โปรแกรมสร้างและพัฒนาเกมสำเร็จรูป

Game Development Kit

โดย

นาย วรวิทย์ รัตนธเนศวิไล

นาย วิทวัส ยงเจริญ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

สาขาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2541

ปริญญานิพนธ์ปีการศึกษา 2541

ภาควิชา วิศวกรรมศาสตร์คอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง โปรแกรมสร้างและพัฒนาพัฒนาเกมสำเร็จรูป
(Game Development Kit)

ผู้จัดทำ

นาย วรวิทย์	รัตนธเนศวิไล	รหัสประจำตัว 38014438
นาย วิทวัส	ยงเจริญ	รหัสประจำตัว 38014464

..... อาจารย์ที่ปรึกษา

(.....)

โปรแกรมสร้างและพัฒนาเกมสำเร็จรูป

นาย วรวิทย์ รัตนธเนศวิไล
นาย วิทวัส ยงเจริญ
ดร. บุญธีร์ เครือตราชู
ปีการศึกษา 2541

บทคัดย่อ

โปรแกรมสร้างและพัฒนาเกมสำเร็จรูปนี้ถือเป็นโปรแกรมที่ได้ออกแบบมาสำหรับอำนวยความสะดวกให้กับการสร้างและพัฒนาเกมโดยใช้หลักการออกแบบและเขียนโปรแกรมเชิงวัตถุ(Object-Oriented Design and Programming) ทำให้ง่ายต่อการพัฒนา โดยในโปรแกรมนี้มีองค์ประกอบที่สำคัญอยู่ 4 ส่วนคือ

1) ส่วนของการสร้างฉากและแผนที่ 2) ส่วนของการสร้างวัตถุ
3) ส่วนของการสร้างเหตุการณ์ 4) ส่วนของระบบการขับเคลื่อนเกม

โดยในแต่ละส่วนนี้ถูกออกแบบแยกกันและจะมีการสร้างแฟ้มข้อมูลในแต่ละส่วนขึ้น และส่วนที่ทำให้เกมสามารถทำงานได้ก็คือส่วนของระบบการขับเคลื่อนเกมซึ่งขึ้นอยู่กับเกมแต่ละรูปแบบว่าจะมีการทำงานในส่วนนี้อย่างไร โดยส่วนของระบบการขับเคลื่อนเกมนี้จะนำแฟ้มข้อมูลที่ได้สร้างจากส่วนต่างๆ มาทั้งหมดพร้อมกับทำการกำหนดค่าสำหรับสิ่งต่างๆที่ต้องใช้เพื่อทำให้เกมทำงานได้

ในการทำให้ตัวเกมทำงานได้ตามที่ต้องการนั้นจะใช้หลักการที่ใช้เหตุการณ์เป็นตัวกำหนดการทำงานของแต่ละวัตถุรวมทั้งเหตุการณ์อื่นๆของเกมซึ่งเรียกว่า **Event-Driven Control** ซึ่งเป็นการควบคุมการทำงานของระบบหลักที่ใช้ในระบบปฏิบัติการไมโครซอฟท์วินโดวส์ รวมทั้งในตัวโปรแกรมได้มีการใช้ไมโครซอฟท์ไดเร็กเอ็กเวอร์ชัน 5 (Microsoft DirectX 5.0) ในการติดต่อกับการทำงานอุปกรณ์ต่างๆบนเครื่องคอมพิวเตอร์เช่น การแสดงผลบนจอภาพ(Direct Draw) การรับข้อมูลจากอุปกรณ์ต่างๆ (Direct Input) เช่น คีย์บอร์ด และการส่งเสียงออกทางลำโพง(Direct Sound) ในการติดต่อกับอุปกรณ์เหล่านี้จะทำได้ด้วยความรวดเร็ว และมีประสิทธิภาพมากขึ้นกว่าการใช้งานผ่านทางระบบปฏิบัติการไมโครซอฟท์วินโดวส์

Game Development Kit

Mr. Worawit Rattanataneswilai

Mr. Witawat Yongchareon

Dr. Boontee Kruatrachue Advisor

1998

Abstract

Game Development Kit is designed for conventional in creating and development game, which use the Object-Oriented Design and Programming technology that can be development easy. The overall part of this program are shown below

1. Map Editor
2. Object Editor
3. Event Editor
4. Game Runtime

All of above part in this program will be designed independently and each part will generate its own file. Engine that make the game can running is the Game Runtime part, which depend on the type of game that it working on. Game Runtime will load all of file, which created by Game Tool, and initialize all of component in the Game Runtime System for preparing to start the game.

The concept of Game Runtime, running the game and controlling all of game components, is design to use the model of Event Driven Control, which be used in Microsoft Windows. Include using Microsoft DirectX version 5.0 technology for helping the way to communicate and control hardware on the computer, such as displaying image on screen by using DirectDraw , getting input control from input device by using DirectInput , playing sound by using DirectSound etc. This DirectX technology help all of these doing more faster and more efficiency than use normal Application Program Interface (API) that provide by Microsoft Windows.

กิตติกรรมประกาศ

วิทยานิพนฉบับนี้คงไม่อาจเสร็จได้ด้วยดีหากไม่ได้รับความช่วยเหลือ และความร่วมมือจากหลายๆฝ่ายด้วยกัน บุคคลแรกที่ต้องกล่าถึงเพราะเป็นส่วนสำคัญที่ทำให้วิทยานิพนธ์ฉบับนี้เสร็จลงได้ก็คือ ดร. บุญธีร์ เครือตราชู อาจารย์ที่ปรึกษาวิทยานิพนธ์ ที่ให้ความเอาใจใส่ คำแนะนำ และช่วยเหลือในหลายๆด้านตลอดเสมอมา ซึ่งต้องขอขอบพระคุณเป็นอย่างสูง

และต้องขอขอบพระคุณบุคคลที่สำคัญที่สุดที่ทำให้ข้าพเจ้าได้มีวันนี้ ก็คือ บิดา มารดา อันเป็นที่เคารพรักยิ่ง ซึ่งได้เลี้ยงดูผู้เขียนมาเป็นอย่างดี พร้อมทั้งให้โอกาสในการศึกษาอย่างเต็มที่ และยังให้กำลังใจเอาใจใส่เสมอมาในทุกๆด้านอันหาที่เปรียบมิได้ ข้าพเจ้าขอระลึกในพระคุณอันสุดประมาณ และขอกราบขอบพระคุณมา ณ.ที่นี้

และขอกราบขอบพระคุณอาจารย์ทุกๆท่านที่ได้สั่งสอนวิชาความรู้ต่างๆที่เป็นส่วนสำคัญให้ความสำเร็จของปริญญานิพนธ์นี้ เป็นไปได้ด้วยดี และต้องขอขอบพระคุณอาจารย์ บรรจง ปิยธำรงค์ ที่ทำให้ข้าพเจ้าได้มีสถานที่และอุปกรณ์ในการทำปริญญานิพนธ์นี้

สุดท้ายนี้ต้องขอขอบใจเพื่อนๆชาว Gang4D และน้องๆทุกคนที่มีส่วนร่วมในการทำให้ปริญญานิพนธ์นี้สำเร็จไปด้วยดี และช่วยเหลือข้าพเจ้าในทุกๆด้าน

สารบัญ

สารบัญภาพ

หน้าที่

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มา

เกม เป็นโปรแกรมที่สร้างขึ้นมาเพื่อให้ผู้เล่นมีความเพลิดเพลินและสนุกสนาน ช่วยให้ผู้เล่นผ่อนคลายจากการทำงาน และถือเป็นการพักผ่อนไปในตัว และในปัจจุบันนี้เกมต่างๆได้รับความนิยมอย่างสูง เกือบทุกคนที่จะมีเกมไว้เล่นบนคอมพิวเตอร์ ขณะที่บริษัทผลิตเกมทั้งหลายได้สร้างและพัฒนาเกมต่างๆ ออกมาได้อย่างรวดเร็ว การเขียนโปรแกรมเกมใหม่ๆได้อย่างรวดเร็วนั้นเป็นไปได้ยากในระดับหนึ่งที่นักเขียนโปรแกรมจะเขียนได้ จึงจำเป็นต้องมีการสร้างโปรแกรมที่ใช้ในการสร้างและพัฒนาเกมขึ้นมาเพื่ออำนวยความสะดวกในการสร้างและพัฒนาเกมใหม่ๆให้ได้รวดเร็วทันใจกับความต้องการของนักเล่นเกมที่มีอยู่ในปัจจุบันนี้ โดยในแต่ละโปรแกรมที่ใช้ในการสร้างเกมนั้นก็จะมีลักษณะและรูปแบบของเกมที่สร้างหรือพัฒนาได้เฉพาะ ดังนั้นการที่มีเกมออกมาสู่ตลาดได้หลายๆรูปแบบนั้นจำเป็นต้องมีการทำโปรแกรมสร้างและพัฒนาเกมในรูปแบบนั้นๆออกมาใช้ในแต่ละบริษัท และเมื่อต้องการนำเกมรูปแบบที่เคยสร้างมา?ล้วนั้นมาพัฒนาให้ดียิ่งขึ้นหรือมีการเพิ่มเติมอะไรบางอย่างเข้าไปก็สามารถทำได้โดยง่ายและรวดเร็วเนื่องจากมีโปรแกรมที่ได้ทำไว้แล้วนั้นนำมาพัฒนาในส่วนนี้

โปรแกรมสร้างและพัฒนาเกมสำเร็จรูป ได้ใช้หลักในการออกแบบเชิงวัตถุ(Object-Oriented Design) และในทำนองเดียวกันก็เขียนโปรแกรมด้วยหลักการเขียนโปรแกรมเชิงวัตถุ(Object-Oriented Programming) เนื่องจากการใช้หลักการนี้จึงทำให้โปรแกรมสร้างและพัฒนาเกมสำเร็จรูปสามารถพัฒนาได้ง่ายและรวดเร็วถ้าต้องการจะเปลี่ยนรูปแบบเกมที่จะสร้าง

สำหรับโปรแกรมสร้างเกมนี้มีความสามารถที่จะสร้างเกมและนำมาเล่นได้โดยที่ผู้ใช้งานไม่จำเป็นต้องมีความรู้เกี่ยวกับการเขียนโปรแกรมเลยแม้แต่น้อยซึ่งเป็นผลทำให้ผู้ที่ต้องการสร้างเกมขึ้นมาเล่นเองแต่ไม่มีความสามารถในการเขียนโปรแกรมสามารถสร้างเกมที่ตนเองต้องการขึ้นมาได้ และนี่เป็นจุดประสงค์ที่สำคัญที่สุดสำหรับการสร้างโปรแกรมนี้ เนื่องด้วยการเขียนเกมอันยุ่งยาก,ช้า และต้องการคนที่มีความชำนาญเฉพาะด้าน โดยปรกติแล้วโปรแกรมส่วนใหญ่จะใช้สำหรับการสร้างข้อมูลของเกมซึ่งไม่ได้สร้างการทำงานของเกม แต่ในโปรแกรมสร้างและพัฒนาเกมสำเร็จรูปนี้จะมีการสร้างส่วนของระบบการขับเคลื่อนของเกมด้วย และอีกอย่างที่สำคัญคือโปรแกรมนี้ได้ใช้ความสามารถของไมโครซอฟท์ไดเร็กเอ็กเวอร์ชัน 5.0 (Microsoft DirectX5.0) ซึ่งช่วยให้การเขียนเกมมีประสิทธิภาพมากยิ่งขึ้น

จากที่กล่าวมาทั้งหมดนี้ จึงได้มีการคิดและสร้างโปรแกรมขึ้นมาโดยเป็นชื่อที่เรียกว่า "**โปรแกรมสร้างและพัฒนาเกมสำเร็จรูป (Game Development Kit)**"

1. 2 วัตถุประสงค์

1. เพื่อสร้างโปรแกรมที่ทำให้ผู้ใช้สามารถสร้างและพัฒนาเกมด้วยตนเองโดยที่ผู้ใช้ไม่จำเป็นต้องมีความรู้ในการเขียนโปรแกรมแต่อย่างใด

2. เพื่อสร้างโปรแกรมที่สามารถนำไปพัฒนาต่อให้มีความสามารถเพิ่มขึ้นไปอีกได้
3. เพื่อสร้างโปรแกรมที่ตอบสนองความต้องการของตลาดเกมในปัจจุบันนี้รวมทั้งในอนาคต
4. เพื่อนำโปรแกรมที่ได้สร้างนี้นำไปสร้างและพัฒนาเกมออกมาให้ผู้สนใจได้ทดลองเล่น
5. เพื่อนำโปรแกรมที่ได้สร้างนี้ไปพัฒนาต่อให้มีความสามารถสูงขึ้นไปอีก ทำให้สามารถสร้างและพัฒนาเกมในรูปแบบใหม่ๆได้
6. เพื่อทดลองและศึกษาระบบการทำงานของเกมที่มีอยู่ในปัจจุบันนี้ รวมทั้งรูปแบบที่ยังไม่มี
7. เพื่อคิดค้นรูปแบบของโปรแกรมที่ยังไม่ค่อยมีคนสร้างขึ้นมา

1.3 ประโยชน์ที่ได้รับ

1. รู้จักการออกแบบและเขียนโปรแกรมเชิงวัตถุ (Object-Oriented Design and Programming)
2. รู้จักการออกแบบการทำงานของเกม
3. รู้จักการออกแบบเทคนิคต่างๆที่ใช้ในการสร้างโปรแกรมให้มีประสิทธิภาพ
4. รู้จักการใช้งานโปรแกรมที่ใช้ในการเขียนโปรแกรมเช่น ไมโครซอฟท์ไดเร็กเอ็ก(Microsoft DirectX) ,บอร์แลนด์ซีพลัสพลัสบิวเดอร์(Borland C++Builder) และ โปรแกรมต่างๆที่ใช้ในการสร้างและแต่งรูปภาพ
5. สามารถสร้างและพัฒนาเกมดังที่ต้องการได้
6. ทำให้บุคคลทั่วไปได้รับความสะดวกในการสร้างและพัฒนาเกม
7. เป็นจุดเริ่มต้นในการสร้างและพัฒนาเกมให้ดียิ่งขึ้นไป

1.4 ขอบเขตและวิธีดำเนินงาน

ในการสร้างโปรแกรมนี้นั้นจำเป็นต้องใช้ความรู้ต่างๆดังนี้

1. การออกแบบโปรแกรมเชิงวัตถุ(Object-Oriented Design)
2. การเขียนภาษาซี โดยเขียนเป็นแบบเชิงวัตถุ(Object-Oriented Programming)
3. การเข้าใจและการใช้งานไมโครซอฟท์ไดเร็กเอ็กเวอร์ชัน 5.0(Microsoft DirectX5.0)
4. หลักการที่ใช้ในการเก็บและแสดงผลภาพ(Tile-Based)
5. หลักการที่ใช้ในการติดต่อกับไมโครซอฟท์วินโดวส์(Microsoft Windows)
6. หลักการที่ใช้ในการออกแบบการทำงานของเกม(Game Runtime)
7. หลักการที่ใช้ในการเก็บและใช้ข้อมูลให้มีประสิทธิภาพ(Data Structure)
8. หลักการที่ใช้ในการเก็บข้อมูลต่างๆลงในแฟ้มข้อมูล(File Structure)
9. หลักการที่ใช้ในการในการควบคุมการทำงานของเกมโดยใช้เหตุการณ์เป็นตัวกำหนด(Event-Driven Control)

จากความรู้ทั้งหมดนี้สามารถนำมาสร้างโปรแกรมนี้ได้โดยมีขั้นตอนในการทำดังนี้

1. ออกแบบโครงสร้างหลักของโปรแกรม

2. แบ่งโครงสร้างหลักของโปรแกรมออกเป็นส่วนย่อยๆ
3. ออกแบบโครงสร้างภายในของโครงสร้างย่อยๆเหล่านั้น
4. ออกแบบความสัมพันธ์ระหว่างโครงสร้างย่อยๆเหล่านั้น
5. ออกแบบการเก็บและการใช้ข้อมูลในแต่ละส่วน
6. ออกแบบระบบการทำงานของเกม(Game Runtime)
7. เก็บข้อมูลต่างๆที่ใช้ในการเขียนโปรแกรม(Document)
8. เริ่มเขียนโปรแกรมในแต่ละส่วน
9. ทดสอบและแก้ไขข้อผิดพลาดในแต่ละส่วน
10. เชื่อมต่อโปรแกรมในแต่ละส่วนเข้าด้วยกัน
11. ทดสอบและแก้ไขข้อผิดพลาดของโปรแกรมที่เชื่อมกันแล้ว
12. สร้างต้นแบบของโปรแกรมที่ทำสำเร็จในครั้งแรก
13. ปรับปรุงโปรแกรมให้มีประสิทธิภาพมากยิ่งขึ้นโดยแยกพัฒนาในแต่ละส่วน
14. ทดสอบและแก้ไขโปรแกรมที่ได้ปรับปรุงแล้ว
15. สร้างต้นแบบของโปรแกรมที่ได้มีการปรับปรุงแล้ว

เนื่องจากการสร้างโปรแกรมนี้นั้นไม่ได้ระบุความต้องการและข้อจำกัดของโปรแกรมไว้อย่างแน่นอนซึ่งขึ้นอยู่กับความเป็นไปได้ในการออกแบบและเขียนโปรแกรม และจำเป็นที่จะต้องทดสอบการทำงานต่างๆในแต่ละส่วนซึ่งอาจจะมีผลให้ต้องมีการเปลี่ยนแปลงโครงสร้างบางโครงสร้างเพื่อให้โปรแกรมทำงานได้ดีขึ้น

บทที่ 2
หลักการและทฤษฎีพื้นฐาน

2.1 ส่วนประกอบของเกม

ส่วนประกอบของเกมในมุมมองของนักพัฒนาโปรแกรมโดยทั่ว ๆ ไปแล้วจะประกอบด้วย

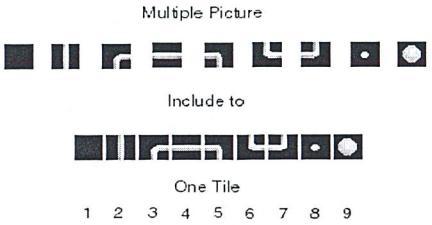
1. ส่วนที่เป็นฉากและแผนที่
2. ส่วนที่เป็นวัตถุที่ไม่มีการเปลี่ยนแปลง เช่น ต้นไม้ ก้อนหิน บ้าน เป็นต้น
3. ส่วนที่เป็นวัตถุที่มีการเปลี่ยนแปลงได้ เช่น ตัวละครผู้เล่น ศัตรู ตัวประกอบฉาก เป็นต้น
4. ส่วนที่เป็นเหตุการณ์ของเกมซึ่งจะเกิดขึ้นกับวัตถุต่างๆ หรือบนฉาก เช่น คนโดนยิง เป็นต้น
5. ส่วนที่เป็นบทบาทของวัตถุและบทบาทรวมในเกมซึ่งเป็นการกำหนดการดำเนินเรื่องของตัวละครและตัวเกม เช่น คนพูดข้อความเมื่อโดนยิง เป็นต้น
6. ส่วนขับเคลื่อนเกมซึ่งทำหน้าที่เป็นตัวควบคุมเหตุการณ์ และส่วนนี้เองที่เป็นตัวกำหนดประเภทของเกมนั้น ๆ ว่าจะเป็นเกมประเภทใด มีวิธีเล่นอย่างไร และมีเนื้อเรื่องอย่างไร

2.2 การทำงานของเกม

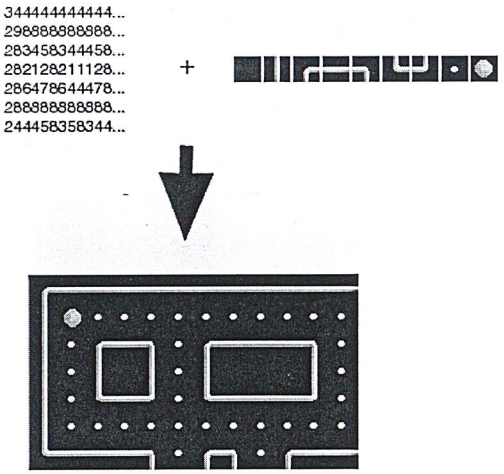
จากที่ได้กล่าวไปแล้วว่าเกมประกอบด้วย 6 ส่วนหลัก การทำงานของเกมนั้นก็จะมีส่วนที่เป็นส่วนขับเคลื่อนที่จะทำการวนลูปเพื่อตรวจสอบการทำงานของวัตถุต่าง ๆ ที่มีการกำหนดเหตุการณ์ไว้ว่าจะสร้างเหตุการณ์ต่าง ๆ ขึ้นมา เช่น ตัวละครของผู้เล่น ศัตรู หีบสมบัติ สิ่งของสำคัญในฉาก เป็นต้น เมื่อส่วนขับเคลื่อนได้รับเหตุการณ์จากผู้ส่งแล้วก็จะประมวลเหตุการณ์เทียบกับระบบการตรวจสอบเหตุการณ์ แล้วจัดการกับวัตถุที่ได้รับเหตุการณ์นั้น ๆ ในรูปแบบที่กำหนดไว้ในบทบาท(Script) เช่น ยานบินของผู้เล่นถูกศัตรูโจมตี เหตุการณ์ที่เกิดขึ้นก็อาจจะเป็น พลังชีวิตลด หรือ ยานบินของผู้เล่นระเบิด ซึ่งจะเกิดเหตุการณ์ใดนั้นก็ขึ้นอยู่กับว่าเหตุการณ์ในแต่ละวัตถุถูกกำหนดเอาไว้อย่างไรและมีการทำงานเมื่อใด

2.3 Tile-based game

Tile-based game นั้นหมายถึงเกมที่ใช้รูปแบบการแสดงผลที่อยู่ในรูปของ Tilemap แทนการใช้การแสดงผลในรูปของบิทแมพ(Bitmap) ก่อนที่จะพูดถึงไทล์แมพ(Tilemap) นั้นก็จะต้องพูดถึง ไทล์(Tile) ก่อนว่าคืออะไร ซึ่งจริง ๆ แล้วไทล์(Tile) ก็คือ ชุดของสิ่งของที่มีขนาดเท่า ๆ กันและถูกนำมาเชื่อมต่อกันเป็นชิ้นเดียว ซึ่งในที่นี้ก็คือรูปภาพนั่นเอง ส่วนการแสดงผลแบบไทล์แมพ(Tilemap) นั้นก็คือ การนำข้อมูลในรูปของไฟล์แบบไบนารีไฟล์ มาทำการเทียบกับไฟล์ไทล์ แล้วจึงนำรูปภาพในไฟล์ไทล์ที่เทียบได้มาทำการแสดงผลทางหน้าจอ



รูปที่ 2.1 แสดงรายละเอียดของไทล์



รูปที่ 2.2 แสดงไทล์แมพ(Tilemap) ที่ได้มาจากนำข้อมูลในไฟล์ไทล์(Tile File)มาใช้ร่วมกับข้อมูลที่ได้จากไบนารี่ไฟล์

จากรูปที่ 2.2 แสดงการทำไทล์แมพ(Tilemap) จะเห็นได้ว่าการทำเกมแบบไทล์แมพ(Tilemap) นั้นมีข้อดีที่สำคัญคือช่วยในการลดขนาดของข้อมูลที่จะเก็บได้มากโดยแทนที่จะต้องเก็บฉากและส่วนประกอบของเกมทุกพิกเซลในระบบบิทแมพ(Bitmap) ก็จะเก็บเพียงไฟล์ข้อมูลแบบไบนารีกับไฟล์ไทล์(Tile File) เท่านั้น นอกจากนั้นยังช่วยในการกำหนดตำแหน่งของฉากได้อย่างคร่าว ๆ ได้อีกด้วย

2.4 ไดเร็กเอ็ก(DirectX)

เป็นกลุ่มของเทคโนโลยีที่พัฒนาโดยไมโครซอฟท์ เพื่อให้โปรแกรมทางด้านมัลติมีเดียบน เครื่องคอมพิวเตอร์ส่วนบุคคล(Personal Computer: PC) ที่ใช้ระบบปฏิบัติการวินโดวส์มีประสิทธิภาพสูงยิ่งขึ้น เช่น

สามารถแสดงสีได้เหมือนจริง,เล่นภาพวีดีโอได้ดีขึ้น เมื่อเทียบกับการเขียนโปรแกรมบนระบบปฏิบัติการวินโดวส์แบบปกติ และไม่ต้องคำนึงถึงรายละเอียดทางฮาร์ดแวร์มากเกินไปเหมือนกับการพัฒนาโปรแกรมบนระบบปฏิบัติการดอส

สำหรับเทคโนโลยีไดเร็กเอ็ก(DirectX) นั้นก็จะประกอบด้วยองค์ประกอบย่อยอีกหลาย ๆ ส่วนซึ่งมีหน้าที่ต่าง ๆ ดังนี้

1.ไดเร็กดรอ(DirectDraw) เป็นเทคโนโลยีที่ช่วยในการเร่งความเร็วของฮาร์ดแวร์ในส่วนของการแสดงผลเพื่อให้สามารถทำได้เร็วยิ่งขึ้น ซึ่งจะทำให้มีความเร็วในการแสดงผลมากกว่าการแสดงผลผ่านทางส่วนเชื่อมต่อกับอุปกรณ์ทางกราฟิก(Graphics Device Interface: GDI) ของวินโดว์

2.ไดเร็กอินพุท(DirectInput) เป็นเทคโนโลยีที่ช่วยในการควบคุมการใช้งานอุปกรณ์อินพุตได้หลากหลาย โดยใช้งานไดเร็กอินพุท(DirectInput)ผ่านส่วนที่ทำหน้าที่เชื่อมต่อกับตัวโปรแกรมประยุกต์(Application Program Interface) ของวินโดวส์

3.ไดเร็กซาวด์(DirectSound) เป็นเทคโนโลยีที่ช่วยในการผสมและเล่นเสียงในหลายรูปแบบทั้งการใช้ความสามารถของฮาร์ดแวร์และซอฟท์แวร์

4.ไดเร็กสามดี(Direct3D) เป็นเทคโนโลยีที่มีส่วนเชื่อมต่อกับโปรแกรมประยุกต์(API) ระดับสูงเพื่อช่วยในการสร้างโปรแกรมประยุกต์ที่เกี่ยวกับภาพสามมิติ

5.ไดเร็กเพลย์(DirectPlay) เป็นเทคโนโลยีที่มีส่วนเชื่อมต่อกับโปรแกรมประยุกต์(API) ที่ทำหน้าที่ช่วยในการสร้างโปรแกรมติดต่อผ่านโมเด็มหรือเน็ตเวิร์ก โดยใช้โปรโตคอลหลายชนิด

จากที่ได้กล่าวถึงคุณสมบัติต่าง ๆ ของไดเร็กเอ็กแล้วจะเห็นได้ว่าไดเร็กเอ็กนั้นมีประโยชน์ในการช่วยในการสร้างเกมเป็นอย่างมาก เช่น ในเรื่องของการแสดงผลไดเร็กดรอจะมีส่วนในการจัดการทำบัฟเฟอร์(Buffer)ของหน้าจอหลักเพื่อที่จะทำให้สามารถทำการเลื่อน(scroll)ภาพได้เรียบมากขึ้นหรือการจัดทำชั้นของการแสดงผล(layer) เพื่อช่วยในการเคลื่อนวัตถุไปบนฉากได้ง่ายขึ้นโดยไม่ต้องคอยกังวลเรื่องการเปลี่ยนแปลงของฉากหลัง หรือการจัดการเกี่ยวกับการรับอินพุทของเกม เราก็สามารถใช้งานฟังก์ชันต่าง ๆ ผ่านทางไดเร็กอินพุท (DirectInput) ซึ่งช่วยในการจัดการข้อมูลทางอินพุท

2.5 หลักการทำงานของไดเร็กเอ็ก

ในส่วนของโปรแกรมที่พัฒนาขึ้นมานี้ได้มีการใช้ไดเร็กเอ็กเพียงบางส่วนคือ ส่วนของการแสดงผล(ใช้ไดเร็กดรอ: DirectDraw) และส่วนของการจัดการอินพุท(ใช้ไดเร็กอินพุท: DirectInput) ซึ่งก็มีหลักการทำงานขั้นพื้นฐานดังนี้

ไดเร็กดรอ(DirectDraw)

ทำหน้าที่จัดการให้โปรแกรมทางด้านกราฟฟิกสามารถติดต่อกับวีดีโอเมมโมรีและวีดีโอฮาร์ดแวร์ได้โดยตรง โดยไม่ต้องติดต่อผ่านทางส่วนเชื่อมต่อกับอุปกรณ์ทางกราฟิก (GDI) ของวินโดวส์ซึ่งทำงานได้ช้า(GDI ของวินโดวส์ถูกออกแบบมาเพื่อช่วยในการแสดงผลสำหรับโปรแกรมหลาย ๆ โปรแกรมที่มีการใช้งานร่วมกัน)

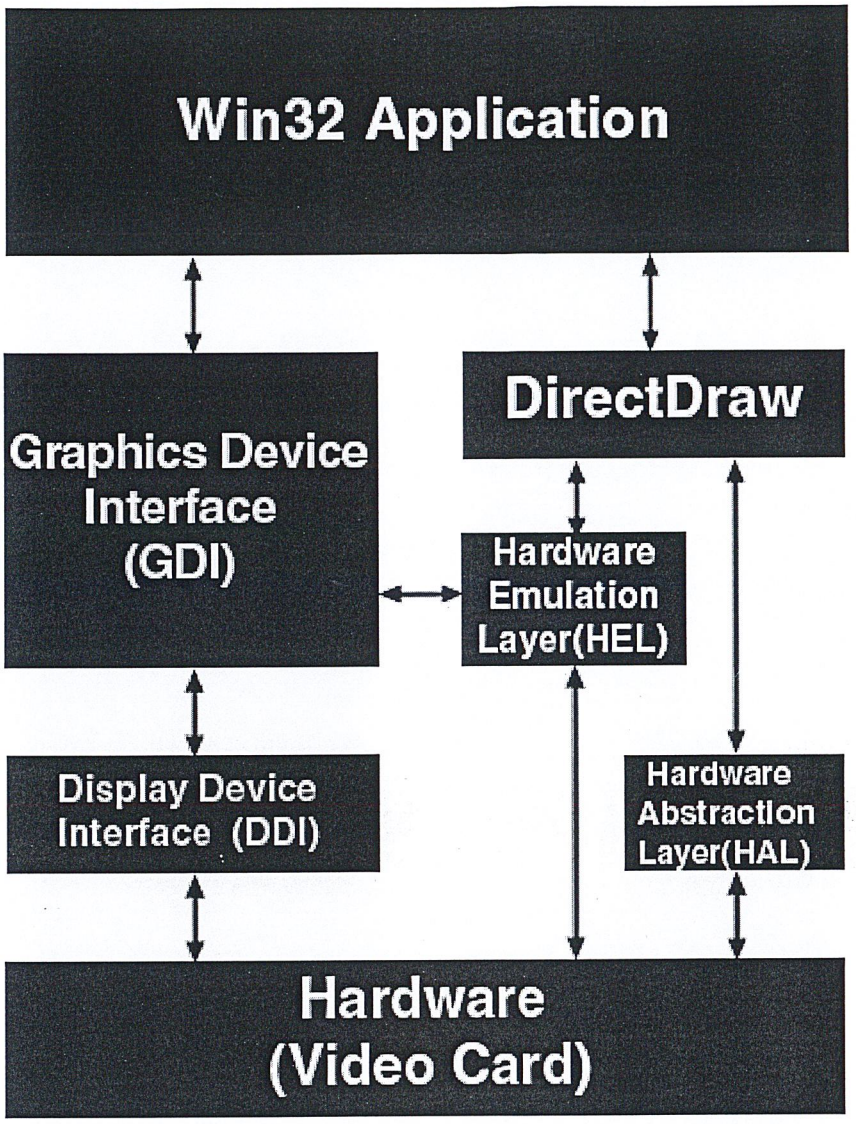
ไดเร็กดรอเป็นส่วนเชื่อมต่อกับโปรแกรม(API)ในระดับต่ำที่จะทำการติดต่อกับวีดีโอเมมโมรีโดยตรง

ด้วยการใช้สถาปัตยกรรม HAL/HEL (Hardware Abstraction Layer/Hardware Emulation Layer)ซึ่งจะทำให้เราสามารถเข้าถึงความสามารถเฉพาะของการ์ดแต่ละรุ่นได้โดยไม่จำเป็นต้องเขียนโปรแกรมเฉพาะการ์ดแสดงผลแต่ละอัน

จากภาพแสดงให้เห็นว่าการทำงานของไดเร็กดรอ(DirectDraw) นั้นไม่ผ่านส่วนเชื่อมต่อกับอุปกรณ์ทางกราฟิก(GDI) แต่จะผ่าน HEL และ HAL แทนซึ่งจะมีรายละเอียดดังนี้

HAL เป็นอินเตอร์เฟสที่เจาะจงกับฮาร์ดแวร์ของแต่ละผู้ผลิต แต่ส่วนใหญ่จะเป็นฟังก์ชันมาตรฐานที่ต้องมีอยู่แล้วในการ์ดแสดงผลทั่วไป ซึ่งเป็นหน้าที่ของผู้ผลิตการ์ดแสดงผลนั้นต้องเขียนไดร์เวอร์สำหรับสนับสนุนการทำงานของ HAL โดยเฉพาะ ฟังก์ชันกราฟิกของ HAL จะต้องเป็นความสามารถโดยตรงของชิปบนการ์ดแสดงผล แต่ถ้าการ์ดแสดงผลไม่สนับสนุนฟังก์ชันนั้น ก็จะมีรายงานให้กับไดเร็กดรอ(DirectDraw)

HEL เป็นส่วนที่ทำหน้าที่จำลองการทำงานของฮาร์ดแวร์ให้มาทำงานด้วยซอฟท์แวร์โดยใช้หน่วยประมวลผลหลัก(เนื่องจากไม่สามารถที่จะทำให้ฮาร์ดแวร์ทุกตัวมีการทำงานที่สนับสนุนทุก ๆ ฟังก์ชันตามที่แอปพลิเคชันต้องการได้)



รูปที่ 2.3 แสดงหลักการทำงานของไดเร็กดรอ(DirectDraw)

การแสดงผลของไดเร็กดรอ(DirectDraw) จะเป็นการแสดงผลโดยใช้พื้นผิวหน่วยความจำ(Surface เป็นหน่วยความซึ่งเป็นผืนเดียวกันเพื่อเก็บภาพสำหรับการแสดงผล) โดยปกติแล้วจะใช้อยู่ 2 พื้นผิว คือ primary surface เป็นพื้นผิวหน่วยความจำหลักที่เก็บข้อมูลที่กำลังแสดงบนหน้าจอ และ Back buffer surface เป็นพื้นผิวหน่วยความจำชั่วคราวไว้สำหรับเป็น Buffer ในการแสดงผล ดังนั้นการแสดงผลก่อนอื่นจะทำการแก้ไขที่ Back buffer surface ก่อนเมื่อทำการแก้ไขแล้วจึงทำการสลับ(Flip) พื้นผิวหน่วยความจำระหว่าง primary surface กับ Back buffer surface เพื่อนำข้อมูลใน Back buffer surface ขึ้นไปแสดงผลที่หน้าจอ

ไดเร็กอินพุท(DirectInput)

เป็นส่วนเชื่อมต่อกับโปรแกรมประยุกต์(API) สำหรับใช้ในการติดต่ออุปกรณ์ที่เป็นอินพุทของคอมพิวเตอร์ ซึ่งการใช้ไดเร็กอินพุท(DirectInput) นั้นจะต้องคำนึงถึงประเภทของอุปกรณ์อินพุทด้วยถ้าต่างชนิดกันจะต้องจัดการต่างกันแต่ถ้าเป็นอุปกรณ์ประเภทเดียวกันแต่ต่างรุ่นกันก็ไม่จำเป็นต้องเขียนโปรแกรมต่างกัน นอกจากอุปกรณ์นั้นมีความพิเศษที่ต่างจากอุปกรณ์ประเภทเดียวกันก็จะต้องเขียนโปรแกรมจัดการเอง

การจัดการอุปกรณ์ทั้งหมดบนไดเร็กอินพุท(DirectInput) จะใช้วิธีการโพลลิ่ง (polling) ซึ่งเป็นการวนลูปเพื่อตรวจสอบการร้องของบริการจากอุปกรณ์แต่ละตัว เพื่อใช้ในงาน real-time การโพลลิ่งนั้นอาจทำให้ได้รับข้อมูลช้าไปบ้างในกรณีที่ยังวนไปไม่ถึงอุปกรณ์ที่กำลังมีการส่งข้อมูล ส่วนอุปกรณ์บางอย่างก็สามารถใช้วิธีการส่งข้อความ(message) ผ่านทางฟังก์ชันของไดเร็กอินพุท(DirectInput) ได้ ในกรณีนี้เราก็สามารถใช้ทั้งโพลลิ่งร่วมกับการส่งข้อความก็ได้

2.6 การใช้งานไดเร็กเอ็ก

ไดเร็กเอ็ก(DirectX) จริง ๆ แล้วจะอยู่ในรูปของไลบารีต่าง ๆ ที่ทางไมโครซอฟท์ ได้ทำการพัฒนาขึ้นมาและรวมอยู่ในรูปของชุดพัฒนาสำหรับไดเร็กเอ็ก(DirectX SDK) ซึ่งถ้าต้องการนำมาใช้งานก็จะต้องมีการสร้างฟังก์ชันการทำงานที่ต้องการก่อนโดยในฟังก์ชันนั้นจะไปทำการเรียกใช้ไดเร็กเอ็ก(DirectX) ไลบารีที่ต้องการใช้งานอีกทีหนึ่ง และในโปรเจคนี้ก็ได้ทำการพัฒนาชุดของฟังก์ชันที่มีชื่อว่า "**GDK**" ขึ้นมาเพื่อรองรับการทำงานของโปรแกรมช่วยในการสร้างและพัฒนาเกมที่ได้ทำการเขียนขึ้น

บทที่ 3
โปรแกรมสร้างและพัฒนาเกมสำเร็จรูป

โปรแกรมสร้างและพัฒนาเกมสำเร็จรูป เป็นเครื่องมือที่ใช้ในการพัฒนาเกมในรูปแบบต่าง ๆ ที่ใช้ลักษณะพื้นฐานด้านเหตุการณ์ต่าง ๆในการดำเนินเกม เช่น เกมประเภทผจญภัย (Adventure), เกมยานยิง (Shooting) เป็นต้น โดยตัวของ โปรแกรมสร้างและพัฒนาเกมสำเร็จรูป ที่ได้ทำการพัฒนาขึ้นมานี้จะรองรับการทำเกมประเภทยานยิงเพียงอย่างเดียวแต่ตัวโปรแกรมสร้างและพัฒนาเกมสำเร็จรูปตัวนี้ก็ได้ใช้หลักการพัฒนาตามแบบการเขียนโปรแกรมเชิงวัตถุ เพื่อรองรับการพัฒนาเพิ่มเติมให้สามารถที่จะใช้ในการสร้างเกมประเภทอื่น ๆ ได้อีก ตามที่ได้ทำการกำหนดโครงสร้างต่าง ๆ รองรับไว้อยู่พอสมควร

สำหรับตัวโปรแกรมสร้างและพัฒนาเกมสำเร็จรูปนี้จะช่วยให้ผู้ใช้สามารถทำการสร้างเกมตามที่ผู้ใช้ต้องการ โดยที่ผู้ใช้งานไม่จำเป็นต้องมีความสามารถในการเขียนโปรแกรมเลย เพียงอาศัยความเข้าใจหลักการกำหนดเหตุการณ์ (event) เพื่อดำเนินเนื้อหาของเกมก็เพียงพอแล้วที่จะทำการสร้างเกม เช่น การกำหนดเหตุการณ์ของตัวละครของผู้เล่นให้มีความสามารถในการ เคลื่อนที่ เก็บอุปกรณ์เสริม (option) หรือ การโจมตีศัตรู เป็นต้น แต่เหตุการณ์ที่ได้ยกตัวอย่างไปนั้นเป็นเพียงเหตุการณ์พื้นฐาน เพราะนอกจากผู้ใช้ต้องเข้าใจถึงเหตุการณ์เหล่านี้แล้วผู้ใช้ยังต้องคำนึงถึงเหตุการณ์อื่น ๆ ที่จะใช้ในการดำเนินเกมอีกด้วย เช่น การตายของผู้เล่นจะทำให้เกิดเหตุการณ์ให้ทำการจบเกม หรือ การเคลื่อนที่ไปจนถึงจุดสิ้นสุดของฉากจะต้องทำให้มีเหตุการณ์ต่อเนื่องโดยการสร้างหัวหน้าประจำฉากขึ้นมาต่อสู้กับผู้เล่น เป็นต้น

3.1 หน้าที่ของ โปรแกรมสร้างและพัฒนาเกมสำเร็จรูป

โปรแกรมสร้างและพัฒนาเกมสำเร็จรูปจะทำหน้าที่เป็นตัวสร้างข้อมูลของเกมที่ผู้ใช้ทำการสร้างตามความต้องการของผู้ใช้แล้วนำข้อมูลเหล่านั้นเก็บข้อมูลในรูปแบบที่กำหนดไว้ตามที่ตัวเกมรันไทม์ต้องการ เพื่อที่จะใช้เป็นข้อมูลดิบของตัวเกมรันไทม์นำไปใช้ในการสร้างเกม ควบคุมการเคลื่อนไหว เหตุการณ์ต่าง ๆ ในขณะดำเนินเกม ตามที่ผู้ใช้ได้กำหนดสร้างขึ้นมาในตัว โปรแกรมสร้างและพัฒนาเกมสำเร็จรูป ข้อมูลต่าง ๆ ที่ถูกสร้างขึ้นมานี้ก็จะทำการเก็บอยู่ในรูปแบบของแฟ้มข้อมูลแต่ละแฟ้มที่ทำหน้าที่แยกกันโดยใช้โครงสร้างการเก็บข้อมูลตามที่ได้ตกลงไว้ เช่น

แฟ้มข้อมูล gamesystem.env ทำหน้าที่ในการเก็บรายละเอียดพื้นฐานของเกม เช่น เกมที่สร้างนี้มีชื่อว่าอะไร , จำนวนของฉาก (Map)ในเกม , จำนวนของวัตถุต่าง ๆ ในเกม เป็นต้นรวมไปถึงบอกชื่อแฟ้มข้อมูลที่เป็นแฟ้มข้อมูลหลักหรือแฟ้มข้อมูลพื้นฐานอื่นของเกม เช่น แฟ้มข้อมูลที่บอกข้อมูลพื้นฐานของฉากต่าง ๆ ในเกม เป็นต้น ซึ่งโดยปกติแล้วจะเป็นแฟ้มข้อมูลที่มีนามสกุลเป็น .gbl (*.gbl) แต่โดยค่าพื้นฐานแล้วจะเป็น Maps.gbl , Statics.gbl , Dynamics.gbl , Players.gbl , Event.gbl เป็นต้น

แฟ้มข้อมูล gameshooting.env ทำหน้าที่เก็บรายละเอียดของเกมในส่วนที่เป็นข้อมูลพื้นฐานของเกมประเภทยานยิงซึ่งออกแบบไว้สำหรับรองรับการพัฒนาต่อเพื่อให้ตัว โปรแกรมสร้างและพัฒนาเกม

สำเร็จรูป นี้สามารถที่จะพัฒนาเพื่อให้สร้างเกมประเภทอื่น ๆ ได้โดยการเพิ่มรายละเอียดเข้าไปโดยอาจจะมีการเพิ่มตัว gameadventure.env เพื่อให้สามารถสร้างเกมแนวผจญภัยได้เป็นต้น

แฟ้มข้อมูล Maps.gbl , Statics.gbl , Dynamic.gbl , Players.gbl , Event.gbl ก็เป็นส่วนที่บอกรายละเอียดพื้นฐานของส่วนต่าง ๆ ในเกมเช่น ไดเรกทอรีที่เก็บฉาก , รายชื่อของแฟ้มข้อมูลที่ต้องทำการเปิดเพื่อใช้ในการสร้างวัตถุประเภทที่ไม่สามารถเคลื่อนที่ได้ , วัตถุที่สามารถเคลื่อนที่ได้ เป็นต้น

แฟ้มข้อมูลที่มีนามสกุล .Map (*.Map) เป็นแฟ้มข้อมูลที่ใช้เก็บข้อมูลของฉากแต่ละฉากที่ใช้ภายในเกมนั้น

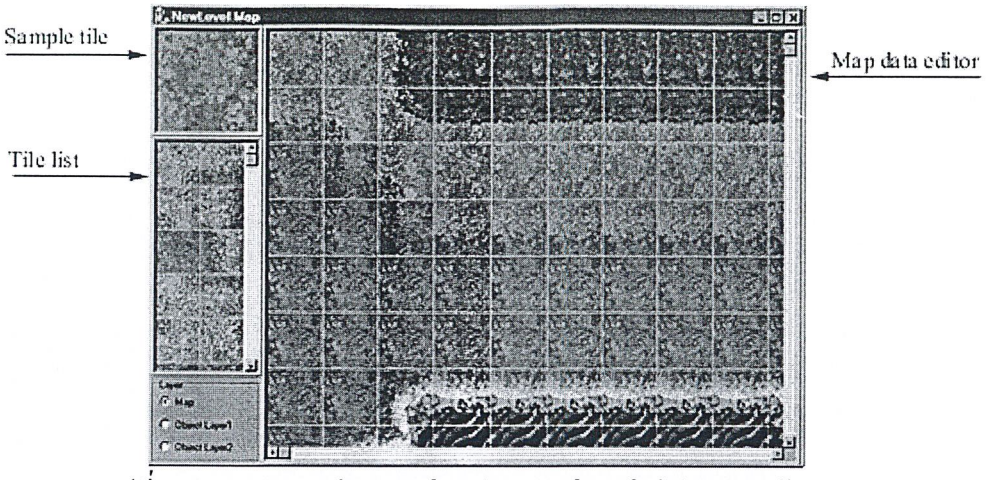
แฟ้มข้อมูลที่มีนามสกุล .Obj (*.Obj) เป็นแฟ้มข้อมูลที่ใช้เก็บข้อมูลของวัตถุต่าง ๆ ที่ใช้ภายในเกม (1 วัตถุต่อแฟ้มข้อมูล 1 แฟ้ม) โดยไม่มีการแยกประเภทว่าเป็นวัตถุประเภทที่ไม่สามารถเคลื่อนที่ได้ , วัตถุที่สามารถเคลื่อนที่ได้ โดยวัตถุแต่ละประเภทนั้นจะอยู่ในไดเรกทอรีเฉพาะของตนเองซึ่งจะบอกว่าเป็นไดเรกทอรีใดในตัวโครงสร้างของแฟ้มข้อมูล Static.gbl , Dynamic.gbl ทั้งนี้ก็เพื่อที่จะเพิ่มความยืดหยุ่นในการเพิ่มประเภทของวัตถุต่าง ๆ ลงไปในตัวเกมในกรณีที่ต้องการในภายหลังสำหรับเกมประเภทอื่น ๆ

นอกจากนั้นก็ยังมีแฟ้มข้อมูลพิเศษอื่น ๆ อีกเช่นแฟ้มข้อมูลนามสกุล .gdk (*.gdk) เป็นแฟ้มข้อมูลที่ใช้เก็บข้อมูลพื้นฐานที่ใช้ในตัวของโปรแกรมสร้างและพัฒนาเกมสำเร็จรูปเท่านั้นแต่ไม่ได้ใช้กับตัวเกมรันไทม์เป็นต้น

