การเขียนเกมส์บน DOS ด้วยภาษาซี (โดยใช้ DJGPP)

Game Programming with C programming language (by DJGPP)

โดย

นายปิติ เชิญฉนอมวงศ์

นายพรธเนศร์ แซ่โค้ว

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

สาขาวิศวกรรมคอมพิวเตอร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2540

การเขียนเกมส์บน DOS ด้วยภาษาซี (โดยใช้ DJGPP)

Game Programming with C programming language (by DJGPP)

โดย

นายปิติ เชิญฉนอมวงศ์ 37014255

นายพรธเนศร์ แซ่โค้ว 37014278

อาจารย์ที่ปรึกษา

ดร. เอื้อน ปิ่นเงิน

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

สาขาวิศวกรรมคอมพิวเตอร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2540

ปริญญานิพนธ์ปีการศึกษา 2540

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การเขียนเกมส์บน DOS ด้วยภาษาซี (โดยใช้ DJGPP)

ผู้จัดทำ

1. นายปิติ เชิญถนอมวงศ์ 37014255

2. นายพรธเนศร์ แซ่โค้ว 37014278

..... อาจารย์ที่ปรึกษา

(ดร. เอื้อน ปิ่นเงิน)

การเขียนเกมส์บน DOS ด้วยภาษาซี (โดยใช้ DJGPP)

นายปิติ เชิญถนอมวงศ์
นายพรธเนศร์ แซ่โค้ว

อาจารย์ที่ปรึกษา
ดร. เอื้อน ปิ่นเงิน
ปีการศึกษา 2540

บทคัดย่อ

ปริญญานิพนธ์นี้เป็นการนำความรู้และเทคนิคต่าง ๆ ทางด้านการเขียนโปรแกรม มาประยุกต์ใช้โดยจะคำนึงถึงหลัก Object-oriented ไปด้วย โดยแต่ละออปเจกต์ในโครงงานจะแยกออกจากกันอย่างชัดเจน ซึ่งจะทำให้ง่ายต่อการเปลี่ยนแปลงแก้ไข หรือเพิ่มออปเจกต์ต่าง ๆ อีกทั้งยังได้ใช้ DJGPP เข้ามาช่วยในการสร้าง ทำให้เกมส์ที่เขียนขึ้นสามารถทำงานบนหน่วยความจำส่วนขยาย (extended memory) ใน protected mode ได้ ซึ่งจะช่วยให้การทำงานต่าง ๆ มีความเร็วมากกว่าการทำงานบน real mode ซึ่งจะถูกจำกัดหน่วยความจำไว้เพียง 640 กิโลไบต์ เท่านั้น

นอกจากนี้แล้วยังได้นำความรู้ในการบริหารการใช้ ทรัพยากรต่าง ๆ เช่น หน่วยความจำ เป็นต้นมาประยุกต์ใช้ในโครงงานเพื่อให้มีประสิทธิภาพมากที่สุดอีกด้วย

Game Programming with C programming language (by DJGPP)

Piti Cherntanomwong

Porntaned Saekoe

Project Advisor

Dr. Ouen Pinngern

1997

Abstract

Knowledge and game programming techniques are applied in this thesis. In same design, we emphasize on Object-oriented for which objects are separated individually. Objects can be changed, modified, deleted or added easily. Furthermore, we used DJGPP in this thesis. DJGPP makes our game can run under extended memory under protected mode, which is faster than in real mode, which is fixed only 640 Kbytes.

Moreover we use resource management methods, such as memory management. So that we can utilize resource at the highest efficiently.

กิตติกรรมประกาศ

ขอขอบพระคุณในความกรุณา และคำแนะนำของท่านอาจารย์ต่อไปนี้

1. ดร. เอื้อน ปิ่นเงิน
2. อ.วิบูลย์ พร้อมพานิชย์
3. อ. อภิเนตร อูนากูล

และอาจารย์ภาควิศวกรรมคอมพิวเตอร์ทุกท่าน รวมทั้งคำแนะนำ และกำลังใจจากเพื่อน ๆ น้อง ๆ ทุกคน ที่ทำให้โครงงานชิ้นนี้สำเร็จลุล่วงไปได้ด้วยดี

ขอขอบพระคุณเป็นอย่างสูง

นายปีติ เชิญฉนอมวงศ์ 37014255

นายพรธเนศร์ แซ่โค้ว 37014278

ผู้จัดทำ

สารบัญ

หน้า

สารบัญตาราง

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มา

เนื่องจากปัจจุบันมีการนำซอฟต์แวร์ มาประยุกต์ใช้ในงานต่าง ๆ มากมาย ดังนั้นเทคนิคการเขียนโปรแกรมให้ตรงกับความต้องการของผู้ใช้จึงเป็นสิ่งจำเป็นและมีการพัฒนาไปอย่างรวดเร็ว รวมถึงเทคนิคของการนำปัญญาประดิษฐ์มาใช้ในโปรแกรมด้วย

ปัญญาประดิษฐ์ (Artificial Intelligent) เป็นการทำให้คอมพิวเตอร์สามารถมีความคิดเป็นของตนเองได้ โดยผู้เขียนได้พยายามกำหนดหนทางแก้ไขให้คอมพิวเตอร์เมื่อพบกับสถานะการณ์ต่าง ๆ ทำให้คอมพิวเตอร์ไม่จำเป็นต้องมีคนเป็นผู้ตัดสินใจให้อยู่ตลอดเวลา สำหรับโปรแกรมที่ได้มีการใส่ปัญญาประดิษฐ์ เช่น โปรแกรมเล่นหมากรุก ซึ่งปัจจุบันบริษัท IBMได้พัฒนาจนมีความสามารถคิดล่วงหน้าไปได้หลายล้านตา โดยใช้เวลาเพียงเสี้ยววินาที ทำให้ความสามารถของมันขึ้นมาทัดเทียมกับแชมป์โลกหมากรุกคนปัจจุบัน เลยทีเดียว

เกมส์ในโครงงานนี้ก็มีการใส่ปัญญาประดิษฐ์ให้กับศัตรูทุกตัวให้สามารถตัดสินใจกระทำอิริยาบทต่าง ๆ ได้เอง โดยขึ้นอยู่กับสถานะการณ์ขณะนั้น เช่น ยิงกระสุนเมื่อพระเอกอยู่ในรัศมีการโจมตี หรือ วิ่งหนีหากมันได้รับบาดเจ็บ ซึ่งจะอธิบายอยู่ในภาคผนวก ก. ดังนั้นมันจึงดูเหมือนว่ามีสมองเป็นของตัวเอง

ในขณะนี้การเขียนเกมส์คอมพิวเตอร์บน DOS ได้มีการใช้เทคนิคพิเศษมาช่วยคือ การใช้ DJGPP หรือ DOS/4GW เข้ามาช่วย ซึ่งในโครงงานนี้จะใช้ DJGPP ซึ่งเป็นตัวคอมไพเลอร์ภาษาซี และซีพลัสพลัส โดย DJGPP จะช่วยให้โปรแกรมนั้น ๆ สามารถดึงเอาหน่วยความจำส่วนขยาย (extended memory) ใน protected mode มาใช้ได้ ซึ่งปกติระบบ DOS จะอนุญาตให้ใช้ได้ใน real mode เท่านั้น ทำให้โปรแกรมสามารถใช้หน่วยความจำได้จำกัดแค่ 640 กิโลไบต์ เท่านั้น

เกมส์เป็นโปรแกรมที่มีการใช้ภาพเข้ามาสื่อสารแทนตัวอักษรช่วยให้ผู้เล่นเข้าใจได้ง่ายขึ้น และได้รับความสนุกสนานเพลิดเพลิน การเขียนเกมจึงเป็นหนทางหนึ่งในการพัฒนาความสามารถในการเขียนโปรแกรมให้ดียิ่งขึ้น นอกจากนี้ในปัจจุบันเกมส์คอมพิวเตอร์ได้เข้ามามีบทบาทในการใช้คอมพิวเตอร์เป็นอย่างมาก และการที่จะเขียนเกมส์ให้ตรงกับความต้องการของผู้เล่นมากที่สุดนั้นเป็นสิ่งยาก และท้าทายมากในปัจจุบัน

1.2 วัตถุประสงค์ของงานวิจัย

1.2.1 เพื่อศึกษาเทคนิคและวิธีการเขียนเกมส์คอมพิวเตอร์

1.2.2 เพื่อศึกษาการนำเอาปัญญาประดิษฐ์ (Artificial Intelligence) เข้ามาใช้ในโปรแกรม

1.2.3 เพื่อพัฒนาการเขียนโปรแกรม โดยอ้างอิงหลัก Object-oriented ซึ่งจะทำให้การเปลี่ยนแปลงแก้ไข หรือเพิ่ม ออปเจกต์ต่าง ๆ ได้ง่ายขึ้น

1.3 ขอบเขตของงานวิจัย

เนื่องจากโครงงานนี้ต้องการศึกษา การเขียนเกมส์ และนำปัญญาประดิษฐ์เข้ามาใช้ในโปรแกรมดังที่อยู่ในหัวข้อวัตถุประสงค์ ประกอบกับผู้ทำโครงงานนี้มิได้มีพื้นฐานทางด้านการเขียนเกมส์มาก่อน ถึงแม้ว่าจะมีความสนใจทางด้านนี้อยู่ก็ตาม อีกทั้งยังมีข้อจำกัดเรื่องเวลา ดังนั้นโครงงานนี้จึงอาจจะเป็นเพียงแบบจำลองที่ยังไม่เสร็จสมบูรณ์เหมือนเกมส์ทั่วไปที่มีอยู่ทั่วไป โดยขอบเขตที่ผู้ทำโครงงานได้กำหนดไว้คือ

1.3.1 สร้างตัวละครไม่น้อยกว่า 5 ตัวละคร ซึ่งแต่ละตัวละครจะมีเอกลักษณ์เฉพาะตัว และมีการใส่ปัญญาประดิษฐ์ให้แก่แต่ละตัวละครแตกต่างกันไปตามความเหมาะสม

1.3.2 สร้างเกมส์ให้มีฉากประมาณ 2 ฉากเป็นอย่างน้อย

1.3.3 พยายามพัฒนาเกมส์โดยยึดหลักความเหมือนจริง และความต้องการของผู้เล่นให้ได้มากที่สุด ดังนั้นเกมส์ที่สร้างจะต้องมีความยืดหยุ่นสูง และตอบสนองความต้องการของผู้เล่นได้มากที่สุด

ถึงแม้ปริญญานิพนธ์ชิ้นนี้จะจบลงไปแล้วแต่ผู้ทำโครงงานยังมีความต้องการที่จะพัฒนาตัวโครงงานให้มีความสามารถเพิ่มสูงขึ้นเท่าที่จะเป็นไปได้

1.4 วิธีดำเนินงาน

เนื่องจากผู้ทำโครงงานยังไม่มีพื้นฐานทางด้านการเขียนเกมส์มาก่อน ซึ่งการเขียนเกมส์ค่อนข้างจะเป็นซอฟท์แวร์ที่แตกต่างจะซอฟท์แวร์อื่นอยู่พอสมควร ดังนั้นขั้นตอนการดำเนินการจึงแบ่งเป็นส่วนหลัก ๆ ได้ดังนี้

1.4.1 ศึกษาขั้นตอนต่าง ๆ ที่ใช้ในการเขียนเกมส์

1.4.2 ศึกษาวิธีการนำหน่วยความจำส่วนขยาย (extended memory) มาใช้งาน

1.4.3 ออกแบบฟังก์ชันพื้นฐานในการเขียนเกมส์

1.4.4 ออกแบบตัวละคร

1.4.5 ใส่ปัญญาประดิษฐ์ให้กับตัวละครต่าง ๆ

1.4.6 ออกแบบฉาก

1.4.7 ปรับปรุงและพัฒนาฟังก์ชันต่าง ๆ โดยอ้างอิงความเหมือนจริง และความต้องการของผู้เล่นเป็นหลัก

ซึ่งในบทต่าง ๆ ถัดจากบทนี้จะเริ่มอธิบายขั้นตอนต่าง ๆ โดย

บทที่ 2 **จะแนะนำการนำหน่วยความจำส่วนขยาย (extended memory) มาใช้งาน**

บทที่ 3-8 **จะแนะนำถึงการหน้าที่การทำงานในส่วนต่าง ๆ ของระบบ**

บทที่ 2

DJGPP และ Allegro

2.1 DJGPP

DJGPP คือ C/C++ คอมไพเลอร์ (compiler) ตัวหนึ่งที่สามารถใช้กับ 32 บิต protect-mode บน Intel 32 บิต โพรเซสเซอร์ ซึ่งทำงานบนระบบ MS-DOS และระบบอื่น ๆ ที่สามารถเข้ากันได้ (compatible) ซึ่งทำการพัฒนาโดย นาย DJ Delorie และ เพื่อน ซึ่ง DJGPP จะทำให้เราสามารถนำหน่วยความจำที่เป็นส่วนขยาย (Extended memory) มาใช้ได้

DJGPP รุ่น 2 เป็นโปรแกรมที่สามารถทำงานด้วยตัวเองได้โดยไม่ต้องใช้โปรแกรมอื่นเข้ามาช่วย ยกเว้น DPMI server (Dos Protected Mode Interface) ซึ่ง DJGPP จะสามารถทำงานได้บน 32 บิต DPMI , 4 กิกะไบต์ flat address และยังสามารถใช้หน่วยความจำเสมือน (virtual memory) ได้ถึง 256 เมกกะไบต์

2.1.1 ระบบที่สนับสนุนการทำงานของ DJGPP

DJGPP ต้องการโพรเซสเซอร์ ตั้งแต่รุ่น 386sx ขึ้นไป และต้องมีเนื้อที่ในหน่วยความจำ (disk space) ไม่น้อยกว่า 15-35 เมกกะไบต์, มีหน่วยความจำชั่วคราว (ram) ไม่น้อยกว่า 64 กิโลไบต์ เพื่อใช้ CWSDPMI (DPMI Server) แต่ถ้าต้องการความเร็วก็ควรจะใช้อย่างน้อย 2.5-4 เมกกะไบต์ สำหรับการแปล (compile) โปรแกรมขนาดใหญ่ ส่วนโพรเซสเซอร์สนับสนุน (co-processor) ที่ใช้ในการประมวลผลทางคณิตศาสตร์ ถ้าไม่มีจะต้องติดตั้ง emulator เพื่อใช้ประมวลผลเลขทศนิยมยกกำลัง

ถ้าระบบที่ใช้อยู่ยังไม่มี DPMI Server ก็จะต้องทำการติดตั้ง CWSDPMI เพิ่มด้วยแต่ถ้าระบบมีอยู่แล้วก็ไม่จำเป็นต้องติดตั้งใหม่ ระบบที่มี DPMI Server อยู่แล้ว เช่น วินโดวส์,โอเอส 2, วินโดวส์ 95,วินโดวส์ เอ็นที

2.1.2 การติดตั้ง DJGPP

ในการสร้างโปรแกรมภาษา ซี จะต้องมีไฟล์เหล่านี้

djdev201.zip

gcc2721b.zip

bnu27b.zip

ส่วน ภาษา ซีพลัสพลัส จะต้องมีเพิ่มคือ

gpp2721b.zip

lgp271b.zip

สำหรับคู่มือการใช้ มีเขียนอยู่ใน txi390b.zip และถ้าหากเรายังไม่มี DPMI Server จะต้องมี csdpmi3b.zip ด้วย โดยจะสามารถหาไฟล์เหล่านี้ได้ที่ เวบไซต์

http://www.delorie.com

2.1.3 ขั้นตอนการติดตั้ง DJGPP

2.1.3.1 สร้างไดเรกทอรี DJGPP (C:\djgpp\) ถ้าในเครื่องมี DJGPP รุ่นที่ 1 อยู่แล้วให้ลบทุกไฟล์ที่มีอยู่ในไดเรกทอรี ยกเว้น go32.exe

2.1.3.2 กระจาย ไฟล์ต่าง ๆ ออก ลงในไดเรกทอรี DJGPP

2.1.3.3 เมื่อเสร็จแล้วใน ตั้งค่า DJGPP environment เพื่อใช้ชี้ไปไฟล์ djgpp.env ใน DJGPP และใส่ Path ของไดเรกทอรี bin ด้วย ซึ่งทั้งหมดนี้จะต้องเพิ่มลงในไฟล์ autoexec.bat ดังนี้

```
set DJGPP=C:\DJGPP\DJGPP.ENV
set PATH=C:\DJGPP; %PATH%
```

2.1.3.4 เรียกไฟล์ go32-v.exe ดังนี้

```
go32-v2
```

ซึ่งจะแสดงให้เห็นว่าเรามีหน่วยความจำของ DPMI และ swap space ที่ DJGPP สามารถใช้ได้เท่าไร เช่น

```
DPMI memory available: 8020 Kb
DPMI swap space available: 39413 Kb
```

ซึ่งค่าจริง ๆ นั้นจะขึ้นอยู่กับจำนวนหน่วยความจำ ที่เรามีอยู่ในระบบ

2.1.4 การคอมไพล์ (compile) DJGPP

การคอมไพล์ ใช้คำสั่ง GCC ในดอส (DOS) หากต้องการคอมไพล์ ไฟล์ซี เพียงไฟล์เดียวใช้คำสั่งดังนี้

```
gcc myfile.c -o myfile.exe -lm
```

ซึ่ง lm คือ การติดต่อกับ library lib\libm.a ซึ่งใช้ในการคำนวณทางคณิตศาสตร์

หากต้องการคอมไพล์ ไฟล์ซี หรือ ซีพลัสพลัส ให้เป็น ออปเจกต์ ไฟล์ (object file) ใช้คำสั่งดังนี้

```
gcc -c -wall myfile.c          (สำหรับไฟล์ซี)
gcc -c -wall myfile.cc         (สำหรับไฟล์ซีพลัสพลัส)
```

ซึ่ง -wall เป็นการสั่งให้มีข้อความเตือน

หากต้องการติดต่อกันหลาย ๆ ไฟล์ ออปเจกต์ ไปเป็นไฟล์ที่ใช้งานได้ (executed file) ใช้คำสั่งดังนี้

```
gcc -o myprog.exe mymain.o mysub1.o mysub2.o
```

โดยจะสร้างไฟล์ myprog.exe ซึ่งจะสามารถทำงานบนระบบดอส (DOS) ได้

หากต้องการติดต่อไฟล์ซีพลัสพลัส ให้ใช้ gxx แทน gcc ดังนี้

```
gxx -o myprog.exe mymain.o mysub1.o mysub2.o
```

โดยเราสามารถรวมทั้ง 2 ขั้นตอน ได้เลย ดังนี้

```
gcc -wall -o myprog.exe mymain.c mysub1.c mysub2.c
```

2.1.5 การใช้ส่วนประกอบต่าง ๆ ของ DJGPP

DJGPP ไม่ได้มีส่วนประกอบต่าง ๆ ติดมาด้วย ดังนั้นเราสามารถเลือกใช้โปรแกรมใด ๆ ก็ได้ในการเขียนโปรแกรม และสามารถให้ผลลัพธ์ได้เหมือนกับ IDE ซึ่งผู้ใช้หลาย ๆ คนได้ใน GNU Emacs ที่สามารถคอมไพล์ DJGPP ได้ ซึ่ง Emacs เป็น Editor ตัวหนึ่งที่มีความสามารถมากจะใช้ทำเอกสารที่สามารถเรียกขึ้นมาอ่านได้โดยไม่ต้องอาศัยโปรแกรมอื่นช่วย แต่ Editor ทุกตัวจะมีข้อเสียคือเมื่อนำมาใช้กับ DJGPP แล้วจะไม่สามารถทดสอบหาข้อผิดพลาดของโปรแกรมได้ (debug)

2.1.6 การทดสอบหาข้อผิดพลาดของโปรแกรม (debug)

เพื่อจะหาข้อผิดพลาดของโปรแกรม เราจะต้องคอมไพล์ ไฟล์ซี ด้วยคำสั่งดังนี้

```
gcc -c -wall -g mymain.c
gcc -c -wall -g mysub1.c
gcc -c -wall -g mysub2.c
```

และติดต่อด้วย -g ดังนี้

```
gcc -g -o myprog.exe mymain.o mysub1.o mysub2.o
```

จากนั้นก็สั่งให้โปรแกรมทำงานด้วยคำสั่งดังนี้

```
fsdb myprog.exe
```

หรือ `gdb myprog.exe`

หรือ `edebug32 myprog.exe`

สำหรับ fsdb มีคำสั่งช่วยเหลือ ด้วยการกด F1 ,ใน gdb มีอยู่ในไฟล์ info.exe และ edubug32 กด h

2.2 Allegro

Allegro เป็นฟังก์ชัน (function) ที่เก็บอยู่ในรูป Library สำหรับใช้เขียนเกมคอมพิวเตอร์โดยเฉพาะซึ่งเขียนให้สามารถใช้กับ DJGPP โดยให้มีลักษณะคล้ายภาษาซี และ แอสเซมบลี

2.2.1 ลักษณะพิเศษของ Allegro

Allgro ถูกสร้างมาให้ใช้กับ VGA mode 13h,mode X และ 256 color SVGA modes ใน Allegro จะประกอบไปด้วยหลาย ๆ ฟังก์ชัน เช่น ฟังก์ชัน วาดรูป (Drawing) ก็สามารถทำได้หลายอย่าง เช่น กำหนดพิกัดของจุด,หาตำแหน่งพิกัดของจุด,วาดรูปเรขาคณิต,แรเงา และรวมถึงการนำข้อความ (text) มาแสดงในรูปของโพลีกอน (polygon) ,วงกลม,เล่นแสงเงา ต่าง ๆ และยังสามารถทำงานเกี่ยวกับ sprite,รูปบิตแมบ bitmap ,การแปลงรูป (transparency) และยังสามารถทำข้อความด้วยลักษณะอักษรที่เรา

ต้องการได้ด้วย นอกจากนี้ยังสนับสนุนการแสดงออกทางหน้าจอ (screen) ได้โดยตรง หรือเขียนลงบนหน่วยความจำบิตแมบ (bitmap memory) ได้เลย และยังสนับสนุนการทำภาพเคลื่อนไหว (animation) อีกด้วย

