนวนเงินหากทำการซื้อสำเร็จ

```
void TradeWindowHandler::Buy(ItemInfoStruct inf, unsigned int num)
      {
         int price = inf.price * num;
         player->SetMoney(player->money - price);
```

```
        for(unsigned int i = 0; i< num; i++)
        {
            Item *item = new Item(eventcenter);
            item->Initialize(eventcenter->GetEngine(), inf, eventcenter);
            player->AddItem(item);
        }
        UpdatePage();
        SetDescription(DescriptionDefault, "", "", inf.price);
    }
```

ถัดมาหากเป็นไอเท็มสำหรับขาย ก็จะทำการลดจำนวนไอเท็ม และเพิ่มเงินในจำนวนที่ตั้งไว้ โดยใช้เมธอด TradeWindowHandler::Sell

```
void TradeWindowHandler::Sell(ItemSlot *s, unsigned int num)
    {
        int price = s->item[0]->GetPrice()/2 * num;
        player->SetMoney(player->money + price);
        SetDescription(DescriptionDefault, "", "", s->item[0]->GetPrice());
        for(unsigned int i = 0; i< num; i++)
        {
            Item *item = player->DropItem(s->item[s->stack-1]);
            item->FlushModel();
            delete item;
        }
        UpdatePage();
    }
```

ในหน้าต่างเมนูดังแสดงในรูปที่ 4-14 จะเป็นการเปรียบเทียบเมื่อทำการเลือกไอเท็มที่ตั้งอยู่กับร้านจะแสดงเป็นการซื้อไอเท็ม เมื่อเลือกไอเท็มที่อยู่ใน Inventory ของผู้เล่นจะแสดงเป็นการขายไอเท็ม

รูปที่ 4-15 แสดงหน้าต่างการซื้อขายไอเท็มภายในเกม

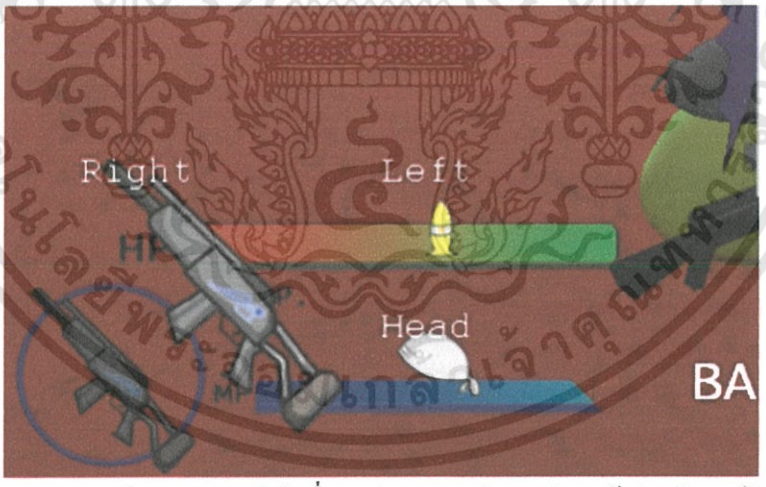
4.5.4 การประยุกต์ระบบเครื่องแต่งกาย

เครื่องแต่งกายจะเป็นส่วนที่ต่อยอดมาจากระบบไอเท็ม ซึ่งมีคุณสมบัติคือเมื่อนำมาสวมใส่ให้ผู้เล่นแล้วจะเสริมค่าสถานะจนกว่าจะถอดออก และเมื่อสวมใส่แล้วจะสามารถโจมตีได้ ซึ่งก่อนที่ตัวละครจะสวมใส่ได้จะต้องมีการสร้างช่องสวมใส่ให้กับผู้เล่นก่อน แล้วจึงสามารถสลับไปในช่องที่ตรียมไว้เพื่อสวมใส่ ซึ่งในที่นี้แบ่งเป็นส่วนมือซ้าย ส่วนมือขวา โดยจะมีโค้ดบางส่วนที่ใช้ในการรองรับระบบทักษะ ที่ขึ้นอยู่กับอาวุธแต่ละประเภทด้วย ผ่านเมธอด Player::EquipRightHand และ Player::EquipLeftHand

```
void Player::EquipRightHand()
        {
          Character::EquipRightHand();
          Image *img = (Image*)gui->GetHandler()->GetElementByName("itemava");
          Text *text = (Text*)gui->GetHandler()->GetElementByName("item_amount");
          text->SetText("");
          if(righthand->stack > 0)
          {
            img->Visible(true);
            img->SetTexture(righthand->button->GetImage(0)->GetTexture());
            if(righthand->item[0]->GetType() == ObjectType_Item_Gun)
            {
              if(lefthand->stack > 0) if(lefthand->item[0]->GetType() == ObjectType_Item_Bullet)
                text->SetText(ToString<int>(lefthand->stack));
            }
```

```
        else if(righthand->item[0]->GetType() == ObjectType_Item_Bow)
        {
            if(lefthand->stack > 0) if(lefthand->item[0]->GetType() == ObjectType_Item_Arrow)
                text->SetText(ToString<int>(lefthand->stack));
        }
        else
        {
            if(righthand->stack > 1)
                text->SetText(ToString<int>(righthand->stack));
        }
        SetVariable("weaponrange", GetHitRange(righthand->item[0]->GetAttackRange()));
    }
    else
    {
        SetVariable("weaponrange", 0);
        img->Visible(false);
    }
    if(GetEquipRight())
    {
//ถ้าถืออาวุธอยู่ ให้เลือกชุดทักษะที่อาวุธประเภทนี้สามารถใช้ได้
        SkillSet *cur = null;
        GameObjectType type = GetEquipRight()->GetType();
        switch (type)
        {
        case ObjectType_Item_Sword: cur = onehand; break;
        case ObjectType_Item_Heavy: cur = heavy; break;
        case ObjectType_Item_Gun: case ObjectType_Item_Bow:
            cur = range; break;
        }
        if(cur)
        {
            cur >ActivatcPassivc();
```

```
            }
        }
        gui->RefreshSkillSlot();
    }
void Player::EquipLeftHand()
    {
        Character::EquipLeftHand();
        Text *text = (Text*)gui->GetHandler()->GetElementByName("item_amount");
        if(lefthand->stack > 0)
        {
            if(lefthand->item[0]->GetType() == ObjectType_Item_Bullet ||
                lefthand->item[0]->GetType() == ObjectType_Item_Arrow)
            {
                text->SetText(ToString<int>(lefthand->stack));
            }
        }
    }
```



รูปที่ 4-16 แสดงส่วนการสวมใส่เครื่องแต่งกายภายในเกมในหน้าต่างจัดการไอเท็ม

เมื่อสามารถสวมใส่เครื่องแต่งกายได้แล้ว ถัดมาก็คือตัดสินใจเกี่ยวกับเครื่องแต่งกายชนิดนั้นว่าเป็นอาวุธหรือไม่ หากเป็นอาวุธ ผู้เล่นก็จะสามารถทำการโจมตีได้ โดยอาวุธแต่ละชนิดจะมีการกำหนดระยะว่า

สามารถโจมตีได้ในระยะเท่าใด ผ่านการสร้างขอบเขตการโจมตี (Area of damage) หากอยู่ไกลเกินไป การโจมตีก็จะไม่เป็นผล โดยส่วนของเมธอดนี้จะทำหน้าที่กำหนดขอบเขตที่อาวุธสามารถโจมตีได้

```
Item *item = inventory->GetRightHandSlot()->item[0];
    float maxRange = 0;
//AttackRange แบ่งเป็น VeryFar, far, Medium, Near และ VeryNear โดยจะเก็บเป็นตัวแปร int บอกว่าเป็นประเภทใด และคืนค่าที่เป็นค่าระยะในหน่วยพิกเซลไปให้
    if((item->GetAttackRange() & 0x10) != 0) maxRange = HitRange_VeryFar;
    else if((item->GetAttackRange() & 0x08) != 0) maxRange = HitRange_Far;
    else if((item->GetAttackRange() & 0x04) != 0) maxRange = HitRange_Medium;
    else if((item->GetAttackRange() & 0x02) != 0) maxRange = HitRange_Near;
    else if((item->GetAttackRange() & 0x01) != 0) maxRange = HitRange_VeryNear;
    D3DXVECTOR3 dest, vector = D3DXVECTOR3(vec.x, vec.y, vec.z);
    Monster *mon = (Monster*)GetTarget();
    if(!mon) dest = D3DXVECTOR3(vec.x, vec.y, vec.z);
    else
    {
      dest = mon->GetLocation();
      if(D3DXVec3Length(&(GetLocation() - dest)) > maxRange)
        dest = D3DXVECTOR3(vec.x, vec.y, vec.z);
    }
    if(D3DXVec3Length(&(GetLocation() - dest)) <= maxRange)
    {
      CharacterStatus *stat = (CharacterStatus*)GetStatus(CharacterStatus::Throw);
      stat->Reset();
      SetStatus(stat);
      Turn(dest);
    }
```

จากนั้นในกรณีที่ไอเท็มที่ไม่ใช่อาวุธถูกสวมใส่ จะสามารถโยนออกไปได้ ซึ่งจะใช้วิธีการโยนแบบ Projectile เริ่มจากตรวจสอบตำแหน่งของตัวละคร การแสดงผลให้ไอเท็มลอยไปยังเป้าหมายที่กำหนดเอาไว้ และผลที่เกิดขึ้นเมื่อสิ่งของชิ้นนั้นกระทบลงพื้น จากนั้นจึงหายไป

โดยเมธอด Character::Throw ทำหน้าที่ตรวจสอบคุณสมบัติของไอเท็มก่อนที่จะทำการโยน และเมธอด ProjectileStatus_Fly::Update ทำหน้าที่แสดงผลไอเท็มที่ถูกโยนออกไปในลักษณะ Projectile โดยในส่วนนี้ได้มีการประยุกต์ระดับความแม่นยำในการโยนมาแล้ว

```
void Character::Throw(Item *item, D3DXVECTOR3 destination)
        {
          Projectile *proj = new Projectile(eventcenter);
          float maxRange = 0;
          if((item->GetAttackRange() & 0x10) != 0) maxRange = HitRange_VeryFar;
          else if((item->GetAttackRange() & 0x08) != 0) maxRange = HitRange_Far;
          else if((item->GetAttackRange() & 0x04) != 0) maxRange = HitRange_Medium;
          else if((item->GetAttackRange() & 0x02) != 0) maxRange = HitRange_Near;
          else if((item->GetAttackRange() & 0x01) != 0) maxRange = HitRange_VeryNear;
          float b = D3DXVec3Length(&(item->GetLocation() - destination)) / maxRange;
          item->GetModel()->CancelTransformOverride();
          float speedm = ConvertToFloat(GetVariable("proj_speed"));
          float atkm = ConvertToFloat(GetVariable("throw_dmg"));
          float acc = ConvertToFloat(GetVariable("throw_acc"));
          acc = 100 - acc;
          if(acc > 0)
          {
            destination.x += ((float)Random(0, (int)acc)) - acc/2;
            destination.z += ((float)Random(0, (int)acc)) - acc/2;
          }
          proj->Initialize(item->GetModel(), item->GetLocation(), destination, "sin", b, this, speedm);
          proj->SetScript(item->GetOnThrowScript());
          proj->SetValue1(item->GetWeight() * atkm);
          proj->SetColisionRadius(item->GetColisionRadius());
          eventcenter->GetFrame()->GetScene3D()->AddObject(proj);
          proj->UpdateMainLoop(eventcenter->GetEngine(), 0);
          delete item;
        }
```

```
bool ProjectileStatus_Fly::Update(Object *object, EventCenter *eventcenter, void* extraparam)
{
  model->Update();
  Projectile *proj = (Projectile*)object;
  a += speed / length;
  D3DXVECTOR3 location = INTERPOLATE(origin, destination, a);
  D3DXVECTOR3 rotation = proj->GetRotation();
// บอกว่าเป็นการเคลื่อนไหวแบบโยน หรือลอยขึ้นถึงจุดหนึ่งและตกลงมาบนพื้น คำนวณโดยใช้ค่า
sin ของค่า a ซึ่งจะเปลี่ยนทุกๆเฟรม
  if(movement =="sin")
  {
    float angle = D3DXToRadian(a * 180);
    location.y += 80* b *sin(angle);
    rotation.x += D3DXToRadian(5)*sin(rotation.y);
    rotation.z += D3DXToRadian(5)*cos(rotation.y);
    proj->SetRotation(rotation);
  }
  proj->SetLocation(location);
  SetLocation(location);
  Object *in = eventcenter->GetNearestObject(proj, (GameObjectType)(ObjectType_Monster |
ObjectType_Default));
  bool intersect = false;
  if(in) intersect = in->Intersects(proj->GetLocation(), proj->GetColisionRadius());
  if(intersect|| proj->Intersects(destination) || a >= 1)
  {
    proj->Hit((intersect && ((in->GetType() & ObjectType_Monster) != 0))?in : null);
    a = 0.1f;
    eventcenter->GetFrame()->GetScene3D()->DeleteNode(proj, true, true);
  }
  return true;
}
```

รูปที่ 4-17 แสดงการโยนไอเท็มแบบ Projectile ภายในเกม

4.5.5 การประยุกต์ระบบสถานะพิเศษและทักษะ

สถานะพิเศษ(Attribute) จะมีคุณลักษณะที่แสดงผลโดยตรงกับระบบสถานะของผู้เล่น โดยอาจจะส่งผลทางสนับสนุน หรือลดความสามารถของผู้เล่น แล้วหายไปภายในระยะเวลาหนึ่ง ส่วนของทักษะก็จะใช้คุณลักษณะของสถานะพิเศษเช่นกัน โดยเฉพาะทักษะรูปแบบ Active ใช้วิธีการปรับปรุงให้ผู้เล่นมีสถานะพิเศษเพิ่มขึ้นชั่วขณะหนึ่งระหว่างโจมตี และกลับมาเป็นปกติเมื่อการโจมตีจบลงแล้ว

สำหรับการแสดงผลสถานะพิเศษ มีเมธอด Attribute::Update ทำหน้าที่อัพเดทว่า สถานะพิเศษนั้นยังคงอยู่หรือไม่ และแสดงผลออกมาพร้อมกับไอคอน จากนั้นเมื่อเวลาหมดลง ก็ทำการลบการไอคอนและผลสถานะออกพร้อมๆ กับเวลาที่หมดลงด้วยเมธอด Character::RemoveAttribute

```
void Attribute::Update()
        {
          GameSystem *system = eventcenter->GetGameSystem();
          if((unsigned int)(system->GetGameTime() - lastTick) >= info.tick)
          {
            lastTick = system->GetGameTime();
            eventcenter->RaiseScript(info.onTick, owner, owner->GetTarget());
          }
          if((unsigned int)(system->GetGameTime() - start) >= info.duration)
          {
```

```
            eventcenter->RaiseScript(info.onDeactivate, owner);
            eventcenter->RaiseScript("SetCallerToObject2D, gui, att_description, SetText, '', this,
owner);
            icon->RemoveEventListener("ButtonMouseOver", Icon_MouseOver, this);
            icon->RemoveEventListener("ButtonMouseOut", Icon_MouseOut, this);
            icon->GetScene()->RemoveObject(icon);
            owner->RemoveAttribute(this);// must be last line to execute in this function
          }
        }
void Character::RemoveAttribute(Attribute *attribute)
        {
          LinkedList<Attribute*> *pointer = this->attribute, *prev = null;
          while(pointer)
          {
            if(pointer->data == attribute)
            {
              if(prev) prev->next = pointer->next;
              else this->attribute = pointer->next;
              pointer->next = null;
              delete pointer->data;
              pointer->data = null;
              break;
            }
            else
            {
              prev = pointer;
              pointer = pointer->next;
            }
          }
        }
```

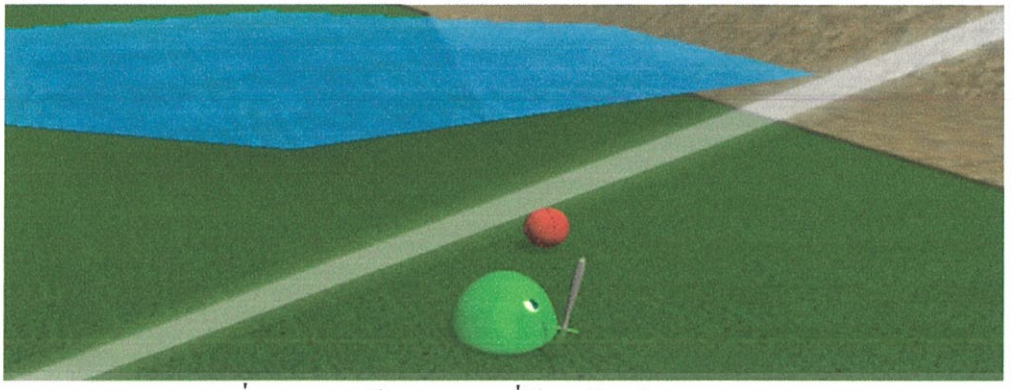
ถัดมาในส่วนของทักษะ ที่มีการออกแบบลักษณะให้ผู้เล่นได้ออกท่าทางนั้น จะมีวิธีการตรวจสอบให้ผู้เล่นตอบสนองระหว่างใช้ทักษะ โดยทำการการเรียกพื้นที่สี่เหลี่ยม 2 มิติขึ้นมา จากนั้นให้ผู้เล่นกดและลากไปปล่อยอีกจุดที่กำหนด ทำให้เกิดเส้นทางที่ Pointer พากผ่านจากจุดหนึ่งไปยังจุดหนึ่ง ใช้เป็นตัวอ้างอิงว่าทักษะที่ใช้ออกไปนั้นถูกศัตรูหรือไม่ เพื่อจำลองให้เสมือนว่าผู้เล่นโจมตีถูกศัตรูด้วยมือตัวเอง

ในการคำนวณการตัดผ่านของเส้นกับศัตรูนั้น จะนำวิธีการการเลือกวัตถุผ่านหน้าจอ (Picking) ดังรายละเอียดในบทที่ 2 เข้ามาใช้ผ่านเมธอด Picker::Pick

```
bool Picker::Pick(MGE_Model *model, float *dist)
        {
          D3DXMATRIX xWorld, tran, rot, scale;
          D3DXMatrixTranslation(&tran, model->GetLocation().x, model->GetLocation().y, model-
>GetLocation().z);
          D3DXMatrixRotationYawPitchRoll(&rot, model->GetYaw(), model->GetPitch(), model-
>GetRoll());
          D3DXMatrixScaling(&scale, model->GetScaleX(), model->GetScaleY(), model-
>GetScaleZ());
          xWorld = scale * rot * tran;
          D3DXMATRIX mat;
          BOOL hit = FALSE;
          LPD3DXFRAME frame = model->GetFrame();
          if(frame)
            Pick(frame, &hit, dist, xWorld);
          return hit == TRUE;
        }
        void Picker::Pick(LPD3DXFRAME frame, BOOL *hit, float *dist, D3DXMATRIX xWorld)
        {
          D3DXMESHCONTAINER_EXTENDED *meshContainer =
(D3DXMESHCONTAINER_EXTENDED*)frame->pMeshContainer;
          while(meshContainer)
          {
            BOOL h = FALSE;
            float d = 999999;
```

```
            D3DXMATRIX mat;
            D3DXVECTOR3 objorigin, objdirection;
            LPD3DXMESH mesh = (meshContainer->pSkinInfo) ? meshContainer->exSkinMesh :
meshContainer->MeshData.pMesh;
            if(meshContainer->pSkinInfo)
               D3DXMatrixInverse(&mat, 0, D3DXMatrixIdentity(&D3DXMATRIX()));
            else
               D3DXMatrixInverse(&mat, 0, &(frame->TransformationMatrix * xWorld));
            D3DXVec3TransformCoord(&objorigin, &origin, &mat);
            D3DXVec3TransformNormal(&objdirection, &direction, &mat);
            D3DXVec3Normalize(&objdirection, &objdirection);
            D3DXIntersect(mesh, &objorigin, &objdirection, &h, 0, 0, 0, &d, 0, 0);
            *hit = *hit || h;
            if(*dist > d) *dist = d;
            meshContainer = (D3DXMESHCONTAINER_EXTENDED*)meshContainer-
>pNextMeshContainer;
         }
         if(frame->pFrameSibling)
            Pick(frame->pFrameSibling, hit, dist, xWorld);
         if(frame->pFrameFirstChild)
            Pick(frame->pFrameFirstChild, hit, dist, xWorld);
}
```

เมื่อสามารถคำนวณการตัดผ่านของเส้นได้แล้ว จึงนำไปใช้กับทักษะภายในเกม โดยรูปที่ 4-15 แสดงเส้นพาดสีขาวที่ได้จากการลากเส้นเพื่อใช้ทักษะพิเศษของตัวละคร



รูปที่ 4-18 การใช้ทักษะพิเศษที่ต้องสร้างเส้นพาดผ่านศัตรู

4.5.6 การประยุกต์ใช้ปัญญาประดิษฐ์

ในส่วนนี้จะแบ่งได้เป็นส่วนของศัตรู และส่วนของคู่หู รายละเอียดดังแสดงในบทที่ 3 หัวข้อปัญญาประดิษฐ์ภายในเกม

ส่วนของศัตรูโดยทั่วไปแล้วนั้น จะมีคุณลักษณะคล้ายกับ NPC ผสมกับคุณลักษณะของผู้เล่น กล่าวคือสามารถเคลื่อนที่ หรือตอบสนองได้ เหมือน NPC และมีคุณลักษณะคล้ายกับผู้เล่นคือ มีค่าสถานะเป็นของตัวมันเอง ซึ่งในส่วนนี้ถูกจัดการด้วยการนำเอา AI เข้ามาประยุกต์ใช้ ให้ศัตรูสามารถเคลื่อนที่และโจมตีผู้เล่นได้อย่างมีรูปแบบ โดยแบ่งเป็นเมธอดต่างๆ สำหรับจัดการรูปแบบการเคลื่อนไหวและโจมตี ได้แก่ การค้นหาและการโจมตีผู้เล่น (Search and Attack) ด้วยเมธอด MonsterStatus_Move::Update หารหนีจากผู้เล่นด้วยเมธอด MonsterStatus_Flee::Update และการโจมตีแล้วหนี (Hit and Run) เป็นการผสมผสานระหว่างเมธอด 2 เมธอดดังกล่าวลงไป นั่นคือเมื่อศัตรูถูกกระตุ้นแล้ว มอนสเตอร์จะเดินเข้าหาผู้เล่นและโจมตี (Search & Attack) จากนั้นก็หนีออกมาระยะเวลาหนึ่ง แล้วกลับไปค้นหาและโจมตีอีกครั้งต่อไป

```
bool MonsterStatus_Move::Update(Object *object, EventCenter *eventcenter, void* extraparam)
        {
          Monster *mon = (Monster*)object;
          MGE *base = eventcenter->GetEngine();
          model->Update();

          if(!mon->Visible()) return true;

          mon->Move(mon->GetDestination(), base);
          D3DXVECTOR2 dest2 = D3DXVECTOR2(mon->GetDestination().x, mon-
>GetDestination().z);
          {
```

```
            mon->SetDestination(mon->GetTarget()->GetLocation());
            SkillInfoStruct skill = mon->GetUsingSkill();
          //ถ้าผู้เล่นอยู่ในระยะแล้ว ให้ใช้ทักษะโจมตีเลย
            if(D3DXVec3Length(&(mon->GetLocation() - mon->GetTarget()->GetLocation())) <=
skill.range)
              mon->UseSkill(skill);
          }
          return true;
        }
bool MonsterStatus_Flee::Update(Object *object, EventCenter *eventcenter, void* extraparam)
        {
          model->Update();
          Monster *mon = (Monster*)object;
          if(!mon->Visible()) return true;
          Character *target = mon->GetTarget();
          mon->Turn(target->GetLocation());
          D3DXVECTOR3 rotation = mon->GetRotation();
        //ค่าที่จะส่งไปต้องอยู่ในหน่วย Radian นั่นคือต้องแปลงจากองศาเป็น Radian ก่อน
          rotation.y += D3DXToRadian(180);
          mon->SetRotation(rotation);
          mon->Move(eventcenter->GetEngine());
          if(fleeCount > 0) if(++curFleeCount >= fleeCount)
          {
            if(mon->IsHitAndRun() && mon->GetTarget() != null)
            {
              mon->SetStatus(CharacterStatus::Move);
            }
            else
            {
              mon->SetStatus(CharacterStatus::Idle);
            }
          }
```

```
        return true;
    }
```

นอกเหนือจากนั้นเป็นการกำหนดเงื่อนไขพิเศษเมื่อศัตรูมีระดับ HP ลดลงถึงจุดที่กำหนดซึ่งในจุดนี้จะเป็นการประยุกต์นำเอาสถานะพิเศษ รวมไปถึงสถานะพิเศษและทักษะมาใช้ให้ศัตรูมีความสามารถบางอย่างคล้ายกับผู้เล่นก็ได้

