บทที่ 3

การออกแบบคลังโปรแกรมสนับสนุนการทำสไปรต์

ในบทนี้จะกล่าวถึงหลักการทำงานของระบบการทำภาพเคลื่อนไหวและการออกแบบส่วนประกอบที่สำคัญในการทำภาพเคลื่อนไหวของสไปรต์ได้แก่ โครงสร้างข้อมูลของสไปรต์ โครงสร้างข้อมูลของสภาพแวดล้อมที่เกี่ยวข้องในการจัดการสไปรต์ ขั้นตอนวิธีของตัวจักรในการทำภาพเคลื่อนไหว และการส่งข้อความ

หลักการทำงานของระบบจัดการทำภาพเคลื่อนไหว

จากหลักการพัฒนาโปรแกรมภายใต้วินโดวส์และการสร้างสไปรต์ในบทที่ผ่านมา สามารถนำมาประยุกต์ในการสร้างระบบจัดการภาพเคลื่อนไหวได้ โดยให้ระบบการทำงานของภาพเคลื่อนไหวมีหลักการทำงานดังนี้

1. สไปรต์ (Sprite)

- มีการสร้างสไปรต์จากบิตแมพที่เตรียมไว้แล้ว

- ควบคุมสไปรต์โดยกระบวนคำสั่ง (Procedure Animation Control) ทำให้สไปรต์จะมีการเคลื่อนไหวหรือดำเนินการใดๆตามเงื่อนไขที่ระบุไว้ในกระบวนคำสั่งเฉพาะตัวของสไปรต์นั้นๆ โดยใช้การทำงานเชิงข้อความหรือกระบวนคำสั่งโดยตรงก็ได้

2. สภาพแวดล้อมของระบบ (Environment)

- ดูแลภาพพื้นหลัง ภาพพื้นหลังอาจเป็นภาพที่สร้างโดยโปรแกรมประยุกต์หรืออ่านเข้ามาจากหน่วยความจำสำรองก็ได้

- จัดการสไปรต์ทั้งหมดเช่นการจัดลำดับความสำคัญของสไปรต์ในระบบ สอบถามจำนวนสไปรต์ ควบคุมการเข้าถึงตัวสไปรต์

3. ตัวจักรทำภาพเคลื่อนไหว (Animation Engine)

ใช้เทคนิค Offscreen Bitmap วิธีนี้ทำให้ไม่เกิดการกระพริบหรือลดการกระพริบของภาพบนจอได้ (Barkakati, Nabajyoti 1993) เนื่องจากเราใช้หน่วยความจำหลักส่วนหนึ่งให้ทำหน้าที่จำลองจอภาพ จากนั้นการเปลี่ยนแปลงภาพใดๆก็ตามจะทำที่หน่วยความจำส่วนนี้ก่อน เมื่อทำเสร็จแล้วจึงทำสำเนาไปแสดงบนจอภาพเพียงครั้งเดียว เรียกหน่วยความจำส่วนนี้ว่า hidden page ซึ่งการจัดสรร hidden page นี้ในวินโดวส์มีฟังก์ชันสนับสนุนให้แล้วได้แก่ฟังก์ชัน CreateCompatibleDC() และ CreateCompatibleBitmap() ดังตัวอย่างต่อไปนี้

```
HDC       hDCHiddenPage, hDC;
HBITMAP   hBmpHiddenPage;

hDC = GetDC(NULL);
hDCHiddenPage   = CreateCompatibleDC(hDC);
hBmpHiddenPage = CreateCompatibleBitmap(hDC, nWidth, nHeight);
SelectObject(hDCHiddenPage,  hBmpHiddenPage);
```

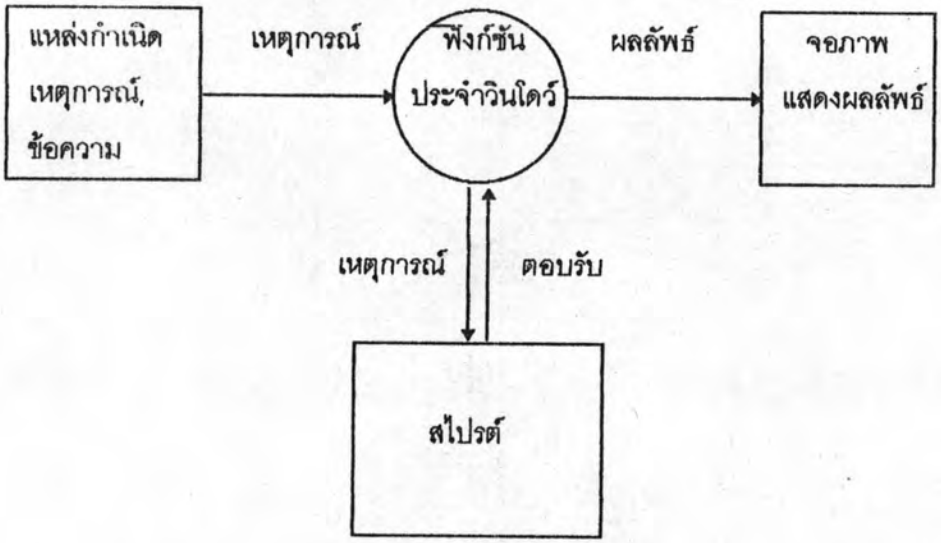
จากตัวอย่างนี้เราได้ hDCHiddenPage เป็น hidden page ขนาด nWidth x nHeight จุดภาพตามต้องการและนำไปใช้งานได้เช่นเดียวกับจอภาพ เมื่อจัดการกับ hidden page นี้เรียบร้อยแล้วก็ทำสำเนาไปยังจอภาพได้ทันที (ด้วยฟังก์ชัน BitBlt()) แต่ต้องคำนึงถึงส่วนรวมเสมอนั่นคือเมื่อไม่จำเป็นต้องใช้ hidden page แล้วให้คืนหน่วยความจำที่เป็น hidden page แก่ระบบทันที