การเพิ่มเสียงลงในเกมก็สามารถเรียกใช้ฟังก์ชัน ใน Allegro ได้เช่นกัน โดยสามารถใส่ดนตรี และสามารถแสดงผลพร้อมกันได้ถึง 8 เสียง การควบคุมเสียงและสิ่งต่าง ๆ ได้โดยการเปิด/ปิด (note on, not off) สำหรับ Allegro จะสามารถใช้ได้กับ การ์ดเสียงของ Adlib, SB, SB pro,SB16 และ MPU-401

การควบคุม สามารถใช้อุปกรณ์ต่าง ๆ ได้เช่น เมาส์ (mouse) , คันบังคับ (joystick), และคีย์บอร์ด (keyboard) และใช้ interrupt timer ที่มีความเร็วสูง มีโปรแกรมย่อย (routines) ที่ใช้สำหรับอ่านและเขียนไฟล์ที่ถูกอัด (compressed files) มีฟังก์ชัน สนับสนุนการคำนวณทางคณิตศาสตร์ . lookup table trig และ เวกเตอร์ 3 มิติ

2.2.2 ส่วนสนับสนุน Allegro

Allegro ต้องการฮาร์ดแวร์ ที่สนับสนุนการทำงานอย่างน้อยดังนี้ โพรเซสเซอร์ รุ่น 386 ขึ้นไป ,การ์ดแสดงภาพ ระดับ VGA แต่ถ้าเป็นไปได้ควรจะใช้ โพรเซสเซอร์ รุ่น 486 ขึ้นไป ถ้าต้องการใช้การ์ดแสดงภาพระดับ SVGA ก็ได้เช่นกัน

สำหรับเรื่องเสียง Allegro จะสนับสนุนการทำงานกับ SB(mono),SB Pro(stereo) และ SB 16 ซึ่ง Allegro จะมีไดรเวอร์ ของมิดี้ (midi driver) สำหรับ OPL2 FM synth ที่มีอยู่ใน Adlib และ SB

2.2.3 การติดตั้ง Allegro

2.2.3.1 สร้างไดเรกทอรี Allegro เช่น C:\Allegro และระเบิดไฟล์ต่าง ๆ (unzip) ลงในdirectory

2.2.3.2 เรียก make.exe

2.2.3.3 เมื่อต้องการเรียกใช้ Allegro ในโปรแกรมให้เขียน

```
#include <allegro.h>
```

ไว้ต้นโปรแกรม

2.2.3.4 เมื่อคอมไพล์ไฟล์ให้เพิ่ม -lalleg ลงท้ายคำสั่งด้วย เช่น

```
gcc foo.c -o foo.exe -lalleg
```

2.2.3.5 ถ้าใช้ Rhide ให้ไปที่เมนูของ Option/libraries และพิมพ์ alleg ลงในช่องแรก

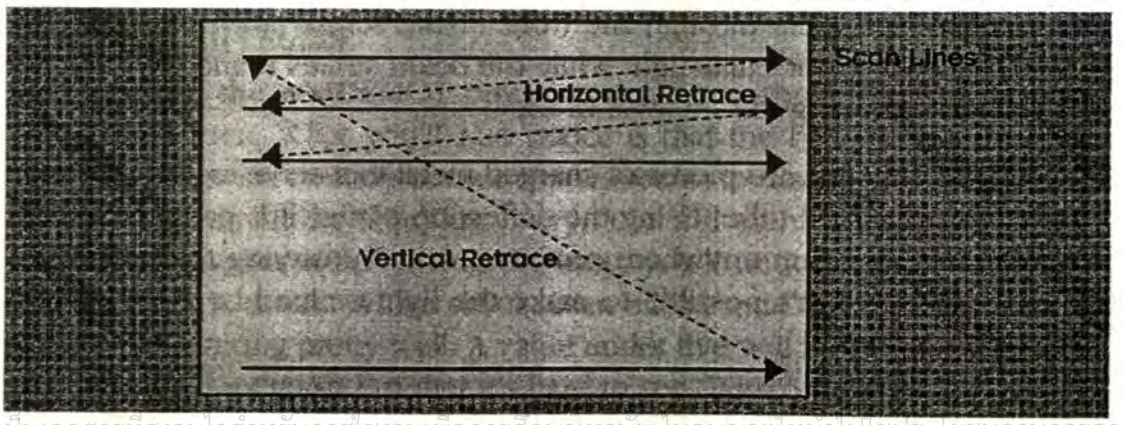
บทที่ 3
ความรู้เบื้องต้นเกี่ยวกับโหมดการแสดงภาพ และการทำงานกับไฟล์ pcx

3.1 การทำงานของ ส่วนแสดงภาพ (video display)

ส่วนแสดงภาพ (video display, video monitor หรือ cathode ray tube) ประกอบด้วย ปืนอิเลคตรอน (electron gun) ,แผ่นหักเห (deflection plates) และ แผ่นแก้วฉาบฟอสฟอรัส (phosphor coated glass screen) บนหน้าจอ เราจะเห็นเป็นแผ่นแก้ว แต่จริง ๆ แล้วแผ่นแก้วนี้เป็นส่วนปลายของหลอดแก้วรูปทรงปีระมิด ปืนอิเลคตรอนจะยิงลำแสงอิเลคตรอน โดยฟอสฟอรัสบนแผ่นแก้ว เปล่งเป็นแสงออกมาให้เราเห็นซึ่งฟอสฟอรัสจะยังคงเปล่งแสงต่อไป ขณะที่ลำแสงยังโดนมันอยู่ และยังคงอยู่อีกสักพักหลังจากที่ลำแสงไม่โดนมันแล้ว

แผ่นหักเหเป็นอุปกรณ์ที่บังคับตำแหน่งที่ลำแสงอิเลคตรอนจะไปโดยบนแผ่นแก้ว และใช้ความเข้มของลำแสงอิเลคตรอนในการปรับความมืดหรือสว่างของแสงที่เปล่งออกมาจากฟอสฟอรัส บนแผ่นแก้วจะมีฟอสฟอรัสฉาบอยู่ 3 ประเภท คือ ประเภทที่เปล่งแสงสีแดง,สีเขียว และ สีน้ำเงิน โดยฉาบเป็นสามเหลี่ยมที่ประกอบด้วยจุด 3 จุด ต่อกันไปเรื่อย ๆ จนคลุมทั้งแผ่นแก้วบนส่วนแสดงภาพ (video display) จะมีปืนอิเลคตรอนอยู่ 3 กระบอก แต่ละกระบอกจะถูกใช้เพื่อยิงไปยังจุด ๆ หนึ่งบนสามเหลี่ยมด้วยการปรับความเข้มของแต่ละกระบอก ทำให้เกิดสีที่แตกต่างกัน

การแสดงภาพระดับ VGA จะตรวจดูค่าของแต่ละสีในหน่วยความจำ และนำเสนอด้วยความเข้มของสี 3 สี คือ แดง,น้ำเงิน และ เขียว รวมกันเป็นสีที่ต้องการ VGA จะเริ่มทำงานบนส่วนแสดงภาพ (video display)จากซ้ายไปขวา ซึ่งทำได้โดยการควบคุมผ่านแผ่นหักเห ลำแสงจะเคลื่อนไปเรื่อย ๆ ขณะที่การ์ดแสดงภาพ จะส่งข้อมูลของสีอย่างต่อเนื่อง เราจะเห็นบรรทัดบนหน้าจอปรากฏขึ้นมา การเคลื่อนของลำแสงมีลักษณะดังรูป



รูป 3-1 รูปแบบการแสดงภาพของส่วนแสดงภาพ

เมื่อลำแสงมาถึงปลายด้านขวาของแผ่นแก้ว มันจะปิดตัวเองไปชั่วขณะหนึ่ง และเคลื่อนไป ยังปลายด้านซ้ายของแผ่นแก้ว โดยเลื่อนลงมา 1 บรรทัด เรียกการทำงานลักษณะนี้ว่า การทับรอยเดิมในแนวแกนนอน (horizontal retrace) การหยุดยิงลำแสงของปืนอิเลคตรอนเป็นการรับประกันว่าจะไม่เกิดแสงใด ๆ ขึ้น เมื่อลำแสงเคลื่อนที่กลับไปทางด้านซ้าย การทับรอยเดิมในแนวแกนนอน (horizontal retrace) จะเกิดขึ้นในแต่ละบรรทัดจนกระทั่งมาถึงบรรทัดสุดท้ายของหน้าจอ เมื่อลำแสงมาถึงปลายด้านขวาและไม่มีบรรทัดที่อยู่ต่ำลงไปจะเกิดการหยุดยิงลำแสอีกครั้ง และเคลื่อนลำแสงไปยังจุดซ้ายบนของหน้าจอ เรียกการทำงานลักษณะนี้ว่า การทับรอยเดิมในแนวแกนตั้ง (vertical retrace) ซึ่งการทำงานทั้งหมดนี้เรียกว่า 1 หน้าการแสดงผล (video frame) ซึ่งภาพที่เราเห็นปรากฏบนหน้าจออาจเกิดจากการที่แสดงข้อมูลเดิมของหน้าจอ หรือแสดงข้อมูลใหม่ของหน้าจอ การทำงานทั้งหมดนี้จะเกิด 17 ครั้งต่อวินาที ใน mode 13h

3.2 การแสดงภาพ

ในที่นี้ได้ใช้ฟังก์ชันในการตั้งค่ารูปแบบของการแสดงผลหน้าจอ จาก library Allegro (set_gfx_mode()) โดยใช้รูปแบบการแสดง 320*200 และ 640*480

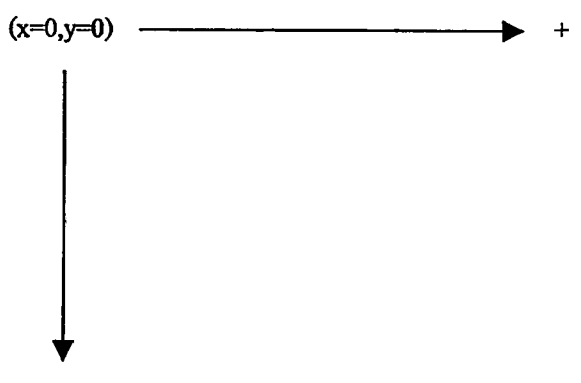
รูปแบบแบบ 320*200 หรือ mode 13h มีขนาดความกว้างของหน้าจอ 320 pixels สูง 200 pixels 256 สี สามารถเรียกใช้โดย interrupt 10h, ah = 0, al = 13h การที่จะนำข้อมูลที่พร้อมที่จะแสดงออกสู่หน้าจอนั้น สามารถทำได้โดยนำข้อมูลดังกล่าวเขียนลงไปในหน่วยความจำตำแหน่งต่าง ๆ ที่ใช้แทนตำแหน่งของ pixel ต่าง ๆ บนหน้าจอ (video memory) ซึ่งใน mode 13h นี้จะเริ่มที่ segment A000h offset 0000h เป็นพื้นที่ 320*200 ช 64,000 ไบต์ การแทนค่าตำแหน่ง pixel บนหน้าจอ ด้วยตำแหน่งต่าง ๆ บนหน่วยความจำเพื่อแสดงข้อมูล ทำได้โดยใช้สมการ

memory_location = y*320 + x

เมื่อ x แทนตำแหน่ง pixel บนแกน x และ y แทนตำแหน่ง pixel บนแกน y หรือใช้ในลักษณะ

memory_location = (y << 8) + (y << 6) + x

โดย "<<" คือการเลื่อนซ้ายไปกี่ตำแหน่ง



3.3 การเตรียมข้อมูลของไฟล์ พีซีเอ็กซ์ (.pcx) เพื่อแสดงภาพ

ไฟล์พีซีเอ็กซ์ มีการเก็บข้อมูลด้วยการเข้ารหัสแบบ RLE (Run Length Encoding) คือเก็บว่าแต่ละสีมีจำนวน pixel เท่าไร โดยจัดเก็บตามตำแหน่งการแสดงบนหน้าจอ จากซ้ายไปขวา และ บนลงล่าง

โครงสร้างของไฟล์พีซีเอ็กซ์ ประกอบด้วย

3.3.1 ส่วนหัว (header) เก็บสารสนเทศ (information) เช่น รุ่น,ขนาดของรูป เป็นต้น ส่วนหัวนี้มีขนาด 128 ไบต์

3.3.2 ส่วนที่เก็บข้อมูลของรูป (data) ในลักษณะของ RLE มีขนาดและแต่ขนาดของแต่ละรูป

3.3.3 ที่เหลือ 768 ไบต์ ต่อจากส่วนที่ 2 เป็นส่วนเก็บข้อมูลของรูปแบบของสี (pattern) เรียงตามลำดับตั้งแต่ pattern 0-255 (แต่ละ pattern จะแทน 1 สี โดยที่แต่ละสีจะถูกแทนด้วยค่าของ rgb คือ แดง,เขียว และ น้ำเงิน รวม 3 ไบต์)

เนื่องจากไฟล์พีซีเอ็กซ์ มีการเข้ารหัส จึงต้องมีโปรแกรมที่ทำการถอดรหัสตัวข้อมูลก่อนนำไปใช้ และการนำไปใช้จะต้องมีการตั้งค่า pattern สีของหน้าจอให้เป็นแบบเดียวกับ pattern สีของรูปด้วย โดยตัวข้อมูลจะอยู่ในรูปของ offset ที่ระบุตำแหน่งใน index ของ pattern สี ที่ 0-255

3.4 การแก้ไขถาดสี (palette) ของหน้าจอ

เนื่องจากในรูป 256 สี แต่ละรูปจะมีถาดสี (palette) เป็นของตนเอง การแสดงภาพบนหน้าจอจึงจำเป็นต้องแก้ไขถาดสี (palette) ของหน้าจอให้ตรงกับรูปนั้น ๆ เสียก่อน มิฉะนั้นจะทำให้สีที่แสดงขึ้นมาไม่ถูกต้อง เนื่องจากการอ้างอิงด้วย index ทำให้ได้ข้อมูลที่ผิดพลาดเกิดขึ้น ทั้งนี้การแก้ไขเป็นการแก้ไขค่าของ rgb ภายในแต่ละ index เท่านั้น

ในการแสดงภาพระดับ VGA มีการใช้ช่องสัญญาณ (port) เกี่ยวกับสีดังนี้

port 0x3c6 : palette mask register

port 0x3c7 : palette read register

port 0x3c8 : palette write register

port 0x3c9 : palette data register

การ อ่าน เขียน ถาดสี ในแต่ละ index ทำได้ดังนี้

3.4.1 ให้ mask register มีค่า 0xFF ทุกครั้งก่อนที่จะทำการอ่าน หรือ เขียน

3.4.2 เลือก index ที่ต้องการจะใช้

3.4.3 ใช้ช่องสัญญาณ (port) อ่าน,เขียน และ ข้อมูล ในการทำงาน

การทำงานสามารถเขียนเป็นภาษาซี ได้ดังนี้

```
typedef struck RGB_color_typ
{  unsigned char red;
   unsigned char green;
   unsigned char blue;
 } RGB_color , *RGB_color_ptr;

void Set_Palette_Register ( int which_entry, RGB_color_ptr color)
{  asm cli;
   // Step 1
   outportb(0x3c6,0xff);

   // Step 2
   outportb(0x3c8,which_entry);

   // Step 3
   outportb(0x3c9,color->red);
   outportb(0x3c9,color->green);
   outportb(0x3c9,color->blue);
   asm sti;
}
void Get_Palette_Register ( int which_entry, RGB_color_ptr color)
{  asm cli;
   // Step 1
   outportb(0x3c6,0xff);

   // Step 2
   outportb(0x3c7,which_entry);

   // Step 3
   color->red = inportb(0x3c9);
   color->green = inportb(0x3c9);
   color->blue = inportb(0x3c9);
   asm sti;
}
```

3.5 เทคนิคในการเพิ่มความนุ่มนวลของการแสดงภาพ

การเขียนข้อมูลลงบนหน่วยความจำการแสดงภาพ (video memory) ตรง ๆ อาจทำให้ภาพที่เห็นเกิดการแตก (flicker) ขึ้นได้ ทั้งนี้เนื่องจากการปรับปรุงภาพใหม่ (update) ของหน่วยความจำการแสดงภาพ (video memory) ที่ไม่ถูกจังหวะกับการทำงานของส่วนการแสดงภาพ (video display) การทำงานของ VGA ที่ทำการตรวจหน่วยความจำการแสดงภาพ (video memory) และส่งค่าสีไปแสดงในบางครั้งอาจจะไปตรงกับจังหวะที่ตัวโปรแกรม กำลังแก้ไขค่าในหน่วยความจำการแสดงผล (video memory) ตัวอย่างเช่น โปรแกรมได้ทำการลบค่าข้อมูลของรูปเก่าออกไปแล้ว แต่ยังเขียนข้อมูลของรูปใหม่ลงบนหน่วยความจำการแสดงผล (video memory) ไม่เสร็จ เมื่อ VGA ทำการตรวจจะได้ข้อมูลไม่เต็มรูป

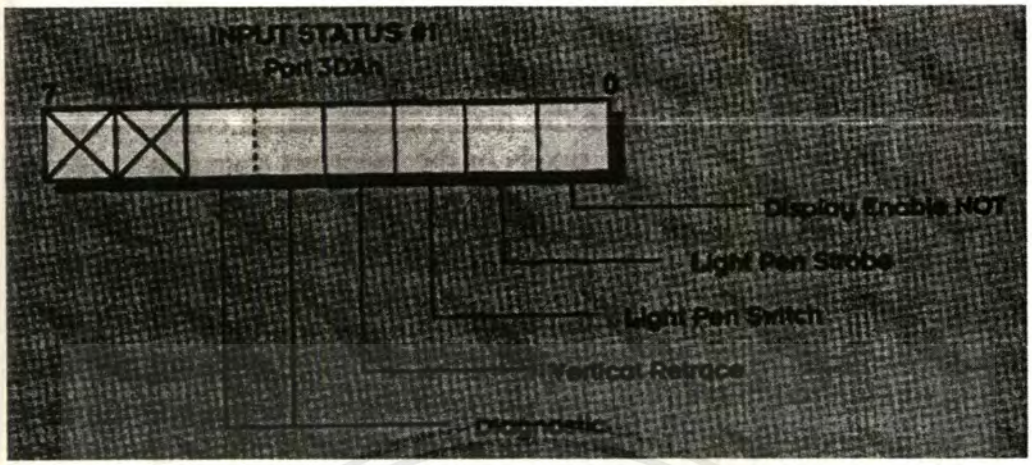
ในตัวโปรแกรมของโปรเจค ได้ใช้เทคนิค double buffering คือการนำข้อมูลที่ต้องการแสดงบนหน้าจอมาเขียนลงบนหน่วยความจำที่ทำหน้าที่เป็นส่วนแสดงภาพสำรอง (video buffer) เพื่อนำไปเขียนลงบนหน่วยความจำการแสดงภาพ (video memory) จริง ๆ ในคราวเดียว การแก้ไขต่าง ๆ ไม่ว่าจะมากหรือน้อยจะถูกทำงานบนส่วนแสดงภาพสำรอง (video buffer) ฉะนั้นเราจะเห็นภาพที่สมบูรณ์เท่านั้น

ข้อเสียของวิธี double buffering นี้ คือ

3.5.1 ทำให้ความเร็วของการแสดงภาพช้าลง คือ เราจะไม่เห็นภาพที่ถูกเปลี่ยนแปลงในทันที แต่โดยทั่วไปแล้ว ด้วยความสามารถของการ์ดแสดงภาพ (display card) ในปัจจุบันเราจะไม่เห็นความแตกต่างของความเร็วส่วนนี้

3.5.2 ทำให้จำเป็นต้องมีการจองหน่วยความจำเพิ่มอีก 1 เท่า เพื่อมาเป็นส่วนแสดงผลสำรอง (video buffering)

อีกเทคนิคหนึ่ง คือ การป้องกันไม่ให้การทำงานของ VGA มาทำงานเวลาเดียวกับการแก้ไขหน่วยความจำการแสดงภาพ (video memory) จากความรู้เรื่องของ การทับรอยเดิมในแนวแกนนอน และแนวแกนตั้ง (horizontal retrace , vertical retrace) ที่จะมีการหยุดการยิงลำแสงชั่วขณะหนึ่ง จึงมีเทคนิคที่จะแก้ไขหน่วยความจำการแสดงภาพ (video memory) ในขณะที่ส่วนแสดงภาพ (video display) อยู่ในสถานะดังกล่าวซึ่งสถานะที่เหมาะสมที่สุดคือ การทับรอยเดิมในแนวแกนตั้ง (vertical retrace) ที่จะมีการหยุดยิงลำแสงเป็นระยะเวลานานที่สุด (wait_vertical_retrace) การตรวจสอบสถานะของส่วนแสดงภาพ (video display) ทำได้โดยการตรวจสอบจาก ช่องสัญญาณ (port) 3DAh ที่ทำหน้าที่เป็น register สถานะ (status register) ของส่วนแสดงภาพ (video display) ที่ bit vertical retrace และ display enable not



รูป 3-2 แสดง input Status #1 register

การทับรอยเดิมในแนวแกนตั้ง	Display Enable Not	VGA Operation
0	0	Scanning pixels
1	0	Not possible
0	1	Horizontal Retrace
1	1	Vertical Retrace

ตาราง 3-1 แสดงการทับรอยเดิมในแนวแกนตั้ง และ Display Enable Not Bit Combinations

บทที่ 4
การทำงานโดยรวมของระบบ

ระบบในโครงงานนี้จะเป็นการวนซ้ำจนกว่าจะจบการทำงาน (โดยการกดปุ่ม ESC) โดยใน 1 รอบของการทำงานนั้น จะแบ่งการทำงานออกเป็น 8 ขั้นตอนหลัก ซึ่งทุก ๆ รอบของการทำงานจะต้องผ่านการทำงานทั้ง 8 ขั้นตอนนี้ทั้งสิ้น