รูปที่3.1 แสดงเมนูหลักของโปรแกรมสร้างและพัฒนาเกมสำเร็จรูป

3.2 ส่วนประกอบของ โปรแกรมสร้างและพัฒนาเกมสำเร็จรูป

1. โปรแกรมสร้างฉาก (MapEditor) เป็นส่วนที่ทำหน้าที่ในการสร้างฉากต่าง ๆ ภายในเกม และอาจกล่าวได้ว่าส่วนของโปรแกรมสร้างฉากนี้เป็นส่วนหลักของ โปรแกรมสร้างและพัฒนาเกมสำเร็จรูป เพราะเป็นส่วนที่ทำหน้าที่ในการเชื่อมต่อกับตัวโปรแกรมสร้างตัวอื่นๆด้วย



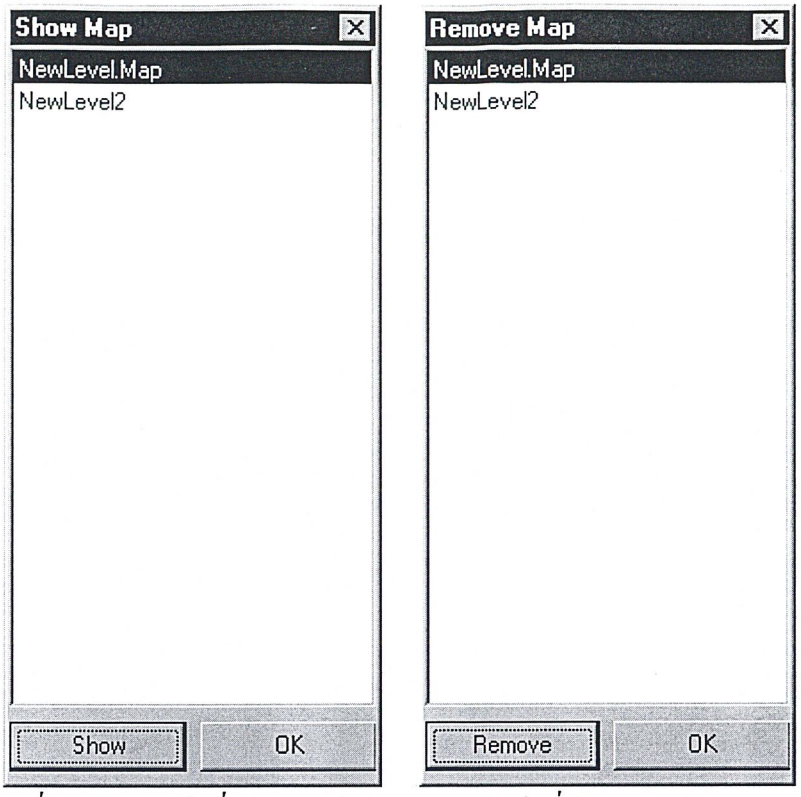
รูปที่3.2 แสดงรายละเอียดของส่วนประกอบต่าง ๆ ในโปรแกรมสร้างฉาก

ไทล์ตัวอย่าง – เป็นตัวที่แสดงภาพไทล์ที่ผู้ใช้ทำการเลือกอยู่ในปัจจุบัน

ไทล์ลิสต์ – เป็นตัวที่ทำหน้าที่แสดงภาพไทล์ทั้งหมดที่ใช้ในฉากที่จะทำการสร้างขณะนั้นซึ่งจะมีจำนวนเท่าใดนั้นก็ขึ้นอยู่กับตัวแฟ้มข้อมูลไทล์ (*.tle) ที่ได้ทำการโหลดเข้าไปขณะทำการขอเปิด ฉากใหม่ขึ้นมา

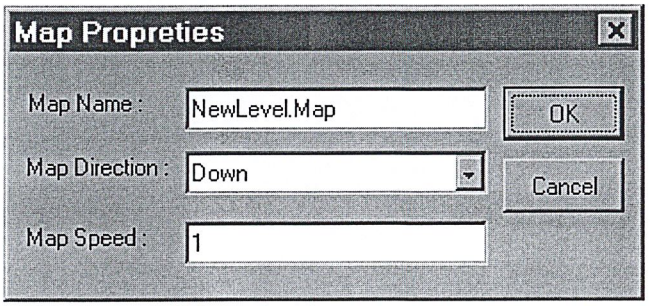
ส่วนแสดงผลฉาก – เป็นตัวที่ทำหน้าที่ในการแสดงภาพของฉากที่ได้ทำการสร้างไปเรียบร้อยแล้ว และถ้าต้องการสร้างฉากอย่างไรก็ทำการเลือกไทล์จากไทล์ลิสต์ ขึ้นมาแล้วนำมาวางลงบนตัวของส่วนแสดงผลฉากแล้วโปรแกรมก็จะทำการนำหมายเลขของไทล์ นั้นไปเก็บยังที่เก็บข้อมูลของฉากพร้อมกับทำการแสดงภาพขึ้นมาบนหน้าจอ

นอกจากนี้ยังมีส่วนที่เป็น ฟอร์มแสดงฉาก (Show map form) และ ฟอร์มที่ใช้ลบฉากออกจากเกม (Remove Map form) ที่ทำหน้าที่ในการเรียกฉากที่ต้องการขึ้นมาดู หรือทำการลบฉากที่ไม่ต้องการออกจากเกม



รูปที่3.3 แสดงฟอร์มที่ใช้ในการแสดงฉาก และ ฟอร์มที่ใช้ในการลบฉากจากเกม

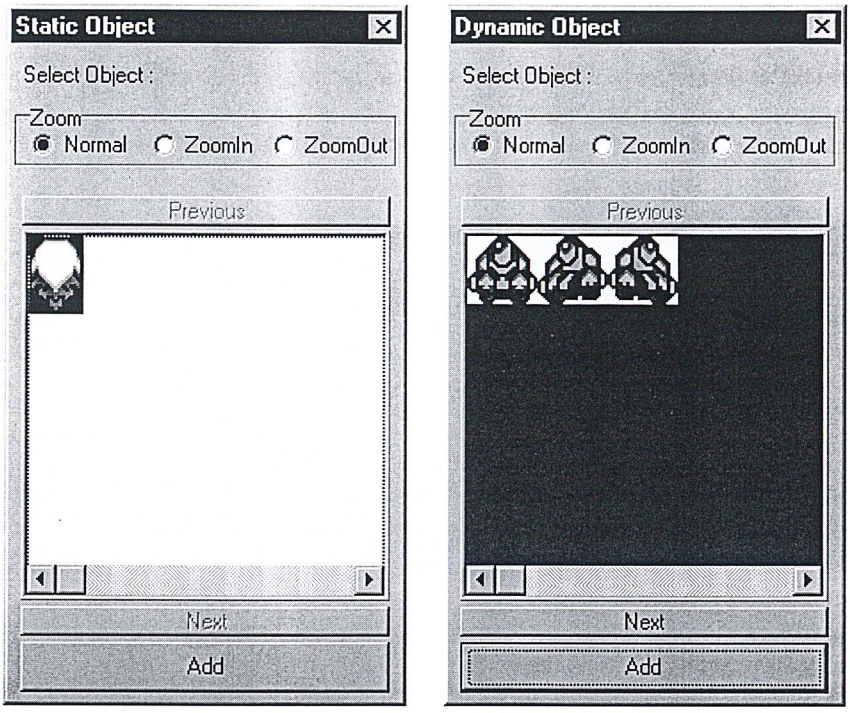
นอกจากนั้นในส่วนของโปรแกรมสร้างฉากก็จะมีส่วนที่ใช้ในการกำหนดรายละเอียดของ ฉาก (MapProperties) ที่ได้ทำการสร้างขึ้นซึ่งจะมีลักษณะดังภาพ



รูปที่3.4 แสดงลักษณะของฟอร์มที่ใช้ในการแก้ไขคุณสมบัติของฉาก

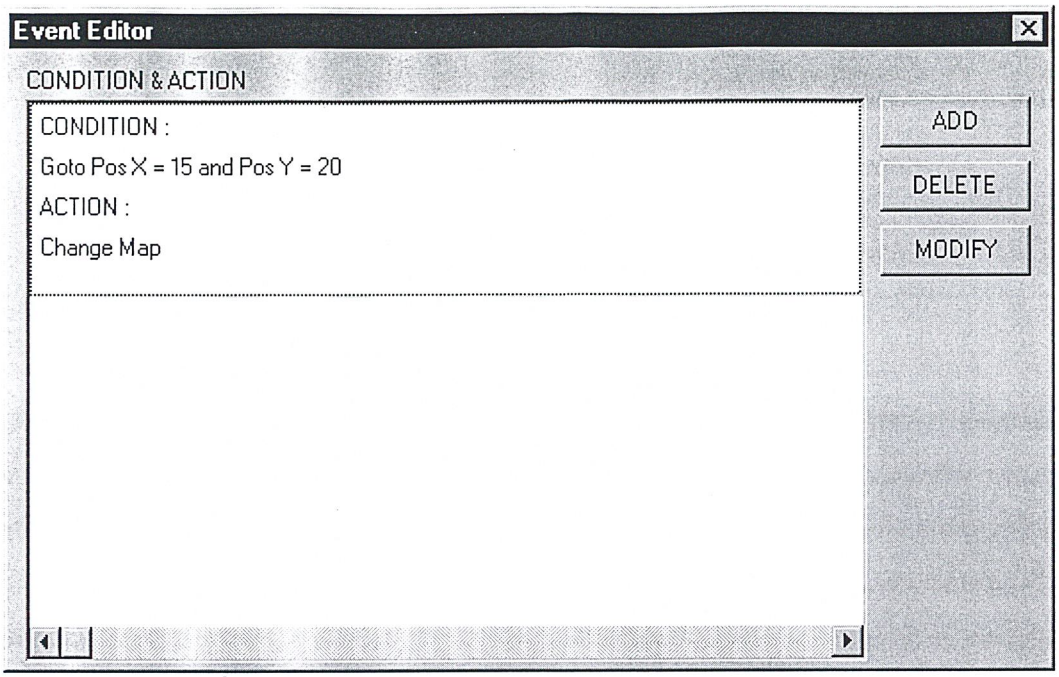
2. โปรแกรมสร้างวัตถุจะเป็นส่วนที่ทำหน้าที่ในการสร้างวัตถุต่าง ๆ ในตัวเกมตามที่ผู้ใช้ต้องการ โดยทำการเลือกไทล์ (รูปของวัตถุ) ที่ต้องการแล้วนำไปวางบนตัวแสดงผลของฉากตัวโปรแกรมก็จะทำการสร้างและเก็บรายละเอียดของวัตถุนั้นตามที่ผู้ใช้ต้องการในรูปของแฟ้มข้อมูลวัตถุ (*.Obj) ซึ่งจะเป็นวัตถุประเภทใดนั้นก็จะขึ้นอยู่กับตัวของโปรแกรมสร้างวัตถุที่ผู้ใช้ได้เลือกมาซึ่งก็มีอยู่ 2 ประเภท คือ

วัตถุที่ไม่สามารถเคลื่อนไหวได้ , วัตถุที่สามารถที่เคลื่อนที่ได้ และตัวของโปรแกรมสร้างวัตถุก็จะมีลักษณะดังภาพ

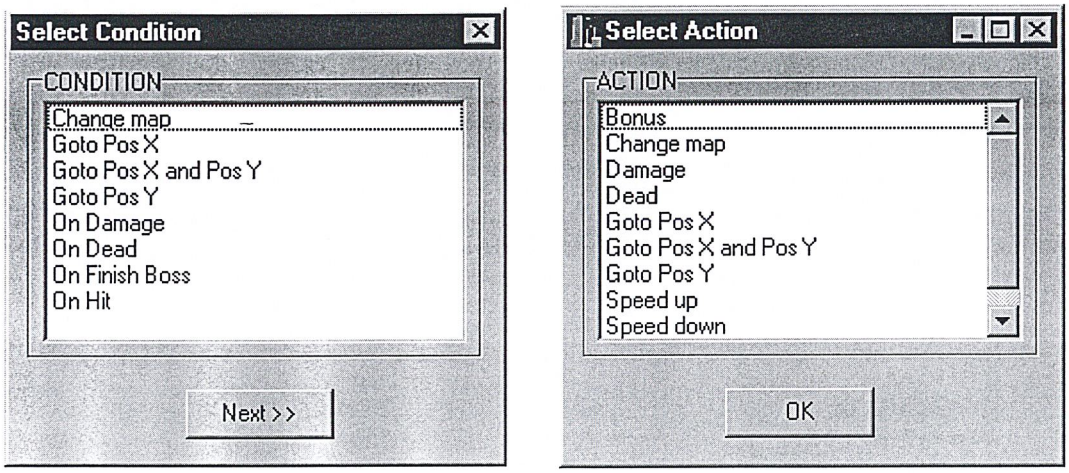


รูปที่3.5 แสดงลักษณะของโปรแกรมสร้างวัตถุแบบไม่สามารถเคลื่อนไหวได้ และ แบบเคลื่อนไหวได้

3. โปรแกรมสร้างเหตุการณ์ (Event Editor) เป็นโปรแกรมที่ใช้ในการสร้างเหตุการณ์และเงื่อนไขต่าง ๆที่ใช้ในการดำเนินเกมโดยจะเป็นลักษณะของเหตุการณ์แบบต่าง ๆที่มักมีการใช้บ่อยในเกมประเภทยานยิง ซึ่งจะถูกเตรียมไว้ในลักษณะของชุดเหตุการณ์แล้วนำมาแสดงผลเพื่อให้เลือกรูปแบบเหตุการณ์ที่ต้องการ โดยจะมีลักษณะดังนี้



รูปที่3.6 แสดงโปรแกรมสร้างเหตุการณ์ในส่วนของเมนูหลัก



รูปที่3.7 แสดงโปรแกรมสร้างเหตุการณ์ในส่วนของการเลือกเงื่อนไข และ การกระทำ

บทที่ 4

Game Runtime

4.1 แนวคิดในการทำงานของ Game Runtime

สำหรับแนวคิดในการทำงานของเกมคือการนำข้อมูลต่างๆที่ผู้สร้างได้ทำการสร้างขึ้นมาจากโปรแกรมสร้างและพัฒนาเกมสำเร็จรูปแล้วนั้นมาใช้ในการทำงานของเกมโดยให้เกมนั้นเป็นเกมที่ถูกต้องตามความต้องการของผู้สร้าง โดยผู้สร้างนั้นไม่จำเป็นต้องรู้ในรายละเอียดการทำงานของเกมแต่ประการใด ทำให้ไม่ยุ่งยากต่อการสร้างและพัฒนาเกม ซึ่งมีผลทำให้ขั้นตอนในการสร้างและพัฒนาเกมในทุกๆส่วนรวดเร็ว ลดความซ้ำซ้อนของการทำงานโดยไม่จำเป็น เมื่อผู้สร้างได้สร้างข้อมูลต่างๆตามที่ต้องการแล้วก็จะนำข้อมูลเหล่านั้นมาใช้ในการทำงานของ Game Runtime ซึ่งเป็นโปรแกรมอีกส่วนหนึ่งที่อยู่ในโปรแกรมสร้างและพัฒนาเกมสำเร็จรูปนี้ และถือได้ว่าเป็นส่วนที่ทำให้ผู้สร้างได้เห็นถึงผลของการสร้างอย่างแท้จริง โดยข้อมูลที่นำมาใช้กับ Game Runtime นั้นจะต้องมีรูปแบบ และโครงสร้างที่ตรงตามที่โปรแกรมกำหนดไว้เท่านั้น ซึ่งหมายความว่าการนำข้อมูลอื่นๆเข้ามาใช้ใน Game Runtime จะไม่เป็นผลและอาจทำให้เกิดข้อผิดพลาดได้

หน้าที่ที่สำคัญของ Game Runtime

นำข้อมูลที่ได้จากการสร้างทั้งหมดเข้ามาใช้ในการทำงานของเกม

ทำการจำลองการทำงานของเกมให้เป็นไปตามรูปแบบที่กำหนดไว้

ทำการโต้ตอบกับผู้เล่นในลักษณะที่กำหนดไว้

ทำการแสดงผลในลักษณะที่กำหนดไว้

ขั้นตอนการทำงานของ Game Runtime

ในการทำงานของ Game Runtime นั้นได้ใช้คลาส GGOS และ GGShooting เพื่อควบคุมการทำงานทั้งหมดให้เป็นไปตามลักษณะที่ต้องการโดยมี Code ดังนี้

```
//---------------------------------------------------------------------------
#pragma hdrstop

#include "GGOS.h"

//---------------------------------------------------------------------------
#pragma package(smart_init)
//---------------------------------------------------------------------------
char DEBUGSTR[]="Debugging";
```

```
GGOS *refGGOS;
//---------------------------------------------------------------------------
GGOS::GGOS(void)
{
    b_UseForTool=true;
}
//---------------------------------------------------------------------------
GGOS::GGOS(void *hInst,void *hWnd)
{
   m_hWnd=hWnd;
   m_hInst=hInst;
   m_MapCount=0;
   m_DynamicObjectCount=0;
   m_StaticObjectCount=0;
   m_CurrentDir = new char[255];
   GetCurrentDirectory(255,m_CurrentDir);
   m_CurrentMap=0;
   b_UseForTool=false;
}
//---------------------------------------------------------------------------
GGOS::~GGOS()
{
   if(!b_UseForTool)
   {
      delete m_Screen;
      delete[] m_CurrentDir;
      delete[] m_Mapfile;
      delete[] m_Playerfile;
      delete[] m_DynamicObjectfile;
      delete[] m_StaticObjectfile;
   }
}
//---------------------------------------------------------------------------
```

```
bool GGOS::LoadGameSystemEnv(const char *filename)
{ // เป็นฟังก์ชันในการ load ข้อมูลเริ่มต้นที่จำเป็นต่อการทำงานของเกม
   TMemoryStream *mem1 = new TMemoryStream;
   mem1->LoadFromFile(filename);
   if(mem1->Size !=sizeof(GAMESYSTEMENV)) return false;
   mem1->Read(&GameSystemEnv,sizeof(GAMESYSTEMENV));
   delete mem1;
   return true;
}
//---------------------------------------------------------------------------
bool GGOS::InitialGameSystem()
{ // เป็นฟังก์ชันในการสร้างและกำหนดค่าเริ่มต้นต่างๆที่ใช้ในการทำงานของเกม
MakeScreen(GameSystemEnv.gseScreenWidth,GameSystemEnv.gseScreenHeight,GameSystemEnv.gs
eScreenBPP);
   // GGOS use 1 palette/screen/game
   m_Screen->LoadPalette(((AnsiString)GameSystemEnv.gsePaletteFilename).c_str());
   SetWindow(0,0,GameSystemEnv.gseScreenWidth,GameSystemEnv.gseScreenHeight);
   if(!LoadMapFile(((AnsiString)GameSystemEnv.gseGlobalMapFilename).c_str())) return false;
   if(!LoadPlayerFile(GameSystemEnv.gsePlayerFilename)) return false;
   if(!LoadDynamicObjectFile(((AnsiString)GameSystemEnv.gseGlobalDynaObjFilename).c_str()))
return false;
   // This GGOS must load Static Object
   if(!LoadStaticObjectFile(((AnsiString)GameSystemEnv.gseGlobalStaticObjFilename).c_str())) return
false;
   m_CurrentMapDynamicObjectCount=0;
   m_CurrentMapStaticObjectCount=0;
   //SetCurrentMap(0);
}
//---------------------------------------------------------------------------
bool GGOS::LoadMapFile(const char *mapfile)
{ //เป็นฟังก์ชันที่ใช้ในการ load ไฟล์ฉากและแผนที่ของเกมทั้งหมด
   TMemoryStream *mem1 = new TMemoryStream;
   mem1->LoadFromFile(mapfile);
   if(mem1->Size > sizeof(GLOBALMAPHEADER))
```

```
    {
      mem1->Seek(0,soFromBeginning);
      mem1->Read(&gmHeader,sizeof(GLOBALMAPHEADER));
      m_MapDirectory = AnsiString(m_CurrentDir)+"\\"+(AnsiString)gmHeader.gmDirectory+"\\"; //
Last "\\" is bug if gmDirectory is NULL !!
      int size=mem1->Size-sizeof(GLOBALMAPHEADER);
      char *m_AllMapFileNames = new char[size];
      mem1->Read(m_AllMapFileNames,size);
      if(!MakeMap(m_AllMapFileNames,gmHeader.gmNumbers)) return false;
      delete mem1;
      return true;
    } else
    {
    delete mem1;
    return false;
    }
}
//---------------------------------------------------------------------------
bool GGOS::MakeMap(char *allmapfile,int mapcount)
{ //เป็นฟังก์ชันที่ใช้ในการสร้าง GDXMap Object
    if(mapcount<=0) return false;
    int start=1;
    int stop=0;
    m_Mapfile = new AnsiString[mapcount];
    AnsiString allmapfiles(allmapfile);
    for(int i=0;i<mapcount;i++)
    {
      stop = allmapfiles.Pos(" ");
      m_Mapfile[i] = allmapfiles.SubString(start,stop-1);
      m_Mapfile[i].Insert(m_MapDirectory,0);
      allmapfiles.Delete(start,stop);
    }
    m_MapCount=mapcount;
    m_Map = new GMap*[mapcount];
```

```
    for (int i=0;i<mapcount;i++)
    {
      m_Map[i] = new GMap(m_Screen);
      if(!m_Map[i]->LoadFromFile(m_Mapfile[i].c_str())) return false;
    }
    return true;
}
//---------------------------------------------------------------------------
void GGOS::ReleaseAllMap()
{ //เป็นฟังก์ชันที่ใช้ในการ delete ทุก Map Object
    if(m_MapCount==0) return;
    for(int i=0;i<m_MapCount;i++)
    {
      delete m_Map[i];
    }
}
//---------------------------------------------------------------------------
bool GGOS::LoadDynamicObjectFile(const char *objfile)
{ //เป็นฟังก์ชันที่ใช้ในการ load ไฟล์ของวัตถุที่สามารถเคลื่อนที่ได้ (DynamicObject)
    TMemoryStream *mem1 = new TMemoryStream;
    mem1->LoadFromFile(objfile);
    if(mem1->Size > sizeof(GLOBALOBJECTHEADER))
    {
      mem1->Seek(0,soFromBeginning);
      mem1->Read(&goDynamic,sizeof(GLOBALOBJECTHEADER));
      m_DynamicObjDirectory =
AnsiString(m_CurrentDir)+"\\"+(AnsiString)goDynamic.goDirectory+"\\"; // Last "\\" is bug if
goDirectory is NULL !!
      int size=mem1->Size-sizeof(GLOBALOBJECTHEADER);
      char *m_AllObjectFileNames = new char[size];
      mem1->Read(m_AllObjectFileNames,size);
      if(!MakeDynamicObject(m_AllObjectFileNames,goDynamic.goNumbers)) return false;
      delete mem1;
      return true;
```

```
    } else
    {
    delete mem1;
    return false;
    }
}
//---------------------------------------------------------------------------
bool GGOS::MakeDynamicObject(char *allobjfile,int objcount)
{ //เป็นฟังก์ชันที่ใช้ในการสร้าง GDynamicObject Object
   if(objcount<=0) return false;
   int start=1;
   int stop=0;
   m_DynamicObjectCount=objcount;
   m_DynamicObjectfile = new AnsiString[objcount];
   AnsiString allobjfiles(allobjfile);
   for(int i=0;i<objcount;i++)
   {
      stop = allobjfiles.Pos(" ");
      m_DynamicObjectfile[i] = allobjfiles.SubString(start,stop-1);
      m_DynamicObjectfile[i].Insert(m_DynamicObjDirectory,0);
      allobjfiles.Delete(start,stop);
   }
   //*** Use for only GDynamicObject ***
   m_DynamicObject = new GDynamicObject*[objcount];
   int mapid;
   for (int i=0;i<objcount;i++)
   {
      m_DynamicObject[i] = new GDynamicObject(this,m_Screen);
      if(!m_DynamicObject[i]->LoadFromFile(m_DynamicObjectfile[i].c_str())) return false;
      if(m_DynamicObject[i]->GetMapID()==-1) mapid=0; else mapid=m_DynamicObject[i]->
GetMapID();
      m_DynamicObject[i]->SetDefault(m_Screen->GetBack(),m_Map[mapid],&m_Window);
      if(!m_DynamicObject[i]->InitialChildObject()) return false;
   }
```

```
    return true;
}
//---------------------------------------------------------------------------
void GGOS::ReleaseAllDynamicObject()
{ //เป็นฟังก์ชันที่ใช้ในการ delete GDynamicObject ทุก Object
   if(m_DynamicObjectCount==0) return;
   for(int i=0;i<m_DynamicObjectCount;i++)
   {
      delete m_DynamicObject[i];
   }
}
//---------------------------------------------------------------------------
void GGOS::Update()
{
   UpdateInput();
   UpdateMap();
   UpdatePlayer();
   UpdateObjects();
   m_Screen->Flip();
}
//---------------------------------------------------------------------------
void GGOS::ShowObject(GObject *object)
{ //เป็นฟังก์ชันที่ใช้ในการแสดงผล Object
   if((m_CurrentMap==object->GetMapID())||(object->GetMapID()==-1))
   {  // Show if Current Map is the same as object map
      // or object map is -1 (all map)
      object->Show();
   }
}
//---------------------------------------------------------------------------
bool GGOS::LoadPlayerFile(const char *objfile)
{ //เป็นฟังก์ชันที่ใช้ในการ load Player ไฟล์
   TMemoryStream *mem1 = new TMemoryStream;
   mem1->LoadFromFile(objfile);
```

```
    if(mem1->Size > sizeof(GLOBALOBJECTHEADER))
    {
      mem1->Seek(0,soFromBeginning);
      mem1->Read(&goPlayer,sizeof(GLOBALOBJECTHEADER));
      m_PlayerDirectory = AnsiString(m_CurrentDir)+"\\"+(AnsiString)goPlayer.goDirectory+"\\"; //
  Last "\\" is bug if goDirectory is NULL !!
      int size=mem1->Size-sizeof(GLOBALOBJECTHEADER);
      char *m_AllPlayerFileNames = new char[size];
      mem1->Read(m_AllPlayerFileNames,size);
      if(!MakePlayerObject(m_AllPlayerFileNames,goPlayer.goNumbers)) return false;
      delete mem1;
      return true;
    } else
    {
    delete mem1;
    return false;
    }
  }
  //---------------------------------------------------------------------------
  bool GGOS::MakePlayerObject(char *allobjfile,int objcount)
  { //เป็นฟังก์ชันที่ใช้ในการสร้าง GPlayer Object
    if(objcount<=0) return false;
    int start=1;
    int stop=0;
    m_PlayerCount=objcount;
    m_Playerfile = new AnsiString[objcount];
    AnsiString allobjfiles(allobjfile);
    for(int i=0;i<objcount;i++)
    {
      stop = allobjfiles.Pos(" ");
      m_Playerfile[i] = allobjfiles.SubString(start,stop-1);
      m_Playerfile[i].Insert(m_PlayerDirectory,0);
      allobjfiles.Delete(start,stop);
    }
```

```
    //*** Use for only GPlayer ***
    m_Player = new GPlayer*[objcount];
    int mapid;
    for (int i=0;i<objcount;i++)
    {
      m_Player[i] = new GPlayer(this,m_Screen);
      if(!m_Player[i]->LoadFromFile(m_Playerfile[i].c_str())) return false;
      if(m_Player[i]->GetMapID()==-1) mapid=0; else mapid=m_Player[i]->GetMapID();
      m_Player[i]->SetDefault(m_Screen->GetBack(),m_Map[mapid],&m_Window);
      if(!m_Player[i]->InitialChildObject()) return false;
    }
    return true;
}
//---------------------------------------------------------------------------
void GGOS::ReleaseAllPlayerObject()
{ //เป็นฟังก์ชันที่ใช้ในการ delete GPlayer Object ทุกตัว
    if(m_PlayerCount==0) return;
    for(int i=0;i<m_PlayerCount;i++)
    {
      delete m_Player[i];
    }
}
//---------------------------------------------------------------------------
bool GGOS::LoadStaticObjectFile(const char *objfile)
{ //เป็นฟังก์ชันที่ใช้ในการ load วัตถุที่ไม่สามารถเคลื่อนที่ได้ GStaticObject
    TMemoryStream *mem1 = new TMemoryStream;
    mem1->LoadFromFile(objfile);
    if(mem1->Size > sizeof(GLOBALOBJECTHEADER))
    {
      mem1->Seek(0,soFromBeginning);
      mem1->Read(&goStatic,sizeof(GLOBALOBJECTHEADER));
      m_StaticObjDirectory = AnsiString(m_CurrentDir)+"\\"+(AnsiString)goStatic.goDirectory+"\\"; //
Last "\\" is bug if goDirectory is NULL !!
      int size=mem1->Size-sizeof(GLOBALOBJECTHEADER);
```

```
        char *m_AllObjectFileNames = new char[size];
        mem1->Read(m_AllObjectFileNames,size);
        MakeStaticObject(m_AllObjectFileNames,goStatic.goNumbers);
        delete mem1;
        return true;
    } else
    {
    delete mem1;
    return false;
    }
}
//---------------------------------------------------------------------------
bool GGOS::MakeStaticObject(char *allobjfile,int objcount)
{ //เป็นฟังก์ชันที่ใช้ในการสร้าง GStaticObject object
    if(objcount<=0) return false;
    int start=1;
    int stop=0;
    m_StaticObjectCount=objcount;
    m_StaticObjectfile = new AnsiString[objcount];
    AnsiString allobjfiles(allobjfile);
    for(int i=0;i<objcount;i++)
    {
        stop = allobjfiles.Pos(" ");
        m_StaticObjectfile[i] = allobjfiles.SubString(start,stop-1);
        m_StaticObjectfile[i].Insert(m_StaticObjDirectory,0);
        allobjfiles.Delete(start,stop);
    }
    //*** Use for only GDynamicObject ***
    m_StaticObject = new GStaticObject*[objcount];
    int mapid;
    for (int i=0;i<objcount;i++)
    {
        m_StaticObject[i] = new GStaticObject(this,m_Screen);
        if(!m_StaticObject[i]->LoadFromFile(m_StaticObjectfile[i].c_str())) return false;
```

```
    if(m_StaticObject[i]->GetMapID()==-1) mapid=0; else mapid=m_StaticObject[i]->GetMapID();
    m_StaticObject[i]->SetDefault(m_Screen->GetBack(),m_Map[mapid],&m_Window);
    if(!m_StaticObject[i]->InitialChildObject()) return false;
  }
  return true;
}
//---------------------------------------------------------------------------
void GGOS::ReleaseAllStaticObject()
{ //เป็นฟังก์ชันที่ใช้ในการ delete GStaticObject object ทุกตัว
  if(m_StaticObjectCount==0) return;
  for(int i=0;i<m_StaticObjectCount;i++)
  {
    delete m_StaticObject[i];
  }
}
//---------------------------------------------------------------------------
int GGOS::HitObject(GObject *gobject)
{ //เป็นฟังก์ชันที่ใช้ในการตรวจสอบว่าวัตถุที่ให้มานั้นชนกับวัตถุใดในฉากหรือไม่
  // Check Hit only Object in Current Map
  for(int i=0;i<m_CurrentMapDynamicObjectCount;i++)
  {
    // Check All Dynamic Objects
    if(gobject->Hit((GObject*)m_CurrentMapDynamicObject[i]))
    {
      return m_CurrentMapDynamicObject[i]->GetID();
    }
  }
  for(int i=0;i<m_CurrentMapStaticObjectCount;i++)
  {
    // Check All Static Objects
    if(gobject->Hit((GObject*)m_CurrentMapStaticObject[i]))
    {
      return m_CurrentMapStaticObject[i]->GetID();
    }
```

```
    }
    return (-1);
}
//---------------------------------------------------------------------------
void GGOS::Run()
{ //เป็นฟังก์ชันที่ใช้สั่งให้ Object ทุกๆ Object ทำงานโดย Function GObject::Run()
    // Run All Dynamic Object
    for(int i=0;i<m_DynamicObjectCount;i++)
    {
        m_DynamicObject[i]->Run();
    }
    // Run all Static Object
    for(int j=0;j<m_StaticObjectCount;j++)
    {
        m_StaticObject[j]->Run();
    }
    // Run all Player Object
    for(int k=0;k<m_PlayerCount;k++)
    {
        m_Player[k]->Run();
    }
}
//---------------------------------------------------------------------------
GObject *GGOS::GetObject(int objID)
{ //เป็นฟังก์ชันที่ใช้ในการเอาวัตถุที่มี ID ตามที่กำหนดออกมา
    //Search Object ID in all Object List
    GObject *resultObj;
    // Check for DynamicObject
    for(int i=0;i<m_DynamicObjectCount;i++)
    {
        if(m_DynamicObject[i]->GetID()==objID)
        {
            resultObj = (GObject*)m_DynamicObject[i];
            return resultObj;
```

```
        }
    }
    // Check for StaticObject
    for(int j=0;j<m_StaticObjectCount;j++)
    {
        if(m_StaticObject[j]->GetID()==objID)
        {
            resultObj = (GObject*)m_StaticObject[j];
            return resultObj;
        }
    }
    // Check for Player Object
    for(int k=0;k<m_PlayerCount;k++)
    {
        if(m_Player[k]->GetID()==objID)
        {
            resultObj = (GObject*)m_Player[k];
            return resultObj;
        }
    }
    return NULL; // If not found any Object
}
//---------------------------------------------------------------------------
bool GGOS::SetCurrentMap(int map)
{ //เป็นฟังก์ชันที่ใช้ในการกำหนดฉากปัจจุบัน
    //Set m_Current Map and
    //Make Dynamic Object and Static object List in Current Map
    return(MakeCurrentMapObjectList(map));
}
//---------------------------------------------------------------------------
bool GGOS::MakeCurrentMapObjectList(int map)
{ //เป็นฟังก์ชันที่ใช้ในการสร้าง List ของวัตถุต่างๆที่อยู่ในฉากปัจจุบัน
    if((map>=0)&&(map<m_MapCount))
    {
```

```
m_CurrentMap=map;
int dynamiccount=0;
int staticcount=0;
for(int i=0;i<m_DynamicObjectCount;i++)
{
  if((m_DynamicObject[i]->GetMapID()==map)||(m_DynamicObject[i]->GetMapID()==-1))
  {
    dynamiccount++;
  }
}
for(int i=0;i<m_StaticObjectCount;i++)
{
  if((m_StaticObject[i]->GetMapID()==map)||(m_StaticObject[i]->GetMapID()==-1))
  {
    staticcount++;
  }
}
m_CurrentMapDynamicObject = new GDynamicObject*[dynamiccount];
m_CurrentMapStaticObject = new GStaticObject*[staticcount];
m_CurrentMapDynamicObjectCount = dynamiccount;
m_CurrentMapStaticObjectCount = staticcount;
// Make Dynamic Object list
int count=0;
for(int j=0;j<m_DynamicObjectCount;j++)
{
  if((m_DynamicObject[j]->GetMapID()==map)||(m_DynamicObject[j]->GetMapID()==-1))
  {
    m_CurrentMapDynamicObject[count++] = m_DynamicObject[j];
  }
}

// Make Static Object List
count=0;
for(int j=0;j<m_StaticObjectCount;j++)
```

```
    {
      if((m_StaticObject[j]->GetMapID()==map)||(m_StaticObject[j]->GetMapID()==-1))
      {
        m_CurrentMapStaticObject[count++] = m_StaticObject[j];
      }
    }
    return true;
  } else return false;
}
//---------------------------------------------------------------------------
```

สำหรับ GGShooting Class ได้สืบทอดมาจาก GGOS Class ซึ่งเป็นคลาสพื้นฐานสำหรับการทำงานของเกมทั่วไปแบบไม่เฉพาะเจาะจงรูปแบบของเกม โดย GGShooting Class จะเพิ่มการทำงานของเกมประเภท Shooting ลงไปเพื่อควบคุมการทำงานของเกมให้เป็นไปตามรูปแบบของเกมประเภทยานยิง โดยมี Code ดังนี้

```
//---------------------------------------------------------------------------
#pragma hdrstop

#include "GGShooting.h" _

//---------------------------------------------------------------------------
#pragma package(smart_init)
//---------------------------------------------------------------------------
GGShooting::GGShooting(void)
{
  b_UseForTool=true;
}
//---------------------------------------------------------------------------
GGShooting::GGShooting(void *hInst,void * hWnd) : GGOS(hInst,hWnd)
{
  m_Speed=1;
  b_UseForTool=false;
}
//---------------------------------------------------------------------------
GGShooting::~GGShooting()
```

```
{
}
//--------------------------------------------------------------------
void GGShooting::UpdateInput()
{ //เป็นฟังก์ชันที่ใช้ในการรับและตรวจสอบข้อมูลจาก keyboard รวมทั้งกำหนดว่าจะต้องทำอะไร
  if(GetAsyncKeyState(VK_UP))
  {
    m_Player[0]->MoveUp(m_Player[0]->GetVelY());
    if(m_Player[0]->GetScreenPosY()<0) m_Player[0]->MoveDown(m_Player[0]->GetVelY());
    if(HitObject(m_Player[0])!=-1)
    {
      m_Player[0]->MoveDown(m_Player[0]->GetVelY());
    }
  }
  if(GetAsyncKeyState(VK_DOWN))
  {
    m_Player[0]->MoveDown(m_Player[0]->GetVelY());
    if(m_Player[0]->GetScreenPosY()>m_PlayerBottomLimit) m_Player[0]->MoveUp(m_Player[0]->
GetVelY());
    if(HitObject(m_Player[0])!=-1)
    {
      m_Player[0]->MoveUp(m_Player[0]->GetVelY());
    }
  }
  if(GetAsyncKeyState(VK_LEFT))
  {
    m_Player[0]->SetFrame(1);
    if(m_Player[0]->GetScreenPosX()<128 && m_Map[m_CurrentMap]->GetMapPosX()>0)
    {
      m_Player[0]->MoveLeft(m_Player[0]->GetVelX());
      m_Map[m_CurrentMap]->ScrollLeft(m_Speed);
      if(HitObject(m_Player[0])!=-1)
      {
        m_Player[0]->MoveRight(m_Player[0]->GetVelX());
```

```
            }
        } else if(m_Map[m_CurrentMap]->GetMapPosX()==0)
        {
            m_Player[0]->MoveLeft(m_Player[0]->GetVelX());
            if(HitObject(m_Player[0])!=-1)
            {
                m_Player[0]->MoveRight(m_Player[0]->GetVelX());
            }
            if(m_Player[0]->GetScreenPosX()<0) m_Player[0]->MoveRight(m_Player[0]->GetVelX());
        } else
        {
            m_Player[0]->MoveLeft(m_Player[0]->GetVelX());
            if(HitObject(m_Player[0])!=-1)
            {
                m_Player[0]->MoveRight(m_Player[0]->GetVelX());
            }
        }
    }
    if(GetAsyncKeyState(VK_RIGHT))
    {
        m_Player[0]->SetFrame(2);
        if(m_Player[0]->GetScreenPosX()>412 && m_Player[0]->GetPosX()<m_Map[m_CurrentMap]->
GetMapPixelWidth()-m_Player[0]->m_Tile->GetBlockWidth())
        {
            m_Player[0]->MoveRight(m_Player[0]->GetVelX());
            m_Map[m_CurrentMap]->ScrollRight(m_Speed);
            if(HitObject(m_Player[0])!=-1)
            {
                m_Player[0]->MoveLeft(m_Player[0]->GetVelX());
            }
        } else if(m_Map[m_CurrentMap]->GetMapPosX()==128)
        {
            m_Player[0]->MoveRight(m_Player[0]->GetVelX());
            if(HitObject(m_Player[0])!=-1)
```

```
        {
            m_Player[0]->MoveLeft(m_Player[0]->GetVelX());
        }
        if(m_Player[0]->GetScreenPosX()>m_PlayerRightLimit) m_Player[0]->MoveLeft(m_Player[0]->GetVelX());
    } else
    {
        m_Player[0]->MoveRight(m_Player[0]->GetVelX());
        if(HitObject(m_Player[0])!=-1)
        {
            m_Player[0]->MoveLeft(m_Player[0]->GetVelX());
        }
    }
}
if(GetAsyncKeyState(VK_CONTROL))
{
    // Test ChangeMap by Event
    GRemoteEvent *remoteEvent = new GRemoteEvent(m_Player[0],1,GO_HIT,MP_NONE,NULL);
    m_Player[0]->Notify(remoteEvent);
}
if(GetAsyncKeyState(VK_SPACE))
{
    m_PlayerMissileTick++;
    if(m_PlayerMissileTick>ShootingEnv.sgePlayerMissileTick)
    {
        if(PlayerMissile->m_Child->Run(m_PlayerMissileNow,true,true))
        {
            if(m_PlayerMissileNow<PlayerMissile->GetTotalChild()-1)
            {
                m_PlayerMissileNow++;
            } else m_PlayerMissileNow=0;
        }
        m_PlayerMissileTick=0;
    }
```

```
    }
    if(!((GetAsyncKeyState(VK_LEFT)||GetAsyncKeyState(VK_RIGHT))))
    {
      m_Player[0]->SetFrame(0);
    }
}
//---------------------------------------------------------------------------
void GGShooting::UpdatePlayer()
{ //เป็นฟังก์ชันที่ใช้ในการเลื่อน Player ขึ้นตามความเร็วที่กำหนด
   if(m_Map[m_CurrentMap]->GetMapPosY()>=m_Speed)
   {
      m_Player[0]->MoveUp(m_Speed);
      if(HitObject(m_Player[0])!=-1)
      {
        m_Player[0]->MoveDown(m_Speed);
      }
   }
}
//---------------------------------------------------------------------------
void GGShooting::UpdateMap()
{ //เป็นฟังก์ชันที่ใช้ในการเลื่อนฉากขึ้นตามความเร็วที่กำหนด
   if(m_Map[m_CurrentMap]->GetMapPosY()>=m_Speed)
   {
      m_Map[m_CurrentMap]->WrapScrollUp(m_Speed);
   }
}
//---------------------------------------------------------------------------
void GGShooting::Update()
{ //เป็นฟังก์ชันที่ใช้ในการ Update วัตถุและการแสดงผลทั้งหมด
   Run(); // Run RunGenerator & Listener
   UpdateInput();
   if(ShootingEnv.sgeMapScrollSpeed!=0)
   {
     UpdateMap(); // Wrap Scrolling Up by Map speed.
```

```
    }
    m_Map[m_CurrentMap]->DrawClipped(m_Screen->GetBack(),&m_Window);
    UpdateObjects(); // Another object is below player object.
    UpdatePlayerMissile();

    // Update Enemy Missile still tested
    UpdateEnemyMissile();

    if(ShootingEnv.sgeMapScrollSpeed!=0)
    {
      UpdatePlayer();
    }
    m_Screen->Flip();
}
//---------------------------------------------------------------------------
void GGShooting::UpdateObjects()
{ //เป็นฟังก์ชันที่ใช้ในการแสดงวัตถุทั้งหมดที่อยู่ในฉาก
    // Update Object only in Current Map
    // Show All Static Objects
    for(int i=0;i<m_CurrentMapStaticObjectCount;i++)
    {
      ShowObject((GObject*)m_CurrentMapStaticObject[i]);
    }
    // Show All Dynamic Objects
    for(int i=0;i<m_CurrentMapDynamicObjectCount;i++)
    {
      ShowObject((GObject *)m_CurrentMapDynamicObject[i]);
    }
    // Show All Player Objects
    for(int i=0;i<m_PlayerCount;i++)
    {
      ShowObject((GObject *)m_Player[i]);
    }
}
```

```
//---------------------------------------------------------------------
void GGShooting::UpdatePlayerMissile()
{ //เป็นฟังก์ชันที่ใช้ในการแสดงวัตถุที่เป็น Player Missile
  if(PlayerMissile->m_Child->Active)
  {
    ShowObject((GObject*)PlayerMissile);
    GDynamicObject *obj;
    for(int i=0;i<PlayerMissile->GetTotalChild();i++)
    {
      if(PlayerMissile->m_Child->isRunning(i))
      {
        if((PlayerMissile->m_Child->Child[i]->GetScreenPosY()<=0)||(PlayerMissile->m_Child->
Child[i]->GetScreenPosY()>=m_Screen->GetHeight()))
        { // Missile go out of screen
          PlayerMissile->m_Child->Stop(i);
        } else
        {
          obj = (GDynamicObject*)PlayerMissile->m_Child->Child[i];
          obj->MoveUp(obj->GetVelY());
          int hit=HitObject(obj);
          if(hit!=-1)
          {
            GObject *hitobj = GetObject(hit);
            GRemoteEvent *remoteEvent = new
GRemoteEvent(hitobj,2,GO_HIT,MP_NONE,NULL);
            hitobj->Notify(remoteEvent);
            PlayerMissile->m_Child->Stop(i);
            //obj->MoveDown(obj->GetVelY());
          }
        }
      }
    }
  }
}
```

```
//---------------------------------------------------------------------
void GGShooting::UpdateEnemyMissile()
{ //เป็นฟังก์ชันที่ใช้ในการแสดงวัตถุที่เป็น Enemy Missile
   for(int i=0;i<m_CurrentMapEnemyMissileCount;i++)
   {
      RunEnemyMissile(m_CurrentMapEnemyMissile[i]);
   }
}
//---------------------------------------------------------------------
void GGShooting::RunEnemyMissile(GDynamicObject *enemyMissile)
{ //เป็นฟังก์ชันที่ใช้ในการกำหนดและควบคุมการทำงานของวัตถุ Enemy Missile ทั้งฉาก
   GDynamicObject *obj;
   // define Enemy Missile Script
   SCRIPT script;
   script.sID=0;
   script.sClassID=CID_GDYNAMICOBJ;
   script.sActive=true;
   script.sType=ST_CHILDMOVEDOWN;
   script.sParam1 = 0; // Child Id 0 move down.

   for(int i=0;i<enemyMissile->GetTotalChild();i++)
   {
      if(enemyMissile->m_Child->isRunning(i))
      {
         if((enemyMissile->m_Child->Child[i]->GetScreenPosY()<=0)||(enemyMissile->m_Child->
Child[i]->GetScreenPosY()>=m_Screen->GetHeight()))
         { // Missile go out of screen
            enemyMissile->m_Child->Stop(i);
         } else
         {
            obj = (GDynamicObject*)enemyMissile->m_Child->Child[i];
            if(enemyMissile->hasScript())
            {
               enemyMissile->m_Script->SetMainActive(true);
```

```
            script.sParam1=i;
            enemyMissile->m_Script->CheckScript(script);
          } else
          {
            obj->MoveDown();
          }
          if(m_Player[0]->Hit((GObject*)obj))
          {
            GObject *hitobj = (GObject*)m_Player[0];
            // Only EventID = 2 the object that hit will be destroy
            // And all Object that want to receive event 2 must be defined.
            GRemoteEvent *remoteEvent = new GRemoteEvent(hitobj,2,GO_HIT,MP_NONE,NULL);
            hitobj->Notify(remoteEvent);
            enemyMissile->m_Child->Stop(i);
            //obj->MoveDown(obj->GetVelY());
          }
        }
      } else
      {
        //enemyMissile->m_Child->Run(i,true,true);
      }
    }
}
//---------------------------------------------------------------------------
bool GGShooting::LoadShootingEnv(const char *filename)
{ //เป็นฟังก์ชันที่ใช้ในการ load ไฟล์ที่ใช้ในการทำงานของเกมประเภทยานยิง
  TMemoryStream *mem1 = new TMemoryStream;
  mem1->LoadFromFile(filename);
  if(mem1->Size != sizeof(SHOOTINGGAMEENV)) return false;
  mem1->Read(&ShootingEnv,sizeof(SHOOTINGGAMEENV));
  if(ShootingEnv.sgeMapScrollSpeed==0) m_Speed=1;
  else m_Speed=ShootingEnv.sgeMapScrollSpeed;
  delete mem1;
  AnsiString file = m_PlayerDirectory+ShootingEnv.sgePlayerMissileFile;
```

```
    if(!LoadPlayerMissile(file.c_str())) return false;
    return true;
}
//---------------------------------------------------------------------------
void GGShooting::InitialShootingGame()
{ //เป็นฟังก์ชันที่ใช้ในการสร้างและกำหนดค่าเริ่มต้นต่างๆที่ใช้ในการทำงานของเกมประเภทยานยิง
    // Initial Player Movement Limitation on Screen
    m_PlayerRightLimit = m_Screen->GetWidth()-m_Player[0]->m_Tile->GetPixelWidth();
    m_PlayerBottomLimit = m_Screen->GetHeight()-m_Player[0]->m_Tile->GetPixelHeight();
    m_PlayerMissileNow=0;
    m_PlayerMissileTick=0;
    // Enable Player Missile
    PlayerMissile->m_Child->SetMainActive(true);
    SetCurrentMap(0);
}
//---------------------------------------------------------------------------
bool GGShooting::LoadPlayerMissile(const char *filename)
{ //เป็นฟังก์ชันที่ใช้ในการ load ไฟล์ Player Missile
    PlayerMissile = new GDynamicObject(this,m_Screen);
    if(!PlayerMissile->LoadFromFile(filename)) return false;
    PlayerMissile->SetDefault(m_Screen->GetBack(),m_Map[PlayerMissile->GetMapID
()],&m_Window);
    if(!PlayerMissile->InitialChildObject()) return false;
    return true;
}
//---------------------------------------------------------------------------
bool GGShooting::MakeCurrentMapEnemyMissile(int map)
{ //เป็นฟังก์ชันที่ใช้ในการสร้างวัตถุ Enemy Missile ที่อยู่ทั้งหมดในฉาก
    if((map>=0)&&(map<m_MapCount))
    {
        int enemyMissileCount=0;
        for(int i=0;i<m_CurrentMapDynamicObjectCount;i++)
        {
            if(m_CurrentMapDynamicObject[i]->GetType()==DO_ENEMYMISSILE)
```

```
        {
          if(m_CurrentMapDynamicObject[i]->isMainChild())
          {
            enemyMissileCount++;
          }
        }
      }
      m_CurrentMapEnemyMissile = new GDynamicObject*[enemyMissileCount];
      m_CurrentMapEnemyMissileCount = enemyMissileCount;
      int count=0;
      for(int i=0;i<m_CurrentMapDynamicObjectCount;i++)
      {
        if(m_CurrentMapDynamicObject[i]->GetType()==DO_ENEMYMISSILE)
        {
          m_CurrentMapEnemyMissile[count++] = m_CurrentMapDynamicObject[i];
        }
      }
      return true;
    } else return false;
}
//---------------------------------------------------------------------------
bool GGShooting::SetCurrentMap(int map)
{ //เป็นฟังก์ชันที่ใช้ในการกำหนดฉากปัจจุบัน
  if(!MakeCurrentMapObjectList(map)) return false;
  if(!MakeCurrentMapEnemyMissile(map)) return false;
  return true;
}
//---------------------------------------------------------------------------
```