4. การสื่อสารระหว่างสไปรต์กับสภาพแวดล้อม

เนื่องจากวินโดวส์สนับสนุนการทำงานในเชิงข้อความอยู่แล้ว ดังนั้นระบบจัดการภาพเคลื่อนไหวจึงให้สไปรต์มีการสื่อสารกับสภาพแวดล้อมด้วยข้อความข่าวสารที่กำหนดขึ้นมาใหม่เพื่อใช้ในกับระบบนี้โดยเฉพาะ

จากที่กล่าวมาแล้วจะเห็นว่าหลักการทำงานของระบบจัดการภาพเคลื่อนไหวนั้นจะเกี่ยวข้องกับตัวสไปรต์ สภาพแวดล้อมของโปรแกรม ตัวจักรในการสร้างภาพเคลื่อนไหวและการสื่อสารกันระหว่างสไปรต์กับสภาพแวดล้อมของโปรแกรม

แผนภาพต่อไปนี้แสดงความสัมพันธ์ของส่วนที่เกี่ยวข้องกับระบบ



รูปที่ 3.1 แสดงความสัมพันธ์ระหว่างส่วนที่เกี่ยวข้อง

จากรูปที่ 3.1 ในระบบจะต้องมีฟังก์ชันประจำวินโดว์ของโปรแกรมประยุกต์ ซึ่งจะคอยรับข้อความข่าวสารจากแหล่งกำเนิด เช่น ข้อความจากตัวจับเวลา ข้อความคำสั่งเกี่ยวกับเมนู ข้อความการยกเลิกระบบ เป็นต้น ฟังก์ชันประจำวินโดว์นี้จะตรวจสอบหรือคัดเลือกว่าข้อความใดที่จะใช้ในตัวฟังก์ชันเองก็จะประมวลผลไป แต่ถ้าเป็นข้อความหรือเหตุการณ์ที่เกี่ยวกับสไปรต์ ฟังก์ชันประจำวินโดว์ก็จะส่งข้อความนั้นไปให้สไปรต์หรือฟังก์ชันประจำตัวสไปรต์แทน จากนั้นสไปรต์แต่ละตัวที่ได้รับข้อความก็จะประมวลผลไปตามเหตุการณ์หรือข้อความที่ได้รับนั้น เมื่อฟังก์ชันประจำตัวสไปรต์จัดการตัวเองเรียบร้อยก็จะบอกฟังก์ชันประจำวินโดว์และเป็นหน้าที่ของฟังก์ชันประจำวินโดว์ที่จะเรียกตัวจักรทำภาพเคลื่อนไหวตามเหตุการณ์ที่ต้องการเช่น ให้ทำงานทุกครั้งที่ได้รับข้อความ WM_TIMER หรือข้อความ WM_ANIMATE (ผู้เขียนกำหนดเอง)

การทำงานของระบบจัดการภาพเคลื่อนไหว

ในวิทยานิพนธ์ได้ออกแบบให้ระบบจัดการภาพเคลื่อนไหวมีหลักการทำงานดังนี้

1. เมื่อมีการเขียนโปรแกรมภายใต้วินโดวส์ จะต้องมีฟังก์ชันหลักของโปรแกรมประยุกต์คือ ฟังก์ชัน WinMain() และมีฟังก์ชันประจำวินโดว์ซึ่งทำหน้าที่ควบคุมการทำงานและทำหน้าที่จัดการทำงานตามข้อความที่ได้รับ (ชื่อของฟังก์ชันนี้ถูกกำหนดโดยผู้เขียนโปรแกรม) นอกจากนี้ฟังก์ชันประจำวินโดว์ยังทำหน้าที่เป็นผู้ส่งข้อความต่างๆให้วินโดว์ลูกและสไปรต์อีกด้วย

2. สไปรต์ถูกออกแบบให้เป็นวินโดว์ลูกชนิดหนึ่ง ดังนั้นจึงมีคุณสมบัติบางประการของวินโดว์ด้วยนั่นหมายความว่าสไปรต์จะต้องมีฟังก์ชันประจำตัวสไปรต์ (ซึ่งเป็นฟังก์ชันประจำวินโดว์นั่นเอง) ซึ่งจะทำหน้าที่เป็นตัวจัดการข้อความ (message handler) ให้กับสไปรต์ ผู้เขียนโปรแกรมจึงสามารถกำหนดพฤติกรรมของสไปรต์ได้ที่นี่ ทำให้สไปรต์ทำงานตามเหตุการณ์ (event-driven) เหมือนวินโดว์ทั่วๆไป