4.1 การทำงาน 8 ขั้นตอนหลัก ของแต่ละรอบของการวนซ้ำ

การทำงานทั้ง 8 ขั้นตอน นี้จะเริ่มตั้งแต่ การดึงข้อมูล (load) มาจากไฟล์ต่าง ๆ , การตรวจสอบแป้นพิมพ์ เป็นต้น โดยทั้ง 8 ขั้นตอน จะประกอบด้วย

4.1.1 การติดต่อกับไฟล์

4.1.2 การตัดสินใจของตัวละคร

4.1.3 การเขียนข้อมูลลงบนส่วนการแสดงภาพสำรอง

4.1.4 การแสดงภาพออกทางหน้าจอ

4.1.5 การแก้ไขและเตรียมข้อมูลสำหรับการแสดงภาพครั้งต่อไป

4.1.6 การตรวจสอบการชนกันของออปเจกต์ (collision detection)

4.1.7 การจัดการกับแอตทริบิวต์ที่เกี่ยวกับหน้าจอของตัวละคร

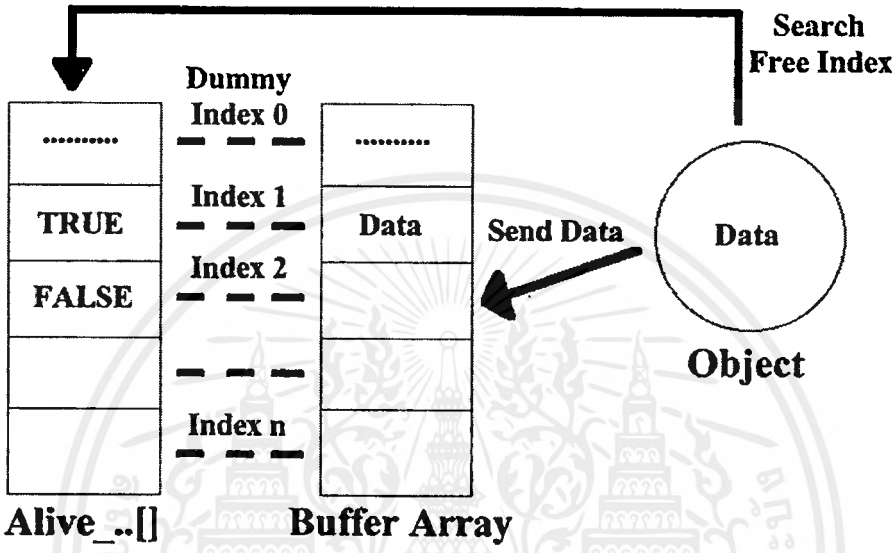
4.1.8 การจัดการในด้านอื่น ๆ

โดยทั้ง 8 ขั้นตอนจะอธิบายแยกเป็นบท ซึ่งจะต่อท้ายจากบทนี้เป็นต้นไป

4.2 การทำงานกับหน่วยความจำสำรอง (buffer)

ในระบบจะมีการเตรียมหน่วยความจำสำรอง (buffer) ในรูปของอาร์เรย์ ไว้ใช้เก็บข้อมูลต่าง ๆ ของ ออปเจกต์ ทั้ง sprite (list[]) และ block (list_block[]) โดยใช้อาร์เรย์อีกชุดหนึ่งซึ่งมีจำนวนห้อง (slot)เท่ากับอาร์เรย์ของออปเจกต์ (array of buffer) ในการเก็บข้อมูล แสดงการถูกใช้งานของอาร์เรย์เหล่านั้น (Alive_Sprite[] สำหรับ list[] และ Alive_Block[] สำหรับ list_block[]) ถ้าเป็น TRUE ที่ index เดียวกันบนอาร์เรย์ของออปเจกต์ หมายถึงมีการเก็บข้อมูลที่ต้องการอยู่ และถ้าเป็น FALSE แสดงว่าห้อง (slot) ของ index นั้นว่าง หรือไม่ต้องการข้อมูลในห้อง (slot) นั้นแล้ว

สำหรับการนำข้อมูลของออปเจกต์ มาเก็บในอาร์เรย์ของออปเจกต์ จะใช้ห้องแรกสุดที่ว่าง โดยตรวจจาก Alive_Sprite[] สำหรับ sprite และ Alive_Block สำหรับ block (** ฟังก์ชัน Search_Free_Index () และ Search_Free_Index_Block() **) แล้วจึงให้ค่า TRUE แก่ index ที่ต้องการใช้เก็บข้อมูล



รูป 4-1 โครงสร้างการใช้อาร์เรย์ของระบบ

การแสดงการเลิกใช้งานห้อง (slot) ของอาร์เรย์ของออปเจกต์ (release buffer) ทำได้โดยให้ค่า FALSE แก่ Alive_..[] ที่ index เดียวกัน ซึ่งสาเหตุของการเลิกใช้งานห้อง (slot) เกิดได้จาก

4.2.1 ออปเจกต์นั้นอยู่นอกขอบเขตของการเก็บข้อมูล ซึ่งในระบบนี้จะเก็บข้อมูลของทุก ออปเจกต์ ที่อยู่ภายใน 3 หน้าจอ (screen) โดยที่หน้าจอหนึ่งๆ จะมีขนาด 320*200 pixels

4.2.2 เกิดการตายของ sprite คือ hp ของตัวละครเท่ากับ 0

4.2.3 sprite ของ อาวุธ และ กระสุน ที่จะ หายไปเมื่อเกิดการชนกับ block หรือ sprite ของตัวละครฝ่ายตรงข้าม

4.2.4 sprite ของสิ่งของที่จะหายไปเมื่อตัวละครฝ่ายผู้เล่นได้เดินมาชนเพื่อเก็บ (** การตรวจสอบสภาวะของห้อง (slot) ว่าสมควรถูกเลิกใช้งานหรือไม่ และ การให้ค่า FALSE แก่ Alive_..[] ทำโดยฟังก์ชัน Recycle_List() **)

4.3 รายละเอียดของออปเจกต์ที่ใช้

เพื่อการทำงานที่เป็นระบบ และ ง่ายต่อความเข้าใจ จึงได้กำหนดออปเจกต์ที่ใช้ในระบบ เป็น 2 ประเภท ได้แก่ sprite และ block

4.3.1 sprite

เป็นออปเจกต์ที่ใช้เก็บข้อมูลต่าง ๆ ของตัวละคร ออปเจกต์ประเภทนี้จะ มีความคิด , สามารถเคลื่อนไหวได้ และมีความสัมพันธ์กับชุดของรูปประจำตัวละคร (ชุดของเฟรม) ที่จะนำเสนอ ทำให้เห็นว่าสามารถเปลี่ยนรูปร่างท่าทางได้

แอตทริบิวต์ต่าง ๆ ที่มีในออปเจกต์ประเภทนี้ ได้แก่

4.3.1.1 name : ชนิดของตัวละคร

4.3.1.2 index_list : index ของอาร์เรย์ของออปเจกต์ ที่เก็บข้อมูลของตัวละคร (array of buffer)

4.3.1.3 x , y : ค่าของตำแหน่งมุมซ้ายบนของ sprite เมื่อเทียบจากตำแหน่งของหน้าจอที่แสดงอยู่ในปัจจุบัน

4.3.1.4 T_dx , T_dy : ระยะห่างระหว่างด้านซ้ายของ sprite จนถึงจุดแรกของรูปในแนวแกน x ที่เป็นส่วนหนึ่งของตัวละคร และ ระยะห่างระหว่างด้านบนของ sprite จนถึงจุดแรกของรูปในแนวแกน y ที่เป็นส่วนหนึ่งของตัวละคร ตามลำดับ

4.3.1.5 PT_dx , PT_dy : ค่าของ T_dx และ T_dy ของรูป ในการวนซ้ำ (loop) ที่ผ่านมา

4.3.1.6 x_on_map , y_on_map : ค่าของตำแหน่งมุมซ้ายบนของ sprite เมื่อเทียบกับฉาก

4.3.1.7 x_start , y_start : ค่า x_on_map และ y_on_map ณ. ตำแหน่งที่ตัวละครปรากฏตัว

4.3.1.8 x_vel , y_vel : ความเร็วในแกน x และ ความเร็วในแกน y ของ sprite ตามลำดับ

4.3.1.9 w , h : ความกว้าง และความสูงของ sprite ตามลำดับ โดยจะมีค่าเท่ากับความกว้างและ ความสูงของรูปที่จะถูกแสดงโดย sprite นั้น ๆ

4.3.1.10 T_w , T_h : ระยะห่างระหว่างด้านซ้ายของ sprite จนถึงจุดที่ไกลที่สุดของรูปในแนวแกน x ที่เป็นส่วนหนึ่งของตัวละคร และ ระยะห่างระหว่างด้านบนของ sprite จนถึงจุดที่ไกลที่สุดของรูปในแนวแกน y ที่เป็นส่วนหนึ่งของตัวละคร ตามลำดับ

4.3.1.11 PT_w , PT_h : ค่าของ T_w และ T_h ของรูปในการวนซ้ำที่ผ่านมาล่าสุด num_frames : จำนวนเฟรมทั้งหมดของ sprite

4.3.1.12 cur_frame : เลขที่ของเฟรมปัจจุบันที่ใช้

4.3.1.13 direction : แสดงทิศทางการหันหน้าของตัวละคร เก็บค่า RIGHT หรือ LEFT

4.3.1.14 d_scr : ความสามารถในการป้องกันไม่ให้ sprite หลุดออกจากหน้าจอ เก็บค่า TRUE สำหรับการป้องกัน และ FALSE สำหรับการไม่ป้องกัน

4.3.1.14 in_action : อยู่ในระหว่างอิริยาบถใด ๆ หรือไม่ เพราะว่าการอยู่ในอิริยาบถหนึ่ง ๆ จะ ทำให้การทำงานกับ sprite นั้น ๆ เปลี่ยนไป เช่น ความเร็ว เป็นต้น เก็บค่า TRUE สำหรับการแสดงว่าอยู่ในอิริยาบถ และ FALSE สำหรับการแสดงว่าไม่ได้อยู่ในอิริยาบถใด ๆ

4.3.1.15 what_action : ชนิดของอิริยาบถที่ทำอยู่

4.3.1.16 step_action : แอตทริบิวต์เสริมที่ใช้แสดงว่ากำลังอยู่อิริยาบถขั้นตอนใด

4.3.1.17 have_shot : แสดงว่า sprite ได้มีการสร้างลูกหรือไม่ (ในระบบใช้แสดงว่ามีการยิงกระสุนหรือไม่)

4.3.1.18 index_child : index_list ของ sprite ที่เป็นลูก

4.3.1.19 index_parent : index_list ของ sprite ที่เป็นพ่อ

4.3.1.20 index_attack : index_list ของ sprite อื่นที่ชนกับ sprite นั้น ๆ

4.3.1.21 allow_other_over : แสดงการอนุญาตให้ sprite อื่นสามารถทับได้ (ใช้ในการอนุญาตให้เกิดการซ้อนกันของตัวละคร) โดย TRUE หมายถึง การอนุญาต และ FALSE หมายถึง การไม่อนุญาต

4.3.1.22 now_over_other : index_list ของที่ถูก sprite นั้น ๆ ทับอยู่

4.3.1.23 collide : ระบุลักษณะการชนกันที่เกิดขึ้นกับ sprite นั้น ๆ โดยแบ่งลักษณะการชนกันได้ดังนี้

4.3.1.23.1 BLK_UNDER : sprite อยู่ใต้ block ที่ชน

4.3.1.23.2 BLK_UPPER : sprite อยู่เหนือ block ที่ชน

4.3.1.23.3 BLK_LEFT : sprite อยู่ทางซ้ายของ block ที่ชน

4.3.1.23.4 BLK_RIGHT : sprite อยู่ทางขวาของ block ที่ชน

4.3.1.23.5 SPR_UNDER : sprite อยู่ใต้อีก sprite หนึ่ง

4.3.1.23.6 SPR_UPPER : sprite อยู่เหนืออีก sprite หนึ่ง

4.3.1.23.7 SPR_LEFT : sprite อยู่ทางซ้ายของอีก sprite หนึ่ง

4.3.1.23.8 SPR_RIGHT : sprite อยู่ทางขวาของอีก sprite หนึ่ง

4.3.1.23.9 TRUE : เกิดการชนกัน โดยไม่สนใจทิศทาง (ในระบบจะกำหนดค่านี้ให้กับชนิดของตัวละครที่ต้องการให้หายไปเมื่อเกิดการชน)

4.3.1.23.10 FALSE : ไม่มีการชน

มีการกำหนดค่าให้เป็น FALSE ในทุกการวนซ้ำ (loop) ก่อนการตรวจสอบการชน

4.3.1.24 on_floor : sprite อยู่บน block หรือไม่ (TRUE หรือ FALSE) ใช้กับฟังก์ชัน Gravity()

4.3.1.25 gravity_on : ต้องการให้ sprite อยู่ภายใต้แรงดึงดูดหรือไม่ ถ้าเป็น TRUE หมายถึง ให้ sprite นั้นอยู่ภายใต้แรงดึงดูด เช่น เดินตกเหวได้ เป็นต้น แต่ถ้าเป็น FALSE หมายถึง ให้ sprite นั้นไม่ต้องอยู่ภายใต้แรงดึงดูด หรือ เหาะได้

4.3.1.26 wounded : แสดงว่า sprite นั้นได้รับบาดเจ็บอยู่หรือไม่ ถ้าเป็น TRUE แสดงว่าบาดเจ็บ และถ้าเป็น FALSE แสดงว่าไม่บาดเจ็บ โดยค่าจะถูกกำหนดให้เป็น FALSE ในทุกการวนซ้ำ (loop) ก่อนการตรวจสอบการชน

4.3.1.27 no_collide : แสดงว่าความต้องการให้ sprite นั้นเกิดการชนกับ sprite ได้หรือไม่

TRUE สำหรับการไม่เกิดการชน กรณีนี้ sprite จะสามารถซ้อนกันโดยไม่ต้องตรวจสอบการชน และ FALSE ซึ่งเป็นค่าปกติที่กำหนด คือ ให้มีการชนกันของ sprite

4.3.2 block

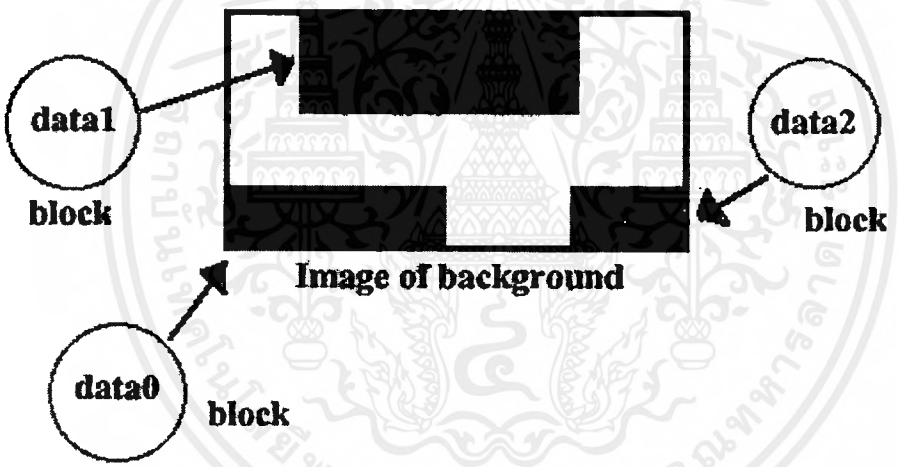
เป็นออปเจกต์ที่เก็บข้อมูลวัตถุต่าง ๆ ซึ่งจะไม่มีชีวิต โดยการกำหนดค่าข้อมูลให้ block จะต้องสอดคล้องกับรูปของฉาก เพราะ ฉากคือรูปของ block และ block เก็บข้อมูลของวัตถุต่าง ๆ ในฉาก เช่น พื้น , กำแพง , เพดาน เป็นต้น ขนาดของ block จะต้องเป็นขนาดของรูปจริง ๆ ของวัตถุนั้นบนฉาก

แอตทริบิวต์ต่าง ๆ ที่มีในออปเจกต์ประเภทนี้ ได้แก่

4.4.1 x_on_map , y_on_map : ตำแหน่งของจุดมุมซ้ายบนของ block บนฉาก

4.4.2 x, y : ตำแหน่งของจุดมุมซ้ายบนของ block บนหน้าจอ

4.4.3 w, h : ความกว้าง และความสูงของ block



รูป 4-2 ความสัมพันธ์ระหว่าง block และฉาก

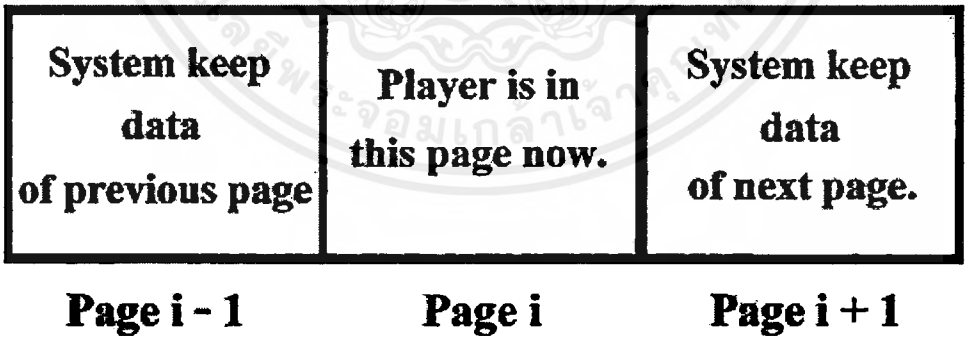
บทที่ 5
การเตรียมข้อมูลของตัวละคร การตัดสินใจของตัวละคร และการแสดงภาพ

5.1 หน้าที่

ทำการจัดเตรียมข้อมูลของแต่ละออปเจกต์ที่อยู่ในรัศมี 3 หน้าจอ จากตำแหน่งของตัวละครฝ่ายผู้เล่น และจัดการเตรียมอิริยาบทที่แต่ละตัวละครจะแสดงโดยจะขึ้นอยู่กับสถานะการณ์แต่ละสถานะการณ์ รวมทั้งการเตรียมการการแสดงภาพออกทางหน้าจอ

5.2 การติดต่อกับไฟล์

การกำหนดรายละเอียดให้กับแต่ละฉากสามารถทำได้ โดยการเก็บข้อมูลเหล่านั้นลงไปในไฟล์ที่มีสกุล *.fin ในแต่ละการวนซ้ำ (loop) ของการทำงานของระบบ จะมีการตรวจสอบตำแหน่งของตัวละครของผู้เล่น เพื่อเตรียมข้อมูลของฉากให้ครบทั้ง 3 หน้าจอ ซึ่งการเตรียมนี้ทำได้โดยการดึงข้อมูลจากไฟล์ของฉากนั้น ๆ



รูป 5-1 การทำงานกับข้อมูลของ 3 หน้าจอ

การทำงานลักษณะนี้สามารถสรุปได้ว่า ข้อมูลของตัวละครตัวเดิมจะถูกอ่านขึ้นมาอีกครั้ง เมื่อตัวละครของผู้เล่นได้ออกจาห่างจากจุดที่ตัวละครนั้น ๆ ปรากฏ เป็นระยะทาง 2 หน้าจอขึ้นไป กรณีที่ตัวละครฝ่ายศัตรูตามตัวละครของผู้เล่นไปยังหน้าจออื่นจนพ้น 2 หน้าจอ เมื่อตัวละครของผู้เล่นวกกลับมา

ยังหน้าจอที่เป็นที่ปรากฏตัวครั้งแรกของตัวละครฝ่ายศัตรู โดยที่มีตัวละครฝ่ายศัตรูยังคงตามมาด้วย ระบบจะป้องกันไม่ให้เกิดการสร้างตัวละครซ้ำซ้อน ด้วยการตรวจสอบจากจุดที่จะให้เกิดตัวละครใหม่ เทียบแล้วจะต้องไม่ซ้ำกับจุดเริ่มต้น (แอตทริบิวต์ x_start และ y_start) ของตัวละครอื่นที่ยังคงมีชีวิตอยู่ (Alive_Sprite ของ index นั้น ๆ เป็น TRUE)

ข้อมูลจากไฟล์ที่ตัวระบบนำมาใช้ ได้แก่

5.2.1 จำนวนหน้าจอทั้งหมดของฉากนั้น ๆ ขนาด 8 บิต

5.2.2 ข้อมูลของออปเจกต์ ประเภท block ได้แก่

5.2.2.1 x ขนาด 32 บิต

5.2.2.2 y ขนาด 32 บิต

5.2.2.3 ความกว้าง ขนาด 32 บิต

5.2.2.4 ความสูง ขนาด 32 บิต

5.2.3 ข้อมูลของออปเจกต์ ประเภท sprite ได้แก่

5.2.3.1 x ขนาด 32 บิต

5.2.3.2 y ขนาด 32 บิต

5.2.3.3 ชนิดของตัวละคร ขนาด 8 บิต

ฟังก์ชันต่าง ๆ ที่ทำหน้าที่ในการติดต่อกับไฟล์ ได้แก่ Load_One_Page() , Prepare_Sprite() , Prepare_Block ()

5.3 การตัดสินใจของตัวละคร

เนื่องจากเกมส์ที่เขียนขึ้นในโครงงานนี้จะมีตัวละครซึ่งจะแทนด้วยออปเจกต์ต่าง ๆ ในส่วนนี้จะเป็นการเลือกการตัดสินใจทำอิริยาบทต่าง ๆ ของตัวละครแต่ละตัวทั้งตัวละครที่ผู้เล่นบังคับและศัตรูที่คอมพิวเตอร์เป็นผู้กำหนด เช่น เดิน กระโดด ยิงกระสุน เป็นต้น