สรุปขั้นตอนการทำงานของ Game Runtime โดยคร่าวๆนั้นมีดังนี้

1. Load ข้อมูลที่เป็น Game System Environment จากแฟ้มข้อมูล
   เป็นการนำข้อมูลที่เป็นมาตรฐานสำหรับเกมทุกรูปแบบซึ่งไม่ได้เฉพาะเจาะจงว่าจะใช้กับเกมประเภทใดประเภทหนึ่งขึ้นมา โดยใช้ Function GGOS::LoadGameSystemEnv ("gamesystem.env")

2. ทำการสร้าง และกำหนดค่าต่างๆที่จำเป็นต้องใช้ในการทำงานของเกมจาก Game System Environment ที่ได้นำขึ้นมาแล้ว โดยใช้ Function GGOS::InitialGameSystem() ซึ่งมีรายละเอียดการทำงานดังนี้
   2.1 ทำการสร้างระบบการแสดงผลสำหรับเกมนั้น ซึ่งก็คือการสร้าง GDXScreen Object ขึ้นมา
   2.2 ทำการ Load ข้อมูลที่เป็นตารางสี (Palette) จากแฟ้มข้อมูลที่กำหนดไว้
   2.3 ทำการกำหนดขนาดของการแสดงผล
   2.4 ทำการ Load ข้อมูลที่เป็นฉากและแผนที่ทั้งหมดของเกม (Map File)
   2.5 ทำการ Load ข้อมูลที่เป็นผู้เล่นในเกม(Player File)
   2.6 ทำการ Load ข้อมูลที่เป็นวัตถุที่สามารถเคลื่อนที่ได้ในเกมทั้งหมด (Dynamic Object File)
   2.7 ทำการ Load ข้อมูลที่เป็นวัตถุที่ไม่สามารถเคลื่อนที่ได้ในเกมทั้งหมด (Static Object File)
   2.8 ทำการกำหนดค่าคงที่ต่างๆ
3. Load ข้อมูลที่เป็น Shooting Game System Environment จากแฟ้มข้อมูล โดยใช้ Function GGShooting::LoadShootingEnv("shooting.env")
4. ทำการสร้างและกำหนดค่าต่างๆที่จำเป็นต้องใช้ในการทำงานของเกม ในรูปแบบของเกมประเภทยานยิง โดยใช้ Function GGShooting::InitialShootingGame() ซึ่งมีรายละเอียดการทำงานดังนี้
   4.1 ทำการกำหนดค่าความเร็วการเคลื่อนที่ของฉาก
   4.2 ทำการ Load ข้อมูลที่เป็นรูปแบบกระสุนของยานผู้เล่น
   4.3 ทำการกำหนดค่าคงที่ต่างๆ
5. ทำการเรียกฟังก์ชัน GGShooting::Update() ซึ่งฟังก์ชันนี้จะถูกเรียกวนซ้ำไปเรื่อยๆจนกว่าจะจบเกม โดยมีหน้าที่การทำงานดังนี้
   5.1 ทำให้วัตถุทั้งหมดที่เป็น Event Generator ทำงานเพื่อให้แต่ละวัตถุนั้นสามารถที่จะตรวจสอบได้ว่ามีเหตุการณ์ตามที่กำหนดไว้เกิดขึ้นหรือไม่ โดยถ้ามีเหตุการณ์เกิดขึ้นแล้วมันก็จะส่งเหตุการณ์นั้นไปยังวัตถุเป้าหมายซึ่งได้กำหนดไว้แล้วเช่นกัน และเมื่อส่งไปถึงแล้ววัตถุเป้าหมายก็จะทำงานตามที่ได้กำหนดไว้ใน Remote Event Listener ในทันที
   5.2 ทำการรับข้อมูลจากอุปกรณ์รับข้อมูล และนำไปตรวจสอบว่าจะทำอะไรต่อไป
   5.3 ทำการ update การแสดงผลของฉาก
   5.4 ทำการ update การแสดงผลของวัตถุทุกๆวัตถุที่อยู่ในฉากปัจจุบันที่แสดงอยู่ ซึ่งมีวัตถุที่สามารถเคลื่อนที่ได้ วัตถุที่ไม่สามารถเคลื่อนที่ได้ ผู้เล่น และกระสุน ในส่วนของการ update การแสดงผลนี้จะมีขั้นตอนในการตรวจสอบการชนกันของวัตถุด้วย ซึ่งสามารถ

กำหนดให้มีการทำงานแบต่างๆกันได้เมื่อมีการชนกันของวัตถุ โดยกำหนดได้ใน Event Generator และ Remote Event Listener

5.5 ทำการแสดงผลจริงออกมาทางหน้าจอ โดยทำการสลับ(Flip)ระหว่าง Front Surface และ Back Surface ซึ่งเป็น Surface ที่ใช้ในการแสดงผล โดยมีลักษณะเป็น Double Buffer

6. ถ้ามีการกดปุ่ม ESC ก็จะหยุดและออกจากการทำงานของ Game Runtime

ในรายละเอียดการทำงานของ Game Runtime นั้นค่อนข้างที่จะสลับซับซ้อนมาก เพราะเนื่องมาจาก Game Runtime นี้ถูกออกแบบมาให้ทำงานแบบระบบกระตุ้นโดยเหตุการณ์ (Event Driven Control) ซึ่งจะมีการสร้างและรับเหตุการณ์ที่ได้สร้างขึ้นแล้วนั้น เพื่อที่จะทำงานตามที่ต้องการ โดยจะกล่าวถึงรายละเอียดของระบบ Event Driven Control อีกที

4.3 คลาสต่างๆที่ใช้ในการทำงานของ Game Runtime

แบ่งเป็นกลุ่มออกได้เป็น 2 กลุ่มคือ

กลุ่มของ Class ที่ใช้ในการติดต่อกับ DirectX ซึ่งมีดังนี้

1.1 GDXSurface Class

1.2 GDXScreen Class

1.3 GDXTile_Class

1.4 GDXError Class

กลุ่มของ Class ที่ใช้ในการทำงานหลักของ Game Runtime ซึ่งมีดังนี้

1.5 GGOS Class

1.6 GGShooting Class

1.7 GObject Class

1.8 GDynamicObject Class

1.9 GStaticObject Class

1.10 GPlayer Class

1.11 GChild Class

1.12 GEvent Class

1.13 GRemoteEvent Class

1.14 GEventGenerator Class

1.15 GMap Class

4.4 ตัวอย่างการทำงานของ Game Runtime

สำหรับตัวอย่างนั้นจะประกอบไปด้วย

1. GDynamicObject 3 Object ซึ่งจะกำหนดเป็น 1 วัตถุที่เป็นศัตรู (Dyna1), 1 วัตถุที่เป็นลูกกระสุนของศัตรู (Dyna2) และ 1 วัตถุที่เป็นลูกกระสุนของผู้เล่น(PMissile)
2. GPlayer 1 Object ซึ่งเป็นวัตถุผู้เล่น
3. GDXMap 2 Object สำหรับฉาก 2 ฉาก

และจะมี file ดังนี้

1. Dyna1.obj , Dyna2.obj , PMissile.obj
2. Player.obj
3. Map1.map , map2.map
4. Gamesystem.env
5. Shooting.env
6. Dynaobj.gbl
7. Map.gbl
8. Player.gbl

โดยวัตถุต่างๆมีการกำหนด EventGenerator และ EventListener ดังนี้

1. วัตถุ Dyna1 ซึ่งเป็นวัตถุศัตรูจะมีการกำหนดให้สามารถยิงกระสุนออกมาเมื่ออยู่ที่ตำแหน่ง 0 บนจอถาพได้-
2. วัตถุ Dyna2 ซึ่งเป็นวัตถุลูกกระสุนศัตรูจะมีการกำหนดให้สามารถสร้างและเคลื่อนที่ออกจากวัตถุ Dyna1 ได้
3. วัตถุ Player ซึ่งเป็นวัตถุผู้เล่นจะมีการกำหนดให้สามารถเปลี่ยนฉากได้เมื่อกด Left-CTRL
4. วัตถุ PMissile ซึ่งเป็นวัตถุกระสุนผู้เล่นไม่ต้องกำหนดเหตุการณ์แต่อย่างใด

เมื่อรัน Game Runtime ก็จะมีขึ้นตอนการทำงานดังนี้

1. GGOS จะอ่าน gamesystem.env ขึ้นมาโดยใช้ฟังก์ชัน GGOS::LoadGameSystemEnv ("gamesystem.env")
2. GGOS จะสร้างและกำหนดค่าเริ่มต่างๆของเกมโดยใช้ฟังก์ชัน GGOS::InitialGameSystem()
   2.1 GGOS สร้าง Screen โดยใช้ฟังก์ชัน GGOS::MakeScreen(GameSystemEnv.gseScreenWidth,GameSystemEnv.gseScreenHeight,GameSystemEnv.gseScreenBPP)
   2.2 GGOS สร้าง Palette โดยใช้ฟังก์ชัน m_Screen->LoadPalette(((AnsiString) GameSystemEnv.gsePaletteFilename).c_str())
   2.3 GGOS กำหนดขนาดของการแสดงผล โดยใช้ฟังก์ชัน SetWindow(0,0,GameSystemEnv.gseScreenWidth,GameSystemEnv.gseScreenHeight)

2.4 GGOS load และสร้างวัตถุ GDXMap ขึ้นมาโดยใช้ฟังก์ชัน LoadMapFile (((AnsiString)GameSystemEnv.gseGlobalMapFilename).c_str()) ซึ่งจะได้วัตถุของ Map1 และ Map2

2.5 GGOS load และสร้างวัตถุ GPlayer ขึ้นมาจากฟังก์ชัน LoadPlayerFile (GameSystemEnv.gsePlayerFilename) ซึ่งจะได้วัตถุของ Player

2.6 GGOS load และสร้างวัตถุ GDynamicObject ขึ้นมาโดยใช้ฟังก์ชัน LoadDynamicObjectFile(((AnsiString)GameSystemEnv.gseGlobalDynaObjFilename) .c_str()) ซึ่งจะได้วัตถุของ Dyna1 และ Dyna2

3. GGShooting จะอ่าน shooting.env ขึ้นมาโดยใช้ฟังก์ชัน GGShooting::LoadShootingEnv ("shooting.env")

3.1 GGShooting load และสร้างวัตถุ GDynamicObject ที่เป็น PlayerMissile ขึ้นมาซึ่งจะได้วัตถุของ PMissile

4. GGShooting จะสร้างและกำหนดค่าเริ่มต่างๆของเกมโดยใช้ฟังก์ชัน GGShooting::InitialShootingGame()

4.1 GGShooting กำหนดค่าเริ่มต้นสำหรับตัวแปรต่างๆที่ใช้ในการทำงานของเกมประเภทยานยิง

4.2 GGShooting กำหนดให้ฉากเริ่มต้นที่ฉาก 0 โดยใช้ฟังก์ชัน GGShooting::SetCurrentMap(0)

5. GGShooting จะเข้าสู่ loop ของทำการ update โดยใช้ฟังก์ชัน GGShooting::Update()

5.1 GGShooting จะทำการสั่งให้ Object ทั้งหมดซึ่งก็คือ Player , Dyna1 ,Dyna2 ,PMissile ให้ทำการเช็คตัวเองว่าสามารถกำเนิดเหตุการณ์ได้หรือไม่โดย GGShooting จะใช้ฟังก์ชัน GGOS::Run() และใน Run() นั้นจะมีการสั่งให้ Object ทุกตัวใช้ฟังก์ชัน GObject::Run() อีกทีหนึ่ง โดยถ้าตรวจสอบได้ว่าสามารถกำเนิดเหตุการณ์ได้ วัตถุนั้นก็จะสร้างวัตถุเหตุการณ์ (GRemoteEvent Object) และส่งไปยังวัตถุปลายทางที่ได้กำหนดไว้(ที่มี GEventListener Object อยู่) โดยผ่านฟังก์ชัน GObject::Notify(GRemoteEvent *remoteEvent) และเมื่อวัตถุปลายทางได้รับวัตถุเหตุการ์แล้วมันก็จะตรวจสอบดูว่าจะต้องทำอะไรเพื่อตอบสนองเหตุการณ์ที่ได้รับมา

5.2 GGShooting จะทำการรับและตรวจสอบ keyboard โดยใช้ฟังก์ชัน GGShooting::UpdateInput()

5.3 GGShooting จะทำการ update ฉากโดยใช้ฟังก์ชัน GGShooting::UpdateMap()

5.4 GGShooting จะทำการวาดฉากลงใน Screen Object ที่ได้สร้างขึ้น

5.5 GGShooting จะทำการแสดงผลวัตถุทุกๆวัตถุที่อยู่ในฉากปัจจุบันโดยใช้ฟังก์ชัน GGShooting::UpdateObjects() ซึ่งก็คือวัตถุ Dyna1,Dyna2

5.6 GGShooting จะทำการแสดงผลวัตถุที่เป็นลูกกระสุนของผู้เล่นโดยใช้ฟังก์ชัน

GGShooting::UpdatePlayerMissile() ซึ่งก็คือวัตถุ PMissile

5.7 GGShooting จะทำการแสดงผลวัตถุที่เป็นกระสุนของศัตรูทั้งหมดในฉากนั้นๆ โดยใช้ฟังก์ชัน GGShooting::UpdateEnemyMissile() ซึ่งก็คือวัตถุ Dyna2

5.8 GGShooting จะทำการแสดงผลวัตถุผู้เล่นโดยใช้ฟังก์ชัน GGShooting::UpdatePlayer() ซึ่งก็คือวัตถุ Player

5.9 GGShooting จะทำการ Flip ระหว่าง Back buffer ไปที่ Front Buffer เพื่อให้แสดงผลออกทางจอภาพโดยใช้คำสั่ง m_Screen->Flip() โดยที่ m_Screen เป็นวัตถุของ GDXScreen ที่ได้สร้างไว้ในตอนต้น

6. กลับไปยังข้อ 5 และทำเช่นนี้เรื่อยๆจนกว่าจะออกจากเกมโดยการกด ESC

บทที่ 5
วัตถุในเกม (Object In Game)

แนวคิดหลักในการทำงานสำหรับเกมที่ได้ออกแบบขึ้นมานี้นั้นก็คือการมองสิ่งต่างๆที่มีอยู่ในเกมทุกๆสิ่งยกเว้นฉากเป็นวัตถุ (Object) ที่มีคุณสมบัติและวิธีการทำงานเฉพาะแบบตามที่วัตถุนั้นๆจำเป็นต้องมีซึ่งขึ้นอยู่กับว่าวัตถุนั้นเป็นวัตถุอะไรและมีความสามารถและการทำงานอย่างไรบ้าง การจำลองรูปแบบการทำงานของวัตถุนั้นเป็นแนวคิดจากหลักการของการออกแบบเชิงวัตถุ(Object Oriented Design) ซึ่งได้นำมาประยุกต์ใช้ในการออกแบบวัตถุในเกมนี้ หลักการวิเคราะห์และพิจารณาวัตถุต่างๆที่มีในเกมนั้นจะต้องดูว่าเกมทุกประเภทมีอะไรที่เหมือนกัน มีการทำงานอะไรบ้างที่เหมือนกันไม่ว่าจะเป็นเกมประเภทไหนก็ตาม การพิจารณาองค์ประกอบต่างๆที่จะต้องมีในเกมทุกๆประเภทนี้ทำให้เราสามารถออกแบบโครงสร้างขององค์ประกอบและการทำงานที่จำเป็นจต่อการทำงานของเกมได้ นอกจากนี้ยังต้องพิจารณาถึงความสัมพันธ์ต่างๆที่เกิดขึ้นในเกม ความสัมพันธ์ระหว่างวัตถุต่างๆในเกม การเกิดเหตุการณ์ขึ้นในวัตถุใดๆ ซึ่งจะกล่าวถึงในบทของ “เหตุการณ์ในเกม(Event In Game)” และในการทำงานหลักของเกมนั้นก็ได้มีการควบคุมการทำงานของวัตถุทุกๆวัตถุที่อยู่ในเกมอีกทีหนึ่งเพื่อให้วัตถุต่างๆเหล่านั้นทำงานไปได้ด้วยความถูกต้องและทำให้วัตถุต่างๆสามารถติดต่อสื่อสารกันได้

เมื่อเรามองสิ่งต่างๆในเกมเป็นวัตถุแล้วก็จะต้องกำหนดว่าวัตถุนั้นคืออะไร มีคุณสมบัติอะไร มีการทำงานอย่างไร มีความสัมพันธ์อะไรบ้าง มีสิ่งที่ทำให้เกิดการเปลี่ยนแปลงคุณสมบัติของวัตถุนั้นๆอะไรบ้าง และเกิดขึ้นได้อย่างไร เมื่อไร รวมทั้งการทำงานที่วัตถุทุกๆวัตถุต้องทำเพื่อทำให้เกมนั้นสามารถดำเนินไปได้ตามที่ได้กำหนดและสร้างไว้บนโปรแกรมสร้างและพัฒนาเกมสำเร็จรูปนั้น การกำหนดและสร้างวัตถุต่างๆที่มีอยู่ในเกมจะต้องคำนึงถึงรูปแบบหรือประเภทของเกมที่ได้ทำขึ้นว่าจำเป็นต้องมีวัตถุอะไรบ้าง มีเพื่ออะไร

ในส่วนนี้ผู้สร้างได้ออกแบบคลาสที่เป็นรูปแบบมาตรฐานของวัตถุต่างๆในเกมไว้ในคลาสที่ชื่อว่า “GObject “ ซึ่งมีการใส่ค่าคุณสมบัติและการทำงานต่างๆที่จำเป็นต่อการใช้งานไว้ รวมทั้งยังได้สร้างคลาสสืบทอดลงไปอีกซึ่งก็คือ “GDynamicObject Class” และ “GStaticObject Class” โดยใน GDynamicObject Class นั้นเป็นคลาสที่เฉพาะเจาะจงลงไปในวัตถุที่สามารถเคลื่อนที่ได้ โดยจะมีคุณสมบัติและฟังก์ชันการทำงานที่ทำให้วัตถุนั้นสามารถเคลื่อนที่ได้ตามที่ได้กำหนดไว้ และใน GStaticObject Class นั้นเป็นคลาสที่เฉพาะเจาะจงลงไปในวัตถุที่ไม่สามารถเคลื่อนที่ได้ คลาสทั้งสองคลาสที่ได้สืบทอดลงจาก GObject Class นี้ทำให้การมองวัตถุในเกมเป็นรูปธรรมชัดเจนมากยิ่งขึ้น

นอกจากนี้คลาสที่สำคัญที่ได้เป็นวัตถุอีกคลาสหนึ่งก็คือ คลาสผู้เล่น (GPlayer Class) ซึ่งเป็นคลาสที่สืบทอดมาจาก GDynamicObject Class ทำให้คลาสผู้เล่นนั้นมีคุณสมบัติและการทำงานที่เอามาจากคลาส GDynamicObject ลงมาอีกทีหนึ่ง และยังเพิ่มคุณสมบัติและฟังก์ชันการทำงานเฉพาะสำหรับคลาสผู้เนนี้ลงไปอีกเพื่อให้เป็นรูปธรรมชัดเจนมากยิ่งขึ้นสำหรับผู้เล่นในเกม

จากคลาส GObject ,GDynamicObject ,GStaticObject และ GPlayer นั้นถือได้ว่าเป็นคลาสพื้นฐานที่จำเป็นต้องมีในเกมและสามารถที่จะนำไปสร้างคลาสอื่นๆที่สืบทอดลงไปเพื่อให้วัตถุนั้นเป็นรูปธรรมชัดเจนตามที่ต้องการได้

วัตถุทุกๆวัตถุสามารถทำงานเป็นตัวกำเนิดและรับเหตุการณ์ได้ซึ่งในขั้นตอนการทำงานของ Game Runtime นั้นจะมีการเรียก Function Run() จากวัตถุนั้นๆเพื่อทำการตรวจสอบว่าวัตถุนั้นสามารถกำเนิดเหตุการณ์ได้หรือไม่ โดยถ้าสามารถกำเนิดได้ก็จะทำการสร้างวัตถุเหตุการณ์และส่งไปยังวัตถุปลายทางที่กำหนดไว้ผ่านทาง Function Notify(GRemoteEvent *remoteEvent) ซึ่งเป็นฟังก์ชันของวัตถุที่สามารถรับเหตุการณ์ได้(มี GEventListener Object อยู่)

บทที่ 6
เหตุการณ์ในเกม (Event In Game)

6.1 หลักการ

สำหรับแนวคิดการทำงานของวัตถุต่างๆที่อยู่ในเกมนั้นได้ใช้รูปแบบการจัดการ และควบคุมวัตถุเหล่นนั้นด้วยการกระตุ้นด้วยเหตุการณ์(Event Driven Control) ซึ่งเป็นรูปแบบที่สามารถใช้ได้ในเกมทุกประเภทที่วัตถุต่างๆในเกมนั้นต้องการขั้นตอน หรือวิธีการทำงานตามที่กำหนดไว้ เพราะการใช้ระบบเหตุการณ์กระตุ้นนี้สามารถเป็นแนวคิดพื้นฐานของธรรมชาติและระบบปฏิบัติการโดยส่วนมาก การใช้การควบคุมการทำงานจากเหตุการณ์ที่เกิดขึ้นจากภายนอก หรือภายในวัตถุเองก็ตามเป็นเสมือนการทำให้เกิดระบบ(System) ขึ้นมาซึ่งระบบนั้นก็จะประกอบไปด้วยตัววัตถุ และ การทำงานของวัตถุนั้นๆที่ทำให้ระบบสามารถทำงานได้และถูกต้างตามที่ต้องการ

เนื่องจาการจำลองเหตุการณ์ต่างๆที่จะเกิดขึ้นในระบบนั้นจะต้องวิเคราะห์และพิจารณาถึงองค์ประกอบของระบบทั้งหมด และวิธีการทำงานของระบบนั้นว่ามันสามารถที่จะดำเนินงานไปได้อย่างไร อะไรที่เป็นตัวแปลในการทำงานบ้าง เมื่อพิจารณาโครงสร้างโดยรวมของระบบได้ทั้งหมดแล้วก็จะนำไปสู่การวิเคราะห์ความสัมพันธ์ระหว่างองค์ประกอบต่างๆที่มีอยู่ในระบบซึ่งในเกมก็คือวัตถุต่างๆที่อยู่ในเกมและสามารถทำให้เกมทำงานได้ตามที่ต้องการนั่นเอง

เมื่อพิจารณาความสัมพันธ์ของวัตถุต่างๆที่มีในเกมโดยรวมได้แล้วก็จำเป็นที่จะต้องกำหนด และออกแบบโครงสร้างการทำงานของความสัมพันธ์นั้นๆว่าจะสร้างและทำให้เกิดขึ้นได้อย่างไรในการทำงานของเกม โดยผู้สร้างได้ออกแบบการทำงานของระบบด้วยการกระตุ้นของเหตุการณ์นี้ตามรูปแบบมาตรฐานของ Jini ™ Distributed Event ซึ่งเป็นมาตรฐานที่ใช้ในการกำหนดและสร้างโครงสร้างการทำงานของเหตุการณ์แบบกระจาย(Distributed Event) ที่เป็นเทคโนโลยีของบริษัท Sun Microsystem ซึ่งในมาตรฐานนี้ถูกพัฒนาในภาษาจาวา(Java) ในปี 1999 โดยการกระจายของเหตุการณ์นี้ทำให้วัตถุที่อยู่กันคนละเครื่องหรือเครื่องเดียวกันนั้นสามารถที่จะเกิดเหตุการณ์ขึ้นและส่งเหตุการณ์นั้นไปยังวัตถุที่ต้องการได้ ด้วยเทคโนโลยีนี้ทำให้ระบบการทำงานของเกมนี้สามารถที่จะพัฒนาให้เป็นไปตามรูปแบบที่มีวัตถุอยู่กันคนละเครื่องบนอินเตอร์เนต และสามารถที่จะติดต่อสื่อสาร หรือมีการทำงานที่ขึ้นอยู่กับกันและกัน ซึ่งเป็นเสมือนความสัมพันธ์ที่กระจายไปตามวัตถุต่างๆในระบบเกมที่อยู่บนเครื่องอื่นๆได้ แต่ในการสร้างและระบบนี้นั้นยังไม่ได้ถูกออกแบบมาเพื่อการทำงานสำหรับระบบเหตุการณ์กระจายนี้โดยตรงจะต้องมีการออกแบบส่วนอื่นๆที่จำเป็นเพิ่มเติมจากคลาสพื้นฐานต่างๆที่ได้ออกแบบและทการสร้างไว้แล้วออกไปอีก ซึ่นำไปสู่การเล่นเกมที่มีรูปแบบที่มีประสิทธิภาพยิ่งขึ้นในโลของโลกาภิวัฒน์นี้

การทำงานของระบบที่กระตุ้นด้วยเหตุการณ์นี้ก็สามารถที่จะใช้ในการทำงานของระบบที่อยู่บนเครื่องเครื่องเดียวกันได้เหมือนกัน ซึ่งก็จะเป็นการส่งวัตถุเหตุการณ์ไปยังวัตถุที่ต้องการบนเครื่องเดียวกันในระบบการทำงานของเกมนั่นเอง

ในการออกแบบคลาสต่างๆที่ใช้ในการทำงานของระบบกระตุ้นด้วยเหตุการณ์นี้ ได้แบ่งออกเป็น 3 คลาสหลักคือ

1. GEventGenerator Class

เป็นคลาสที่ทำหน้าที่ในการกำเนิดเหตุการณ์ให้กับวัตถุ โดยเงื่อนไขของการกำเนิดเหตุการณ์นั้นก็จะขึ้นอยู่กับตัววัตถุนั้นเอง หรืออาจจะขึ้นอยู่กับองค์ประกอบภายนอกก็ได้แล้วแต่จะกำหนดไว้ ซึ่งเงื่อนไขเหล่านี้จะต้องมีการกำหนดไว้แล้วที่แน่นอนสำหรับแต่ละวัตถุใดๆ หรือสำหรับระบบใดๆก็ตาม การกำเนิดเหตุการณ์ที่ไม่ได้กำหนดไว้ จะไม่เป็นผลเพราะเสมือนกับว่าเหตุการณ์ที่เกิดขึ้นโดยไม่ได้กำหนดไว้ล่วงหน้านั้นไม่มีวัตถุอื่นที่รับรู้และสามารถนำไปพิจารณาได้ว่าเหตุการณ์ที่ได้เกิดขึ้นนั้นเป็นเหตุการณ์อะไร คลาสนี้จะมีหน้าที่สำคัญในการตรวจสอบว่าวัตถุนั้นสามาถที่จะเกิดเหตุการณ์ดังที่กำหนดไว้ได้หรือไม่ โดยถ้าสามารถกำเนิดเหตุการณ์นั้นได้ก็จะสร้างวัตถุเหตุการณ์(RemoteEvent Object) ขึ้นโดยมีคุณสมบัติตามทีได้กำหนดไว้สำหรับเหตุการณ์นั้นๆ และส่งไปยังวัตถุที่ต้องการจะรับเหตุการณ์(RemoteEventListener Object) ซึ่งได้กำหนดไว้แล้วว่าเป็นวัตถุตัวใดเช่นกัน โดยโครงสร้างภายในสำหรับคลาสนี้แสดงดังรูป

2. GRemoteEvent Class

เป็นคลาสที่ทำน้าที่ในการเก็บค่าต่างๆสำหรับเหตุการณ์ที่ได้กำเนิดขึ้นแล้ว โดยคลาสนี้จะถูกสร้างเป็น Object และส่งไปยังวัตถุที่ต้องการจะรับเหตุการณ์นั้นๆ โดยจะส่งผ่านฟังก์ชัน Notify(GRemoteEvent *remoteEvent) ที่มีอยู่ในวัตถุที่เป็น GEvent Object โดยโครงสร้างภายในสำหรับคลาสนี้แสดงดังรูป

3. GEvent Class (RemoteEventListener Class)

เป็นคลาสที่ใช้ในการรับวัตถุเหตุการณ์ที่ได้กำเนิดและส่งมาจาก GEventGenerator Object เพื่อที่จะนำมาตรวจสอบดูว่าเหตุการณ์ที่ได้เกิดขึ้นแล้วนั้นคืออะไร และวัตถุนี้จะตอบสนองเหตุการณ์นั้นอย่างไร โดยการกำหนดว่าถ้าเกิดเหตุการณ์ตามที่ต้องการขึ้นแล้วจะให้วัตถุนั้นมีการทำงานหรือให้ระบบ ทำอะไรและทำอย่างไร เพื่อให้ได้ผลลัพธ์ตามที่ได้กำหนดไว้ โดยในคลาสนี้จะต้องมีการกำหนดฟังก์ชันการทำงานและค่าตัวแปรต่างๆที่ต้องใช้ในแต่ละเหตุการณ์เป็นฟังก์ชันฟังก์ชันไป โดยในคลาสนี้มีฟังก์ชัน Notify(GRemoteEvent *remoteEvent) ไว้เพื่อทำหน้าที่ในการรับ GRemoteEvent Object จากวัตถุที่ได้กำเนิดวัตถุเหตุการณ์และส่งมายังวัตถุที่ต้องการรับเหตุการณ์นั้น โดยโครงสร้างภายสำหรับคลาสนี้แสดงดังรูป

6.2 การตรวจสอบการเกิดเหตุการณ์ในวัตถุ

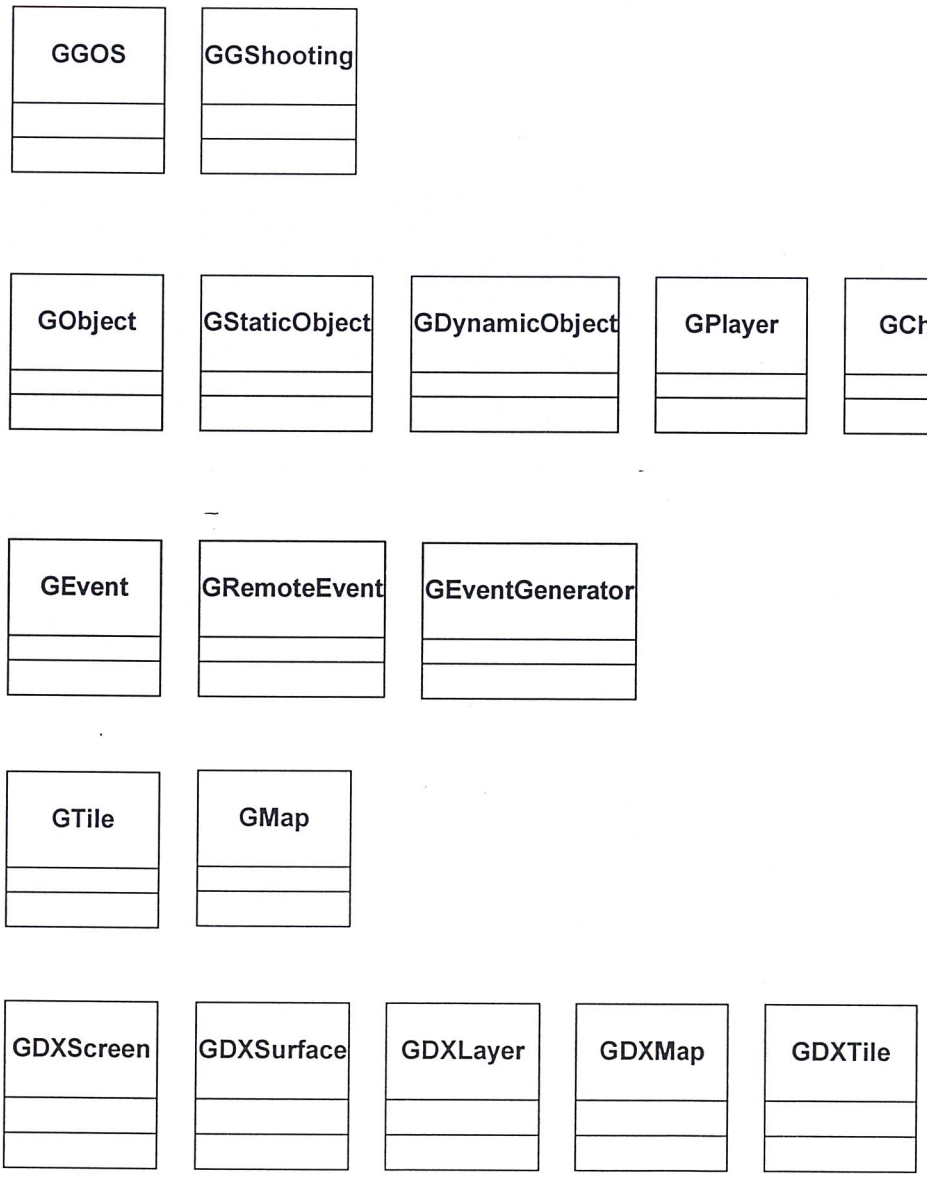
สำหรับการตรวจสอบว่าเหตุการณ์ใดๆในวัตถุที่ได้กำหนดไว้แล้วนั้นสามารถที่จะเกิดขึ้นได้หรือไม่นั้นจะใช้ระบบตรวจสอบจากภายนอกวัตถุนั้น ซึ่งจะอยู่ในคลาส GGOS ที่เป็นคลาสหลักในการ

กำหนดและควบคุมทำงานทั้งหมดของเกม โดยในการตรวจสอบวัตถุว่าสามารถที่จะเกิดเหตุการณ์ได้หรือไม่ GGOS ก็จะเข้าไปเรียกฟังก์ชัน Check() ของวัตถุที่มี GEventGenerator Object อยู่เพื่อที่จะให้ GEventGenerator Object นั้นทำการตรวจสอบเงื่อนไขของการเกิดเหตุการณ์ว่าสามารถกำเนิดได้หรือยัง ถ้าสามารถกำเนิดได้ก็จะสร้าง GRemoteEvent Object สำหรับเหตุการณ์นั้นๆตามที่ได้กำหนดไว้แล้วส่งไปยัง GEvent Object ที่อยู่ในวัตถุเป้าหมายที่ต้องการจะรับเหตุการณ์นั้นๆ โดยมีเงื่อนไขในการส่งวัตถุเหตุการณ์ตามที่กำหนดไว้ในโครงสร้างของ EVENTGENERATOR ซึ่งได้กล่าวไว้ในบทของ “โครงสร้างของแฟ้มข้อมูลที่ใช้ใน Game Runtime” และสำหรับโครงสร้างของข้อมูลที่เก็บว่าถ้ามีเหตุการณ์อะไรเกิดขึ้นแล้วจะตอบสนองต่อเหตุการณ์นั้นอย่างไรก็จะต้องกำหนดไว้ในโครงสร้างของ EVENTLISTENER ซึ่งได้กล่าวไว้ในบทของ “โครงสร้างของแฟ้มข้อมูลที่ใช้ใน Game Runtime” เช่นเดียวกัน ทันทีที่วัตถุเป้าหมายได้รับวัตถุเหตุการณ์ที่ได้ส่งมาแล้วนั้น GEvent Object ก็จะทำงานโดยการเข้าไปตรวจสอบว่าวัตถุเหตุการณ์ที่ได้รับมานั้นคืออะไร และจะตอบสนองเหตุการณ์นั้นอย่างไรบ้าง ตามที่ได้กำหนดไว้

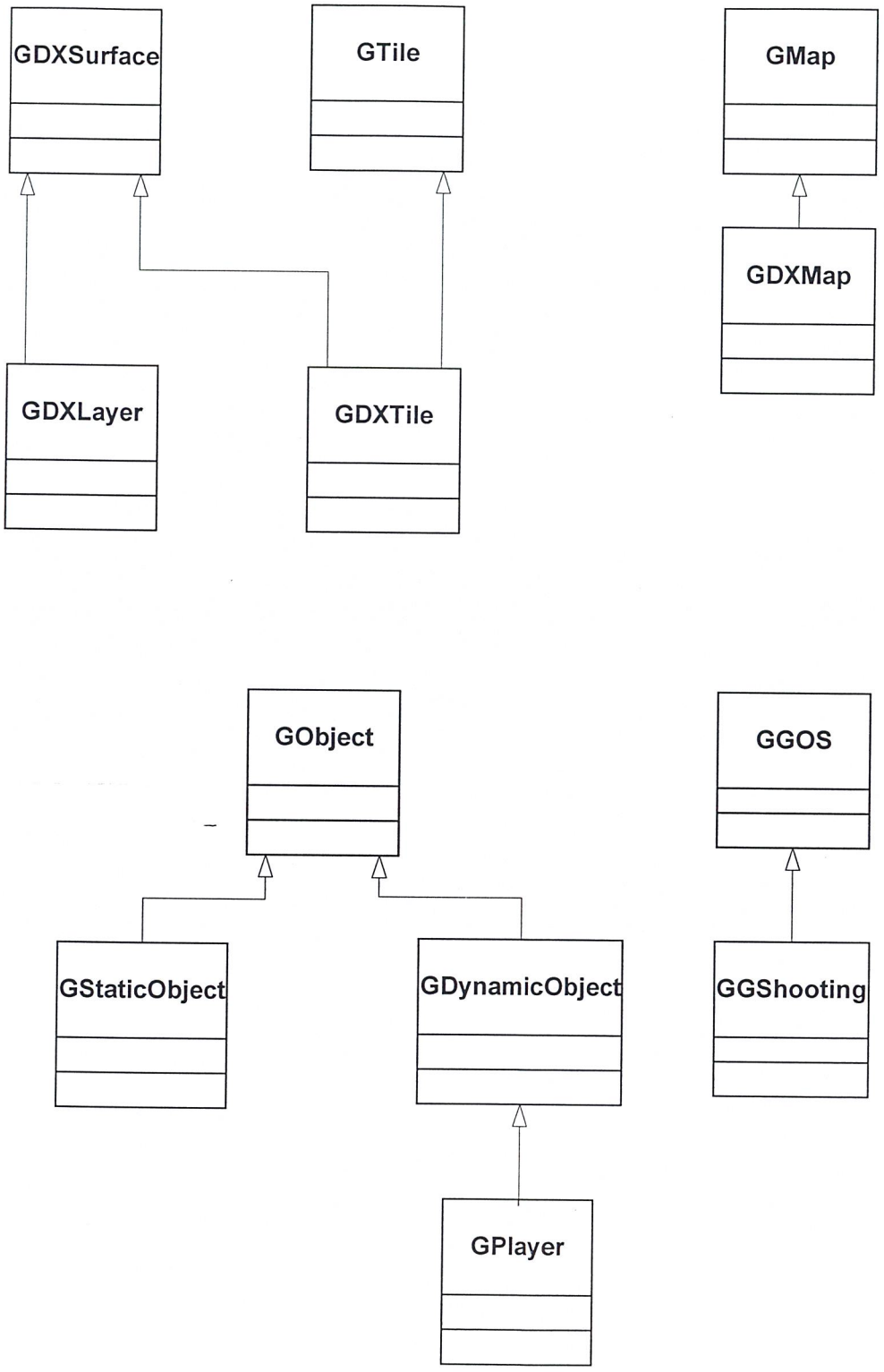
การทำงานของระบบในส่วนที่ทำหน้าที่ควบคุมการกำเนิดเหตุการณ์นี้ เป็นส่วนหนึ่งในหน้าที่หลักของ Game Runtime และเป็นหัวใจสำคัญอย่างยิ่งที่ทำให้เกมสามารถทำงานได้ และเป็นไปตามที่ผู้สร้างต้องการ จึงเป็นเหตุผลสำคัญที่ต้องมีการออกแบบและพัฒนาโครงสร้างการทำงานของระบบกระตุ้นด้วยเหตุการณ์นี้ขึ้นมาใช้ ไม่ใช่เฉพาะเกมประเภทใดประเภทหนึ่งแต่เกือบทุกประเภทได้ใช้หลักการในส่วนนี้

บทที่ 7
คลาสต่างๆที่ใช้ในการทำงานของ Game Runtime

ในการออกแบบการทำงานของ Game Runtime นั้นจะใช้แนวคิดแบบเชิงวัตถุ (Object Oriented Design) โดยจะแยกวัตถุต่างๆที่จำเป็นต่อการสร้างและการทำงานของเกมออกจากกันก่อน และพิจารณาความสัมพันธ์ที่แต่ละวัตถุนั้นมีต่อกัน รวมทั้งวิธีการที่วัตถุแต่ละวัตถุในเกมนั้นจะติดต่อสื่อสารกันโดยคลาสทั้งหมดที่ใช้ใน Game Runtime แสดงได้ดังรูปข้างล่างนี้



รูปที่7.1 แสดงคลาสทั้งหมดที่ใช้ใน Game Runtime



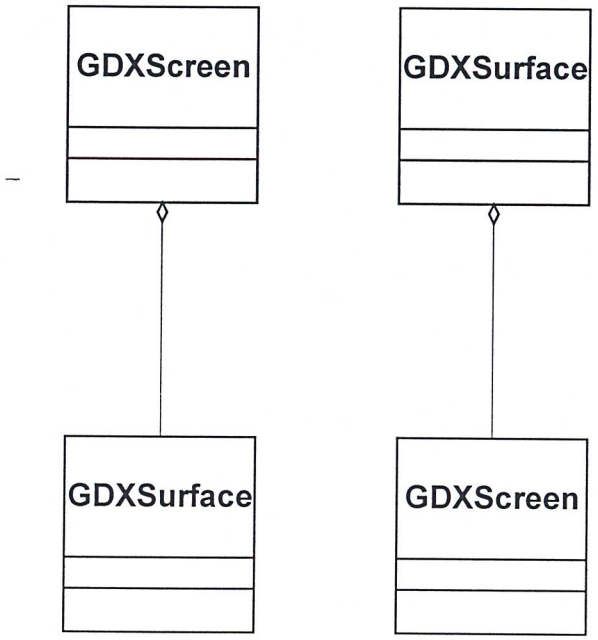
รูปที่7.2 แสดงคลาสความสัมพันธ์ของคลาสทั้งหมดที่เป็นการสืบทอด (Inheritance)

โดยคลาสทั้งหมดนี้สามารถพิจารณาได้เป็น 3 ส่วนคือส่วนของคลาสที่ทำหน้าที่ในการติดต่อกับ DirectX API และคลาสที่ทำหน้าที่ในการทำงานส่วนบนเหนือคลาสที่ทำหน้าที่ติดต่อกับ DirectX API อีก

ทีหนึ่งซึ่งเป็นคลาสที่ไม่ได้มีการทำงานเฉพาะเจาะจงลงไปในเกมประเภทใดประเภทหนึ่ง และส่วนของคลาสที่มีการใช้งานเฉพาะเจาะจงลงไปในแต่ละประเภทของเกม ซึ่งจะมีรายละเอียดในแต่ละส่วนดังนี้

7.1 ส่วนที่ประกอบไปด้วยคลาสที่มีการทำงานเพื่อติดต่อกับ DirectX API(Application Program Interface)

ในส่วนนี้จะประกอบไปด้วย Class ที่ถูกออกแบบมาเพื่อใช้ในการเก็บข้อมูลและฟังก์ชันการทำงานต่างๆ ที่จำเป็นต่อการติดต่อกับอุปกรณ์ภายนอกเช่น การ์ดแสดงผล คีย์บอร์ดและอื่นๆ โดยในโปรแกรมสร้างแบะพัฒนาเกมสำเร็จรูปนี้จะใช้ API ของ DirectX ในส่วนที่เป็นการติดต่อกับการ์ดแสดงผลเท่านั้น ซึ่งทำให้เพิ่มประสิทธิภาพในด้านของความเร็วของการแสดงผลในเกม เช่นการแสดงผลภาพบนจอ การทำ Double Buffering ซึ่งหลักการเหล่านี้ DirectX จะเป็นตัวจัดการให้ทั้งหมดเราเพียงแต่เป็นคนกำหนดและเรียกใช้ฟังก์ชันที่มีให้มาแล้วเท่านั้นเองคลาสที่ออกแบบไว้ในส่วนนี้นั้นมีดังนี้คือ GDXScreen Class,GDXSurface Class, GTile Class , GDXTile Class และ GDXLayer Class และแสดงความสัมพันธ์ของคลาสต่างๆที่ประกอบอยู่ในคลาสเหล่านี้ได้ดังรูปข้างล่างนี้



รูปที่7.3 แสดงองค์ประกอบของคลาส GDXScreen และ GDXSurface (Class Aggregation)