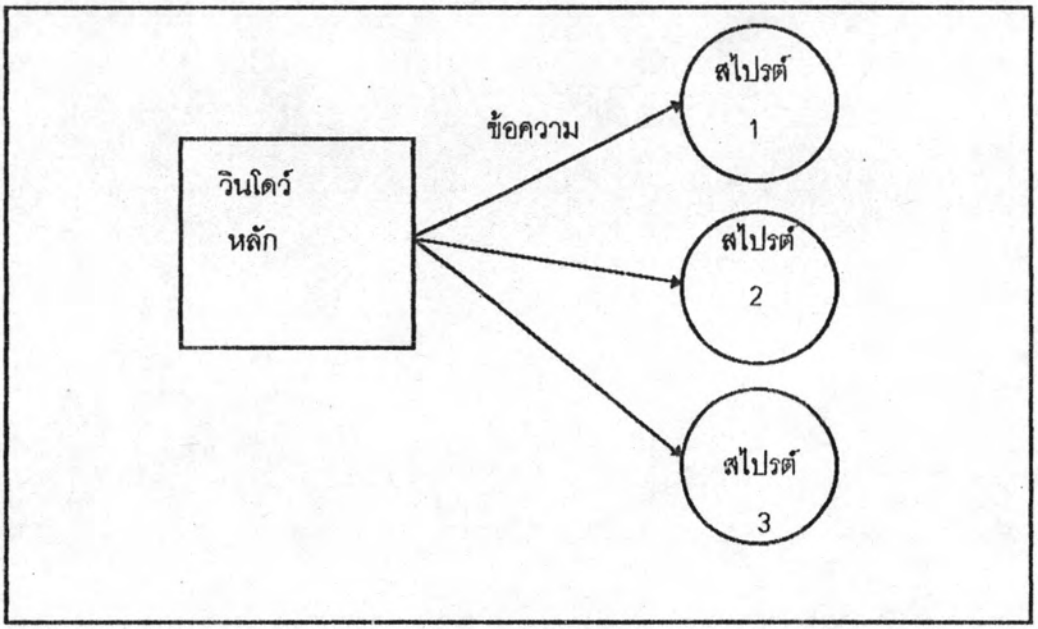
3. เนื่องจากสไปรต์มีคุณสมบัติของวินโดว์ ดังนั้นมันจึงสามารถมีสไปรต์ที่เป็นลูกได้ ตัวอย่างของสไปรต์ที่เป็นลูกของสไปรต์อีกทีเช่น เครื่องบินรบเป็นสไปรต์ตัวแม่และมีจรวดนำวิถีเป็นสไปรต์ตัวลูก

4. วินโดว์หลักหรือวินโดว์ที่เป็นวินโดว์แม่จะควบคุมสไปรต์ได้ 2 วิธี คือ

4.1. ส่งข้อความให้สไปรต์

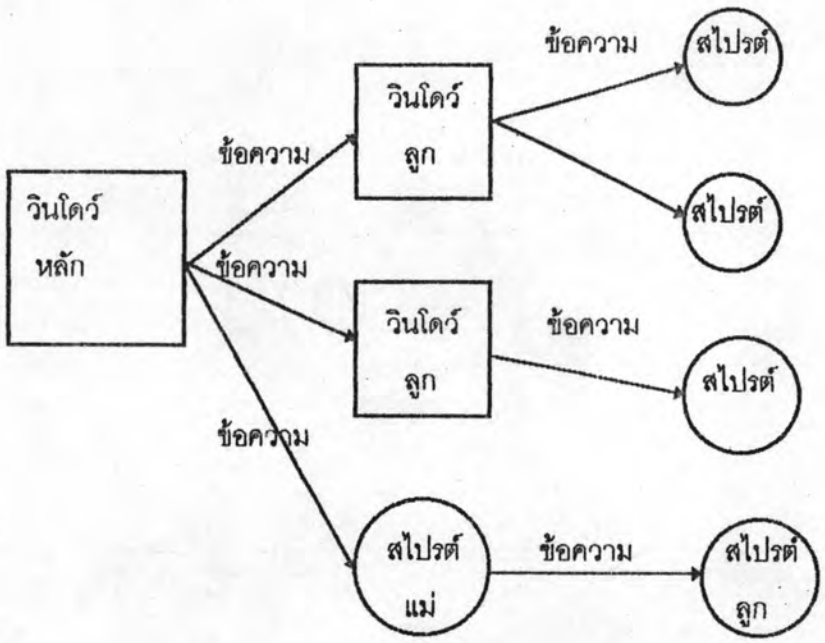
4.2. ใช้ฟังก์ชันในคลังโปรแกรมจัดการกับสไปรต์โดยตรง

จากหลักการทำงานของระบบดังกล่าวจะเห็นว่าวินโดว์หลักหรือวินโดว์ตัวแม่จะเป็นผู้ควบคุมหรือชักใยการทำงานของสไปรต์ผ่านทางการส่งข้อความ ดังรูปตัวอย่างต่อไปนี้



รูปที่ 3.2 แสดงการติดต่อระหว่างวินโดว์หลักกับสไปรต์

ในโปรแกรมประยุกต์สามารถแบ่งวินโดว์หลักออกเป็นวินโดว์ลูกได้หลายบาน ในวินโดว์ลูกแต่ละบานก็สามารถมีสไปรต์ในความดูแลของมันเองได้ดังรูปต่อไปนี้



รูป 3.3 แสดงความสัมพันธ์ระหว่างวินโดว์กับสไปรต์

จากรูปที่ 3.2 ในวินโดว์หลักมีสไปรต์เป็นสมาชิกอยู่ 3 ตัว วินโดว์หลักชักใยสไปรต์ทั้งสามตัวโดยการส่งข้อความไปให้เนื่องจากตัววินโดว์หลักเองก็ทำงานตามเหตุการณ์ ทำให้การเขียนโปรแกรมสามารถควบคุมการส่งข้อความไปยังสไปรต์ได้ กล่าวคือวินโดว์หลักสามารถเลือกที่จะส่งข้อความไปให้สไปรต์ทั้งหลายได้ตามเหตุการณ์ที่เกิดขึ้น และจะส่งข้อความให้สไปรต์ทุกตัวหรือจะเลือกส่งให้บางตัวแล้วแต่เหตุการณ์และเงื่อนไขได้ ต่อมาจากรูปที่ 3.3 ในโปรแกรมประยุกต์สามารถมีวินโดว์ย่อยๆได้ตามความต้องการโดยที่ในแต่ละวินโดว์ย่อยเหล่านั้นมีสไปรต์ในความดูแลของมันด้วย ดังนั้นในวินโดว์ใดก็ตามที่มีสไปรต์จะต้องมีการเก็บบันทึกข้อมูลเกี่ยวกับสภาพแวดล้อมด้วยเสมอ

<u>การออกแบบโครงสร้างข้อมูลของสไปรต์</u>

จากความหมายของสไปรต์ตามที่กล่าวถึงในบทที่ผ่านมานั้น ได้ออกแบบให้สไปรต์มีส่วนประกอบของกลุ่มโครงสร้างข้อมูลดังนี้