5.3.1 ตัวละครฝ่ายพระเอก

ตัวละครฝ่ายพระเอกจะเป็นตัวละครที่ผู้เล่นบังคับในเกมส์จะชื่อว่า อัศวิน (KNIGHT) ซึ่งเป็นพระเอกในเกมส์ สำหรับ อัศวิน นี้เนื่องจากเป็นการบังคับจากผู้เล่นโดยตรง การแสดงอิริยาบทต่าง ๆ จึงขึ้นอยู่กับการกดผ่านแป้นพิมพ์ของผู้เล่น ดังนั้นในส่วนนี้จึงมีฟังก์ชันสำหรับตรวจสอบแป้นพิมพ์ซึ่งจะตรวจสอบปุ่มที่ถูกกด

5.3.1.1 ความรู้เบื้องต้นในการทำงานติดต่อกับแป้นพิมพ์ (keyboard)

เมื่อมีการกดปุ่มบนแป้นพิมพ์ จะทำให้เกิดข้อมูลขึ้นข้อมูลหนึ่งซึ่งจะหน้าที่แทนสภาวะของแป้นพิมพ์ว่ามีปุ่มใดบ้างที่ถูกกดอยู่ ค่าข้อมูลนี้จะถูกส่งไปที่ ส่วนควบคุมแป้นพิมพ์ (keyboard controller) ที่อยู่ใน พีซี เพื่อแปลงให้เป็นข้อมูลที่ตัวระบบเข้าใจ เรียกข้อมูลนี้ว่า ข้อมูลที่ถูกตรวจพบ (scan code) หลังจากนั้นส่วนควบคุมแป้นพิมพ์ จะส่งสัญญาณ interrupt 9 เพื่อบอกให้โพรเซสเซอร์ ได้รู้ว่าเกิดเหตุการณ์ขึ้นบนแป้นพิมพ์ หรือเป็นการเรียกใช้ โปรแกรมย่อยตอบรับการขัด

จังหวะ (software interrupt service routine , ISR) ของ interrupt 9 ที่อยู่บน BIOS นั่นเอง การทำงานของ ISR ของแป้นพิมพ์ จะเริ่มด้วยการอ่านข้อมูลจากส่วนควบคุมแป้นพิมพ์ เพื่อตรวจสอบว่าอะไรเกิดขึ้น ซึ่งข้อมูลนี้ก็คือ ข้อมูลที่ถูกตรวจสอบ (scan code) นั่นเอง ข้อมูลที่ถูกตรวจสอบ (scan code) มีขนาด 8 บิต โดย บิตสูง (high bit) จะเป็น "0" เมื่อได้กดปุ่ม และเป็น "1" เมื่อปล่อย ส่วนบิต ที่เหลือเก็บค่าของปุ่มพิมพ์ (key) เช่น code 1Eh บอกว่า ปุ่ม "A" ได้ถูกกด และ B3h บอกว่า ปุ่ม "." ได้ถูกต้องปล่อย เป็นต้น

รูป 5-2 รูปแบบของข้อมูลที่ถูกตรวจสอบ

ค่าของ ข้อมูลที่ถูกตรวจสอบ (scan code) จะถูกแปลงเป็นรหัสแอสกี (ASCII) ต่อไปโดยใช้ข้อมูลของแฟลก (flag) ที่เก็บข้อมูลว่ามีการกดปุ่มพิมพ์ (key) พิเศษด้วยหรือไม่ เช่น Ctrl , Alt , Shift เป็นต้น สนับสนุน ค่า ASCII ที่ถูกแปลงขึ้นมานี้จะถูกจัดเก็บเป็นคิว (queue) แล้วจะมีการเรียกใช้ข้อมูลในคิว (queue) นี้ผ่าน BIOS หรือ DOS

การติดต่อกับส่วนควบคุมแป้นพิมพ์ เราใช้ 2 ช่องสัญญาณอินพุท/เอาท์พุท (I/O port) และ 60h และ 64h ช่องสัญญาณ 60h ทำหน้าที่เป็นช่องสัญญาณอินพุท (input port) สำหรับเก็บข้อมูลที่ถูกตรวจพบ (scan code) จากส่วนควบคุมแป้นพิมพ์ และเป็นช่องสัญญาณเอาท์พุท (output port) สำหรับการส่งข้อมูลบางอย่างไปที่ส่วนควบคุมแป้นพิมพ์ หรือแป้นพิมพ์ สำหรับช่องสัญญาณ 64h เป็น register สถานะ (status register) สำหรับตอนอ่าน และเป็น register คำสั่ง (command register) สำหรับตอนเขียน

5.3.1.2 การติดต่อกับแป้นพิมพ์ภายในโปรเจค

ภายในโปรเจค นี้ได้ทำการเขียน ISR ของแป้นพิมพ์ขึ้นมาใหม่ โดยทำการจอง index ขนาดเท่ากับ จำนวนปุ่มพิมพ์ทั้งหมดบนแป้นพิมพ์ คือ 128 ปุ่ม ไว้เก็บข้อมูลว่าปุ่ม หนึ่ง ๆ ถูกกดอยู่หรือไม่เพื่อความสะดวกในการใช้งาน เพราะในเกมส์จะใช้ปุ่มพิมพ์ ไม่ครบทุกปุ่ม ฉะนั้นการทำงานของโปรแกรม จึงอาศัยการตรวจสอบว่าปุ่มที่ต้องการถูกกดอยู่หรือไม่

ซึ่งปุ่มที่ใช้ในเกมส์จะประกอบด้วย

5.3.1.2.1 ปุ่มลูกศรทางซ้าย แสดงการเคลื่อนที่ไปทางซ้าย

5.3.1.2.2 ปุ่มลูกศรทางขวา แสดงการเคลื่อนที่ไปทางขวา

5.3.1.2.3 ปุ่มลูกศรขึ้นบน แสดงการเคลื่อนที่ขึ้นข้างบน

5.3.1.2.4 ปุ่มลูกศรลงล่าง แสดงการเคลื่อนที่ลงข้างล่าง

5.3.1.2.5 ปุ่ม Space bar แสดงการกระโดด

5.3.1.2.6 ปุ่ม Ctrl แสดงท่าทางการยิงกระสุน

5.3.1.2.7 ปุ่ม "A" แสดงท่าทางการฟันดาบ

5.3.1.2.8 ปุ่ม Esc ออกจากโปรแกรม

โดยใน 1 รอบการวนซ้ำจะมีการเรียกฟังก์ชันตรวจสอบการกดแป้นพิมพ์ 2 ครั้ง ดังนั้นความเร็วของตัวละครจะเพิ่มขึ้นหากมีการกดปุ่มค้างไว้

5.3.2 ตัวละครฝ่ายศัตรู

ตัวละครฝ่ายศัตรูจะถูกควบคุมโดยคอมพิวเตอร์ซึ่งจะเป็นตัวกำหนดอิริยาบทต่าง ๆ ที่แต่ละตัวละครจะแสดงออกมา ซึ่งจะขึ้นอยู่กับสถานะการณ์โดยใช้หลักปัญญาประดิษฐ์ (Artificial Intelligent) เข้ามาช่วย โดยแต่ละตัวจะมีรัศมีการมองเห็นไม่เท่ากัน และเมื่อพระเอกเข้ามาอยู่ในรัศมีการโจมตีก็จะมีการโจมตีซึ่งจะขึ้นอยู่กับแต่ละตัวละคร โดยในการเลือกการตัดสินใจนี้จะเป็นการตรวจสอบทุกตัวละครที่มีชีวิตอยู่ในรัศมี 3 หน้าจอ ซึ่งจะป้อนข้อมูลของพระเอกทั้งหมด เช่น อิริยาบทของพระเอก , ความเร็ว , ระยะห่าง หรือ พลังชีวิต และข้อมูลของตัวมันเอง เช่น พิกัดที่มันอยู่ , ความเร็ว เป็นต้น เพื่อใช้ในการคำนวณ และผลลัพธ์ของส่วนนี้คือ อิริยาบทที่เหมาะสมของแต่ละตัวละครที่จะกระทำต่อไป เช่น ทำการกระโดด การยิงกระสุน การเดิน เป็นต้น

5.4 การเขียนข้อมูลลงบนส่วนการแสดงภาพสำรอง

ก่อนจะแสดงข้อมูลออกทางหน้าจอได้นั้นจะต้องนำข้อมูลของภาพต่าง ๆ อันประกอบด้วย

- รูปของฉากบนหน้าจอนั้น (background)
- รูปของตัวละครในเฟรม (frame) ที่ต้องการ
- รูปของข้อความต่าง ๆ เช่น ค่าพลังชีวิตของผู้เล่น (hp)

มาเขียนใส่ลงบนหน่วยความจำที่ได้จองไว้ด้วยขนาดเท่ากับหน้าจอ ตามเทคนิคของการทำ Double Buffering

ข้อมูลที่มีการขนถ่ายในส่วนนี้ของระบบ คือ ค่าของ index ของชุดสี (palette)

5.4.1 การเขียนรูปของฉาก

เนื่องจากได้ทำการเก็บรูปของฉากทั้งหมดไว้ในหน่วยความจำตั้งแต่ต้นโปรแกรม ทำให้สามารถอ้างอิงข้อมูลของรูปในทุกจุดได้โดยผ่าน offset

ทำการคัดลอก (copy) ข้อมูลจากหน่วยความจำที่เก็บรูป โดยเริ่มจากตำแหน่งที่ offset ชี้อยู่ ไปใส่ ณ. ตำแหน่งแรก ในหน่วยความจำแสดงภาพสำรองจนได้ขนาดเท่ากับ 1 หน้าจอ ในที่นี้คือ 320 * 200 หรือเขียนจนเต็มหน่วยความจำแสดงภาพสำรองนั่นเอง

5.4.2 การเขียนรูปของตัวละคร

ได้มีการเรียกใช้ฟังก์ชัน Show_Sprite() ซึ่งจะมีการกระบวนการทำงาน ที่เริ่มจากการคัดลอกข้อมูลของฉากที่ถูกเก็บอยู่ในหน่วยความจำแสดงภาพสำรอง ในทุก ๆ ตำแหน่งที่จะนำข้อ-มูลรูป

ของออปเจกต์ไปแทนที่ มาเก็บไว้ในหน่วยความจำประจำของออปเจกต์นั้น ๆ ซึ่งในออปเจกต์จะมีตัวชี้ (pointer) ที่ชี้ไปยังหน่วยความจำนี้ คือ แอตทริบิวต์ background แล้วจึงทำการเขียนค่า index ของแต่ละตำแหน่งบนรูป ลงบนหน่วยความจำแสดงภาพสำรอง ทั้งนี้สำหรับ index ที่มีค่าเป็น 0 จะไม่เขียนลงไป เพื่อให้ส่วนนั้นของรูปเห็นเป็นสีของฉากแทน (transparent)

การทำงานในส่วนนี้ยังสามารถนำไปใช้เป็นส่วนหนึ่งของเทคนิคการแสดงรูปของตัวละครที่ซ้อนกันได้ โดยอาศัยหลักการที่ว่าตัวละครที่จะถูกทับจะต้องมีการเรียก Show_Sprite() ของตัวละครนั้นก่อนตัวละครที่จะทับ ซึ่งจะทำให้บริเวณที่จะเกิดการซ้อนและได้เขียนข้อมูลของตัวละครที่จะถูกทับไปแล้ว ถูกแทนที่ด้วยข้อมูลของตัวละครที่จะมาทับ

5.4.3 การเขียนรูปของข้อความต่าง ๆ

คล้ายกับกรณีของการเขียนรูปของตัวละคร แต่จะไม่มีการเก็บข้อมูลของฉากจากหน่วยความจำแสดงภาพสำรอง เพราะว่าตำแหน่งที่จะนำรูปของข้อความต่าง ๆ ออกแสดงบนหน้าจอนี้จะเป็นตำแหน่งเดิมเสมอ

การเขียนข้อมูลทั้งสามประเภทนี้ลงบนหน่วยความจำแสดงภาพสำรอง จำเป็นที่จะต้องมีการเรียงลำดับการเขียนก่อน – หลัง เพื่อให้เกิดการซ้อนกันของรูปที่ถูกต้อง เช่น ถ้าเขียนข้อมูลของตัวละครก่อนแล้วจึงเขียนข้อมูลของฉาก จะทำให้ข้อมูลของรูปตัวละครทั้งหมดถูกทับโดยฉาก ทำให้ไม่เห็นรูปของตัวละคร เป็นต้น

5.5 การแสดงภาพออกทางหน้าจอ

เป็นส่วนที่ทำการแสดงภาพในส่วนการแสดงภาพสำรองที่เตรียมไว้พร้อมแล้ว ออกทางหน้าจอด้วย mode 13h

5.5.1 การรอการทับรอยเดิมในแนวแกนนอน (wait vertical retrace)

สำหรับการแสดงภาพในโครงงานนี้นั้น จะมีเทคนิคหลัก ๆ คือ จะต้องมีการรอการทับรอยเดิมในแนวแกนตั้ง (vertical retrace) ก่อนที่จะแสดงภาพแต่ละครั้งออกไป มิฉะนั้นอาจจะทำให้การแสดงภาพเกิดความไม่สมบูรณ์ของภาพได้ เช่น การการแตกของภาพ (flicker) หรือแสดงภาพได้ไม่ครบ อย่างที่ได้อ้างถึงแล้วในบทที่ 3

ดังนั้นจะก่อนที่จะแสดงภาพที่เตรียมไว้ออกทางหน้าจอทุกครั้งจะต้องเรียกฟังก์ชัน vsync(); ก่อนทุกครั้ง โดยฟังก์ชัน vsync(); นี้เป็นฟังก์ชันที่อยู่ใน library ของ Allegro.h ฟังก์ชัน vsync(); นี้ จะมีการรอการเกิดการทับรอยเดิมในแนวแกนตั้ง (vertical retrace) ก่อนจึงจะจบการทำงาน

ดังนั้นหากมีการเรียกฟังก์ชันนี้ ขณะที่ทำการทับรอยเดิมในแนวแกนนอน (horizontal retrace) อยู่และยังมีบรรทัดเหลืออยู่ ฟังก์ชัน vsync(); ก็จะทำการรอจนกว่าจะเกิดการทับรอยเดิมในแนวแกนนอน (horizontal retrace) ครั้งสุดท้าย และเริ่มทำการทับรอยเดิมในแนวแกนตั้ง (vertical retrace) ก่อน จึงจะหลุดการการทำงาน

5.5.2 การคัดลอกข้อมูลลงบนหน่วยความจำการแสดงภาพ

หลังจากจบการทำงานของฟังก์ชัน vsync(); แล้ว จึงจะเริ่มนำภาพที่เตรียมไว้ในส่วนแสดงภาพสำรองมาแสดงออกทางหน้าจอ โดยใช้ฟังก์ชัน blit จะทำการคัดลอก (copy) ข้อมูลซึ่งเป็นภาพจากต้นฉบับ (source) มายังเป้าหมาย (destination) ซึ่งจะมีการกำหนดจุดเริ่มต้นของต้นฉบับ และจุดเริ่มต้นของเป้าหมาย และขนาดของข้อมูล

blit (source,destination,x1,y1,x2,y2,width,height);

โดย source คือ ต้นฉบับ

destination คือ เป้าหมาย

x1,y1 คือ พิกัดเริ่มต้นของต้นฉบับที่จะทำการคัดลอก โดย 0,0 หมายถึง มุมบนซ้าย

x2,y2 คือ พิกัดเริ่มต้นของเป้าหมาย โดย 0,0 หมายถึง มุมบนซ้าย

width คือ ขนาดความกว้างของภาพที่ต้องการคัดลอก

height คือ ขนาดความสูงของภาพที่ต้องการคัดลอก

ทั้ง width และ height ไม่จำเป็นต้องเท่ากับขนาดของต้นฉบับ หรือ เป้าหมายก็ได้ แต่ขึ้นอยู่กับความต้องการว่าต้องการจะคัดลอกมาด้วยขนาดเท่าใด

โดยในโปรแกรมนี้จะใช้

blit (output,screen,0,0,0,0,SCREEN_W,SCREEN_H);

โดย output คือ ส่วนแสดงภาพสำรองที่ใช้เตรียมภาพที่จะนำมาแสดง

screen คือ หน่วยความจำการแสดงภาพ (video memory) ซึ่งระบบจะแสดงสิ่งต่าง ๆ ที่อยู่ในหน่วยความจำการแสดงภาพ (video memory)

0,0,0,0 คือ เริ่มคัดลอกจากตำแหน่งที่ 0,0 ของต้นฉบับ มายังตำแหน่ง 0,0 ของเป้าหมาย

SCREEN_W,SCREEN_H คือ ขนาดความกว้าง และความสูงที่ต้องการคัดลอก

ซึ่งหลังจากจบการทำงานของฟังก์ชัน blit แล้ว ภาพที่เราต้องการก็จะปรากฏอยู่บนหน้าจอ

บทที่ 6
การแก้ไขและเตรียมข้อมูลสำหรับการแสดงภาพครั้งต่อไป

6.1 หน้าที่

ในส่วนนี้จะแบ่งการทำงานได้เป็น 4 ขั้นตอนใหญ่ ๆ คือ

6.2 การตรวจสอบสภาวะของออปเจกต์เพื่อพิจารณาการเลิกใช้อาร์เรย์ของออปเจกต์

6.3 การเขียนค่าข้อมูลของฉากกลับคืนสู่ที่เดิม (ณ. ตำแหน่งที่ถูกตัวละครทับ)

6.4 การแก้ไขค่าของแอตทริบิวต์ที่เกี่ยวกับการแสดงค่าทางตำแหน่งของตัวละครบนฉาก

6.5 การหาเลขที่ของเฟรมของรูปตัวละครที่จะแสดงในครั้งต่อไป ซึ่งจะต้องเหมาะสมกับอิริยาบทของตัวละคร และแสดงถึงความต่อเนื่องของอิริยาบทที่กำลังแสดงอยู่

6.6 การทำงานกับการซ้อนกันของตัวละคร

6.2 การพิจารณาออปเจกต์เพื่อการเลิกใช้อาร์เรย์ของออปเจกต์

ในส่วนนี้จะมีการตรวจสอบค่าในแอตทริบิวต์ต่าง ๆ ของออปเจกต์เพื่อเป็นเกณฑ์ในการพิจารณา โดยที่สาเหตุของการเลิกใช้อาร์เรย์ของออปเจกต์ประเภท sprite ได้กล่าวไว้แล้วข้างต้น (บทที่ 4) จึงสรุปคร่าว ๆ ดังนี้

- ตำแหน่งของ sprite อยู่นอก 3 หน้าจอ
- ตาย
- เกิดการชน (สำหรับบางชนิดของ sprite)

ขอเพิ่มเติมในกรณีพิเศษสำหรับห้องที่ไม่เคยถูกใช้มาก่อนของอาร์เรย์ เนื่องจากถ้าสภาวะข้อมูลที่ถูกเก็บอยู่ในอาร์เรย์ไม่ตรงกับ 3 กรณีที่กล่าวข้างต้น จะหมายถึงการที่อาร์เรย์ห้องนั้นได้ถูกใช้งาน (Alive_Sprite = TRUE) จึงมีโอกาสเป็นไปได้ที่ข้อมูลในห้องที่ไม่เคยถูกใช้มาก่อนของอาร์เรย์ไม่ตรงกับกรณีที่ทั้ง 3 และจะถูกกำหนดว่ากำลังถูกใช้งาน ทำให้เกิดข้อผิดพลาดขึ้นได้ การป้องกันมีแนวคิดคร่าว ๆ คือการแยกห้องเหล่านี้ออกจากห้องที่ได้เคยถูกใช้งาน เมื่อพิจารณาความแตกต่างของห้องทั้งสองกลุ่ม จะพบว่ากลุ่มของห้องที่ยังไม่เคยถูกใช้งานมาก่อนจะมีแอตทริบิวต์ name เป็น 0 (หรือ BLANK) จึงใช้จุดนี้ตรวจสอบได้

กระบวนการในการเลิกใช้อาร์เรย์ของ sprite ประกอบด้วย

6.2.1 การเลิกใช้อาร์เรย์การกระโดด

6.2.2 การให้ค่า FALSE แก่ Alive_Sprite[] ณ. index เดียวกัน

6.2.3 เพื่อความแน่นอนที่มากขึ้นจึงกำหนดให้ค่า x_on_map มีค่าเป็นความกว้างของ sprite ที่ติดลบ เป็นการรับรองว่า sprite นี้จะไม่มีทางอยู่บนหน้าจอใดได้อีก หรือเป็นการรับรองว่าข้อมูลในห้องนี้จะไม่ถูกนำมาใช้อีก จนกว่าจะมี sprite อื่นนำข้อมูลใหม่มาลงให้

สำหรับสาเหตุของการเลิกใช้อาร์เรย์ของออปเจกต์ประเภท block มีเพียงกรณีที่ตำแหน่งของ block อยู่นอก 3 หน้าจอเท่านั้น การพิจารณากรณีพิเศษของห้องที่ไม่เคยใช้มาก่อนใช้หลักการเดียวกับ sprite แต่เกณฑ์ในการแยกกลุ่มของห้อง คือ ห้องที่ไม่เคยถูกใช้มาก่อนจะเก็บค่าในแอตทริบิวต์ w (ความกว้าง) และ h (ความสูง) เป็น 0 ทั้งคู่