ถัดมาในส่วนของคู่หู หรือก็คือตัวช่วยของผู้เล่น จะมีลักษณะ AI ที่มีลักษณะในการสนับสนุนผู้เล่นตรงข้ามกับมอนสเตอร์ จากโดยเมธอด Partner::OwnerAttack ทำหน้าที่จัดการในส่วนนี้โดย เริ่มตรวจสอบว่าผู้เล่นเริ่มทำการโจมตีแล้วหรือไม่ จากนั้นจึงทำการสนับสนุนด้วยพฤติกรรมที่แตกต่างกันไปตามสถานการณ์หรือตามที่ผู้เล่นปรับแต่งเอาไว้ โดยมีเมธอด Partner::ChooseSkill ในการเลือกชุดทักษะที่จะใช้ออกไปตามเงื่อนไขที่ระบุไว้ในการออกแบบส่วนบทที่ 3 หัวข้อปัญญาประดิษฐ์ของคู่หู (Partner AI)

```
void Partner::OwnerAttack()
    {
      if(!Visible()) return;
      CharacterStatus *stat = (CharacterStatus*)GetCurrentStatus();
      if(stat != GetStatus(CharacterStatus::Idle) && stat != GetStatus(CharacterStatus::Move))
        return;
    //ถ้ากำลังติด cooldown ระหว่างจากการร่ายครั้งที่แล้วอยู่จะไม่ทำอะไร
      if(castDelay != 0)
        return;
    //เลือกชุดทักษะที่จะใช้
      unsigned int num = ChooseSkillSet();
    //ถ้าเกิดข้อผิดพลาดขึ้นมาระหว่างการเลือก ให้ออกทันที
      if(num >= BatSkillSet::Num)
        return;
    //ถ้า ชุดที่เลือกได้ เป็นชุด Buff หรือ heal ซึ่งเป็นชุดที่เสริมกำลัง ให้เลือกผู้เล่นเป็นเป้าหมาย ถ้าไม่ใช่ ให้เลือก มอนสเตอร์ที่ผู้เล่นเลือกอยู่เป็นเป้าหมาย
      if(num == BatSkillSet::Debuff || num == BatSkillSet::Destruction) SetTarget(owner->GetTarget());
      else SetTarget(owner);
    //เลือกทักษะที่จะใช้
```

```
            Skill *s = ChooseSkill((BatSkillSet::Name)num);
            if(s && (s->Available()))
            {
               s->Activate();
               castDelay = (float)(((num * 2) +1)* UpdatePerSec);
            }
            else
            {
             //ถ้าเลือกไม่ได้ ให้ทำการฟื้นค่า MP แทน
               SetMP(GetMP() + (*GetMaxMPPointer())/50.f);
               castDelay = SecondToFrame(5);
            }
            maxCount = castDelay;
         }
         unsigned int Partner::ChooseSkillSet()
         {
            float chan[BatSkillSet::Num] = {0};
            float total = 0;
            for(int i = 0; i< BatSkillSet::Num; i++)
            {
              if(skill[i]->Available() && skill[i]->chance > 0)
              {
                 chan[i] = skill[i]->chance;
                 total += chan[i];
                 if((i == BatSkillSet::Debuff || i == BatSkillSet::Destruction) && owner->GetTarget() ==
null)
                 {
                    total -= chan[i];
                    chan[i] = 0;
                 }
              }
              else
```

```
            {
                chan[i] = 0;
            }
        }
        float ratio = 0;
        if(total > 0)
            ratio = 100.f/total;
        else
            return -1;
        //สุ่มค่าใหม่ให้อยู่ระหว่าง 0-100
        float c = 0;
        for(int i = 0; i< BatSkillSet::Num; i++)
        {
            chan[i] *= ratio;
            chan[i] = c + chan[i];

            c = chan[i];
        }
        int ran = Random();
        unsigned int num = 0;
        for(int i = 0; i< BatSkillSet::Num; i++)
        {
            if(ran < chan[i])
            {
                num = i;
                break;
            }
        }
        return num;
    }
Skill* Partner::ChooseSkill(BatSkillSet::Name num)
    {
```

```
        Skill *s = null;
        if(num == BatSkillSet::Debuff || num == BatSkillSet::Buff)
        {
            bool found = false;
            for(int i = 0; i< 4 && !found; i++)
            {
                if(skill[num]->GetSkill(0, i)->Available())
                    found = true;
            }
            if(found)
            {
                do
                {
                    s = skill[num]->GetSkill(0, Random(0, 4));
                } while(!s->Available());
            }
            else
                s = null;

            float chance = 60 + 0.5f*GetDEF();
            if(Random() > chance)
                s = null;
            int level = 0;
            if(owner->GetTarget())
                level = owner->GetTarget()->GetLevel();
            else
            {
                Object *obj[100] – {null};
                int len = 100;
                eventcenter->GetFrame()->GetScene3D()->GetObjectByType(obj, &len,
ObjectType_Monster);
                if(len > 0)
```

```
            {
                Character *cha = (Character*)obj[Random(0, len)];
                level = cha->GetLevel();
            }
            else
                level = -1;
        }
        if(level < owner->GetLevel() && level >= 0)
        {
//Random เป็นเมธอดที่สุ่มค่า โดยส่งค่าต่ำสุดที่ต้องการ และ Range ที่อยากให้สุ่มไปให้
            if(skill[num]->GetSkill(0, 0)->Available() && skill[num]->GetSkill(0, 2)->Available())
                s = skill[num]->GetSkill(0, Random(0, 2)*2);
            else if(skill[num]->GetSkill(0, 0)->Available())
                s = skill[num]->GetSkill(0, 0);
            else if(skill[num]->GetSkill(0, 2)->Available())
                s = skill[num]->GetSkill(0, 2);
            else
                s = null;
        }
        else if(level >= 0)
        {
            if(skill[num]->GetSkill(0, 1)->Available() && skill[num]->GetSkill(0, 3)->Available())
                s = skill[num]->GetSkill(0, (Random(0, 2)*2)+1);
            else if(skill[num]->GetSkill(0, 1)->Available())
                s = skill[num]->GetSkill(0, 1);
            else if(skill[num]->GetSkill(0, 3)->Available())
                s = skill[num]->GetSkill(0, 3);
            else
                s = null;
        }
        else
        {
```

```
            s = null;
        }
    }
    else if(num == BatSkillSet::Destruction)
    {
        Character *t = owner->GetTarget();
        float dmg = CalculateDamage(0);
        float hp = *t->GetHPPointer();
        if(hp > dmg * 1.8f && skill[num]->GetSkill(0, 3)->Available())
            s = skill[num]->GetSkill(0, 3);
        else if(hp > dmg * 1.2f && skill[num]->GetSkill(0, 2)->Available())
            s = skill[num]->GetSkill(0, 2);
        else if(hp > dmg && skill[num]->GetSkill(0, 1)->Available())
            s = skill[num]->GetSkill(0, 1);
        else if(skill[num]->GetSkill(0, 0)->Available())
            s = skill[num]->GetSkill(0, 0);
        else
            s = null;
    }
    else if(num == BatSkillSet::Heal)
    {
        float hpr = (*owner->GetHPPointer() / *owner->GetMaxHPPointer());
        if(hpr < 0.5f && skill[num]->GetSkill(0, 3)->Available())
            s = skill[num]->GetSkill(0, 3);
        else if(hpr < 0.7f && skill[num]->GetSkill(0, 2)->Available())
            s = skill[num]->GetSkill(0, 2);
        else if(hpr < 0.85f && skill[num]->GetSkill(0, 1)->Available())
            s = skill[num]->GetSkill(0, 1);
        else if(hpr < 0.94f && skill[num]->GetSkill(0, 0)->Available())
            s = skill[num]->GetSkill(0, 0);
        else
            s = null;
```

```
        }
        return s;
    }
```

4.5.7 การประยุกต์ระบบภารกิจ

ระบบภารกิจสามารถประยุกต์ใช้ได้เมื่อระบบข้างต้นถูกวางไว้เรียบร้อยแล้ว โดยคุณลักษณะสำคัญของระบบภารกิจคือการตรวจเงื่อนไขเป็นลำดับขั้น กล่าวคือ ภารกิจแต่ละภารกิจจะมีระยะช่วงตั้งแต่เริ่มจนจบภารกิจ (Progress) ต่างกันเช่น ทำการคุยกับ NPC ตัวอื่น แล้วจบภารกิจ หรือต้องคุยหลายครั้ง จึงจะจบภารกิจ โดยภารกิจนั้นจะสามารถแบ่งออกได้เป็น 2 รูปแบบได้แก่การคุยกับ NPC ตัวอื่นและการปราบศัตรูชนิดที่กำหนดตามจำนวนที่กำหนดไว้ โดยระบบภารกิจทั้งสองรูปแบบ มีการแบ่งหน้าที่ให้กับเมธอด Quest_Talk::Check และ Quest_Kill::Check ตามลำดับ

```
void Quest_Talk::Check(Object *object, string action)
    {
        if(!object) return;
        if(object->GetName().compare(npcName) == 0 && action.compare("talk") == 0)
        {
            line->Progress();
        }
    }
void Quest_Kill::Check(Object *object, string action)
    {
        if(!object) return;
        if(action.compare("kill") == 0)
        {
            if(object->GetName().compare(monName) == 0)
            {
    //DisplayMessage จะแสดงข้อความที่เราส่งไปให้ ไปบนหน้าจอ
                eventcenter->GetFrame()->DisplayMessage("Quest " + line->GetName() + " progressed:
Kill "
                    + monName + " " + ToString<unsigned int>(++cur) + "/" + ToString<unsigned
int>(num) + ".");
```

```
                Progress();
                if(cur == num)
                {
                   line->Progress();
                }
            }
        }
    }
```

รูปที่ 4-19 การแสดงข้อความ Progress ของภารกิจเมื่อกำจัดศัตรูได้ตามที่กำหนด

4.5.8 การประยุกต์ระบบเซฟ

ระบบเซฟทำหน้าที่ในการบันทึกข้อมูลต่างๆ ที่จำเป็นต่อการนำมาใช้เพื่อปรับค่าต่างๆ ให้กลับมาอยู่ในจุดที่ผู้เล่นเคยเล่นเอาไว้ ได้แก่สถานที่ที่เซฟ ค่าพลังต่างๆ ของตัวละคร ไอเท็มที่ตัวละครมี ทักษะที่เรียนรู้ไปแล้ว รวมไปถึงภารกิจที่ทำไปแล้ว ซึ่งข้อมูลเหล่านี้จะทำการเก็บไว้เป็นเซฟไฟล์ (สกุล.pss)

การปรับค่าต่างๆ ให้กลับมานั้น แยกการทำงานเป็นเมธอดจัดการในแต่ละส่วน ได้แก่ข้อมูลอาวุธที่ผู้เล่นสวมใส่อยู่ผ่านเมธอด Player::GetEquipmentInfo ข้อมูลไอเท็มที่ผู้เล่นมีอยู่ผ่านเมธอด Player::GetInventoryInfo และข้อมูลค่าพลังและสถานะต่างๆ จากนั้นจึงเข้าสู่เมธอด Player::Load เพื่อนำเอาข้อมูลต่างๆ ที่ได้มาไปปรับปรุงค่าให้กลับมาเป็นดังเดิม จากนั้นจึงเข้าสู่เมธอด GameSystem::GetInfo เพื่อเรียกข้อมูลเกี่ยวกับระบบและ GameSystem::Load เพื่อโหลดเอาสถานะที่เกี่ยวกับระบบต่างๆ กลับมา

```
void Player::GetEquipmentInfo(PlayerInfoStruct *inf)
        {
         if(lefthand->stack > 0)
         {
            inf->left.id = ConvertToInt(lefthand->item[0]->GetID());
            inf->left.num = lefthand->stack;
         }
         else
         {
            inf->left.id = -1;
            inf->left.num = 0;
         }
         if(righthand->stack > 0)
         {
            inf->right.id = ConvertToInt(righthand->item[0]->GetID());
            inf->right.num = righthand->stack;
         }
         else
         {
            inf->right.id = -1;
            inf->right.num = 0;
         }
         if(head->stack > 0)
         {
            inf->head.id = ConvertToInt(head->item[0]->GetID());
            inf->head.num = head->stack;
         }
         else
         {
            inf->head.id = -1;
            inf->head.num = 0;
         }
```

```
        }
void Player::GetInventoryInfo(PlayerInfoStruct *inf)
        {
          for(int i = 0; i< 5; i++) for(int j = 0; j< 5; j++)
          {
            if(slot[i][j]->stack > 0)
            {
              inf->inventory[i][j].id = ConvertToInt(slot[i][j]->item[0]->GetID());
              inf->inventory[i][j].num = slot[i][j]->stack;
            }
            else
            {
              inf->inventory[i][j].id = -1;
              inf->inventory[i][j].num = 0;
            }
          }
        }
PlayerInfoStruct Player::GetInfo()
        {
          PlayerInfoStruct inf;
          inf.location = GetLocation();
          inf.rotation = GetRotation();
          inf.level = eventcenter->GetFrame()->GetScene3D()->GetLevel();
          inf.player.name = playername;
          inf.player.level = level;
          inf.player.atk = atk;
          inf.player.def = def;
          inf.player.tec = agi;
          inf.player.ap = ap;
          inf.player.sp = sp;
          inf.player.timeplayed = timeplayed;
          inf.player.money = GetMoney();
```

```
        inf.heavy = heavy->GetInfo();
        inf.range = range->GetInfo();
        inf.onehand = onehand->GetInfo();
        eventcenter->GetGameSystem()->GetChestHandler()->GetInfo(&inf);
        GetEquipmentInfo(&inf);
        GetInventoryInfo(&inf);
        inf.batskill = partner->GetInfo();
        return inf;
    }
SystemInfoStruct GameSystem::GetInfo()
    {
        SystemInfoStruct inf;
        inf.currenttime = GetGameTime();
        inf.globalvar = GetVariable();
        LinkedList<Quest*> *pointer = quest;
        for(int i = 0; pointer != null && i < 50; i++)
        {
          inf.quest[i] = pointer->data->GetInfo();
          pointer = pointer->next;
        }
        return inf;
    }
void Player::Load(PlayerInfoStruct inf)
    {
    //ถ้าในข้อมูลที่เซฟไว้ มีตำแหน่งที่เซฟอยู่ด้วย
        if(inf.location != D3DXVECTOR3(0, 0, 0))
          SetLocation(inf.location);
        SetRotation(inf.rotation);
        LoadInventory(inf);
        eventcenter->GetGameSystem()->GetChestHandler()->Load(inf);
        SetPlayerName(inf.player.name);
        SetLevel(inf.player.level);
```

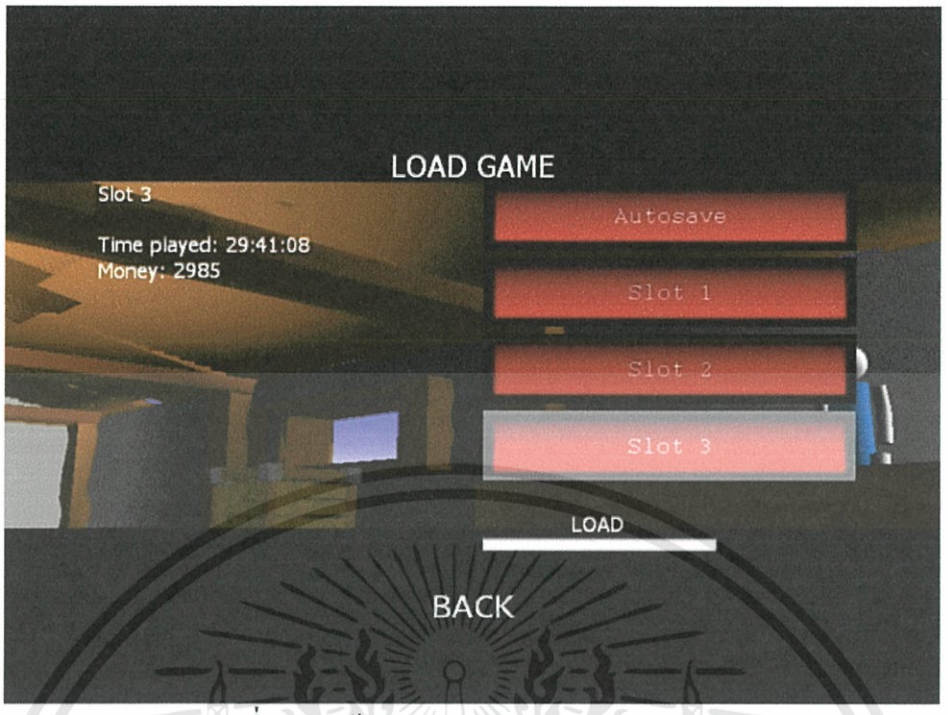
```
        SetATK(inf.player.atk);
        SetDEF(inf.player.def);
        SetAGI(inf.player.tec);
        SetAP(inf.player.ap);
        SetSP(inf.player.sp);
        exp = inf.player.exp;
        timeplayed = inf.player.timeplayed;
        money = inf.player.money;
        heavy->LoadInfo(inf.heavy);
        range->LoadInfo(inf.range);
        onehand->LoadInfo(inf.onehand);
        LoadEquipment(inf);
        partner->Load(inf.batskill);
        gui->RefreshSkillSlot();
    }
void Player::LoadInventory(PlayerInfoStruct inf)
    {
    for(int i = 0; i< 5; i++) for(int j = 0; j< 5; j++)
    {
      int id = inf.inventory[i][j].id;
      unsigned int num = inf.inventory[i][j].num;
      if(slot[i][j]->stack > 0)
      {
        for(int k = 0; k< slot[i][j]->stack; k++)
        {
          slot[i][j]->item[k]->FlushModel();
          delete slot[i][j]->item[k];
          slot[i][j]->item[k] = null;
        }
        slot[i][j]->stack = 0;
        slot[i][j]->button->GetImage(0)->SetTexture(slot[i][j]->buttondefaultimage[0]);
        slot[i][j]->button->GetImage(1)->SetTexture(slot[i][j]->buttondefaultimage[1]);
```

```
                slot[i][j]->button->GetImage(2)->SetTexture(slot[i][j]->buttondefaultimage[2]);
            }
            slot[i][j]->button->SetText(slot[i][j]->defaulttext);
            if(id < 0 || num == 0) return;
            ItemInfoStruct iTemp;
            eventcenter->RaiseEvent(new Event_RequestItemInfo(id, &iTemp));
            for(unsigned int k = 0; k< num; k++)
            {
                Item *item = new Item(eventcenter);
                item->Initialize(eventcenter->GetEngine(), iTemp, eventcenter);
                slot[i][j]->item[k] = item;
                slot[i][j]->stack++;
            }
            slot[i][j]->button->GetImage(0)->SetTexture(slot[i][j]->item[0]->GetImage());
            slot[i][j]->button->GetImage(1)->SetTexture(slot[i][j]->item[0]->GetImage());
            slot[i][j]->button->GetImage(2)->SetTexture(slot[i][j]->item[0]->GetImage());
            slot[i][j]->button->SetText(ToString(num));
        }
    }
void Player::LoadEquipment(PlayerInfoStruct inf)
    {
        UnequipLeft();
        UnequipRight();
        UnequipHead();
        if(lefthand->stack > 0)
        {
            for(int i = 0; i< lefthand->stack; i++)
            {
                lefthand->item[i]->FlushModel();
                delete lefthand->item[i];
                lefthand->item[i] = null;
            }
```

```
    lefthand->stack = 0;
    lefthand->button->GetImage(0)->SetTexture(lefthand->buttondefaultimage[0]);
    lefthand->button->GetImage(1)->SetTexture(lefthand->buttondefaultimage[1]);
    lefthand->button->GetImage(2)->SetTexture(lefthand->buttondefaultimage[2]);
  }
  if(righthand->stack > 0)
  {
    for(int i = 0; i< righthand->stack; i++)
    {
      righthand->item[i]->FlushModel();
      delete righthand->item[i];
      righthand->item[i] = null;
    }
    righthand->stack = 0;
    righthand->button->GetImage(0)->SetTexture(righthand->buttondefaultimage[0]);
    righthand->button->GetImage(1)->SetTexture(righthand->buttondefaultimage[1]);
    righthand->button->GetImage(2)->SetTexture(righthand->buttondefaultimage[2]);
  }
  if(head->stack > 0)
  {
    for(int i = 0; i< head->stack; i++)
    {
      head->item[i]->FlushModel();
      delete head->item[i];
      head->item[i] = null;
    }
    head->stack = 0;
    head->button->GetImage(0)->SetTexture(head->buttondefaultimage[0]);
    head->button->GetImage(1)->SetTexture(head->buttondefaultimage[1]);
    head->button->GetImage(2)->SetTexture(head->buttondefaultimage[2]);
  }
  ItemInfoStruct iTemp;
```

```
if(inf.left.num > 0)
{
   eventcenter->RaiseEvent(new Event_RequestItemInfo(inf.left.id, &iTemp));
   for(unsigned int i = 0; i< inf.left.num; i++)
   {
      Item *item = new Item(eventcenter);
      item->Initialize(eventcenter->GetEngine(), iTemp, eventcenter);
      lefthand->item[lefthand->stack++] = item;
   }
   lefthand->button->GetImage(0)->SetTexture(lefthand->item[0]->GetImage());
   lefthand->button->GetImage(1)->SetTexture(lefthand->item[0]->GetImage());
   lefthand->button->GetImage(2)->SetTexture(lefthand->item[0]->GetImage());
}
if(inf.right.num > 0)
{
   eventcenter->RaiseEvent(new Event_RequestItemInfo(inf.right.id, &iTemp));
   for(unsigned int i = 0; i< inf.right.num; i++)
   {
      Item *item = new Item(eventcenter);
      item->Initialize(eventcenter->GetEngine(), iTemp, eventcenter);
      righthand->item[righthand->stack++] = item;
   }
   righthand->button->GetImage(0)->SetTexture(righthand->item[0]->GetImage());
   righthand->button->GetImage(1)->SetTexture(righthand->item[0]->GetImage());
   righthand->button->GetImage(2)->SetTexture(righthand->item[0]->GetImage());
}
if(inf.head.num > 0)
{
   eventcenter->RaiseEvent(new Event_RequestItemInfo(inf.head.id, &iTemp));
   for(unsigned int i = 0; i< inf.head.num; i++)
   {
      Item *item − new Item(eventcenter);
```

```
                item->Initialize(eventcenter->GetEngine(), iTemp, eventcenter);
                head->item[head->stack++] = item;
            }
            head->button->GetImage(0)->SetTexture(head->item[0]->GetImage());
            head->button->GetImage(1)->SetTexture(head->item[0]->GetImage());
            head->button->GetImage(2)->SetTexture(head->item[0]->GetImage());
        }
        EquipLeftHand();
        EquipRightHand();
        EquipHead();
    }
void GameSystem::Load(SystemInfoStruct inf)
    {
        gametime = inf.currenttime;
        SetVariable("Hour", gametime.hour);
        SetVariable(inf.globalvar);
        LinkedList<Quest*> *pointer = quest;
        while(pointer != null)
        {
            for(int i = 0; i< 50 && inf.quest[i].HasInfo(); i++)
            {
                if(inf.quest[i].name == pointer->data->GetName())
                {
                    pointer->data->LoadInfo(inf.quest[i]);
                    break;
                }
            }
            pointer = pointer->next;
        }
    }
```



รูปที่ 4-20 หน้าต่างสำหรับโหลดเกมจากเมนูหลัก

4.5.9 การสร้างเอฟเฟคในฉาก 3 มิติ

วิธีการสร้างเอฟเฟคนั้น มีคล้ายกับการเรียกวัตถุเข้ามาอยู่ในฉาก 3 มิติทั่วไป แต่จะมีคุณลักษณะพิเศษคือ เป็นโมเดล 3 มิติที่จะไม่มีการแสดงโมเดลอยู่ในฉากถาวร กล่าวคือจะเป็นการแสดงโมเดล 3 มิติพร้อมอนิเมชั่นในระยะที่กำหนด แล้วหายไปจากฉาก

โดยเมธอด Scene3D::AddEffect ทำหน้าที่เรียกเอฟเฟคเข้ามาในฉาก จากนั้นหากเอฟเฟคดังกล่าวมีคุณลักษณะที่ติดตามเป้าหมายได้ จะมีการอัพเดทตำแหน่งที่แสดงผลด้วยเมธอด Character::UpdateEffect

จากนั้นเมื่อเอฟเฟคหมดเวลาการแสดงผลลง จึงเข้าสู่เมธอด Scene3D::RemoveEffect เพื่อลบโมเดลเอฟเฟคออกจากฉาก ซึ่งในส่วนนี้จะไม่ทำลายโมเดลเอฟเฟคออกไปโดยตรง เนื่องจากบางเอฟเฟคมักมีการถูกเรียกซ้ำๆ อยู่ตลอดเวลา โดยมีเมธอด Scene3D::Cache ทำหน้าที่ในการย้ายโมเดลเอฟเฟคไปเก็บในหน่วยความจำชั่วคราวแทน จากนั้นก่อนจะเรียกใช้เอฟเฟคครั้งต่อไป จึงตรวจสอบความจำก่อนว่ามีเอฟเฟคที่ต้องการอยู่หรือไม่ และเอากลับมาใช้ใหม่หากพบ ผ่านเมธอด Scene3D::PopEffectFromCache