1. ภาพบิตแมพของสไปรต์ (Sprite Bitmap)
2. ตำแหน่งของสไปรต์ (Sprite Coordinate)
3. ภาพพื้นหลังที่สไปรต์เคลื่อนที่ผ่าน (Background)
4. ข้อมูลการควบคุมสไปรต์ (Sprite Control Element)
5. ฟังก์ชันประจำตัวของสไปรต์ (Sprite Procedure)

ภาพบิตแมพของสไปรต์

สไปรต์มีได้ทั้งแบบภาพเดียวและแบบหลายภาพ ในกรณีที่เป็นแบบภาพเดียวเช่น สไปรต์ที่เป็นก้อนเมฆ สไปรต์ที่เป็นดวงอาทิตย์ หรือสไปรต์ที่ไม่มีการเปลี่ยนรูปร่าง ภาพบิตแมพที่ใช้จะมี 1 คู่ (2 ภาพบิตแมพ) ถ้าสไปรต์นั้นมีการเปลี่ยนแปลงรูปร่างเช่น สไปรต์ที่เป็นรูปคนมีการยกแขนขึ้นลงได้ จะต้องมีการเก็บภาพบิตแมพที่เป็นอิริยาบทเหล่านั้นของสไปรต์ด้วย จึงทำให้มีการเก็บภาพบิตแมพของสไปรต์ชนิดนี้หลายคู่

การที่อ้างถึงการเก็บภาพบิตแมพของสไปรต์เป็นคู่นั้น เนื่องจากต้องการให้ภาพบิตแมพที่วางลงไปบนภาพพื้นหลังสามารถมองเห็นตามช่องว่างของภาพได้ โดยมีวิธีการดังนี้

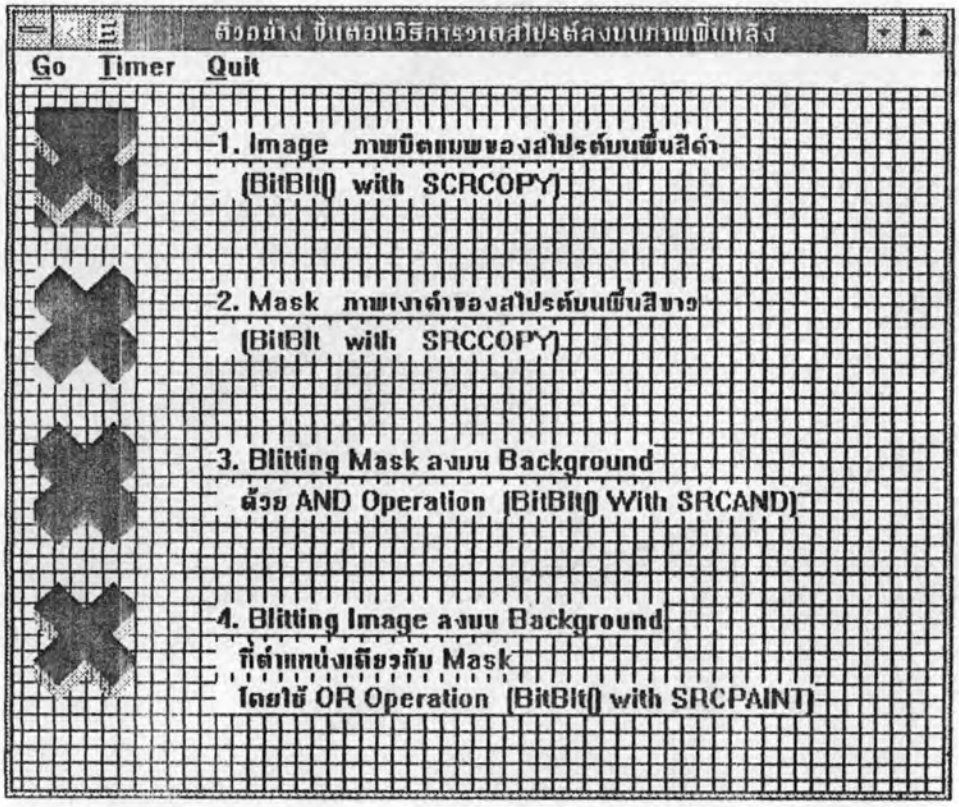
1. วาดภาพของสไปรต์อยู่บนพื้นสีดำ (Picture) ดังภาพที่ 1 ในรูปที่ 3.4
2. วาดภาพเงาดำของสไปรต์บนพื้นสีขาว (Mask) ดังภาพที่ 2 ในรูปที่ 3.4
3. ใช้วิธีการดำเนินการระดับบิตแบบ Raster (Raster Operation)

การวาดสไปรต์โดยใช้การดำเนินการแบบ Raster

วิธีการวาดภาพสไปรต์ลงบนภาพพื้นหลังมีดังนี้

1. วาดภาพบิตแมพที่เป็นเงาของสไปรต์ (Mask) ลงบนบิตแมพของภาพพื้นหลัง โดยใช้การดำเนินการ Raster ชนิด AND ด้วยฟังก์ชัน BitBlt() ใช้ Operator SRCAND จะได้ผลลัพธ์ดังภาพที่ 3 ในรูปที่ 3.4

2. วาดภาพบิตแมพของสไปรต์ลงบนผลลัพธ์ในข้อที่ 1 โดยใช้การดำเนินการ Raster ชนิด OR ด้วยฟังก์ชัน BitBlt() ใช้ Operator SRCPAINT จะได้ผลลัพธ์ดังภาพที่ 4 ในรูปที่ 3.4