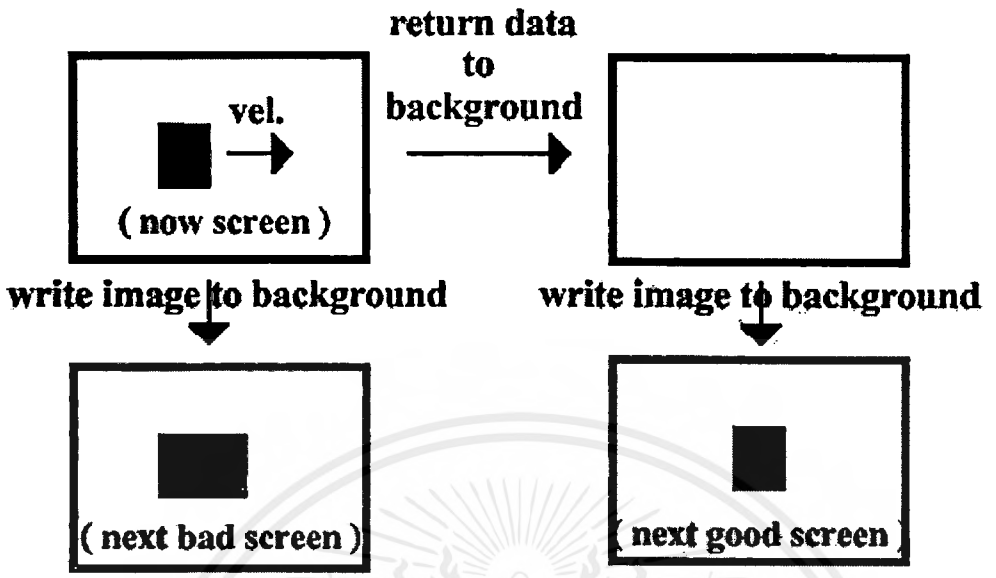
กระบวนการในการเลิกใช้ห้องของอาร์เรย์ทำเพียง การให้ค่า FALSE แก่ Alive_Block[] ที่ index เดียวกัน

6.3 การคืนข้อมูลของฉาก

เป็นการคัดลอกข้อมูลที่ถูกเก็บอยู่ในหน่วยความจำประจำของออปเจกต์กลับคืนสู่หน่วยความจำแสดงภาพสำรอง ซึ่งข้อมูลนี้ได้ถูกคัดลอกมาเก็บไว้ เมื่อตอนเรียกใช้ฟังก์ชัน Show_Sprite() ในการเขียนรูปลงบนหน่วยความจำแสดงภาพสำรอง การเขียนข้อมูลให้กลับคืนเหมือนเดิมจะเป็นการป้องกันไม่ให้คงเหลือภาพของตัวละครไว้ ณ. ตำแหน่งที่ได้แสดงออกหน้าจอไปแล้ว เมื่อตัวละครได้ไปสู่ตำแหน่งใหม่ เนื่องจากการแสดงภาพบนหน้าจอเป็นเพียงการนำข้อมูลของ index สีที่อยู่ในหน่วยความจำแสดงภาพ (video memory) ออกแสดงเท่านั้น ถ้ามีข้อมูลที่ผิดอยู่บนหน่วยความจำ การแสดงภาพก็จะผิดตามไปด้วย

การทำงานตรงนี้ และการเก็บข้อมูลของฉากในฟังก์ชัน Show_Sprite() อาจเป็นการสูญเปล่า ในกรณีที่มีการเขียนข้อมูลของฉากของหน้าจอปัจจุบันลงสู่หน่วยความจำแสดงภาพสำรองทุกครั้งของการวนซ้ำ (loop) แต่ตัวระบบก็ยังคงทำงานผ่านกระบวนการเหล่านี้เพื่อป้องกันการผิดพลาดจากการเขียนข้อมูลของฉาก และในกรณีที่ฉากมีขนาดเพียงหน้าจอเดียวการทำงานด้วยกระบวนการนี้ โดยปราศจากการเขียนซ้ำของฉาก ก็จะเป็นวิธีที่เร็วกว่า

สำหรับการเก็บ และคืนข้อมูลของฉาก ณ. ตำแหน่งที่ตัวละครจะทับนี้ จะทำกับเฉพาะตัวละครที่อยู่ภายในหน้าจอเดียวกับตัวละครของผู้เล่นเท่านั้น (ค่า x อยู่ระหว่าง 0 ถึง 320 และ ค่า y อยู่ระหว่าง 0 ถึง 200) เพราะ ถ้าให้ตัวละครทุกตัวที่อยู่บนทั้ง 3 หน้าจอได้เข้าสู่กระบวนการนี้อาจทำให้เกิดการผิดพลาดได้ เนื่องจากมีการติดต่อกับหน่วยความจำที่อยู่ภายนอกของหน่วยความจำการแสดงภาพสำรอง



รูป 6-1 การทำงานกับหน่วยความจำการแสดงภาพสำรอง

6.4 การแก้ไขค่าของแอตทริบิวต์

ในส่วนนี้จะมีการแก้ไขค่าในแอตทริบิวต์ส่วนใหญ่ของ sprite ดังนี้

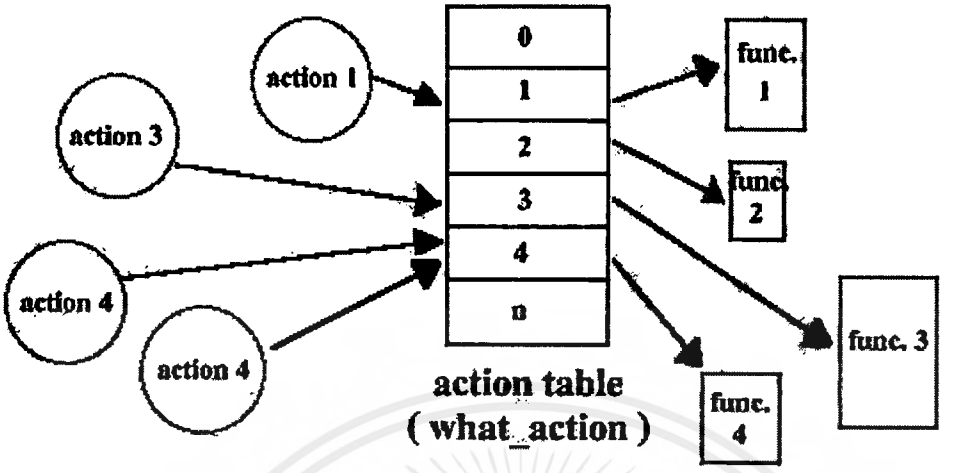
6.4.1 ตำแหน่งของตัวละครบนฉาก : x_on_map และ y_on_map

6.4.2 ข้อมูลเกี่ยวกับอิริยาบทของตัวละคร : what_action , in_action , step_action , x_vel , y_vel และ have_shot

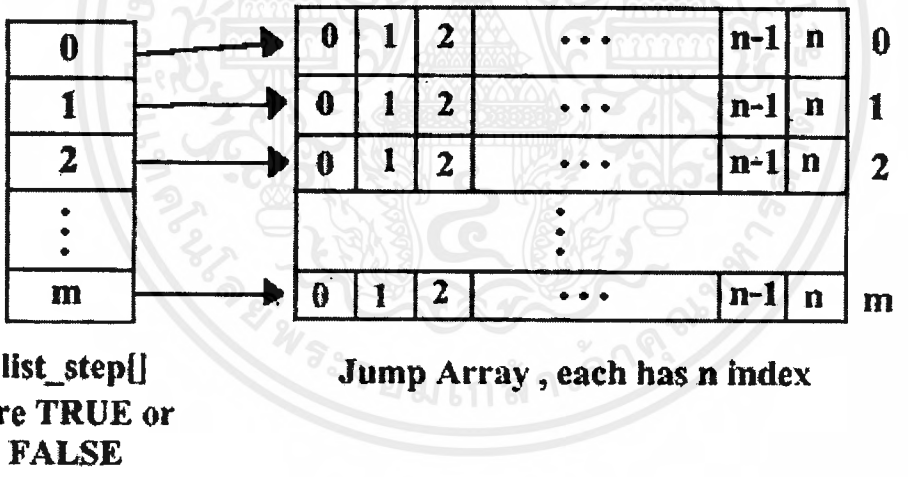
เปรียบได้กับส่วนที่จะทำให้ความต้องการในการที่จะแสดงอิริยาบทของแต่ละตัวละครสัมฤทธิ์ผล โดยจะมีฟังก์ชันของแต่ละอิริยาบท (what_action) ซึ่งมีกระบวนการทำงานที่แตกต่างกันไป คอยสนับสนุนให้เกิดการกระทำตามอิริยาบทที่ตัวละครต้องการ และแนวคิดที่ว่า ตัวละครที่ต่างกันสามารถใช้อิริยาบทเดียวกันได้

ตัวอย่างฟังก์ชันอิริยาบท Jump()

มีโครงสร้างการจัดเก็บข้อมูลคล้ายกับการทำงานกับอาร์เรย์ของออปเจกต์ คือ มีการจองอาร์เรย์ที่ใช้ในการกระโดดจำนวน m อาร์เรย์ โดยที่แต่ละอาร์เรย์จะมีทั้งหมด n index (ในระบบใช้ 30 index ต่อ 1 อาร์เรย์) ซึ่งอาร์เรย์เหล่านี้จะถูกแบ่งกันใช้โดยตัวละครต่าง ๆ ที่แสดงการกระโดด ในขณะที่ตัวละครหนึ่ง ๆ อยู่ในระหว่างการกระโดด อาร์เรย์ 1 อาร์เรย์จะต้องถูกใช้ไปในการเก็บข้อมูลของการกระโดดเสมอ ฉะนั้นในเวลาหนึ่ง ๆ สามารถมีตัวละครที่อยู่ในระหว่างการกระโดดพร้อมกันได้เป็นจำนวน m index ตามที่ได้จองไว้ นอกจากนี้มีการอาศัยอาร์เรย์อีกชุดหนึ่งที่จองไว้ให้มีจำนวน index เท่ากับจำนวนอาร์เรย์การกระโดดทั้งหมด (m index) เพื่อคอยบันทึกการใช้งานของอาร์เรย์การกระโดดแต่ละอาร์เรย์ ซึ่งทำหน้าที่เช่นเดียวกับ Alive_Sprite[] และ Alive_Block[] ดังที่ได้กล่าวไปแล้ว



รูป 6-2 โครงสร้างการทำงานตามอิริยาบทของแต่ละออปเจกต์



รูป 6-3 โครงสร้างของการเก็บข้อมูลการกระโดด

มีการกำหนดการใช้งานแต่ละห้อง (slot) ในอาร์เรย์การกระโดด ดังนี้

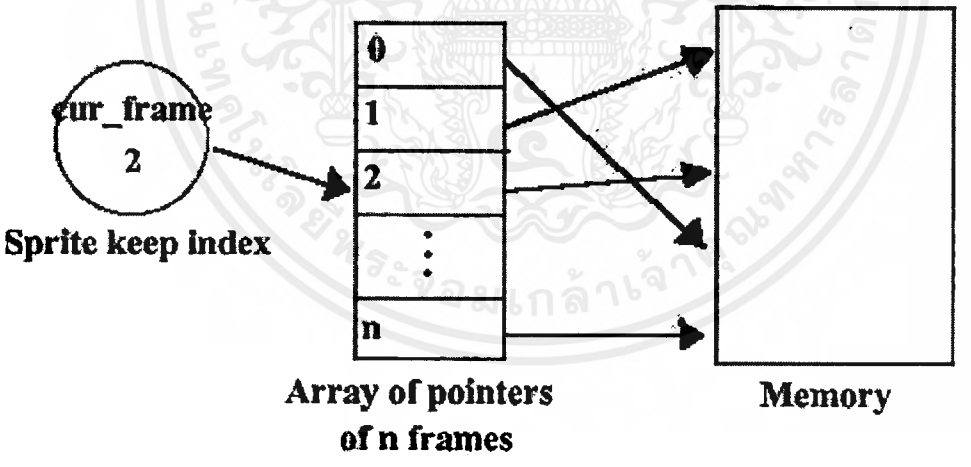
- ห้องที่ 0 เก็บจำนวนห้องทั้งหมดที่ใช้จริงในอาร์เรย์นั้น ๆ (เลขที่ของห้องสุดท้ายที่ใช้ บวก 1)
- ห้องที่ 1 เก็บความเร็วแนวแกน x ก่อนกระโดดของตัวละคร เพื่อใช้ในการคำนวณความสามารถในการเคลื่อนที่แนวแกน x ของตัวละครขณะที่กระโดด
- ห้องที่ 2 เก็บตำแหน่งในแนวแกน y ของตัวละครก่อนกระโดด เพื่อใช้ในการควบ-คุมการกระโดดให้กลับมาสู่ที่ตำแหน่งเดิมในแนวแกน y

- ห้องที่ 3 เป็นต้นไป เก็บความเร็วแนวแกน y ที่จะให้ตัวละครเคลื่อนที่ขณะกระโดด โดยที่ 1 ห้อง จะเก็บความเร็วของ 1 ขั้นตอนการกระโดด (1 slot : 1 step of jumping)

การทำงานของการกระโดด จะอาศัยแอตทริบิวต์หนึ่งของ sprite คือ step_action เป็นตัวแสดงว่าปัจจุบันอยู่ในขั้นตอนใดของการกระโดด และ ควรจะมีการเคลื่อนที่ในแนวแกน y เป็นระยะทางเท่าใด การกระโดดจะเสร็จสิ้นเมื่อได้ทำครบทุกขั้นตอนการกระโดด (step_action = จำนวนห้องทั้งหมดที่ใช้ซึ่งเก็บในห้องที่ 0) หรือเกิดการชนระหว่างตัวละครกับสิ่งอื่น ๆ เช่น block , ขอบจอ เป็นต้น

6.5 การเตรียมรูปที่เหมาะสม

เนื่องจากโครงสร้างของ sprite มีการทำงานกับชุดรูปของตัวละคร โดยแบ่งเป็น 1 รูปต่อ 1 เฟรม ต่อ 1 ตัวชี้ (pointer) ที่จะชี้ไปยังตำแหน่งบนหน่วยความจำที่เก็บรูปนั้นอยู่ การเลือกแสดงภาพจึงเป็นการเลือกว่าจะใช้ตัวชี้ (pointer) ใด นอกจากนี้ยังมีแอตทริบิวต์หนึ่ง คือ cur_frame ที่จะเก็บเลขเฟรมที่แสดงอยู่ในปัจจุบัน จากโครงสร้างนี้ แต่ละ sprite จะมีชุดของรูปเป็นของมันเอง ซึ่งแต่ละรูปอาจถูกตัวละครหลายตัวใช้ก็ได้



รูป 6-4 โครงสร้างการทำงานกับเฟรมภาพของ sprite

การเลือกรูปที่เหมาะสมนั้น คือการกำหนดรูปที่ตรงกับอิริยาบท และรูปที่ต่อเนื่องจากรูปในปัจจุบัน ในกรณีที่เป็นอิริยาบทเดียวกัน ซึ่งสามารถตรวจดูอิริยาบทของตัวละครได้จากแอตทริบิวต์ what_action ใน sprite , ตรวจดูรูปที่แสดงในปัจจุบันจากแอตทริบิวต์ cur_frame ม , ตรวจดูลำดับขั้นปัจจุบันของอิริยาบทได้จากแอตทริบิวต์ step_action และ ตรวจดูทิศทางการหันหน้าของตัวละครจากแอตทริบิวต์ direction หลังจากผ่านกระบวนการพิจารณาแล้ว (โดยมากใช้ IF-ELSE ในการตัดสินใจ)

นำเลขเฟรมที่จะแสดงเก็บไว้ในแอตทริบิวต์ cur_frame และให้ค่าที่ถูกต้องสำหรับทิศทางการหันหน้าของตัวละครแก่ direction

หลังจากสามารถกำหนดรูปที่จะนำแสดงได้แล้ว จะนำรูปดังกล่าวมาผ่านการหาขนาดของรูปตัวละครจริง ๆ ที่อยู่ในรูปนั้น ๆ (การกำหนดค่า T_dx , T_w , T_dy และ T_h ของรูป) และเก็บค่าขนาดของรูปตัวละครจริง ๆ รูปเก่าลงใน PT_dx , PT_w , PT_dy และ PT_h เพื่อใช้ในการพิจารณาเกี่ยวกับการชน ซึ่งจะได้กล่าวโดยละเอียดในบทถัดไป ตามด้วยการเปลี่ยนแปลงค่าของตำแหน่งในแนวแกน y ให้เหมาะสมกับรูปของตัวละครจริง ๆ ที่ได้เปลี่ยนไป ซึ่งก็จะกล่าวโดยละเอียดในบทถัดไปเช่นกัน

6.6 การทำงานกับการซ้อนกันของตัวละคร

จากการทำงานเกี่ยวกับการซ้อนกันโดยการเรียกใช้ฟังก์ชัน Show_Sprite() กับตัวละครที่จะถูกทับก่อน แล้วจึงกระทำกับตัวละครที่จะทับ ดังที่ได้กล่าวมาแล้วในบทที่ 7 ทำให้ข้อมูลของฉากที่ถูกเก็บโดยตัวละครที่ทับไม่ถูกต้อง คือจะได้ข้อมูลของรูปของตัวละครที่ถูกทับปนมาด้วย ในขณะที่ตัวละครที่ถูกทับจะสามารถเก็บข้อมูลของฉากได้ถูกต้อง ฉะนั้นการคืนข้อมูลให้กับฉากของส่วนนี้ จะทำงานสลับกับตอนที่เก็บข้อมูล โดยตัวละครที่ทับจะต้องคืนข้อมูลก่อน แล้วจึงให้ตัวละครที่ถูกทับ คืนข้อมูลเพื่อทับข้อมูลของหน่วยความจำแสดงภาพสำรอง ในบริเวณที่เคยซ้อนกันให้ถูกต้อง

บทที่ 7
การตรวจสอบการชนกันของออปเจกต์
(collision detectoin)

7.1 หน้าที่

ทำการตรวจสอบการชนกัน (collision detection) ซึ่งจะต้องทำการตรวจสอบทุก ๆ กรณีของการชนกัน เช่น การชนกันของศัตรูทั้งหมด การชนกันของตัวละครที่ผู้เล่นบังคับและศัตรู การชนกันระหว่างตัวละครทั้งหมดกับฉาก หรือแม้กระทั่งการถูกยิงหรือถูกชก ก็ใช้การตรวจสอบการชนกัน (collision detection) ทั้งสิ้น โดยใน โครงงานนี้ทำการตรวจสอบการชนกัน (collision detection) อยู่ 2 ลักษณะคือ การตรวจสอบการชนกันของขอบของออปเจกต์ (bounding collision detection) และ การตรวจสอบการชนกันของออปเจกต์จริง ๆ (pixel collision detection)

7.2 การตรวจสอบการชนกันของขอบของออปเจกต์ (bounding collision detection)

เนื่องจากภาพของออปเจกต์ต่าง ๆ จะมีลักษณะดังนี้ คือ มีกรอบและมีตัวของออปเจกต์จริง ๆ อยู่ภายในกรอบนั้น ๆ ดังรูป

รูป 7-1 แสดงตัวอย่างออปเจกต์

จะเห็นว่าระหว่างขอบของออปเจกต์และตัวของออปเจกต์จริง ๆ นั้นมีช่องว่างอยู่มาก ดังนั้นเพื่อความเหมือนจริงจึงต้องแบ่งการตรวจสอบการชนกัน (collision detection) ออกเป็น 2 แบบ ดังที่กล่าวมาข้างต้น

การตรวจสอบการชนกันของขอบของออปเจกต์ (bounding collision detection) นั้นจะมีขั้นตอนในการทำดังนี้

7.2.1 ตรวจสอบว่าออปเจกต์ทั้ง 2 ออปเจกต์ที่ต้องการตรวจสอบนั้นมีอยู่จริง โดยตรวจสอบดังนี้

ขอบซ้ายของออปเจกต์ <= ขอบขวาของออปเจกต์

ขอบบนของออปเจกต์ <= ขอบล่างของออปเจกต์

7.2.2 หาขอบเขตของออปเจกต์ทั้ง 2 ออปเจกต์ ที่จะทำการตรวจสอบ

7.2.3 ขั้นตอนต่อไปจะใช้วิธี ตรวจสอบโดยสมมุติว่าไม่มีการชนกันคือ

ถ้า ขอบซ้ายของออปเจกต์ 1 > ขอบขวาของออปเจกต์ 2 หรือ

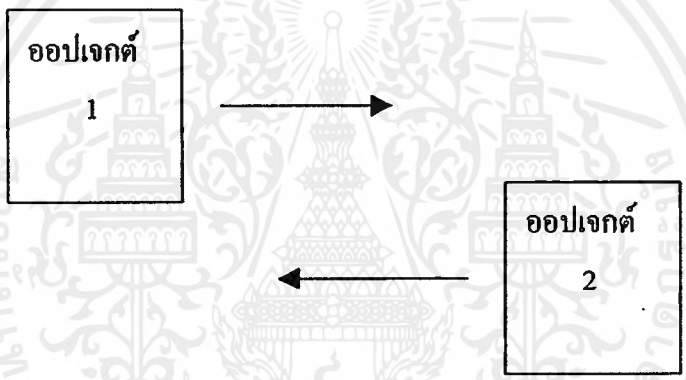
ถ้า ขอบซ้ายของออปเจกต์ 2 > ขอบขวาของออปเจกต์ 1 หรือ

ถ้า ขอบบนของออปเจกต์ 1 > ขอบล่างของออปเจกต์ 2 หรือ

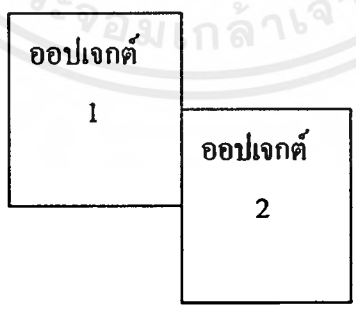
ถ้า ขอบบนของออปเจกต์ 2 > ขอบล่างของออปเจกต์ 1

แสดงว่าไม่มีการชนกัน ให้ คืนค่า false

7.2.4 ถ้าเกิดกรณีนอกเหนือจากนี้ถือว่ามีการชนกัน



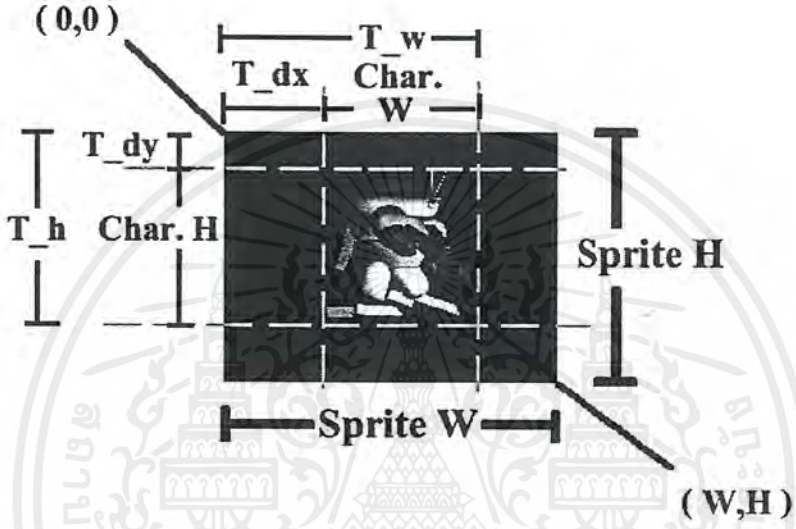
รูป 7-2 แสดงก่อนเกิดการชนกันของออปเจกต์ 2 ออปเจกต์