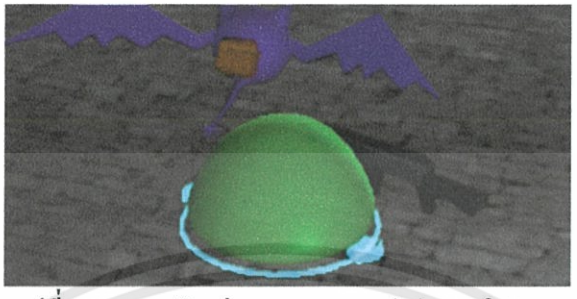
```
void Scene3D::AddEffect(Effect *effect)
        {
          if(this->effect)
          {
             LinkedList<Effect*> *pointer = this->effect;
```

```
            while(pointer->next != null) pointer = pointer->next;
            pointer->next = new LinkedList<Effect*>();
            pointer = pointer->next;
            pointer->data = effect;
          }
          else
          {
            this->effect = new LinkedList<Effect*>();
            this->effect->data = effect;
          }
        }
void Character::UpdateEffect()
        {
          LinkedList<Effect*> *ePointer = effect, *ePrev = null;
          while(ePointer)
          {
            if(ePointer->data)
            {
        //ให้หมุนและขยับตามตัวละคร
              ePointer->data->SetLocation(GetLocation());
              ePointer->data->SetRotation(GetRotation());
            }
            else
            {
              if(ePrev) ePrev->next = ePointer->next;
              else effect = ePointer->next;
              ePointer->next = null;
              delete ePointer;
              if(ePrev) ePointer = ePrev;
              else ePointer = effect;
            }
            ePrev = ePointer;
```

```
            if(ePointer)
                ePointer = ePointer->next;
        }
    }
bool Scene3D::RemoveEffect(Effect *effect)
    {
        LinkedList<Effect*> *pointer = this->effect, *previous = null;
        while(pointer != null)
        {
            if(pointer->data == effect)
            {
                Cache(pointer->data);
                pointer->data = null;
                return true;
            }
            previous = pointer;
            pointer = pointer->next;
        }
        return false;
    }
void Scene3D::Cache(Effect *element)
    {
        if(cache)
        {
            LinkedList<Effect*> *pointer = cache;
            while(pointer->next != null && pointer->data != null)
            {
                pointer = pointer->next;
            }
            if(pointer->data != null)
            {
                pointer->next = new LinkedList<Effect*>();
```

```
                pointer = pointer->next;
            }
            pointer->data = element;
        }
        else
        {
            cache = new LinkedList<Effect*>();
            cache->data = element;
        }
    }
Effect* Scene3D::PopEffectFromCache(string name)
    {
        if(name == "") return null;

        LinkedList<Effect*> *pointer = cache;
        while(pointer)
        {
            if(pointer->data)
            {
                if(pointer->data->GetName() == name)
                {
                    Effect *eff = pointer->data;
                    pointer->data = null;
                    return eff;
                }
            }
            pointer = pointer->next;
        }
        return null;
    }
```

เมื่อมีการใส่เอฟเฟคแล้ว จากรูปที่ 4-18 แสดงให้เห็นเอฟเฟควงสีฟ้ารอบตัวละครสไลม์ ซึ่งเป็นเอฟเฟคที่เกิดขึ้นเมื่อได้รับผลจากเวทมนตร์ สังเกตได้ว่าเอฟเฟคจะไม่ได้รับผลจากแสงเงาในฉาก ทำให้มชดูเด่นและคมชัดกว่า



รูปที่ 4-21 ภาพตัวอย่างการแสดงเอฟเฟคภายในฉาก

4.5.10 การประยุกต์ระบบเสียง

ระบบเสียงภายในเกมรองรับไฟล์เพลงเฉพาะสกุล .wav โดยตัวโปรแกรมสามารถรองรับไฟล์ที่จะใช้เล่นประกอบได้ถึง 200 ช่องเสียง โดยจะใช้ช่อง 0 แทนเสียงเพลงประกอบฉาก และช่องอื่นๆ สำหรับรองรับเสียงเอฟเฟคอื่นๆ โดยมีส่วนประกอบของโค้ดทำหน้าที่ในส่วนของระบบเสียงดังนี้

```
Element *ele = (Element*)param1;
int *arr = null;
if(ele == null)
{
    arr = (int*)SoundChannel_Unknown;
}
else if(ele->GetObjectType() == "Object3D")
{
    Object *obj = (Object*)param1;
    switch(obj->GetType())
{
                case ObjectType_Projectile:
                        arr = (int*)SoundChannel_Projectile;
                        break;
                case ObjectType_Player: case ObjectType_Partner:
                        arr = (int*)SoundChannel_Player;
```

```
                break;
            case ObjectType_Monster:
                arr = (int*)SoundChannel_Monster;
                break;
                case ObjectType_NPC:
                arr = (int*)SoundChannel_NPC;
                break;
            default:
                if((obj->GetType() & ObjectType_Item) != 0)
                {
                    arr = (int*)SoundChannel_Item;
                }
                else
                {
                frame->DisplayMessage("Unknown object3d type: " + obj->GetType());
                arr = null;
                return false;
                }
            break;
                }
        }
    else if(ele->GetObjectType() == "Object2D" || ele->GetObjectType() == "Scene2D")
        {
            arr = (int*)SoundChannel_Object2D;
        }
        else
        {
            frame->DisplayMessage("Unknown ObjectType: " + ele->GetObjectType());
        }
    if(arr == null) return false;
    int channel = eventcenter->GetEngine()->FindEmptySoundChannel(arr[0], arr[1]);
    if(channel > 0)
```

```
{
int file = datacenter->GetSound(ConvertToInt(command[1]));
int loop = ConvertToInt(command[2]);
if(loop <= 0) loop = 1;
else if(loop >= 64) loop = 63;
if(file >= 0)
eventcenter->GetEngine()->playSound(channel, file, loop);
}
else
{
frame->DisplayMessage("No available sound channel for sound #" + command[1]);
}
return ContinueScript(eventcenter, command, 2, length, param1, param2);
```

ในจุดนี้ตัวโปรแกรมเกมได้มีการแสดงผลและระบบต่าง ๆ จากการออกแบบเรียบร้อยทุกส่วนแล้ว ในหัวข้อถัดไปจะแสดงรายละเอียดของสคริปต์ ที่ใช้เป็นตัวเรียกเมธอดและระบบต่างๆ ที่ทำการประยุกต์เรียบร้อยแล้ว

4.6 รายละเอียดของไฟล์และสคริปต์ที่พัฒนาขึ้น

ไฟล์และสคริปต์ที่พัฒนาขึ้นนี้ มีจุดประสงค์หลักคือทำให้สามารถใส่ตรรกะ (Logic) ของตัวเกมเพื่อให้สามารถแก้ไขและสร้างระบบเกมได้ง่ายขึ้น รวมไปถึงช่วยให้เกิดการทดสอบโปรแกรมอย่างมีประสิทธิภาพ

4.6.1 การทำงานของไฟล์ .pss

ไฟล์ .pss เป็นไฟล์ Text ที่ผู้จัดทำได้พัฒนาขึ้นมาใช้ร่วมกับโปรแกรมเกม ถือว่าเป็น Resource File ของโปรแกรม สำหรับเรียกใช้ทรัพยากรต่างๆ เข้ามาในเกมได้ ดังที่กล่าวในบางส่วนของการประยุกต์ใช้ระบบแล้ว ซึ่งไฟล์เหล่านี้สามารถแก้ไขได้โดยไม่ต้องทำการ Compile ใหม่

รายละเอียดไฟล์ต่างๆ อ้างอิงจากตำแหน่งของไฟล์ภายในโครงงาน ซึ่งจะแบ่งได้เป็น Frame จัดการเกี่ยวกับการสร้างฉาก Game เกี่ยวกับระบบภายในเกม Model เกี่ยวกับการจัดเก็บโมเดลต่างๆ ภายในเกม และ Sound เกี่ยวกับระบบเสียงภายในเกม ดังแสดงในตารางที่ 4-1

ตารางที่ 4-1 แสดงตำแหน่งและคุณลักษณะของไฟล์ .pss

โฟลเดอร์ Frame	
ตำแหน่งไฟล์	**หน้าที่**
/<ชื่อฉาก>/Main.pss	ระบุ Path ไปยังไฟล์ที่เกี่ยวกับการสร้างฉากที่อยู่ในโฟลเดอร์เดียวกัน และตั้งสคริปต์เริ่มต้นที่จะทำการโหลดเมื่อเข้าฉากเช่น ตั้งให้แสดงหน้าจอ HUD Interface ตั้งมุมกล้องให้ตามผู้เล่น เป็นต้น
/<ชื่อฉาก>/object.pss	ระบุตำแหน่งและข้อมูลการทำ Instance โดยใช้เลข id อ้างอิงจากไฟล์ ModelInfo.pss ที่อยู่ในโฟลเดอร์ Model
/<ชื่อฉาก>/npc.pss	ระบุตำแหน่งของ NPC ที่จะให้ปรากฏในฉากนั้นๆ โดยอ้างอิงจาก id ในไฟล์ NPCInfo.pss ที่อยู่ในโฟลเดอร์ Game
/<ชื่อฉาก>/ spawnspot. pss	ระบุตำแหน่งที่ศัตรูจะเกิด เวลาที่ใช้ในการเกิดของศัตรู พร้อมตั้งชื่อ โดยอ้างอิงศัตรูจาก id ในไฟล์ MonsterInfo.pss ที่อยู่ในโฟลเดอร์ Game
ตำแหน่งไฟล์	**หน้าที่**
Scene3DInfo.pss	ระบุชื่อและตำแหน่ง Path ไปยังไฟล์ Main.pss ของแต่ละฉาก ให้โปรแกรมสามารถเข้าไปอ่านรายละเอียดภายในฉากได้ โดยฉากที่อยู่เป็นอันดับแรกจะถูกโหลดเข้ามาแสดงผลเมื่อเปิดโปรแกรม คุณสมบัตินี้ใช้ในการเข้าถึงฉากได้ง่ายกับใช้อ้างอิงในเซฟไฟล์
Scene2DInfo.pss	ระบุชื่อและตำแหน่ง Path ไปยังไฟล์ที่เกี่ยวข้องกับการแสดงฉาก 2 มิติให้โปรแกรมสามารถเข้าไปอ่านรายละเอียดภายในฉาก ประกอบไปด้วยหน้าต่าง gui, เมนูภายในเกม, หน้าต่าง Inventory, หน้าต่างแสดงแผนที่, หน้าต่างแสดงค่าพลังของผู้เล่น, หน้าต่างแสดงทักษะของผู้เล่น, หน้าต่างแสดงค่าพลังของคู่หู, หน้าต่างสำหรับบทสนทนา, หน้าต่างไอเท็ม, หน้าต่างการซื้อขายไอเท็ม, หน้าต่างเซฟเกม, หน้าต่างโหลดเกม และอื่น ๆ
โฟลเดอร์ Game	
ตำแหน่งไฟล์	**หน้าที่**
/Conversation/<ชื่อบทสนทนา>.pss	ระบุรายละเอียดของบทสนทนา โดยแต่ละบทสนทนา 1 ช่วง ประกอบไปด้วยข้อความที่จะแสดง หน้าของผู้พูด ตำแหน่งของหน้าผู้พูด ชื่อผู้พูด และสคริปต์ที่จะถูกเรียก
/Monster/<ชื่อไฟล์ศัตรู>.pss	ระบุรายละเอียดของศัตรูแต่ละตัว ประกอบไปด้วยโมเดลที่ใช้, Keyของอนิเมชั่น, ค่าพลังต่าง ๆ, รูปแสดงตัวแทน, รูปแบบ AI และทักษะที่ศัตรูใช้ได้ รวมไปถึงสคริปต์ที่ศัตรูอาจจะเรียก

ตำแหน่งไฟล์	หน้าที่
/NPC/<ชื่อไฟล์ NPC>.pss	ระบุรายละเอียดของNPC แต่ละตัว ประกอบไปด้วยโมเดลที่ใช้, Key ของอนิเมชั่น, ระยะที่ผู้เล่นสามารถตอบสนองได้ และสคริปต์ที่จะถูกเรียกเมื่อตอบสนองกับ NPC ตัวนั้น ๆ
/Monster/<ชื่อไฟล์ภารกิจ>.pss	ระบุรายละเอียดของภารกิจแต่ละอัน ประกอบไปด้วยชื่อและรายละเอียดภารกิจ และสคริปต์ที่จะถูกเรียกเมื่อภารกิจผ่านไปถึงจุดๆ หนึ่ง
ConversationInfo.pss	ระบุ id อ้างอิงและ Path ของไฟล์ข้อมูลบทสนทนา
MonsterInfo.pss	ระบุ id อ้างอิงและ Path ของไฟล์ข้อมูลศัตรู
NPCInfo.pss	ระบุ id อ้างอิงและ Path ของไฟล์ข้อมูล NPC
QuestInfo.pss	ระบุ id อ้างอิงและ Path ของไฟล์ข้อมูลภารกิจ
Font.pss	ระบุ id อ้างอิงและคุณลักษณะของ font ที่ใช้ในการแสดงข้อความในฉาก 2 มิติ
WayPointInfo.pss	ระบุ Path ที่จะให้ NPC หรือวัตถุใดๆ ที่เคลื่อนที่ในลักษณะตายตัว ให้เคลื่อนที่ไปตามเส้นทางที่ระบุไว้ภายในไฟล์นี้
Item.pss	ระบุ id และผลต่างๆ ที่เกียวข้อกับระบบไอเท็มภายในเกม
TradeInfo.pss	ระบุรูปแบบของร้านค้าและของที่จะขาย โดยอ้างอิง id จากไฟล์ Item.pss
UnknownItemList.pss	ระบุไอเท็มที่จะได้รับเมื่อนำไปแลกเปลี่ยนกับร้านแลกของ โดยอ้างอิง id จากไฟล์ Item.pss
PlayerInfo.pss	ระบุโมเดล, Keyของอนิเมชั่น, การเรียนรู้ทักษะต่างๆ ของผู้เล่น
Partner.pss	ระบุโมเดล, Keyของอนิเมชั่น, การเรียนรู้เวทมนตร์ของคู่หู
AttributeInfo.pss	ระบุรายละเอียดและผลที่เกิดขึ้นของสถานะพิเศษภายในเกม
SkillInfo.pss	ระบุรายละเอียดของทักษะทั้งหมดที่มีในเกมผ่านสคริปต์
Control.pss	ระบุปุ่มที่ใช้ในการควบคุมเกมทั้งหมด
Effect.pss	ระบุid และโมเดลของเอฟเฟคภายในเกม
Script.pss	ระบุชุดของสคริปต์ สำหรับให้สคริปต์บางประเภทเรียก
โฟลเดอร์ Model	
ตำแหน่งไฟล์	**หน้าที่**
ModelInfo.pss	ระบุ id, Pathที่อยู่ของโมเดล 3 มิติ และระยะ Collision ของโมเดล
โฟลเดอร์ Sound	
ตำแหน่งไฟล์	**หน้าที่**
Sound.pss	ระบุ id และ Path ของไฟล์เสียงเอฟเฟค

4.6.2 สคริปต์

สคริปต์ต่างๆ ทำหน้าที่ในการเรียกเมธอดภายในโปรแกรมโดยตรง ซึ่งเป็นส่วนสำคัญที่ทำให้เกิดตรรกะของเกมขึ้นมา โดยแต่ละสคริปต์ดังแสดงในตารางที่ 4-2 จะมีหน้าที่เหมือนกัน แต่อาจมีบทบาทแตกต่างกันตามแต่ชนิดของไฟล์ .pss ในช่วงแรกของการพัฒนาจะสามารถกรอกสคริปต์เหล่านี้ผ่าน Console Box ได้โดยตรง เพื่อให้สามารถทดสอบผลของไอเท็มและระบบเกมอื่นๆ ได้อย่างสะดวก

โดยค่า arg ของสคริปต์บางตัวนั้น สามารถใส่เป็นค่าที่กำหนดได้ โดยที่ค่า val1 val2 ห้ามเป็นค่า # ทั้งหลาย ยกเว้น #Variable หรือ #GlobalVar ดังต่อไปนี้

#Variable(val) ค่าตัวแปรที่ชื่อ val จากตัววัตถุที่เรียกสคริปต์

#GlobalVar(val) ค่าตัวแปรทั่วไปที่ชื่อ val

#Random(val1/val2) สุ่มเลขจำนวนเต็มระหว่าง val1 และ val2

#HitRange(val1) ระยะโจมตีของตัวละคร โดยหาก val1 เป็น vNear คือใกล้มาก Nearคือใกล้ Medium คือปานกลาง Far คือ ไกล และ vFar คือไกลมาก

#Status(val1) ค่าสถานะของตัวละครตามโมเดลเช่น Idle

#CharAttribute(val1/val2/val3) ค่าสถานะ val2 ของตัวละครชื่อ val1 โดยค่าที่ออกไปจะคูณกับ val3 ก่อน

#Multiply(val1/val2) นำค่า val1 คูณกับ val2

#Add(val1/val2) นำค่า val1 บวกกับ val2

#Subtract(val1/val2) นำค่า val1 ลบกับ val2

#Divide(val1/val2) นำค่า val1 หารกับ val2

ตารางที่ 4-2 แสดงรายละเอียดของสคริปต์

สคริปต์	ผลที่ได้
DebugString,<arg>	แสดงข้อความ arg บนคอนโซลดีบั๊กของ Visual Studio เช่น DebugString, “Error”
DisplayMessage, <arg1>	แสดงข้อความ arg1 บนหน้าจอด้านบนซ้ายของเกม
ShowScene2D, <arg>	ให้หน้าต่าง 2 มิติชื่อ arg (อ้างอิงจากไฟล์ Scene2DInfo.pss) ให้ปรากฏออกมา เช่น ShowScence2D,gui คือให้แสดงหน้าจอ gui ของเกมขึ้นมา
HideScene2D , <arg1>, <arg2>	ซ่อนหน้าต่าง 2 มิติชื่อ arg1 ที่แสดงอยู่แล้วบนจอให้หายไป และแทนที่ด้วยหน้าจอ 2 มิติ arg2 (อ้างอิงจากไฟล์ Scene2DInfo.pss) หากไม่ระบุจะซ่อนหน้าต่างที่ระบุเท่านั้นเช่น HideScence2D,gui, save ยกเลิกการแสดง gui และแสดงหน้าต่างเซฟแทน
freezegame	หยุดเวลาและตัวเกม
unfreezegame	ทำให้เกมกลับมาจากการ freeze ใช้คู่กับ freezegame เช่น เข้าหน้าต่างเมนู

ตำแหน่งไฟล์	หน้าที่
RunScript, <arg1>	เรียกสคริปต์ชื่อ arg1 (อ้างอิงจากไฟล์ Script.pss)
SetCallerToObject3D, <arg1>,<arg2>,<arg3>	ตั้งให้วัตถุ 3 มิติชื่อ arg1 ที่อยู่ในฉากเป็นตัวเรียกสคริปต์ใดๆ ใช้คู่กับสคริปต์บางสคริปต์ที่ต้องการวัตถุในการเรียก หากใช้ทั้ง arg1, arg2, arg3 จะรับเป็นค่าตัวเลขแสดงตำแหน่งแทน แล้วจะเลือกวัตถุ 3 มิติที่อยู่ใกล้ตำแหน่งนั้นเป็นตัวเรียกสคริปต์
SetCallerToObject2D, <arg1>, <arg2>	ตั้งวัตถุ 2 มิติในฉาก arg1 ที่มีชื่อว่า arg2 เป็นตัวเรียกสคริปต์
SetCallerToScene2D, <arg1>	ตั้งฉาก 2 มิติชื่อว่า arg1 เป็นตัวเรียกสคริปต์
HighlightSource, <arg1>	เปิดหรือปิดไฮไลท์ของวัตถุที่เรียกสคริปต์ โดยที่หากค่า arg มากกว่า 0 จะเปิด ถ้าเท่ากับ 0 จะปิด หากค่าคือ toggle จะสลับเปิด/ปิดกับค่าปัจจุบัน
SetVisible, <arg1>	ตั้งให้วัตถุที่เรียกสคริปต์หายไปหาก arg1 เป็น False
SetCameraLocation, <arg1>, <arg2>, <arg3>	ย้ายตำแหน่งของกล้องไปที่ตำแหน่ง x = arg1, y = arg2, z = arg3
SetCameraAngleX, <arg1>	ย้ายมุมกล้องแนวเลียดกับพื้นไปที่มุม arg1
SetCameraAngleY, <arg1>	ย้ายมุมกล้องแนวสูงไปที่มุม arg1
SetCameraFollow, <arg1>	ให้กล้องย้ายไปและเคลื่นที่ตามวัตถุไอดี arg1 โดยอัตโนมัติ
PanCameraToAngle, <arg1>,<bool>, <arg2>	ให้กล้องค่อยๆ หมุนไปยังมุม arg1 ทิศทางทวนเข็มนาฬิกา(True) หรือ ทิศทางตามเข็มนาฬิกา(False) ภายในจำนวน arg2 เฟรม
PanCameraToPosition, <arg1>, <arg2>, <arg3>, <arg4>	ให้กล้องค่อยๆ เคลื่อนตัวไปยังมุม จุด x = arg1, y = arg2, z = arg3 ภายในจำนวน arg4 เฟรม
PanCameraToObject, <arg1>, <arg2>	ให้กล้องค่อยๆ ย้ายตำแหน่งไปยังวัตถุ id arg1 ภายในจำนวน arg2 เฟรม
AdvanceTime, arg1	ทำให้เวลาในเกมถูกเดินเพิ่มไป arg1 ชั่วโมงทันที
SetObjectLocation, <arg1>, <arg2>, <arg3>	ย้ายวัตถุที่เรียกสคริปต์ไปอยู่ตำแหน่ง x = arg1, y = arg2, z = arg3 ของฉากนั้น ๆ

ตำแหน่งไฟล์	หน้าที่
DestroyObject3D, <arg1>	ลบวัตถุชื่อ arg1 ออกจากฉาก หาก arg1 คือ self จะวัตถุที่เรียกสคริปต์ออกไป
MoveForward	ให้ตัวละครเคลื่อนไปข้างหน้า 1 เฟรมจากจุดเดิม
ChangeLevel, <arg1>, <arg2>, <arg3>, <arg4>	เปลี่ยนฉากสามมิติไปฉากที่ชื่อ arg1 (อ้างอิงจากไฟล์ Scenc3DInfo.pss)โดยให้ผู้เล่นไปปรากฏตัวในตำแหน่ง x = arg2, y = arg-, z = arg4 ของฉากใหม่ หากไม่ระบุ ตำแหน่งผู้เล่นจะปรากฏตัวในพิกัด x y z เดียวกับฉากที่อยู่ปัจจุบัน
AddEffectAtCaller, <arg1>	ให้เอฟเฟคไอดี arg1 (อ้างอิงจากไฟล์ Effect.pss) ไปปรากฏที่ตำแหน่งของวัตถุที่เรียกสคริปต์
RemoveEffect	นำเอฟเฟคที่เรียกสคริปต์ออกจากฉาก
SetStatus, <arg1>	ตั้งให้วัตถุ 3 มิติที่เรียกอยู่ในสถานะ arg1 (อ้างอิงจาก key ของโมเดล)
SetAnimationKey, <arg1>	ตั้งให้วัตถุ 3 มิติที่เรียกแสดงอนิเมชั่นชื่อ arg1 (อ้างอิงจาก key ของโมเดล)
ShowDialogBox, <arg1>	สร้างกล่องคำพูดเหนือวัตถุ 3 มิติที่เรียกสคริปต์ข้อความ arg1
SeizeControl,<arg1>	ตัดเข้าฉาก Cutscene หาก arg1 มีค่าเป็น 1 และออกจาก Cutscene กลับมาเป็นสถานะปกติเมื่อมีค่าเป็น 0
StartConversation, <arg1>	เริ่มบทสนทนาที่ arg1 (อ้างอิงจากไฟล์ ConversationInfo.pss) ใช้คู่กับ SeizeControl
SetVariable, <arg1>, <arg2>	ตั้งค่าตัวแปรชื่อ arg1 ให้กับสิ่งที่เรียกสคริปต์เป็นค่า arg2 โดยตัวแปรนี้จะขึ้นอยู่กับสิ่งที่เรียกสคริปต์เท่านั้น
SetGlobalVariable, <arg1>, <arg2>	ตั้งตัวแปรทั่วไปชื่อ arg1 ให้มีค่า arg2 โดยตัวแปรทั่วไปสามารถนำไปตรวจสอบกับเงื่อนไขที่ระบุไว้กับฉาก, NPC หรือศัตรู ในส่วนของ Variable ได้ โดยจะเรียกสคริปต์ที่กำหนดไว้ทันทีเมื่อผู้เล่นเข้าฉาก หรือค่านี้เกิดการเปลี่ยนแปลงเป็นค่าที่กำหนดไว้ในส่วน Variable นั้น ๆ
SetCharaterScript, <arg1>, <arg2>	ตั้งสคริปต์ของผู้เล่นหรือศัตรูเมื่อ arg1 (ontakedamage เมื่อถูกโจมตี, onhpbelow เมื่อHP ต่ำกว่าที่กำหนด หรือ ondie เมื่อ HP หมด) ให้ตัวละครที่เรียกสคริปต์นี้เป็นสคริปต์ที่ arg2 (อ้างอิงจากไฟล์ Script.pss)
SetQuestProgress, <arg1>, <arg2>	ทำให้เควสที่ชื่อ arg1 (อ้างอิงจากไฟล์ QuestInfo.pss) มีเนื้อเรื่อง (Progress) อยู่ในขั้นที่ arg2
ConvertToNPC	เปลี่ยนวัตถุ 3 มิติที่เรียกสคริปต์เป็น NPC