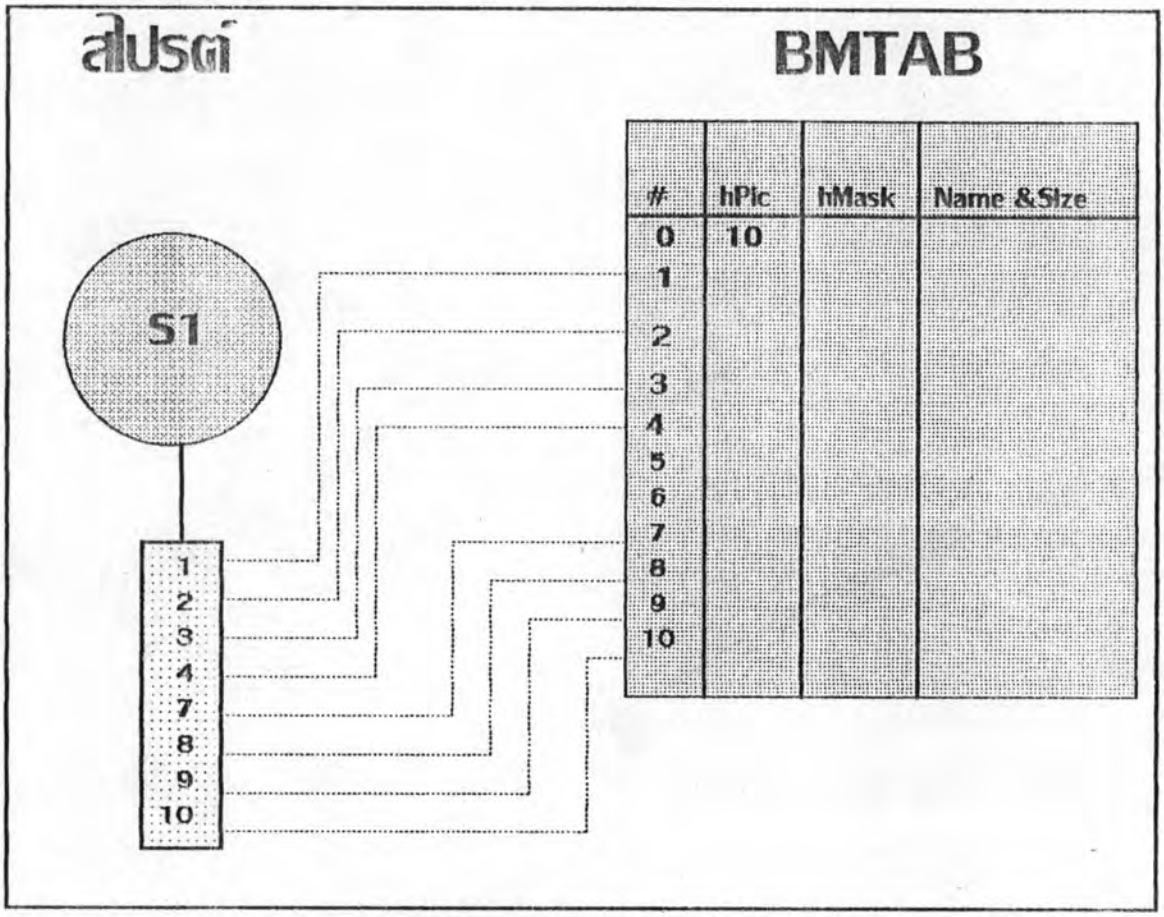


รูปที่ 3.4 แสดงขั้นตอนการวาดสไปรต์ลงบนภาพพื้นหลัง

จากรูปที่ 3.4 เป็นขั้นตอนในการวาดภาพของสไปรต์โดยนำภาพบิตแมพมาใช้ ฟังก์ชันที่ใช้ในการ Blitting ซึ่งวินโดวส์ได้จัดเตรียมไว้ให้คือฟังก์ชัน BitBlt() สำหรับ Raster Operator นั้นวินโดวส์มีให้ใช้หลายตัวแต่ที่จำเป็นต้องใช้มีเพียง 3 ตัวได้แก่ SRCCOPY SRCAND และ SRCPAINT จากตัวอย่างในรูปที่ 3.4 นี้ทำการ Blitting ลงบนจอภาพโดยตรงแต่ในการทำงานของตัวจักรทำภาพเคลื่อนไหวจะจัดการลงใน hidden page เสียก่อนแล้วจึง Blitting ผลลัพธ์ที่ได้ไปยังจอภาพ

จากวิธีการดังกล่าวจึงออกแบบให้มีการเก็บภาพบิตแมพแต่ละภาพเป็น Picture กับ Mask โดยผู้เขียนโปรแกรมวาดเพียงภาพที่เป็น Picture (ภาพสไปรต์บนพื้นสีดำ) ส่วนภาพที่เป็น Mask นั้นในคลังโปรแกรมได้จัดเตรียมฟังก์ชันอำนวยความสะดวกในการสร้าง Mask ไว้ให้แล้ว

เนื่องจากสไปรต์หลายตัวอาจใช้ภาพบิตแมพชุดเดียวกันได้ ดังนั้นจึงออกแบบให้มีการเก็บภาพบิตแมพไว้ในคลังภาพ โดยให้สไปรต์มีดัชนีชี้ไปที่คลังภาพที่ต้องการ ทำให้มีการอ่านข้อมูลภาพเข้ามาเพียงครั้งเดียว ดังตัวอย่างในภาพต่อไปนี้



รูปที่ 3.5 แสดงคลังภาพบิตแมพ

จากรูปที่ 3.5 แสดงการเก็บภาพบิตแมพไว้ในคลังภาพเพื่อให้สไปรต์หลายตัวสามารถใช้ร่วมกันได้ โดยที่ตัวสไปรต์เองจะต้องมีดัชนีชี้ไปที่คลังภาพด้วยว่าใช้ภาพใดบ้าง

ดังนั้นจึงต้องมีการออกแบบโครงสร้างข้อมูลของคลังภาพบิตแมพด้วยโดยมีส่วนประกอบดังต่อไปนี้