7.1.1 GDXScreen Class

เป็นคลาสที่ใช้ในการกำหนดและสร้างการแสดงผลต่างๆบนจอเช่น การสร้างหน้าจอแสดงผลแบบวินโดว์ หรือแบบเต็มจอ(Full Screen) และการใช้ Buffer ซึ่งเป็นพื้นผิวการแสดงผลชนิดหนึ่ง (Surface) ในการแสดงผลโดยมี Buffer 2 อันคือ Buffer ข้างหน้า(Front Buffer) และ Buffer ข้างหลัง(Back Buffer) ซึ่งเวลาแสดงผลบนจอนั้นภาพที่อยู่ใน Buffer ข้างหน้าจะถูกนำออกมาแสดงให้เห็นเท่านั้น ส่วนภาพที่อยู่ใน Buffer ข้างหลังก็จะไม่ถูกนำออกมาแสดงผลโดนเวลาทำงานนั้นเราก็จะทำการวาดภาพที่ต้องการลงไปใน Buffer ข้างหลังก่อนและเมื่อวาดเสร็จหมดทุกภาพที่ต้องการจะแสดงผลบนหน้าจอเดียวกันแล้วค่อยทำการสลับ Front และ Back Buffer ซึ่ง Back Buffer ก็จะถูกนำออกมาแสดงให้เห็นบนจอแทน ส่วน Front Buffer ก็จะถูกนำไปอยู่ข้างหลังเพื่อใช้ในการเก็บาพและรอที่จะสลับกัลออกมาแสดงผลบนหน้าจออีกทีหนึ่ง ซึ่งในการทำงานของการแสดงผลในเกมนั้นก็จะใช้หลักการนี้ซึ่งเรียกว่า Double Buffering และเป็นหลักการในการแสดงผลอย่างหนึ่งที่เพิ่มประสิทธิภาพในการแสดงผลให้รวดเร็วยิ่งขึ้น และยังทำให้หน้าจอนั้นไม่กระพริบด้วย อันเนื่องมาจากการเขียนภาพทั้งหมดลงให้เสร็จก่อนแล้วค่อยนำมาแสดงผล นั้นรวดเร็วกว่าการเขียนภาพลงไปเพื่อแสดงผลเลยในแต่ละภาพซึ่งช้ากว่ามาก โดยในคลาส GDXScreen นี้มีฟังก์ชันการทำงานที่สำคัญดังนี้

1. bool CreateFullScreen(void *hWnd,DWORD Width,DWORD Height,DWORD BPP,bool bVGA=false) เป็นฟังก์ชันการทำงานที่ใช้ในการสร้างหน้าจอการแสดงผลแบบเต็มจอ(Full Screen) โดยสามาถกำหนดความกว้าง ความสูงและจำนวนสีที่จะแสดงได้
2. bool CreateWindowed(void *hWnd,int Width,int Height) เป็นฟังก์ขันการทำงานที่ใช้ในการสร้างหน้าจอการแสดงผลแบบวินโดว์ซึ่งสามารถกำหนดขนาดความกว้างและความสูงของวินโดว์ได้
3. bool LoadBitmap(const char *szFilename) เป็นฟังก์ชันการทำงานที่ใช้ในการ load รูปภาพจากแฟ้มข้อมูลประเภทบิตแมพ(Bitmap) ขึ้นมาอยู่บน Front Buffer เพื่อใช้ในการแสดงผลเลย
4. bool LoadBitmap(Classes::TStream *Stream) เป็นฟังก์ชันการทำงานที่ใช้ในการ load รูปภาพจาก stream ซึ่งเป็น Class ที่ใช้ในการรับส่งข้อมูลบนหน่วยความจำ
5. bool LoadPalette(const char *szFilename) เป็นฟังก์ชันการทำงานที่ใช้ในการ load ข้อมูลที่เป็นตารางสี(palette) ที่ต้องใช้ในการแสดงผลของรูปภาพบนหน้าจอ
6. void Fill(DWORD FillColor) เป็นฟังก์ชันการทำงานที่ใช้ในการใส่สีตามที่กำหนดลงบนทั้งหน้าจอ
7. HRESULT Flip(void) เป็นฟังก์ชันการทำงานที่ใช้ในการสลับพื้นผิวการแสดงผลระหว่าง Front Buffer และ Back Buffer
8. int GetWidth(void) เป็นฟังก์ชันการทำงานที่ใช้ในการนำค่าความกว้างของหน้าจอออกมาใช้
9. int GetHeight(void) เป็นฟังก์ชันการทำงานที่ใช้ในการนำค่าความสูงของหน้าจอออกมาใช้
10. int GetBPP(void) เป็นฟังก์ชันการทำงานที่ใช้ในการนำจำนวนบิตต่อพิกเซล ออกมาใช้

11. GDXSurface *GetFront(void) เป็นฟังก์ชันการทำงานที่ทำหน้าที่ในการนำ Front Buffer ออกมาใช้
12. GDXSurface *GetBack(void) เป็นฟังก์ชันการทำงานที่ทำหน้าที่ในการนำ Back Buffer ออกมาใช้
13. inline LPDIRECTDRAWPALETTE GetPalette(void) เป็นฟังก์ชันการทำงานที่ทำหน้าที่ในการนำข้อมูลตารางสีออกมาใช้

7.1.2 GDXSurface Class

เป็นคลาสที่ใช้ในการกำหนดและสร้างพื้นผิวของการแสดงผล(Surface) ซึ่งในการแสดงผลของภาพแต่ละภาพนั้นอาจจะถูกกำหนดให้เป็นแต่ละพื้นผิวก็ได้ โดยในแต่ละพื้นผิวนี้ก็จะสามารถนำมาแสดงผลบนจอได้โดยการนำไปใส่ไว้ใน GDXScreen Class ซึ่งในตอนเริ่มต้นของการสร้างพื้นผิวนั้นจะต้องมีการกำหนด GDXScreen Object ที่ได้สร้างไว้แล้วก่อนเพื่อที่จะให้พื้นผิวการแสดงผลต่างๆที่ได้สร้างขึ้นมาจากคลาสนี้ได้ถูกนำไปแสดงผลบนหน้าจอนั้นได้ ซึ่งจริงๆแล้วคลาส GDXScreen เองก็ประกอบไปด้วยพื้นผิวการแสดงผลอยู่สองตัวคือ Front Buffer และ Back Buffer ดังที่ได้กล่าวไปแล้วแต่ Front Buffer และ Back Buffer นั้นเป็นพื้นผิวที่มีขนาดเม่ากับขนาดทั้งหมดของการแสดงผล แต่ใน GDXSurface Class นี้ถือได้ว่าพื้นผิวของการแสดงผลนั้นมีขนาดเท่าไรก็ได้ไม่จำกัดขึ้นอยู่กับว่าเราจะสร้างและใช้งานในขนาดที่มากน้อยเพียงใด และการทำงานที่สำคัญมากของ GDXSurface Class นี้ก็คือการเก็บและแสดงผลของพื้นหิวที่มีอยู่นั้นลงบนหน้าจอและสามารถมีการกำหนดลักษณะการแสดงผลได้ต่างๆรูปแบบกันไป โดยใน GDXSurface Class นี้มีฟังก์ชันการทำงานที่สำคัญดังนี้

1. bool Create(GDXScreen *pScreen,int Width,int Height) เป็นฟังก์ชันการทำงานที่ใช้ในการสร้างพื้นผิวการแสดงผลที่อยู่บนหน้าจอการแสดงผลที่สร้างโดย GDXScreen Class และสามารถกำหนดความกว้างและความสูงของพื้นผิวการแสดงผลนี้ได้
2. void SetDest(int t,int l,int b,int r) เป็นฟังก์ชันการทำงานที่กำหนดขอบเขตของพื้นผิวการแสดงผลปลายทางที่เป็นสี่เหลี่ยมโดยการกำหนดขอบเขตนี้จำเป็นต้องใช้ในการนำพื้นผิวนี้ไปใส่บนอีกพื้นผิวหนึ่งได้ดังที่กำหนดไว้
3. void SetSrc(int t,int l,int b,int r) เป็นฟังก์ชันการทำงานที่กำหนดขอบเขตของพื้นผิวการแสดงผลต้นทางที่เป็นสี่เหลี่ยมโดยการกำหนดขอบเขตนี้จำเป็นต้องใช้ในการนำส่วนของพื้นผิวที่กำหนดไว้ไปลงบนอีกพื้นผิวหนึ่ง
4. void ColorKey(int col) เป็นฟังก์ชันการทำงานที่กำหนดสีโปร่งใส(Transparent Color)
5. int GetPixel(int X,int Y) เป็นฟังก์ชันการทำงานที่ใช้ในการนำข้อมูลสีที่ตำแน่ง X,Y ใดๆบนพื้นผิวนี้ออกมาใช้
6. int GetColorKey(void) เป็นฟังก์ชันการทำงานที่ใช้ในการนำข้อมูลสีที่เป็นสีโปร่งใสออกมาใช้

7. void TextXY(int X,int Y,COLORREF Col,LPCSTR pString) เป็นฟังก์ชันการทำงานที่ใช้ในการเขียนข้อความลงบนพื้นผิวแสดงผลนี้ โดยสามารถกำหนดตำแหน่ง และสีได้
8. void ChangeFont(const char *FontName,int Width,int Height,int Attributes=FW_NORMAL) เป็นฟังก์ชันการทำงานในการเปลี่ยนรูปแบบของตัวอักษรตามที่ต้องการ โดยสามารถกำหนดความกว้างและความสูงของตัวอักษรได้
9. HRESULT Draw(GDXSurface *lpDDS) เป็นฟังก์ชันการทำงานที่ใช้ในการวาดพื้นผิวนี้ลงบนอีกพื้นผิวหนึ่ง
10. DrawFast(int X,int Y,GDXSurface *lpDDS) เป็นฟังก์ชันการทำงานที่ใช้ในการวาดพื้นผิวนี้ลงบนอีกพื้นผิวหนึ่ง โดยสามารถกำหนดตำแน่งเริ่มต้นได้
11. DrawTrans(int X,int Y,GDXSurface *lpDDS) เป็นฟังก์ชันการทำงานที่ใช้ในการวาดพื้นผิวนี้ลงบนอีกพื้นผิวหนึ่งที่มีการวาดแบบ Transparent Mode หรือก็คือการวาดที่ไม่วาดสีที่เป็นสีโปร่งใส โดยสามารถกำหนดตำแน่งเริ่มต้นได้
12. DrawClipped(int X,int Y,GDXSurface *lpDDS,LPRECT ClipRect) เป็นฟังก์ชันการทำงานที่ใช้ในการวาดพื้นผิวนี้ลงอีกพื้นผวหนึ่งซึ่งสามารถกำหนดขอบเขตของพื้นผิวปลายทางของการวาดได้ และสามารถกำหนดตำแน่งเริ่มต้นในการวาดได้

7.1.3 GTile Class

เป็นคลาสที่ใช้ในการกำหนดและสร้างรูปภาพที่ต้องการจะแสดงผล ซึ่งคลาสนี้มีการทำงานที่เกี่ยวข้องกับการกำหนดค่าต่างๆที่จำเป็นต่อการสร้างภาพ รวมทั้งยังใช้ในการ load และ save ข้อมูลทั้งหมดในคลาสนี้อีกด้วย ซึ่งแบ่งเป็น 2 ส่วนคือ ส่วนหัวของแฟ้มข้อมูล และส่วนของข้อมูลที่เป็นบิทแมพ โดยเมื่อทำการกำหนดค่าต่างๆที่บอกถึงคุณสมบัติของภาพนี้แล้วก็จะสามารถเก็บข้อมูลเหล่านี้ลงไปในแฟ้มข้อมูลได้ และในทางกลับกันก็สามารถที่จะ load ข้อมูลเหล่านี้จากแฟ้มข้อมูลที่ได้เก็บไว้แล้วนั้นกลับขึ้นมาเพื่อใช้งานต่อไปได้ โดย GTile Class นี้มีฟังก์ชันการทำงานที่สำคัญดังนี้

1. bool SaveToFile(const char *Filename) เป็นฟังก์ชันการทำงานที่ใช้ในการเก็บข้อมูลของคลาสนี้ทั้งหมดลงในแฟ้มข้อมูล
2. bool LoadFromFile(const char *Filename) เป็นฟังก์ชันการทำงานที่ใช้ในการ load ข้อมูลของคลาสนี้ทั้งหมดจากแฟ้มข้อมูลขึ้นมาเพื่อใช้งานต่อไป
3. bool LoadBitmap(Graphics::TBitmap *bitmap) เป็นฟังก์ชันการทำงานที่ใช้ในการกำหนดภาพที่เป็น Bitmap Class ให้กับคลาสนี้ ซึ่งก็คือการกำหนดภาพที่ต้องการนั่นเอง
4. bool GetBitmap(Graphics::TBitmap *bitmap) เป็นฟังก์ชันการทำงานที่ใช้ในการนำเอาภาพที่เป็น Bitmap Class ที่เก็บอยู่ในคลาสนี้ออกไปใช้งาน
5. void SetBlockWidth(int BlockWidth) เป็นฟังก์ชันการทำงานที่ใช้ในการกำหนดความกว้างของภาพซึ่งถูกแบ่งออกเป็นแฟรมย่อยๆ

6. int GetBlockWidth(void) เป็นฟังก์ชันการทำงานที่ใช้ในการนำเอาความกว้างของภาพซึ่งถูกแบ่งออกเป็นแฟรมย่อยๆ ออกมาใช้งาน
7. int GetBlockHeight(void) เป็นฟังก์ชันการทำงานที่ใช้ในการนำเอาความสูงของภาพซึ่งถูกแบ่งออกเป็นแฟรมย่อยๆ ออกมาใช้งาน
8. void SetBlockHeight(int BlockHeight) เป็นฟังก์ชันการทำงานที่ใช้ในการกำหนดความสูงของภาพซึ่งถูกแบ่งออกเป็นแฟรมย่อยๆ
9. int GetBlockNumbers(void) เป็นฟังก์ชันการทำงานที่ใช้ในการนำจำนวนของภาพที่ถูกแบ่งออกเป็นเฟรมออกมาใช้งาน
10. void SetBlockNumbers(int Num) เป็นฟังก์ชันการทำงานที่ใช้ในการกำหนดจำนวนของภาพย่อยๆที่เป็นเฟรม
11. int GetColorKey(void) เป็นฟังก์ชันการทำงานที่นำเอาค่าสีที่โปร่งใสออกมาใช้งาน
12. void SetColorKey(int colorkey) เป็นฟังก์ชันการทำงานที่ใช้ในการกำหนดค่าสีที่โปร่งใสให้กับภาพที่อยู่ในคลาสนี้
13. int GetMaskSize(void) เป็นฟังก์ชันการทำงานที่ใช้ในการนำจำนวนของ Mask Bit ที่ใช้ในคลาสนี้ออกมาใช้งาน
14. void SetMaskSize(int msize) เป็นฟังก์ชันการทำงานที่ใช้ในการกำหนดจำนวน Mask Bit ที่จะใช้ในคลาสนี้
15. void SetName(const char *Tilename) เป็นฟังก์ชันการทำงานที่ใช้ในการกำหนดชื่อของคลาสนี้
16. char* GetName(void) เป็นฟังก์ชันการทำงานที่ใช้ในการนำชื่อของคลาสนี้ออกมาใช้งาน
17. void SetID(int id) เป็นฟังก์ชันการทำงานที่ใช้ในการกำหนดเลขประจำตัวของคลาสนี้
18. int GetID(void) เป็นฟังก์ชันการทำงานที่ใช้ในการนำเลขประจำตัวของคลาสนี้ออกมาใช้งาน

7.1.4 GDXTile Class

เป็นคลาสที่ใช้ในการกำหนดและสร้างรูปภาพที่ต้องการจะแสดงผล ซึ่งในคลาสนี้นั้นจะเป็นคลาสที่สืบทอด(Inherit) จาก GDXSurface Class และ GTile Class ซึ่งเป็นคลาสที่มองเห็นเป็นวัตถุมากขึ้น เนื่องจากการออกแบบคลาสนี้มีจุดประสงค์เพื่อทำให้เราสามารถมองภาพ 1 ภาพเป็นวัตถุได้ โดยในคลาสนี้จะเน้นไปทางด้านการ Load และ Save ข้อมูลภาพลงบนแฟ้มข้อมูล หรือจาก stream ก็ได้ ซึ่งทำให้อำนวยความสะดวกในการใช้งานมาก โดยหลักการของไทล์(Tile) ก็คือการมีภาพที่เป็นส่วนๆนั้นอยู่รวมกันเป็นภาพใหญ่ภาพเดียว และใช้ตัวเลข(Frame number) ในการบอกว่าภาพที่ต้องการนั้นคือภาพไหน โดย GDXTile Class นี้มีฟังก์ชันการทำงานที่สำคัญดังนี้

1. bool Create(GDXScreen *pScreen,const char *szFilename,int w,int h,int num) เป็นฟังก์ชันการทำงาน(Constructure)ที่ใช้ในการสร้างไทล์จากแฟ้มข้อมูลประเภทบิทแมพลงบน GDXScreen Object ที่ได้กำหนดไว้
2. bool Create(GDXScreen *pScreen,Classes::TMemoryStream *Stream,int w,int h,int num) เป็นฟังก์ชันการทำงาน(Constructure)ที่ใช้ในการสร้างไทล์จากสายข้อมูล(Stream)ประเภทบิทแมพลงบน GDXScreen Object ที่ได้กำหนดไว้
3. bool LoadFromFile(GDXScreen *pScreen,const char *szFilename) เป็นฟังก์ชันการทำงานที่ใช้ในการ load ข้อมูลจากแฟ้มข้อมูลประเภทบิทแมพลงบน GDXScreen Object ที่ได้กำหนดไว้
4. void SetFrame(int frame) เป็นฟังก์ชันการทำงานที่ใช้ในการกำหนดหมายเลขของภาพย่อย (Frame) ที่ต้องการจะแสดง
5. void SetPos(int x,int y) เป็นฟังก์ชันการทำงานที่กำหนดตำแน่งของการแสดงผลบน GDXScreen Object
6. int GetPixelWidth(void) เป็นฟังก์ชันการทำงานที่นำความกว้างของภาพที่ในคลาสนี้ออกมาใช้
7. int GetPixelHeight(void) เป็นฟังก์ชันการทำงานที่นำความสูงของภาพที่ในคลาสนี้ออกมาใช้
8. **และฟังก์ชันทุกฟังก์ชันที่อยู่ใน GDXSurface Class และ GTile Class จะถูกสืบทอดลงมา (เฉพาะส่วนที่เป็น public และ protected) เพื่อใช้ในคลาสนี้ด้วย**

7.1.5 GDXLayer Class

เป็นคลาสที่ใช้ในการกำหนดและสร้างพื้นผิวการแสดงผลที่สามารถแบ่งออกได้เป็น ชั้นๆ(Layer) ซึ่งแต่ละชั้นสามารถที่จะแสดงได้อิสระและซ้อนทับกันได้ ซึ่งคลาสนี้ได้สืบทอดลงมาจาก GDXSurface Class โดยมีฟังก์ชันการทำงานที่ใช้ในการเลื่อนพื้นผิวในแต่ละชั้นนั้นให้ไปในทิศทางและความเร็วที่ต้องการได้ โดย GDXLayer Class นี้มีฟังก์ชันการทำงานที่สำคัญดังนี้

1. GDXLayer(GDXScreen *pScreen,const char *szFilename) เป็นฟังก์ชันการทำงานที่ใช้ในการสร้างชั้นของพื้นผิวจากแฟ้มข้อมูลที่เป็นรูปภาพในมาตรฐานของบิทแมพ
2. GDXLayer(GDXScreen *pScreen,Classes::TMemoryStream *pStream) เป็นฟังก์ชันการทำงานที่ใช้ในการสร้างชั้นของพื้นผิวจากสายข้อมูลที่เป็ฯรูปภาพในมาตรฐานของบิทแมพ
3. void ScrollUp(int Offset) เป็นฟังก์ชันการทำงานที่ใช้ในการเลื่อนพื้นผิวในชั้นนี้ขึ้นไปเป็นจำนวนที่ต้องการ
4. void ScrollDown(int Offset) เป็นฟังก์ชันการทำงานที่ใช้ในการเลื่อนพื้นผิวในชั้นนี้ลงไปเป็นจำนวนที่ต้องการ

5. void ScrollLeft(int Offset) เป็นฟังก์ชันการทำงานที่ใช้ในการเลื่อนพื้นผิวในชั้นนี้ไปทางซ้ายเป็นจำนวนที่ต้องการ
6. void ScrollRight(int Offset) เป็นฟังก์ชันการทำงานที่ใช้ในการเลื่อนพื้นผิวในชั้นนี้ไปทางขวาเป็นจำนวนที่ต้องการ
7. void MoveTo(int XOffset,int YOffset) เป็นฟังก์ชันการทำงานที่ใช้ในการเลื่อนพื้นผิวในชั้นนี้ไปยังตำแหน่งที่ต้องการ
8. และฟังก์ชันในการวาดพื้นผิวซึ่งได้สืบทอดลงมาจาก GDXSurface Class

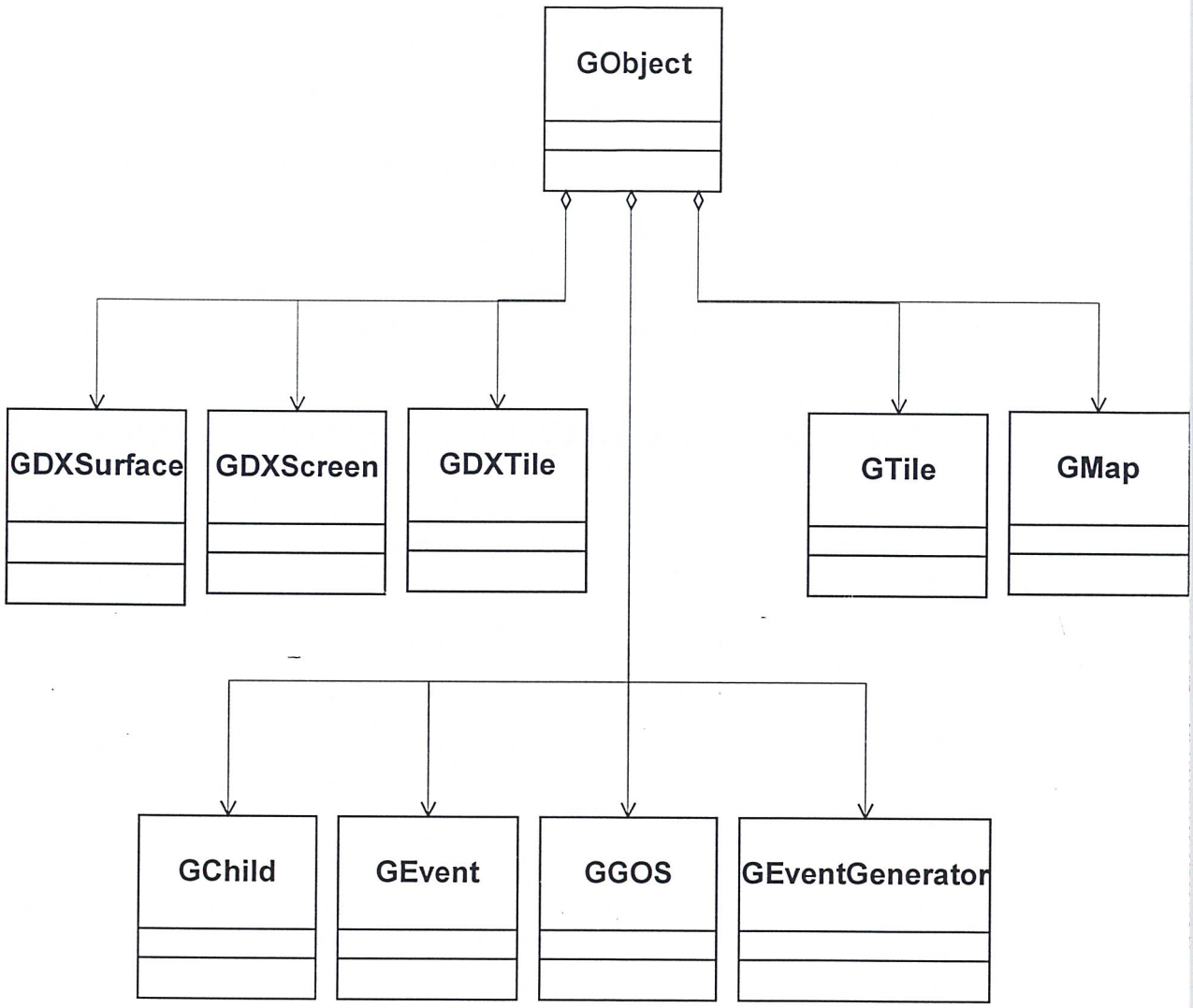
7.2 ส่วนที่ประกอบไปด้วยคลาสที่มีการทำงานระดับบนของ Game Runtime ที่ไม่เฉพาะเจาะจงในแต่ละประเภทของเกม

ในส่วนนี้จะประกอบไปด้วยคลาสที่มีหน้าที่การทำงานและการใช้งานที่อยู่ในระดับบนของ Game Runtime ซึ่งการออกแบบคลาสต่างๆที่อยู่ในส่วนนี้ทำให้เราสามารถมองเห็นคลาสเหล่านั้นเป็นองค์ประกอบต่างๆที่สำคัญต่อการทำงานของเกม และถือได้ว่าเป็นคลาสหลักที่จำเป็นต้องมีในการทำงานของเกมอีกด้วย เช่น คลาสของผู้เล่น คลาสของฉากและแผนที่ คลาสของวัตถุต่างๆที่อยู่ในฉาก และคลาสของระบบการทำงานของเกม คลาสของเหตุการณ์ ซึงคลาสต่างๆเหล่านี้จะถูกนำไปใช้งานในการทำงานของเกมที่ไม่เฉพาะเจาะจงลงไปในแต่ละประเภทของเกมไม่ว่าจะเป็นเกมประเภทใหนก็จำเป็นที่จะต้องมีองค์ประกอบของคลาสเหล่านี้อยู่ เพื่อให้การทำงานหลักของเกมทุกๆประเภทมีลักษณะที่เป็นรูปแบบและมาตรฐานการออกแบบโครงสร้างที่เหมือนกัน โดยคลาสต่างๆในส่วนนี้ได้ถูกวิเคราะห์และพิจารณาเป็นอย่างดีเพื่อที่จะให้เป็นแกนสำคัญในการพัฒนาเกมในรูปแบบอื่นๆต่อไป ทำให้การพัฒนาเกมในรูปแบบอื่นๆเป็นไปได้ง่ายและรวดเร็วมากยิ่งขึ้น

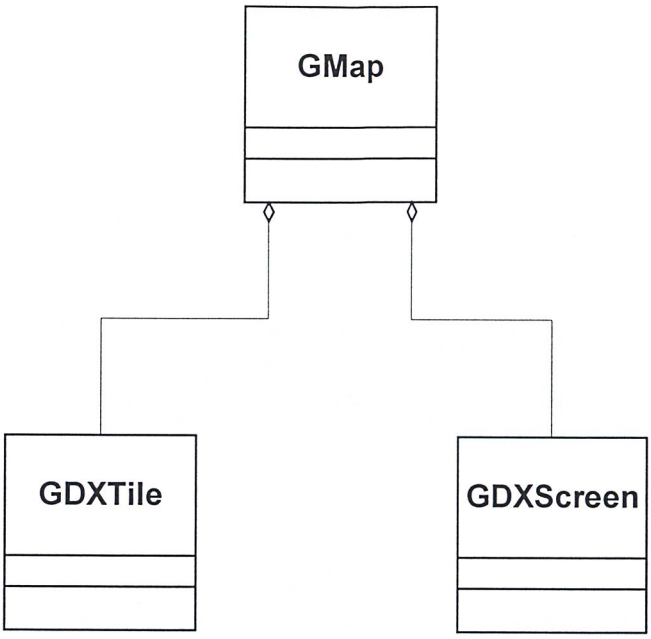
ในขึ้นตอนการออกแบบคลาสต่างๆที่อยู่ในส่วนนี้จะต้องวิเคราะห์การทำงานหลักๆของเกมว่ามีอะไรบ้าง มีการทำงานอย่างไร ทำไม และเมื่อไรถึงต้องมีการเรียกใช้งานคลาสต่างๆเหล่านี้ โดยแบ่งการทำงานหลักๆของเกมออกเป็นกลุ่มย่อยๆ และแยกพิจารณาว่าในการทำงานของเกมนั้นจำเป็นต้องมีองค์ประกอบอะไรบ้าง มีการติดต่อสื่อสารกันระหว่างวัตถุต่างๆที่มีอยู่ในเกมอย่างไร การทำงานหลักของวัตถุต่างๆที่มีอยู่ในเกม ซึ่งการวิเคราะห์และพิจารณาสิ่งต่างๆเหล่านี้จะทำให้เราสามารถออกแบบและกำหนดโครงสร้างการทำงานของคลาสต่างๆที่จำเป็นในขั้นพื้นฐานได้ และจากคลาสพื้นฐานซึ่งไม่ได้ขึ้นอยู่กับประเภทของเกมนั้นก็จะถูกนำไปพัฒนาหรือนำไปประยุกต์เพื่อให้มีโครงสร้างการทำงานที่เฉพาะเจาพจงลงไปในแต่ละประเภทของเกมมากยิ่งขึ้น โดยอาจจะมีการกำหนดคลาสที่สืบทอดคลาสหลักลงไปอีก เพื่อทำให้การทำงานของคลาสที่สืบทอดลงไปนั้นมีความเฉพาะเจาะจงในการทำงานและการใช้งานยิ่งขึ้น

เนื่องจากคลาสต่างๆที่อยู่ในกลุ่มนี้มีการทำงานที่อยู่ในระดับสูงสุดของ Game Runtime ซึ่งค่อนข้างที่จะเป็นนามธรรม (Abstract) และยังไม่สามารถมองเห็นถึงการทำงานที่แท้จริงของเกมได้ จึงจำเป็นที่จะต้องมีการสร้างคลาสต่างๆที่กำหนดเจาะจงการทำงานของเกมในรูปแบบต่างๆที่ต้องการลงไปว่าจะให้เกมที่ต้องการนั้นมีลักษณะอย่างไร มีข้อมูลอะไรบ้างที่จำเป็นต้องใช้ในการทำงานของเกม มีวัตถุอะไรเพิ่มเติมหรือสืบทอดลงไปอีกเพื่อให้เป็นเกมที่ต้องการ ซึ่งคลาสต่างๆที่ออกแบบมาเพื่อระบุประเภทของเกมนี้

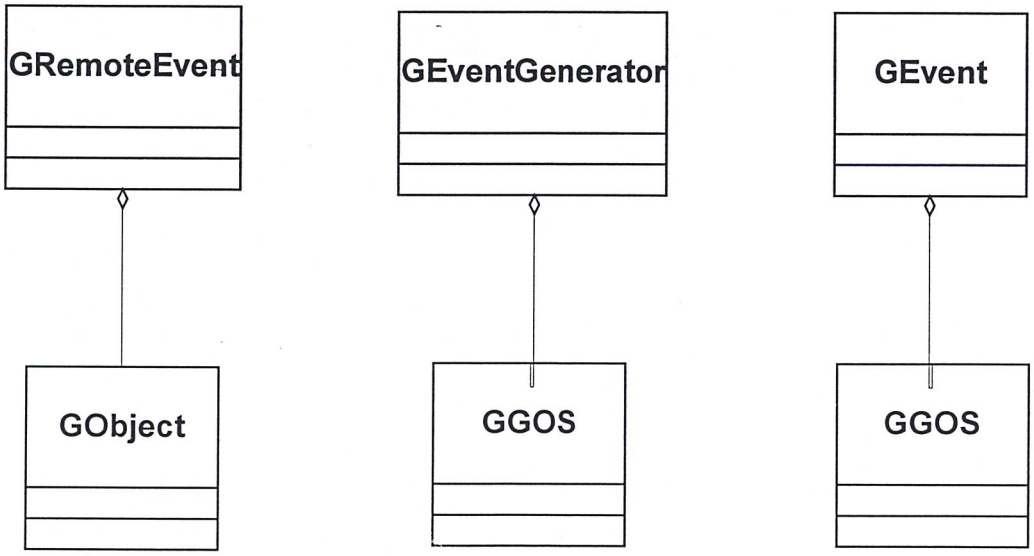
จะกล่าวในหัวข้อที่ 3 อีกทีหนึ่ง และคลาสที่ออกแบบไว้ในส่วนนี้นั้นมีดังนี้คือ GObject Class,GStaticObject Class ,GDynamicObject Class , GPlayer Class ,GChild Class, GEventGenerator Class, GRemoteEvent Class, GEvent Class, GMap Class และ GGOS Class และแสดงความสัมพันธ์ของคลาสต่างๆที่ประกอบอยู่ในคลาสเหล่านี้ได้ดังรูปข้างล่างนี้



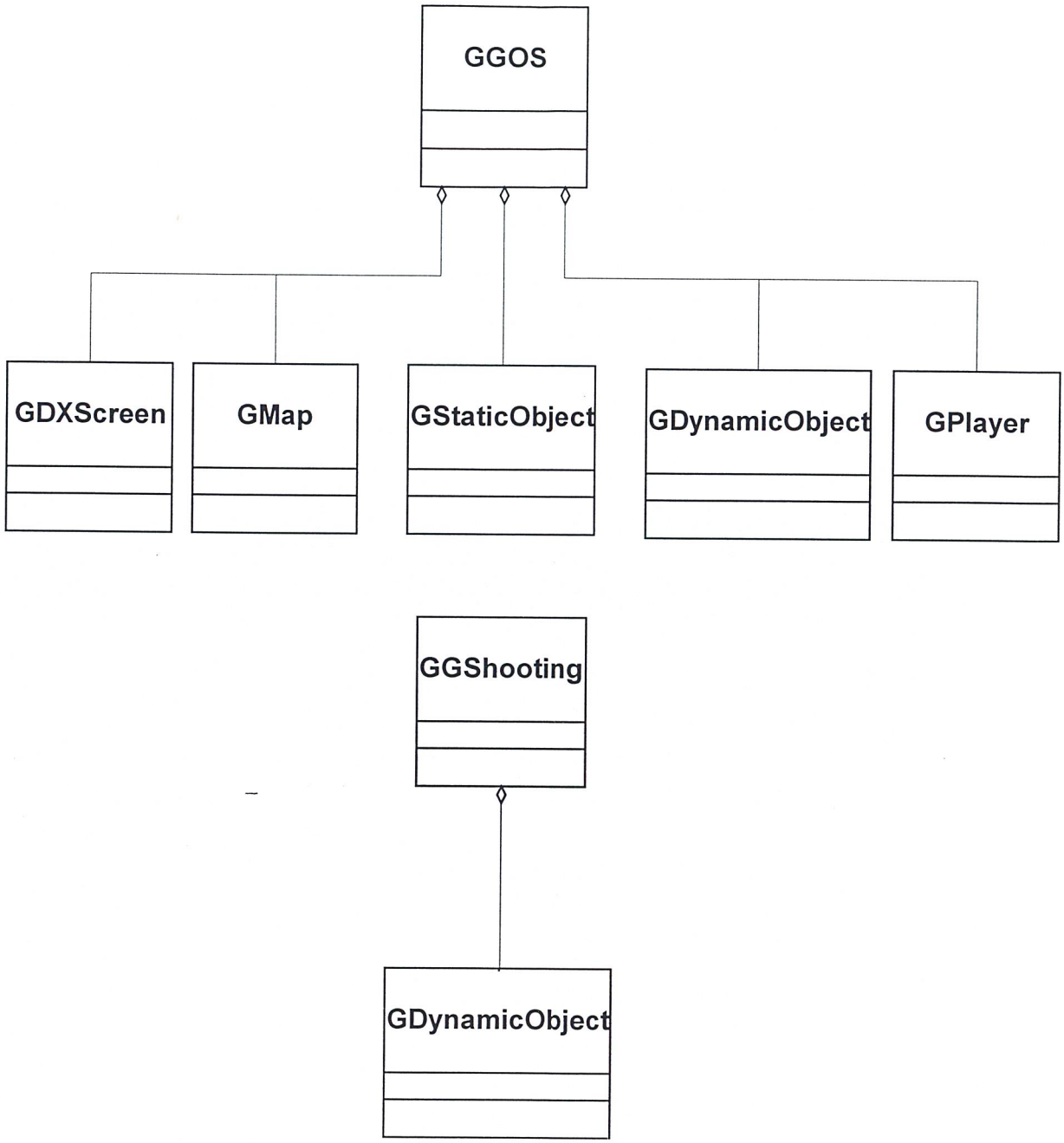
รูปที่7.4 แสดงองค์ประกอบของคลาส GObject (Class Aggregation)



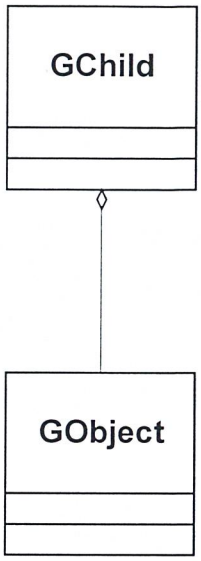
รูปที่7.5 แสดงองค์ประกอบของคลาส GMap (Class Aggregation)



รูปที่7.6 แสดงองค์ประกอบของคลาส GEvent GEventGenerator และ GRemoteEvent (Class Aggregation)



รูปที่7.7 แสดงองค์ประกอบของคลาส GGOS และ GGShooting (Class Aggregation)



รูปที่7.8 แสดงองค์ประกอบของคลาส GChild (Class Aggregation)

7.2.1 GObject Class

เป็นคลาสที่ใช้ในการกำหนดและสร้างวัตถุต่างๆที่มีอยู่ในเกมโดยแนวคิดที่ว่า ทุกๆสิ่งที่อยู่ในเกมที่ไม่ใช้ฉากหรือแผนที่จะเป็นวัตถุ และวัตถุนั้นก็จะต้องมีคุณสมบัติของแต่ละตัว รวมทั้งมีการทำงานที่แตกต่างกันไป โดยสิ่งเหล่านี้สามารถที่จะกำหนดได้ในคลาสที่สืบทอดคลาสนี้ลงไป โดยในคลาสนี้จะไม่ได้บ่งบอกถึงชนิดของวัตถุที่แท้จริง และไม่ได้เจาะจงว่าจะเป็นวัตถุประเภทใด เป็นเสมือนวัตถุที่เป็นนามธรรม และเป็นต้นแบบของการนำไปใช้งานต่อไป

โดยคลาส GObject นี้ได้ออกแบบมาเพื่อการเก็บข้อมูลที่จำเป็นต่อการทำงานหลักๆ ที่ต้องมีในวัตถุทุกๆวัตถุในเกม และมีฟังก์ชันการทำงานต่างๆที่เป็นแกนสำคัญที่ทำให้วัตถุทุกๆวัตถุทำงานได้ นอกจากนี้ยังมีโครงสร้างการทำงานของการควบคุมการทำงานวัตถุแบบเหตุการณ์กระตุ้น(Event Driven Control) อีกด้วยซึ่งเมื่อมีการกำหนดเหตุการณ์ต่างๆที่วัตถุสามารถจะสร้างหรือสามารถที่จะรับเพื่อนำไปทำงานต่อได้ กากำหนดเหตุการณ์เหล่านี้ทำให้วัตถุต่างๆที่มีอยู่ในเกมสามารถที่จะมีการติดต่อสื่อสารกันได้ โดยการส่ง RemoteEvent Object ไปให้ยังอีกวัตถุหนึ่งที่ได้ขอไว้ในตอนแรกซึ่งได้กล่าวถึงรายละเอียดของการทำงานในส่วนของการควบคุมวัตถุแบบกระตุ้นเหตุการณ์นี้แล้วในบทของ “เหตุการณ์ในเกม (Event In Game)”

นอกจากนี้วัตถุยังสามารถมีความสัมพันธ์กับวัตถุอื่นๆได้ในรูปแบบของพ่อ (Parent) กับลูก (Child) ซึ่งเพิ่มประสิทธิภาพในการทำงานของวัตถุขึ้นไปอีกระดับหนึ่งทำให้เราสามารถที่จำกำหนดและควบคุมการทำงานของวัตถุที่มีความสัมพันธ์นี้ได้มากยิ่งขึ้น ซึ่งได้กล่าวถึงรายละเอียดของวัตถุและการทำงานของวัตถุนี้อีกทีแล้วในบทของ “วัตถุในเกม (Object In Game)”

ในคลาส GObject นี้ได้มีการเก็บและใช้งานคลาสต่างๆที่จำเป็นต่อการทำงานของวัตถุมากมาย ซึ่งล้วนแล้วแต่เป็นส่วนประกอบที่ได้สร้างขึ้นจากคลาสที่อยู่ในส่วนของหัวข้อที่ 1, 2 และในส่วนนี้ดังที่จะกล่าวต่อไปในคลาสอื่นๆ โดย GObject Class นี้มีฟังก์ชันการทำงานที่สำคัญดังนี้

1. ฟังก์ชันที่ใช้ในการแสดงผลของวัตถุ ซึ่งจะมีการเรียกใช้ฟังก์ชันการทำงานของ GTile Class อีกทีหนึ่ง
2. ฟังก์ชันที่ใช้ในการสร้างวัตถุนี้ขึ้นมาโดยสร้างจาก GObject object ด้วยกันเองหรือก็คือการทำ copy constructure นั่นเอง
3. void SetDefault(GDXSurface *lpDDS,GMap *gdxmap,LPRECT ClipRect) เป็นฟังก์ชันการทำงานที่ใช้ในการกำหนดค่าเริ่มต้นของพื้นผิวการแสดงผล คลาสของฉากและแผนที่ และขนาดของหน้าจอแสดงผล
4. void SetFrame(int frame) เป็นฟังก์ชันการทำงานที่ใช้ในการกำหนดเลขลำดับของเฟรมซึ่งเป็นภาพย่อยที่เก็บอยู่ในวัตถุนี้
5. void SetMapID(int map) เป็นฟังก์ชันการทำงานที่ใช้ในการกำหนดหมายเลขของฉากหรือแผนที่ที่วัตถุนี้อยู่
6. int GetMapID(void) เป็นฟังก์ชันการทำงานที่ใช้ในการนำหมายเลขฉากหรือแผนที่ที่วัตถุนี้อยู่ออกมาใช้งาน
7. int GetClassID() เป็นฟังก์ชันการทำงานที่นำหมายเลขคลาสของวัตถุนี้ออกมาใช้งาน
8. void SetPos(int x,int y,int z) เป็นฟังก์ชันการทำงานที่ใช้ในการกำหนดตำแหน่งของวัตถุนี้ โดยมีทั้ง X,Y และ Z ซึ่ง Z เป็นระดับความสูงของวัตถุ
9. int GetPosX() เป็นฟังก์ชันการทำงานที่ใช้ในการนำค่าตำแหน่งของวัตถุนี้ในแนวแกน X
10. int GetPosY() เป็นฟังก์ชันการทำงานที่ใช้ในการนำค่าตำแหน่งของวัตถุนี้ในแนวแกน Y
11. int GetPosZ() เป็นฟังก์ชันการทำงานที่ใช้ในการนำค่าตำแหน่งของวัตถุนี้ในแนวแกน Z
12. int GetScreenPosX() เป็นฟังก์ชันการทำงานที่ใช้ในการนำค่าตำแหน่งของวัตถุนี้ในแนวแกน X เมื่อเทียบกับจอภาพ
13. int GetScreenPosY() เป็นฟังก์ชันการทำงานที่ใช้ในการนำค่าตำแหน่งของวัตถุนี้ในแนวแกน Y เมื่อเทียบกับจอภาพ
14. void SetPosX(int x) เป็นฟังก์ชันการทำงานที่ใช้ในการกำหนดตำแหน่งของวัตถุในแนวแกน X
15. void SetPosY(int y) เป็นฟังก์ชันการทำงานที่ใช้ในการกำหนดตำแหน่งของวัตถุในแนวแกน Y
16. void SetPosZ(int z) เป็นฟังก์ชันการทำงานที่ใช้ในการกำหนดตำแหน่งของวัตถุในแนวแกน Z

17. void SetColorKey(int colorkey) เป็นฟังก์ชันการทำงานที่ใช้ในการกำหนดค่าสีที่โปร่งใสของวัตถุนี้
18. bool LoadFromFile(const char *filename) เป็นฟังก์ชันการทำงานที่ใช้ในการ load เอาข้อมูลที่ใช้ในคลาสนี้ทั้งหมดขึ้นมาใช้งาน
19. bool LoadToolFromFile(const char *filename) เป็นฟังก์ชันการทำงานที่ใช้ในการ load เอาข้อมูลที่ใช้ในคลาสนี้ทั้งหมดขึ้นมาใช้งานสำหรับโปรแกรมสร้างและพัฒนาเกมสำเร็จรูป
20. bool SaveToFile (const char *filename) เป็นฟังก์ชันการทำงานที่ใช้ในการเก็บข้อมูลทั้งหมดที่จำเป็นในคลาสนี้ลงไปในแฟ้มข้อมูล
21. void SetID(int id) เป็นฟังก์ชันการทำงานที่ใช้ในการกำหนดเลขประจำตัวของวัตถุนี้
22. int GetID() เป็นฟังก์ชันการทำงานที่ใช้ในการนำข้อมูลที่เป็นเลขประจำตัวของวัตถุนี้ออกมาใช้งาน
23. int GetChildID() เป็นฟังก์ชันการทำงานที่ใช้ในการนำค่าประจำตัวของวัตถุนี้ออกมาใช้งานในกรณีที่วัตถุนี้เป็นลูก
24. int GetParentID() เป็นฟังก์ชันการทำงานที่ใช้ในการนำเลขประจำตัวของพ่อออกมาใช้งานในกรณีที่วัตถุนี้เป็นลูก
25. void SetParentID(int id) เป็นฟังก์ชันการทำงานที่ใช้ในการกำหนดหมายเลขประจำตัวของพ่อให้กับวัตถุนี้
26. bool Hit(GObject *gobject) เป็นฟังก์ชันการทำงานที่ใช้ในการตรวจสอบว่าวัตถุนี้มีส่วนทับกันกับวัตถุอื่นหรือไม่
27. bool InitialChildObject() เป็นฟังก์ชันการทำงานที่ใช้ในการสร้างวัตถุที่เป็นลูกสำหรับวัตถุนี้
28. void SetMainChild(bool b) เป็นฟังก์ชันการทำงานที่ใช้ในการกำหนดให้วัตถุนี้เป็นต้นแบบของวัตถุที่เป็นลูก และเมื่อมีการกำหนดให้วัตถุนี้มีลูกแล้วจะต้องมีการเรียกใช้ฟังก์ชันในข้อ 29 - 37
29. void SetChildPositionAsParent(bool b) เป็นฟังก์ชันการทำงานที่ใช้ในการกำหนดให้วัตถุนี้เกิดขึ้นในตำแหน่งที่เป็นตำแหน่งสัมพัทธ์กับพ่อ
30. void SetRelPosX(int x) เป็นฟังก์ชันการทำงานที่ใช้ในการกำหนดตำแหน่งในแนวแกน X ของวัตถุนี้เทียบกับตำแหน่งของพ่อ
31. void SetRelPosY(int y) เป็นฟังก์ชันการทำงานที่ใช้ในการกำหนดตำแหน่งในแนวแกน Y ของวัตถุนี้เทียบกับตำแหน่งของพ่อ
32. void SetDieAsParent(bool b) เป็นฟังก์ชันการทำงานที่ใช้ในการกำหนดว่าถ้าพ่อตายแล้วจะให้วัตถุนี้ซึ่งเป็นลูกตายด้วยหรือไม่ และต้องมีการเรียกใช้ฟังก์ชันในข้อที่ 33 ด้วย
33. void SetDestroyOnDie(bool b) เป็นฟังก์ชันการทำงานที่ใช้ในกำหนดว่าเมื่อวัตถุนี้ตายจะให้ทำลายทิ้งหรือไม่ ในกรณีที่วัตถุนี้เป็นลูก