ตำแหน่งไฟล์	หน้าที่
AddAtk/AddDef/AddAgi /AddAp/AddSp, <arg1>	เพิ่มค่า atk หรือ def หรือ agi หรือ ap หรือ sp ให้ผู้เล่นเป็นค่า arg1 ถาวรเช่น AddAtk,1 เพิ่มค่าพลัง Atk ให้ผู้เล่นถาวร 1 แต้ม
GainEXP, <arg1>	เพิ่มค่า exp เป็นจำนวน arg1 ให้กับผู้เล่น
SetHPMultiplier, <arg1>	ตั้งตัวคูณค่า HP ของสิ่งที่เรียกเป็นค่า arg1 ทำให้มี HP เพิ่มขึ้นเป็น arg1 เท่า
SetDefMultiplier, <arg1>	ตั้งตัวคูณค่า Def ของสิ่งที่เรียกเป็นค่า arg1 ทำให้มี Def เพิ่มขึ้นเป็น arg1 เท่า
AddItem, <arg1>, <arg2>	ให้ไอเท็มid arg1 (อ้างอิงจากไฟล์ Item.pss) พุ่งออกมาจากวัตถุ 3 มิติที่เป็นตัวเรียก โดยมีโอกาส arg2 เปอร์เซน โดย arg2 คือค่าจำนวนเต็มระหว่าง 0 - 100
heal, <arg1>	เพิ่ม HP ของตัวละครที่เรียกสคริปต์เป็นค่า arg1 แสดงตัวเลขเป็นสีเขียว
poison, <arg1>	ลด HP ของตัวละครที่เรียกสคริปต์เป็นค่า arg1 โดยไม่คำนวณสูตรความเสียหาย
GainMP,<arg1>	เพิ่ม SP ให้กับผู้เล่น หากผู้เล่นเป็นผู้เรียก เพิ่ม MP ให้คู่หู หากคู่หูเป็นผู้เรียก เป็นจำนวน arg1 หน่วย
CollectItem, <arg1>	เก็บไอเท็ม โดยให้ตัวละครไอดี arg1 เป็นผู้เก็บ หาก arg1 มีค่าเป็น collector จะให้ตัวละครที่อยู่ใกล้สุดเก็บ
GainMoney, <arg1>	เพิ่มเงินให้ผู้เล่น arg1 หน่วย
ConsumeItem, <arg1>	ทำให้ไอเทมของผู้เล่นที่สวมใส่อยู่ในตำแหน่ง arg1 (lefthand, righthand หรือ head) หายไป 1 ชิ้น
ReleaseProjectile, <arg1>	ไอเทมของผู้เล่นที่สวมใส่อยู่ในตำแหน่ง arg1 (lefthand, righthand หรือ head) ถูกโยนออกไปในรูปแบบ projectile
AttackTarget, <arg1>	โจมตีเป้าหมายที่ตัวละครเลือก หากตัวเรียกเป็น projectile จะโจมตีไปตามค่า arg1 หากตัวละครเรียก จะโจมตีเป้าหมายโดยที่บวกค่า arg1 เข้าไปกับค่า Atk ของผู้เล่น
CreatAttackArcAt, <arg1>, <arg2>, <arg3>, <arg4>, <arg5>, <arg6>, <arg7>, <arg8>, <arg9>	โจมตีเป็นบริเวณส่วนหนึ่งของวงกลม โดยให้ศูนย์กลางอยู่ที่ x = arg1, y = arg2, z = arg3) แล้วให้มองไปที่ตำแหน่ง x = arg4, y = arg5, z = arg6 โดยมีมุมโจมตี arg7 มีค่าพลังโจมตีเพิ่มขึ้นจากการคำนวณสูตรโจมตีปกติ arg8 เท่าโดยที่ arg9 คือ เป้าหมายที่กำหนด โดย attackall คือถูกโจมตีได้ทั้งผู้เล่นและศัตรูที่อยู่ในระยะ dontattackplayer ผู้เล่นจะไม่ถูกโจมตีแม้อยู่ในระยะ dontattackmonster ศัตรูจะไม่ถูกโจมตีแม้อยู่ในระยะ
CreateAttackSphere, <arg1>, <arg2>, <arg3>	สร้างขอบเขตที่จะสร้างความเสียหายจากวัตถุ 3 มิติที่เรียก มีรัศมีเป็น arg1 สร้างความเสียหายเพิ่มจากที่ตัวละครเรียกเท่ากับ arg2 และ arg3 คือเป้าหมายที่กำหนด

ตำแหน่งไฟล์	หน้าที่
CreateAttackArc, <arg1>, <arg2>, <arg3>, <arg4>	สร้างของเขตการโจมตีข้างหน้าตัวละครที่เรียก เป็นบริเวณส่วนหนึ่งของวงกลม โดยมีรัศมี arg1 ความกว้าง arg2 มีค่าพลังโจมตีเพิ่มเป็นค่า arg3 และ arg4 คือเป้าหมายที่กำหนด โดย attackall คือถูกโจมตีได้ทั้งผู้เล่นและศัตรูที่อยู่ในระยะ dontattackplayer ผู้เล่นจะไม่ถูกโจมตีแม้อยู่ในระยะ dontattackmonster ศัตรูจะไม่ถูกโจมตีแม้อยู่ในระยะ
AddAttribute, <arg1>	เพิ่มสถานะ arg1 (อ้างอิงจากไฟล์ AttributeInfo.pss) ให้เป้าหมายของตัวละครที่เรียกสคริปต์ หากไม่มีเป้าหมาย สถานะนี้จะถูกเพิ่มให้กับตัวละครที่เรียกเอง
EditAttribute, <arg1>, <arg2>, <arg3>	แก้ข้อมูลสถานะไอดี arg1 แก้ค่า arg2 ให้มีค่าใหม่เป็น arg3
SetCharacterLevel, <arg1>	ตั้งให้ตัวละครที่เรียกสคริปต์มีเลเวล arg1 โดย atk, def, agi จะถูกสุ่มดัดแปลงให้สอดคล้องกับเลเวลใหม่
SetCallerToSpawnPoint, <arg1>	ตั้งจุด spawn ไอดี arg1 (อ้างอิงจากไฟล์ Monster.pss) เป็นตัวเรียกสคริปต์
SpawnMonster, <arg1>	ให้ศัตรูเกิดจากจุดเกิดที่เป็นตัวเรียกสคริปต์ทันทีเป็นจำนวน arg1 ตัว
CreateProvokeSphere, <arg1>	ทำให้ศัตรูที่อยู่รอบๆวัตถุที่เรียกสคริปต์นี้ในรัศมี arg1 วิ่งเข้าไปโจมตีผู้เล่น
PlaySound, <arg1>, <arg2>	เล่นเสียงประกอบไอดี arg1 (อ้างอิงจากไฟล์ Sound.pss) เป็นจำนวน arg2 รอบ
StartBGM, <arg1>	เล่น BGM จาก Path arg1 ทันที โดยหากต้องการเปลี่ยนเพลงต้องทำการ StopBGM ก่อน
StopBGM	หยุดเล่น BGM ที่เล่นอยู่ในฉากนั้นทันที

บทที่ 5
สรุปผลการดำเนินงาน และข้อเสนอแนะ

5.1 สรุปผลการดำเนินงาน

เกมรูปแบบ 3 มิติสำหรับเล่นบนเครื่องคอมพิวเตอร์พีซีนั้น ส่วนใหญ่ล้วนแล้วแต่ใช้ความสามารถของ DirectX เข้ามาใช้ในส่วนของการแสดงผล รวมไปถึงอุปกรณ์นำเข้าข้อมูลเพื่อควบคุมเกมสำหรับเครื่องคอมพิวเตอร์พีซีนั้น มักจะทำออกมาเพื่อรองรับข้อมูลนำเข้าจากเมาส์และคีย์บอร์ดเท่านั้น ด้วยความสนใจในความสามารถของ DirectX ในการประมวลผลภาพกราฟิก 3 มิติ และความสามารถของ Wiimote ที่ทำให้ผู้เล่นสามารถสัมผัสประสบการณ์ใหม่ๆ ผ่านตัวเกมได้ โครงงานพิเศษนี้จึงเน้นไปที่กระบวนการการสร้างเกม 3 มิติ ตั้งแต่เริ่มต้น จนได้เป็นเกมรีโอวาโน ที่มีความสามารถดังกล่าวข้างต้นออกมา

โดยคณะผู้จัดทำ ได้ทำการพัฒนาโปรแกรมเกมรีโอวาโนนี้ขึ้นมา โดยเริ่มตั้งแต่การออกแบบเกม ไปจนถึงการสร้างระบบเกมทั้งหลายในรูปแบบของเกม Action-RPG จากผลการดำเนินงาน ทำให้ได้มาซึ่งวิธีการประยุกต์ใช้ DirectX การควบคุมเกมด้วย Wiimote ขั้นตอนวิธีต่างๆ ในการจัดการเกี่ยวกับระบบของเกมในทุกๆ ระบบ และทำการใส่เนื้อหาของเกมต่างๆ เข้าไป จนได้มาเป็นเกมรีโอวาโน ที่มีระบบและความสามารถต่างๆ ตามการออกแบบที่วางไว้ขึ้นมาได้เป็นผลสำเร็จตามจุดประสงค์ที่วางไว้

5.2 ข้อเสนอแนะ

ในขั้นตอนระหว่างการพัฒนาตัวเกมนั้นได้พบกับปัญหาและอุปสรรคหลายประการ โดยเฉพาะปัญหาที่เกิดขึ้นในช่วงของการนำโปรแกรมไป Implement ให้เข้ากับการออกแบบที่วางไว้ ทำให้เมื่อมีการเพิ่มข้อมูลเข้าไปในตัวเกมแล้ว เมื่อทำการทดสอบสิ่งที่เพิ่มเข้ามา จะพบกับ Bug และข้อผิดพลาดต่างๆ ปรากฏออกมาบ่อยครั้ง ซึ่งส่วนใหญ่จะเป็นปัญหาที่ผู้จัดทำคาดไม่ถึง และมักค้นพบด้วยความบังเอิญ ซึ่งเหตุบังเอิญเหล่านี้ มักเป็นส่วนที่เป็นปัญหาเมื่อเล่นจริง เช่น เมื่อทำการโจมตีด้วยไอเท็มสำหรับโยนออกไประหว่างที่ไอเท็มชิ้นปัจจุบันกำลังลอยอยู่กลางอากาศ แล้วทำการหยุดเกม เปลี่ยนเอาไอเท็มชิ้นอื่นมาใส่แทนทีทันที จะทำให้โปรแกรมเกิดข้อผิดพลาดและหยุดการทำงาน เป็นต้น ซึ่งปัญหานี้เป็นปัญหาเกี่ยวกับการทำงานร่วมกันระหว่างระบบไอเท็ม และระบบการแสดงผล

กล่าวโดยสรุปคือในการพัฒนาเกมนั้น สิ่งจำเป็นนอกจากความรู้ในการออกแบบเกม และประยุกต์ใช้ระบบต่างๆ ให้เนื้อหาภายในเกมมีความน่าสนใจแล้ว จำเป็นต้องมีการทดสอบระบบเกมอย่างสม่ำเสมอด้วย โดยจำเป็นต้องมีการทดสอบเริ่มตั้งแต่ระบบใหม่ที่เพิ่มเข้ามา แล้วย้อนไปยังระบบเก่าที่ทำงานได้แล้ว เพื่อให้มั่นใจว่าระบบที่เพิ่มเข้ามานั้น ไม่ส่งผลให้เกิดข้อผิดพลาดเพิ่มขึ้นมา จนทำโปรแกรมเกมหยุดการทำงาน

สำหรับการพัฒนาโครงงานชิ้นนี้ต่อไป อาจเป็นไปได้หลายเส้นทาง ได้แก่

- พัฒนาและปรับปรุงระบบการประมวลผลและแสดงผลของภาพ 3 มิติของโปรแกรมเกมนี้ ให้ใช้ทรัพยากรของเครื่องคอมพิวเตอร์ลดลง ด้วยการค้นหาเทคนิคต่างๆ เข้ามาประยุกต์ใช้เพิ่มเติม
- ประยุกต์ใช้ส่วนของการนำเข้าข้อมูลจาก Wiimote ในส่วนคุณสมบัติอื่นๆ ที่อุปกรณ์ควบคุมเกมอื่นไม่มี เนื่องจากขอบเขตของปัญหานี้ กำหนดว่าในการควบคุมเกม จะสามารถใช้ได้ทั้งอุปกรณ์พื้นฐานทั่วไป และ Wiimote เพื่อให้ผู้เล่นได้มีทางเลือกในการเล่นเกมเพิ่มขึ้น จึงมีการประยุกต์ใช้ในระดับหนึ่งเท่านั้น อีกทั้งยังมีอุปกรณ์เสริมที่ใช้กับ Wiimote นอกจาก Nunchuck อีก ซึ่งเป็นจุดเด่นอีกประการหนึ่งของ Wiimote
- พัฒนาตัวช่วยสำหรับการพัฒนาระบบเกม โดยใช้พื้นฐานจากสคริปต์ที่สร้างขึ้นมา ให้ได้มาซึ่งระบบสำหรับพัฒนาเกมใหม่ๆ หรือต่อยอดจากเกมเดิมต่อไป เนื่องจากส่วนของการประยุกต์ใช้ด้วยสคริปต์นั้น จำเป็นต้องมีการแก้ไขให้สอดคล้องกันกับที่ส่วนของโปรแกรมรองรับ รวมไปถึงข้อผิดพลาดในการพิมพ์ได้ง่ายกว่า ซึ่งไม่เหมือนกับการเขียนด้วยโปรแกรมที่มีตัว Debugger สำหรับตรวจสอบข้อผิดพลาดของการทำงานได้ทันที

เอกสารอ้างอิง

[1] Anonymous. (ไม่ปรากฏปีพิมพ์). DirectX Tutorial. สืบค้นเมื่อ 25 ธันวาคม 2011 จาก http://directxtutorial.com/

[2] Keith Ditchburn. (ไม่ปรากฏปีพิมพ์). Picking. สืบค้นเมื่อ 4 พฤษภาคม 2012, จาก http://www.toymaker.info/Games/html/picking.html

[3] Hubert Nguyen. (2007). Chapter 8: Summed-Area Variance Shadow Maps. สืบค้นเมื่อ 30 เมษายน 2012, จาก http.developer.nvidia.com/GPUGems3/gpugems3_ch08.html

[4] Chad Vernon. (2012). Frustum Culling. สืบค้นเมื่อ 5 พฤษภาคม 2012, จาก www.chadvernon.com/blog/resources/directx9/frustum-culling/

[5] Anonymous. (ไม่ปรากฏปีพิมพ์). Wii Remote. สืบค้นเมื่อ 16 มิถุนายน 2011 จาก http://en.wikipedia.org/wiki/Wii_Remote

[6] Anonymous. (2010). WiiYourSelf!. สืบค้นเมื่อ 20 มิถุนายน 2010 จาก http://wiiyourself.gl.tter.org/

[7] Anonymous. (2012). DirectX9 Hardware Instancing. สืบค้นเมื่อ 15 กรกฎาคม 2012, จาก http://www.gamedev.net/topic/625073-directx9-hardware-instancing-draws-only-one-instance/

[8] Nati Namvong. (2007). ทำ Wii Sensor bar อย่างง่าย. สืบค้นเมื่อ 30 ตุลาคม 2012, จาก http://www.gamedev.net/topic/625073-directx9-hardware-instancing-draws-only-one-instance/

ภาคผนวก ก.
การเชื่อมต่อ Wiimote กับคอมพิวเตอร์

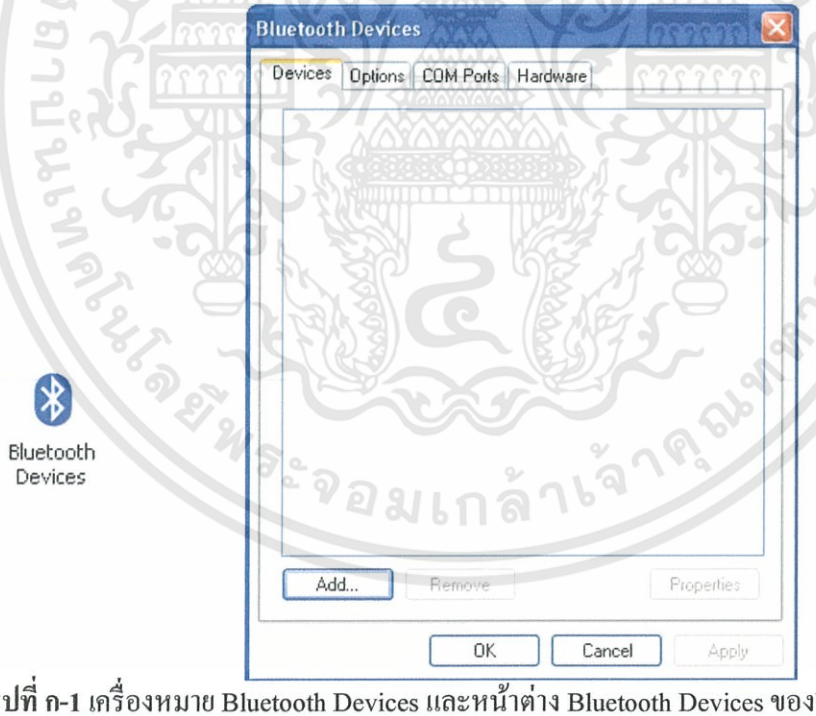
ภาคผนวก ก.
การเชื่อมต่อ Wiimote กับคอมพิวเตอร์

การเชื่อมต่อ Wiimote กับคอมพิวเตอร์

สำหรับการติดตั้ง Wiimote ผ่าน Bluetooth นั้น ในระบบปฏิบัติการ WindowsXP และ Windows7 จะมีการตั้งค่าคลล้ายคลึงกัน โดยหากคอมพิวเตอร์ที่ไม่มีการติดตั้ง Bluetooth ให้เสียบตัว USB Bluetooth เข้ากับเครื่องคอมพิวเตอร์ และรอให้ Windows ติดตั้ง Driver ของ Bluetooth ให้เรียบร้อยก่อน จากนั้นเตรียมถ่าน AA 2 ก้อนใส่วีโมตให้พร้อมก่อนเริ่มทำการเชื่อมต่อ โดยการเชื่อมต่อของ Windows XP และ Windows 7 นั้น จะแตกต่างกันเล็กน้อย รายละเอียดดังต่อไปนี้

การติดตั้ง Wiimote ผ่าน Bluetooth สำหรับ Windows XP

1. เข้าสู่ Control Panel เลือกหัวข้อ Bluetooth Devices หรือกดเครื่องหมาย  ที่อยู่บนtaskbar

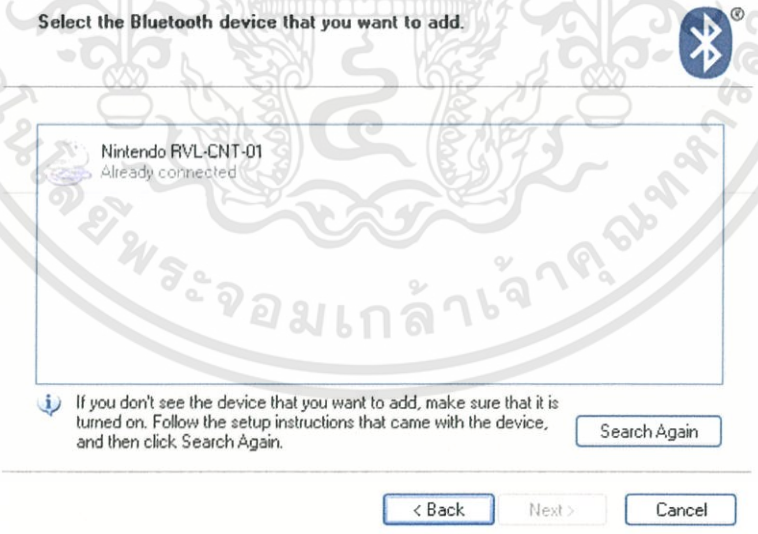


รูปที่ ก-1 เครื่องหมาย Bluetooth Devices และหน้าต่าง Bluetooth Devices ของWindows XP

2. ทำการเพิ่ม Wiimote เป็นอุปกรณ์ Bluetooth โดยเริ่มจากกดปุ่มAdd... จะมีหน้าต่าง Add Bluetooth Device Wizard ขึ้นมา เลือกหัวข้อ My device is set up and ready to be found แล้วกดปุ่ม Next

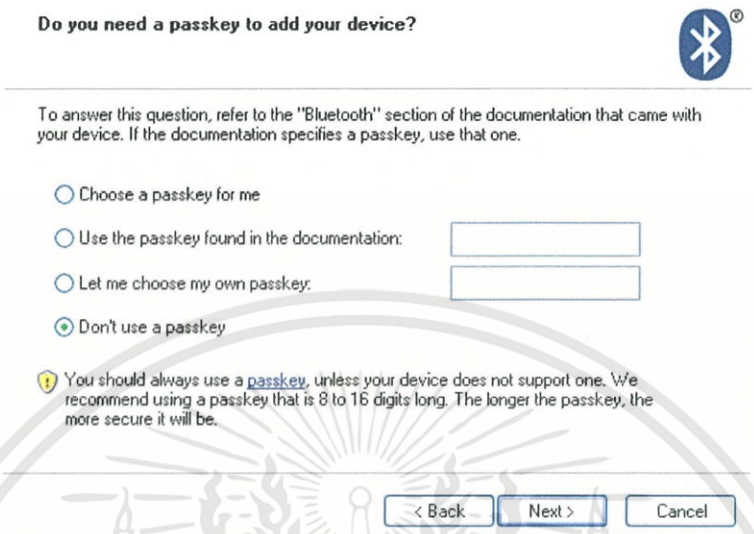
รูปที่ ก-2 หน้า Welcome ของ Add Bluetooth Device Wizard

3. เข้าสู่หน้าจอค้นหาอุปกรณ์ Bluetooth ให้กดปุ่ม 1 และ 2 ของ Wiimote ค้างเอาไว้ โดยสังเกตแสงสีฟ้าจากวีโมตจะกระพริบเมื่อกดปุ่ม 1 และ 2 พร้อมกัน ให้กดค้างไว้จนกว่าคอมพิวเตอร์จะตรวจพบวีโมตในชื่อ Nintendo RVL-CNT-01 จากนั้นเลือกแล้วกด Next > เพื่อทำขั้นตอนต่อไป



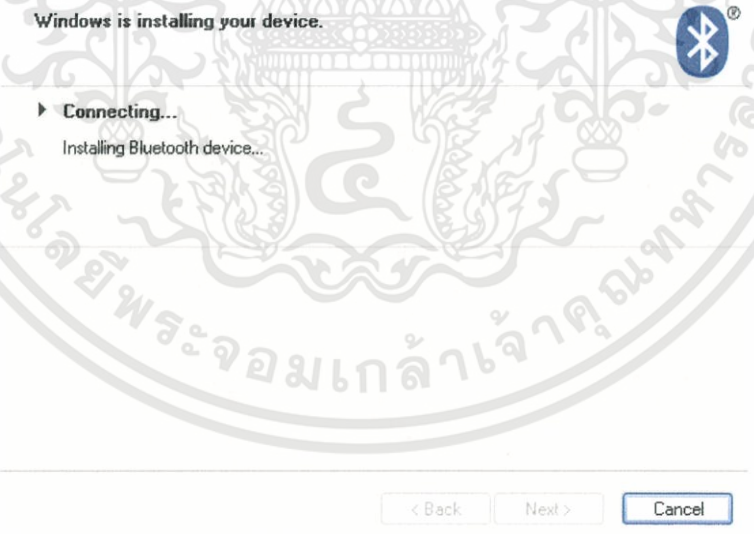
รูปที่ ก-3 หน้าจอค้นหาอุปกรณ์ Bluetooth เมื่อพบวีโมตแล้ว

4. เข้าสู่หน้าต่างกำหนด Passkey ซึ่งในที่นี้ให้เลือกเป็น Don't use a passkey เนื่องจาก Wiimote ไม่มี passkey ในการเชื่อมต่ออยู่แล้ว จากนั้นกด Next > เพื่อเข้าสู่ขั้นตอนถัดไป