รูป 7-3 แสดงการเกิดการชนกันของออปเจกต์ 2 ออปเจกต์

7.3 การเพิ่มประสิทธิภาพให้กับ Function Collision

ได้มีการพัฒนากระบวนการทำงานของการตรวจสอบการชน ให้สามารถทำงานกับกรอบของรูปตัวละครจริง ๆ และยังให้สามารถตรวจสอบการชนกันของรูปตัวละครจริง ๆ (เดิมจะทำงานกับกรอบของรูป ของออปเจกต์) ซึ่งทำให้เกิดความสมจริงมากยิ่งขึ้น



รูป 7-4 แสดงกรอบต่าง ๆ ในออปเจกต์

จากรูปในส่วนที่เป็นสีดำจะไม่ถูกแสดงเมื่อมีการแสดงภาพ (transparent) ซึ่งส่วนนี้จะมีสีเป็น index ที่ 0 ในชุดสีทั้ง 256 สี (** Function transparent_bit_blt() **)

ค่าของ T_dx ได้จาก ระยะห่างจากด้านซ้ายมือของออปเจกต์ (จุด x = 0) จนถึง จุดที่ใกล้ที่สุดในแนวแกน x ของออปเจกต์ที่มี index สีไม่เป็น 0

ค่าของ T_w ได้จาก ระยะห่างจากด้านซ้ายมือของออปเจกต์ จนถึง จุดที่ไกลที่สุดในแนวแกน x ของออปเจกต์ที่มี index สีไม่เป็น 0

ค่าของ T_dy ได้จาก ระยะห่างจากด้านบนของออปเจกต์ (จุด y = 0) จนถึงจุดที่ใกล้ที่สุดในแนวแกน y ของออปเจกต์ที่มี index สีไม่เป็น 0

ค่าของ T_h ได้จาก ระยะห่างจากด้านบนของออปเจกต์จนถึงจุดที่ไกลที่สุดในแนวแกน y ของออปเจกต์ที่มี index สีไม่เป็น 0

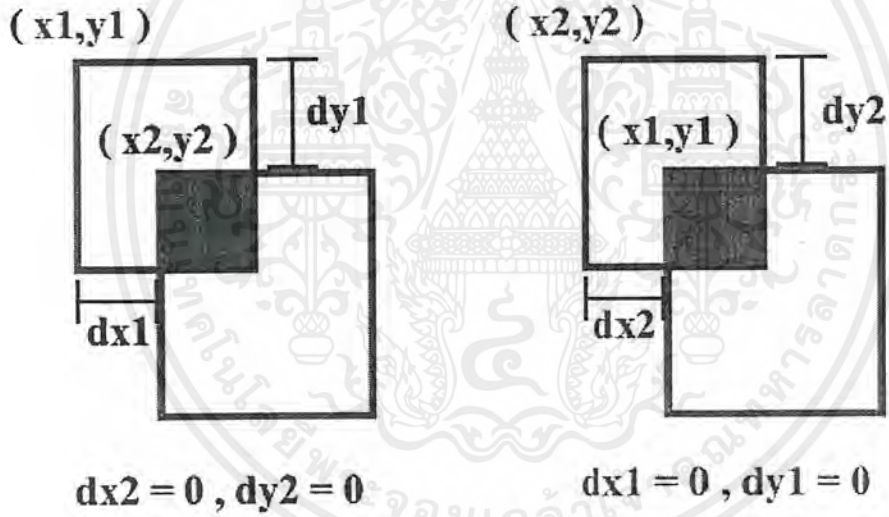
(** การหาค่า T_dx , T_w , T_dy , T_h ทำโดย Function Set_Truth_Dim() **)

เมื่ออยู่บนหน้าจอ จากจุด x จนถึง x + w และจากจุด y จนถึง y + h จะเป็นกรอบของ ออปเจกต์ โดยที่ จากจุด x + T_dx จนถึง x + T_w และจากจุด y + T_dy จนถึง y + T_h จะเป็นกรอบของรูปตัวละครจริง ๆ

วิธีการตรวจสอบการชนยังคงใช้แนวคิดเดิม คือ ใช้กรอบในการพิจารณา (bounding collision detection) นอกจากนี้ได้เพิ่มโปรแกรมในส่วนที่จะหน้าที่ตรวจสอบการชนกันของรูปตัวละครจริง ๆ (pixel collision detection) โดยมีวิธีการในการตรวจสอบ ดังนี้

7.3.1 จะทำเมื่อเกิดการชนกันของกรอบเสียก่อน เพื่อเป็นการประหยัดเวลา เพราะโปรแกรมส่วนนี้ใช้วิธีการที่ค่อนข้างจะกินเวลาพอสมควร

7.3.2 การพิจารณาจะทำเพียงในกรอบที่เกิดขึ้นจากการซ้อนกันของออปเจกต์ เมื่อชนกัน โดยทำการตรวจสอบว่า ณ. จุดเดียวกันบนหน้าจอ เมื่อเทียบเป็นจุดบนแต่ละออปเจกต์แล้ว มีสีของ index ไม่เป็น 0 ทั้งคู่หรือไม่ ถ้าไม่เป็นทั้งคู่ แสดงว่าเกิดการชนกันของตัวรูปจริง ๆ แต่ถ้าเป็นกรณีอื่นก็จะไปทำการตรวจสอบที่จุดต่อไป บนหน้าจอที่อยู่ภายในกรอบที่พิจารณาจนครบทุกจุด และถ้าทุกจุดไม่เกิดการชนจึงสรุปได้ว่ายังไม่เกิดการชน ของรูปตัวละครจริง ๆ เช่น



รูป 7-5 แสดง แนวคิดในการพิจารณา Pixel Collision

จากรูป จะจับคู่เปรียบเทียบระหว่างจุด (x1 + dx1 + n , y1 + dy1 + m) ของรูปออปเจกต์ ที่ 1 บนหน้าจอ กับจุด (x2 + n , y2 + m) ของรูปออปเจกต์ที่ 2 บนหน้าจอ ซึ่งเป็นจุดเดียวกันบนหน้าจอ และเมื่อเทียบเป็นจุดบน ออปเจกต์ แล้ว จะเป็นการเปรียบเทียบระหว่างจุด (dx1 + n , dy1 + m) ของออปเจกต์ที่ 1 กับ (n , m) ของออปเจกต์ที่ 2

เมื่อ 0 < n < ความกว้างของออปเจกต์ที่ 1 (w) - dx1 และ

0 < m < ความสูงของออปเจกต์ที่ 1 (h) - dy1

(** ทำโดย Function Chk_Pixel_Collide() **)

7.3.3 การทำงานหลังจากตรวจพบการชนกันของออปเจกต์

หลังจากที่ตรวจพบว่าเกิดการชนกันขึ้นแล้ว จะมีหลักการพิจารณาเพื่อทำงานต่อไปดังนี้

7.3.3.1 กรณีที่เกิดการชนกันของ sprite กับ sprite และออปเจกต์ใด ออปเจกต์ หนึ่ง หรือทั้งสองออปเจกต์ อนุญาตให้ออปเจกต์ทับได้ (attribute allow_other_over เป็น TRUE) จะจัดการให้ทั้งสองออปเจกต์ซ้อนกันไปได้

7.3.3.2 กรณีที่เกิดการชนกันของ sprite กับ sprite และไม่มีออปเจกต์ใดที่อนุญาตให้ออปเจกต์อื่นทับได้ จะจัดให้ทั้งสองออปเจกต์อยู่ติดกันพอดีในตำแหน่งที่เหมาะสม

(** Function Set_Smooth_Collide_Pix2() **)

7.3.3.3 กรณีที่เกิดการชนกันของ sprite กับ block จะจัดให้ sprite อยู่ติดกับ block พอดีในตำแหน่งที่เหมาะสม

(** Function Set_Smooth_Collide_B_Block() **)

7.3.4 หลักการพิจารณาหาตำแหน่งที่เหมาะสม ใน Function Set_Smooth_Collide_Pix2()

เนื่องจากเป็นการชนกันระหว่าง sprite ซึ่งอาจเคลื่อนมาชนกันด้วยความเร็วมากกว่า 0 ของทั้งคู่ หรือออปเจกต์หนึ่งมีความเร็วมากกว่า 0 และอีกออปเจกต์หนึ่งเป็น 0 จึงใช้ความเร็วในการพิจารณา โดยให้ออปเจกต์ที่มีความเร็วสูงกว่าเป็นฝ่ายปรับตำแหน่ง ขณะที่ออปเจกต์ที่มีความเร็วน้อยกว่าเป็นจุดฐาน ฉะนั้นจุดมุ่งหมายของ Function นี้จึงเป็นการสรุปและกำหนดค่าว่าจะให้ออปเจกต์ที่มีความเร็วสูงกว่าอยู่ทางด้านบน , ด้านล่าง , ด้านซ้าย หรือด้านขวาของออปเจกต์ที่มีความเร็วต่ำกว่า ซึ่งสรุปกระบวนการตัดสินใจได้ดังนี้

7.3.4.1 ถ้าออปเจกต์ที่มีความเร็วสูงกว่าได้เคลื่อนที่ไปทางขวา (x_vel > 0) ให้อยู่ทางซ้ายของออปเจกต์ที่มีความเร็วต่ำกว่า

7.3.4.2 ถ้าเคลื่อนที่ไปทางซ้าย (x_vel < 0) ให้อยู่ทางขวา

7.3.4.3 ถ้าเคลื่อนที่ไปทางด้านบน (y_vel < 0) ให้อยู่ทางด้านล่าง

7.3.4.4 ถ้าเคลื่อนที่ไปทางด้านล่าง (y_vel > 0) ให้อยู่ทางด้านบน

ทั้งนี้อาจเกิดกรณีที่การตัดสินใจไม่ตรงกับความเป็นจริงได้ เช่น ความเร็วเป็น 0 ทั้งคู่ แต่เกิดการชนกัน เนื่องจากรูปเปลี่ยนไป (ขนาดของออปเจกต์เท่าเดิม แต่ขนาดของรูปจริง ๆ ที่อยู่ในออปเจกต์เปลี่ยนไป) เป็นต้น

7.3.5 หลักการพิจารณาหาตำแหน่งที่เหมาะสมใน Function Set_Smooth_Collide_ B_Block()

สำหรับการชนกันระหว่าง sprite กับ block จะให้ sprite เป็นฝ่ายที่จะต้องปรับตำแหน่งให้อยู่ติดกับ block พอดี เนื่องจาก block จะไม่มีความเร็ว จุดประสงค์ของ Function จะเป็นไปในลักษณะเดียวกับ Function Set_Smooth_Collide_Pix2() คือ สรุปและกำหนดค่าให้ sprite อยู่ด้านใดของ block โดยใช้ตำแหน่งในอดีตของ sprite เป็นหลักในการพิจารณา ต่างกับ Function Set_Smooth_Collide_Pix2() ที่

ใช้ความเร็วเป็นหลักในการพิจารณา ทั้งนี้ตำแหน่งที่ใช้เป็นตำแหน่งของรูปตัวละครจริง ๆ โดยมีกระบวนการตัดสินใจดังนี้

7.3.5.1 ถ้าในอดีต ตำแหน่งในแกน y ของ sprite < ตำแหน่งในแกน y ของ block จะให้ sprite อยู่เหนือ block

7.3.5.2 ถ้าในอดีต ตำแหน่งในแกน y ของ sprite > ตำแหน่งในแกน y ของ block จะให้ sprite อยู่ใต้ block

7.3.5.3 ถ้าในอดีต ตำแหน่งในแกน x ของ sprite < ตำแหน่งในแกน x ของ block จะให้ sprite อยู่ทางซ้าย

7.3.5.4 ถ้าในอดีต ตำแหน่งในแกน x ของ sprite > ตำแหน่งในแกน x ของ block จะให้ sprite อยู่ทางขวา

โดยจะเลือกทำอย่างใดอย่างหนึ่ง ต่างกับ Function Set_Smooth_Collide_Pix2() ที่อาจทำทั้งกรณีของ x_vel และ y_vel

7.3.6 การหาตำแหน่งในอดีตของรูปตัวละครจริง ๆ

ซึ่งการหาตำแหน่งในอดีตนั้นจำเป็นที่จะต้องใช้ attribute PT_dx , PT_w , PT_dy และ PT_h ที่จะใช้เก็บค่าในอดีตของ T_dx , T_w , T_dy และ T_h ตามลำดับ เพราะว่าการเปลี่ยนรูปของตัวละครมีผลต่อตำแหน่งของรูปตัวละครจริง ๆ โดย

7.3.6.1 ตำแหน่งล่างสุดของรูปในอดีต คำนวณได้จาก

y ของออปเจกต์ในปัจจุบัน - y_vel + PT_h + T_h - PT_h

สาเหตุที่ต้องบวก T_h และ ลบ PT_h เพราะว่า ขนาดของรูปของตัวละครจริง ๆ มีการเปลี่ยนแปลงที่บ่อย และขนาดของรูปนี้ส่งผลต่อเหตุการณ์ต่างๆ ที่มีผลต่อตัวละคร เช่น ถ้าตำแหน่งล่างสุดของรูปอยู่สูงกว่าเดิม ก็จะเข้าสู่สถานะของการที่ตัวละครกำลังลอยตัวอยู่กลางอากาศ เป็นต้น เพื่อเป็นการป้องกัน จึงให้ตำแหน่งล่างสุดของรูปของตัวละครจริง ๆ คงอยู่ที่เดิม ฉะนั้นค่า y ของออปเจกต์มีโอกาสเปลี่ยนแปลงได้จากสมการ y ของออปเจกต์หลังการปรับแต่ง = y ของออปเจกต์ก่อนการปรับแต่ง + PT_h - T_h

7.3.6.2 ตำแหน่งบนสุดของรูปในอดีต คำนวณได้จาก

y ของออปเจกต์ในปัจจุบัน - y_vel + PT_dy + T_h - PT_h

7.3.6.3 ตำแหน่งซ้ายสุดของรูปในอดีต คำนวณได้จาก

x ของออปเจกต์ในปัจจุบัน - x_vel + PT_dx

7.3.6.4 ตำแหน่งขวาสุดของรูปในอดีต คำนวณได้จาก

x ของออปเจกต์ในปัจจุบัน - x_vel + PT_w

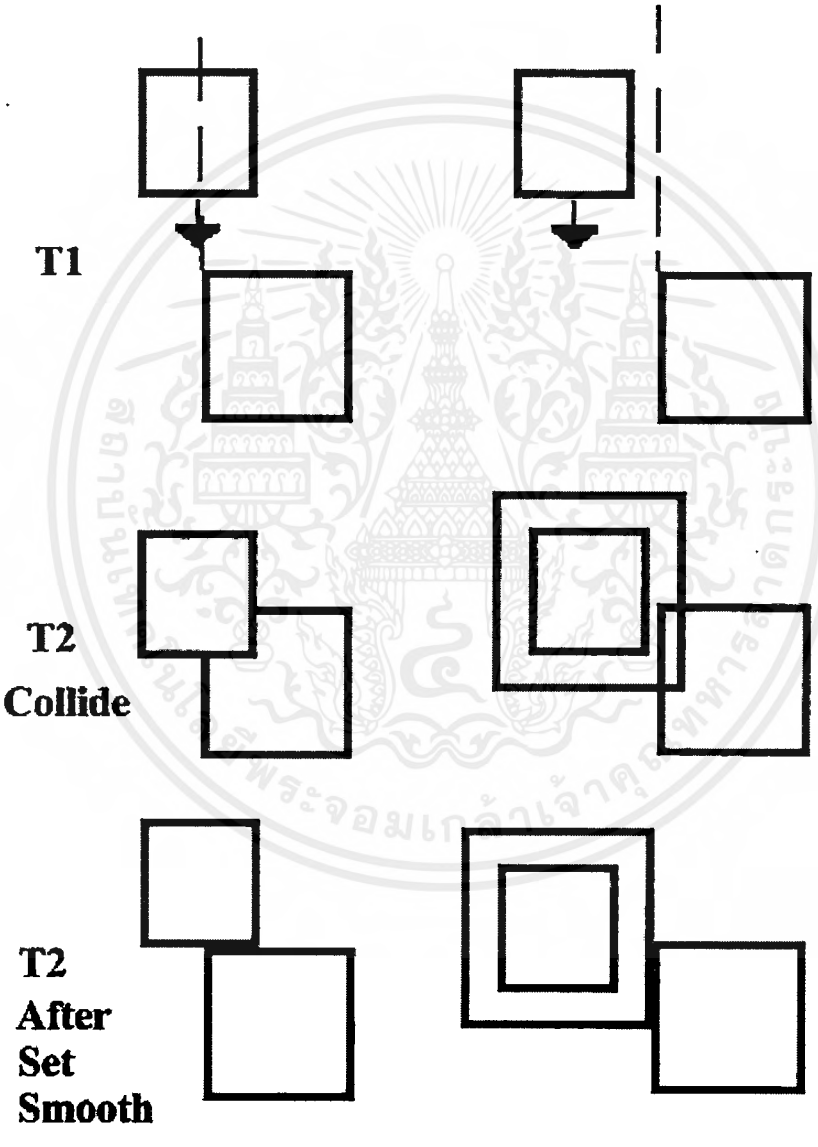
7.3.7 กรณีพิเศษเพิ่มเติมที่ใช้ในการตัดสิน

เนื่องจากรูปมีโอกาสเปลี่ยนแปลงตลอดเวลา จึงมีโอกาสที่ตัวละครได้เดินพ้น block และกำลังอยู่ในสถานะการตกจากที่สูง (y_vel > 0) แล้วรูปเกิดเปลี่ยนขนาดทำให้เกิดการชนกับ block

โดยที่ค่าตำแหน่งต่ำสุดของรูปในอดีตอยู่เหนือ block นั้น ซึ่งตามหลักความเป็นจริงแล้วตัวละครสมควรที่จะตกต่อไป การพิจารณาจึงต้องเพิ่มกรณีนี้เข้าไป

7.3.7.1 ถ้าในอดีต ตัวละครอยู่ในขอบเขตของ block ให้อยู่เหนือ block

7.3.7.2 ถ้าในอดีต ตัวละครอยู่นอกขอบเขตของ block ให้อยู่ด้านข้างของ block



รูป 7-6 แสดงการเกิดขึ้นของกรณีพิเศษ

บทที่ 8
การจัดการกับแอตทริบิวต์ที่เกี่ยวกับหน้าจอของตัวละคร และการจัดการในด้านอื่น ๆ

8.1 หน้าที่

8.2 การปรับแต่งตำแหน่งบนหน้าจอของตัวละครของผู้เล่นให้เหมาะสม

8.3 กำหนดค่าในแอตทริบิวต์ที่เกี่ยวข้องกับหน้าจอของทั้ง sprite และ block ได้แก่ x และ y ซึ่งแสดงถึงการที่สามารถปรากฏตัวบนหน้าจอเดียวกับตัวละครของผู้เล่นได้

8.4 การหาตำแหน่งรูปของฉากที่สอดคล้องกับตำแหน่งของตัวละครของผู้เล่นบน

8.5 การจัดการในด้านอื่น ๆ

อาจกล่าวโดยสรุปได้ว่าเป็นการทำงานกับทุกสิ่งให้สอดคล้องกับตำแหน่งปัจจุบันของตัวละครของผู้เล่นบนหน้าจอ

8.2 การปรับตำแหน่งบนหน้าจอของตัวละครของผู้เล่น

ในระบบนี้ยึดหลักที่ว่าตัวละครของผู้เล่นจะต้องอยู่ที่กึ่งกลางของหน้าจอเสมอ ยกเว้นในหน้าจอแรกของฉาก ที่ผู้เล่นมีสิทธิเคลื่อนที่บนหน้าจอได้ทางซีกซ้ายของหน้าจอ และ ในหน้าจอสุดท้ายที่จะสามารถเคลื่อนที่บนหน้าจอทางซีกขวาของหน้าจอ โดยตำแหน่งบนหน้าจอนี้ (x และ y) จะต้องสัมพันธ์กับตำแหน่งบนฉาก (x_on_map และ y_on_map)

สำหรับกรณีที่ตัวละคร ไม่ได้อยู่ในหน้าจอแรก และหน้าจอสุดท้ายของฉาก มีความสัมพันธ์ ดังนี้

x = x_on_map – (จำนวนหน้าจอ * ความกว้างของหน้าจอ)

สำหรับกรณีที่ตัวละครของผู้เล่นมาถึงกึ่งกลางหน้าจอเป็นครั้งแรก จะทำการคำนวณค่า x ให้กึ่งกลางของออปเจกต์อยู่กึ่งกลางของหน้าจอพอดี แต่เนื่องจากค่า x ได้หยุดการขึ้นอยู่กับ x_on_map เป็นครั้งแรก จึงจำเป็นที่จะต้องให้ x_on_map ขึ้นอยู่กับค่า x แทนเพื่อรักษาความสัมพันธ์ของทั้งสองแอตทริบิวต์ให้คงอยู่ ดังสมการ

x = ((ความกว้างของหน้าจอ – ความกว้างของออปเจกต์ของผู้เล่น) /2) – 1

และ x_on_map = (จำนวนหน้าจอ * ความกว้างของหน้าจอ) + x

หลังจากการมาถึงกึ่งกลางจอครั้งแรก ค่าของ x จะไม่ถูกเปลี่ยนแปลงแก้ไขอีก จนกว่าจะถึงกึ่งกลางของหน้าจอสุดท้าย ในขณะที่ค่าของ x_on_map จะถูกเปลี่ยนแปลงต่อไปเพื่อแสดงถึงการเคลื่อนไหวของผู้เล่นบนฉาก