1. ภาพบิตแมพของสไปรต์บนพื้นสีดำ
2. ภาพบิตแมพเงาดำของสไปรต์บนพื้นขาว
3. ชื่อของภาพบิตแมพนั้น
4. ความกว้างและความสูงของบิตแมพนั้น

สามารถอธิบายโครงสร้างข้อมูลนี้ในรูปแบบของภาษาซีได้ดังนี้

```
typedef  struct tagBmTab {
         HBITMAP    hPic;          // ภาพบิตแมพบนพื้นดำ
         HBITMAP    hMask;         // ภาพเงาดำบนพื้นขาว
         LPSTR      Name;          // ชื่ออ้างอิงของภาพบิตแมพ
         int        bmWidth;       // ความกว้างของภาพบิตแมพ
         int        bmHeight;      // ความสูงของภาพบิตแมพ
} BMTAB;                           // ชื่ออ้างอิงของโครงสร้างที่จะใช้ต่อไป
```

ในการเขียนโปรแกรมภายใต้วินโดวส์จะถือว่าภาพบิตแมพเป็นวัตถุ (object) ที่เป็นทรัพยากรของระบบ การอ้างอิงถึงภาพบิตแมพกระทำได้โดยใช้ตัวชี้ที่เรียกว่า แฮนเดิลของบิตแมพ (handle to bitmap)

เนื่องจากภาพของสไปรต์อาจมีได้หลายภาพดังนั้นคลังภาพจะใช้แถวลำดับ 1 มิติของโครงสร้างในการเก็บภาพเช่น สไปรต์ตัวหนึ่งมีภาพที่ใช้ในการเปลี่ยนแปลงอิริยาบทอยู่ทั้งหมด 7 ภาพ แถวลำดับของโครงสร้างคลังภาพก็จะมีสมาชิก 7 ชุด เมื่อมีการเก็บภาพในแถวลำดับของโครงสร้างคลังภาพ โครงสร้างข้อมูลของตัวสไปรต์ก็จะต้องมีตัวแปรที่ใช้ในการควบคุมการเลือกภาพในคลังภาพมาใช้งานด้วย ตัวแปรเหล่านี้ใช้เก็บข้อมูลเกี่ยวกับจำนวนภาพ ลำดับของภาพในคลัง ทิศทางการเลือกภาพเช่นเลือกจากภาพแรกไปภาพสุดท้ายหรือเลือกจากภาพสุดท้ายมาภาพแรกหรือเลือกเฉพาะภาพใดภาพหนึ่งเท่านั้นเป็นต้น

ตำแหน่งของสไปรต์

ข้อมูลในกลุ่มนี้ใช้อ้างอิงตำแหน่งการเคลื่อนที่ของตัวสไปรต์บนจอภาพโดยสัมพันธ์กับวินโดว์ที่แสดงภาพของสไปรต์นั้น ได้แก่ ตำแหน่งปัจจุบัน ตำแหน่งที่ผ่านมา ทิศทางการเคลื่อนที่ของสไปรต์เป็นต้น ตำแหน่งของสไปรต์นั้นอ้างเป็น (X,Y,Z) โดยที่ X,Y แทนตำแหน่งในวินโดว์ ส่วน Z ใช้อ้างถึงระดับการเคลื่อนที่ของตัวสไปรต์หรือเพลน (plane) เพื่อใช้ในการจัดลำดับการแสดงภาพของสไปรต์และตรวจสอบการชนกันของสไปรต์กรณีที่อยู่บนระดับเดียวกันและมีตำแหน่ง X,Y เหลื่อมล้ำกัน

ภาพพื้นหลังที่สไปรต์เคลื่อนที่ผ่าน

ในวินโดว์ที่สไปรต์มีการเคลื่อนไหวอยู่นั้นอาจมีการใช้ภาพพื้นหลังเป็นภาพบิตแมพที่สร้างขึ้นล่วงหน้าไว้แล้วด้วยโปรแกรมสร้างภาพหรือเป็นภาพกราฟิกที่สร้างขึ้นในขณะที่ดำเนินโปรแกรมก็ได้

นอกจากนี้ภาพพื้นหลังของวินโดว์อาจเป็นสีพื้นธรรมดาที่ไม่มีลวดลายกราฟิกใดๆ แต่อย่างไรก็ตามเราจำเป็นจะต้องมีการเก็บบันทึกข้อมูลของภาพพื้นหลังไว้

ข้อมูลการควบคุมสไปรต์

ในการควบคุมพฤติกรรมการเคลื่อนไหวของสไปรต์นั้นจะต้องมีกลุ่มข้อมูลบางอย่างเป็นสมาชิกในโครงสร้างข้อมูลของสไปรต์ด้วย ตัวอย่างเช่น ระดับความสำคัญของตัวสไปรต์ อายุของสไปรต์ สถานะของสไปรต์ เป็นต้น กลุ่มข้อมูลนี้จะมองเห็นโดยตัวจักรในการทำภาพเคลื่อนไหว นอกจากนี้วินโดว์หลักยังสามารถร้องขอข้อมูลนี้จากการส่งข้อความร้องขอข้อมูลมาที่ตัวสไปรต์ได้