34. void SetTotalChild(int count) เป็นฟังก์ชันการทำงานที่ใช้ในการกำหนดจำนวนของลูกที่จะสร้างได้ทั้งหมดสำหรับวัตถุนี้
35. void SetRedupable(bool b) เป็นฟังก์ชันการทำงานที่ใช้ในการกำหนดว่าถ้าวัตถุนี้ตายแล้วจะให้สร้างใหม่ได้หรือไม่
36. void SetCreateActive(bool b) เป็นฟังก์ชันการทำงานที่ใช้ในการกำหนดว่าถ้าวัตถุที่เป็นลูกนี้ถูกสร้างขึ้นแล้วจะทำงานทันทีหรือไม่
37. void SetCreateVisible(bool b) เป็นฟังก์ชันการทำงานที่ใช้ในการกำหนดว่าถ้าวัตถุที่เป็นลูกนี้ถูกสร้างขึ้นแล้วจะแสดงให้เห็นทันทีหรือไม่
38. bool isMainChild() เป็นฟังก์ชันการทำงานที่ใช้ในการตรวจสอบว่าวัตถุนี้เป็นวัตถุต้นแบบของลูกหรือไม่
39. bool isCreatePositionAsParent() เป็นฟังก์ชันการทำงานที่ใช้ในการตรวจสอบว่าวัตถุนี้สร้างขึ้นแล้วจะมีตำแหน่งเป็นตำแหน่งสัมพัทธ์กับพ่อหรือไม่
40. bool isDieAsParent() เป็นฟังก์ชันการทำงานที่ใช้ในการตรวจสอบว่าวัตถุนี้จะตายทันทีเมื่อพ่อตายหรือไม่
41. int GetTotalChild() เป็นฟังก์ชันการทำงานที่ใช้ในการนำจำนวนลูกที่วัตถุนี้สร้างขึ้นออกมาใช้งาน
42. bool isRedupable() เป็นฟังก์ชันการทำงานที่ใช้ในการตรวจสอบว่าวัตถุนี้สามารถสร้างลูกขึ้นมาใหม่เมื่อลูกตายแล้วได้หรือไม่
43. void SetGameOS(GGOS *ggos) เป็นฟังก์ชันการทำงานที่ใช้ในการกำหนด GGOS Object ซึ่งเป็น Object ที่กำหนดและควบคุมการทำงานทั้งหมดของเกม
44. GGOS *GetGameOS() เป็นฟังก์ชันการทำงานที่ใช้ในการนำ GGOS Object ที่ถูกกำหนดไว้ในวัตถุนี้ออกมาใช้งาน
45. void Run() เป็นฟังก์ชันการทำงานที่ใช้ในการทำให้วัตถุนี้ทำงานเองได้ โดยจะมีการทำงานย่อยๆในส่วนต่างๆเช่น การทำงานของ Event Generator การทำงานของ Script และอื่นๆ
46. void SetEventGeneratorNumbers(int num) เป็นฟังก์ชันการทำงานที่ใช้ในการกำหนดจำนวนเหตุการณ์ว่าวัตถุนี้จะสามารถกำเนิดเหตุการณ์ได้ทั้งหมดกี่เหตุการณ์
47. void SetEventNumbers(int num) เป็นฟังก์ชันการทำงานที่ใช้ในการกำหนดจำนวนเหตุการณ์ที่จะให้วัตถุนี้รับเหตุการณ์จากวัตถุอื่นตามที่ต้องการ
48. bool isEventGenerator() เป็นฟังก์ชันการทำงานที่ใช้ในการตรวจสอบว่าวัตถุนี้เป็นวัตถุประเภทกำเนิดเหตุการณ์หรือไม่
49. bool isRemoteEventListener() เป็นฟังก์ชันการทำงานที่ใช้ในการตรวจสอบว่าวัตถุนี้เป็นวัตถุประเภทรับเหตุการณ์หรือไม่
50. void Notify(GRemoteEvent *remoteEvent) เป็นฟังก์ชันการทำงานที่ใช้ในการบอกว่ามีเหตุการณ์เกิดขึ้นแล้วและเหตุการณ์ที่เกิดขึ้นจะถูกส่งมาเป็น GRemoteEvent Object

51. bool RunGenerator() เป็นฟังก์ชันการทำงานที่ใช้ในการกำหนดให้เริ่มการทำงานของวัตถุนี้ในกรณีที่เป็นวัตถุที่กำเนิดเหตุการณ์ได้

7.2.2 GStaticObject Class

เป็นคลาสที่ใช้ในการกำหนดและสร้างวัตถุที่ไม่สามารถเคลื่อนที่ได้ ซึ่งเป็นวัตถุที่เวลานำมาใส่ในเกมแล้วก็จะเป็นแค่วัตถุที่แสดงบนฉากหรือแผนที่เฉยๆ ไม่สามารถทำให้มันเคลื่อนที่ได้ แต่อาจจะมีการเปลี่ยนแปลงตำแหน่งได้อันเนื่องมาจากการกำหนดค่าตำแหน่งใหม่ให้กับวัตถุนี้ ซึ่งไม่ใช่การเคลื่อนที่แต่อย่างใด ในคลาสนี้จะไม่มีคุณสมบัติหรือฟังก์ชันการทำงานที่เพิ่มเติมไปจาก GObject Class แต่ประการใด เพียงแต่ออกบแบบและกำหนดมาเพื่อให้มองวัตถุในเกมดูเป็นรูปธรรมมากยิ่งขึ้นแต่เมื่อไรก็ตามที่ต้องการจะสร้างคลาสของวัตถุที่ไม่สามารถเคลื่อนที่ได้ที่มีคุณสมบัติเพิ่มเติม หรือมีฟังก์ชันการทำงานที่แตกต่างออกไปจากคลาส GObject นั้นก็เพียงแต่ทำการสืบทอดจาก GObject Class เพื่อนำไปใช้งานต่อเท่านั้นเอง

7.2.3 GDynamicObject Class

เป็นคลาสที่ใช้ในการกำหนดและสร้างวัตถุที่สามารถเคลื่อนที่ได้ ซึ่งเป็นวัตถุที่เวลานำมาใส่ในเกมแล้ววัตถุนี้จะมีการเคลื่อนที่ไปมาบนฉากหรือแผนที่ตามที่กำหนดไว้ได้ ซึ่งในคลาสนี้จะคุณสมบัติและฟังก์ชันการทำงานเพิ่มเติมไปจากคลาส GObject ที่ได้สืบทอดมา เพื่อเป็นวัตถุที่สามารถเคลื่อนที่ได้ตามที่ต้องการ นอกจากนี้คลาส GDynamicObject ยังมีการ load และ save แฟ้มข้อมูลของตนเองเพื่อใช้สำหรับข้อมูลที่ได้เพิ่มเข้าไป แต่ก็ยังคงใช้ฟังก์ชันการทำงานพื้นฐานได้จากคลาส GObject โดยในคลาส GDynamicObject Class นี้มีฟังก์ชันการทำงานที่สำคัญดังนี้

1. ฟังก์ชันที่ใช้ในการสร้าง GDynamic Object ขึ้นมาซึ่งเป็นการเรียกใช้ฟังก์ชันของ GObject Class อีกทีหนึ่ง
2. void SetVelX(int velx) เป็นฟังก์ชันการทำงานที่ใช้ในการกำหนดความเร็วของวัตถุนี้ในแนวแกน X
3. void SetVelY(int vely) เป็นฟังก์ชันการทำงานที่ใช้ในการกำหนดความเร็วของวัตถุนี้ในแนวแกน Y
4. void SetLife(int life) เป็นฟังก์ชันการทำงานที่ใช้ในการกำหนดพลังชีวิตของวัตถุนี้
5. void SetType(DYNAMICOBJTYPE type) เป็นฟังก์ชันการทำงานที่ใช้ในการกำหนดชนิดของวัตถุนี้ว่าเป็นประเภทใด โดยมีชนิดดังนี้คือ DO_ENEMY , DO_NEUTRAL ,DO_PLAYER
6. int GetVelX() เป็นฟังก์ชันการทำงานที่ใช้ในการนำค่าตำแหน่งของวัตถุนี้ในแนวแกน X ออกมาใช้งาน
7. int GetVelY() เป็นฟังก์ชันการทำงานที่ใช้ในการนำค่าตำแหน่งของวัตถุนี้ในแนวแกน Y

ออกมาใช้งาน

8. DYNAMICOBJTYPE GetType() เป็นฟังก์ชันการทำงานที่ใช้ในการนำค่าชนิดของวัตถุนี้ออกมาใช้งาน
9. int GetClassID() เป็นฟังก์ชันการทำงานที่ใช้ในการนำเลขประจำคลาสนี้ออกมาใช้งาน
10. bool LoadFromFile(const char *filename) เป็นฟังก์ชันการทำงานที่ใช้ในการ load ข้อมูลที่ใช้ในคลาสนี้จากแฟ้มข้อมูล
11. bool SaveToFile(const char *filename) เป็นฟังก์ชันการทำงานที่ใช้ในการเก็บข้อมูลที่ใช้ในคลาสนี้ลงไปในแฟ้มข้อมูล
12. bool LoadToolFromFile(const char *filename)) เป็นฟังก์ชันการทำงานที่ใช้ในการ load ข้อมูลที่ใช้ในคลาสนี้จากแฟ้มข้อมูลเพื่อใช้ในโปรแกรมสร้าง และพัฒนาเกมสำเร็จรูป
13. void MoveUp(int count) เป็นฟังก์ชันการทำงานที่ใช้ในการเคลื่อนที่วัตถุนี้ขึ้นไปเป็นจำนวนที่ต้องการ
14. void MoveDown(int count) เป็นฟังก์ชันการทำงานที่ใช้ในการเคลื่อนที่วัตถุนี้ลงไปเป็นจำนวนที่ต้องการ
15. void MoveLeft(int count) เป็นฟังก์ชันการทำงานที่ใช้ในการเคลื่อนที่วัตถุนี้ไปทางซ้ายเป็นจำนวนที่ต้องการ
16. void MoveRight(int count) เป็นฟังก์ชันการทำงานที่ใช้ในการเคลื่อนที่วัตถุนี้ไปทางขวาเป็นจำนวนที่ต้องการ

7.2.4 GPlayer Class

เป็นคลาสที่ใช้ในการกำหนดและสร้างวัตถุที่เป็นผู้เล่น โดยที่คลาสนี้จะไม่มีคุณสมบัติและฟังก์ชันการทำงานเพิ่มเติมไปจาก Object Class แต่อย่างใดเพียงแต่ได้ทำการออกแบบมาเพื่อให้สามารถมองคลาสนี้ออกเป็นผู้เล่นได้ชัดเจนยิ่งขึ้น แต่คลาสนี้เป็นคลาสที่สำคัญในการอ้างถึงในส่วนของการทำงานหลักของเกมซึ่งในการทำงานที่ติดต่อรับข้อมูลกับอุปกรณ์รับข้อมูลเช่น keyboard แล้วจะต้องใช้ GPlayer Object ในการอ้างเท่านั้น ซึ่งมีความจำเป็นที่จะต้องกำหนดและสร้างคลาสนี้ขึ้นมาอย่างน้อย 1 Object ในเกมและในการทำงานของเกมนั้นจะมีการควบคุม GPlayer Object นี้ให้เป็นไปตามที่ผู้เล่นต้องการโดยถ้ามี GPlayer Object 1 Object ก็แสดงว่าเป็นเกมที่สามารถเล่นได้ 1 คนหรือถ้ากำหนดเป็น 2 Object ก็สามารถเล่นได้ 2 คน ซึ่งขึ้นอยู่กับประเภทของเกมด้วย

7.2.5 GChild Class

เป็นคลาสที่ใช้ในการกำหนดและสร้างวัตถุที่เป็นวัตถุลูกสำหรับวัตถุใดๆก็ตามที่มีการกำหนดให้เป็นวัตถุต้นแบบของการสร้างวัตถุลูก โดยคลาสนี้จะถูกใช้ในการสร้างและควบคุมวัตถุที่เป็นวัตถุลูกทั้งหมดที่ถูกสร้างขึ้นโดยวัตถุต้นแบบนั้นๆ ซึ่งในคลาสนี้จะมีวัตถุที่เป็นวัตถุลูกเก็บอยู่ โดย GChild Class นี้มีฟังก์ชันการทำงานที่สำคัญดังนี้

1. ฟังก์ชันที่ใช้ในการสร้าง GChild Object นี้ขึ้นมาทำงานเพื่อใช้ในการกำหนดและสร้างวัตถุลูกต่อไป
2. void Show(int childID) เป็นฟังก์ชันการทำงานที่ใช้ในการแสดงวัตถุลูกใดๆบนหน้าจอ
3. void SetActive(int childID) เป็นฟังก์ชันการทำงานที่ใช้ในการกำหนดให้วัตถุลูกใดๆเริ่มทำงาน
4. void SetMainActive(bool b) เป็นฟังก์ชันการทำงานที่ใช้ในการกำหนดให้ GChidl Object นี้เริ่มทำงานเพื่อที่จะทำให้วัตถุลูกทั้งหมดที่อยู่ในคลาสนี้ทำงานได้
5. bool Run(int childID,bool active,bool visible) เป็นฟังก์ชันการทำงานที่ใช้ในการเริ่มการทำงานของวัตถุลูกใดๆในคลาสนี้ซึ่งสามารถกำหนดว่าเมื่อเริ่มการทำงานแล้วจะให้ ทำงานหรือแสดงผลทันทีหรือไม่ก็ได้
6. bool Stop(int childID) เป็นฟังก์ชันการทำงานที่ใช้ในการกำหนดให้วัตถุลูกใดๆหยุดการทำงาน
7. void Debug(AnsiString filename) เป็นฟังก์ชันการทำงานที่ใช้ในการสร้างแฟ้มข้อมูลที่ใช้ในการแสดงผลการสร้างหรือการทำงานของคลาสนี้

7.2.6 GEventGenerator Class

เป็นคลาสที่ใช้ในการกำหนดและสร้างเหตุการณ์ที่สามารถกำเนิดได้จากวัตถุใดๆที่กำหนดเป็นวัตถุที่สามารถกำเนิเหตุการณ์ได้โดยวัตถุที่สามารถกำเนิเหตุการณ์ได้นั้นจะต้องมีการกำหนดค่าต่างๆที่ต้องมีในคลาสนี้เพื่อให้วัตถุนั้นสามารถกำเนิดเหตุการณ์ได้ตามต้องการ และในการส่งเหตุการณ์ที่เกิดขึ้นนั้นไปยังวัตถุเป้าหมายได้โดยการส่ง GRemoteEvent Object ไป โดยหลักการการทำงานของการกำเนิดเหตุการณ์ในคลาสนี้นั้น ได้กล่าวในบทของ “เหตุการณ์ในเกม(Event In Game)” และคลาสนี้เป็นส่วนประกอบที่สำคัญในคลาส GObject และถูกนำไปใช้งานต่อไป โดย GEventGenerator Class นี้มีฟังก์ชันการทำงานที่สำคัญดังนี้

1. GRemoteEvent *MakeRemoteEvent(int eventID,EVENTMESSAGE msg,MESSAGEPARAM msgParam,GObject *source,GObject *handback) เป็นฟังก์ชันการทำงานที่ใช้ในการสร้าง GRemoteEvent Object ขึ้นมาเพื่อส่งไปยังวัตถุเป้าหมายที่ต้องการรับเหตุการณ์นี้
2. GENERATORRESULT CheckGeneratorMsg(GObject *self,EVENTGENERATOR eventGenerator) เป็นฟังก์ชันการทำงานที่ใช้ในการตรวจสอบวัตถุที่สามารถกำเนิดเหตุการณ์นั้นว่าเหตุการณีที่ต้องการจะให้สร้างนั้นสามารถที่จะกำเนิดได้หรือยังโดยจะตรวจสอบจากฟังก์ชันที่ได้กำหนดไว้แล้วว่าเมื่อไรจึงจะสร้างเหตุการณ์ที่กำหนดได้ และเมื่อสร้างเหตุการณ์แล้วก็จะถูกส่งไปยังวัตถุเป้าหมายที่ต้องการ

3. bool NotifyRemoteEvent(GRemoteEvent *remoteEvent,EVENTGENERATOR &eventGenerator) เป็นฟังก์ชันการทำงานที่ใช้ในการส่งเหตุการณ์ที่ได้กำเนิดขึ้นแล้วนั้นไปยังวัตถุเป้าหมาย
4. void InitialGenerator(GGOS *ggos,int num) เป็นฟังก์ชันการทำงานที่ใช้ในการกำหนดและสร้างคลาสนี้ขึ้น
5. bool SaveToStream(TMemoryStream *stream) เป็นฟังก์ชันการทำงานที่ใช้ในการเก็บข้อมูลในคลาสนี้ลงในสายข้อมูล
6. bool LoadFromStream(TMemoryStream *stream) เป็นฟังก์ชันการทำงานที่ใช้ในการ load ข้อมูลที่ใช้ในคลาสนี้ขึ้นมาจากสายข้อมูล
7. int GetTotalGenerator() เป็นฟังก์ชันการทำงานที่ใช้ในการนำจำนวนเหตุการณ์ที่คลาสนี้กำเนิดได้ทั้งหมดออกมาใช้งาน
8. void Check(GObject *self) เป็นฟังก์ชันการทำงานที่ใช้ในการตรวจสอบว่าวัตถุที่กำเนิดเหตุการณ์นี้นั้นสามารถสร้างเหตุการณ์ได้หรือไม่ ถ้าสร้างได้ก็จะสร้างและส่งไปยังวัตถุเป้าหมาย
9. และฟังก์ชันการทำงานต่างๆที่มีผลทำให้เกิดเหตุการณ์ใดๆขึ้นซึ่งในส่วนนี้จะต้องมีการกำหนดไว้ล่วงหน้าว่าจะมีเหตุการณ์อะไรที่จะเกิดขึ้นบ้าง และเกิดได้ด้วยสาเหตุอะไร

7.2.7 GRemoteEvent Class

เป็นคลาสที่ใช้ในการกำหนดและสร้างวัตถุเหตุการณ์(Event Object) ขึ้นมาเพื่อใช้ในการส่งวัตถุนี้ไปยังวัตถุเป้าหมายที่ต้องการรับวัตถุเหตุการณ์นี้ โดยคลาสนี้เป็นเสมือนคลาสที่ห่อหุ้มข้อมูลของเหตุการณ์และส่งให้ไปถึงเป้าหมายเท่านั้นเอง โดย GRemoteEvent Class นี้นั้นมีฟังก์ชันการทำงานที่สำคัญดังนี้

1. GRemoteEvent(GObject *source,int eventID,EVENTMESSAGE msg,MESSAGEPARAM MsgParam,GObject *handback) เป็นฟังก์ชันการทำงานที่ใช้ในการสร้างวัตถุเหตุการณ์นี้ขึ้นตามที่กำหนด
2. GObject* GetSource() เป็นฟังก์ชันการทำงานที่ใช้ในการนำ GObject object ที่เป็นตัวกำเนิดเหตุการณ์นี้ขึ้นออกมาใช้งาน
3. int GetMsg() เป็นฟังก์ชันการทำงานที่ใช้ในการนำชนิดของเหตุการณ์ออกมาใช้งาน
4. int GetMsgParam() เป็นฟังก์ชันการทำงานที่ใช้ในการนำค่าพารามิเตอร์ของชนิดของเหตุการณ์นี้ออกมาใช้งาน
5. GObject* GetRegisTrationObject() เป็นฟังก์ชันการทำงานที่ใช้ในการนำวัตถุที่ได้กำหนดไว้เพื่อให้เป็นวัตถุที่ถูกส่งไปให้ทำงานหรือกระทำการตามที่ได้กำหนดไว้
6. int GetID() เป็นฟังก์ชันการทำงานที่ใช้ในการนำเลขประจำตัวของเหตุการณ์นี้ออกมาใช้งาน

7.2.8 GEvent Class

เป็นคลาสที่ใช้ในการกำหนดและสร้างวัตถุที่ใช้ในการรับเหตุการณ์ต่างๆที่เกิดขึ้นจากวัตถุที่กำเนิดเหตุการณ์นั้นๆ ซึ่งเหตุการณ์ที่ต้องการจะรับมาเพื่อการทำงานของวัตถุนั้นๆจะต้องถูกกำหนดไว้ในคลาสนี้ว่าถ้าได้รับเหตุการณ์ใดๆที่ต้องการแล้วจะให้วัตถุที่ได้รับนี้ทำอะไร อย่างไร ซึ่งเป็นเสมือนการกำหนดให้วัตถุทำงานตามที่ต้องการเมื่อเกิดเหตุการณ์ที่ต้องการขึ้น โดยได้กล่าวถึงรายละเอียดในเรื่องหลักการรับเหตุการณ์ไว้ในบทของ “เหตุการณ์ในเกม(Event In Game)” ซึ่งใน GEvent Class นี้มีฟังก์ชันการทำงานที่สำคัญดังนี้

1. bool SaveToStream(TMemoryStream *stream) เป็นฟังก์ชันการทำงานที่ใช้ในการเก็บข้อมูลของคลาสนี้ทั้งหมดลงไปในสายข้อมูล(Stream)
2. void InitialEvent(GGOS *ggos,int num) เป็นฟังก์ชันการทำงานที่ใช้ในการกำหนดและสร้างคลาสนี้ขึ้น
3. bool LoadFromStream(TMemoryStream *stream) เป็นฟังก์ชันการทำงานที่ใช้ในการ load ข้อมูลที่จำเป็นต่อการทำงานในคลาสนี้ขึ้นมาใช้งานจากสายข้อมูล
4. void Notify(GObject *self,GRemoteEvent *remoteEvent) เป็นฟังก์ชันการทำงานที่ใช้ในการรับวัตถุเหตุการณ์(GRemoteEvent Object) จากวัตถุที่กำเนิดเหตุการณ์
5. EVENTRESULT CheckRemoteEvent(GObject *self,GRemoteEvent *remoteEvent,EVENTLISTENER eventListener) เป็นฟังก์ชันการทำงานที่ใช้ในการตรวจสอบว่าวัตถุเหตุการณ์ที่ได้รับมานั้นคืออะไร และต้องทำอะไร
6. EVENTRESULT DoEvent(GObject *self,GRemoteEvent *remoteEvent,EVENTLISTENER eventListener) เป็นฟังก์ชันการทำงานที่ใช้ในการเริ่มการทำงานตามที่ได้กำหนดไว้เมื่อได้รับเหตุการณ์นั้นแล้ว
7. และฟังก์ชันการทำงานต่างๆที่ได้กำหนดไว้เพื่อการทำงานที่ตอบสนองสำหรับเหตุการณ์ที่ได้รับมานั้น โดยในส่วนนี้จะเป็นฟังก์ชันที่ได้กำหนดไว้ล่วงหน้าว่าถ้าเกิดเหตุการณ์อะไร และจะให้ทำอะไร อย่างไร

7.2.9 GMap Class

เป็นคลาสที่ใช้ในการกำหนดและสร้างฉากและแผนที่ที่ใช้ในเกม ซึ่งในคลาสนี้จะเป็นคลาสที่มีหน้าที่สำคัญในส่วนของการแสดงผลฉากหรือแผนที่ที่ได้สร้างมานั้นขึ้นบนจอ และเป็นคลาสที่ใช้ในการอ้างตำแหน่งของวัตถุทุกๆวัตถุอีกด้วย ซึ่งแต่ละวัตถุก็จะมีค่า MapID เก็บไว้เพื่อบอกว่าวัตถุนั้นอยู่บนฉากหรือแผนที่อะไร และนอกจากนี้คลาสนี้ยังมีการกำหนดส่วนหัวของแฟ้มข้อมูลเพื่อใช้ในการเก็บข้อมูลต่างๆที่จำเป็นต่อการใช้งานในคลาสนี้อีกด้วยโดย GMap Class นี้มีฟังก์ชันการทำงานที่สำคัญดังนี้

1. GMap(GDXScreen *Screen) เป็นฟังก์ชันการทำงานที่ใช้ในการสร้างฉากหรือแผนที่ให้กับ GDXScreen Object ที่ได้กำหนดไว้

2. UNIT GetID(void) เป็นฟังก์ชันการทำงานที่ใช้ในการนำเลขประจำตัวของฉากหรือแฟนที่นี้ออกมาใช้งาน
3. char* GetTileFilename(void) เป็นฟังก์ชันการทำงานที่ใช้ในการนำชื่อแฟ้มข้อมูลที่เก็บไทล์ของฉากหรือแผนที่นี้ออกมาใช้งาน
4. int GetTileNum(void) เป็นฟังก์ชันการทำงานที่ใช้ในการนำจำนวนไทล์ทั้งหมดที่ใช้ในฉากหรือแผนที่นี้ออกมาใช้งาน
5. int GetWidth(void) เป็นฟังก์ชันการทำงานที่ใช้ในการนำค่าความกว้างของฉากหรือแผนที่นี้ออกมาใช้งาน
6. int GetHeight(void) เป็นฟังก์ชันการทำงานที่ใช้ในการนำค่าความสูงของฉากหรือแผนที่นี้ออกมาใช้งาน
7. int GetSpeed(void) เป็นฟังก์ชันการทำงานที่ใช้ในการนำค่าความเร็วที่ใช้ในการเลื่อนของฉากออกมาใช้งาน
8. int GetDirection(void) เป็นฟังก์ชันการทำงานที่ใช้ในการนำค่าทศทางของการเลื่อนฉากออกมาใช้งาน
9. SetMapPosX(int PosX) เป็นฟังก์ชันการทำงานที่ใช้ในการกำหนดตำแหน่งเริ่มต้นของฉากหรือแผนที่ในแนวแกน X
10. SetMapPosY(int PosY) เป็นฟังก์ชันการทำงานที่ใช้ในการกำหนดตำแหน่งเริ่มต้นของฉากหรือแผนที่ในแนวแกน Y
11. int GetMapPosX(void) เป็นฟังก์ชันการทำงานที่ใช้ในการนำค่าตำแหน่งในแนวแกน X ของฉากหรือแผนที่นี้ออกมาใช้งาน
12. int GetMapPosY(void) เป็นฟังก์ชันการทำงานที่ใช้ในการนำค่าตำแหน่งในแนวแกน Y ของฉากหรือแผนที่นี้ออกมาใช้งาน
13. void MoveTo(int PosX,int PosY) เป็นฟังก์ชันการทำงานที่ใช้ในการกำหนดตำแหน่งการเริ่มต้นของฉากหรือแผนที่
14. void ScrollUp(int Offset) เป็นฟังก์ชันการทำงานที่ใช้ในการเลื่อนฉากขึ้นเป็นจำนวนที่กำหนด
15. void ScrollDown(int Offset) เป็นฟังก์ชันการทำงานที่ใช้ในการเลื่อนฉากลงเป็นจำนวนที่กำหนด
16. void ScrollLeft(int Offset) เป็นฟังก์ชันการทำงานที่ใช้ในการเลื่อนฉากไปทางซ้ายเป็นจำนวนที่กำหนด
17. void ScrollRight(int Offset) เป็นฟังก์ชันการทำงานที่ใช้ในการเลื่อนฉากไปทางขวาเป็นจำนวนที่กำหนด

18. void WrapScrollUp(int Offset) เป็นฟังก์ชันการทำงานที่ใช้ในการเลื่อนฉากขึ้นไปเป็นจำนวนที่กำหนดและถ้าขึ้นจนสุดฉากแล้วก็จะย้อนกลับมายังจุดเริ่มต้นใหม่เพื่อทำการเลื่อนต่อไปได้
19. void WrapScrollDown(int Offset) เป็นฟังก์ชันการทำงานที่ใช้ในการเลื่อนฉากลงไปเป็นจำนวนที่กำหนดและถ้าขึ้นจนสุดฉากแล้วก็จะย้อนกลับมายังจุดเริ่มต้นใหม่เพื่อทำการเลื่อนต่อไปได้
20. void WrapScrollLeft(int Offset) เป็นฟังก์ชันการทำงานที่ใช้ในการเลื่อนฉากไปทางซ้ายไปเป็นจำนวนที่กำหนดและถ้าขึ้นจนสุดฉากแล้วก็จะย้อนกลับมายังจุดเริ่มต้นใหม่เพื่อทำการเลื่อนต่อไปได้
21. void WrapScrollRight(int Offset) เป็นฟังก์ชันการทำงานที่ใช้ในการเลื่อนฉากไปทางขวาไปเป็นจำนวนที่กำหนดและถ้าขึ้นจนสุดฉากแล้วก็จะย้อนกลับมายังจุดเริ่มต้นใหม่เพื่อทำการเลื่อนต่อไปได้
22. bool LoadFromFile(const char *Filename) เป็นฟังก์ชันการทำงานที่ใช้ในการ load ข้อมูลของฉากหรือแผนที่ที่จำเป็นต่อการใช้งานในคลาสนี้ขึ้นมา

7.2.10 GGOS Class

เป็นคลาสที่ใช้ในการกำหนดและสร้างการทำงานของเกมที่เป็นแกนกลางซึ่งไม่ได้เฉพาะเจาะจงลงไปว่าเป็นการทำงานของเกมประเภทใดแต่เกมทุกประเภทจะต้องมีคลาสนี้เพื่อให้เป็นรูปแบบและมาตรฐานในการสร้างเกมประเภทอื่นๆต่อไป โดยการทำงานของคลาสนี้เปรียบเสมือนการสร้าง และควบคุมระบบการทำงานทั้งหมดของเกม (Game Operating System) ซึ่งสร้างและจัดการวัตถุต่างๆในเกมให้ทำงานได้ การควบคุม และการกำหนดค่าต่างๆที่จำเป็นต่อการทำงานของเกม, การจัดการกับขั้นตอนการแสดงผลบนหน้าจอ, การควบคุมการทำงานของฉากและแผนที่, การรับอินพุท, การ load และ save ข้อมูลต่างๆที่จำเป็นในการทำงานของเกม ซึ่งในคลาสนี้จะประกอบด้วยคลาสที่ใช้ในการทำงานของเกมที่ได้กล่าวมาแล้วทั้งหมด โดย GGOS Class นี้นั้นมีฟังก์ชันการทำงานที่สำคัญดังนี้

1. GGOS(void *hInst,void *hWnd) เป็นฟังก์ชันการทำงานที่ใช้ในการสร้างคลาสนี้จากค่าที่กำหนดให้
2. bool LoadGameSystemEnv(const char *filename) เป็นฟังก์ชันการทำงานที่ใช้ในการ load ข้อมูลที่จำเป็นต่อการทำงานของเกม
3. bool InitialGameSystem() เป็นฟังก์ชันการทำงานที่ใช้ในการกำหนดค่าเริ่มต้นและสร้าง Object ต่างๆที่มีอยู่ในคลาสนี้
4. void UpdatePlayer() เป็นฟังก์ชันการทำงานที่ใช้ในการ update ข้อมูลและการแสดงผลสำหรับผู้เล่นในเกม
5. void UpdateInput()เป็นฟังก์ชันการทำงานที่ใช้ในการ update ข้อมูลจากอุปกรณ์อินพุท เช่น keyboard

6. void UpdateMap() เป็นฟังก์ชันการทำงานที่ใช้ในการ update ข้อมูลและการแสดงผลของฉากและแผนที่
7. void UpdateObjects() เป็นฟังก์ชันการทำงานที่ใช้ในการ update ข้อมูลและการแสดงผลของวัตถุทั้งหมดที่มีอยู่ในเกม
8. int HitObject(GObject *gobject) เป็นฟังก์ชันการทำงานที่ใช้ในการตรวจสอบว่าวัตถุที่ให้ไปนั้นชนกับวัตถุอื่นหรือไม่ ถ้าชนก็จะให้ค่าประจำตัวของวัตถุที่ชนกลับคืนมา
9. void Run() เป็นฟังก์ชันการทำงานที่ใช้ในการทำให้วัตถุต่างๆที่ทำงานด้วยตนเอง
10. void SetWindow(int xmin,int ymin,int xmax,int ymax) เป็นฟังก์ชันการทำงานที่ใช้ในการกำหนดขนาดของการแสดงผล
11. void MakeScreen(int resx,int resy,int bpp) เป็นฟังก์ชันการทำงานที่ใช้ในการสร้างการแสดงผลที่มีขนาดและจำนวนสีตามต้องการ
12. void ReleaseAllMap() เป็นฟังก์ชันการทำงานที่ใช้ในการลบ GMap Object ที่มีอยู่ทั้งหมดในเกม
13. GMap *GetMap(int n) เป็นฟังก์ชันการทำงานที่ใช้ในการนำ GMap Object ที่ต้องการออกมาใช้งาน
14. GMap **GetMapList() เป็นฟังก์ชันการทำงานที่ใช้ในการนำ GMap Object ทั้งหมดออกมาใช้งาน
15. int GetCurrentMap() เป็นฟังก์ชันการทำงานที่ใช้ในการนำมายเลขประจำฉากหรือแผนที่ปัจจุบันขณะเกมทำงานออกมาใช้งาน
16. bool LoadMapFile(const char *filename) เป็นฟังก์ชันการทำงานที่ใช้ในการ load แฟ้มข้อมูลที่เป็นฉากหรือแผนที่ขึ้นมาใช้งาน
17. bool SetCurrentMap(int map) เป็นฟังก์ชันการทำงานที่ใช้ในการกำหนดหมายเลขฉากหรือแผนที่ปัจจุบัน
18. void ShowObject(GObject *object) เป็นฟังก์ชันการทำงานที่ใช้ในการแสดงวัตถุที่ต้องการ
19. GObject *GetObject(int objID) เป็นฟังก์ชันการทำงานที่ใช้ในการนำวัตถุที่ต้องการออกมาใช้งาน
20. int GetDynamicObjectCount() เป็นฟังก์ชันการทำงานที่ใช้ในการนำจำนวน GDynamicObject Object ทั้งหมดที่มีอยู่ในเกมออกมาใช้งาน
21. GDynamicObject **GetDynamicObjectList() เป็นฟังก์ชันการทำงานที่ใช้ในการนำ GDynamicObject object ทั้งหมดในเกมออกมาใช้งาน
22. GDynamicObject *GetDynamicObject(int n) เป็นฟังก์ชันการทำงานที่ใช้ในการนำ GDynamicObject object ที่ต้องการในเกมออกมาใช้งาน
23. void ReleaseAllDynamicObject() เป็นฟังก์ชันการทำงานที่ใช้ในการลบ GDynamicObject object ทั้งหมดในเกม

24. bool LoadDynamicObjectFile(const char *filename) เป็นฟังก์ชันการทำงานที่ใช้ในการ load แฟ้มข้อมูลที่เป็นวัตถุที่สามารถเคลื่อนที่ได้ทั้งหมดในเกมขึ้นมา
25. int GetPlayerObjectCount() เป็นฟังก์ชันการทำงานที่ใช้ในการนำจำนวน GPlayer Object ทั้งหมดที่มีอยู่ในเกมออกมาใช้งาน
26. GPlayer **GetPlayerObjectList() เป็นฟังก์ชันการทำงานที่ใช้ในการนำ GPlayer object ทั้งหมดในเกมออกมาใช้งาน
27. GPlayer *GetPlayerObject(int n) เป็นฟังก์ชันการทำงานที่ใช้ในการนำ GPlayer object ที่ต้องการในเกมออกมาใช้งาน
28. bool LoadPlayerFile(const char *filename) เป็นฟังก์ชันการทำงานที่ใช้ในการ load แฟ้มข้อมูลที่เป็นวัตถุที่เป็นผู้เล่นทั้งหมดในเกมขึ้นมา
29. void ReleaseAllPlayerObject() เป็นฟังก์ชันการทำงานที่ใช้ในการลบ GPlayer object ทั้งหมดในเกม
30. int GetStaticObjectCount()เป็นฟังก์ชันการทำงานที่ใช้ในการนำจำนวน GStaticObject Object ทั้งหมดที่มีอยู่ในเกมออกมาใช้งาน
31. GStaticObject **GetStaticObjectList() เป็นฟังก์ชันการทำงานที่ใช้ในการนำ GStaticObject object ทั้งหมดในเกมออกมาใช้งาน
32. GStaticObject *GetStaticObject(int n) เป็นฟังก์ชันการทำงานที่ใช้ในการนำ GStaticObject object ที่ต้องการในเกมออกมาใช้งาน
33. bool LoadStaticObjectFile(const char *filename) เป็นฟังก์ชันการทำงานที่ใช้ในการ load แฟ้มข้อมูลที่เป็นวัตถุที่ไม่สามารถเคลื่อนที่ได้ทั้งหมดในเกมขึ้นมา
34. void ReleaseAllStaticObject()เป็นฟังก์ชันการทำงานที่ใช้ในการลบ GStaticObject object ทั้งหมดในเกม
35. void RunGenerator() เป็นฟังก์ชันการทำงานที่ใช้ในการทำให้วัตถุที่สามารถกำเนิดเหตุการณ์ได้(Event Generator Object)ทำงานด้วยตนเอง

7.3 ส่วนที่ประกอบไปด้วยคลาสทีมีการทำงานระดับบนของ Game Runtime ที่เฉพาะเจาะจงเกมประเภทยานยิง(Shooting Game)

ในส่วนนี้จะประกอบไปด้วยคลาสที่ทำหน้าที่ในการกำหนดและควบคุมการทำงานของเกมประเภทที่เป็นแนวยานยิงเท่านั้นซึ่งไม่สามารถนำไปใช้ในแกมประเภทอื่นๆได้ และโครงสร้างการทำงานของคลาสในส่วนนี้จะมีการกำหนดเฉพาะเจาะจงลงไปว่าเกมประเภทยานยิงนั้นต้องมีองค์ประกอบอะไรเพิ่มเติมจากคลาสชั้นบนบ้างคลาสต่างๆในส่วนนี้จะเป็นรูปธรรมขัดขึ้นในเกมประเภทแนวยานยิงนี้ โดยคลาสที่อยู่ในส่วนนี้มีเพียงคลาสเดียวคือ GGShooting Class

7.3.1 GGShooting Class

เป็นคลาสที่ทำหน้าที่ในการกำหนดและสร้างระบบการทำงานของเกมประเภทยานยิงทั้งหมดซึ่งคลาสนี้ได้สืบทอดลงมาจาก GGOS Class และคลาสนี้จะมีการควบคุมให้การทำงานของเกมเป็นไปตามรูปแบบที่ผู้สร้างต้องการ โดยจะเป็นประเภทแนวยานยิงเท่านั้น การควบคุมหลักๆคือ การสร้างวัตถุต่างๆที่มีอยู่ในเกมทั้งหมด การแสดงผลวัตถุต่างๆ การควบคุมการกำเนิดและส่งเหตุการณ์ที่จะเกิดขึ้น การรับอินพุทจากผู้เล่น การ load ข้อมูลต่างๆที่จำเป็นต่อการทำงานของเกมขึ้นมาใช้งาน การกำหนดค่าต่างๆที่จำเป็น โดยในคลาส GGShooting นี้มีฟังก์ชันการทำงานที่สำคัญดังนี้

1. GGShooting(void *hInst,void *hWnd) เป็นฟังก์ชันการทำงานที่ใช้ในการสร้างระบบเกมประเภทยานยิงนี้
2. bool LoadShootingEnv(const char *filename) เป็นฟังก์ชันการทำงานที่ใช้ในการนำข้อมูลที่จำเป็นต่อการทำงานของระบบเกมประเภทยานยิงนี้ขึ้นมาจากแฟ้มข้อมูล
3. void InitialShootingGame() เป็นฟังก์ชันการทำงานที่ใช้ในการกำหนดและสร้างค่าเริ่มต้นต่างๆที่ใช้ในเกม
4. void UpdateInput() เป็นฟังก์ชันการทำงานที่ใช้ในการ update ข้อมูลจากอุปกรณ์รับข้อมูลภายนอก
5. void UpdatePlayer() เป็นฟังก์ชันการทำงานที่ใช้ในการ update ข้อมูลต่างๆที่เป็นของผู้เล่น
6. void UpdateMap() เป็นฟังก์ชันการทำงานที่ใช้ในการ update ข้อมูลและการแสดงผลที่อยู่บนฉาก
7. void UpdateObjects() เป็นฟังก์ชันการทำงานที่ใช้ในการ update ข้อมูลและการแสดงผลของวัตถุต่างๆ

บทที่ 8
คลาสต่าง ๆ ที่ใช้ในโปรแกรมสร้างและพัฒนาเกมสำเร็จรูป

8.1 GDKGame Class

เป็นคลาสที่ทำหน้าที่เป็นคลาสอ้างอิงหลักสำหรับโปรแกรมสร้างและพัฒนาเกมสำเร็จรูปในการเปิดหรือบันทึกแฟ้มข้อมูลเพื่อใช้กับตัวโปรแกรมสร้างและพัฒนาเกมสำเร็จรูปโดยจะเป็นคลาสที่ทำหน้าที่เก็บชื่อของเกมที่ทำการสร้างขึ้นรวมถึงค่าไดเรกทอรีที่กำหนดเป็นไดเรกทอรีรากของเกมนั้น

1. char* GetName(void) เป็นฟังก์ชันที่ทำหน้าที่ในการเรียกขอชื่อของเกมที่กำลังทำการสร้างในขณะนั้นจากตัวคลาส GDKGame
2. char* GetGameDirectory(void) เป็นฟังก์ชันที่ทำหน้าที่ในการเรียกขอค่าไดเรกทอรีรากที่ใช้เก็บข้อมูลในการสร้างเกมในขณะนั้นจากตัวคลาส GDKGame
3. void SetName(const char *NewName) เป็นฟังก์ชันที่ทำหน้าที่ในการเก็บค่าชื่อของเกมที่กำลังทำการสร้างในขณะนั้นเข้าสู่คลาส GDKGame
4. void SetGameDirectory(const char *NewGameDirectory) เป็นฟังก์ชันที่ทำหน้าที่ในการเก็บค่าไดเรกทอรีรากเข้าสู่คลาส GDKGame
5. LoadFromFile(AnsiString Filename) เป็นฟังก์ชันที่ทำเปิดแฟ้มข้อมูลเพื่อทำการโหลดข้อมูลเข้าสู่คลาส GDKGame
6. SaveToFile(AnsiString Filename) เป็นฟังก์ชันที่ทำการบันทึกข้อมูลที่อยู่ในคลาส GDKGame ลงสู่แฟ้มข้อมูล (นามสกุลของแฟ้มข้อมูลนี้ คือ *.gdk)

8.2 GGOSTool Class

เป็นคลาสที่สืบทอดมาจากคลาส GGOS ของส่วนรันไทม์ คลาสนี้มีหน้าที่ในการเก็บข้อมูลซึ่งถือเป็นข้อมูลหลักของตัวเกมในด้านของการทำงานที่เป็นระบบของเกม เช่น ค่าความกว้าง – ยาว ของสกรีนที่ต้องการสร้างขึ้นในขณะรันไทม์ หรือ ค่าของ SerialID ของวัตถุต่างๆที่มีในเกม

1. int GetSerialID(void) เป็นฟังก์ชันที่ทำหน้าที่เรียกดูค่าของ SerialID ที่จะทำการกำหนดให้กับวัตถุต่าง ๆ ในเกม
2. int GetScreenWidth(void) เป็นฟังก์ชันที่ทำหน้าที่เรียกดูค่าความกว้างของสกรีนที่ต้องการสร้างโดย DirectDraw ในขณะรันไทม์
3. int GetScreenHeight(void) เป็นฟังก์ชันที่ทำหน้าที่เรียกดูค่าความสูงของสกรีนที่ต้องการสร้างโดย DirectDraw ในขณะรันไทม์
4. int GetScreenBPP(void) เป็นฟังก์ชันที่ทำหน้าที่เรียกดูค่าความละเอียดของสกรีน หน่วยเป็นบิทต่อพิกเซลของสกรีนที่ต้องสร้างในขณะรันไทม์

5. char* GetPaletteFilename(void) เป็นฟังก์ชันที่ทำหน้าที่เรียกดูชื่อของพาเลทแฟ้มข้อมูลที่กำหนดเป็นพาเลทมาตรฐานของเกม
6. void SetSerialID(int NewSerialID) เป็นฟังก์ชันที่ทำหน้าที่กำหนดค่าของ SerialID ของวัตถุในเกมให้กับคลาส GGOSTool
7. void SetScreenWidth(int NewScreenWidth) เป็นฟังก์ชันที่ทำหน้าที่ในการกำหนดค่าความกว้างของสกรีนให้กับคลาส GGOSTool
8. void SetScreenHeight(int NewScreenHeight) เป็นฟังก์ชันที่ทำหน้าที่ในการกำหนดค่าความสูงของสกรีนให้กับคลาส GGOSTool
9. void SetScreenBPP(int NewScreenBPP) เป็นฟังก์ชันที่ทำหน้าที่ในการกำหนดค่าความละเอียดของสกรีนให้กับคลาส GGOSTool
10. void SetPaletteFilename(char *NewPaletteFilename) เป็นฟังก์ชันที่ทำหน้าที่ในการกำหนดชื่อแฟ้มข้อมูลที่เป็นตัวอย่างของแฟ้มข้อมูลพาเลทของเกมในคลาส GGOSTool
11. bool LoadFromFile(AnsiString Filename) เป็นฟังก์ชันที่ใช้ในการโหลดคลาส GGOSTool จากแฟ้มข้อมูล GameSystem.env
12. bool SaveToFile(AnsiString Filename) เป็นฟังก์ชันที่ทำการบันทึกข้อมูลในคลาส GGOSTool ลงแฟ้มข้อมูลGameSystem.env

8.3 GGShootingTool Class

เป็นคลาสที่ทำการสืบทอดมาจากคลาส GGShooting ของส่วนรันไทม์ มีหน้าที่ในการเก็บข้อมูลที่เป็นข้อมูลพื้นฐานของเกมแนวยานยิงโดยเฉพาะเพื่อประโยชน์ในด้านของความยืดหยุ่นต่อการขยายความสามารถของโปรแกรมสร้างและพัฒนาเกมสำเร็จรูปในการสร้างเกมประเภทอื่น

1. int GetMapScrollSpeed(void) เป็นฟังก์ชันที่ใช้ในการเรียกดูข้อมูลความเร็วในการเลื่อนของฉากแบบอัตโนมัติของเกมประเภทยานยิง
2. void SetMapScrollSpeed(int NewMapScrollSpeed) เป็นฟังก์ชันที่ทำหน้าที่ในการกำหนดค่าความเร็วในการเคลื่อนที่ของฉากแบบอัตโนมัติให้กับคลาส GGShootingTool
3. bool LoadFromFile(AnsiString Filename) เป็นฟังก์ชันที่ทำหน้าที่โหลดข้อมูลของคลาส GGShootingTool จากแฟ้มข้อมูล Shooting.env
4. bool SaveToFile(AnsiString Filename) เป็นฟังก์ชันที่ทำหน้าที่ในการบันทึกข้อมูลของคลาส GGShootingTool ลงสู่แฟ้มข้อมูล Shooting.env

8.4 GGMap Class

เป็นคลาสที่ทำหน้าที่ในการเก็บข้อมูลที่เป็นข้อมูลหลักของฉากต่าง ๆ ที่ใช้ในเกม เช่น จำนวนฉากทั้งหมดที่มีอยู่ในเกม ค่าไดเรกทอรีรากที่เก็บแฟ้มข้อมูลของฉากต่าง

1. char* GetMapDirectory(void) เป็นฟังก์ชันที่ทำหน้าที่ในการเรียกดูค่าของไดเรกทอรีรากที่ทำการเก็บแฟ้มข้อมูลของฉากต่าง ๆ ไว้
2. int GetMapNumber(void) เป็นฟังก์ชันที่ทำหน้าที่ในการเรียกดูค่าจำนวนของฉากทั้งหมดที่มีอยู่ในเกมที่ทำการสร้างขึ้นในขณะนั้น
3. AnsiString GetMapList(void) เป็นฟังก์ชันที่ทำหน้าที่ขอดูรายชื่อของฉากทั้งหมดที่ทำการสร้างขึ้นในเกมนั้น
4. void SetMapDirectory(const char *NewMapDirectory) เป็นฟังก์ชันที่ทำหน้าที่ในการกำหนดค่าไดเรกทอรีรากของแฟ้มข้อมูลฉากต่าง ๆ ในเกม
5. void SetMapNumber(int NewMapNumber) เป็นฟังก์ชันที่ทำหน้าที่ในการกำหนดจำนวนฉากทั้งหมดในเกมให้กับคลาส GGMap
6. SetMapList(AnsiString NewMapList) เป็นฟังก์ชันที่ทำหน้าที่ในการกำหนดรายชื่อของฉากทั้งหมดภายในเกมให้กับคลาส GGMap
7. bool LoadFromFile(AnsiString Filename) เป็นฟังก์ชันที่ทำหน้าที่ในการเก็บข้อมูลของคลาส GGMap ลงสู่แฟ้มข้อมูล Maps.gbl
8. bool SaveToFile(AnsiString Filename) เป็นฟังก์ชันที่ทำหน้าที่ในการโหลดข้อมูลจากแฟ้มข้อมูล Maps.gbl เข้าสู่คลาส GGMap