8.3 การกำหนดค่าของแอตทริบิวต์ในออปเจกต์อื่น ๆ

การกำหนดให้ตัวละครของผู้เล่นอยู่กึ่งกลางหน้าจอ ทำให้การเคลื่อนที่ทุกอย่างที่เห็นบนหน้าจอมีการทำงานคล้ายกับทุกสิ่งเคลื่อนมาหาตัวละครของผู้เล่น แต่โดยจริง ๆ แล้วการแสดงรูปของออปเจกต์ต่าง ๆ เกิดขึ้นจากความสัมพันธ์ที่ว่าตำแหน่งของผู้เล่นบนฉากได้เข้าใกล้ตำแหน่งของออปเจกต์นั้นบนฉาก หรืออาจกล่าวได้ว่าสิทธิที่ออปเจกต์อื่น ๆ จะมีโอกาสได้ถูกนำขึ้นแสดงบนหน้าจอนั้น เกิดจากตำแหน่งของผู้เล่น เมื่อ x คือตำแหน่งของออปเจกต์บนหน้าจอและออปเจกต์ที่สามารถนำขึ้นแสดงบนหน้าจอได้ จะต้องมีค่าใน x อยู่ในกรอบของหน้าจอ (ในที่นี้ค่าของ x จะอยู่ระหว่าง 0 และ 320 , ค่าของ y อยู่ระหว่าง 0 และ 200) ฉะนั้นค่า x และ y ของออปเจกต์อื่น ๆ จึงขึ้นอยู่กับตำแหน่งของออปเจกต์บนฉาก , ตำแหน่งของผู้เล่นบนฉาก และตำแหน่งของผู้เล่นบนหน้าจอ สรุปเป็นสมการได้ดังนี้

x ของออปเจกต์ = x_on_map ของออปเจกต์ - x_on_map ของผู้เล่น + x ของผู้เล่น

สำหรับค่าของ y ก็สามารถใช้สมการข้างต้นนี้ แต่ในที่นี้จะไม่กล่าวถึง เพราะตัวระบบยังไม่อนุญาตให้มีการเคลื่อนที่ผ่านหน้าจอในแนวแกน y

ออปเจกต์ประเภท sprite จะมีความสามารถในการกำหนดให้ sprite นั้น ๆ สามารถหลุดออกจากหน้าจอ หรือไม่ให้หลุดจากหน้าจอก็ได้ (แอตทริบิวต์ d_scr) โดยจะตรวจสอบการหลุดจากหน้าจอจากค่าของแอตทริบิวต์ x และ y ร่วมกับความกว้าง และความสูงของรูปของตัวละครจริง ๆ การป้องกันจะเป็นการไม่อนุญาตให้ส่วนหนึ่งส่วนใดของตัวละครได้หลุดออกไปจากหน้าจอ ฉะนั้นถ้าตรวจพบว่า x + T_dx < 0 หรือ x + T_w > ความกว้างของหน้าจอ หรือ y + T_dy < 0 หรือ y + T_h > ความสูงของหน้าจอ ก็จะต้องกำหนดค่าตำแหน่งของ sprite นั้นใหม่ให้อยู่ชิดกรอบหน้าจอพอดี ถ้าไม่ระบุให้ป้องกันก็จะผ่านกระบวนการนี้ไปสำหรับ sprite นั้น ๆ (**ฟังก์ชัน Detect_Edge() **)

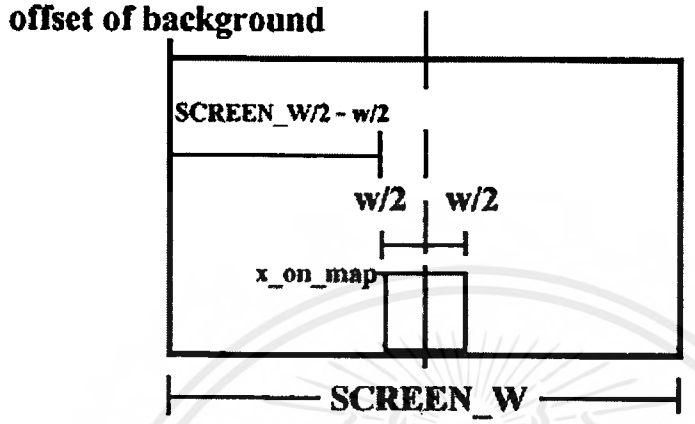
8.4 การหาตำแหน่งของฉาก

ฉากที่จะนำแสดงขึ้นบนหน้าจอก็เป็นอีกสิ่งหนึ่งที่ขึ้นอยู่กับตำแหน่งของตัวละครของผู้เล่น โดยส่วนนี้จะเป็นการคำนวณหาค่าของ offset ของหน่วยความจำที่เก็บรูปของฉากทั้งหมด ที่ถูกต้องกับตำแหน่งของผู้เล่น ด้วยสมการ

offset = x_on_map ของผู้เล่น – (((ความกว้างของฉาก – ความกว้างของออปเจกต์) / 2) – 1)

ซึ่งก็คือการหาค่าตำแหน่งในแนวแกน x ของจุดในฉาก ที่อยู่ทางซ้ายของกึ่งกลางตัวละครของผู้เล่น ในระยะห่างเท่ากับครึ่งหน้าจอ ซึ่งจะมีกรณีพิเศษ คือ ในหน้าจอแรกที่ตัวละครผู้เล่นสามารถเคลื่อนไหวบนหน้าจอซีกซ้ายได้ ทำให้ค่า offset ติดลบ จึงแก้ด้วยการกำหนดค่าให้เป็น 0 แทน และ ในหน้าจอสุดท้าย ตัวละครของผู้เล่นสามารถเคลื่อนที่บนหน้าจอซีกขวาได้ ทำให้จากจุด offset ไปเป็นขนาดเท่ากับขนาดของหน้าจอ ซึ่งเป็นบริเวณที่จะถูกคัดลอกไปใส่ในหน่วยความจำแสดงภาพสำรอง ได้เกินขอบเขต

ของรูปของฉากที่เตรียมไว้ ทำให้เกิดข้อผิดพลาด จึงแก้ด้วยการกำหนดให้ offset หยุดการเพิ่มขึ้น ณ. ค่าที่ทำให้อยู่ห่างจากตำแหน่งสุดท้ายของฉากเท่ากับขนาดของหน้าจอ



รูป 8-1 แสดงแนวคิดในการคำนวณค่าของ offset

8.5 การจัดการในด้านอื่น ๆ

8.5.1 แรงดึงดูดของพื้นโลก (gravity)

เนื่องจากเกมส์ที่เขียนขึ้นได้สมมุติฉากให้อยู่บนพื้นโลก ดังนั้นจึงต้องมีการกำหนดแรงดึงดูดของพื้นโลกให้กับทุกตัวละคร เพื่อความเหมือนจริง โดยแต่ละตัวละครจะมีแอททริบิวต์สำหรับกำหนดค่าแรงดึงดูดของพื้นโลกคือ T_sprite.gravity หากตัวละครใดต้องการให้อยู่ภายใต้แรงดึงดูดของโลกก็จะถูกกำหนดค่าให้เป็น TRUE แต่ถ้าไม่ต้องการก็กำหนดค่าให้เป็น FALSE

เมื่อตัวละครใดถูกกำหนดแอททริบิวต์แรงดึงดูดให้เป็น TRUE แล้วจะต้องยืนอยู่บนพื้นเท่านั้น ไม่สามารถจะบังคับให้ลอยตัวอยู่ในอากาศได้ เช่น ตัวละคร อัศวิน ที่ถูกบังคับโดยผู้เล่นจะไม่สามารถกดปุ่มลูกศรขึ้นได้ เนื่องจากถูกกำหนดแอททริบิวต์นี้ให้เป็น TRUE และตัวละครฝ่ายศัตรูหากกำหนดแอตทริบิวต์ นี้ให้เป็น TRUE แล้ว หากเดินผ่านหลุมที่มีขนาดกว้างกว่าขนาดของตัวมันก็จะทำให้มันตกหลุมตามแรงดึงดูดของโลก

8.5.2 การลดความเร็วลงเมื่อปะทะกับแรงเฉื่อย

จากการที่ระบบของเกมส์นี้ทำงานเป็นรอบการวนซ้ำ ดังนั้นตัวละครฝ่ายพระเอกที่ถูกบังคับด้วยผู้เล่นจึงจำเป็นต้องมีการลดความเร็วลงตอนท้ายของรอบการวนซ้ำ (loop) เพื่อให้ความเร็วลดลงเหมือนกับมีแรงเฉื่อย มิฉะนั้นจะทำให้ตัวละครมีความเร็วคงที่ และไม่หยุดเคลื่อนที่ถึงแม้ปุ่มบังคับจะถูกปล่อยแล้วก็ตาม

สำหรับการลดความเร็วนี้จะตรวจสอบเฉพาะตัวละครพระเอกเท่านั้น (เนื่องจากตัวละครฝ่ายศัตรูจะมีฟังก์ชันการตัดสินใจของแต่ละตัวอยู่แล้ว) โดยตรวจสอบว่าขณะนั้นความเร็วเป็นศูนย์

หรือไม่ถ้าไม่ก็ทำการลดความเร็วลงไป 2 หน่วย ซึ่งหากผู้เล่นกดปุ่มบังคับการเคลื่อนที่ค้างไว้ ในรอบการทำงานจะมีการเพิ่มความเร็วอยู่ 2 ที่ดังนั้นหากกดปุ่มค้างไว้ถึงแม้จะถูกลดความเร็ว ความเร็วรวมก็จะไม่ลดลง

8.5.3 การลดความเร็วให้เท่ากับศูนย์

นอกเหนือจากการลดความเร็วลงครั้งละ 2 หน่วย เพื่อแสดงแรงเฉื่อยแล้ว ยังมีบางกรณีที่จำเป็นต้องลดความเร็วของตัวละครทั้งหมดให้กลายเป็นศูนย์ด้วย เช่น กรณีที่ตัวละครถูกตรวจจับว่ามีการชนกันก็จะทำให้ความเร็วที่มีอยู่ลดลงเป็นศูนย์ได้ หรือการตกจากที่สูงก็จะกำหนดให้ความเร็วของตัวละครนั้นมีความเร็วเป็นศูนย์เช่นกัน

บทที่ 9
บทสรุป และวิจารณ์

โครงงานชิ้นนี้ได้ทำขึ้นมาเพื่อศึกษา การเขียนเกมส์ และศึกษาการใช้ปัญญาประดิษฐ์ใน โปรแกรม ซึ่งนำมาประยุกต์ใช้ในเกมส์ โดยแต่ละตัวละครจะมีการใส่ปัญญาประดิษฐ์ซึ่งเป็นเอกลักษณ์ ของแต่ละตัวละครนั้น ๆ โดยจะแยกออกเป็นไฟล์ .h เช่น slime.h , rameng.h , ball.h เป็นต้น ภายใน ไฟล์ .h นั้นจะประกอบด้วย

- การเตรียมข้อมูลภาพของตัวละครนั้นทั้งหมดมาเก็บไว้ในตัวแปร
- การเตรียมภาพของอิริยาบทต่าง ๆ ของตัวละครนั้น ๆ
- สมองของตัวละครนั้น ๆ ที่จะเป็นตัวเลือกอิริยาบทต่าง ๆ ที่เหมาะสม

เมื่อสมองได้เลือกอิริยาบทของตัวละครนั้น ๆ ได้แล้วจะถูกส่งออกมาในตัวแปรชื่อ T_sprite-> what_action หลังจากนั้นก็จะเตรียมภาพที่จะแสดงอิริยาบทนั้นต่อไป

การที่เรามีไฟล์ .h เป็นของแต่ละตัวละครนั้น มีข้อดีคือ ทำให้เราสามารถเพิ่ม ,ลด หรือเปลี่ยน แปลงตัวละครได้ง่าย โดยทำการแก้ไขที่ไฟล์ .h ของแต่ละตัวละครนั้น ๆ ซึ่งไม่ต้องเกี่ยวข้องกับตัวละคร ตัวอื่น

ภายในเกมส์จะมีการตรวจสอบการชนกันของตัวละครทั้งหมด ทั้งการชนกันของตัวละครด้วยกัน เอง และระหว่างตัวละครกับ block ซึ่งก็จะมีข้อกำหนดแตกต่างกันไปตามที่ได้กำหนดไว้ เช่น ถ้าพระเอก ชนกับศัตรู กำหนดให้พระเอกต้องบาดเจ็บ เป็นต้น

สำหรับการสร้างฉากนั้น ได้ใช้โปรแกรมวาดภาพทั่วไป เช่น Paint Shop Pro 4 ซึ่งมีเครื่องมือที่ สะดวก และที่สำคัญยังสามารถสร้างฉากที่มีความยาวไม่จำกัดได้ (หลาย ๆ หน้าจอรวมอยู่ใน 1 ไฟล์) แต่ ข้อเสียของ Paint Shop Pro 4 ก็มีคือ ไม่สามารถ Transparent ภาพได้ทำให้ความสามารถในการสร้างฉาก ถูกจำกัด

ส่วนการสร้างตัวละคร ได้ใช้โปรแกรม 3D STUDIO MAX ซึ่งเป็นโปรแกรมสำหรับวาดภาพ 3 มิติโดยเฉพาะ ทำให้การสร้างตัวละครให้เหมือนจริงนั้นมีความสะดวกและรวดเร็วยิ่งขึ้น

ข้อดีของระบบ

- เนื่องจากมองตัวละครต่าง ๆ เป็นออปเจกต์ จึงสะดวกในการเพิ่มและเปลี่ยนแปลงราย ละเอียดของแต่ละตัวละคร ผ่านทางไฟล์ .h ดังที่ได้กล่าวมาแล้วข้างต้น
- ขณะนี้การเก็บรูปภาพของเฟรมต่าง ๆ ของตัวละครเป็นการเก็บที่แยก 1 ไฟล์ต่อ 1 ภาพ ทำ ให้สะดวกต่อการแก้ไขข้อผิดพลาดเฉพาะรูปได้ แต่การเก็บแยกไฟล์เช่นนี้ ก็เป็นการเปิด โอกาสให้ผู้อื่นสามารถนำรูปไปแก้ไขดัดแปลงได้โดยง่ายเช่นกัน
- ตัวระบบสามารถรองรับขนาดของตัวละครที่หลากหลายได้ โดยที่ทุกเฟรมต้องมีขนาดเท่า กัน เพราะมองตัวละครเป็นออปเจกต์หนึ่งซึ่งย่อยต่อไปอีกไม่ได้แล้ว และการทำงานเรื่อง

การชนกันของตัวละคร ที่ใช้แนวคิดของ Pixel Collision ซึ่งทำให้การตรวจสอบการชนกันของตัวละครที่มีขนาดต่างกันมาก ๆ สามารถทำได้โดยละเอียด

- ความสามารถในการเพิ่มฉากซึ่งทำได้ไม่ยากโดย
  1. การวาดรูปของฉาก
  2. การนำข้อมูลของออปเจกต์ต่าง ๆ ที่อยู่ในฉาก มาเก็บในไฟล์ .fin ตามโครงสร้างที่กำหนด

ข้อเสียของระบบ

- ความสามารถของโปรแกรม ซึ่งสามารถทำงานได้เฉพาะกับ block ที่เป็นสี่เหลี่ยมเท่านั้น ยังไม่สามารถตรวจสอบ block ในรูปทรงอื่น ๆ เช่น ทรงกลมได้เลย แต่หากมีเวลาในการศึกษาค้นคว้าก็น่าจะสามารถทำได้เช่นกัน
- การเก็บข้อมูลของออปเจกต์ต่าง ๆ ในฉากยังคงไม่สะดวกเท่าใดนัก เนื่องจากในขณะนี้การใส่ข้อมูลยังคงแยกจากการวาดรูป ทำให้ผู้สร้างฉากต้องจดตำแหน่ง และ ขนาดของออปเจกต์ประเภท block มาเขียนใส่โปรแกรม แล้วจึงวิ่งโปรแกรมนั้นเพื่อเก็บข้อมูลในรูปของ binary code อีกทั้งการวางตำแหน่งของตัวละครบนฉากที่ไม่สามารถตรวจสอบความเหมาะสมได้ในขณะทำการกำหนดตำแหน่ง ฉะนั้นเพื่อความสะดวกในการเพิ่มฉาก จึงสมควรที่จะต้องมีการพัฒนา tool ที่สามารถวาดรูปพร้อมทั้งเขียนข้อมูลของออปเจกต์ลงไปในไฟล์ .fin ไปพร้อม ๆ กัน
- ในกรณีที่ออปเจกต์มีความเร็วมากกว่า ความหนาของออปเจกต์ที่จะถูกชนมาก ๆ จะทำให้ไม่สามารถตรวจสอบว่าเกิดการชนกันได้ เพราะการชนกันจะตรวจสอบได้ก็ต่อเมื่อส่วนหนึ่งส่วนใดของออปเจกต์ทั้งสองได้ซ้อนทับกันเท่านั้น แต่ในกรณีนี้จะไม่เกิดการซ้อนทับ ออปเจกต์ที่วิ่งเข้ามาด้านหน้า จะไปอยู่ด้านหลังของอีกออปเจกต์

เนื่องจากเกมส์ที่เขียนขึ้นมีระยะเวลาจำกัด ประกอบกับผู้จัดทำเพิ่งเริ่มศึกษาทางด้านนี้ ทำให้เกมส์ที่ทำขึ้นไม่สมบูรณ์ตามที่ตั้งใจไว้ตั้งแต่ตอนกำหนดเป้าหมายของโครงงาน เช่น อาจจะมีความตื่นเต้นน้อยไปบ้าง เป็นต้น แต่หากมีเวลาหลังจากนี้ผู้จัดทำจะศึกษาเพิ่มเติม และพัฒนาให้ดียิ่งขึ้น

นายปิติ เชิญถนอมวงศ์ 37014255
นายพรธเนศร์ แซ่โค้ว 37014278
ผู้จัดทำ

เอกสารอ้างอิง

[1] http://www.delorie.com/djgpp ณ วันที่ 5 ส.ค. 2540

[2] http://www.talula.demon.co.uk/allegro ณ วันที่ 5 ส.ค. 2540

[3] http://www.realtime.no/~hoie/ctut.htm ณ วันที่ 10 ส.ค. 2540

[4] http://www.realtime.no/~hoie/game-prog.htm ณ วันที่ 10 ส.ค. 2540

[5] http://www.asti.dost.gov.ph/~jay/gameprog/tutorial.html ณ วันที่ 20 ส.ค. 2540

[6] http://gpmega.home.ml.org ณ วันที่ 20 ส.ค. 2540

[7] http://www.ideasoftware.com/game_c.htm ณ วันที่ 20 ส.ค. 2540

[8] PC Game Programming Explorer / Dave Roberts,p. cm. ,ISBN 1-883577-07-1

ภาคผนวก ก
แนะนำตัวละครต่าง ๆ ในเกมส์

ตัวละครในเกมส์จะแบ่งได้เป็น 2 ฝ่าย คือ ตัวละครพระเอกซึ่งเป็นตัวละครที่ผู้เล่นบังคับ และตัวละครฝ่ายศัตรู ซึ่งจะมีคอมพิวเตอร์เป็นผู้กำหนดการแสดงอิริยาบทต่าง ๆ ให้ ในบทนี้จะแนะนำตัวละครทั้งหมดรวมทั้งอิริยาบททั้งหมดของตัวละคร

อัศวิน (KNIGHT)

อัศวิน เป็นตัวละครพระเอกที่ผู้เล่นเป็นผู้บังคับ ดังนั้นอิริยาบทต่าง ๆ จึงขึ้นอยู่กับผู้เล่นจะกำหนด ซึ่งอัศวินจะมีอิริยาบทต่าง ๆ ดังนี้

1. เดิน ซึ่งจะสามารถเดินได้ทั้ง ซ้าย และขวา ด้วยความเร็วสูงสุดที่ 8 หน่วย

รูป ก-1 แสดงการเดินของอัศวิน

2. กระโดด อัศวินจะสามารถกระโดดได้ นอกจากนี้อัศวินยังสามารถทำอิริยาบทฟันดาบพร้อมกับกระโดดได้อีกด้วย ดังรูป

รูป ก-2 แสดงการกระโดด

3. ฟันดาบ สำหรับการโจมตีศัตรูท่าพื้นฐานของ อัศวิน คือ การใช้ดาบฟันศัตรู ดังรูป

รูป ก-3 แสดงการฟันดาบ

4. การยิงบอลอัคคี เป็นการโจมตีขั้นสูงของ อัศวิน จะสามารถทำได้เมื่อเพิ่มระดับเป็นระดับ 2 หรือ 3

หยุ่น หยุ่น (slime)

หยุ่น ๆ เป็นศัตรูมีลักษณะเป็นก้อนกลมสีน้ำเงิน ซึ่งจะเป็นตัวละครฝ่ายศัตรูที่มีความสามารถด้อยที่สุด โดยถ้าอยู่ในสภาวะปกติ หยุ่น ๆ จะกระโดดอยู่กับที่ แต่ถ้าพระเอกเข้ามาอยู่ในรัศมีการโจมตีเมื่อไร หยุ่น ๆ จะกระโดดเข้าใส่ทำให้พระเอกบาดเจ็บได้

รูป ก-4 แสดงภาพหยุ่น ๆ

นักรบสีทอง (Big Richard)

เป็นศัตรูที่มีรูปร่างคล้ายหุ่นยนต์ โดยปกติจะเดินเฝ้ายามในคอบเขตของตนเอง แต่ถ้าพระเอกเดินเข้ามาอยู่ใกล้เกินไปก็จะทำการพุ่งเข้าชนทันที และที่สำคัญมันสามารถวิ่งหนีกระสุนของพระเอกได้ด้วย