ฟังก์ชันประจำตัวของสไปรต์

เนื่องจากต้องการให้สไปรต์แต่ละตัวสามารถมีพฤติกรรมในการเคลื่อนไหวเมื่อมีเหตุการณ์ใดๆที่สไปรต์ตัวนั้นรู้จักเกิดขึ้น หรือกล่าวอีกนัยหนึ่งคือสไปรต์จะเคลื่อนไหวตามเหตุการณ์ (event-driven) ซึ่งเป็นแบบเดียวกับคุณสมบัติในการทำงานของวินโดว์ ดังนั้นจึงออกแบบให้สไปรต์เป็นวินโดว์ลูก (child window) ชนิดไม่มีกรอบและพื้นหลังของวินโดว์ (ไม่มี border และ background) เมื่อสไปรต์เป็นวินโดว์ชนิดหนึ่งจึงต้องมีฟังก์ชันประจำวินโดว์ ในส่วนนี้จึงเป็นหน้าที่ของผู้เขียนโปรแกรมว่าต้องการจะให้สไปรต์มีพฤติกรรมอย่างไรก็มากำหนดการทำงานตามเหตุการณ์ในฟังก์ชันประจำสไปรต์ สำหรับฟังก์ชันประจำตัวของสไปรต์นั้นตามปกติควรจะมีให้กับสไปรต์แต่ละตัว แต่ในกรณีที่ต้องการให้สไปรต์ตัวหนึ่งมีพฤติกรรมเหมือนกับสไปรต์อีกตัวหนึ่งก็สามารถทำได้ โดยกำหนดให้สไปรต์เหล่านี้มีฟังก์ชันประจำสไปรต์เป็นฟังก์ชันเดียวกัน นอกจากนี้ในคลังโปรแกรมที่พัฒนาขึ้นก็ได้จัดทำฟังก์ชันประจำตัวสไปรต์ที่มีพฤติกรรมการเคลื่อนไหวแบบทั่วๆไปไว้ให้แล้วโดยที่ผู้เขียนโปรแกรมสามารถใช้ได้เลยโดยใช้วิธีส่งข้อความไปให้กับสไปรต์ที่ใช้ฟังก์ชันประจำตัวที่สร้างไว้ให้นี้ สำหรับการส่งข้อความเพื่อบอกเหตุการณ์ให้สไปรต์นั้น ให้ส่งไปที่ตัวสไปรต์ได้เลยโดยไม่จำเป็นต้องส่งไปที่ฟังก์ชันประจำสไปรต์ตามแบบวินโดว์เนื่องจากในคลังโปรแกรมได้จัดทำฟังก์ชันอำนวยความสะดวกนี้ไว้ให้แล้วไม่ว่าจะเป็นการส่งข้อความให้สไปรต์แบบเฉพาะตัวหรือส่งให้ทุกตัว

ตามที่กล่าวมาแล้วนั้นเป็นกลุ่มข้อมูลที่ควรจะมีในโครงสร้างข้อมูลของสไปรต์และนำมาออกแบบเป็นโครงสร้างข้อมูลตามรูปแบบของภาษาซีได้ดังนี้

```
typedef struct  tagSprite {
    UINT        sId;                    // หมายเลขประจำตัวสไปรต์
    int         CurX, CurY, CurZ;       // ตำแหน่งปัจจุบัน
    int         LastX, LastY, LastZ;    // ตำแหน่งที่ผ่านมา
    MOVE        MoveProp;               // ข้อมูลที่ใช้ในการเคลื่อนที่ตามเส้นทาง
    HSPRPATH    hPath;                  // เส้นทางการเคลื่อนที่ของสไปรต์
    char        MaxFrame,  CurFrame;    // ควบคุมการเปลี่ยนภาพ
    HBMTAB      hBmTab;                 // ดัชนีชี้คลังภาพของสไปรต์
    HFRAMELIST  hFrameList;             // แถวลำดับหมายเลขของภาพในคลังภาพ
    HBITMAP     bmBkGd;                 // ดัชนีชี้ภาพพื้นหลัง
    int         bmWidth, bmHeight;      // ขนาดปัจจุบันของสไปรต์
    int         LastWidth, LastHeight;  // ขนาดที่ผ่านมาของสไปรต์
    char        MoveDir;                // ทิศทางการเคลื่อนที่ของสไปรต์
    char        bmDir;                  // ทิศทางการเลือกภาพในคลังภาพ
    char        Size;                   // มีการเปลี่ยนขนาดของสไปรต์หรือไม่
    char        Priority,  CurPriority; // ลำดับความสำคัญของสไปรต์
    char        Status;                 // สถานภาพของสไปรต์
    int         Timer, LifeTime;        // อายุของสไปรต์
    HWND        hDispWnd;               // วินโดว์ที่สไปรต์เคลื่อนไหวอยู่
    HWND        hWnd;                   // วินโดว์ประจำตัวสไปรต์
    WNDPROC     SprProc;                // ฟังก์ชันประจำตัวของสไปรต์
    char        CollStatus;             // สถานภาพการชนกันของสไปรต์
    COLLISION   Collision;              // ตารางขอบเขตของสไปรต์ ใช้ตรวจสอบ
} SPRITE;                               // การชนกัน
```

จากโครงสร้างของสไปรต์ดังกล่าวจะเห็นว่ามีชนิดของข้อมูลใหม่ได้แก่ MOVE HSPRPATH HFRAMELIST และ COLLISION โดยข้อมูลชนิด MOVE เก็บสถานะในการเคลื่อนที่ตามเส้นทางที่กำหนดใน HSPRPATH ซึ่งเป็นแฮนเดิลสำหรับหน่วยความจำที่เก็บเส้นทางของสไปรต์นั่นเอง โครงสร้างข้อมูลของ MOVE มีสมาชิกดังนี้

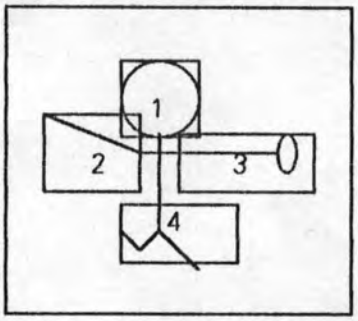
```
typedef  struct  tagMove {
        int     dX, dY, ErrorTerm, Counter;     // ใช้ในการคำนวณตำแหน่งตามเส้นทาเดิน
        char    InitFlag;                       // บอกสถานะการเคลื่อนที่ตามเส้นทาง
} MOVE;

typedef  struct  tagSprPath {
        int         CurPoint;                   // ลำดับของเส้นทางที่กำลังเดินอยู่
        int         MaxPoint;                   // จำนวนเส้นตรงทั้งหมดที่เป็นเส้นทาง
        LPPOINT     lpPoint;                    // Pointer ชี้แถวลำดับของเส้นทาง
} SPRPATH;
```