รูปที่ ก-4 หน้าต่างกำหนด Passkey

5. คอมพิวเตอร์จะทำการติดต่อกับตัว Wiimote ในขั้นตอนนี้หากไฟจาก Wiimote หยุดกระพริบไปแล้วให้กดปุ่ม 1 และ 2 ของ Wiimote ค้างไว้อีกครั้งจนกว่าวีโมตจะได้รับการติดตั้งสำเร็จ



รูปที่ ก-5 หน้าต่างติดตั้งอุปกรณ์ Bluetooth

6. กดปุ่ม Finish เพื่อจบการเชื่อมต่อ Wiimote กับคอมพิวเตอร์

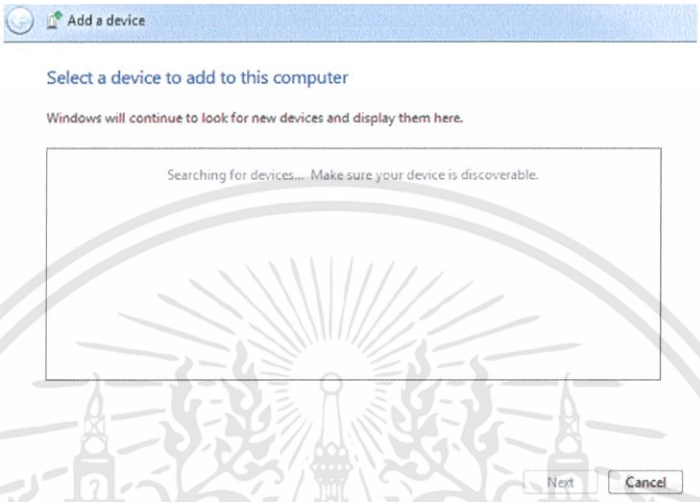
รูปที่ ก-6 หน้าต่างสุดท้ายของการAdd Bluetooth Device Wizard

7. เมื่อทำการAdd วีโมตสำเร็จแล้ว จะปรากฏอยู่ใน List ของอุปกรณ์ที่เชื่อมต่อแล้ว

รูปที่ ก-7 หน้าต่าง Bluetooth Devices เมื่อเชื่อมต่อ Wiimote เรียบร้อยแล้ว

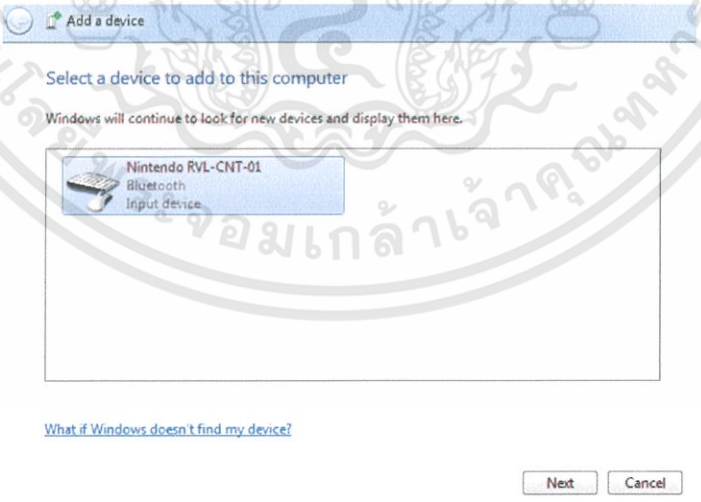
การติดตั้ง Wiimote ผ่าน Bluetooth สำหรับ Windows7

1. เข้าสู่ Control Panel แล้วเลือกหัวข้อ Hardware and Sound เลือกหัวข้อย่อย Add a Device หรือเข้าสู่หน้าต่างจุดการอุปกรณ์ Bluetooth โดยตรงด้วยการคลิกที่ไอคอน  จากนั้นเลือกหัวข้อ Add a Device จะได้หน้าต่างค้นหาอุปกรณ์ขึ้นมา



รูปที่ ก-8 หน้าต่างค้นหาอุปกรณ์ของ Windows7

2. กดปุ่ม 1 และ 2 ของ Wiimote ค้างเอาไว้เพื่อให้ Wiimote ถูกค้นพบด้วยตัวรับ Bluetooth หากสำเร็จจะปรากฏอุปกรณ์ในชื่อ Nintendo RVL-CNT-01 หากไม่พบให้ทำการค้นหาใหม่ หากพบแล้วให้กดปุ่ม  เพื่อเข้าสู่ขั้นตอนต่อไป

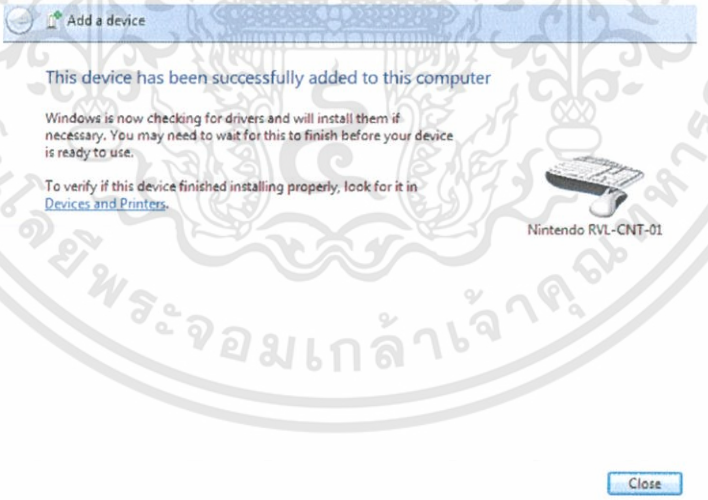


รูปที่ ก-9 หน้าต่างแสดงอุปกรณ์ที่ตรวจพบ

3. เข้าสู่หน้าต่าง Select Paring Action ให้เลือกหัวข้อ Pair without using a code เพื่อทำการเชื่อมต่อ Wiimote โดยไม่ต้องมีการกำหนดโค้ดในการสื่อสาร

รูปที่ ก-10 หน้าต่าง Select Paring Action ของ Windows 7

4. รอให้การเชื่อมต่อเสร็จสิ้น โดยระหว่างรอให้กดปุ่ม 1 และ 2 ของ Wiimote ค้างเอาไว้อีกครั้ง จนกว่าระบบจะทำการติดตั้ง Driver ที่รองรับให้เรียบร้อย จะแสดงหน้าต่างแจ้งว่าการเชื่อมต่อเสร็จสิ้นแล้ว กดปุ่ม Close เพื่อจบการติดตั้งอุปกรณ์

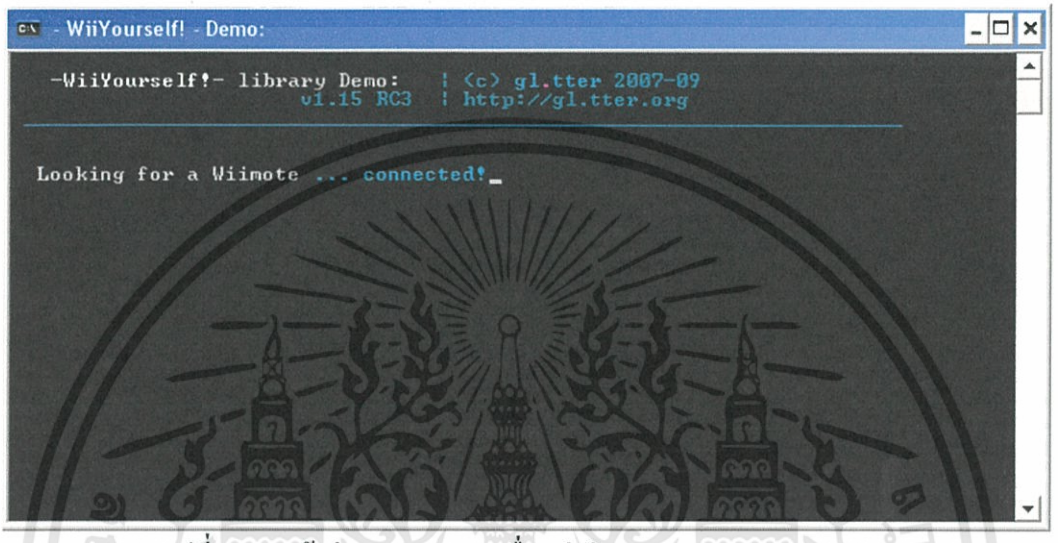


รูปที่ ก-11 หน้าต่างแสดงผลการเชื่อมต่อสำเร็จ

ทดสอบการทำงานของ Wiimote ผ่าน Demo ของ WiiYourself!

เมื่อเชื่อมต่อวีโมตกับคอมพิวเตอร์เรียบร้อยแล้ว จึงทำการทดสอบการใช้งานกับ Library ซึ่ง Wii Yourself! มีโปรแกรม Demo เพื่อทดสอบการทำงานกับวีโมตดังนี้

1. รันตัว Demo จะได้หน้าต่าง Command Prompt ขึ้นมา ซึ่งหากทำการเชื่อมต่อสำเร็จ โปรแกรมจะแจ้งว่าเชื่อมต่อได้แล้ว



รูปที่ ก-12 หน้าต่างแสดงผลการเชื่อมต่อใน Demo ของ WiiYourself!

2. เมื่อรอสักครู่ โปรแกรมจะแสดงข้อความให้ลองกดปุ่มต่างๆ เป็นการยืนยันวีโมตสามารถติดต่อกับคอมพิวเตอร์ได้แล้ว

3. สำหรับส่วนของการให้แสง IR อย่างง่าย สามารถทำได้โดยการใช้หลอด IR แบบคลื่น นำมาเสียบเข้ากับถ่านลิเธียมแบบก้อนกลมดังแสดงในรูปที่ ก-13 แล้ววางในตำแหน่งใต้จอ จากนั้นทดสอบกับ Demo หากหัวข้อส่วน IR แสดงตัวเลขตำแหน่ง แสดงว่ามีการตรวจพบแสง IR

รูปที่ ก-13 การสร้างแหล่งกำเนิด IR อย่างง่าย

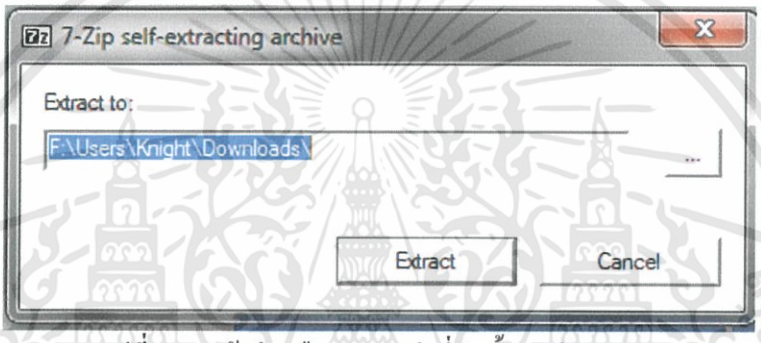
ภาคผนวก ข.
การติดตั้งเกมรีโอวาโน

ภาคผนวก ข.
การติดตั้งเกมรีโอวาโน

การติดตั้งเกมรีโอวาโน

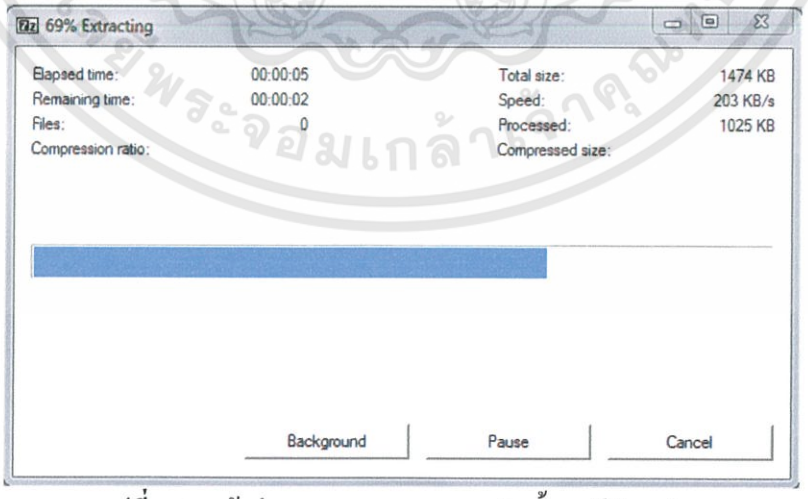
การติดตั้งเกมรีโอวาโน มีขั้นตอนดังต่อไปนี้

1. ดับเบิ้ลคลิกที่ไฟล์แพคเกจชื่อว่า Reovano.exe จากนั้นจะแสดงหน้าต่างเลือกตำแหน่งที่จะทำการติดตั้งตัวเกม โดยสามารถเปลี่ยนได้ด้วยการกดปุ่ม ... เพื่อเปลี่ยนตำแหน่งที่จะติดตั้ง โดยตัวเกมสามารถนำไปวางไว้ที่ไหนก็ได้ตามต้องการ จากนั้นกดปุ่ม Extract เพื่อเข้าสู่ขั้นตอนถัดไป



รูปที่ ข-1 หน้าต่างเลือกตำแหน่งที่ติดตั้งเกมรีโอวาโน

2. เข้าสู่หน้าจอแสดง Progress ของการติดตั้ง ให้รอจนหน้าต่างนี้หายไป จะได้โฟลเดอร์เกมชื่อ Reovano เข้าไปในโฟลเดอร์แล้วหาไฟล์ชื่อ ProjectSlime.exe แล้วทำการรันไฟล์ดังกล่าวจะสามารถเปิดตัวเกมได้ทันที หากเครื่องทำการติดตั้ง DirectX เอาไว้เรียบร้อยแล้ว



รูปที่ ข-2 หน้าต่างแสดง Progress การติดตั้งเกมรีโอวาโน

ภาคผนวก ค.
วิธีการเล่นเกมรีโอวาโน

ภาคผนวก ค.

วิธีการเล่นเกมรีโอวาโน

รายละเอียดของทรัพยากรที่ต้องการ

เกมรีโอวาโน (Reovano) เป็นเกมแนว Action-RPG ในมุมมอง 3 มิติที่มีความสามารถในการรองรับข้อมูลนำเข้าจาก Wiimote มีความต้องการทางทรัพยากรเครื่องขั้นต่ำต่อไปนี้

- ระบบปฏิบัติการ : Windows XP Service Pack 3 เป็นต้นไป หรือ Windows 7
- CPU : 2.20 GHz
- RAM : 2.00 GB
- Display Adapter Driver แบบ HD (ไม่สามารถรองรับแบบพื้นฐานได้เนื่องจากต้องการทรัพยากรในการสร้างฉาก 3 มิติ)

การควบคุมเกม

เกมสามารถบังคับได้ 2 ช่องทางได้แก่การใช้เมาส์และคีย์บอร์ด หรือใช้ Wiimote ร่วมกับ Nunchuck ในการควบคุมตามที่ผู้เล่นต้องการ โดยเกมจะมีการควบคุมพื้นฐานด้วยเมาส์และคีย์บอร์ด แต่หากก่อนเปิดโปรแกรมเกม ผู้เล่นทำการเชื่อมต่อ Wiimote เข้ากับคอมพิวเตอร์ไว้แล้ว ตัวเกมจะสามารถรองรับการควบคุมจาก Wiimote ได้ทันที โดยปุ่มของ Wiimote และ Nunchuck จะถูกใช้แทนปุ่มต่างๆ บนคีย์บอร์ดและอาศัยการตรวจจับแสง IR ของ Wiimote แทนตำแหน่งของ Pointer ที่ได้จากเมาส์ รายละเอียดดังตารางที่ ค-1 ซึ่งรายละเอียดการควบคุมดังกล่าวจะได้รับการอธิบายไว้ภายในเกม

ตารางที่ ค-1 แสดงปุ่มที่ใช้ควบคุมเกม

คำสั่ง	เมาส์/คีย์บอร์ด	Wiimote / Nunchuck
ขยับตัวละครไปข้างหน้า	คีย์บอร์ด: W	Nunchuck: ก้านควบคุม
ขยับตัวละครไปทางซ้าย	คีย์บอร์ด: A	
ขยับตัวละครไปทางขวา	คีย์บอร์ด: S	
ขยับตัวละครไปข้างหลัง	คีย์บอร์ด: D	
หมุนมุมกล้องตามเข็มนาฬิกา	เมาส์:คลิกขวาและDragไปทางซ้าย	Wiimote: -
หมุนมุมกล้องทวนเข็มนาฬิกา	เมาส์:คลิกขวาและDragไปทางขวา	Wiimote: +
เลื่อนกล้องเข้าหาตัวละคร	เมาส์:Scroll Up	Wiimote: 1
เลื่อนกล้องออกห่างตัวละคร	เมาส์:Scroll Down	Wiimote: 2

คำสั่ง	เมาส์/คีย์บอร์ด	Wiimote / Nunchuck
ตอบสนองบทพูด	คีย์บอร์ด: Spacebar	Wiimote: A
ตอบสนองกับวัตถุในเกม	คีย์บอร์ด: E	Wiimote: B
หยุดเกม/เข้าเมนู	คีย์บอร์ด: Tab	Wiimote: Home
ล็อคเป้าหมาย(เมื่อมีศัตรู)	คีย์บอร์ด: Spacebar	Nunchuck: Z
ปลดล็อคเป้าหมาย(เมื่อมีศัตรู)	คีย์บอร์ด: Shift	Nunchuck: C
โจมตีปกติ	เมาส์: คลิกซ้าย	Wiimote: A
ใช้ทักษะที่ตั้งไว้ที่ช่อง 1	คีย์บอร์ด: 1	-
ใช้ทักษะที่ตั้งไว้ที่ช่อง 2	คีย์บอร์ด: 2	-
ใช้ทักษะที่ตั้งไว้ที่ช่อง 3	คีย์บอร์ด: 3	-
ใช้ทักษะที่ตั้งไว้ที่ช่อง 4	คีย์บอร์ด: 4	-

นอกเหนือจากคำสั่งควบคุมพื้นฐานดังกล่าว อาวุธแต่ละประเภทจะมีลักษณะการออกท่าทักษะพิเศษแตกต่างกันไปหลังกดใช้ดังตารางที่ ค-2 โดยผู้เล่นต้องออกท่าให้ทันภายในระยะเวลาที่กำหนด ซึ่งแต่ละท่านั้นออกแบบให้ใช้กับ Wiimote แต่ยังสามารถใช้เมาส์ได้เช่นกัน

ตารางที่ ค-2 แสดงวิธีการใช้ทักษะพิเศษ

ชื่อท่า	อาวุธ	วิธีใช้
Bash	มือเดียวและโล่ (One Hand & Shield)	กดโจมตีปกติที่ขอบจอด้านขวา แล้วลากไปปล่อยทางด้านซ้าย ศัตรูที่อยู่ในระยะโจมตีจะได้รับความเสียหาย
Ram	มือเดียวและโล่ (One Hand & Shield)	กดโจมตีปกติที่ขอบจอด้านล่าง แล้วลากเข้าไปส่วนบนของเพื่อใช้ท่า
Swing	อาวุธหนัก (Heavy Weapon)	กดโจมตีปกติที่ขอบจอด้านซ้าย ลากผ่านตัวศัตรูและปล่อยทางด้านขวา ศัตรูที่ถูก Pointer เลื่อนผ่านจะได้รับความเสียหาย
Smash	อาวุธหนัก (Heavy Weapon)	กดโจมตีปกติจากขอบจอด้านบน แล้วลากลงมาปล่อยมุมล่างของจอ ศัตรูที่อยู่รอบตัวผู้เล่นในระยะจะได้รับความเสียหาย
Snipe	อาวุธทางไกล (Range Weapon)	เมื่อกดใช้ Pointer จะเปลี่ยนเป็นรูปเป้า เลื่อนไปกดโจมตีปกติที่ศัตรูตัวที่ต้องการเพื่อยิงสร้างความเสียหาย

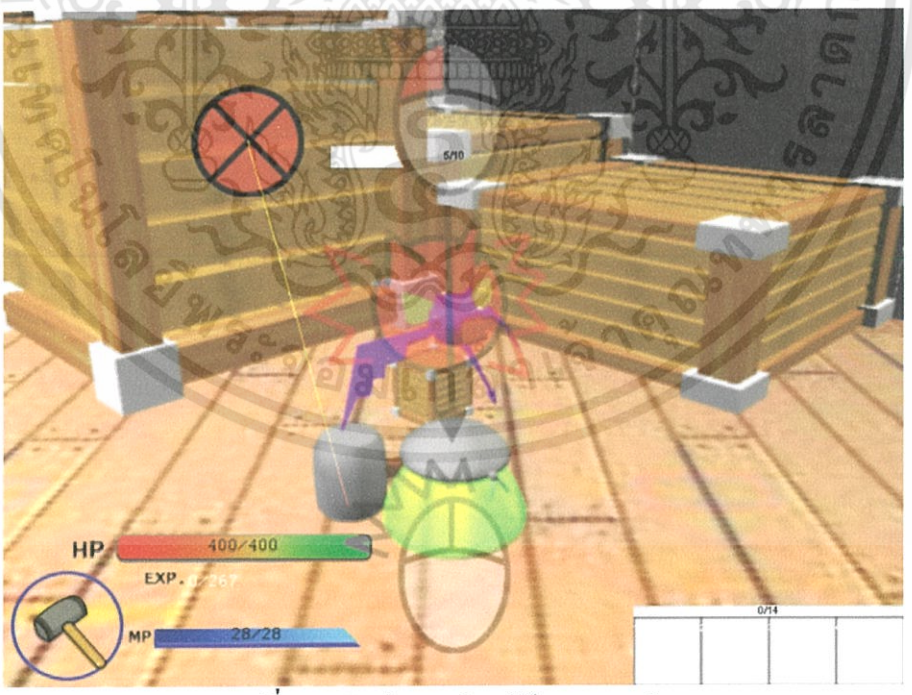
ชื่อท่า	อาวุธ	วิธีใช้
Firework	อาวุธทางไกล (Range Weapon)	กดโจมตีปกติที่ส่วนต่างๆ ของจอ 4 จุดที่กำหนดตามลำดับ เมื่อกดครบแล้วจึงสร้างความเสียหายเป็นวงกว้าง

วิธีการเล่นเบื้องต้น

เนื่องจากเกมเป็นรูปแบบ Action-RPG วิธีการเล่นจึงค่อนข้างเป็นไปได้หลากหลาย โดยมีเนื้อเรื่องรองรับเพื่อให้ผู้เล่นได้สวมบทบาทและเดินเรื่องตามที่กำหนดเอาไว้ให้บรรลุจนถึงตอนจบของเรื่อง

เงื่อนไขการเล่นเบื้องต้น สามารถจำแนกได้หลักๆ ดังต่อไปนี้

- อาวุธแต่ละชนิดจะมีวิธีการโจมตีปกติต่างกันได้แก่ อาวุธมือเดียว ให้กดโจมตีแล้วลากสะบัดจากด้านขวา ไปทางด้านซ้ายแล้วปล่อยปุ่ม อาวุธหนัก ให้กดโจมตีโดยลากสะบัดจากด้านบนลงด้านล่างจอแล้วปล่อยปุ่ม และอาวุธระยะไกล กดโจมตีค้างระยะหนึ่งจนวงกลมหายไปจะเป็นการยิงออกมา ตัวอย่างเช่น อาวุธหนักดังรูปที่ ค-1 เมื่อคลิกจะเกิดวงกลมสีแดงมีกากบาทขึ้น แล้วให้ลากลงมาด้านล่าง เมื่อปล่อยเมาส์จะกลายเป็นการโจมตีปกติ
- ศัตรูภายในเกมจะสร้างความเสียหายให้กับเรา หากผู้เล่นถูกโจมตี ค่า HP จะลดลงตามความเสียหายที่ได้รับ หากลดลงจนเหลือ 0 หรือตาย ผู้เล่นจะถูกหักเงินครึ่งหนึ่งจากที่มีอยู่ และถูกส่งกลับไปยังบ้าน แต่ในบางกรณีเช่นเนื้อเรื่องที่จะต้องสู้กับหัวหน้า (Boss) ของศัตรู ตัวเกมจะให้ผู้เล่นย้อนกลับไปในจุดที่กำหนดไว้แล้วเล่นใหม่จากจุดนั้น เพื่อให้เนื้อเรื่องปะติดปะต่อกันได้อย่างเหมาะสม