8.5 GGStaticObject Class

เป็นคลาสที่ทำการเก็บข้อมูลพื้นฐานของวัตถุแบบเคลื่อนที่ไม่ได้ทั้งหมดที่มีอยู่ภายในเกม เช่น ไดเรกทอรีรากที่เก็บแฟ้มข้อมูลของวัตถุที่เคลื่อนที่ไม่ได้ จำนวนวัตถุที่เคลื่อนที่ไม่ได้ เป็นต้น

1. int GetStaticObjectNumber(void) ฟังก์ชันที่ใช้ในการเรียกดูค่าจำนวนของวัตถุที่ไม่สามารถเคลื่อนที่ได้ทั้งหมดที่มีอยู่ภายในเกม
2. AnsiString GetStaticObjectList(void) ฟังก์ชันที่ทำหน้าที่ในการเรียกดูรายชื่อของแฟ้มข้อมูลทั้งหมดของวัตถุที่ไม่สามารถเคลื่อนที่ได้ภายในเกม
3. AnsiString GetStaticObjectDirectory(void) ฟังก์ชันที่ทำหน้าที่ในการเรียกดูไดเรกทอรีรากที่เก็บข้อมูลแฟ้มข้อมูลของวัตถุที่ไม่สามารถเคลื่อนที่ได้
4. void SetStaticObjectNumber(int NewStaticObjectNumber) ฟังก์ชันที่ทำหน้าที่ในการกำหนดจำนวนทั้งหมดของวัตถุที่ไม่สามารถเคลื่อนที่ได้ในเกมให้กับคลาส GGStaticObject
5. void SetStaticObjectList(AnsiString NewStaticObjectList) ฟังก์ชันที่ทำหน้าที่ในการกำหนดรายชื่อของแฟ้มข้อมูลทั้งหมดที่เป็นวัตถุที่ไม่สามารถเคลื่อนที่ได้ภายในเกมให้กับคลาส GGStaticObject
6. void SetStaticObjectDirectory(const char *NewStaticObjectDirectory) ฟังก์ชันที่ทำหน้าที่ในการกำหนดค่าไดเรกทอรีรากที่เก็บแฟ้มข้อมูลของวัตถุที่ไม่สามารถเคลื่อนที่ได้

7. bool LoadFromFile(AnsiString Filename) ฟังก์ชันที่ทำหน้าที่ในการโหลดแฟ้มข้อมูล Statics.gbl เข้าสู่คลาส GGStaticObject
8. bool SaveToFile(AnsiString Filename) ฟังก์ชันที่ทำหน้าที่ในการบันทึกค่าในคลาส GGStaticObject ลงสู่แฟ้มข้อมูล Static.gbl

8.6 GGDynamicObject Class

เป็นคลาสที่ทำการเก็บข้อมูลพื้นฐานของวัตถุแบบเคลื่อนที่ได้ทั้งหมดที่มีอยู่ภายในเกม เช่น ไดเรกทอรีรากที่เก็บแฟ้มข้อมูลของวัตถุที่เคลื่อนที่ได้ จำนวนวัตถุที่เคลื่อนที่ได้ทั้งหมดภายในเกม เป็นต้น

1. char* GetDynamicObjectDirectory(void) ฟังก์ชันที่ทำหน้าที่ในการเรียกดูไดเรกทอรีรากที่เก็บข้อมูลแฟ้มข้อมูลของวัตถุที่สามารถเคลื่อนที่ได้
2. int GetDynamicObjectNumber(void) ฟังก์ชันที่ใช้ในการเรียกดูค่าจำนวนของวัตถุที่สามารถเคลื่อนที่ได้ทั้งหมดที่มีอยู่ภายในเกม
3. AnsiString GetDynamicObjectList(void) ฟังก์ชันที่ทำหน้าที่ในการเรียกดูรายชื่อของแฟ้มข้อมูลทั้งหมดของวัตถุที่สามารถเคลื่อนที่ได้ภายในเกม
4. void SetDynamicObjectDirectory(const char *NewDynamicObjectDirectory) ฟังก์ชันที่ทำหน้าที่ในการกำหนดจำนวนทั้งหมดของวัตถุที่สามารถเคลื่อนที่ได้ในเกมให้กับคลาส GGStaticObject
5. void SetDynamicObjectNumber(int NewDynamicObjectNumber) เป็นฟังก์ชันที่ใช้ในการกำหนดค่าไดเรกทอรีรากที่ใช้เก็บแฟ้มข้อมูลของวัตถุที่เคลื่อนที่ได้ทั้งหมดภายในเกม
6. void SetDynamicObjectList(AnsiString NewDynamicObjectList) เป็นฟังก์ชันที่ทำหน้าที่ในการกำหนดรายชื่อแฟ้มข้อมูลของวัตถุที่เคลื่อนที่ได้ทั้งหมดภายในเกมให้กับคลาส GGDynamicObject
7. bool LoadFromFile(AnsiString Filename) เป็นฟังก์ชันที่ทำหน้าที่ในการ โหลดข้อมูลจากแฟ้มข้อมูล Dynamics.gblเข้าสู่คลาส GGDynamicObject
8. bool SaveToFile(AnsiString Filename) เป็นฟังก์ชันที่ทำหน้าที่ในการบันทึกข้อมูลทั้งหมดในคลาส GGDynamicObject ลงสู่แฟ้มข้อมูล Dynamics.gbl

8.7 GDXMap Class

เป็นคลาสที่ทำการสืบทอดมาจากคลาส GMap มีหน้าที่ในการเก็บข้อมูลของฉากที่มีอยู่ในเกม โดยคลาสนี้จะเป็นส่วนที่ทำหน้าที่อยู่ในโปรแกรมสร้างและพัฒนาเกมสำเร็จรูปโดยจะมีการเก็บค่ารายละเอียดต่าง ๆ ที่ไม่มีในคลาส GMap แต่มีความจำเป็นที่จะต้องใช้ในตัวโปรแกรมสร้างและพัฒนาเกมสำเร็จรูป ส่วนตัวคลาส GMap จะถูกใช้อยู่ในตัวโปรแกรมรันไทม์

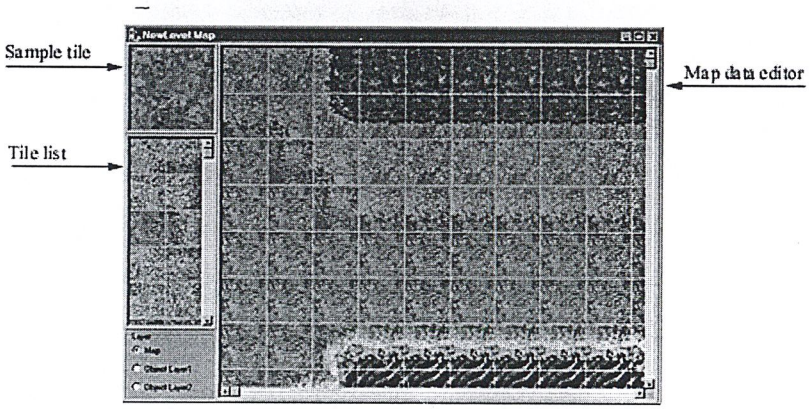
1. void Create(int MapWidth,int MapHeight,int Screen_TW,int Screen_TH) ฟังก์ชันทำหน้าที่ในการกำหนดค่าพื้นฐานให้กับฉากต่าง ๆ ในขั้นตอนการตั้งค่าพื้นฐานให้กับฉาก

2. bool LoadTiles()ฟังก์ชันทำหน้าที่ในการโหลดไทล์ที่ใช้เป็นไทล์มาตรฐานของฉากให้กับ GDXMap เพื่อนำไปทำการสร้างฉาก
3. void SetWidth(int NewWidth) ฟังก์ชันทำการกำหนดค่าความกว้างของฉาก
4. void SetHeight(int NewHeight) ฟังก์ชันทำการกำหนดค่าความสูงของฉาก
5. void SetMapPixelWidth(int NewMapPixelWidth) ฟังก์ชันทำการกำหนดค่าความกว้างของฉากในหน่วยเป็นพิกเซล
6. void SetMapPixelHeight(int NewMapPixelHeight) ฟังก์ชันทำการกำหนดค่าความสูงของฉากในหน่วยเป็นพิกเซล
7. void SetTileFilename(char *Filename) ฟังก์ชันในการกำหนดชื่อแฟ้มข้อมูลไทล์ให้กับคลาส GDXMap
8. void SetTileNum(int NewTileNum) ฟังก์ชันทำการกำหนดค่าจำนวนไทล์ของไทล์ข้อมูลที่ใช้ในฉาก
9. void SetTilePixelWidth(int NewTilePixelWidth) ฟังก์ชันทำการกำหนดค่าความกว้างของไทล์หนึ่งไทล์ในหน่วยเป็นพิกเซล
10. void SetTilePixelHeight(int NewTilePixelHeight) ฟังก์ชันทำการกำหนดค่าความสูงของไทล์หนึ่งไทล์ในหน่วยเป็นพิกเซล
11. Void SetScreen_TW(int screen_tw) ฟังก์ชันทำการกำหนดค่าความกว้างของสกรีนในหน่วยเป็นไทล์ –
12. void SetScreen_TH(int screen_th) ฟังก์ชันทำการกำหนดค่าความสูงของสกรีนในหน่วยเป็นไทล์
13. void SetScreen_W(int screen_w) ฟังก์ชันทำการกำหนดค่าความกว้างของไทล์ในหน่วยเป็นพิกเซล
14. void SetScreen_H(int screen_h) ฟังก์ชันทำการกำหนดค่าความสูงของไทล์ในหน่วยเป็นพิกเซล
15. void SetScreenTileSize(int screen_tw,int screen_th) ฟังก์ชันทำการกำหนดขนาดของไทล์กว้าง – ยาวในหน่วยเป็นไทล์
16. void SetScreenSize(int screen_w,int screen_h) ฟังก์ชันทำการกำหนดขนาดของไทล์ กว้าง – ยาว ในหน่วยเป็นพิกเซล
17. void SetSpeed(int NewSpeed) ฟังก์ชันทำการกำหนดค่าความเร็วในการเคลื่อนที่ของฉากแบบอัตโนมัติ
18. void SetDirection(int NewDirection) ฟังก์ชันในการกำหนดค่าทิศทางการเคลื่อนที่อัตโนมัติของฉาก
19. void SetStaticList(AnsiString NewStaticList) ฟังก์ชันในการกำหนดชื่อแฟ้มข้อมูลทั้งหมดของวัตถุชนิดไม่สามารถเคลื่อนที่ได้ภายในฉาก ๆ นั้น

20. void SetDynamicList(AnsiString NewDynamicList) ฟังก์ชันทำการกำหนดรายชื่อแฟ้มข้อมูลทั้งหมดของวัตถุชนิดที่เคลื่อนที่ได้ภายในฉากนั้นๆ
21. AnsiString GetStaticList(void) ฟังก์ชันทำการเรียกชื่อแฟ้มข้อมูลทั้งหมดของวัตถุที่ไม่สามารถเคลื่อนที่ได้มาทำการตรวจสอบ
22. AnsiString GetDynamicList(void) ฟังก์ชันทำการเรียกชื่อแฟ้มข้อมูลทั้งหมดของวัตถุที่สามารถเคลื่อนที่ได้มาทำการตรจสอบ
23. void PutAllMap(int TileID) ฟังก์ชันในการเก็บข้อมูลทุก ๆ ไทล์ในฉากให้เป็นค่าไทล์ที่ต้องการ
24. bool LoadFromFile(AnsiString Filename) ฟังก์ชันที่ทำหน้าที่ในการโหลดข้อมูลจากฉากทั้งหมดเข้าสู่ตัวคลาส GDXMap
25. bool SaveToFile(AnsiString Filename) เป็นฟังก์ชันที่ทำหน้าที่ในการ เก็บข้อมูลของ GDXMap เข้าสู่แฟ้มข้อมูลที่ต้องการ (นามสกุลเป็น *.Map)

8.8 TMapEditForm Class

คลาสนี้ทำหน้าที่ในการสร้างฟอร์มที่ใช้ในการสร้างฉาก (ส่วนของโปรแกรมสร้างฉาก) และยังทำหน้าที่ในการเชื่อมตัวโปรแกรมสร้างส่วนต่าง ๆ ในโปรแกรมสร้างและพัฒนาเกมสำเร็จรูปอีกด้วยโดยอาศัยการกำหนดเหตุการณ์ต่างบนตัว MapGrid ของฟอร์มนี้



รูปที่8.1 แสดงฟอร์มที่ได้จากคลาส TmapEditForm

1. void AddStaticObject(int X,int Y,int Z,int StaticObjectID) เป็นฟังก์ชันที่ทำหน้าที่ในการเพิ่มวัตถุประเภทไม่สามารถเคลื่อนที่ได้เข้าไปในลิสต์ของวัตถุประเภทเคลื่อนที่ไม่ได้ของฉากนั้น (Static มีโครงสร้างข้อมูลแบบ STATICLIST)

2. void RemoveStaticObject(int StaticObjectID) เป็นฟังก์ชันที่ทำหน้าที่ในการลบวัตถุประเภทเคลื่อนที่ไม่ได้ออกจากลิสต์ของวัตถุประเภทเคลื่อนที่ไม่ได้ของฉากนั้น (ตรงข้ามกับฟังก์ชัน AddStaticObject)
3. void AddDynamicObject(int X,int Y,int Z,int DynamicObjectID) เป็นฟังก์ชันที่ทำหน้าที่ในการเพิ่มวัตถุประเภทเคลื่อนที่ได้เข้าไปในลิสต์ของวัตถุประเภทเคลื่อนที่ได้ของฉากนั้น (Dynamic มีโครงสร้างข้อมูลแบบ DYNAMICLIST)
4. void RemoveDynamicObject(int DynamicObjectID) เป็นฟังก์ชันที่ทำหน้าที่ในการลบวัตถุประเภทเคลื่อนที่ได้ออกจากลิสต์ของวัตถุประเภทเคลื่อนที่ได้ของฉากนั้น (ตรงข้ามกับฟังก์ชัน AddDynamicObject)
5. AnsiString SearchStaticDraw(int Pos_X,int Pos_Y,int Pos_Z) เป็นฟังก์ชันที่ทำหน้าที่ในการค้นหาวัตถุประเภทไม่สามารถเคลื่อนที่ได้ในฉากนั้นที่ตำแหน่ง Pos_X , Pos_Y , Pos_Z ที่ต้องการแล้วทำการคืนค่ากลับมาเป็นชุดอักขระที่บอกค่า ID ของวัตถุทั้งหมดแบบเคลื่อนที่ไม่ได้ ณ ตำแหน่งนั้น
6. AnsiString SearchDynamicDraw(int Pos_X,int Pos_Y,int Pos_Z) เป็นฟังก์ชันที่ทำหน้าที่ในการค้นหาวัตถุประเภทเคลื่อนที่ได้ในฉากนั้นที่ตำแหน่ง Pos_X , Pos_Y , Pos_Z ที่ต้องการแล้วทำการคืนค่ากลับมาเป็นชุดอักขระที่บอกค่า ID ของวัตถุทั้งหมดแบบเคลื่อนที่ได้ ณ ตำแหน่งนั้น
7. bool SearchStatic(int ID) เป็นฟังก์ชันที่ทำหน้าที่ในการหาตำแหน่งแอดเดรสที่เก็บข้อมูลของวัตถุที่ไม่สามารถเคลื่อนที่ได้ตาม ID ของวัตถุที่กำหนดให้
8. bool SearchDynamic(int ID) เป็นฟังก์ชันที่ทำหน้าที่ในการหาตำแหน่งแอดเดรสที่เก็บข้อมูลของวัตถุที่สามารถเคลื่อนที่ได้ตาม ID ของวัตถุที่กำหนดให้
9. void DrawStatic(GStaticObject *StaticObject) เป็นฟังก์ชันที่ทำหน้าที่ในการวาดวัตถุที่ไม่สามารถเคลื่อนที่ได้ลงบน MapGrid
10. void DrawDynamic(GDynamicObject *DynamicObject) เป็นฟังก์ชันที่ทำหน้าที่ในการวาดวัตถุที่สามารถเคลื่อนที่ได้ลงบน MapGrid
11. void UpdateStatic(AnsiString Result) เป็นฟังก์ชันที่ทำหน้าที่ในการปรับปรุงข้อมูลและทำการวาดวัตถุที่ไม่สามารถเคลื่อนที่ได้ทุกตัวบน MapGrid
12. void UpdateDynamic(AnsiString Result) เป็นฟังก์ชันที่ทำหน้าที่ในการปรับปรุงข้อมูลและทำการวาดวัตถุที่สามารถเคลื่อนที่ได้ทุกตัวบน MapGrid
13. int GetStaticCount()เป็นฟังก์ชันที่ทำหน้าที่ในการเรียกดูค่าจำนวนของวัตถุที่ไม่สามารถเคลื่อนที่ได้ที่มีในฉากนั้น
14. int GetDynamicCount()เป็นฟังก์ชันที่ทำหน้าที่ในการเรียกดูค่าจำนวนของวัตถุที่สามารถเคลื่อนที่ได้ที่มีในฉากนั้น

15. AnsiString SaveAllStaticObject(AnsiString StaticObjectList,int StaticObjectCount) เป็นฟังก์ชันที่ทำการบันทึกข้อมูลของวัตถุที่ไม่สามารถเคลื่อนที่ได้ทุก ๆ ตัวที่ทำการสร้างบนฉากนั้นลงในแฟ้มข้อมูล
16. AnsiString SaveAllDynamicObject(AnsiString DynamicObjectList,int DynamicObjectCount) เป็นฟังก์ชันที่ทำการบันทึกข้อมูลของวัตถุที่สามารถเคลื่อนที่ได้ทุก ๆ ตัวที่ทำการสร้างบนฉากนั้นลงในแฟ้มข้อมูล
17. void LoadAllStaticObject(AnsiString StaticObjectList) เป็นฟังก์ชันที่ทำการโหลดข้อมูลของวัตถุที่ไม่สามารถเคลื่อนที่ได้จากลิสต์ของแฟ้มข้อมูลที่กำหนดให้ แล้วสร้างวัตถุที่ไม่สามารถเคลื่อนที่ได้ลงบนฉากแผนที่นั้น
18. void LoadAllDynamicObject(AnsiString DynamicObjectList) เป็นฟังก์ชันที่ทำการโหลดข้อมูลของวัตถุที่สามารถเคลื่อนที่ได้จากลิสต์ของแฟ้มข้อมูลที่กำหนดให้ แล้วสร้างวัตถุที่สามารถเคลื่อนที่ได้ลงบนฉากแผนที่นั้น

8.9 GStaticObject Class

เป็นคลาสที่ตัวโปรแกรมสร้างและพัฒนาเกมสำเร็จรูปทำการใช้ร่วมกับส่วนของโปรแกรมรันไทม์โดยใช้ในการเก็บและอ้างอิงข้อมูลของวัตถุที่ไม่สามารถเคลื่อนที่ได้ แต่ในส่วนของโปรแกรมสร้างและพัฒนาเกมสำเร็จรูปนั้นจะทำการใช้งานคลาสนี้ผ่านทางโครงสร้างแบบลิงค์ลิสต์ STATICLIST โดยจะให้คลาส GStaticObject เป็นคลาสข้อมูลในโครงสร้าง STATICLIST

8.10 GDynamicObject Class

เป็นคลาสที่ตัวโปรแกรมสร้างและพัฒนาเกมสำเร็จรูปทำการใช้ร่วมกับส่วนของโปรแกรมรันไทม์โดยใช้ในการเก็บและอ้างอิงข้อมูลของวัตถุที่สามารถเคลื่อนที่ได้ แต่ในส่วนของโปรแกรมสร้างและพัฒนาเกมสำเร็จรูปนั้นจะทำการใช้งานคลาสนี้ผ่านทางโครงสร้างแบบลิงค์ลิสต์ DYNAMICLIST โดยจะให้คลาส GDynamicObject เป็นคลาสข้อมูลในโครงสร้าง DYNAMICLIST

8.11 TileList Class

เป็นคลาสที่ทำหน้าที่ในการเก็บข้อมูลของชุดของไทล์ต่าง ๆ ตามที่ตัวโปรแกรมสร้างและพัฒนาเกมสำเร็จรูปต้องการ

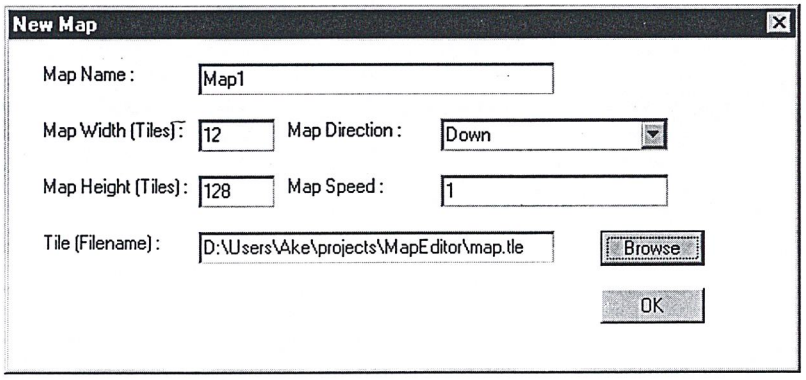
1. AnsiString GetType(void) เป็นฟังก์ชันที่ใช้ในการขอดูค่าที่บอกประเภทของไทล์ลิสต์ว่าเป็นประเภทชุดของวัตถุที่ไม่สามารถเคลื่อนที่ได้ หรือ วัตถุที่สามารถเคลื่อนที่ได้ (แต่เราไม่สามารถทำการกำหนดค่าใหม่ให้กับไทล์ลิสต์ได้จะกำหนดได้เฉพาะตอนสร้างไทล์ลิสต์ขึ้นมาเท่านั้น)

2. int GetTileNumber(void) เป็นฟังก์ชันที่ใช้ในการขอดูค่าที่บอกจำนวนของไทล์ลิสต์ว่าประกอบด้วยลิสต์ของไทล์มีจำนวนไทล์ทั้งหมดในลิสต์เป็นเท่าใด (แต่เราไม่สามารถทำการกำหนดค่าใหม่ให้กับไทล์ลิสต์ได้จะกำหนดได้เฉพาะตอนสร้างไทล์ลิสต์ขึ้นมาเท่านั้น)
3. bool TileList::SaveToFile(void) เป็นฟังก์ชันที่ใช้สำหรับทำการเก็บข้อมูลของไทล์ลิสต์เข้าสู่แฟ้มข้อมูลStatic.lst ถ้าเป็นไทล์ข้อมูลที่เป็นประเภทวัตถุที่ไม่สามารถเคลื่อนที่ได้ และจะบันทึกข้อมูลสู่แฟ้มข้อมูลDynamic.lst ถ้าเป็นไทล์ข้อมูลของวัตถุที่สามารถเคลื่อนที่ได้
4. bool TileList::LoadFromFile(void) เป็นฟังก์ชันที่ทำหน้าที่ในการโหลดแฟ้มข้อมูลเข้าสู่ไทล์ลิสต์

นอกจากคลาสหลัก ๆ ทั้งหมดที่ได้กล่าวมาแล้วนั้นตัวโปรแกรมสร้างและพัฒนาเกมสำเร็จรูปก็ยังประกอบด้วยคลาสอื่น ๆ ที่ใช้ในการสร้างฟอร์มของส่วนย่อยอื่น ๆ ที่ใช้ในการทำงาน ซึ่งได้แก่

8.12 TForm2 Class

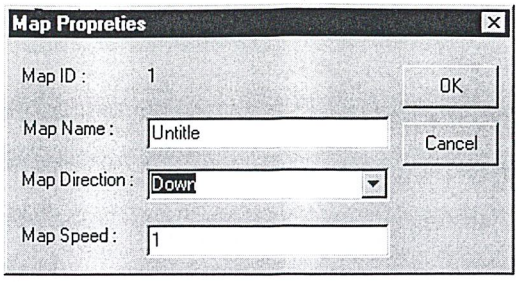
เป็นคลาสที่ทำหน้าที่ในการสร้างฟอร์มสำหรับรับข้อมูลเพื่อทำการสร้างฉากใหม่ในเกมซึ่งจะมีลักษณะรูปร่างของฟอร์มดังภาพ



รูปที่8.2 แสดงลักษณะของฟอร์มสร้างฉากใหม่ซึ่งเป็นของ TForm2 Class

8.13 TForm3 Class

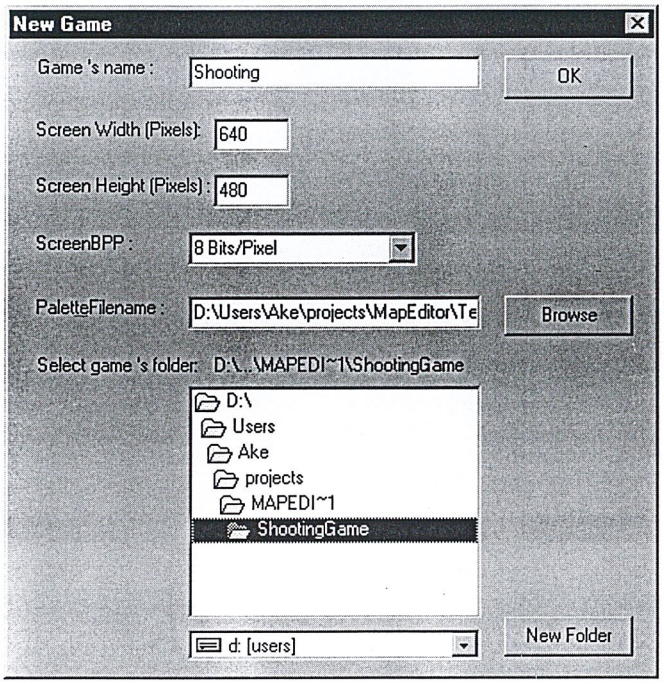
เป็นคลาสที่ใช้ในการสร้างฟอร์มสำหรับรับข้อมูลเพื่อทำการแก้ไขข้อมูลหลักของฉากต่าง ๆ ซึ่งจะมีลักษณะของฟอร์มดังภาพ



รูปที่8.3 แสดงลักษณะของฟอร์มที่ใช้รับข้อมูลคุณสมบัติของฉาก เพื่อทำการแก้ไขซึ่งเป็นของ TForm3 Class

8.14 TForm5 Class

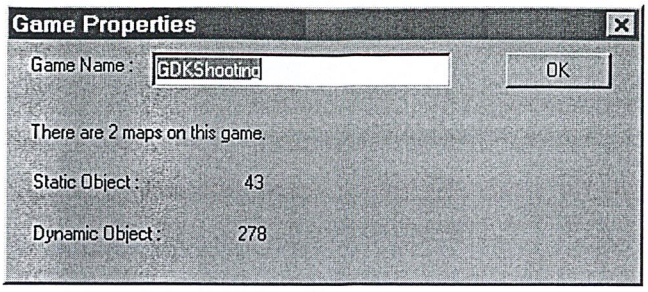
เป็นคลาสที่ใช้ในการสร้างฟอร์มเพื่อรับข้อมูลที่จะใช้ในการสร้างเกมใหม่โดยการใช้โปรแกรมสร้างและพัฒนาเกมสำเร็จรูป โดยฟอร์มที่ได้จะมีลักษณะดังภาพ



รูปที่ 8.4 แสดงฟอร์มที่ใช้ในการรับข้อมูลสำหรับทำการสร้างเกมใหม่ใน โปรแกรมสร้างและพัฒนาเกมสำเร็จรูปซึ่งได้จากคลาส TForm5

8.15 TForm8 Class

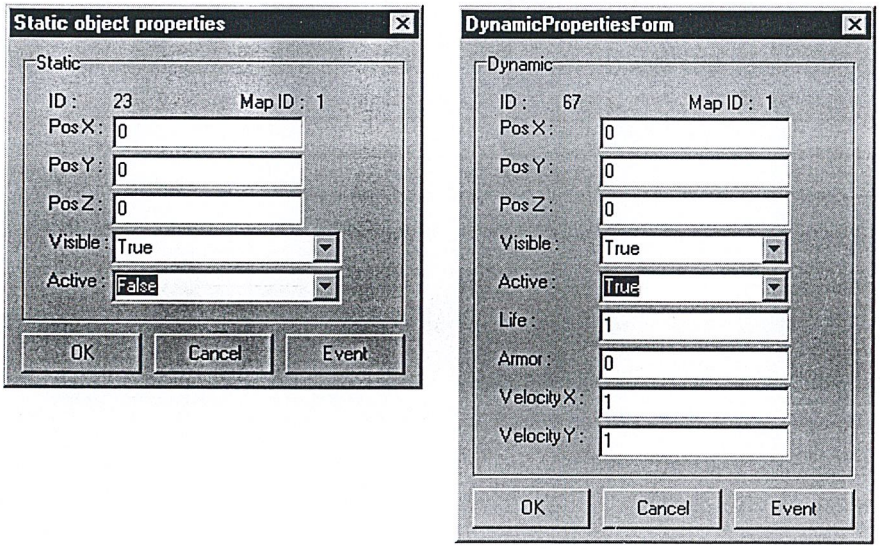
เป็นคลาสที่ทำหน้าที่ในการสร้างฟอร์มเพื่อทำการรับข้อมูลสำหรับการแก้ไขคุณสมบัติพื้นฐานของเกมที่ทำการสร้างในโปรแกรมสร้างและพัฒนาเกมสำเร็จรูปซึ่งมีลักษณะของฟอร์มดังภาพ



รูปที่8.5 แสดงลักษณะของฟอร์มที่ใช้กำหนดคุณสมบัติของเกมที่ได้รับจากคลาส TForm8

8.16 TStaticPropertiesForm และ TDynamicPropertiesForm

เป็นคลาสที่ทำหน้าที่ในการสร้างฟอร์มสำหรับรับข้อมูลเพื่อทำการแก้ไขข้อมูลของตัววัตถุแบบเคลื่อนที่ได้ และ วัตถุแบบเคลื่อนที่ไม่ได้ ซึ่งจะมีลักษณะของฟอร์มดังภาพ



รูปที่8.6 แสดงฟอร์มสำหรับรับข้อมูลเพื่อแก้ไขข้อมูลในตัววัตถุแบบเคลื่อนที่ไม่ได้ (ซ้าย) และ แก้ไขข้อมูลในวัตถุแบบเคลื่อนที่ได้ (ขวา) ซึ่งเกิดจาก TStaticPropertiesForm และ TDynamicObjectForm

บทที่ 9
โครงสร้างของแฟ้มข้อมูลที่ใช้ในการทำงานของเกม

แฟ้มข้อมูลต่างๆที่จำเป็นต้องมีการสร้างขึ้นมาจากโปรแกรมสร้างและพัฒนาเกมสำเร็จรูปนั้นจะต้องมีโครงสร้างของข้อมูลเป็นไปตามที่ได้กำหนดไว้เท่านั้น โดยโครงสร้างข้อมูลในแฟ้มข้อมูลแต่ละแฟ้มนั้นมีดังนี้

9.1 แฟ้มข้อมูลที่เก็บรายละเอียดหลักของเกม เช่น ชื่อแฟ้มข้อมูลที่จำเป็นต้องใช้ ค่าตัวแปลต่างๆที่ใช้ในการทำงานของเกม และข้อมูลที่มีความจำเป็นในการเริ่มการทำงานของเกม (Game System Environment File) ซึ่งจะเป็นข้อมูลที่ไม่ได้เฉพาะเจาะจงลงไปว่าจะถูกใช้ในเกมประเภทใด จึงสามารถนำไปใช้ได้ในทุกๆรูปแบบของเกม โดยมีโครงสร้างดังนี้

```
typedef struct tagGameSystemEnv
{
  char        gseSignature[4];
  int         gseVersion;
  int         gseID;
  char        gseName[20];
  int         gseScreenWidth;
  int         gseScreenHeight;
  int         gseScreenBPP;
  char        gsePaletteFilename[255];
  char        gseGlobalMapFilename[255];
  char        gseGlobalDynaObjFilename[255];
  char        gsePlayerFilename[255];
  char        gseGlobalStaticObjFilename[255];
} GAMESYSTEMENV;
```

โดยมีรายละเอียดของข้อมูลที่เก็บในตัวแปลเหล่านี้ดังนี้

1. gseSignature เก็บอักขระ 4 ตัวที่เป็นสัญลักษณ์ของแฟ้มข้อมูลนี้
2. gseVersion เก็บเลขจำนวนเต็มที่เป็นหมายเลขรุ่นของแฟ้มข้อมูลนี้
3. gseID เก็บเลขจำนวนเต็มที่เป็นหมายเลขลำดับของแฟ้มข้อมูลนี้(ในกรณีมีหลายแฟ้ม)

4. gseName เก็บอักขระ 20 ตัวที่เป็นชื่อของไฟล์นี้
5. gseScreenWidth เก็บเลขจำนวนเต็มที่เป็นขนาดความกว้างของหน้าจอในการแสดงผลของเกม
6. gseScreenHeight เก็บเลขจำนวนเต็มที่เป็นขนาดความสูงของหน้าจอในการแสดงผลของเกม
7. gseScreenBPP เก็บเลขจำนวนเต็มที่เป็นจำนวนบิตต่อจุด(Bit Per Pixel) ซึ่งหมายถึงจำนวนสีที่แสดงได้ในเกมเช่นถ้าเป็น 8 ก็หมายความว่าเกมสามารถแสดงได้ 2^8 หรือ 256 สี
8. gsePaletteFilename เก็บอักขระ 255 ตัวที่เป็นชื่อแฟ้มข้อมูลของตารางสีที่ใช้ในการแสดงผล (Palette)
9. gseGlobalMapFilename เก็บอักขระ 255 ตัวที่เป็นชื่อแฟ้มข้อมูลของแฟ้มข้อมูลที่เก็บชื่อของแฟ้มข้อมูลที่เก็บข้อมูลที่ใช้ในการกำหนดฉากและแผนที่ทั้งหมดในเกม (Map File)
10. gseGlobalDynObjFilename เก็บอักขระ 255 ตัวที่เป็นชื่อแฟ้มข้อมูลของแฟ้มข้อมูลที่เก็บชื่อของแฟ้มข้อมูลที่เก็บข้อมูลที่ใช้ในการกำหนดคุณสมบัติ และค่าต่างๆ สำหรับวัตถุที่สามารถเคลื่อนที่ได้ (Dynamic Object)
11. gsePlayerFilename เก็บอักขระ 255 ตัวที่เป็นชื่อแฟ้มข้อมูลของของแฟ้มข้อมูลที่เก็บข้อมูลที่ใช้ในการกำหนดคุณสมบัติและค่าต่างๆสำหรับผู้เล่นที่อยู่บนเกม (Player Object)
12. gseGlobalStaticObjFilename เก็บอักขระ 255 ตัวที่เป็นชื่อแฟ้มข้อมูลของแฟ้มข้อมูลที่เก็บชื่อของแฟ้มข้อมูลที่เก็บข้อมูลที่ใช้ในการกำหนดคุณสมบัติ และค่าต่างๆ สำหรับวัตถุที่ไม่สามารถเคลื่อนที่ได้ (Static Object)

จากโครงสร้างของข้อมูลที่เก็บมาทั้งหมดนี้จะใช้ในการเริ่มต้นการทำงานของเกม โดยจะถูกเก็บไว้ในแฟ้มข้อมูลที่ชื่อ “gamesystem.env”

9.2 แฟ้มข้อมูลที่เก็บรายละเอียดที่จำเป็นต้องใช้ในการทำงานของเกมประเภทยานยิง(Shooting Game Environment File) โดยในแฟ้มข้อมูลนี้จำเก็บข้อมูลที่มีลักษณะเฉพาะเจาะจงลงไป โดยจะใช้ได้เฉพาะในเกมประเภทยานยิงเท่านั้น โดยมีโครงสร้างดังนี้

```
typedef struct tagShootingGameEnv
{
  char          sgeSignature[4];
  int           sgeVersion;
  int           sgeID;
```

```
    char        sgeName[20];
    int         sgeMapScrollSpeed;
    int         sgePlayerMissileTick;
    char        sgePlayerMissileFile[255];
} SHOOTINGGAMEENV;
```

โดยมีรายละเอียดของข้อมูลที่เก็บในตัวแปลเหล่านี้ดังนี้

1. sgeSignature เก็บอักขระ 4 ตัวที่เป็นสัญลักษณ์ของแฟ้มข้อมูลนี้
2. sgeVersion เก็บเลขจำนวนเต็มที่เป็นหมายเลขรุ่นของแฟ้มข้อมูลนี้
3. sgeID เก็บเลขจำนวนเต็มที่เป็นหมายเลขลำดับของแฟ้มข้อมูลนี้(ในกรณีมีหลายแฟ้ม)
4. sgeName เก็บอักขระ 20 ตัวที่เป็นชื่อของไฟล์นี้
5. sgeMapScrollSpeed เก็บเลขจำนวนเต็มที่ใช้ในการกำหนดความเร็วในการเคลื่อนที่ของฉาก ซึ่งฉากนั้นจะเคลื่อนที่ไปเรื่อยๆตามความเร็วที่กำหนดไว้
6. sgePlayerMissileTick เก็บเลขจำนวนเต็มที่ใช้ในการกำหนดช่วงของระยะเวลาในการยิงกระสุนของผู้เล่นในแต่ละนัด
7. sgePlayerMissileFile เก็บอักขระ 255 ตัวที่เป็นชื่อแฟ้มข้อมูลของแฟ้มข้อมูลที่เก็บข้อมูลที่ใช้ในการกำหนดคุณสมบัติและค่าต่างๆสำหรับกระสุนของผู้เล่น (Missile File)

จากโครงสร้างของข้อมูลที่เก็บมาทั้งหมดนี้จะถูกใช้ในตอนเริ่มต้นการทำงานของเกมโดยจะถูกเก็บไว้ในแฟ้มข้อมูลที่ชื่อ “shooting.env”

9.3 แฟ้มข้อมูลที่เก็บข้อมูลซึ่งเป็นชื่อแฟ้มข้อมูลของฉาก(Global Map) และวัตถุที่สามารถเคลื่อนที่ได้ (Global Dynamic Object) วัตถุที่ไม่สามารถเคลื่อนที่ได้(Global Static Object) และผู้เล่น(Player Object) ที่ถูกใช้ทั้งหมดในเกม(Global Header) โดยมีโครงสร้างดังนี้

```
typedef struct tagGlobalObjectHeader
{
    char        goSignature[4];
    int         goVersion;
    int         goID;
    char        goName[20];
    char        goDirectory[255];
    int         goNumbers;
```

} GLOBALOBJECTHEADER

,GLOBALMAPHEADER,GLOBALPLAYERHEADER

โดยมีรายละเอียดของข้อมูลที่เก็บในตัวแปลเหล่านี้ดังนี้

1. goSignature เก็บอักขระ 4 ตัวที่เป็นสัญลักษณ์ของแฟ้มข้อมูลนี้
2. goVersion เก็บเลขจำนวนเต็มที่เป็นหมายเลขรุ่นของแฟ้มข้อมูลนี้
3. goID เก็บเลขจำนวนเต็มที่เป็นหมายเลขลำดับของแฟ้มข้อมูลนี้(ในกรณีมีหลายแฟ้ม)
4. goName เก็บอักขระ 20 ตัวที่เป็นชื่อของไฟล์นี้
5. goDirectory เก็บอักขระ 255 ตัวที่เป็นชื่อไดเร็กตอรี่ที่เก็บแฟ้มของฉากและแผนที่ วัตถุที่สามารถเคลื่อที่ได้ วัตถุที่ไม่สามารถเคลื่อนที่ได้ และผู้เล่น ที่ถูกใช้ในเกมไว้ทั้งหมด
6. goNumbers เก็บเลขจำนวนเต็มที่เป็นจำนวนของแฟ้มข้อมูลทั้งหมดที่ใช้ในเกม
7. และตามด้วยข้อมูลที่เป็นสตริงของชื่อแฟ้มข้อมูลแต่ละชื่อโดยจะมีอักขระ " " (ช่องว่าง) เป็นตัวแบ่งแยกระหว่างชื่อแต่ละชื่อ

จากโครงสร้างข้อมูลที่เก็บมาทั้งหมดนี้จะถูกใช้ในตอนเริ่มต้นในการทำงานของเกมโดยจะถูกเก็บไว้ในแฟ้มข้อมูลที่ชื่อ "map.gbl" สำหรับฉากและแผนที่ "dynaobj.gbl" สำหรับวัตถุที่สามารถเคลื่อนที่ได้ "staticobj.gbl" สำหรับวัตถุที่ไม่สามารถเคลื่อนที่ได้ และ "player.gbl" สำหรับผู้เล่น

9.4 แฟ้มข้อมูลที่เก็บข้อมูลสำหรับวัตถุต่างๆที่มีอยู่ในเกม ซึ่งจะเก็บในลักษณะของ 1 วัตถุต่อ 1 แฟ้มข้อมูลและจะมีส่วนหัวของแฟ้มข้อมูลต่างกันไปในกรณีของวัตถุที่มีลักษณะที่เพิ่มเติมจากวัตถุปรกติก็จะมีการเพิ่มส่วนหัวของแฟ้มข้อมูลเข้าไปอีกเช่น วัตถุที่มีการกำหนดเหตุการณ์ต่างๆ (Event) หรือวัตถุที่มีการกำหนดการทำงาน (Script) โดยขึ้นอยู่กับว่าเราจะเป็นผู้กำหนดคุณสมบัติการทำงานของวัตถุนั้นอย่างไร โดยโครงสร้างแฟ้มข้อมูลของวัตถุจะแบ่งเป็น 3 โครงสร้างดังนี้

9.4.1 ส่วนหัวของแฟ้มข้อมูล(Object Header) ซึ่งมีโครงสร้างดังนี้

```
typedef struct tagObjectHeader
{
  char        oSignature[4];
  int         oVersion;
  CLASSID     oClassID;
  int         oID;
  char        oName[20];
  int         oTotalEvent;
```

```
int         oTotalEventGenerator;
int         oParentID;
} OBJECTHEADER;
```

โดยมีรายละเอียดของข้อมูลที่เก็บในตัวแปลเหล่านี้ดังนี้

1. oSignature เก็บอักขระ 4 ตัวที่เป็นสัญลักษณ์ของแฟ้มข้อมูลนี้
2. oVersion เก็บเลขจำนวนเต็มที่เป็นหมายเลขรุ่นของแฟ้มข้อมูลนี้
3. oClassID เก็บค่าที่บอกถึงคลาสของวัตถุนี้
4. oID เก็บเลขจำนวนเต็มที่บอกถึงเลขประจำวัตถุนี้
5. oName เก็บอักขระจำนวน 20 ตัวที่เป็นชื่อของวัตถุนี้
6. oTotalEvent เก็บเลขจำนวนเต็มที่บอกถึงจำนวนของเหตุการณ์ที่จะเกิดขึ้นและมีผลต่อการทำงานของวัตถุนี้ ในกรณีที่วัตถุนี้เป็นวัตถุที่รับเหตุการณ์ที่เกิดขึ้นจากวัตถุอื่นหรือตัวมันเอง ซึ่งเกิดจากวัตถุประเภทที่เป็น ตัวกำเนิดเหตุการณ์(Event Generator)
7. oTotalEventGenerator เก็บเลขจำนวนเต็มที่บอกถึงจำนวนเหตุการณีที่วัตถุนี้จะกำเนิดให้กับวัตถุอื่นหรือตัวมันเองซึ่งเป็นวัตถุประเภทที่เป็น ตัวรับเหตุการณ์ (Remote Event Listener)
8. oParentID เก็บเลขจำนวนเต็มที่บ่งบอกถึงพ่อของวัตถุนี้ โดยถ้ามีค่าเป็น –1 แสงดว่าวัตถุนี้ไม่มีพ่อ

9.4.2 ส่วนหัวส่วนที่สองของแฟ้มข้อมูลที่ใช้เก็บข้อมูลต่างของวัตถุ (Object Data Header) ซึ่งมีโครงสร้างดังนี้

```
typedef struct tagObjectDataHeader
{
  int       odMapID;
  bool      odVisible;
  bool      odActive;
  int       odX;
  int       odY;
  int       odZ;
  int       odStartFrame;
  bool      odHasChild;
  bool      odPosAsParent;
```

```
    int         odRelPosX;
    int         odRelPosY;
    bool        odDieAsParent;
    bool        odDestroyOnDie;
    int         odChildTotalCount;
    bool        odCanRedup;
    bool        odCreateActive;
    bool        odCreateVisible;
} OBJECTDATAHEADER;
```

โดยมีรายละเอียดของข้อมูลที่เก็บในตัวแปลเหล่านี้ดังนี้

1. odMapID เก็บเลขจำนวนเต็มของฉากหรือแผนที่ที่วัตถุนี้อยู่ โดยถ้ามีค่าเป็น –1 แสดงว่าวัตถุนี้อยู่บนทุกๆฉาก
2. odVisible เก็บค่าตรรกะของการแสดงผลวัตถุ โดยถ้ามีค่าเป็นจริงแสดงว่าวัตถุนี้จะถูกแสดงให้เห็นและถ้าเป็นเท็จวัตถุนี้ก็จะไม่ถูกแสดงให้เห็น
3. odActive เก็บค่าตรรกะที่เป็นการบอกให้วัตถุนี้ทำงาน หรือไม่
4. odX เก็บเลขจำนวนเต็มที่เป็นตำแหน่งวัตถุในแนวแกน X บนฉากหรือแผนที่
5. odY เก็บเลขจำนวนเต็มที่เป็นตำแหน่งวัตถุในแนวแกน Y บนฉากหรือแผนที่
6. odZ เก็บเลขจำนวนเต็มที่เป็นตำแหน่งวัตถุในแนวแกน Z บนฉากหรือแผนที่ ซึ่งเป็นค่าที่แสดงถึงระดับชั้นของวัตถุเช่นถ้าวัตถุ ก. มีค่า odZ เป็น 0 และวัตถุ ข. มีค่า odZ เป็น 1 แสดงว่าวัตถุ ก. อยู่ต่ำกว่าวัตถุ ข.
7. odStartFrame เก็บเลขจำนวนเต็มที่เป็นเลขลำดับของรูป(Frame) ที่จะใช้ในการแสดงผลทันทีที่มองเห็น(Visible) หรือเริ่มทำงาน (Active) เช่นถ้าวัตถุมีภาพทั้งหมด 3 ภาพและต้องการให้แสดงภาพที่ 1 เมื่อวัตถุเริ่มการทำงานก็จะมีค่า odStartFrame เป็น 0 (ค่า odStartFrame จะเริ่มต้นจาก 0)
8. odHasChild เก็บค่าตรรกะที่บอกว่าวัตถุนี้มีลูกหรือไม่ และถ้ามีลูก(มีค่าเป็นจริง) ก็จะต้องมีการกำหนดข้อมูลต่างๆตั้งแต่ข้อ 9 นี้เป็นต้นไป
9. odPosAsParent เก็บค่าตรรกะที่บอกว่าเมื่อลูกของวัตถุนี้ได้ถูกสร้างขึ้นแล้วจะมีตำแหน่งเริ่มต้นเป็นที่ที่เดียวกับพ่อหรือไม่ ถ้ามีค่าเป็นจริงจะต้องกำหนดข้อมูลในข้อที่ 10 และ 11
10. odRelPosX เก็บเลขจำนวนเต็มที่เป็นตำแหน่งสัมพัทธ์กับตำแหน่งปัจจุบันของพ่อในแนวแกน X เช่นถ้ามีค่าเป็น 10 จะแสดงว่าลูกที่เกิดขึ้นนั้นจะมีค่าตำแหน่งในแนวแกน X เป็น ตำแน่งปัจจุบันของพ่อบวกกับอีก 10 หรือถ้าเป็น –10 ก็จะถูกนำไปลบแทน