รูป ก-5 แสดงภาพนักรบสีทอง

รูป ก-6 แสดงภาพการพุ่งเข้าชน

เวียดกงอันตราย (Rameng)

ศัตรูที่จำลองภาพมาจากนักรบจีน มีท่าไม้ตายคือ ท่าควงสว่าน (supersplash) ซึ่งจะทำให้พระเอกพลังชีวิตลดได้มาก รวมทั้งมีกระสุนที่มีพลังทำลายสูงโดยมันจะยิงกระสุนเมื่อพบว่าพระเอกกำลังบาดเจ็บอยู่

รูป ก-7 แสดงภาพยิงกระสุนของเวียดกง

รูป ก-8 แสดงภาพ ควงสว่านของเวียดกง

บอลพิฆาต (Ball Hunter)

เป็นศัตรูที่มีรูปร่างใหญ่ที่สุดในบรรดาศัตรูทั้งหมด สามารถหายตัวเพื่อหลบกระสุนของคู่ต่อสู้ได้ และยังมีท่าไม้ตายคือ บอลพิฆาต ซึ่งจะเป็นการยิงกระสุนออกมาเป็นชุด ๆ โดยนัดที่ 3 จะเป็นกระสุนขนาดใหญ่ ซึ่งจะหลบได้ยากมาก โดยปกติแล้วมันจะยืนอยู่เฉย ๆ แต่ถ้าคู่ต่อสู้เข้ามาในระยะที่มันมองเห็น มันจะกระโดดเพื่อเป็นการเตือน และถ้าพระเอกยังเข้ามาใกล้อีกก็จะยิงกระสุนเข้าใส่ทันที

รูป ก-9 แสดงท่าบอลพิฆาต

รูป ก-10 แสดงท่ากระโดดของบอลพิฆาต

ภาคผนวก ข
ฟังก์ชันต่าง ๆ ที่ใช้

1. ฟังก์ชันจาก library Allegro
   - allegro_init();
   - install_keyboard();
   - install_mouse();
   - install_timer();
   - fade_out(); ใช้เปลี่ยน pallete สีจากที่เป็นอยู่ เป็นสีดำ
   - set_gfx_mode(); ใช้กำหนด mode ของหน้าจอ
   - install_sound();
   - load_midi(); เตรียมข้อมูลของ file .mid ลงบน type MID*
   - play_midi();
   - clear_keybuf();
   - load_bitmap(); ทำการ decode การเก็บข้อมูลของรูป และเก็บค่าต่าง ๆ ของรูป ใน type BITMAP*
   - set_pallete();
   - blit(); คัดลอกข้อมูลผืนหนึ่งจากหน่วยความจำตำแหน่งหนึ่ง ไปลงในอีกตำแหน่งหนึ่ง
   - clear();
   - vsync(); หยุดโปรแกรมจนกว่าจะถึงช่วง vertical retrace ของจอ
   - load_sample(); เตรียมข้อมูลของ file .wav ลงบน type SAMPLE*
   - play_sample();
   - destroy_bitmap();
   - destroy_midi();
   - destroy_sample();
   - allegro_exit();

2. ฟังก์ชันที่สร้างขึ้นเอง แบ่งเป็นฟังก์ชันที่ถูกใช้โดยส่วนรวม และฟังก์ชันที่ถูกใช้เฉพาะออปเจกต์ ซึ่งโดยมากฟังก์ชันเฉพาะออปเจกต์จะถูกเก็บในไฟล์ .h ของออปเจกต์นั้น ๆ

   2.1 ฟังก์ชันที่ถูกใช้โดยส่วนรวม

      2.1.1 ฟังก์ชันใน spr_dj17.c

      - Chk_Colide_All_Screen(); ทำการตรวจสอบการชนกันของแต่ละออปเจกต์กับ ทุก ๆ ออปเจกต์ ทำการปรับตำแหน่งหลังการชนให้เหมาะสม และเปลี่ยนแปลงข้อมูลต่างๆ ที่มีผลมาจากการชน

- Chk_Keyboard(); ตรวจสอบข้อมูลที่ได้รับจาก keyboard ว่าได้กดปุ่มตามที่กำหนดไว้หรือไม่ ถ้าตรงจะทำการเตรียมค่าเบื้องต้นสำหรับอิริยาบถประจำปุ่มนั้น ๆ ของตัวละคร
- Detect_Edge(); ปรับตำแหน่งของตัวละครที่มี flag d_scr เป็น TRUE เพื่อกันไม่ให้ตัวละครนั้นหลุดออกจากหน้าจอ
- Load_One_Page(); เตรียมข้อมูลของออปเจกต์บนหน้าจอนั้น ๆ
- Now_On_Floor(); ตรวจสอบว่าตัวละครนั้นอยู่บน block หรือไม่
- Prepare_All_Image(); ทำการเตรียมรูปทั้งหมดที่จะใช้
- Prepare_Next_Sprite(); เตรียมข้อมูล , ภาพของตัวละครสำหรับการแสดงใน การวนซ้ำครั้งหน้า ให้เหมาะสมกับชนิดของตัวละคร , ลักษณะของอิริยาบถ และลักษณะในปัจจุบันของตัวละคร
- Recycle_List(); ตรวจสอบคุณสมบัติของแต่ละออปเจกต์ที่ใช้ห้องอยู่ ว่าสมควรคืนห้องหรือไม่
- Relse_Sprite(); แสดงการเลิกใช้ห้องนั้น ๆ ในออปเจกต์อาร์เรย์
- Scroll(); หาค่า offset ที่เหมาะสมบนรูปของฉาก ซึ่งเข้ากันได้กับตำแหน่งของผู้เล่นบนฉาก
- Search_Free_Index(); หา index ของห้องที่ว่างห้องแรกในออปเจกต์อาร์เรย์ ของ sprite
- Search_Free_Index_Block(); หา index ของห้องที่ว่างห้องแรกในออปเจกต์อาร์เรย์ ของ block
- Show_Sprite(); นำรูปของตัวละครมาเขียนลงหน่วยความจำแสดงภาพสำรอง

2.1.2ไฟล์ใน AI.h

- Find_Player(); ตรวจสอบตำแหน่งปัจจุบันของผู้เล่นว่าอยู่ทางด้านซ้าย หรือขวาของออปเจกต์นั้น ๆ
- Find_Player2(); คล้ายกับ Find_Player แต่แสดงตำแหน่งของผู้เล่น โดยแบ่งระยะห่างเป็นใกล้ และไกลด้วย
- Find_Player4(); แบ่งระยะห่างในแต่ละด้านเป็น 3 ระดับ

2.1.3ไฟล์ใน collide.h

- Check_Case_Over(); กำหนดค่าให้กับ flag now_over_other ในกรณีที่การชนกัน มี sprite ใด sprite หนึ่งที่มี flag allow_other_over เป็น TRUE
- Clear_Case_Over(); คืนค่า flag now_over_other ให้กลับเหมือนเดิม หลังจากที่ได้สิ้นสุดการซ้อนกันของคู่ออปเจกต์นั้น ๆห
- Chk_Colide(); ตรวจสอบว่าเกิด bounding collision ระหว่าง sprite หรือไม่
- Chk_Colide_With_Block(); ตรวจสอบว่าเกิด bounding collision ระหว่าง sprite กับ block หรือไม่

- Chk_Pixel_Colide(); ตรวจสอบว่าเกิด pixel collision ระหว่าง sprite หรือไม่
- Set_Smooth_Collide_B_Block(); คำนวณหาตำแหน่งที่เหมาะสมของ sprite หลังจากที่ชนกับ block
- Set_Smooth_Collide_Pix2(); คำนวณหาตำแหน่งที่เหมาะสมของ sprite หลังจากที่ชนกับอีก sprite

2.1.4 ไฟล์ใน ERR.h

- Any_Error(); แสดงว่าเกิดข้อผิดพลาดขึ้นบนตัวระบบ
- Not_Enough(); แสดงข้อผิดพลาดกรณีที่ทรัพยากรไม่พอ
- Wait_Space_Allegro(); คอยการกดปุ่ม space bar หรือ ESC

2.1.5 ไฟล์ใน Load_Fin.h

- Prepare_Block(); นำข้อมูลที่อ่านได้จากไฟล์มาเก็บใน block
- Prepare_Sprite(); นำข้อมูลที่อ่านได้จากไฟล์ มาตรวจสอบชนิดของตัวละคร แล้วจึง กำหนดค่าข้อมูลตามชนิดของตัวละคร

2.16 ไฟล์ใน Random.h

- Ran_dif(); สุ่มตัวเลขจาก 0 ถึงค่าสูงสุดที่กำหนดเข้ามา
- Ran_sign(); สุ่มเครื่องหมายเป็น บวก หรือ ลบ

2.17 ไฟล์ใน SH_IMG.h

- Chg_Icon(); เปลี่ยนภาพให้เหมาะสมกับตำแหน่งของ pointer ในการเลือกของผู้เล่นในปัจจุบัน เช่น เมื่อกดขวา ปุ่มทางขวาจะสว่างขึ้น ในขณะที่ปุ่มทางซ้ายจะมืดลง ทั้งนี้การได้กำหนดโครงสร้าง และออปเจกต์ List_Icon เพื่อให้ใช้งานได้หลากหลายมากขึ้น รายละเอียดของ List_Icon ประกอบด้วย
  - Amount_icon : จำนวนของภาพทั้งหมด หรือค่าสูงสุดที่จะให้ pointer ไปถึงได้
  - **link_on , **link_off : ชุดของภาพเมื่อตอนที่ pointer ชี้อยู่ และเมื่อตอนที่ไม่ได้ชี้อยู่ตามลำดับ มีจำนวนชุดเท่ากับ amount_icon
  - *x , *y : ชุดของตำแหน่งที่จะนำแต่ละชุดของภาพไปแสดงบนหน้าจอ

  ทั้งนี้จะต้องมีการกำหนดค่าเริ่มต้นให้กับออปเจกต์นี้ก่อนที่จะเรียกใช้ Chg_Icon() เช่น Init_Icon1()
- Return_Screen(); คืนข้อมูลให้แก่หน่วยความจำการแสดงภาพสำรอง หลังจากได้มีการแสดงออปเจกต์ MENU บนหน้าจอ
- set_gui_color(); เตรียมค่าสี

- Show_Data(); นำข้อมูลต่าง ๆ ออกแสดงบนหน้าจอ เช่น พลังชีวิตของผู้เล่น ข้อความแสดงการจบฉาก เป็นต้น
- transparent_bit_blt(); เขียนภาพลงบนหน่วยความจำที่ต้องการ ในลักษณะของการ transparent

2.1.8 ไฟล์ใน SPRITE.h

- Gravity(); ถ้าพบว่าตัวละครนั้นไม่ได้อยู่บนพื้น (on_floor = FALSE) และ flag gravity_on เป็น TRUE จะให้ตัวละครนั้นอยู่ในอิริยาบถของการตกจากที่สูง what_action = 4
- Jump(); เป็นการนำค่าเริ่มต้นที่กำหนดไว้ตามโครงสร้างของการเก็บข้อมูลการกระโดด มาใช้ในการกำหนดตำแหน่งใหม่ให้แก่ตัวละคร โดยที่ถ้าพบว่าเกิดการชนในขณะที่กระโดดจะหยุดการกระโดดทันที และเรียกให้มีการตรวจสอบการตกจากที่สูง
- Modify_Enemy_Data(); แก้ไขข้อมูลประจำตัวของศัตรูกรณีที่อาวุธของผู้เล่นชนกับศัตรู และจะมีการแก้ไขข้อมูลประจำตัวของผู้เล่นกรณีที่ศัตรูตาย โดยจะเป็นการเพิ่มค่า exp ของผู้เล่น และถ้าค่าดังกล่าวได้เกินระดับที่กำหนดไว้ ก็จะมีการแก้ไขค่าอื่น ๆ ด้วย เพื่อให้ผู้เล่นเก่งขึ้น
- Modify_Player_Data(); แก้ไขข้อมูลประจำตัวของผู้เล่น ซึ่งจะถูกเรียกกรณีที่เกิดการชนกันระหว่างผู้เล่นกับตัวละครอื่น ๆ ในกรณีที่ผู้เล่นชนกับอาวุธของศัตรู หรือสิ่งของที่อยากให้หายเมื่อผู้เล่นเดินมาชน ฟังก์ชันนี้จะกำหนด flag collide ของออปเจกต์นั้น ๆ เป็น TRUE เพื่อทำให้หายไปใน Recycle_List() หรือผู้เล่นเป็นผู้ทำให้เกิดการเปลี่ยนแปลงสถานะของตัวเอง เช่น การยิงกระสุนที่จะต้องเสียค่า mp เป็นต้น
- Read_Jump_Step(); อ่านข้อมูลจากห้องที่ออปเจกต์นั้น ๆ จองไว้
- Relse_Jump_Step(); คืนห้องของอาร์เรย์การกระโดดให้แก่ระบบ
- Set_Dead_Step(); หน่วงเวลาให้ยังมีการแสดงภาพของตัวละครอยู่สักพัก หลังจากที่ตัวละครนั้นตายจากการถูกโจมตี ใช้ step_action เป็นค่าเริ่มต้น แล้วการวนซ้ำจะลดค่า step_action ลงเรื่อย ๆ เมื่อเป็น 0 จึงคืนห้องของออปเจกต์อาร์เรย์ให้แก่ระบบ what_action = 6
- Search_Free_Jump_Step(); หาค่า index ของห้องที่ว่างห้องแรกในอาร์เรย์การกระโดด และทำการบันทึกเลขที่ของออปเจกต์เพื่อแสดงสิทธิในห้องนั้น
- Set_Jump_Step(); กำหนดค่าเริ่มต้นให้กับอาร์เรย์การกระโดด ตามโครงสร้างที่กำหนดไว้ what_action = 1
- Set_Receive_HP(); กำหนดค่า shield เพื่อให้เป็นอมตะชั่วครู่
- Set_Shot_Step(); กำหนดค่าเริ่มต้นสำหรับการยิงกระสุนทุกชนิดของทุกตัวละคร ใช้ what_action = 2

- Set_Truth_Dim(); ทำการหาค่า T_dx,T_dy,T_w,T_h และกำหนดค่าให้ PT_dx,PT_dy,PT_w,PT_h
- Set_Wound_Jump_Step(); กำหนดค่าเริ่มต้นสำหรับการกระโดดเนื่องจากการบาดเจ็บ โดยอาศัยตำแหน่งของอีกออปเจกต์ที่ชน ช่วยในการพิจารณาว่าจะกระโดดไปในทิศทางใด การกำหนดค่าการกระโดดยึดโครงสร้างเดียวกับการกระโดดทั่ว ๆ ไป what_action = 3 และกำหนดให้ flag wounded เป็น TRUE
- Write_Jump_Step(); ทำการเขียนข้อมูลลงในห้อง โดยกำหนดเลขที่ห้องผ่านเลขที่ของออปเจกต์

2.2 ฟังก์ชันที่ใช้เฉพาะตัวละคร ประกอบด้วยฟังก์ชันในลักษณะคล้าย ๆ กัน คือ

- Check_Not_Found..(); ตรวจสอบว่าไฟล์รูปต่าง ๆ ที่จะถูกใช้อยู่ครบหรือไม่
- Prepare_..(); จัดเก็บรูปลงบนตัวแปรที่เป็น type BITMAP*
- Init_Sprite_..(); กำหนดค่าเริ่มต้นต่าง ๆ ให้กับตัวละครนั้น ๆ ใน type sprite*
- Prepare_next_frame_..(); หาเลขที่เฟรมของรูปของตัวละครที่เหมาะสมสำหรับการแสดงครั้งต่อไปบนหน้าจอ
- AI_..(); กระบวนการตัดสินใจของตัวละครนั้น ๆ
- Destroy_..(); เรียกใช้ destroy_bitmap กับทุกเฟรมของตัวละคร
- ฟังก์ชันการแก้ไขค่าต่าง ๆ ของตัวละคร ตามอิริยาบถขณะนั้น

ภาคผนวก ค
รูปแบบของเกมส์

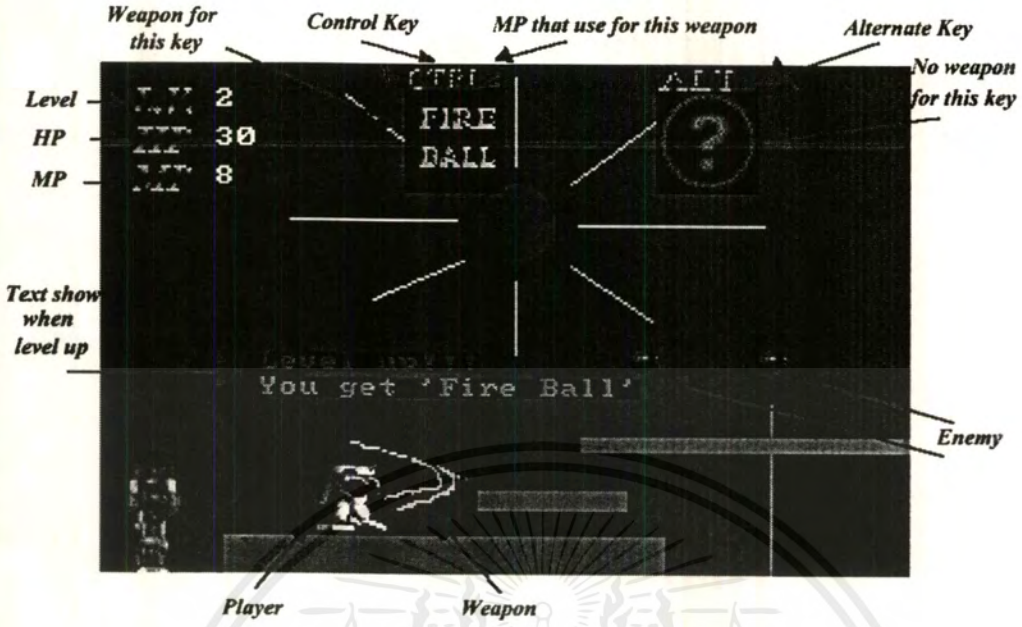
เกมส์ที่สร้างขึ้นนี้อยู่ในลักษณะของเกมส์ ACTION RPG คือ สามารถบังคับตัวละครได้อย่างอิสระ , มีการกระโดด , มีแรงดึงดูด , บังคับตัวละครที่ดูเหมือนมีชีวิต และมีการพัฒนาของตัวละคร ทำให้เกิดความน่าติดตามถึงความสามารถของตัวละครที่จะเพิ่มขึ้นเรื่อย ๆ เมื่อเล่นไปนาน ๆ

นอกจากข้อมูลทั่ว ๆ ไปของ sprite แล้ว ในแต่ละตัวละครจะมีการเก็บข้อมูลเฉพาะตัว เพื่อใช้ในการดำเนินไปของเกมส์ ดังนี้

- level : แสดงถึงระดับความสามารถของตัวละครในปัจจุบัน มีเพียงผู้เล่นเท่านั้นที่จะใช้ค่านี้
- hp : แสดงถึงพลังชีวิตของตัวละคร ถ้าค่านี้เป็น 0 แสดงว่าตัวละครนั้นตาย
- mp : แสดงถึงค่าพลังเวทมนตร์ที่ตัวละครนั้น ๆ มีอยู่ ในตัวละครประเภทอาวุธของผู้เล่น ค่านี้จะหมายถึง ค่าพลังเวทมนตร์ที่จะต้องใช้ไปเมื่อใช้อาวุธนั้น ๆ
- atk : หรือ attack เป็นค่าแสดงถึงความรุนแรงในการโจมตีของตัวละคร ยิ่งมีค่ามากก็จะยิ่งมีความรุนแรงมากตามไปด้วย
- def : หรือ defend แสดงถึงพลังป้องกัน ยิ่งมีมากก็จะยิ่งแข็งแกร่ง
- luc : หรือ lucky แสดงถึง โชคของตัวละคร ยิ่งมีมากก็จะยิ่งโชคดี
- shield : แสดงถึงความเป็นอมตะ ค่านี้โดยปกติจะเป็น 0 แต่จะถูกเพิ่มค่าเมื่อถูกโจมตี หรือเกิดเหตุการณ์บางอย่างที่ต้องการให้เป็นอมตะ ค่านี้ถ้าไม่เท่ากับ 0 จะถูกลดลงเรื่อย ๆ ทีละ 1 ต่อ 1 การวนซ้ำ และถ้าค่านี้เป็นเลขคี่ รูปของตัวละครจะไม่ถูกแสดง ทำให้เมื่อมีการวนซ้ำเรื่อย ๆ จะเห็นเป็นตัวละครนั้นกระพริบ
- view : ความสามารถในการมองเห็น ถูกใช้ในศัตรู เพื่อเป็นตัวกระตุ้นให้เกิดการตัดสินใจ ซึ่งศัตรูจะเริ่มตัดสินใจเมื่อผู้เล่นได้เข้ามาอยู่ในระยะ view ของศัตรูนั้น ๆ
- exp : หรือ experience เป็นค่าแสดงถึงความก้าวหน้าของตัวละคร ถ้าค่านี้เพิ่มถึงระดับหนึ่ง ก็จะมีการแก้ไขค่าอื่น ๆ เพื่อให้ตัวละครเก่งขึ้น (level up !!) สำหรับศัตรูค่านี้จะใช้เป็นค่าที่จะบวกเพิ่มให้กับ exp ของผู้เล่น เมื่อศัตรูถูกฆ่า

ปุ่มต่าง ๆ ที่ใช้

- "A" : การฟัน ซึ่งเป็นอาวุธพื้นฐาน ใช้ได้ไม่จำกัด
- "Space Bar" : กระโดด และ กดเมื่อต้องการให้เกมส์ดำเนินต่อไป
- "←, →, ↑, ↓" : ซ้าย , ขวา , ขึ้น , ลง ตามลำดับ
- "CTRL" : อาวุธพิเศษที่ 1
- "ALT" : อาวุธพิเศษที่ 2
- "F1" : ตารางแสดงค่าสถานะปัจจุบันของผู้เล่น
- "ESC" : ออกจากเกมส์



รูป ค-1 แสดงรายละเอียดของหน้าจอ