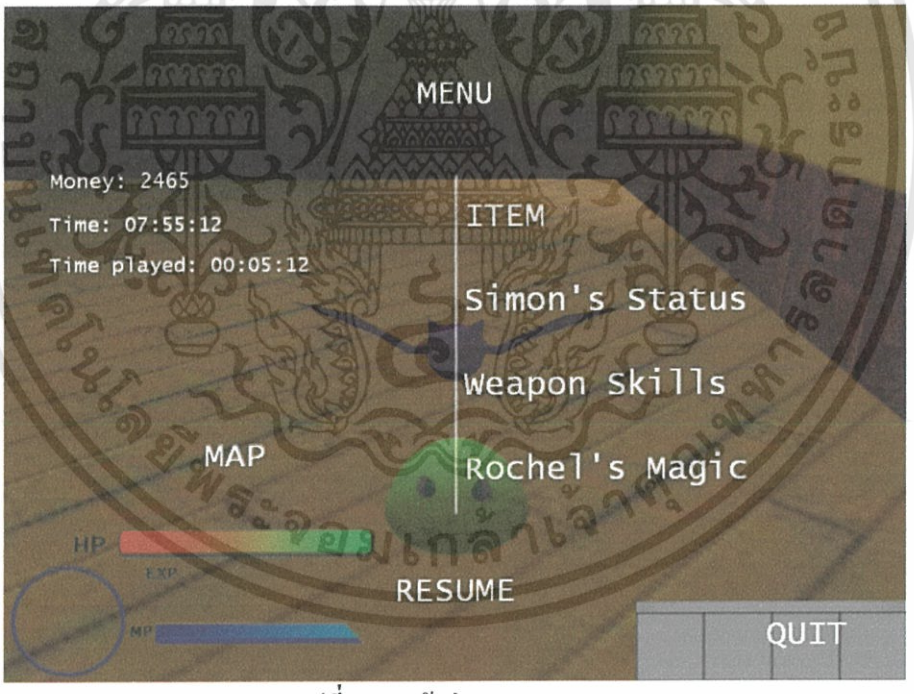


รูปที่ ค-1 ตัวอย่างการโจมตีด้วยอาวุธหนัก

หน้าต่างหยุดเกม

หน้าต่างหยุดเกมสามารถเข้าได้ผ่านคำสั่งควบคุมให้หยุดเกม/เข้าเมนูได้ทุกเมื่อ หากผู้เล่นไม่อยู่ในเหตุการณ์หรือบทสนทนาใด ๆ อยู่ โดยหน้าต่างนี้จะแสดงข้อมูลใช้เป็นส่วนหลางในการเข้าสู้หน้าต่างจัดการอื่น ๆ ได้ โดย

- Money แสดงจำนวนเงินปัจจุบันที่ผู้เล่นมีอยู่
- Current Time แสดงเวลาปัจจุบันภายในโลกเกม
- Playing Time แสดงเวลารวมที่ผู้เล่นเล่นเกมนี้
- Map กดเพื่อเข้าสู่หน้าจอแสดงแผนที่ปัจจุบัน
- Item กดเพื่อเข้าสู่หน้าต่างจัดการไอเท็ม
- Simon's Status กดเพื่อเข้าสู่หน้าต่างจัดการสถานะของตัวละคร
- Weapon Skill กดเพื่อเข้าสู่หน้าต่างจัดการทักษะอาวุธ สำหรับดูรายละเอียดของทักษะอาวุธปัจจุบันและเลือกเรียนรู้ทักษะที่มี
- Rochel's Magic กดเพื่อเข้าสู่หน้าต่างจัดการเวทมนตร์ของคู่หู ดูรายละเอียดของเวทมนตร์ที่คู่หูมีอยู่และปรับโอกาสการร่ายเวทมนตร์แต่ละชนิดของคู่หู



รูปที่ ค-2 หน้าต่างเมนูหยุดเกม

หน้าต่างจัดการไอเท็ม

หน้าจอสำหรับจัดการไอเท็มโดยด้านบนของช่องไอเท็มจะระบุจำนวนเงินที่ผู้เล่นมี และส่วนช่องไอเท็ม โดยไอเท็มแต่ละชิ้นจะมีตัวเลขกำกับระบุจำนวนที่ไอเท็มชิ้นนั้นๆ มีอยู่ โดยสามารถใช้ ทิ้ง ให้คู่หูเอาไปเก็บ หรือนำมาสวมใส่ให้กับผู้เล่นก็ได้ตามแต่ชนิด สามารถสับตำแหน่งได้ตามต้องการด้วยการเลือกและลากไปวางไว้ที่ช่องอื่น หากเป็นไอเท็มสำหรับสวมใส่จะสามารถติดให้กับตัวเองได้ โดยการเลือกและสลับไปวางไว้ที่ช่อง 3 ช่องที่แบ่งไว้เป็น มือซ้าย มือขวา และหัว โดยไอเท็มจะสามารถจำแนกตามชนิดได้ทั้งหมด 11 ประเภทมีเงื่อนไขแยกย่อยในการสวมใส่และการนำไปใช้ดังต่อไปนี้

1. อาวุธมือเดียว (Onehand Weapon) สำหรับสวมใส่ที่มือขวา และเหลือมือซ้ายให้สามารถถือโล่ได้

2.โล่ (Shield) สำหรับสวมใส่ที่มือซ้าย หากไม่ได้สวมใส่อาวุธหนักหรือปืนไว้ที่มือขวา

3.อาวุธหนัก (Heavy Weapon) สำหรับสวมใส่ที่มือขวาได้โดยที่มือซ้ายต้องว่างอยู่

4.อาวุธปืน (Range Weapon) สำหรับสวมใส่ที่มือขวา ต้องการกระสุนปืนถือที่มือซ้าย เพื่อใช้งาน

5.กระสุนปืน (Bullets) สำหรับสวมใส่ที่มือซ้าย ทำให้ปืนสามารถใช้งานได้

6.คันธนู (Bow) สำหรับสวมใส่ที่มือซ้าย ต้องการลูกธนูถือที่มือซ้ายเพื่อใช้งาน แต่จะไม่ได้รับผลจากทักษะประเภทอาวุธระยะไกล

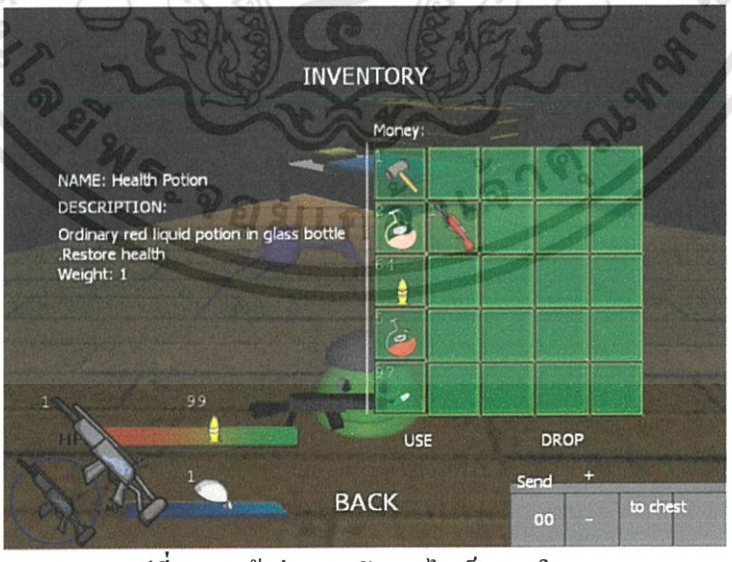
7.ลูกธนู (Arrow) สำหรับสวมใส่ที่มือซ้าย ทำให้ธนูสามารถใช้งานได้

8.หมวก (Headgear) สำหรับสวมใส่ที่หัว เพิ่มสถานะให้ผู้เล่น

9.ของใช้ (Useable) สามารถกดใช้ในหน้าเมนูไอเท็มได้ หรือนำไปถือที่มือขวาแล้วโยนใส่ศัตรู จะให้ผลแตกต่างกันไป

10.จิปาถะ (Misc.) สำหรับขายหรือโยนด้วยการนำมาสวมใส่ที่มือขวา

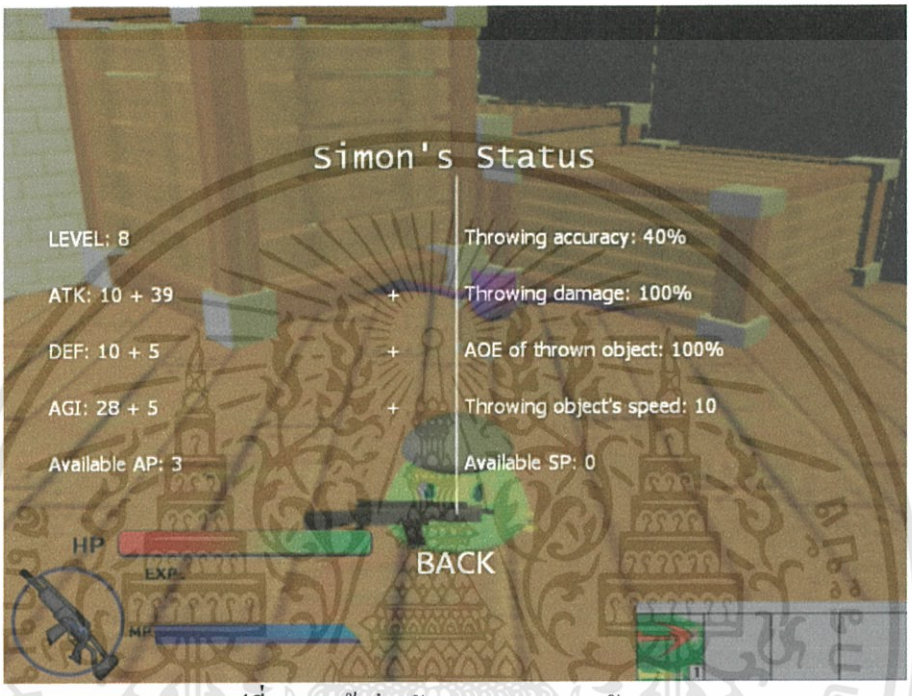
11.ปริศนา (Unknown) สำหรับขาย หรือนำไปแลกที่ร้านแลกของ



รูปที่ ค-3 หน้าต่างเมนูจัดการไอเท็มภายในเกม

หน้าต่างจัดการสถานะตัวละคร

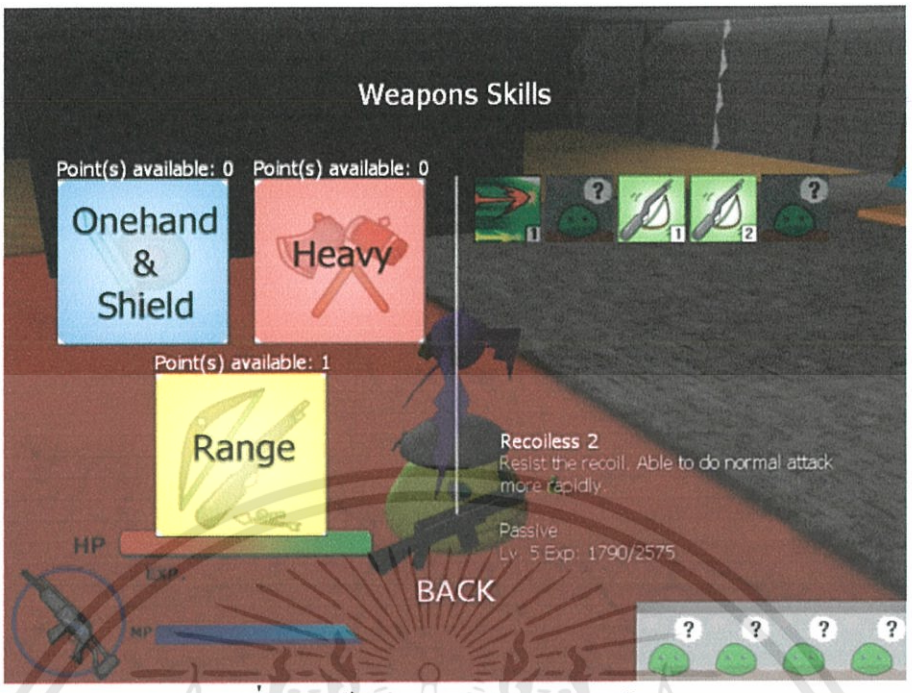
เมนูสำหรับจัดการสถานะของตัวเอก แสดงค่าพลังต่างๆ และเมื่อตัวเอกสะสม EXP ถึงระดับหนึ่งแล้ว จะได้แต้ม AP มา 3 แต้ม ให้ผู้เล่นสามารถเลือกเพิ่มค่าเหล่านี้ลงไปได้อย่างอิสระ โดย ATK ส่งผลให้การโจมตีมีความรุนแรงมากขึ้น DEF ส่งผลให้ได้รับความเสียหายจากการโจมตีต่าง ๆ ลง และ AGI เพิ่มค่า SP สำหรับใช้ทักษะ และลดคูลดาวน์ของทักษะ ทำให้ใช้ทักษะได้บ่อยขึ้น



รูปที่ ค-4 หน้าต่างจัดการสถานะของตัวละคร

หน้าต่างจัดการทักษะอาวุธ

เมนูสำหรับจัดการทักษะที่ตัวเอกมีหรือได้รับมาจากการใช้อาวุธทั้ง 3 ประเภทแยกกันเป็น 3 ชุด โดยเลือกจากไอคอนด้านซ้ายมือ และด้านขวามือจะแสดงทักษะของอาวุธชนิดนั้นๆ ผู้เล่นสามารถจัดการให้ตัวเอกเรียนรู้ทักษะได้ เมื่อได้รับแต้มทักษะมา รวมไปถึงจัดการวางปุ่มลัดโดยการเลือกและลากไปวางไว้ที่ช่องทักษะลัดทั้ง 4 ช่องด้านขวาล่าง เพื่อใช้ทักษะระหว่างเล่น ในกรณีที่เป็นทักษะประเภท Active



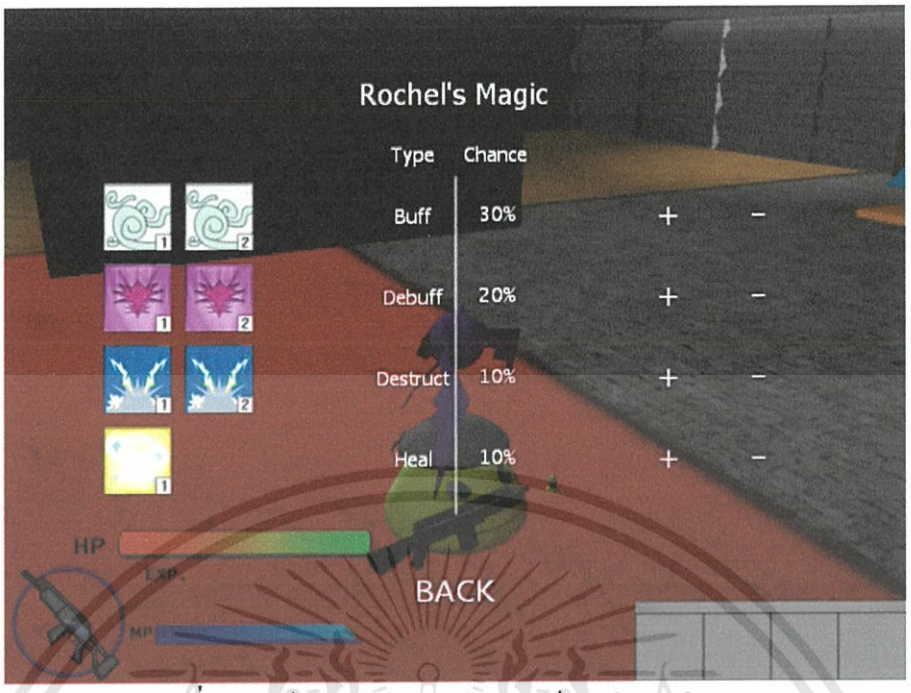
รูปที่ ค-5 หน้าต่างเมนูจัดการทักษะภายในเกม

ทักษะของอาวุธแต่ละชนิดนั้น จะแบ่งได้เป็น 5 แถว โดยแถวที่ 1 – 3 ผู้เล่นจะสามารถเลือกเรียนรู้ได้โดยแบ่งเป็นทักษะประเภท Active 2 ระดับ และ Passive 3 ระดับเมื่อ

- ตัวละครมี Level 10 จะสามารถเรียนรู้ทักษะในแถวที่ 2 ได้หากมีทักษะในแถวที่ 1 ถูกเรียนรู้ไปแล้ว 3 แต้ม
- ตัวละครมี Level 15 จะสามารถเรียนรู้ทักษะในแถวที่ 3 ได้หากมีทักษะในแถวที่ 2 ถูกเรียนรู้ไปแล้ว 3 แต้ม

หน้าต่างจัดการเวทมนตร์ของคู่หู

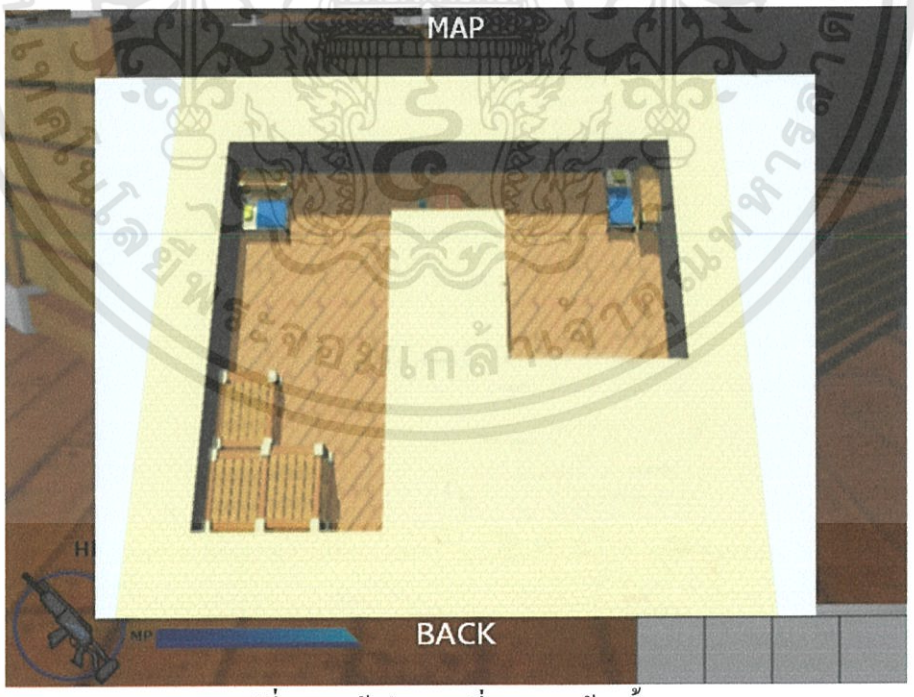
เมนูสำหรับจัดการแผนการช่วยเหลือของคู่หู โดยเมนูนี้จะแสดงเวทมนตร์ที่คู่หูสามารถใช้ได้ในปัจจุบัน ซึ่งค้างคาวจะสามารถใช้เวทมนตร์บทใหม่ได้เมื่อมีระดับสูงขึ้นถึงจุดหนึ่ง โดยในหน้าต่างนี้ผู้เล่นสามารถปรับโอกาสที่จะให้ค้างคาวร่ายเวทย์ในแต่ละสายจาก 4 สายได้ตามต้องการ



รูปที่ ค-6 หน้าต่างเมนูจัดการเวทมนตร์ของคู่หูภายในเกม

หน้าต่างแผนที่

หน้าต่างแผนที่จะแสดงลักษณะฉากของฉากนั้นๆ ให้ผู้เล่นใช้เป็นตัวช่วยนำทาง โดยสามารถเข้ามาดูได้ตามต้องการตลอดเวลา



รูปที่ ค-7 หน้าต่างแผนที่ของฉากบ้านชั้น 2

ภาคผนวก ง.

บทสรุปเกมรีโอวาโน

ภาคผนวก ง.

บทสรุปเกม Reovano

บทสรุปเนื้อเรื่องเกม

ตอนที่ 1

- เริ่มต้นเป็นบทสนทนาเกี่ยวกับเหตุการณ์ที่เกิดขึ้นอยู่บนชายหาด เมื่อคุยจบแล้วให้หันมุมกล้องไปทางซ้ายเล็กน้อยจะพบก้อนหิน ให้ทำการ Interact กับก้อนหิน เพื่อดำเนินเรื่องต่อ
- สะสมก้อนกรวด (Pebble) จำนวนหนึ่งจากก้อนหิน แล้วทำการติดตั้งที่มือขวาของผู้เล่น จากนั้นให้หันมุมกล้องแล้วกด Spacebar เพื่อหากล่องที่มีไอเท็มซ่อนอยู่ จากนั้นทำลายกล่องตามลำดับจนครบ 5 กล่อง
- เมื่อทำลายกล่องครบทั้ง 5 กล่องแล้ว จะเกิดเหตุการณ์กิ้งก่ายักษ์มาปรากฏตัวที่ชายหาด ให้เข้าไปโจมตี เอาชนะให้ได้ แล้วกิ้งก่ายักษ์จะหนีเข้าป่าไป จากนั้นออกจากฉากชายหาดเข้าไปในป่า เพื่อเข้าสู่เนื้อเรื่องตอนที่ 2

ตอนที่ 2

- เข้าสู่ป่าส่วนแรก ในส่วนนี้จะมีศัตรูเกิดประปราย สามารถสู้กับศัตรูเพื่อเก็บประสบการณ์ได้
- เมื่อจะเข้าป่าในส่วนที่สอง จะเกิดเหตุการณ์ขึ้น ตัวเอกจะได้ยินเสียงร้องจากป่าส่วนถัดไป และตัดฉากเข้าสู่ป่าส่วนที่ 2 (Deep Forest)
- เมื่อมาถึงป่าส่วนที่ 2 เราจะพบกิ้งก่านอนเจ็บขาอยู่ เมื่อจบบทสนทนาจะพบว่ามีก็อบลิน (Goblin) ปรากฏตัวออกมา สู้ให้ชนะ หากแพ้ตัวเกมจะให้โหลดเซฟที่เซฟเอาไว้ก่อนที่จะเข้าป่าส่วนที่ 2 ให้
- เมื่อชนะก็อบลินแล้ว ให้ย้อนไปคุยกับกิ้งก่ายักษ์ จากนั้นมันจะพาเราไปจนถึงหน้าเมืองรีโอวาโน แต่ด้วยความเหนื่อยล้า พวกเราจึงสลบไปก่อน จากนั้นเข้าสู่เนื้อเรื่องบทที่ 3

ตอนที่ 3

- ตัวเอกและคู่หูจะตื่นขึ้นมาในห้องนอนของบ้านพักภายในเมือง โดยมีวิคกี้ (Vicky) คอยเราอยู่ข้างเตียง จะมีบทสนทนาสักครู่หนึ่ง โดยเธอแนะนำตัว และบอกว่าสามารถพบเธอได้ที่ร้านอาหาร (Bar) ของเมือง
- ออกจากห้องนอนเข้าสู่ชั้น 1 จะมีชั้นหนังสือ ผู้เล่นสามารถอ่านเพื่อทำความรู้จักและทบทวนระบบเกมได้ จากนั้นออกจากบ้านเข้าสู่ตัวเมือง จุดนี้ผู้เล่นสามารถย้อนกลับไปที่ป่า หรือทำภารกิจย่อยได้ตามอัธยาศัย

- เมื่อถึงเวลา 18:00 น. จะสามารถเข้าร้านอาหารได้ เมื่อเข้าไปตัวเอกและคู่หูจะได้คุยกับไคลน์เจ้าของร้านและวิคกี้ จากนั้นจะเกิดเหตุการณ์ก็อบลินต่างถิ่นบุกเมาทางหลังครัว แล้วหนีไปได้
- หลังจากเหตุการณ์ในร้านอาหารแล้ว โรงงานร้างทางทิศตะวันออกของเมืองจะสามารถเข้าได้ เพื่อเริ่มเนื้อเรื่องตอนที่ 4 ต่อไป

ตอนที่ 4

- เมื่อเข้ามาในโรงงาน ให้ทำการสำรวจ และเข้าไปเรื่อยๆ จนถึงห้องบัญชี (Accounting) จะเป็นห้องที่มีกลไฟสวิทช์อยู่ ให้หาสวิทช์ทั้ง 4 ตัว ให้พบ หากย้อนกลับมายังฉากห้องห้องสายพานผลิต (Manufacture Room) ก่อนจะกดสวิทช์ได้ครบ สวิทช์ทั้งหมดจะถูกรีเซตกลับ ต้องกดใหม่
- เมื่อกดสวิทช์ทั้ง 4 ปุ่มเรียบร้อยแล้ว ให้เข้าสู่ฉากห้องออฟฟิศ (Office) ซึ่งห้องนี้จะมีประตูล็อคอยู่ หากกดสวิทช์ทั้ง 4 จากห้องบัญชีเรียบร้อยแล้ว ประตูจะเปิดได้ แล้วเกิดเหตุการณ์ขึ้น
- หลังเปิดประตูเข้าไป จะเกิดเหตุการณ์ขึ้น แล้วประตูจะถูกล็อค ทำให้เราไม่สามารถออกไปได้ ให้กำจัดก็อบลิน 2 ตัวที่อยู่ในห้องให้ได้ แล้งประตูจะเปิดออก ทำให้สามารถเข้าไปในโกดัง (Warehouse) ได้
- ที่ห้องโกดัง เราจะพบก็อบลินแครงค์ (Crank) กับลูกน้องพยายามเปิดประตูหนีออกจากโรงงานแต่ไม่สำเร็จ แล้วพวกมันจะมาเจอตัวเราเข้า และเกิดการต่อสู้ขึ้น เอาชนะให้ได้แล้วกิ้งก่ายักษ์จะปรากฏตัวเข้ามาที่ประตูอีกทาง จากนั้นแครงค์จะถูกจับได้ และสิ้นสุดเนื้อเรื่องในตอนที่ 4