11. odRelPosY เก็บเลขจำนวนเต็มที่เป็นตำแหน่งสัมพัทธ์กับตำแหน่งปัจจุบันของพ่อในแนวแกน Y เช่นถ้ามีค่าเป็น 10 จะแสดงว่าลูกที่เกิดขึ้นนั้นจะมีค่าตำแหน่งในแนวแกน Y เป็น ตำแน่งปัจจุบันของพ่อบวกกับอีก 10 หรือถ้าเป็น –10 ก็จะถูกนำไปลบแทน
12. odDieAsParent เก็บค่าตรรกะที่บอกว่าให้ลูกนั้นตายตามพ่อหรือไม่ทันทีที่พ่อตาย โดยถ้ามีค่าเป็นจริงจะต้องกำหนดข้อมูลในข้อ13
13. odDestryoOnDie เก็บค่าตรรกะที่บอกว่าเมื่อลูกตายแล้วให้ทำลายทิ้งหรือไม่ ถ้ามีค่าเป็นเท็จ ทันทีที่พ่อตายลูกก็จะแค่หยุดการทำงานทั้งหมดเท่านั้นเอง และถ้ามีค่าเป็นจริงจะต้องกำหนดข้อมูลในข้อ 14
14. odCanRedup เก็บค่าตรรกะที่บอกว่าลูกที่ตายไปแล้วนั้นสามารถถูกสร้างขึ้นมาใหม่ได้หรือไม่
15. odCreateActive เก็บค่าตรรกะที่บอกว่าลูกที่สร้างขึ้นมาแล้วนั้นจะทำงานทันทีหรือไม่
16. odCreativeVisible เก็บค่าตรรกะที่บอกว่าลูกที่สร้างขึ้นมาแล้วนั้นจะถูกแสดงให้เห็นทันทีหรือไม่

9.4.3 ส่วนหัวส่วนที่สามของแฟ้มข้อมูลที่ใช้เก็บข้อมูลของคลาส Tile ซึ่งเป็นคลาสที่เกี่ยวกับภาพและการแสดงผลของวัตถุ (Object Tile Data Header) ซึ่งมีโครงสร้างดังนี้

```
typedef struct tagObjectTileDataHeader
{
  int   otdBlockWidth;
  int   otdBlockHeight;
  int   otdBlockNumbers;
  int   otdColorKey;
  int   otdMaskSize;
  int   otdMaskWidth;
  int   otdMaskHeight;
  int   otdMaskBitUsed;
} OBJECTTILEDATAHEADER;
```

โดยมีรายละเอียดของข้อมูลที่เก็บในตัวแปลเหล่านี้ดังนี้

1. otdBlockWidth เก็บเลขจำนวนเต็มที่บอกถึงความกว้างของภาพที่ใช้ในวัตถุนี้ใน 1 Frame

2. otdBlockHeight เก็บเลขจำนวนเต็มที่บอกถึงความสูงของภาพที่ใช้ในวัตถุนี้ใน 1 Frame
3. otdBlockNumbers เก็บเลขจำนวนเต็มที่บอกถึงจำนวนของภาพย่อยที่ใช้สำหรับวัตถุนี้ซึ่งถูกแบ่งออกเป็น Frame
4. otdColorKey เก็บเลขจำนวนเต็มที่บอกถึงรหัสสีที่จะเป็นสีโปร่งใส (Transparent Color)
5. otdMaskSize เก็บเลขจำนวนเต็มที่บอกถึงขนาดของการ Mask ที่จะใช้ในการแบ่งภาพในแต่ละ Frame ของวัตถุนี้ออกเป็นส่วนย่อยๆ (Mask Bit)เพื่อใช้ในการตรวจสอบการทับกันของวัตถุ(Hit Detection) โดยถ้ามีค่าเป็น 0 แสดงว่าไม่มีการกำหนดให้ใช้การตรวจสอบแบบ Mask bit นี้
6. otdMaskWidth เก็บเลขจำนวนเต็มที่บอกถึงขนาดความกว่างของ Mask bit
7. otdMaskHeight เก็บเลขจำนวนเต็มที่บอกถึงขนาดความสูงของ Mask bit
8. otdMaskBitUsed เก็บเลขจำนวนเต็มที่บอกถึงจำนวนบิตที่ใช้ในการตรวจสอบการทับกันของวัตถุแบบ Mask bit

9.4.4 ส่วนหัวส่วนที่สี่ของแฟ้มข้อมูลที่ใช้เก็บข้อมูลของวัตถุที่สามารถเคลื่อนที่ได้เท่านั้น สำหรับวัตถุที่ไม่สามารถเคลื่อนที่ได้จะไม่มีส่วนนี้ ซึ่งมีโครงสร้างดังนี้

```
typedef struct tagDynamicObjectData
{
  DYNAMICOBJTYPE  dodType;
  int             dodLife;
  int             dodVelX;
  int             dodVelY;
} DYNAMICOBJECTDATA;
```

โดยมีรายละเอียดของข้อมูลที่เก็บในตัวแปลเหล่านี้ดังนี้

1. dodType เก็บค่าที่บอกถึงรูปแบบของวัตถุที่สามารถเคลื่อนที่ได้นี้ เช่น DO_ENEMY คือศัตรู DO_NEUTRAL คือวัตถุประกอบฉาก และ DO_PLAYER คือผู้เล่น
2. dodLife เก็บเลขจำนวนเต็มที่บอกถึงพลังชีวิตของวัตถุนี้
3. dodVelX เก็บเลขจำนวนเต็มที่บอกถึงความเร็วในแนวแกน X ของวัตถุนี้
4. dodVelY เก็บเลขจำนวนเต็มที่บอกถึงความเร็วในแนวแกน Y ของวัตถุนี้

9.4.5 ส่วนหัวส่วนที่ห้าของแฟ้มข้อมูลที่เก็บข้อมูลที่ใช้ในกำเนิดเหตุการณ์ต่างๆสำหรับวัตถุนี้ (Event Generator) ซึ่งมีโครงสร้างดังนี้

```
typedef struct tagEventGenerator
{
  int                egID;
  bool               egActive;
  bool               egNotifyOnlyOnce;
  bool               egIgnoreInactive;
  EVENTMESSAGE       egMsg;
  MESSAGEPARAM       egMsgParam;
  int                egRemoteEventListenerClassID;
  int                egRemoteEventListenerObjID;
  TRIGGER            egTrigger;
  TRIGGERPARAM       egTriggerParam;
  int                egTriggerParam1;
  int                egTriggerParam2;
} EVENTGENERATOR;
```

โดยมีรายละเอียดของข้อมูลที่เก็บในตัวแปลเหล่านี้ดังนี้

1. egID เก็บเลขจำนวนเต็มที่บอกถึงเลขประจำตัวของเหตุการณ์นี้
2. egActive เก็บค่าตรรกะที่บอกว่าเมื่อเกิดเหตุการณ์นี้ขึ้นจะให้ส่งไปยังวัตถุเป้าหมายหรือไม่
3. egNotifyOnlyOnce เก็บค่าตรรกะที่บอกว่าทันทีที่เหตุการณ์เกิดขึ้นแล้ววัตถุนี้จะส่งเหตุการณ์นี้ไปยังวัตถุเป้าหมายเพียงแค่ครั้งเดียวหรือไม่ ถ้ามีค่าเป็นจริงแล้วส่งไปไม่ถึงก็จะไม่มีการส่งซ้ำอีก
4. egIgnoreInactive เก็บค่าตรรกะที่บอกว่าเมื่อมีการส่งเหตุการณ์นี้ไปยังวัตถุเป้าหมาย แล้วเมื่อวัตถุเป้าหมายไม่ได้ทำงาน(Inactive) อยู่มันจะส่งอีกไปเรื่อยๆจนกว่าวัตถุนั้นจะรับเหตุการณ์นี้หรือไม่
5. egMSG เก็บค่าที่บอกถึงชนิดของเหตุการณ์ที่เกิดขึ้น
6. egMsgParam เก็บค่าที่บอกถึงชนิดของพารามิเตอร์ของชนิดของเหตุการณ์นั้นๆ
7. egRemoteEventListenerClassID เก็บค่าที่บอกถึงคลาสของวัตถุเป้าหมายที่เป็น Remote Event Listener Object
8. egRemoteEventListenerObjID เก็บค่าที่บอกถึงเลขประจำตัวของวัตถุเป้าหมายที่

เป็น Remote Event Listener Object

9. egTrigger เก็บค่าที่บอกถึงชนิดของการทำให้เกิดเหตุการณ์นี้ขึ้น
10. egTriggerParam เก็บค่าที่บอกถึงชนิดของพารามิเตอร์ของ egTrigger นี้
11. egTriggerParam1 เก็บค่าที่เป็นพารามิเตอร์ของ egTrigger นี้ ตัวที่ 1
12. egTriggerParam2 เก็บค่าที่เป็นพารามิเตอร์ของ egTrigger นี้ ตัวที่ 2

9.4.6 ส่วนหัวส่วนที่หกของแฟ้มข้อมูลที่เก็บข้อมูลที่ใช้ในการรับเหตุการณ์ต่างๆที่เกิดจากจากวัตถุที่เป็น Event Generator โดยเรียกวัตถุนี้ว่าเป็นว่า Remote Event Listener ซึ่งมีโครงสร้างดังนี้

```
typedef struct tagEventListener
{
  int                elID;
  bool               elActive;
  EVENTMESSAGE       elMsg;
  MESSAGEPARAM       elMsgParam;
  int                elSenderClassID;
  int                elSenderObjID;
  FUNCTION           elFunction;
  FUNCTIONPARAM      elFunctionParam;
  int                elFunctionParam1;
  int                elFunctionParam2;
  int                elFunctionParam3;
  int                elFunctionParam4;
  int                elFunctionParam5;
  int                elFunctionParam6;
} EVENTLISTENER;
```

โดยมีรายละเอียดของข้อมูลที่เก็บในตัวแปลเหล่านี้ดังนี้

1. elID เก็บเลขจำนวนเต็มที่บอกถึงเลขประจำตัวของเหตุการณ์นี้
2. elActive เก็บค่าตรรกะที่บอกว่าเมื่อมีเหตุการณ์ส่งมาจาก Event Generator แล้วจะรับหรือไม่
3. elMSG เก็บค่าที่บอกถึงชนิดของเหตุการณ์ที่เกิดขึ้น
4. elMsgParam เก็บค่าที่บอกถึงชนิดของพารามิเตอร์ของชนิดของเหตุการณ์นั้นๆ
5. elSenderClassID เก็บค่าที่เป็นคลาสของผู้ส่งซึ่งก็คือ Event Generator

6. elSenderObjID เก็บค่าที่เป็นเลขประจำตัวของวัตถุผู้ส่งซึ่งก็คือ Event Generator
7. elFunction เก็บค่าที่บอกชนิดของการทำงานที่จะเกิดขึ้นเมื่อได้รับเหตุการณ์จากผู้ส่งแล้ว
8. elFunctionParam เก็บค่าที่บอกชนิดของพารามิเตอร์ของ elFunction นี้
9. elFunctionParam1 เก็บค่าที่เป็นพารามิเตอร์ของ elFunction ตัวที่ 1
10. elFunctionParam2 เก็บค่าที่เป็นพารามิเตอร์ของ elFunction ตัวที่ 2
11. elFunctionParam3 เก็บค่าที่เป็นพารามิเตอร์ของ elFunction ตัวที่ 3
12. elFunctionParam4 เก็บค่าที่เป็นพารามิเตอร์ของ elFunction ตัวที่ 4
13. elFunctionParam5 เก็บค่าที่เป็นพารามิเตอร์ของ elFunction ตัวที่ 5
14. elFunctionParam6 เก็บค่าที่เป็นพารามิเตอร์ของ elFunction ตัวที่ 6

9.4.7 ส่วนหัวส่วนที่เจ็ดของแฟ้มข้อมูลที่เก็บข้อมูลต่างๆที่ใช้ในการกำหนดค่าต่างๆของรูปภาพที่มีอยู่ในวัตถุนั้น(Tile) ซึ่งมีโครงสร้างดังนี้

```
typedef struct tagTileHeader
{
  char          tSignature[4];
  int           tVersion;
  int           tID;
  char          tName[20];
  int           tBlockWidth;
  int           tBlockHeight;
  int           tBlockNumbers;
  int           tColorKey;
  int           tMaskSize;
} TILEHEADER;
```

โดยมีรายละเอียดของข้อมูลที่เก็บในตัวแปลเหล่านี้ดังนี้

1. tSignature เก็บอักขระ 4 ตัวที่เป็นสัญลักษณ์ของแฟ้มข้อมูลนี้
2. tVersion เก็บเลขจำนวนเต็มที่เป็นหมายเลขรุ่นของแฟ้มข้อมูลนี้
3. tID เก็บเลขจำนวนเต็มที่เป็นหมายเลขลำดับของแฟ้มข้อมูลนี้(ในกรณีมีหลายแฟ้ม)
4. tName เก็บอักขระ 20 ตัวที่เป็นชื่อของไทล์นี้
5. tBlockWidth เก็บเลขจำนวนเต็มที่เป็นความกว้างของบล็อกของภาพย่อยที่ถูกแบ่งออกเป็นเฟรม

6. tBlockHeight เก็บเลขจำนวนเต็มที่เป็นความสูงของบล็อกของภาพย่อยที่ถูกแบ่งออกเป็นเฟรม
7. tBlockNumbers เก็บเลขจำนวนเต็มที่เป็นจำนวนบล็อกย่อยๆทั้งหมดใน 1 ไทล์
8. tColorKey เก็บเลขจำนวนเต็มของรหัสสีที่เป็นสีโปร่งใส
9. tMaskSize เก็บเลขจำนวนเต็มที่เป็นจำนวน Mask Bit ที่ใช้ใน 1 ไทล์
10.

9.4.8 ส่วนหัวส่วนที่แปดของแฟ้มข้อมูลที่เก็บข้อมูลของค่าต่างๆที่ใช้ในการทำ Mask Bit ซึ่งมีโครงสร้างดังนี้

```
typedef struct tagTileMaskHeader
{
  int           tmWidth;
  int           tmHeight;
  int           tmBitUsed;
} TILEMASKHEADER;
```

โดยมีรายละเอียดของข้อมูลที่เก็บในตัวแปลเหล่านี้ดังนี้

1. **tmWidth** เก็บเลขจำนวนเต็มที่เป็นค่าความกว้างขนาดของ Mask Bit แต่ละ Mask
2. **tmHeight** เก็บเลขจำนวนเต็มที่เป็นค่าความสูงขนาดของ Mask Bit แต่ละ Mask
3. **tmBitUsed** เก็บเลขจำนวนเต็มที่เป็นจำนวนบิตที่ใช้ต่อหนึ่ง Mask Bit

1. ส่วนที่เป็นข้อมูลรูปภาพทั้งหมดที่ใช้ในวัตถุนี้ที่เป็นรูปแบบมาตรฐานของบิทแมพ และในข้อมูลส่วนนี้จะถูกนำขึ้นมาใช้งานเวลาแสดงผลจริง

9.5 แฟ้มข้อมูลที่เก็บข้อมูลที่ใช้ในการกำหนดคุณสมบัติและการทำงานของฉากและแผนที่ ซึ่งถูกใช้ในการนำฉากหรือแผนที่ที่ต้องการขึ้นมาแสดงบนหน้าจอและใช้ในการกำหนดตำแหน่งของวัตถุทุกๆวัตถุที่อยู่ในเกมเมื่อเทียบกับฉากหรือแผนที่นั้นๆ โดยมีโครงสร้างดังนี้

```
typedef struct tagMapHeader
{
  char          msSignature[3];
  int           miVersion;
  int           muID;
  char          msName[16];
```

```
    char        msTileFilename[16];
    int         miTileNum;
    int         miWidth;
    int         miHeight;
    int         miMapPixelWidth;
    int         miMapPixelHeight;
    int         miTilePixelWidth;
    int         miTilePixelHeight;
    int         miScreen_TW;
    int         miScreen_TH;
    int         miScreen_W;
    int         miScreen_H;
    int         miSpeed;
    int         miDirection;
} MAPHEADER;
```

โดยมีรายละเอียดของข้อมูลที่เก็บในตัวแปลเหล่านี้ดังนี้

1. msSignature เก็บอักขระ 4 ตัวที่เป็นสัญลักษณ์ของแฟ้มข้อมูลนี้
2. miVersion เก็บเลขจำนวนเต็มที่เป็นหมายเลขรุ่นของแฟ้มข้อมูลนี้
3. muID เก็บเลขจำนวนเต็มที่เป็นหมายเลขลำดับของแฟ้มข้อมูลนี้(ในกรณีมีหลายแฟ้ม)
4. msName เก็บอักขระ 20 ตัวที่เป็นชื่อของฉากนี้
5. msTileFilename เก็บอักขระ 16 ตัวที่เป็นชื่อแฟ้มข้อมูลของไทล์ที่ใช้ในฉากหรือแผนที่นี้
6. msTilenum เก็บเลขจำนวนเต็มที่เป็นจำนวนของไทล์ทั้งหมดที่ใช้ในฉากหรือแผนที่นี้
7. miWidth เก็บเลขจำนวนเต็มที่เป็นความกว้างของฉากหรือแผนที่นี้ในหน่วยของบล็อก
8. miHeight เก็บเลขจำนวนเต็มที่เป็นความสูงของฉากหรือแผนที่นี้ในหน่วยของบล็อก
9. miMapPixelWidth เก็บเลขจำนวนเต็มที่เป็นความกว้างของฉากหรือแผนที่นี้ในหน่วยของพิกเซลล์
10. miMapPixelHeight เก็บเลขจำนวนเต็มที่เป็นความสูงของฉากหรือแผนที่นี้ในหน่วยของพิกเซลล์
11. miTilePixelWidth เก็บเลขจำนวนเต็มที่เป็นความกว้างของไทล์ที่ใช้ในฉากหรือแผนที่นี้ในหน่วยของพิกเซลล์
12. miTilePixelHeight เก็บเลขจำนวนเต็มที่เป็นความสูงของไทล์ที่ใช้ในฉากหรือแผนที่นี้ในหน่วยของพิกเซลล์

13. miScreen_TW เก็บเลขจำนวนเต็มที่เป็นความกว้างของจอแสดงผลที่ใช้ในฉากหรือแผนที่นี้ในหน่วยของไทล์
14. miScreen_TH เก็บเลขจำนวนเต็มที่เป็นความสูงของจอแสดงผลที่ใช้ในฉากหรือแผนที่นี้ในหน่วยของไทล์
15. miScreen_W เก็บเลขจำนวนเต็มที่เป็นความกว้างของจอแสดงผลที่ใช้ในฉากหรือแผนที่นี้ในหน่วยของพิกเซลล์
16. miScreen_H เก็บเลขจำนวนเต็มที่เป็นความสูงของจอแสดงผลที่ใช้ในฉากหรือแผนที่นี้ในหน่วยของพิกเซลล์
17. miSpeed เก็บเลขจำนวนเต็มที่เป็นความเร็วของการเลื่อนฉากและแผนที่ในเกม
18. miDirection เก็บเลขจำนวนเต็มที่เป็นทิศทางของการเลื่อนฉากและแผนที่ในเกม

บทที่ 10
โครงสร้างของแฟ้มข้อมูลที่ใช้ในโปรแกรมสร้างและพัฒนาเกมสำเร็จรูป

สำหรับตัวโปรแกรมสร้างและพัฒนาเกมสำเร็จรูปจะต้องมีการเก็บข้อมูลต่าง ๆ ที่ทำการสร้างขึ้นจากการสร้างเกมในรูปแบบของแฟ้มข้อมูล ซึ่งจะประกอบด้วยแฟ้มข้อมูลต่าง ๆ ดังนี้

10.1 แฟ้มข้อมูลของของคลาส GDKGameHeader จะมีโครงสร้างของข้อมูลดังนี้

GDKGameHeader

โดยในแต่ละตัวแปรจะมีหน้าที่ดังต่อไปนี้ คือ

GDKGameHeader จะเป็นส่วนเก็บข้อมูลแบบเป็นโครงสร้างข้อมูลของตัวเกมเพื่อใช้ในตัวโปรแกรมสร้างและพัฒนาเกมสำเร็จรูป ซึ่งในโครงสร้างข้อมูลนั้นมีรายละเอียดย่อยดังนี้

```
typedef struct tagGDKGameHeader
{
        char    msSignature[3];
        int     miVersion;              //Map version
        char    msName[16];             //Map Filename
        char    msGameDirectory[255];
} GDKGAMEHEADER;
```

msSignature จะเป็นส่วนที่เก็บอักขระ 3 ตัวอักษรที่เป็นสัญลักษณ์ในการแสดงความเป็นแฟ้มข้อมูลของคลาส GDKGame ซึ่งจะมีสัญลักษณ์เป็น “GMF”

miVersion จะเป็นส่วนที่เก็บเลขจำนวนเต็มที่ใช้บอกหมายเลขรุ่นของแฟ้มข้อมูล

msName จะเป็นส่วนที่เก็บอักขระ 16 ตัวอักษร ที่บอกชื่อของเกมที่เก็บข้อมูลลงสู่แฟ้มข้อมูล

msGameDirectory จะเป็นส่วนที่เก็บอักขระ 255 ตัวอักษรเพื่อบอกไดเรกทอรีรากที่ใช้เก็บข้อมูลของเกม

10.2 แฟ้มข้อมูลของคลาส GGOSTool จะมีโครงสร้างของแฟ้มข้อมูลดังต่อไปนี้

GameSystemEnv

SerialID

โดยในแต่ละตัวแปรจะมีหน้าที่ดังต่อไปนี้

GameSystemEnv จะเป็นส่วนที่เก็บโครงสร้างข้อมูลที่เป็นแบบ GAMESYSTEMENV ซึ่งเป็นข้อมูลที่ได้รับมาจากคลาส GGOS ซึ่งจะมีรายละเอียดดังที่ได้กล่าวไว้ในส่วนโครงสร้างของแฟ้มข้อมูลที่ใช้ในโปรแกรมรันไทม์

SerialID เป็นตัวแปรที่เก็บเลขจำนวนเต็มที่ทำหน้าที่ในการบอกหมายเลขรหัสประจำตัวของวัตถุที่ใช้เป็นหมายเลขสุดท้ายในเกมเพื่อที่จะใช้ในการกำหนดรหัสประจำวัตถุให้กับวัตถุตัวใหม่ที่ถูกเพิ่มเข้ามาในเกม

10.3 แฟ้มข้อมูลของคลาส GGShootingTool จะมีโครงสร้างของแฟ้มข้อมูลดังต่อไปนี้

ShootingEnv

โดยในแต่ละตัวแปรจะมีหน้าที่ดังต่อไปนี้

ShootingEnv จะเป็นส่วนที่เก็บโครงสร้างข้อมูลที่เป็นแบบ SHOOTINGGAMEENV ซึ่งเป็นข้อมูลที่ได้รับมาจากคลาส GGShooting ซึ่งจะมีรายละเอียดดังที่ได้กล่าวไว้ในส่วนโครงสร้างของแฟ้มข้อมูลที่ใช้ในโปรแกรมรันไทม์

10.4 แฟ้มข้อมูลของคลาส GGMap จะมีโครงสร้างของแฟ้มข้อมูลดังต่อไปนี้

gmHeader

gmsMapList

โดยแต่ละตัวแปรจะมีหน้าที่ดังต่อไปนี้

gmsHeader จะเป็นส่วนที่เก็บโครงสร้างข้อมูลที่เป็นแบบ GLOBALMAPHEADER ซึ่งเป็นข้อมูลที่จะนำไปใช้กับคลาส GGOS ในขณะรันไทม์

gmsMapList จะเป็นตัวแปรแบบอักขระที่ทำหน้าที่ในการเก็บรายชื่อแฟ้มข้อมูลของฉากทั้งหมดที่มีอยู่ภายในเกม

10.5 แฟ้มข้อมูลของคลาส GGStaticObject จะมีโครงสร้างของแฟ้มข้อมูลดังต่อไปนี้

gmHeader

gmsStaticObjectList

โดยแต่ละตัวแปรจะมีหน้าที่ดังต่อไปนี้

gmHeader จะเป็นส่วนที่เก็บโครงสร้างข้อมูลที่เป็นแบบ GLOBALSTATICOBJECTHEADER ซึ่งเป็นข้อมูลที่จะนำไปใช้กับคลาส GGOS ในขณะรันไทม์

gmsStaticObjectList จะเป็นตัวแปรแบบอักขระที่ทำหน้าที่ในการเก็บรายชื่อแฟ้มข้อมูลของวัตถุที่ไม่สามารถเคลื่อนที่ได้ทั้งหมดที่มีอยู่ภายในเกม

10.6 แฟ้มข้อมูลของคลาส GGDynamicObject จะมีโครงสร้างของแฟ้มข้อมูลดังต่อไปนี้

gmHeader

gmsDynamicObjectList

โดยแต่ละตัวแปรจะมีหน้าที่ดังต่อไปนี้

gmHeader จะเป็นส่วนที่เก็บข้อมูลที่เป็นแบบ GLOBALDYNAMICOBJECTHEADER ซึ่งเป็นข้อมูลที่จะนำไปใช้กับคลาส GGOS ในขณะรันไทม์

gmsDynamicObjectList จะเป็นตัวแปรแบบอักขระที่ทำหน้าที่ในการเก็บรายชื่อแฟ้มข้อมูลของวัตถุที่สามารถเคลื่อนที่ได้ทั้งหมดที่มีอยู่ภายในเกม

10.7 แฟ้มข้อมูลของคลาส GGPlayer จะมีโครงสร้างของแฟ้มข้อมูลดังต่อไปนี้

gmHeader

gmsPlayerList

โดยแต่ละตัวแปรจะมีหน้าที่ดังต่อไปนี้

gmHeader จะเป็นส่วนที่เก็บโครงสร้างข้อมูลที่เป็นแบบ GLOBALPLAYERHEADER ซึ่งเป็นข้อมูลที่จะนำไปใช้กับคลาส GGOS ในขณะรันไทม์

gmsPlayerList จะเป็นตัวแปรแบบอักขระที่ทำหน้าที่ในการเก็บรายชื่อแฟ้มข้อมูลของผู้เล่นทั้งหมดที่มีอยู่ภายในเกม

10.8 แฟ้มข้อมูลของชุดของไทล์ประเภทที่จะใช้เป็นไทล์ของวัตถุที่เคลื่อนที่ไม่ได้

tilestaticlist

โดยแต่ละตัวแปรจะมีหน้าที่ดังต่อไปนี้

tilestaticlist จะเป็นส่วนที่เก็บโครงสร้างข้อมูลที่เป็นแบบ TILELISTHEADER เพื่อที่จะใช้ในการเก็บรายละเอียดชุดของไทล์ที่จะใช้เป็นไทล์ของวัตถุที่เคลื่อนที่ไม่ได้ในตัวโปรแกรมสร้างและพัฒนาเกมสำเร็จรูป ซึ่งจะมีส่วนประกอบย่อยดังต่อไปนี้

```
typedef struct tagTileListHeader
{
        char    tlsSignature[3];
        AnsiString tlsType;
        int tliTileNumber;
} TILELISTHEADER;
```

tleSignature จะเป็นตัวแปรแบบอักขระขนาด 3 ตัวอักษรที่เป็นสัญลักษณ์บอกว่าเป็นแฟ้มข้อมูลที่ใช้เก็บชุดของข้อมูลแบบไทล์ และมีสัญลักษณ์เป็น “TLS”

tlsType จะเป็นตัวแปรแบบอักขระที่ทำหน้าที่ในการแสดงประเภทของข้อมูลที่เก็บในชุดของไทล์ชุดนี้ ซึ่งในกรณีนี้จะมีค่าเป็น “ Statics “

tliTileNumber จะเป็นตัวแปรแบบเลขจำนวนเต็มที่จะเก็บจำนวนของไทล์แบบวัตถุที่ไม่สามารถเคลื่อนที่ได้ในชุดของไทล์ชุดนี้

10.9 แฟ้มข้อมูลของชุดของไทล์ประเภทที่จะใช้เป็นไทล์ของวัตถุที่เคลื่อนที่ได้

tiledynamiclist

โดยแต่ละตัวแปรจะมีหน้าที่ดังต่อไปนี้

tiledynamiclist จะเป็นส่วนที่เก็บโครงสร้างข้อมูลที่เป็นแบบ TILELISTHEADER เพื่อที่จะใช้ในการเก็บรายละเอียดชุดของไทล์ที่จะใช้เป็นไทล์ของวัตถุที่เคลื่อนที่ได้ในตัวโปรแกรมสร้างและพัฒนาเกมสำเร็จรูป ซึ่งจะมีส่วนประกอบย่อยดังต่อไปนี้

```
typedef struct tagTileListHeader
{
        char    tlsSignature[3];
        AnsiString tlsType;
        int tliTileNumber;
} TILELISTHEADER;
```

tleSignature จะเป็นตัวแปรแบบอักขระขนาด 3 ตัวอักษรที่เป็นสัญลักษณ์บอกว่าเป็นแฟ้มข้อมูลที่ใช้เก็บชุดของข้อมูลแบบไทล์ และมีสัญลักษณ์เป็น “TLS”

tlsType จะเป็นตัวแปรแบบอักขระที่ทำหน้าที่ในการแสดงประเภทของข้อมูลที่เก็บในชุดของไทล์ชุดนี้ ซึ่งในกรณีนี้จะมีค่าเป็น “ Dynamic ”

tliTileNumber จะเป็นตัวแปรแบบเลขจำนวนเต็มที่จะเก็บจำนวนของไทล์แบบวัตถุที่สามารถเคลื่อนที่ได้ในชุดของไทล์ชุดนี้

สำหรับแฟ้มข้อมูลประเภทอื่น ๆ ที่จำเป็นต้องใช้ในตัวโปรแกรมสร้างและพัฒนาเกมสำเร็จรูปก็จะเป็นแฟ้มข้อมูลประเภท แฟ้มข้อมูลของฉากต่าง ๆ (นามสกุล *.Map) , แฟ้มข้อมูลวัตถุ (นามสกุล *.Obj) , แฟ้มข้อมูลไทล์ (นามสกุล *.tle) ซึ่งได้อธิบายไว้ในส่วนของโปรแกรมรันไทม์

บทที่ 11
แนวคิดในการพัฒนาโปรแกรมสร้างและพัฒนาเกมสำเร็จรูป
การติดตั้งโปรแกรม และ
ขั้นตอนใช้งานโปรแกรมสร้างและพัฒนาเกมสำเร็จรูป

11.1 แนวคิดที่ใช้ในการพัฒนาตัวโปรแกรมสร้างและพัฒนาเกมสำเร็จรูป

สำหรับหน้าที่หลักของตัวโปรแกรมสร้างและพัฒนาเกมสำเร็จรูปนั้นอย่างที่ได้ทราบกันแล้วว่ามีหน้าที่สำคัญ คือ การทำการสร้างฐานข้อมูลจากไฟล์ข้อมูลรูปภาพต่าง ๆ ตามที่ผู้ใช้งานต้องการเพื่อทำการจัดเก็บให้อยู่ในรูปแบบที่ได้กำหนดมาตรฐานไว้ เพื่อใช้ในตัวโปรแกรมส่วนรันไทม์นำมาสร้างเป็นเกมตามที่ผู้ใช้งานต้องการ ดังนั้นแนวคิดในส่วนนี้ก็จะเป็นตัวผลักดันให้เกิดรูปแบบของโปรแกรมที่ประกอบด้วยฟอร์มของข้อมูลต่าง ๆ ที่จะนำมาให้ผู้ใช้ทำการกรอกข้อมูลตามที่ผู้ใช้ต้องการ แล้วทำการบันทึกข้อมูลต่าง ๆ นั้นลงในฐานข้อมูลไฟล์รูปแบบต่าง ๆ

ซึ่งจะสังเกตเห็นได้ว่าในส่วนของตัวโปรแกรมสร้างและพัฒนาเกมสำเร็จรูปจะประกอบด้วยฟอร์มที่ใช้รับข้อมูลเป็นจำนวนมาก แต่ก็จะประกอบด้วยฟอร์มที่ทำหน้าที่เป็นส่วนหลัก ๆ อยู่ 2 ฟอร์ม ดังจะได้ยกตัวอย่างดังต่อไปนี้คือ ฟอร์มที่ 1 ซึ่งเป็นฟอร์มที่ทำหน้าที่ในการแสดงผลและเก็บเมธอดที่เป็นการทำงานหลักของเมนูต่าง ๆ ที่มีในตัวโปรแกรม และ ตัวฟอร์มที่ใช้ในการสร้างฉากหรือแผนที่ของเกม ซึ่งเป็นฟอร์มที่ประกอบด้วยเมธอดที่ทำหน้าที่ในการสร้างและเชื่อมต่อกับฟอร์มต่าง ๆ ที่เป็นฟอร์มที่รับข้อมูลทั้งหมดในตัวโปรแกรม เป็นต้น นอกจากลักษณะของตัวโปรแกรมจะเป็นในรูปแบบเป็นฟอร์มกรอกข้อมูลแล้ว ยังมีลักษณะพิเศษอีกรูปแบบหนึ่งคือ การเขียนโปรแกรมจะเป็นประเภท อ้างอิงระบบการเกิดเหตุการณ์ (event base system) นั่นก็คือ ตัวโปรแกรมจะทำงานเป็นส่วน ๆ ตามเหตุการณ์ที่เกิดขึ้น เช่น การเลือกคำสั่งการบันทึกข้อมูลโดยเลือกปุ่มบันทึกข้อมูล (save button) ตัวโปรแกรมก็จะทำการเรียกเมธอดการทำงานของ SaveButtonClick ขึ้นมาทำงานเป็นต้น

11.2 การติดตั้งโปรแกรมสร้างและพัฒนาเกมสำเร็จรูป

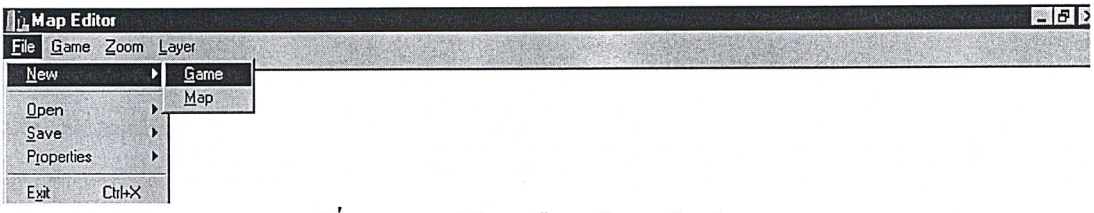
สำหรับในส่วนของการติดตั้งให้เรียกคำสั่ง Setup.exe ที่อยู่ในแผ่นดิสก์เพื่อทำการติดตั้งโปรแกรมลบบนฮาร์ดดิสก์ และไฟล์ข้อมูลที่ทำการติดตั้งแล้วจะประกอบด้วย

- Gameedit.exe – ตัวโปรแกรมสร้างและพัฒนาเกมสำเร็จรูป
- Runtime.exe – ตัวโปรแกรมรันไทม์ที่ใช้ในการรันเกมที่ทำการสร้างขึ้นจากโปรแกรมสร้างและพัฒนาเกมสำเร็จรูป (การใช้งานให้ทำเรียกไฟล์ Runtime.exe ในไดเรกทอรีรากของเกมที่ได้ทำการสร้างขึ้น)
- Map.tle ตัวอย่างไฟล์ไทล์ของฉากที่ใช้ในเกมตัวอย่าง (ถ้าต้องการไทล์รูปแบบอื่นให้ทำการสร้างขึ้นใหม่โดยใช้โปรแกรมจัดการรูปภาพตัวอื่น ๆ เช่น photoshop เป็นต้น
- ExampleBMP เป็นไดเรกทอรีที่เก็บตัวอย่างไฟล์ภาพแบบบิทแม็พ เอาไว้
- ExampleMap เป็นไดเรกทอรีที่เก็บตัวอย่างไฟล์ฉากที่ได้ทำการสร้างขึ้น

- ExampleGame เป็นไดเรกทอรีที่เก็บตัวอย่างเกมที่ถูกสร้างขึ้นโดยใช้โปรแกรมสร้างและพัฒนาเกมสำเร็จรูปเอาไว้
- Sourcecode เป็นไดเรกทอรีที่เก็บไฟล์โค้ดทั้งหมดของโปรแกรมซึ่งจะประกอบด้วยไดเรกทอรีย่อยอีก 2 ไดเรกทอรี คือ ไดเรกทอรี Gameruntime และ ไดเรกทอรี Gameedit
- DxSDK เป็นไดเรกทอรีที่เก็บไฟล์ของชุดพัฒนาโปรแกรมบนไดเร็กเอ็กด้วยภาษาซีซึ่งประกอบด้วยไดเรกทอรีไฟล์ไลบรารีและไฟล์Include ของชุดพัฒนาโปรแกรมบนไดเร็กเอ็ก

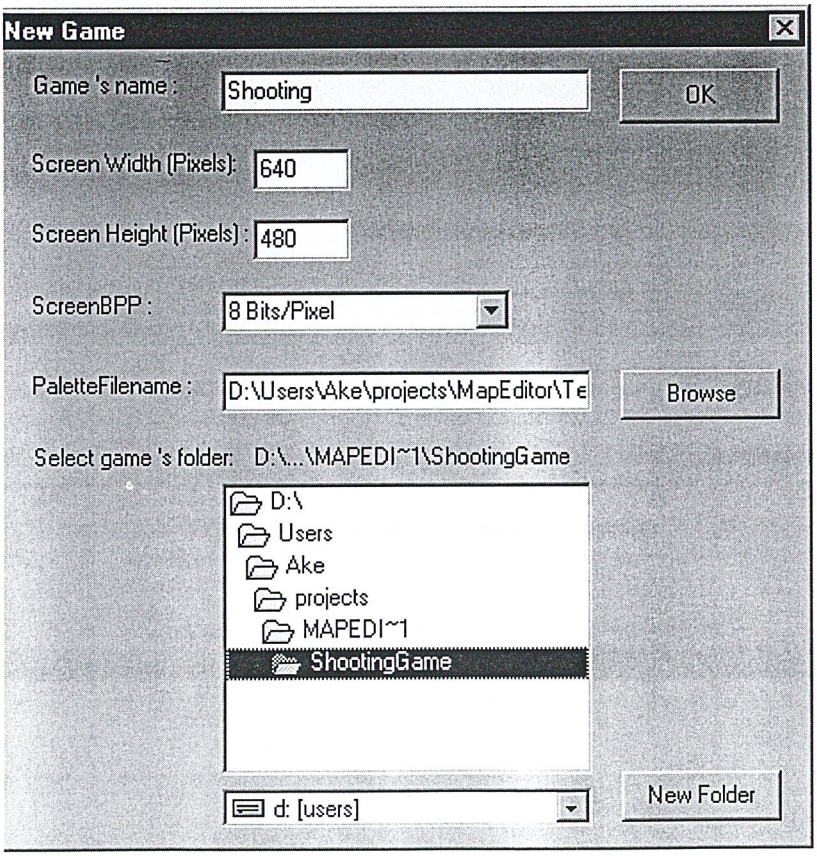
11.3 ขั้นตอนการสร้างและพัฒนาเกมโดยใช้โปรแกรมสร้างและพัฒนาเกมสำเร็จรูป

1 ขั้นตอนแรกให้ทำการสร้างเกมใหม่ โดยการเลือกที่เมนูแล้วเลือกที่หัวข้อ File แล้วเลือกที่หัวข้อ New แล้วเลือก Game



รูปที่11.1 แสดงวิธีการเลือกสร้างเกมใหม่

2 ทำการใส่ข้อมูลพื้นฐานของเกมให้ครบตามที่โปรแกรมต้องการแล้วเลือกปุ่ม OK ดังภาพ

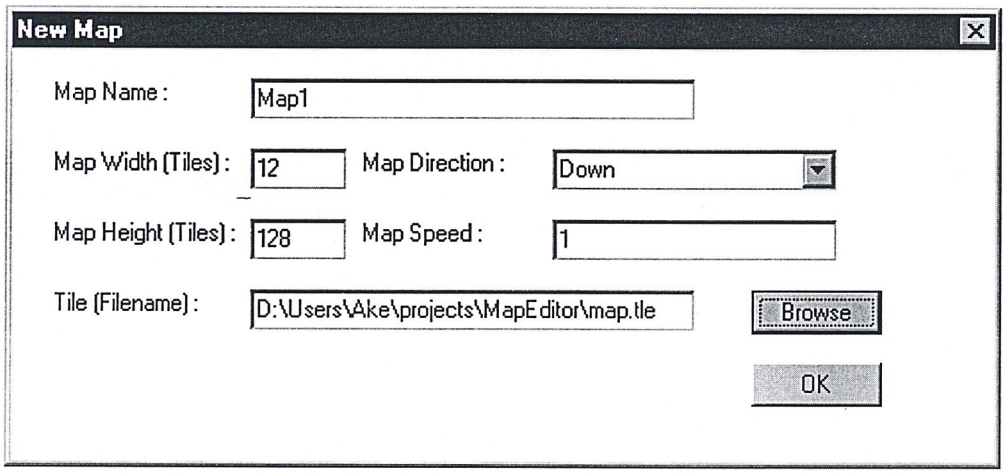


รูปที่11.2 แสดงวิธีการใส่ข้อมูลในการสร้างเกมใหม่

3 ทำการสร้างฉาก โดยการเลือกที่เมนูแล้วเลือกที่หัวข้อ File แล้วเลือกหัวข้อ New แล้วเลือก Map หรืออาจทำการโหลดฉากขึ้นมาจากแฟ้มข้อมูลฉากก็ได้ โดยเลือกที่หัวข้อ Open ตรงเมนูแล้วเลือก Map จากนั้นก็เลือกเปิดแฟ้มข้อมูลตามที่ต้องการ

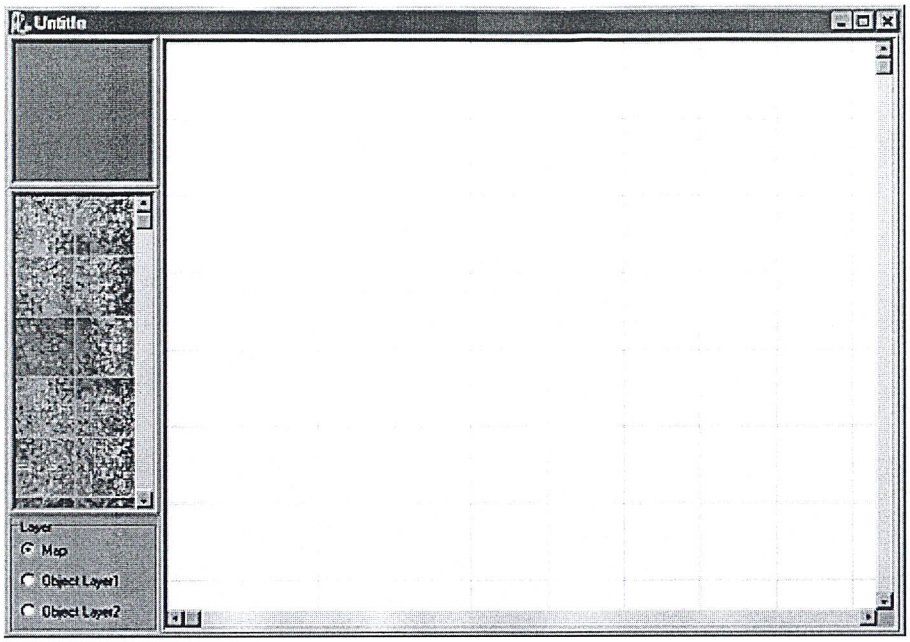
รูปที่11.3 แสดงวิธีการเลือกสร้างฉากใหม่

4 ทำการใส่ข้อมูลพื้นฐานของฉากให้ครบตามที่โปรแกรมต้องการแล้วเลือกปุ่ม OK ดังภาพ



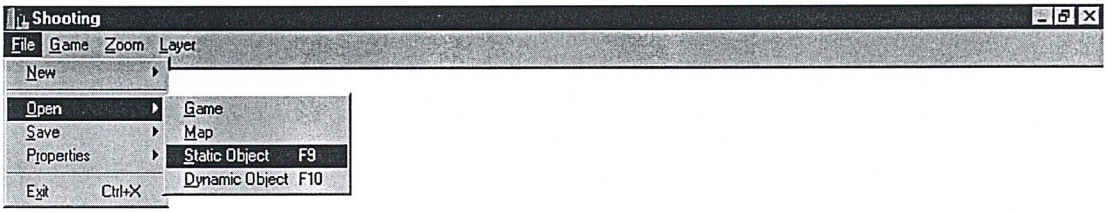
รูปที่11.4 แสดงวิธีการใส่ข้อมูลในการสร้างฉากใหม่

5 การสร้างฉากทำโดยการเลือกไทล์ที่ต้องการจากไทล์ลิสต์ที่อยู่ทางด้านซ้าย เมื่อเลือกแล้วจะปรากฏไทล์นั้นในช่องพรีวิวทางด้านซ้ายบนของฟอร์ม จากนั้นก็เลือกช่องของไทล์ตรง MapGrid ทางด้านขวาตามที่ต้องการ(โดยที่ตำแหน่งของ Layer ในกรอบ Layer ทางซ้ายล่างต้องอยู่ที่ Map เท่านั้น)



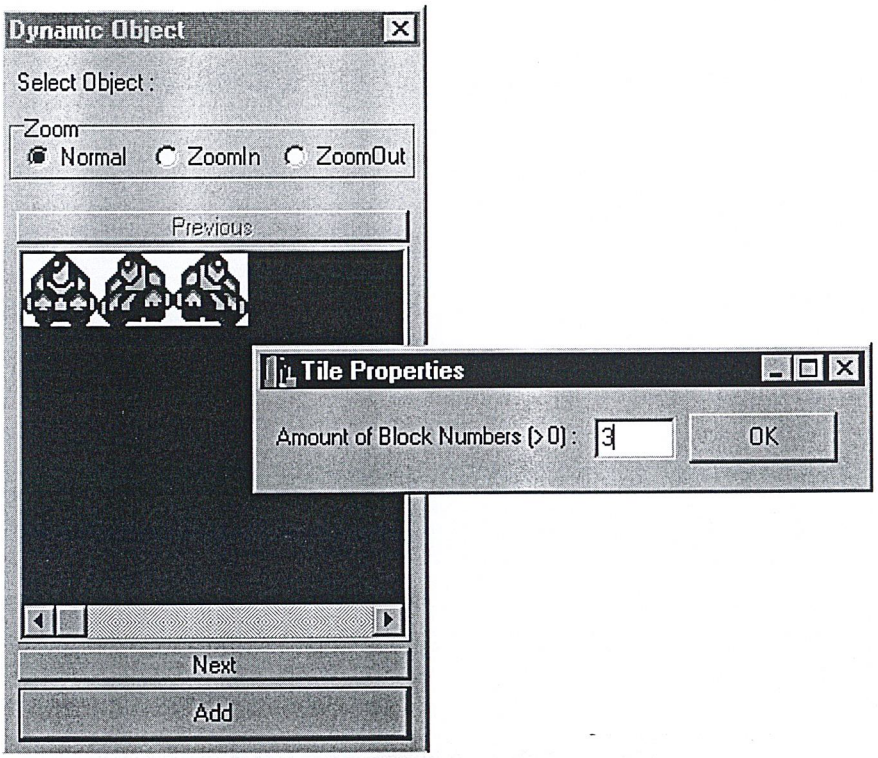
รูปที่11.5 แสดงวิธีการสร้างฉาก

6 การสร้างวัตถุให้ทำการเลือกเมนู File แล้วเลือก Open จากนั้นเลือก Static Object เพื่อเรียกฟอร์มลิสต์ของวัตถุที่ไม่สามารถเคลื่อนที่ได้ออกมา(ถ้าเลือก Dynamic Object จะทำการเรียกฟอร์มลิสต์ของวัตถุที่สามารถเคลื่อนที่ได้ออกมา)