ตอนที่ 5

- หลังเหตุการณ์ในตอนที่ 4 จบลง ตัวเอกก็จะกลับบ้าน และตื่นขึ้นมาในวันใหม่ เมื่อคุยกับคู่หูสักพักก็เกิดการตัดสินใจว่าจะสืบหาเบาะแส ถามไถ่เกี่ยวกับเรื่องของมนุษย์จากลุงไคลน์เจ้าของร้านอาหาร
- ไปที่ร้านอาหาร แล้วุยกับลุงไคลน์ จะได้เบาะแสเกี่ยวกับสุสานว่าจะมีอะไรบางอย่างเกิดขึ้นเมื่อถึงเวลาเที่ยงคืน
- จากนั้น ให้ไปที่สุสานในเวลาก่อนเที่ยงคืน รอหรือทำการสำรวจสุสานรอ เมื่อถึงเวลาเที่ยงคืนตรงจะเกิดเหตุการณ์ขึ้น ทำให้ตัวเอกถูกส่งไปยังชายหาด
-

ตอนที่ 6

- ที่ชายหาด ตัวเอกจะได้พบกับคู่หูตัวปลอม และเกิดการต่อสู้กัน เอาชนะให้ได้จะสามารถกลับไปยังสุสานได้

ตอนที่ 7

- เมื่อตัวเอกตื่นขึ้นจากเหตุการณ์ที่ชายหาด ก็พบว่าคู่หูกำลังพยายามปลุกเราอยู่ และปรากฏรูปปั้นปริศนาที่กลางสุสานอยู่ด้วย คู่หูจึงแนะนำเราให้กลับบ้านก่อน เพื่อเตรียมการสำรวจคฤหาสน์ใต้ดินที่พบซ่อนอยู่ใต้รูปปั้นนั้น
- วันรุ่งขึ้น ให้ไปที่สุสานเมื่อเตรียมการณ์พร้อมแล้ว สำรวจรูปปั้นก็จะพบทางลับลงไปยังคฤหาสน์ใต้ดิน
- ที่ชั้นแรก ให้ทำการสำรวจ จะพบทางลับลงไปชั้นใต้ดินอีกชั้นหนึ่ง อยู่ที่ส่วนหลังสุดของคฤหาสน์ เข้าสู่ชั้นใต้ดินที่ 2
- คฤหาสน์ใต้ดินชั้นที่ 2 นี้จะมีกลไกเขาวงกตบางอย่างอยู่ เมื่อแก้ได้แล้วจะพบทางเข้าสู่ห้องพิธีต่อไป
- ที่ห้องพิธี เราจะพบกับบอสใหญ่ สไลม์ยักษ์รอเราอยู่ ต่อสู้ให้ชนะ จากนั้นมันจะสลายตัวไป และทิ้งหนังสือเอาไว้ จากนั้นจึงเข้าสู่ฉากจบเกม

ข้อมูลของฉากภายในเกม

1. ชายหาด (Beach)

ฉากแรกของเกม ประกอบไปด้วยเรือที่อับปางอยู่ ที่นี่จะมีหินที่ถูกน้ำกัดเซาะ สามารถเก็บก้อนกรวด(Pebble) ได้ และกล่องอุปกรณ์ต่างๆ ที่กระจัดกระจายทั่วชายหาด ไม่มีศัตรูทั่วไปเกิดที่นี่

2. ป่า (Forest)

ป่าส่วนแรก เข้าจากทางฝั่งชายหาด ลักษณะเป็นทางค่อนข้างกว้าง จะมีศัตรูเกิดประปรายในช่วงเริ่มต้น โดยมีศัตรูทั่วไปแบ่งได้เป็น

ตอนกลางวัน: Machi x 3 , BindPlant x 2

ตอนกลางคืน: RedBall x 3 , BindPlant x 2

3. ป่าส่วนลึก (Deep Forest)

ป่าส่วนกลาง เข้าจากทางป่าส่วนแรกหรือป่าส่วนลึกเข้าไป ลักษณะเป็นทางแคบและมีลานตรงกลาง โดยมีศัตรูทั่วไปแบ่งได้เป็น

ตอนกลางวัน: Machi x 1 , BindPlant x 2 ,RedBall x 1

ตอนกลางคืน: Machi x 1 , BindPlant x 2 ,RedBall x 1

4. ป่าส่วนลึกสุด (Deeper Forest)

ป่าส่วนลึก เข้าจากทางป่าส่วนกลางหรือจากประตูเมือง ลักษณะเป็นป่าเปิด พื้นที่ดูกว้างโดยมีศัตรูทั่วไปแบ่งได้เป็น

ตอนกลางวัน: Machi x 2 , BindPlant x 2 ,RedBall x 1

ตอนกลางคืน: Machi x 2 , BindPlant x 1 ,RedBall x 2

5. ประตูหน้าเมือง (Main Gate)

ประตูหน้าเมืองเป็นส่วนที่เข้ามาจากทางป่าส่วนลึกสุด เป็นฉากขั้นระหว่างป่ากับตัวเมือง มีกิ้งก่าคอยเฝ้าอยู่ และมีการเพาะปลูกที่ส่วนนี้ ผู้เล่นอาจได้รับข้อมูลบางอย่างจากจุดที่นี่ได้

6. เมืองรีโอวาโน (Reovano City)

ตัวเมืองถือเป็นศูนย์กลางในการติดต่อไปยังสถานที่ต่างๆ ซึ่งเมืองจะประกอบไปด้วยบ้าน ร้านอาหาร ตลาด โรงงานร้าง สุสาน และแท่นนาฬิกาแดด มี NPC เดินประปราย ให้เราได้พูดคุยด้วยเล็กน้อย

7. บ้านชั้นที่ 1 (House Lobby)

ห้องรับแขกของบ้าน มีชั้นวางหนังสือที่บรรจุข้อมูลการเล่นเบื้องต้นเอาไว้ และหีบเก็บของสำหรับเก็บไอเท็มที่ยังไม่ได้ใช้

8. บ้านชั้นที่ 2 (House Bedroom)

ห้องนอน แบ่งเป็นสองเตียงนอน โดยเตียงนอนของผู้เล่นจะสามารถกดเพื่อพักผ่อนได้ และมีจุดเซฟอยู่ที่ข้างเตียงนอน

9. ร้านอาหาร (Bar)

มีไคลน์เป็นเจ้าของร้าน และวิคกี้เป็นพนักงานอยู่ที่นี่ โดยจะเปิดตั้งแต่ 18:00 – 24:00 นาฬิกา เหตุการณ์พิเศษในช่วงตอนที่ 3 จะเกิดขึ้นที่นี่

10. ตลาด (Market)

ศูนย์รวมร้านค้า ซึ่งแบ่งได้เป็นห้องๆ ประกอบไปด้วย ร้านแลกของ ร้านดาบและโล่ ร้านอาวุธหนัก ร้านอาวุธทางไกล ร้านยา ร้านจิปาถะ และร้านปริศนา

11. สุสาน (Graveyard)

เป็นส่วนติดต่อกับคฤหาสน์ใต้ดิน โดยช่วงเริ่มเกมจะยังไม่เห็นรูปปั้นปริศนาที่ตั้งอยู่ใจกลางสุสาน หากมาที่นี่ตอนกลางคืนนานๆ จะเจอ Ghochu มาแอบโจมตีผู้เล่น

12. โรงงานร้าง: ห้องรับแขก (Reception)

ส่วนแรกของโรงงานร้าง มีประตูเข้าไปยังโกดังแต่ล้อคอยู่ในตอนแรก และอีกฟากเป็นประตูเข้าห้องพักสินค้า มีศัตรูทั่วไปแบ่งได้เป็น

ตอนกลางวัน: Machin x 3, Chu x 2 , Electriball x 1

ตอนกลางคืน: Machin x 2, Chu x 2 , Vermiball x 1, Electriball x 1

13. โรงงานร้าง: ห้องพักสินค้า (Cargo)

ส่วนที่สองของโรงงานร้าง เข้าได้จากทางห้องรับแขก และเชื่อมไปยังห้องสานพานผลิต เป็นห้องที่ค่อนข้างโล่ง มีแท่นโต๊ะปริศนาตั้งอยู่ มีศัตรูทั่วไปแบ่งได้เป็น

ตอนกลางวัน: Machin x 1, Chu x 4, Electriball x 1

ตอนกลางคืน: Machin x 1, Chu x 3, Metchu x1, Vermiball x 1

14. โรงงานร้าง: ห้องสายพานผลิต (Manufacture Room)

ส่วนลึกของโรงงานร้าง ไม่มีศัตรูเกิดที่นี่เพื่อให้ผู้เล่นได้พัก แต่จะมีจุดเซฟ และไอเท็มบางอย่างให้เก็บ โดยเป็นส่วนติดต่อระหว่างห้องพักสินค้ากับห้องบัญชี

15. โรงงานร้าง: ห้องบัญชี (Accounting)

ห้องบัญชีถูกพวกก็อบลินจัดวางของจนกลายเป็นทางวงกตโดยไม่มีใครรู้ และมีสวิตช์กลไกลับซ่อนอยู่ตามจุดต่างๆ ให้ผู้เล่นค้นหา มีศัตรูทั่วไปแบ่งได้เป็น

ตอนกลางวัน: Machin x 3, Metchu x 1, Electriball x 2

ตอนกลางคืน: Metchu x 1, Electriball x 2, Vermiball x 3

16. โรงงานร้าง: ห้องออฟฟิศ (Office)

ห้องออฟฟิศเป็นห้องที่พวกก็อบลินใช้สุมหัวกัน มีประตูกลไกล็อคอยู่ ต้องหาทางกดสวิทช์ที่ห้องบัญชีเพื่อให้ประตูเปิดออก โดยจะเป็นห้องที่มีลักษณะเป็นลานกว้าง มีศัตรูชุกชุมทั่วไปแบ่งได้เป็น

ตอนกลางวัน: Machin x 4, Chu x 1, Metchu x 3, Electriball x 2

ตอนกลางคืน: Machin x 4, Chu x 1, Metchu x 3, Vermiball x 2

17. โรงงานร้าง: โกดัง (Warehouse)

โกดังเป็นห้องสำหรับต่อสู้กับก็อบลินแครงค์ ไม่มีศัตรูเกิดที่นี่

18. คฤหาสน์ใต้ดิน 1 (Mansion B1F)

คฤหาสน์ใต้ดินชั้นแรก จะมีส่วนตัวบ้าน และมีทางลับเข้าสู้ชั้นใต้ดินที่สองเป็นทางลับอยู่ด้านหลังบ้าน ศัตรูทั่วไปแบ่งได้เป็น

ตอนกลางวัน: Machine x 2, DarkBall x 2, Wisp x 2

ตอนกลางคืน: Machine x 3, DarkBall x 1, Wisp x 2

19. คฤหาสน์ใต้ดิน 2 (Mansion B2F)

คฤหาสน์ใต้ดินชั้นสอง มีลักษณะเป็นห้องแคบๆ มีศัตรูทั่วไปชุกชุม และเป็นฉากที่ยากที่สุดก่อนถึงตัวหัวหน้าใหญ่ แบ่งได้เป็น

ตอนกลางวันและกลางคืน: Machine x 1, Vermiball x 1, DarkBall x 2, Wisp x 2

20. คฤหาสน์ใต้ดิน: ห้องพิธี (Stateroom)

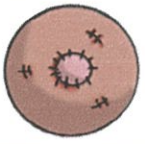
ห้องสุดท้ายที่อยู่ของสไลม์ยักษ์เก่าแก่ ซึ่งเป็นผู้กุมกุญแจสำคัญในการจบเนื้อเรื่องเอาไว้ มีศัตรูจากทั่วเกมเกิดแบบสุ่ม เพื่อสนับสนุนบอสที่ขยับไปไหนไม่ได้

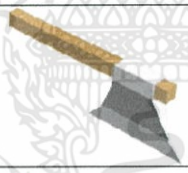
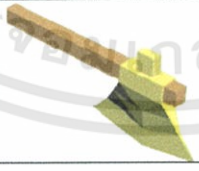
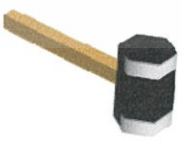
รายละเอียดระบบไอเท็ม (Item)

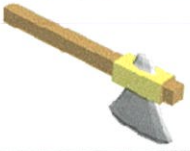
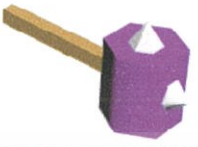
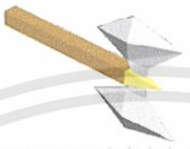
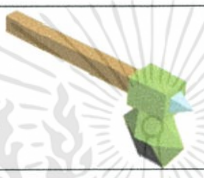
รายละเอียดของไอเท็มภายในเกมทั้งหมด สามารถจำแนกได้ดังตารางที่ ง-1 โดยที่ไอเท็มทุกชนิดจะสามารถขายได้ในราคาครึ่งหนึ่งของราคาที่กำหนด

ตารางที่ ง-1 แสดงรายละเอียดของไอเท็มทั้งหมดภายในเกม

1.อาวุธมือเดียว (Onehand Weapon)				
ชื่อ	**ไอคอน**	**โมเดล3มิติ**	**ราคา**	**รายละเอียด**
Wooden Sword			10	ดาบไม้ธรรมดา (Atk+3)
Survival Knife			150	มีดสั้นสำหรับนักผจญภัย (Atk+8 Agi+2)

ชื่อ	ไอคอน	โมเดล3มิติ	ราคา	รายละเอียด
Iron Sword			320	ดาบทำจากโลหะแข็ง อาจมีสนิมขึ้น(Atk+18)
Dagger			320	กริชทำจากโลหะแข็ง พกพาสะดวก(Atk+13, Agi +5)
Steel Sword			600	ดาบตีจากเหล็กกล้า ความแข็งแรงสูง(Agi +24)
Steel Blade			600	มีดทำจากเหล็กกล้า แหลมมาก (Atk+17 Agi +7)
Silver Sword			1000	ดาบตีจากเงิน คมกริบสะท้อนแสง (Atk+36)
Silver Dagger			1000	กริชเงิน ว่ากันว่ามีพลังชำระล้าง (Atk+29, Tec+11)
Uranium Saber			2500	ดาบยูเรเนียมสีเขียว หยุ่นๆ เหมือนสไลม์ แต่แข็งแกร่งกว่าเหล็กกล้า (Atk+50)
2.โล่ (Shield)				
ชื่อ	**ไอคอน**	**โมเดล3มิติ**	**ราคา**	**รายละเอียด**
Wood Plank			10	เปลือกไม้จากเรือที่ล่ม (Def + 3)
Leather Shield			100	โล่ทำจากหนัง ค่อนข้างหนา (Def +5)

ชื่อ	ไอคอน	โมเดล3มิติ	ราคา	รายละเอียด
Scale Shield			250	โล่ตีเป็นเกล็ดซ้อนกันสวยงาม (Def +9)
Shell Shield			500	โล่ทำจากโลหะมาตีจนเหมือนกระดอง (Def +13)
Iron Shield			700	โล่เหล็ก แข็งแรงทนทาน (Def +15)
Uranium Shield			1200	โล่ยูเรเนียม สะท้อนพลังจนเด้งออกไปได้ (Def +17)
3.อาวุธหนัก (Heavy Weapon)				
ชื่อ	**ไอคอน**	**โมเดล3มิติ**	**ราคา**	**รายละเอียด**
Mallet			20	ค้อนเคาธรรมดา (Atk + 5)
Hatchet			250	ขวานเล็ก เหวี่ยงสะดวก (Atk+12, Def+3)
Pound Hammer			550	ค้อนปอนด์สำหรับงานก่อสร้าง(Atk+27)
Iron Axe			550	ขวานทำจากโลหะแข็ง ใช้ตัดต้นไม้ได้ดี (Atk+21, Def+6)
Sledgehammer			1100	ค้อนทุบสำหรับงานรื้อถอน (Atk+48)

ชื่อ	ไอคอน	โมเดล3มิติ	ราคา	รายละเอียด
Steel Axe			1100	ขวานทำจากเหล็กกล้า ระวังอย่าทำหลุดมือ (Atk+39 , Def+9)
War Hammer			1700	ค้อนมีหนามที่ใช้กันในสงคราม (Atk+53)
Battle Axe			1700	ขวานสองคม ออกแบบมาสำหรับการต่อสู้ (Atk+42 , Def+11)
Halberd			3450	อาวุธที่นำเอาจุดเด่นของขวานและค้อนมารวมกัน (Atk+56)
4.อาวุธปืน (Range Weapon)				
ชื่อ	**ไอคอน**	**โมเดล3มิติ**	**ราคา**	**รายละเอียด**
Pistol			6	ปืนพกธรรมดา (Atk +3, Agi+1)
Mauser			100	สร้างขึ้นสำหรับชนเผ่าผิวเหลืองอันห่างไกล (Atk+6, Agi+2)
Remington 1100			220	ปืนอัดแก๊สสำหรับนักกีฬาผู้เหนื่อยเหนื่อย (Atk+10)
M1 Carbine			220	ไรเฟิลมาตรฐาน เบา ใช้ง่าย (Atk+15, Agi+3)
Benelli M3			560	ยิงได้ทั้งออโต้และไม่ออโต้ แต่โหมดออโต้ยังไม่สมบูรณ์ (Atk+17)

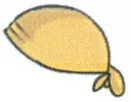
ชื่อ	ไอคอน	โมเดล3มิติ	ราคา	รายละเอียด
Karabiner 98k			560	ไรเฟิลพื้นฐานสำหรับกองทัพ (Atk+22 , Agi+6)
Saiga-12			970	ช็อตกันรูปร่างคล้ายปืนไรเฟิลจากดินแดนตะวันออก (Atk+24)
SKS			970	เก่าแต่พิษสงรุนแรง (Atk+34, Agi+10)
M1897			2000	ปืนช็อตกันในตำนาน(Atk+39)
5.กระสุนปืน (Bullets)				
ชื่อ	**ไอคอน**	**โมเดล3มิติ**	**ราคา**	**รายละเอียด**
Tim Bullet			5	ลูกกระสุนธรรมดา
Lead Bullet			10	ลูกกระสุนทำจากตะกั่ว
Pierce Bullet			20	ลูกกระสุนอณุภาพทะลุทะลวงสูง
6.คันธนู (Bow)				
ชื่อ	**ไอคอน**	**โมเดล3มิติ**	**ราคา**	**รายละเอียด**
Wooden Bow			50	ธนูไม้ธรรมดา (Atk+2)
Longbow			200	ธนูยาวออกแบบมาใช้ในการล่านก (Atk+10)

ชื่อ	ไอคอน	โมเดล3มิติ	ราคา	รายละเอียด
Steel Bow			800	ธนูด้ามเหล็ก ทนทาน แรงดีดสูง (Atk+25)

7.ลูกธนู (Arrow)

ชื่อ	ไอคอน	โมเดล3มิติ	ราคา	รายละเอียด
Wooden Arrow			2	ลูกธนูทำจากไม้เหลาให้คม (Atk+3)
Iron Head ArrowR			10	ลูกธนูหัวทำจากเหล็กเจียรให้คม ปลายแดง (Atk+8)
Iron Head ArrowB			10	ลูกธนูหัวทำจากเหล็กเจียรให้คม ปลายฟ้า (Atk+8)
Spiked Arrow			20	ลูกธนูปลายแหลม (Atk+15)

8.หมวก (Headgear)

ชื่อ	ไอคอน	โมเดล3มิติ	ราคา	รายละเอียด
Shroom Hat			10	เห็ดรูปร่างประหลาด เอามาใส่เป็นหมวกได้ (AllStat +1)
Straw Hat			100	หมวกฟาง กันแดดกันลมได้ดี (AllStat +2)
Bandana			200	ผ้าโพกหัว โพกแล้วดูดีใช้ย่อย (AllStat +3)
Leather Hat			350	หมวกผ้าถัก คงทนสูง (AllStat +4)

ชื่อ	ไอคอน	โมเดล3มิติ	ราคา	รายละเอียด
Chain Bandana			700	ผ้าโพกหัวถักจากใยเหล็ก กันกระสุนได้ (AllStat +5)
Iron Helm			1100	หมวกเหล็กทรงโบราณ ใส่แล้วอุ่นใจ (AllStat +6)
9.ของใช้ (Useable)				
ชื่อ	**ไอคอน**	**โมเดล3มิติ**	**ราคา**	**รายละเอียด**
Medic Herb			6	สมุนไพร สรรพคุณช่วยสมานแผลได้ดี (ฟื้นฟู 30HP)
Mystic Berry			10	ผลไม้ที่มีพลังเวทมนตร์ประจุอยู่ (ฟื้นฟู 5 MP)
Powershoot			40	หน่อไม้ที่มีสารบำรุงกำลัง (Allstat +3, 10วินาที)
Remedy Herb			100	สมุนไพรหายาก ช่วยฟื้นฟูพลังกาย (ฟื้นฟู 20% MaxHP)
Glowing Berry			200	ผลไม้เรืองแสงหายาก ช่วยฟื้นฟูพลังเวทมนตร์ (ฟื้นฟู 20% MaxMP)
Apple			20	แอปเปิ้ลแดง หนึ่งในสี่ผลไม้ป่า กินแล้วสดชื่น (ฟื้นฟู 50HP)
Papaya			20	มะละกอดิบ หนึ่งในสี่ผลไม้ป่า กรอบอร่อย (ฟื้นฟู 5 SP)

ชื่อ	ไอคอน	โมเดล3มิติ	ราคา	รายละเอียด
Banana			20	กล้วยสุก หนึ่งในสี่ผลไม้ป่า หวาน อร่อย (ฟื้นฟู 5 MP)
Orange			20	ส้ม หนึ่งในสี่ผลไม้ป่า เปรี้ยวหวานกำลังดี (ฟื้นฟู 30 HP , 5 MP)
Health Potion			50	ยาฟื้นพลังชีวิต ช่วยรักษาแผลได้ทันที (ฟื้นฟู 100 HP)
Mana Potion			90	ยาที่ประจุพลังเวทเอาไว้ ช่วยฟื้นคืนพลังเวทมนตร์ (ฟื้นฟู 30 MP)
Power Potion			100	ยาชูกำลัง เพิ่มพลังให้ผู้ดื่มชั่วขณะ (ฟื้นฟู 10SP , Allstat+3 นาน 15 วินาที)
Honey Essence			500	น้ำผึ้งสูตรพิเศษสกัดเข้มข้น ของดีเมืองรีโอวาโน (ฟื้นฟู 50SP , Allstat+3 นาน 20 วินาที)
Tablet Drug			50	ยาแบบอัดเม็ด ค่อยๆ ช่วยให้แผลสมานตัว (สถานะRegeneration1)
Capsule Drug			50	ยาแบบอัดแคปซูล ช่วยทำให้เดินได้คล่องขึ้น(สถานะWindwalk1)
Red Energy			150	ลูกพลังงานสีแดง เพิ่มพลังโจมตีชั่วคราว (+20 Atk นาน 20 วินาที)
Blue Energy			150	ลูกพลังงานสีฟ้า เพิ่มพลังป้องกันชั่วคราว (+20 Def นาน 20 วินาที)

ชื่อ	ไอคอน	โมเดล3มิติ	ราคา	รายละเอียด
Green Energy			150	ลูกพลังงานสีเขียว เพิ่มความคล่องชั่วคราว (+20 Agi นาน 20 วินาที)
Cure Liquid			550	ของเหลวสีเขียวเรืองแสง เหมาะสำหรับบำรุงกำลังสไลม์ (ฟื้นฟู 50% MaxHP)
Luminous Liquid			700	ของเหลวสีม่วงเปล่งแสง เหมาะสำหรับฟื้นฟูพลังเวทให้กับค้างคาว (ฟื้นฟู 50% MaxMP)
10.จิปาถะ (Misc.)				
ชื่อ	**ไอคอน**	**โมเดล3มิติ**	**ราคา**	**รายละเอียด**
Pebble			0	เศษก้อนกรวดตามชายหาด
Rock			6	ก้อนหินที่ถูกเซาะจากหินโสโครก
Heavy Shell			10	เปลือกหอย หนักมาก
Cultural Soil			40	ดินสำหรับเพาะปลูก
Palm Oak			40	ลูกปาล์มสำหรับทำน้ำมันพืช หนักมาก
Smiley Face			1	หน้ายิ้มสีเหลืองกลม ๆ
BigSmiley			100	รอยยิ้มโปรโมชั่น!