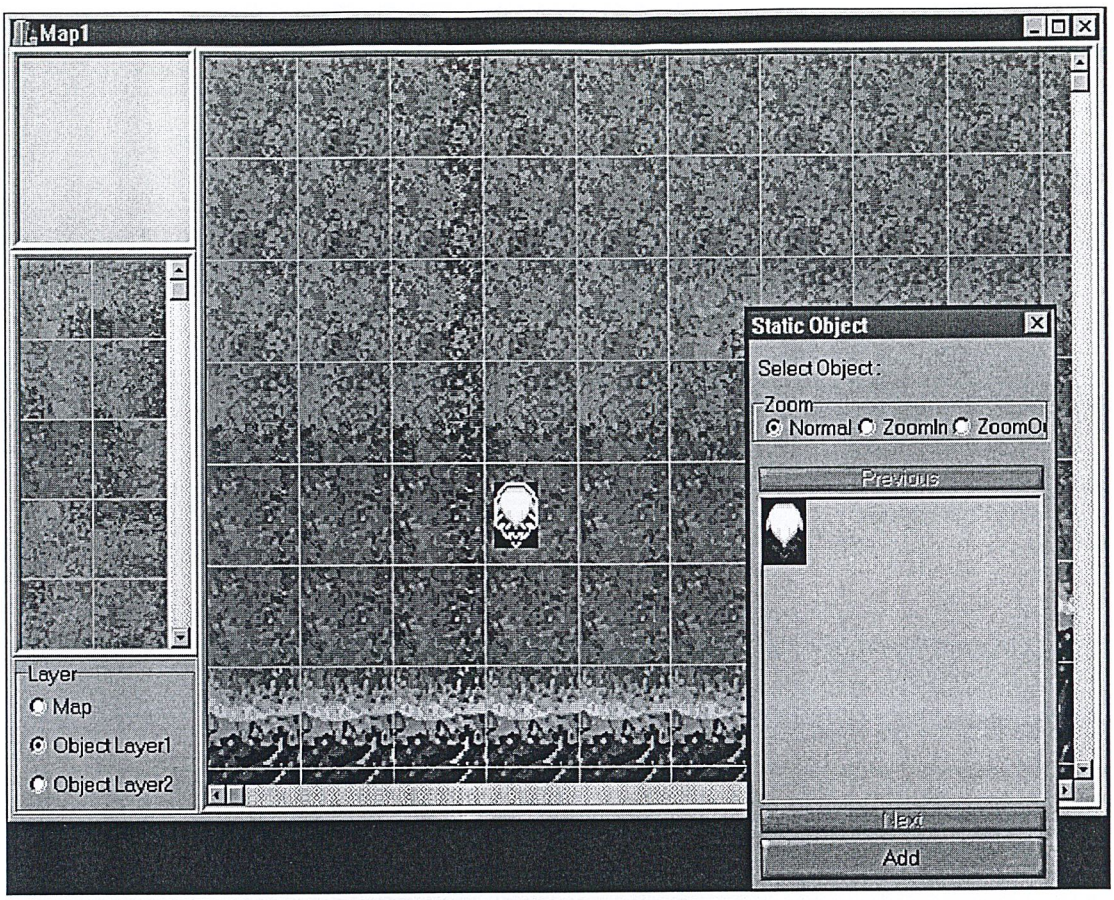
รูปที่11.6 แสดงวิธีการเรียกฟอร์มลิสต์ของวัตถุที่ไม่สามารถเคลื่อนที่ได้

7 ส่วนวิธีการเพิ่มวัตถุลงในลิสต์นั้นให้ทำโดยการกดปุ่ม Add บนลิสต์แล้วเลือกภาพที่ต้องการ จากนั้นกด OK แล้วโปรแกรมจะขึ้นข้อความถามจำนวนเฟรมของภาพ ให้ใส่ข้อมูลที่ถูกต้องตามภาพที่เลือกเข้าไป จากนั้นกด OK



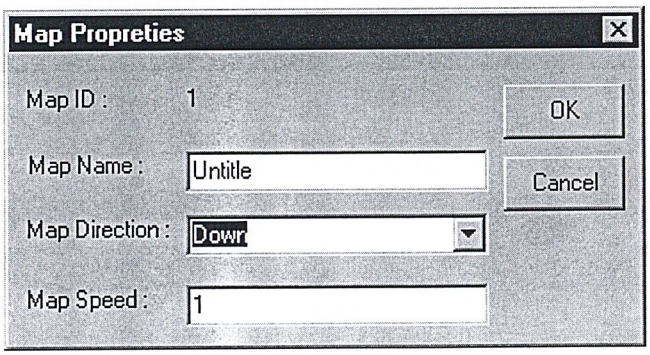
รูปที่11.7 แสดงวิธีการเพิ่มวัตถุลงในลิสต์

8 วิธีในการวางวัตถุลงบนฉากก็ทำโดยการเลือกที่รูปภาพของวัตถุในลิสต์ จากนั้นทำการกดปุ่มซ้ายของเมาส์ (กดปุ่ม Ctrl ของแป้นพิมพ์ค้างไว้ด้วยขณะกดปุ่มซ้ายของเมาส์) เพื่อทำการเพิ่มวัตถุลงในฉากที่ตัว MapGrid ตามที่ได้ออกแบบไว้ (โดยที่ตำแหน่งของ Layer ในกรอบ Layer ทางซ้ายล่างต้องอยู่ที่ Object Layer1 , Object Layer2 เท่านั้น)



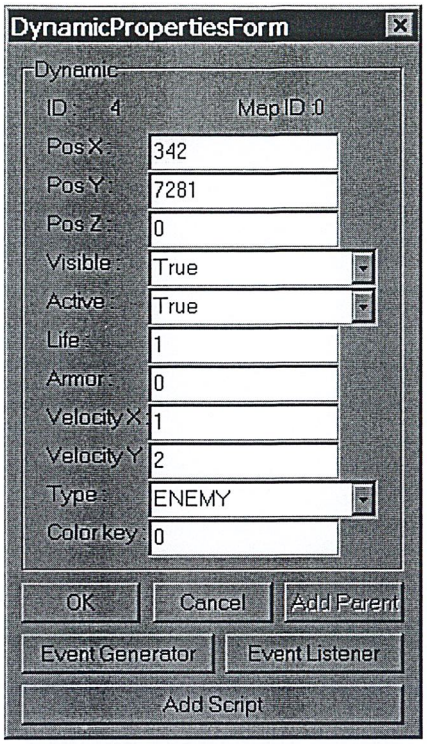
รูปที่11.8 แสดงวิธีการเพิ่มวัตถุลงในฉาก

9 วิธีในการเลือกเปลี่ยนค่าคุณสมบัติของเกมและฉาก ให้เลือกที่เมนูตรงคำสั่ง File แล้วเลือก Properties จากนั้นก็เลือกคำสั่ง Game หรือ Map ที่ต้องการแก้คุณสมบัติหลังจากนั้นก็ทำการใส่ข้อมูลลงในฟอร์มที่เปิดขึ้นมาแล้วกด OK



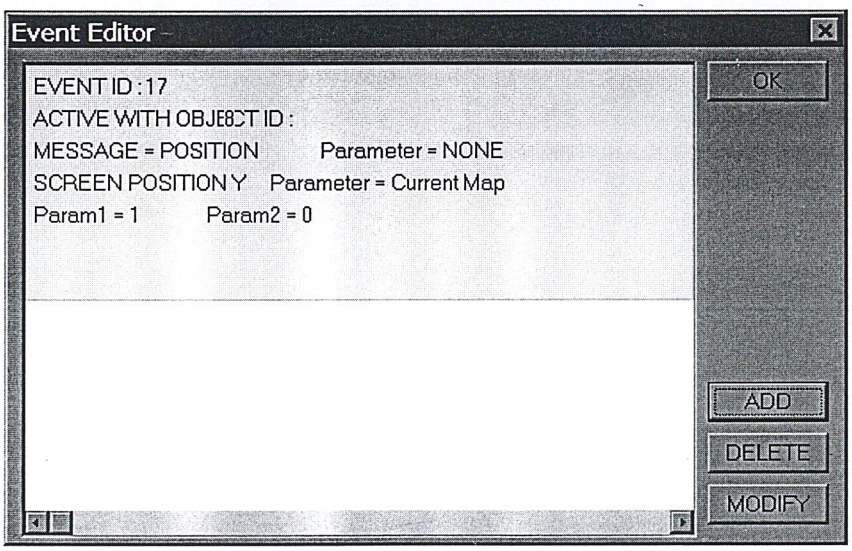
รูปที่11.9 แสดงวิธีการแก้คุณสมบัติของเกมและฉาก

10 วิธีในการเปลี่ยนค่าคุณสมบัติของตัววัตถุ ทำโดยการดับเบิ้ลคลิกที่ตัววัตถุแล้วโปรแกรมจะทำการเปิดฟอร์มคุณสมบัติของวัตถุที่ต้องการแก้ให้เองจากนั้นก็ทำการแก้ไขข้อมูลแล้วกด OK



รูปที่ 11.10 รูปแสดงฟอร์มปรับแต่งค่าคุณสมบัติของวัตถุ

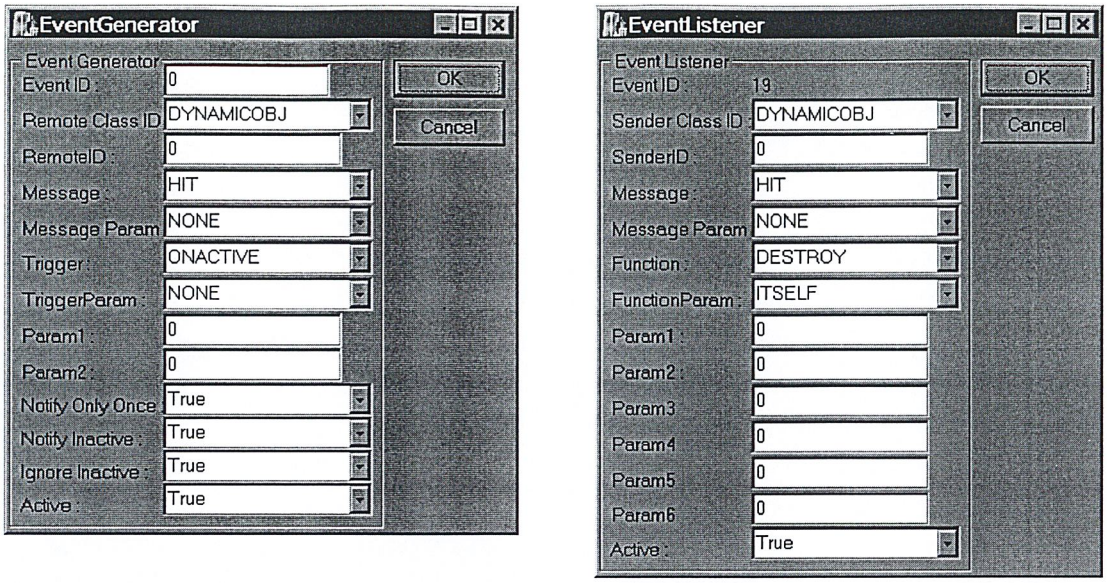
11 วิธีในการกำหนดเหตุการณ์ ให้ทำการเปิดฟอร์มปรับแต่งค่าคุณสมบัติของวัตถุจากนั้นทำการเลือกที่ปุ่ม Event แล้วโปรแกรมจะเปิดฟอร์มสร้างเหตุการณ์ขึ้นมาดังภาพ



รูปที่11.11 แสดงฟอร์มที่ใช้แสดงรายละเอียดเหตุการณ์

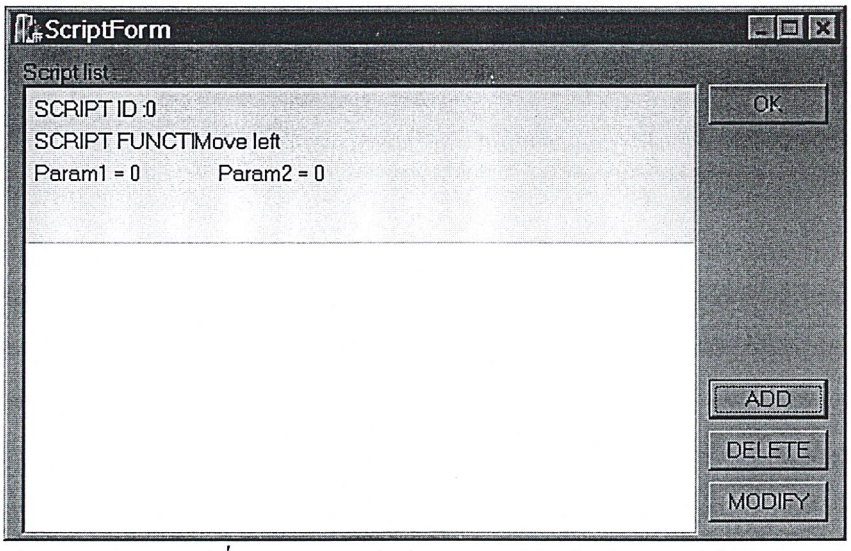
ต่าง ๆ ของวัตถุในเกม

12 การเพิ่มเหตุการณ์ทำได้โดยการกดปุ่ม ADD เมื่อกดแล้วโปรแกรมจะเปิดฟอร์มสร้างเหตุการณ์ขึ้นมา จากนั้นให้ทำการกำหนดเหตุการณ์ในทั้ง 2 ส่วนคือส่วนที่เป็น ตัวเริ่มเหตุการณ์ (event generator) และตัวกระทำตามเหตุการณ์ (event listener) ให้สอดคล้องกัน



รูปที่11.12 แสดงฟอร์มการสร้างเหตุการณ์

13.วิธีในการกำหนดสคริปท์ ให้ทำการกดปุ่มเลือกการปรับแต่งตัววัตถุจากนั้นทำการเลือกที่ปุ่ม Add Script เพื่อทำการแสดงฟอร์มแสดงรายละเอียดของสคริปท์ขึ้นมาดังรูป



รูปที่ 11.13 แสดงฟอร์มแสดงรายละเอียดของสคิปท์

14.การเพิ่มสคริปท์โดยการกดปุ่ม ADD ทำโดยการเลือกการสคริปท์ที่ต้องการจากฟอร์มการสร้างสคริปท์ แล้วกำหนดค่าของข้อมูลลงไปตามที่โปรแกรมต้องการ จากนั้นให้กดปุ่ม OK เพื่อทำการเพิ่มสคริปท์ลงในข้อมูลของวัตถุ

บทที่ 12
บทวิจารณ์และสรุป

จากการที่ได้ทำการสร้างและพัฒนาโปรแกรมสำเร็จรูปรวมถึงส่วนที่เป็นโปรแกรมรันไทม์ตามที่ได้ออกแบบไว้ ผลจากการทดลองใช้งานพบว่าตัวโปรแกรมนั้นสามารถทำการรันได้ตามปกติสามารถทำงานได้ที่ความเร็วสูงมากพอสมควรอันเนื่องมาจากการใช้ DirectX ช่วยในการทำงานขณะรันไทม์โดยเฉพาะฟังก์ชันของตัว DirectDraw ช่วยให้การแสดงผลเป็นไปได้อย่างรวดเร็ว ส่วนปัญหาที่พบก็คือวิธีในการใช้งานตัวโปรแกรมสร้างและพัฒนาเกมสำเร็จรูปจะทำการใช้งานได้ยากเนื่องจาก การขาดฟังก์ชันในการอำนวยความสะดวกในการใช้งานหลายอย่าง เช่น การคัดลอกฉากบางส่วนเพื่อไปทำการสร้าง ณ. ตำแหน่งอื่น ๆ ทำให้ผู้ใช้งานต้องทำการสร้างฉากที่มีลักษณะซ้ำกันบ่อยครั้งซึ่งทำให้เสียเวลา เป็นต้น ส่วนสาเหตุที่ไม่สามารถทำฟังก์ชันอำนวยความสะดวกเพิ่มในโปรแกรมได้เพราะว่าการออกแบบในขั้นตอนแรกไม่ได้ทำการออกแบบการทำงานของฟังก์ชันเอาไว้ก่อนทำให้ทำการเพิ่มเข้าไปในตัวโปรแกรมได้ยากเพราะความซับซ้อนของโปรแกรม นอกจากนั้นยังมีปัญหาในส่วนของเงื่อนไขของเหตุการณ์ต่าง ๆในบางอย่างค่อนข้างจะมีความซับซ้อนทำให้ผู้ที่ไม่มีความรู้ในเรื่องการกำหนดเหตุการณ์ให้เกิดการเชื่อมโยงของตัวเกมรวมถึงผู้ที่ไม่มีความคุ้นเคยกับฟังก์ชันในการสร้างเหตุการณ์ต่าง ๆ ไม่สามารถทำการสร้างเกมตามที่ต้องการ

สรุป โปรแกรมสร้างและพัฒนาเกมสำเร็จรูปที่ได้ทำการพัฒนาขึ้นนี้สามารถทำงานได้ตามต้องการในระดับที่น่าพอใจไม่ว่าจะเป็นในด้านความสามารถในการสร้างเกมประเภทยานยิง หรือความเร็วในการแสดงผล และรวมไปถึงการออกแบบโปรแกรมที่สามารถยืดหยุ่นและง่ายต่อการนำไปพัฒนาต่อไป แต่อาจมีปัญหาในเรื่องความสะดวกในการใช้งานอยู่บ้างสำหรับผู้ที่ไม่คุ้นเคยหรือไม่มีความชำนาญในการใช้งาน

ปัญหาที่พบในขณะทำการพัฒนาโปรแกรม

ปัญหาของการพัฒนาโปรแกรมสร้างและพัฒนาเกมสำเร็จรูปนั้นส่วนมากจะเป็นปัญหาที่เกิดจากความหลากหลายของฟังก์ชันการทำงานที่ต้องมีการออกแบบไว้เพื่อให้รองรับความต้องการทั้งหมดของผู้ใช้ ซึ่งบางฟังก์ชันก็สามารถทำงานได้ดีแต่ในบางฟังก์ชันที่มีความซับซ้อนหรือยุ่งยาก เช่น การกำหนดเหตุการณ์ต่าง ๆ ก็จะทำการพัฒนาได้ยุ่งยากเนื่องจากไม่สามารถคาดเดาความต้องการหรือแนวความคิดทั้งหมดที่ผู้ใช้จะนำมาทำการสร้างเกมทำให้ตัวโปรแกรมที่สร้างขึ้นอาจไม่สามารถรองรับการทำงานที่ต้องการของผู้ใช้ได้ทั้งหมดหรือได้แต่มีคุณภาพที่ไม่ดีพอและอาจทำให้เกิดความผิดพลาดขึ้นมาในตัวโปรแกรม

ภาคผนวก ก

คลาสหลักที่ใช้ในการทำงานของโปรแกรม

โครงสร้างของคลาสที่สำคัญต่างๆที่จำเป็นต่อการทำงานของโปรแกรมสร้างและพัฒนาเกมสำเร็จรูปและเกมรันไทม์จะอยู่ใน GDK.H ซึ่งจำเป็นต้อง include ในส่วนของโปรแกรม แสดงได้ดังข้างล่างนี้

```
//-----------------------------------------------------------------
#ifndef GDKH
#define GDKH
#include "GStruct.h"
#include "GDXClass.h"
#include "GDXError.h"
#include "GDXSurface.h"
#include "GDXScreen.h"
#include "GDXMap.h"
#include "GDXTile.h"
//-----------------------------------------------------------------
class GObject;
class GGOS;
class GEvent;
class GDynamicObject;
class GStaticObject;
class GPlayer;
class GChild;
class GEventGenerator;
class GScript;

class GRemoteEvent
{
  private:
    int EventID;
    EVENTMESSAGE Msg;
    MESSAGEPARAM MsgParam;
    GObject *Source;
```

```
    GObject *HandBack;

  public:
    GRemoteEvent(GObject *source,int eventID,EVENTMESSAGE msg,MESSAGEPARAM
MsgParam,GObject *handback);
    ~GRemoteEvent();
    inline GObject* GetSource(){return Source;}
    inline int GetMsg(){return Msg;}
    inline int GetMsgParam() {return MsgParam;}
    inline GObject* GetRegisTrationObject(){return HandBack;}
    inline int GetID(){return EventID;}
};

class GObject
{
    friend class GChild;
  protected:
    bool b_isGDX;  // if create by Tool =false ,create by game =true
    bool b_UseOutsideTile;  // if Constructure use for GTile will be true
                    // if true GTile will be created in GObject
                    // if false GTile must be PASSED from outside
    GDXScreen *m_Screen;
    GDXSurface *m_Surface;
    GMap *m_Map;
    LPRECT m_Rect;
    int m_ColorKey;
    static int CurrentID;
    const static int ClassID;
    bool b_isRemoteEventListener;
    bool b_isEventGenerator;
    bool b_hasScript;
    int ID;
    int ChildID;
```

```
    GGOS *m_GGOS;

protected:
    OBJECTHEADER m_ObjectHeader;
    OBJECTDATAHEADER m_ObjectDataHeader;
    OBJECTTILEDATAHEADER m_ObjectTileDataHeader;
    int m_Frame;
    int m_MapID;
    int m_PosX,m_PosY,m_PosZ;// Z for Layer ...
    int m_ScreenPosX,m_ScreenPosY;

protected:
    void InitialHeader(void);
    void UpdateData(void);
    bool MakeEventObject(int num,TMemoryStream *mem1);
    bool MakeGeneratorObject(int num,TMemoryStream *mem1);

public:
    GTile *m_GTile;// used for Object Tool
    GDXTile *m_Tile;
    GEvent *m_Event;// Used for RemoteEventListener
    GEventGenerator *m_Generator; // Used for EventGenerator
    GChild *m_Child; // If this Object is Main Child
    GScript *m_Script; // For Object Script
    bool Active;
    bool Visible;
    bool b_TimerRunning;

    // Default Constructure that must LoadFromFile used with OBJ Tool after
    GObject(void);
    // Constructure used for GDXTile ONLY with no header
    //GObject(GDXScreen *screen,const char *filename,int w,int h,int num);
```

```
        // Constructure used for creaet Object in defined Game OS
        // This Construct MUST use for create Object in Game Runtime
        GObject(GGOS *ggos,GDXScreen *screen);

        // Constructure used for GTile
        GObject(GTile *gtile);
        GObject(GObject *mainChild);
        ~GObject();
        void Show(void);
        void Show(GDXSurface *lpDDS);
        void Show(GDXSurface *lpDDS,LPRECT ClipRect);
        void Show(GDXSurface *lpDDS,GMap *gdxmap);
        void Show(GDXSurface *lpDDS,GMap *gdxmap,LPRECT ClipRect);
        inline void SetDefault(GDXSurface *lpDDS,GMap *gdxmap,LPRECT ClipRect)
{m_Surface=lpDDS;m_Map=gdxmap;m_Rect=ClipRect;}
        inline void SetFrame(int frame){m_Frame=frame;}
        inline void SetMapID(int map){m_MapID=map;}
        inline void SetMap(GMap **map,int mapid);
        inline int GetMapID(void){return m_MapID;}
        //inline static int GetClassID(){return ClassID;}
        // ClassID was save in ObjectHeader
        inline int GetClassID(){return m_ObjectHeader.oClassID;}
        inline int GetCurrentID(){return CurrentID;}
        inline void SetPos(int x,int y,int z){m_PosX=x;m_PosY=y;m_PosZ=z;}
        inline int GetPosX(){return m_PosX;}
        inline int GetPosY(){return m_PosY;}
        inline int GetPosZ(){return m_PosZ;}
        inline int GetScreenPosX(){return m_ScreenPosX;}
        inline int GetScreenPosY(){return m_ScreenPosY;}
        inline void SetPosX(int x){m_PosX=x;}
        inline void SetPosY(int y){m_PosY=y;}
        inline void SetPosZ(int z){m_PosZ=z;}
        inline void SetColorKey(int colorkey){m_ColorKey=colorkey;}
```

```
//inline void SetMovement(OBJMOVEMENT move){m_ObjectDataHeader.odMove=move;}
//inline OBJMOVEMENT GetMovement(void){return m_ObjectDataHeader.odMove;}
// Load Object File to screen with Header use for GDXTile in Game
virtual bool LoadFromFile(const char *filename);
// == Load Object File with Header used by Object Tool ONLY ==
virtual bool LoadToolFromFile(const char *filename);
// == These Blow Functions are used by Object Tool ONLY ==
virtual bool SaveToFile (const char *filename); // GTile Must be loaded !!
void SetObjectDataHeader();
// == For Set ID from Game Tool Only (before Saving)
// == If not Set ID before saving ,it will autogen by CurrentID
inline void SetID(int id){m_ObjectHeader.oID=id;}
// == For Get ID from Runtime ID Only !! (Generate by CurrentID)
inline int GetID(){return ID;}
inline int GetToolID() { return m_ObjectHeader.oID; }
// No setChildID() method cause of ChildID will be generated
// by GChild only!!
inline int GetChildID(){return ChildID;}
inline int GetParentID(){return m_ObjectHeader.oParentID;}
inline void SetParentID(int id){m_ObjectHeader.oParentID=id;}
// == Check For Normal Tile (Not have MASK bits ) ==
// For use in GDynamicObject or GStaticObject
// must be FORCE with (GObject*).
virtual bool Hit(GObject *gobject);

// == For Main Child Object ==
// If this Object is Main Child
// 1. Invisible and Inactive
// 2. HasChild = true;
virtual bool InitialChildObject();
inline void SetMainChild(bool b);
inline void SetChildPositionAsParent(bool b){m_ObjectDataHeader.odPosAsParent=b;}
inline void SetDieAsParent(bool b){m_ObjectDataHeader.odDieAsParent=b;}
```

```
inline void SetDestroyOnDie(bool b){m_ObjectDataHeader.odDestroyOnDie=b;}
inline void SetTotalChild(int count){m_ObjectDataHeader.odChildTotalCount=count;}
inline void SetRedupable(bool b){m_ObjectDataHeader.odCanRedup=b;}
inline void SetRelPosX(int x){m_ObjectDataHeader.odRelPosX=x;}
inline void SetRelPosY(int y){m_ObjectDataHeader.odRelPosY=y;}
inline void SetCreateActive(bool b){m_ObjectDataHeader.odCreateActive=b;}
inline void SetCreateVisible(bool b){m_ObjectDataHeader.odCreateVisible=b;}
inline bool isMainChild(){return m_ObjectDataHeader.odHasChild;}
inline bool isCreatePositionAsParent(){return m_ObjectDataHeader.odPosAsParent;}
inline bool isDieAsParent(){return m_ObjectDataHeader.odDieAsParent;}
inline int GetTotalChild(){return m_ObjectDataHeader.odChildTotalCount;}
inline bool isRedupable(){return m_ObjectDataHeader.odCanRedup;}

// == Game OS for this Object ===
inline void SetGameOS(GGOS *ggos){m_GGOS=ggos;}
inline GGOS *GetGameOS(){return m_GGOS;}

// == Timer Function ==
inline bool hasTimer(){return m_ObjectHeader.oHasTimer;}
void SetObjectTimer(UNIT time,bool
once){m_ObjectDataHeader.odTimer=time;m_ObjectHeader.oHasTimer=true;m_ObjectDataHe
ader.odTimerOnce=once;}
bool RunObjectTimer();
bool RunObjectTimer(UNIT time);
UNIT GetObjectTimer(){return m_ObjectDataHeader.odTimer;}
bool KillObjectTimer();
void OnTimer(DWORD time);

// ========== Object Running Functions ========
// 1. Run Script
// 2. Run EventGenerator if b_isEventGenerator
// ============================================
void Run();
```

```
    // ========== Event Functions ==========
    inline void SetEventNumbers(int num);
    inline void SetEventGeneratorNumbers(int num);
    inline bool isEventGenerator(){return b_isEventGenerator;}
    inline bool isRemoteEventListener(){return b_isRemoteEventListener;}
    void Notify(GRemoteEvent *remoteEvent); // use for RemoteEventListener
    bool RunGenerator(); // For runing EventGenerator

    // ========== Script Function ==========
    virtual void SetScriptNumbers(int num);
    virtual bool LoadScript(int num,TMemoryStream *stream);
    inline bool hasScript(){return b_hasScript;}
    bool RunScript();

    // ========== Object Movement Virtual Function ==========
    virtual bool MoveDown(){m_PosY++;}
    virtual bool MoveLeft(){m_PosX--;}
    virtual bool MoveRight(){m_PosX++;}
    virtual bool MoveUp(){m_PosY--;}

};

class GGOS
{
  protected:
    GAMESYSTEMENV GameSystemEnv;
    bool b_UseForTool;
    void *m_hInst;
    RECT m_Window;

    int m_MapCount;
    int m_DynamicObjectCount;
```

```
int m_StaticObjectCount;
int m_PlayerCount;
int m_CurrentMapDynamicObjectCount;
int m_CurrentMapStaticObjectCount;

GLOBALMAPHEADER gmHeader;
GLOBALOBJECTHEADER goDynamic;
GLOBALOBJECTHEADER goPlayer;
GLOBALOBJECTHEADER goStatic;

AnsiString *m_DynamicObjectfile;
AnsiString *m_StaticObjectfile;
AnsiString *m_Playerfile;
AnsiString *m_Mapfile;

AnsiString m_DynamicObjDirectory;
AnsiString m_PlayerDirectory;
AnsiString m_StaticObjDirectory;
AnsiString m_MapDirectory;

char *m_CurrentDir;

int m_CurrentMap;
int m_CurrentMap_StartObjectID;
int m_CurrentMap_TotalObjects;

GMap **m_Map;
// GDK Object Class Instance
GDynamicObject **m_DynamicObject;
GStaticObject **m_StaticObject;
GPlayer **m_Player;

//All Object for Current Map
```

```
    GDynamicObject **m_CurrentMapDynamicObject;
    GStaticObject **m_CurrentMapStaticObject;


protected:
    // ====== Global Functions ====== //
    bool MakeCurrentMapObjectList(int map);

    // ====== Map Functions ====== //
    bool MakeMap(char *allmapfile,int mapcount);

    // ====== Dynamic Object Functions ====== //
    bool MakeDynamicObject(char *allobjfile,int objcount);

    // ====== Player Functions ====== //
    bool MakePlayerObject(char *allobjfile,int objcount);

    // ====== Static Object Functions ====== //
    bool MakeStaticObject(char *allobjfile,int objcount);

public:
    void *m_hWnd;
    GDXScreen *m_Screen;

    // ====== Constructors ====== //
    GGOS(void *hInst,void *hWnd);
    GGOS(void); // Use for Tool

    // ====== Destructor ====== //
    ~GGOS();

    // ====== Game Operating System Functions ====== //
    bool LoadGameSystemEnv(const char *filename);
```

```
        bool InitialGameSystem();

        // Below Virtual function allow to make other Game style inherit from
        // this GGOS class
        virtual void UpdatePlayer(){}
        virtual void UpdateInput(){}
        virtual void Update();
        virtual void UpdateMap(){}
        virtual void UpdateObjects(){}

        // ===== Global Objects Functions ===== //

        // Check Object Hit To Any Object in Game
        // return ObjectID (runtime) for Object that be HIT
        // return -1 = Not Hit Any Object
        virtual int HitObject(GObject *gobject);
        // For Running each Object with below step.
        // 1. Run Object Script
        // 2. Run EventGenerator (if it is)
        void Run();

        // ===== Screen Functions ===== //
        inline void SetWindow(int xmin,int ymin,int xmax,int
ymax){SetRect(&m_Window,xmin,ymin,xmax,ymax);}
        inline void MakeScreen(int resx,int resy,int bpp){m_Screen = new GDXScreen
(m_hWnd,resx,resy,bpp);}

        // ===== Map Functions ===== //
        void ReleaseAllMap();
        inline GMap **GetMapList(){return m_Map;}
        inline GMap *GetMap(int n){return m_Map[n];}
        inline int GetCurrentMap(){return m_CurrentMap;}
        bool LoadMapFile(const char *filename); // Use for Global Map File
```

```
virtual bool SetCurrentMap(int map);

// ===== Base Object Functions ===== //
void ShowObject(GObject *object);
// Get Object that specified by ObjID and if not have
// any Object match ID it will return NULL
GObject *GetObject(int objID);

// ===== Dynamic Object Functions ===== //
inline int GetDynamicObjectCount(){return m_DynamicObjectCount;}
inline GDynamicObject **GetDynamicObjectList(){return m_DynamicObject;}
inline GDynamicObject *GetDynamicObject(int n){return m_DynamicObject[n];}
void ReleaseAllDynamicObject();
bool LoadDynamicObjectFile(const char *filename); // Use for Global Object File

// ===== Player Object Functions ===== //
inline int GetPlayerObjectCount(){return m_PlayerCount;}
inline GPlayer **GetPlayerObjectList(){return m_Player;}
inline GPlayer *GetPlayerObject(int n){return m_Player[n];}
bool LoadPlayerFile(const char *filename);
void ReleaseAllPlayerObject();

// ===== Static Object Functions ===== //
inline int GetStaticObjectCount(){return m_StaticObjectCount;}
inline GStaticObject **GetStaticObjectList(){return m_StaticObject;}
inline GStaticObject *GetStaticObject(int n){return m_StaticObject[n];}
bool LoadStaticObjectFile(const char *filename);
void ReleaseAllStaticObject();

// ===== Debug String to File ===== //
void Debug(AnsiString str,AnsiString filename);
void Debug(int n,AnsiString filename);
void Debug(char *str,AnsiString filename);
```

```
    // ===== Event Functions ===== //
    void RunGenerator();

};
class GDynamicObject : public GObject
{
  protected:
    const static int ClassID;

  public:
    DYNAMICOBJECTDATA m_DynamicObjectData;

    GDynamicObject(void);
    GDynamicObject(GDynamicObject *mainChild);
    GDynamicObject(GGOS *ggos,GDXScreen *screen);
    GDynamicObject(GTile *tile);
    ~GDynamicObject(){}
    inline void SetVelX(int velx){m_DynamicObjectData.dodVelX=velx;}
    inline void SetVelY(int vely){m_DynamicObjectData.dodVelY=vely;}
    inline void SetLife(int life){m_DynamicObjectData.dodLife=life;}
    inline void SetType(DYNAMICOBJTYPE type){m_DynamicObjectData.dodType=type;}
    inline int GetVelX(){return m_DynamicObjectData.dodVelX;}
    inline int GetVelY(){return m_DynamicObjectData.dodVelY;}
    inline int GetLife(){return m_DynamicObjectData.dodLife;}
    inline DYNAMICOBJTYPE GetType(){return m_DynamicObjectData.dodType;}
    inline static int GetClassID(){return ClassID;}
    bool LoadFromFile(const char *filename);
    bool SaveToFile(const char *filename);
    bool LoadToolFromFile(const char *filename);
    void MoveUp(int count);
    void MoveDown(int count);
    void MoveLeft(int count);
```

```
    void MoveRight(int count);
    bool MoveUp(){m_PosY-=m_DynamicObjectData.dodVelY;}
    bool MoveDown(){m_PosY+=m_DynamicObjectData.dodVelY;}
    bool MoveLeft(){m_PosX-=m_DynamicObjectData.dodVelX;}
    bool MoveRight(){m_PosX+=m_DynamicObjectData.dodVelX;}

    // Main Child
    bool InitialChildObject();

    // Object Script
    // ========== Script Function ==========
    void SetScriptNumbers(int num);
    boo! LoadScript(int num,TMemoryStream *stream);
    //void InitialData(void){} Not used in Game Runtime ,Use only in Tool
};
class GChild
{
  private:
    const static int ClassID;
    GObject *m_MainChild; // point to main Child that is GDynamicObject
    bool b_wasDup;
    bool *ChildRunning;

  protected:
    //void MakeChild(GObject *mainChild,int id);
  public:
    GObject **Child;
    bool Active;

  public:
    GChild(void); // Not used
    GChild(GObject *mainChild);
    GChild(GDynamicObject *mainChild);
```

```
    GChild(GStaticObject *mainChild);
    GChild(GPlayer *mainChild);
    GChild(GObject *mainChild,int id);// For Make Child
    ~GChild();
    inline static int GetClassID(){return ClassID;}
    void Show(int childID);
    void SetActive(int childID);
    inline void SetMainActive(bool b){Active=b;}
    inline bool isRunning(int id){return ChildRunning[id];}
    bool Run(int childID,bool active,bool visible);
    bool Stop(int childID);
    void Debug(AnsiString filename);
};

class GPlayer : public GDynamicObject
{
  protected:
    const static int ClassID;
  public:
    GPlayer(void);
    GPlayer(GPlayer *mainChild);
    GPlayer(GGOS *ggos,GDXScreen *screen);
    GPlayer(GTile *tile);
    inline static int GetClassID(){return ClassID;}

    // Main Child
    bool InitialChildObject();
    /*void MoveUp(void);
    void MoveDown(void);
    void MoveLeft(void);
    void MoveRight(void);
    */
};
```

```
class GStaticObject : public GObject
{
   protected:
      const static int ClassID;
   public:
      GStaticObject(void);
      GStaticObject(GStaticObject *mainChild);
      GStaticObject(GGOS *ggos,GDXScreen *screen);
      GStaticObject(GTile *tile);
      ~GStaticObject(){}
      inline static int GetClassID(){return ClassID;}
      // Main Child
      bool InitialChildObject();
      void InitialData(void);
};

class GEventGenerator
{
   public:
      EVENTGENERATOR *Generator;

   protected:
      int TotalGenerator;
      GGOS *m_GGOS;
      GObject *Self;

   protected:
      GRemoteEvent *MakeRemoteEvent(int eventID,EVENTMESSAGE
msg,MESSAGEPARAM msgParam,GObject *source,GObject *handback);
      GENERATORRESULT CheckGeneratorMsg(EVENTGENERATOR eventGenerator);
      void Debug(AnsiString filename,EVENTGENERATOR eventGenerator);
      bool NotifyRemoteEvent(GRemoteEvent *remoteEvent,EVENTGENERATOR
```

```
&eventGenerator);
    // ClassID and ObjectID will be convert to Real Object by below functions
    // But the return Object was cast with (GObject *).
    GObject *ConvertIDToObject(int classID,int objID);
    GObject *GetRealObject(GObject **objList,int objCount,int objID);

  public:
    GEventGenerator();
    ~GEventGenerator();
    bool SaveToStream(TMemoryStream *stream);
    void InitialGenerator(GGOS *ggos,GObject *self,int num);
    bool LoadFromStream(TMemoryStream *stream);
    inline int GetTotalGenerator(){return TotalGenerator;}
    void Check();
    void OnTimer(DWORD time);
    bool NotifyRemoteEvent(int eventID); // Use only for
DoEvent_GF_MAKEEVENT_FP_ITSELF in GEvent.!

  protected:
    //=============================================
    // Below is specific defined function for each Trigger and TriggerParam
    // Name convention Rule:
    // Function is named
    // GENERATORRESULT DoGenerator+Trigger+TriggerParam(GObject
*self,EVENTGENERATOR eventGenerator);
    //=============================================

    // Function       : DoGenerator_GT_POSX_TP_NONE
    // Trigger        : On Position X
    // TriggerParam1  : X Position
    // Description    : Make Event if self object have X Position same as TriggerParam1
    GENERATORRESULT DoGenerator_GT_POSX_TP_NONE(EVENTGENERATOR
eventGenerator);
```

```
    // Function        : DoGenerator_GT_POSY_TP_NONE
    // Trigger         : On Position Y
    // TriggerParam1   : Y Position
    // Description     : Make Event if self object have Y Position same as TriggerParam1
    GENERATORRESULT DoGenerator_GT_POSY_TP_NONE(EVENTGENERATOR
eventGenerator);

    // Function        : DoGenerator_GT_ACTIVE_TP_NONE
    // Trigger         : On Active
    // TriggerParam1   : Active
    // Description     : Make Event if self object has Active same as TriggerParam1
    GENERATORRESULT DoGenerator_GT_ACTIVE_TP_NONE(EVENTGENERATOR
eventGenerator);

    // Function        : DoGenerator_GT_SCREENPOSY_TP_NONE
    // Trigger         : On Screen Position Y
    // TriggerParam1   : Screen Y Position
    // Description     : Make Event if self object have screen Y Position same as TriggerParam1
    GENERATORRESULT DoGenerator_GT_SCREENPOSY_TP_CURRENTMAP
(EVENTGENERATOR eventGenerator);

    // Function        : DoGenerator_GT_ONTIMER_TP_NONE
    // Trigger         : On Timer
    // Description     : Make Event if timer was finish
    GENERATORRESULT DoGenerator_GT_ONTIMER_TP_NONE(EVENTGENERATOR
eventGenerator);
};

class GEvent
{
  public:
    EVENTLISTENER *Event;
```

```
protected:
    int TotalEvent;
    GObject *Self;
    GGOS *m_GGOS;

public:
    GEvent();
    ~GEvent();
    bool SaveToStream(TMemoryStream *stream);
    void InitialEvent(GGOS *ggos,GObject *self,int num);
    bool LoadFromStream(TMemoryStream *stream);
    void Notify(GRemoteEvent *remoteEvent);
    inline int GetTotalEvent(){return TotalEvent;}
    EVENTRESULT CheckRemoteEvent(GRemoteEvent *remoteEvent,EVENTLISTENER
eventListener);
    EVENTRESULT DoEvent(GRemoteEvent *remoteEvent,EVENTLISTENER
eventListener);

protected:
    // ========================
    // Below is Specific defined function for each Msg and MsgParam
    // Name convention rule:
    // Function is named
    // EVENTRESULT DoEvent+Function+FunctionParam(GObject *self,GRemoteEvent
remoteEvent,EVENTLISTENER eventListener);
    // ========================

    // Function       : GF_VISIBLE_FP_ITSELF
    // FunctionParam1  : 0=invisible , 1 = visible
    // Description    : For change Visible property of itself Object
    EVENTRESULT DoEventGF_VISIBLE_FP_ITSELF(GRemoteEvent
*remoteEvent,EVENTLISTENER eventListener);
```

```
    // Function       : GF_CHANGEMAP_FP_PLAYER
    // self           : Player Object
    // FunctionParam1  : Map number (0..GGOS->m_MapCount)
    // FunctionParam2  : Map PosX (0..Mapwidth) for Player
    // FunctionParam3  : Map PosY (0..MapHeight) for Player
    // FunctionParam4  : Map PosZ (0..MapPosZ(from GObject)) for Player1
    // FunctionParam5  : Start Map PosX(0..Mapwidth)
    // FunctionParam6  : Start Map PosY(0..MapHeight)
    // Description     : Change Current Map to Map number
    //                 : and also effect to move player to
    //                 : that map
    EVENTRESULT DoEventGF_CHANGEMAP_FP_PLAYER(GRemoteEvent
*remoteEvent,EVENTLISTENER eventListener);

    // Function       : GF_ACTIVE_FP_ITSELF
    // FunctionParam1  : 1=active,0=inactive
    // Description     : Change self Active Property
    EVENTRESULT DoEventGF_ACTIVE_FP_ITSELF(GRemoteEvent
*remoteEvent,EVENTLISTENER eventListener);

    // Function       : GF_DESTROY_FP_ITSELF
    // FunctionParam1  : 0 = Destroy Object or Main Child (All Child Destroyed)
    //                 : n>0 = Destroy Child number n
    // Description     : Destroy Itself (inactive+invisible)
    EVENTRESULT DoEventGF_DESTROY_FP_ITSELF(GRemoteEvent
*remoteEvent,EVENTLISTENER eventListener);

    // Function       : GF_RUNCHILD_FP_ITSELF
    // FunctionParam1  : Child id if = -1 ,Run all Child
    // Description     : Run own Child if it has(child itself)
    EVENTRESULT DoEventGF_RUNCHILD_FP_ITSELF(GRemoteEvent
*remoteEvent,EVENTLISTENER eventListener);
```

```
// Function       : GF_RUNSCRIPT_FP_ITSELF
// FunctionParam1  : Script id
// Description    : Run own Script if it has
EVENTRESULT DoEventGF_RUNSCRIPT_FP_ITSELF(GRemoteEvent
*remoteEvent,EVENTLISTENER eventListener);

// Function       : GF_STOPSCRIPT_FP_ITSELF
// FunctionParam1  : Script id
// Description    : Stop own Script if it has
EVENTRESULT DoEventGF_STOPSCRIPT_FP_ITSELF(GRemoteEvent
*remoteEvent,EVENTLISTENER eventListener);

// Function       : GF_MAKEEVENT_FP_ITSELF
// FunctionParam1  : self 's EventGenerator ID
// Description    : Generator Event from FunctionParam1 by Make Event
//                : defined by EventGenerator on self object
//                : This event for make event object without
//                : occur by trigger that defined.
EVENTRESULT DoEventGF_MAKEEVENT_FP_ITSELF(GRemoteEvent
*remoteEvent,EVENTLISTENER eventListener);

// Function       : GF_STARTTIMER_FP_ITSELF
// FunctionParam1  : Time (if = 0 then use timer itself)
// Description    : Start Object Timer by specified time
//                or if time =0 then start by m_ObjectDataHeaderodTimer
EVENTRESULT DoEventGF_STARTTIMER_FP_ITSELF(GRemoteEvent
*remoteEvent,EVENTLISTENER eventListener);

// Function       : GF_STOPTIMER_FP_ITSELF
// Description    : Stop Object Timer
EVENTRESULT DoEventGF_STOPTIMER_FP_ITSELF(GRemoteEvent
*remoteEvent,EVENTLISTENER eventListener);
```

```
};

class GScript
{
  public:
    SCRIPT *Script;
    bool Active;

  protected:
    GGOS *m_GGOS;
    int TotalScript;
    GObject *gObject;
    GDynamicObject *gdynaObject;
    bool *ScriptRunning;

  public:
    GScript();
    ~GScript();
    bool SaveToStream(TMemoryStream *stream);
    bool LoadFromStream(TMemoryStream *stream);
    inline int GetTotalScript(){return TotalScript;}
    inline void SetMainActive(bool b){Active=b;}
    void InitialScript(GGOS *ggos,GObject *self,int num);
    void InitialScript(GGOS *ggos,GDynamicObject *dynaself,int num);
    void Debug(AnsiString filename,SCRIPT script);
    void Run();
    bool Run(int id);
    bool Stop(int id);
    SCRIPTRESULT CheckScript(SCRIPT script);

  protected:
    // ================================================================
    // Below is specified functions for each object script
```

```
    // Name convention rule:
    // Function name is
    // SCRIPTRESULT DOScript+ScriptType(SCRIPT script)
    // ===========================================================================

    // Function       : DoScriptST_CHILDMOVEDOWN
    // sParam1        : Child id
    // Description    : Move Object ,as a child, by the dodVelY,dodVelX value
    SCRIPTRESULT DoScriptST_CHILDMOVEDOWN(SCRIPT script);
    SCRIPTRESULT DoScriptST_CHILDMOVELEFT(SCRIPT script);
    SCRIPTRESULT DoScriptST_CHILDMOVERIGHT(SCRIPT script);

    // Function       : DoScriptST_MOVEDOWN
    // Description    : Move Object by the dodVelY ,dodVelX value
    SCRIPTRESULT DoScriptST_MOVEDOWN(SCRIPT script);
    SCRIPTRESULT DoScriptST_MOVELEFT(SCRIPT script);
    SCRIPTRESULT DoScriptST_MOVERIGHT(SCRIPT script);
};
//---------------------------------------------------------------
#endif
```

นอกจากคลาสเหล่านี้แล้วยังมีคลาสอื่นๆอีกที่ใช้ในการทำงานของเกมซึ่งไม่ได้นำมาแสดงในที่นี้