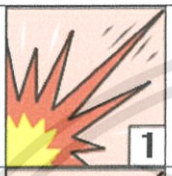
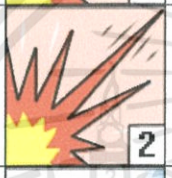
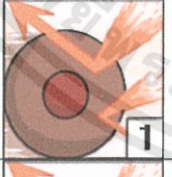
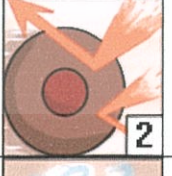
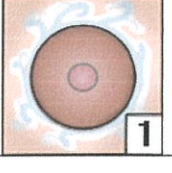
ชื่อ	ไอคอน	โมเดล3มิติ	ราคา	รายละเอียด
Angry Face			1	หน้าโมโหสีแดง ๆ
BigAngry			100	ระเบิดโปรโมชั่น!
Minibomb			100	ระเบิดลูกเล็ก ระวังระเบิดคามือ
Grenade			500	ระเบิดน้อยหน่า เป็นที่นิยมในสงคราม
Destructbomb			1000	ระเบิดทำลายล้าง ส่งเสียงดังสนั่นหวั่นไหว
Knot			10	น๊อต ส่วนประกอบทั่วไปสำหรับเครื่องจักร
Screw			10	สกรู ส่วนประกอบทั่วไปสำหรับเครื่องจักร
Nail			10	ตะปู สำหรับงานไม้
Charcoal			30	ถ่านไฟ สำหรับหุงต้ม
Coal			60	ถ่านหิน เชื้อเพลิงพื้นฐาน
Iron			150	แท่งเหล็ก ที่ถูกหักออกมา
Zinc			150	สังกะสีแผ่นลอน
Ancient Coin			500	เหรียญโบราณที่มีตราเวทมนตร์อยู่
Ancient Blade			500	ดาบเก่าๆ บิ่นแล้ว เหมาะแก่การสะสม

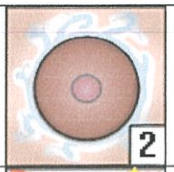
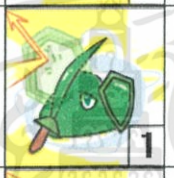
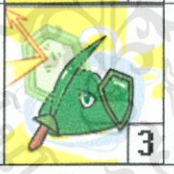
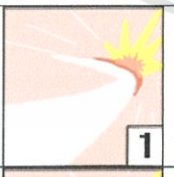
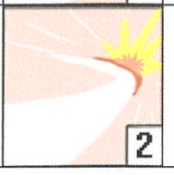
ชื่อ	ไอคอน	โมเดล3มิติ	ราคา	รายละเอียด
Ancient Pole			500	ไม้โบราณที่ยังมีใบไม้เขียวสดอยู่
Ancient Cup			500	ถ้วยโบราณทรงแปลก ๆ
Ancient Bone			500	กระดูกเก่าแก่ของสิ่งมีชีวิต ไม่ทราบที่มา
11.ปริศนา (Unknown)				
ชื่อ	**ไอคอน**	**โมเดล3มิติ**	**ราคา**	**รายละเอียด**
Pink Bulb			10	หน่อต้นไม้ธรรมดา
Blue Bulb			30	หน่อต้นไม้แข็งกลีบฟ้า แก่แล้ว
Green Bulb			50	หน่อต้นไม้กลีบเขียว หายาก
Wooden Box			100	ลังไม้ บรรจุอะไรไว้สักอย่าง
Locked Treasure			200	กล่องสมบัติที่ถูกล็อคอยู่
Safe Container			500	ตู้เซฟปิดผนึกแน่นหนา หายาก

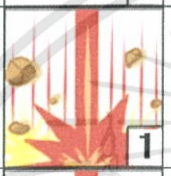
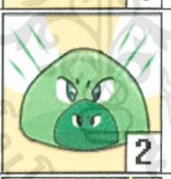
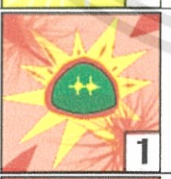
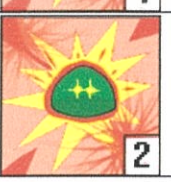
รายละเอียดของทักษะ

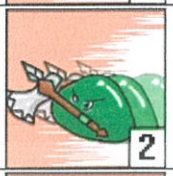
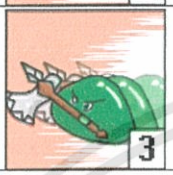
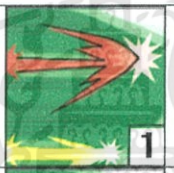
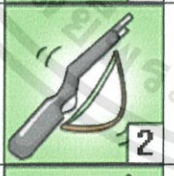
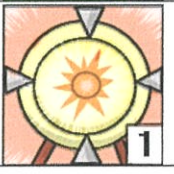
สำหรับทักษะทั้งหมดในเกม มีรายละเอียดดังตารางที่ ง-2

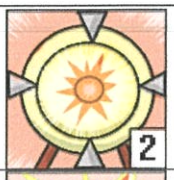
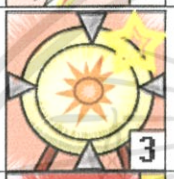
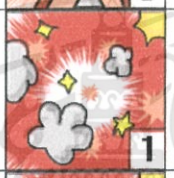
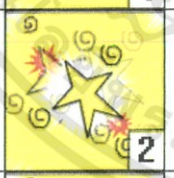
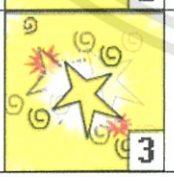
ตารางที่ ง-2 แสดงทักษะที่ตัวเอกสามารถเรียนรู้ได้

Onehand & Shield	**อาวุธมือเดียวและโล่ (One Hand & Shield)**		
ชื่อท่า	**ไอคอน**	**SP ที่ใช้**	**รายละเอียดของท่า**
Bash1		3	รวมพลังแล้วกระแทกเป้าหมายด้วยอาวุธ สร้างความเสียหายพิเศษ(x1.2Dmg)
Bash2		5	รวมพลังแล้วกระแทกเป้าหมายด้วยอาวุธ สร้างความเสียหายพิเศษ(x1.5Dmg)
Tanker1		N/A	ศาสตร์แห่งการป้องกัน เรียกพลังแฝงเพื่อเพิ่มพลังชีวิตเมื่อใช้อาวุธมือเดียวพร้อมโล่(MaxHP x 1.05)
Tanker2		N/A	ศาสตร์แห่งการป้องกัน เรียกพลังแฝงเพื่อเพิ่มพลังชีวิตเมื่อใช้อาวุธมือเดียวพร้อมโล่(MaxHP x 1.1)
Tanker3		N/A	ศาสตร์แห่งการป้องกัน เรียกพลังแฝงเพื่อเพิ่มพลังชีวิตเมื่อใช้อาวุธมือเดียวพร้อมโล่(MaxHP x 1.2)
Wall1		2	ยกโล่ขึ้นป้องกัน ลดความเสียหายที่ได้รับลง (Def x 1.5, 4 วินาที)
Wall2		3	ยกโล่ขึ้นป้องกัน ลดความเสียหายที่ได้รับลง (Def x 2, 6 วินาที)
Endurance1		N/A	ศาสตร์แห่งการป้องกัน ทำให้ตั้งโล่ได้นานขึ้น(+2วินาที)

ชื่อท่า	ไอคอน	SP ที่ใช้	รายละเอียดของท่า
Endurance2		N/A	ศาสตร์แห่งการป้องกัน ทำให้ตั้งโล่ได้นานขึ้น(+4วินาที)
WallExpert		N/A	เทคนิคการป้องกันชั้นสูง ทำให้ตั้งโล่ได้นานเท่าที่ต้องการจนกว่าจะขยับตัว
Ram1		10	พุ่งเข้ากระแทกเป้าหมายด้วยโล่ สร้างความเสียหายพิเศษพร้อมปิดการโจมตีทั้งหมดออกระหว่างใช้ท่า (Dmg = (Atk x 0.8) + (Def x 0.5))
Ram2		15	พุ่งเข้ากระแทกเป้าหมายด้วยโล่ สร้างความเสียหายพิเศษพร้อมปิดการโจมตีทั้งหมดออกระหว่างใช้ท่า (Dmg = (Atk x 0.8) + Def)
Focus1		N/A	เพ่งเล็งการจู่โจมของศัตรู ให้โอกาสลดความเสียหายที่ได้รับจากศัตรูที่ล็อคอยู่(x0.5Dmg , 10%)
Focus2		N/A	เพ่งเล็งการจู่โจมของศัตรู ให้โอกาสลดความเสียหายที่ได้รับจากศัตรูที่ล็อคอยู่(x0.5Dmg , 20%)
Focus3		N/A	เพ่งเล็งการจู่โจมของศัตรู ให้โอกาสลดความเสียหายที่ได้รับ(x0.5Dmg , 30%)
	**อาวุธหนัก** **(Heavy Weapon)**		
ชื่อท่า	**ไอคอน**	**SP ที่ใช้**	**รายละเอียดของท่า**
Swing1		5	เหวี่ยงอาวุธกวาดไปด้านหน้า สร้างความเสียหายพิเศษกับศัตรูด้านหน้า(x1.5Dmg)
Swing2		8	เหวี่ยงอาวุธกวาดไปด้านหน้า สร้างความเสียหายพิเศษกับศัตรูด้านหน้า(x1.8Dmg)

ชื่อท่า	ไอคอน	SP ที่ใช้	รายละเอียดของท่า
Gladiator1		N/A	ศาสตร์แห่งการจู่โจม ปลุกสัญชาติญาณเพื่อเพิ่มพลังโจมตีเมื่อถืออาวุธหนัก(Atk x 1.05)
Gladiator2		N/A	ศาสตร์แห่งการจู่โจม ปลุกสัญชาติญาณเพื่อเพิ่มพลังโจมตีเมื่อถืออาวุธหนัก(Atk x 1.1
Gladiator3		N/A	ศาสตร์แห่งการจู่โจม ปลุกสัญชาติญาณเพื่อเพิ่มพลังโจมตีเมื่อถืออาวุธหนัก(Atk x 1.1)
Smash1		15	กระแทกอาวุธลงพื้นเพื่อสร้างคลื่นกระแทก สร้างความเสียหายพิเศษกับศัตรูรอบตัว(x1.8Dmg)
Smash2		25	กระแทกอาวุธลงพื้นเพื่อสร้างคลื่นกระแทก สร้างความเสียหายพิเศษกับศัตรูรอบตัว(x2Dmg)
BuildUp1		N/A	เสริมสร้างความแข็งแรงให้สามารถรับน้ำหนักได้มากขึ้น เพิ่มพลังป้องกัน แต่ทำให้ใช้ทักษะช้าลง(Def x 1.05 , CD x 1.2)
BuildUp2		N/A	เสริมสร้างความแข็งแรงให้สามารถรับน้ำหนักได้มากขึ้น เพิ่มพลังป้องกัน แต่ทำให้ใช้ทักษะช้าลง(Def x 1.1 , CD x 1.2)
SolidBody		N/A	เสริมสร้างความแข็งแรงให้สามารถรับน้ำหนักได้มากขึ้น เพิ่มพลังป้องกัน แต่ทำให้ใช้ทักษะช้าลง(Def x 1.2 , CD x 1.1)
Rush1		20	ปลดปล่อยพลังแฝงออกมา ทำให้โจมตีได้รุนแรงขึ้น และใช้ทักษะได้บ่อยขึ้นชั่วระยะเวลา (x 1.2Dmg, CD x 0.8 , 15 วินาที)
Rush2		30	ปลดปล่อยพลังแฝงออกมา ทำให้โจมตีได้รุนแรงขึ้น และใช้ทักษะได้บ่อยขึ้นชั่วระยะเวลา (x 1.2Dmg, CD x 0.5 , 20 วินาที)

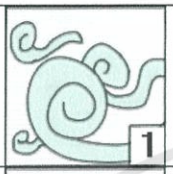
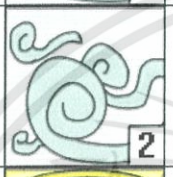
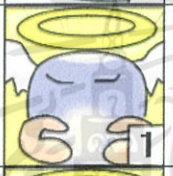
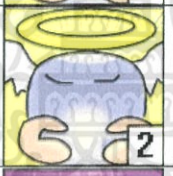
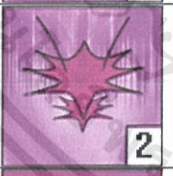
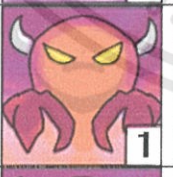
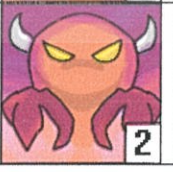
ชื่อท่า	ไอคอน	SP ที่ใช้	รายละเอียดของท่า
Swift1		N/A	ศาสตร์การใช้อาวุธหนักชั้นสูง เพิ่มความคล่องตัวในการเคลื่อนที่ให้กลับคืนมาและช่วยคงสภาวะของRush ให้นานขึ้น (Spd x 1.1 , x1.5 RushTime)
Swift2		N/A	ศาสตร์การใช้อาวุธหนักชั้นสูง เพิ่มความคล่องตัวในการเคลื่อนที่ให้กลับคืนมาและช่วยคงสภาวะของRush ให้นานขึ้น (Spd x 1.2 , x1.8 RushTime)
Swift3		N/A	ศาสตร์การใช้อาวุธหนักชั้นสูง เพิ่มความคล่องตัวในการเคลื่อนที่ให้กลับคืนมาและช่วยคงสภาวะของRush ให้นานขึ้น (Spd x 1.35 , x2 RushTime)
	อาวุธทางไกล (Range Weapon)		
ชื่อท่า	ไอคอน	SP ที่ใช้	รายละเอียดของท่า
Shot1		3	เหนี่ยวไกอย่างรวดเร็ว ยิงต่อเนื่องทันที (2 shot)
Shot2		5	เหนี่ยวไกอย่างรวดเร็ว ยิงต่อเนื่องทันที (3 shot)
Recoiless1		N/A	ต้านแรงดีดที่ได้รับจากการยิง ทำให้สามารถโจมตีปกติได้บ่อยขึ้น(5%)
Recoiless2		N/A	ต้านแรงดีดที่ได้รับจากการยิง ทำให้สามารถโจมตีปกติได้บ่อยขึ้น(10%)
Recoiless3		N/A	ต้านแรงดีดที่ได้รับจากการยิง ทำให้สามารถโจมตีปกติได้บ่อยขึ้น(15%)
Snipe1		10	หยุดอยู่กับที่แล้วเล็งเป้าหมายก่อนยิง สร้างความเสียหายพิเศษและมีโอกาสโดนจุดตายมากขึ้น (x 1.5Dmg, 5% Critical)

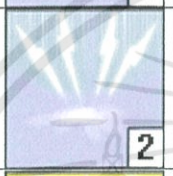
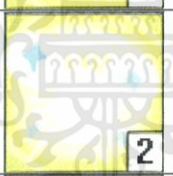
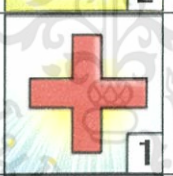
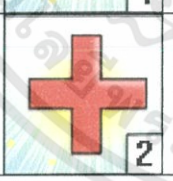
ชื่อท่า	ไอคอน	SP ที่ใช้	รายละเอียดของท่า
Snipe2		15	หยุดอยู่กับที่แล้วเล็งเป้าหมายก่อนยิง สร้างความเสียหายพิเศษและมีโอกาสโดนจุดตายมากขึ้น (x 2Dmg, 10% Critical)
BlankShot1		N/A	ศาสตร์แห่งความว่างเปล่า ให้โอกาสในการยิงแล้วไม่เสียลูกกระสุน(10%)
BlankShot2		N/A	ศาสตร์แห่งความว่างเปล่า ให้โอกาสในการยิงแล้วไม่เสียลูกกระสุน(20%)
Sniper		N/A	เทคนิคการยิงชั้นสูง เพิ่มโอกาสยิงแล้วไม่เสียกระสุนมากขึ้น และโจมตีโดนจุดตายได้บ่อยขึ้น (40%, +3% Critical)
Firework1		30	ประจุพลังใส่กระสุนแล้วยิงออกไป สร้างความเสียหายรุนแรงเป็นบริเวณกว้าง (x2.2Dmg)
Firework2		45	ประจุพลังใส่กระสุนแล้วยิงออกไป สร้างความเสียหายรุนแรงเป็นบริเวณกว้าง (x2.5Dmg)
ImpactShot1		N/A	ใช้สายตาที่เฉียบคมขึ้นในการเล็งเป้าหมาย ทำให้มีโอกาสยิงแล้วศัตรูเกิดอาการชะงัก (5%)
ImpactShot2		N/A	ใช้สายตาที่เฉียบคมขึ้นในการเล็งเป้าหมาย ทำให้มีโอกาสยิงแล้วศัตรูเกิดอาการชะงัก (10%)
ImpactShot3		N/A	ใช้สายตาที่เฉียบคมขึ้นในการเล็งเป้าหมาย ทำให้มีโอกาสยิงแล้วศัตรูเกิดอาการชะงัก (15%)

เวทมนตร์ของคู่หู (Rochel's Magic)

รายละเอียดเวทมนตร์ที่ค้างคาวสามารถเรียนรู้ได้ มีรายละเอียดดังตารางที่ 4-5 แบ่งเป็นสาย Buff , Debuff , Destruction และ Life อย่างละ 4 บทตามลำดับ

ตารางที่ ง-3 แสดงบทเวทมนตร์ที่คู่หูเรียนรู้ได้

ชื่อ	ไอคอน	ระดับ	MPที่ใช้	รายละเอียดของเวทมนตร์
WindWalk1		3	4	เวทสนับสนุนขั้นต้น ทำให้เคลื่อนที่ได้ไวขึ้น(Spd x1.2 ,10วินาที)
WindWalk2		9	9	เวทสนับสนุนขั้นต้น ทำให้เคลื่อนที่ได้ไวขึ้น(x1.5 spd ,10วินาที)
Guardian3		12	12	เวทช่วยเหลือ ช่วยเสริมค่าพลังให้ชั่วขณะหนึ่ง(Allstat+3, 15วินาที)
Guardian4		19	17	เวทช่วยเหลือ ช่วยเสริมค่าพลังให้ชั่วขณะหนึ่ง (Allstat+3, 15วินาที)
Bind1		4	3	เวทปั่นป่วนขั้นต้น ทำให้เป้าหมายติดสถานะเคลื่อนที่ได้ช้าลง (Spd x0.8 , 4วินาที)
Bind2		8	5	เวทปั่นป่วนขั้นต้น ทำให้เป้าหมายติดสถานะเคลื่อนที่ได้ช้าลง (Spd x0.5 , 10วินาที)
Burden3		13	8	เวทคำสาป ทำให้เป้าหมายอ่อนกำลังลง (Allstat -5 , 10วินาที)
Burden4		17	13	เวทคำสาป ทำให้เป้าหมายอ่อนกำลังลง (Allstat -5 , 15วินาที)

ชื่อ	ไอคอน	ระดับ	MPที่ใช้	รายละเอียดของเวทมนตร์
Lightning1		1	2	เวทจู่โจมขั้นต้น สร้างคลื่นไฟฟ้าโจมตีเป้าหมาย (Dmg x0.7)
Lightning2		6	5	เวทจู่โจมขั้นต้น สร้างคลื่นไฟฟ้าโจมตีเป้าหมาย (Dmg x1.0)
WhiteSpear3		11	10	เวททำลาย สร้างหอกไฟฟ้าทิ่มแทงศัตรู (Dmgx0.6 ,x2hits)
WhiteSpear4		15	17	เวททำลาย สร้างหอกไฟฟ้าทิ่มแทงศัตรู (Dmgx0.6 x3hits)
Regen1		2	2	เวทรักษาขั้นต้น ช่วยฟื้นฟูพลังชีวิตเล็กน้อยเมื่อเวลาผ่านไป (2%HP ทุก 3 วินาที นาน15วินาที)
Regen2		6	4	เวทรักษาขั้นต้น ช่วยฟื้นฟูพลังชีวิตเล็กน้อยเมื่อเวลาผ่านไป (5%HP ทุก 3 วินาที นาน15วินาที)
Heal3		7	10	เวทฟื้นฟู รักษาบาดแผลที่ได้รับทันที (20% MaxHP)
Heal4		21	20	เวทฟื้นฟู รักษาบาดแผลที่ได้รับทันที (30% MaxHP)

รายละเอียดศัตรูภายในเกม

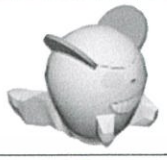
ศัตรูภายในเกมแต่ละตัวจะมีค่าพลัง รูปแบบ AI และผลตอบแทนที่ได้จากการเอาชนะแตกต่างกันไป ดังแสดงในตารางที่ ง-4 นอกเหนือจากนั้น ศัตรูบางประเภทจะมีรูปแบบการโจมตีที่แตกต่างกันไป บางทักษะก็เป็นทักษะที่ผู้เล่นหรือคู่หูใช้ได้ สำหรับรายละเอียดของทักษะและผลของทักษะที่มีศัตรูมีนอกเหนือจากที่ระบุ จะแสดงในตารางที่ ง-5

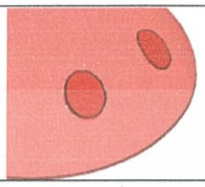
รายละเอียดของ AI แบ่งได้เป็น

- Normal ศัตรูจะไม่เข้ามาโจมตีผู้เล่น จนกว่าผู้เล่นจะทำร้ายมัน
- Aggro ศัตรูจะเข้ามาโจมตีผู้เล่นทันทีที่ผู้เล่นเข้าใกล้ตัว
- Run Away ศัตรูจะวิ่งหนีผู้เล่น หากได้รับความเสียหายเกิน 50% ของ HP ที่มี
- Hit&Run เมื่อศัตรูโจมตีแล้ว จะวิ่งหนีไปจากผู้เล่นชั่วคราว แล้ววกกลับมาโจมตีผู้เล่นต่อ
- Moremove เมื่อศัตรูได้รับความเสียหายถึงระดับหนึ่งจะใช้ทักษะบางอย่างเพิ่มขึ้น หรือมีค่าพลังบางอย่างสูงขึ้น

ตารางที่ ง-4 แสดงรายละเอียดของศัตรู

ชื่อ	โมเดล 3 มิติ	Lv.	HP	ATK	DEF	SPD	EXP	ทักษะ	AI
Machi		2	50	8	3	x1	2	Tackle	Normal, RunAway
Machidrill		10	240	42	25	x1	15	Drill	Aggro
Machina		18	480	86	50	x1.5	43	Chase	Aggro
Redball		4	75	25	1	x0.7	4	Slam	Aggro
VermiBall		12	250	100	1	x1	12	Slam	Aggro
BlackBall		20	500	200	1	x1.2	54	Slam	Aggro
Chu		7	150	36	10	x1.2	11	Head Butt	Hit&Run

ชื่อ	โมเดล 3 มิติ	Lv.	HP	ATK	DEF	SPD	EXP	ทักษะ	AI
Metchu		14	200	45	20	x1.5	31	Head Butt	Hit&Run
Ghochu		21	350	72	40	warp	61	Sweep	Hit&Run
BindPlant		6	80	10	10	0	5	Bite, Bind1	Aggro
Electriball		14	320	35	35	0	20	Lightning1	Aggro
Wisp		21	440	66	44	0	40	Blast, Burden2	Aggro
Goblin (Forest)		5	120	15	15	x1	25	Beat	Aggro, Hit&Run
Goblin2 (Factory)		16	250	40	40	x1	35	Beat	Aggro, Hit&Run
Crank		30	1,000	65	65	x1	65	Beat	Aggro, Hit&Run, , DefUp

ชื่อ	โมเดล 3 มิติ	Lv.	HP	ATK	DEF	SPD	EXP	ทักษะ	AI
ShadowBat		35	2,500	125	30	x1.2	30	Lightning2, Bind2, Burden2, WhiteSpear1	Hit&Run
GiantSlime		40	10,000	200	150	0	-	Random Barrage	Aggro
Lizard		5	100	5	5	x1	5	Beat	Aggro

ตารางที่ ง-5 แสดงรายละเอียดทักษะของศัตรู

ชื่อท่า	รายละเอียด
Tackle	ท่าโจมตีพื้นฐานของมาจิ เอาไหล่กระแทกชนเป้าหมายด้านหน้า (x1 Dmg)
Drill	ท่าโจมตีพื้นฐานของมาจิดริล ใช้หัวสว่านขวิดเป้าหมายด้านหน้า (x1 Dmg)
Chase	ตามเกาะติดเป้าหมาย อาจทำให้ผู้เล่นขยับไม่ได้
Slam	ท่าโจมตีพื้นฐานของบอล กลิ้งทับเป้าหมาย (x1 Dmg)
Head Butt	ท่าโจมตีพื้นฐานของชู เอาหัวกระแทกเป้าหมาย (x1 Dmg)
Sweep	ท่าโจมตีพิเศษของโกห์ชู ใช้หางฟาดแล้วหนีอย่างรวดเร็ว (x1 Dmg, Warp)
Bite	ท่าโจมตีพื้นฐานของไบน์แพลนท์ กัดเป้าหมายที่เดินผ่าน (x1 Dmg)
Blast	ท่าโจมตีพิเศษของวิสป์ สร้างคลื่นระเบิดรอบตัว (x1 Dmg, Area)
Beat	ท่าโจมตีพื้นฐานของก็อบลิน ใช้มือที่สวมปลอกแขนทุบเป้าหมาย (x1 Dmg)
Random Barrage	ท่าโจมตีของสไลม์ยักษ์ เสกสิ่งของขึ้นมาในอากาศแล้วปล่อยให้ตกลงเป้าหมายแบบสุ่ม (x1-3 Dmg, Area)