HFRAMELIST ในการเก็บหมายเลขลำดับของภาพในคลังภาพ และ COLLISION เป็นตารางเก็บขอบเขตสำคัญของสไปรต์เพื่อใช้ตรวจสอบการชนกันในกรณีที่สไปรต์มากกว่า 1 ตัวอยู่ในระนาบเดียวกัน (มีค่าของ CurZ เท่ากัน) HFRAMELIST ถูกออกแบบให้มีโครงสร้างดังนี้

```
typedef  struct  tagFrameList {
        char  BmTabNumber;      // หมายเลขลำดับของภาพในคลัง
        char  Status;           // เปลี่ยนขนาดของภาพหรือไม่ (1=Yes, 0=No)
        char  DeltaWidth;       // + = ขยาย, - = หด    ด้านกว้าง
        char  DeltaHeight;      // + = ขยาย, - = หด    ด้านสูง
} FRAMELIST;
```

การตรวจสอบการชนกันของสไปรต์นั้นอาจเป็นส่วนสำคัญที่ทำให้การดำเนินไปของโปรแกรมช้าลง เนื่องจากสไปรต์ที่เป็นภาพบิตแมพมีรูปร่างต่างๆกันไปซึ่งมีทั้งที่เป็นรูปทรงเรขาคณิตหรือไม่เป็นก็ได้ ในกรณีที่เป็นภาพบิตแมพที่มีรูปทรงไม่แน่นอนเมื่อต้องการตรวจสอบการชนกันอย่างละเอียดที่สุดก็คือการตรวจสอบทีละจุดภาพว่ามีการซ้อนกันหรือไม่ วิธีการนี้ตรวจสอบได้ถูกต้องแต่เสียเวลามากนั่นคือกรณีดีที่สุดได้แก่การตรวจพบการชนกันที่จุดแรกของแต่ละภาพพอดี แต่กรณีที่ช้าที่สุดคือไม่ชนกัน (แต่กรอบของภาพทับกัน) ถ้าภาพบิตแมพขนาดเล็กการตรวจสอบอาจไม่ช้านัก แต่ถ้าเป็นภาพบิตแมพขนาดใหญ่เช่น กว้าง 300 จุด สูง 300 จุด ทั้งสองภาพการตรวจสอบจะช้าและจะยิ่งช้าลงจนใช้ไม่ได้ถ้ามีจำนวนสไปรต์ให้ตรวจสอบหลายๆตัว

ดังนั้นจึงได้ออกแบบให้สไปรต์มีขอบเขตสำคัญในการตรวจสอบการชนกัน โดยให้ขอบเขตนั้นอยู่ในพื้นที่สี่เหลี่ยม ซึ่งพื้นที่สี่เหลี่ยมนี้จะต้องอยู่ในกรอบภาพของสไปรต์และอาจมีหลายพื้นที่ได้ตามต้องการแต่มีข้อเสนอแนะว่าขนาดของพื้นที่สี่เหลี่ยมไม่ใช่ตัวแปรสำคัญ จำนวนของพื้นที่สี่เหลี่ยมสำคัญกว่าเนื่องจากว่ายิ่งมีพื้นที่สี่เหลี่ยมมากเท่าใดจำนวนการตรวจสอบก็จะมากขึ้นตามไปด้วย



รูปที่ 3.6 แสดงตัวอย่างการกำหนดขอบเขตตรวจสอบการชนกัน

จากรูปจะเห็นว่าภาพบิตแมพจะมีกรอบของภาพ ภายในกรอบภาพสามารถกำหนดพื้นที่สี่เหลี่ยมเพื่อใช้ในการตรวจสอบการชนกันได้หลายพื้นที่ตามต้องการ วิธีการนี้อาจไม่ถูกต้องร้อยเปอร์เซนต์แต่สามารถยอมรับได้ในระดับหนึ่งและในด้านความเร็วในการตรวจสอบนับว่าใช้ได้ดี โครงสร้างข้อมูลของขอบเขตสำคัญนี้มีลักษณะดังนี้

```
typedef   RECT   FAR *  LPRECT;
typedef  struct  tagCollTab {              // สำหรับแต่ละภาพบิตแมพ
        LPRECT   lpRect;                   // แถวลำดับของพื้นที่สี่เหลี่ยม
        char     nRect;                    // จำนวนพื้นที่สี่เหลี่ยมในแถวลำดับ
} COLLTAB;
typedef  COLLTAB  FAR *  LPCOLLTAB;
typedef  struct  tagCollision {            // สไปรต์ 1 ตัวอาจมีหลายภาพ
     LPCOLLTAB   lpCollFrameTab;           // แถวลำดับของขอบเขตการตรวจสอบ
     char        nCollFrame;               // จำนวนขอบเขตทั้งหมด
     char        CurCollFrame;             // ขอบเขตของภาพปัจจุบัน
} COLLISION;
```

จากโครงสร้างของสไปรต์ที่ผ่านมาจะเห็นว่ามีการกำหนดสมาชิกบางตัวเป็นโครงสร้างซ้อนกันอยู่และสมาชิกบางตัวมีชนิดเป็นแฮนเดิลของโครงสร้าง ตัวอย่างเช่น ในโครงสร้างของสไปรต์มีสมาชิกที่

เก็บข้อมูลของเส้นทางเดินของสไปรต์เป็น HSPRPATH hPath; แต่โครงสร้างของทางเดินของสไปรต์กำหนดไว้เป็น SPRPATH เนื่องจากว่าในการจัดสรรหน่วยความจำจากวินโดวส์นั้นจะต่างจากระบบอื่นเล็กน้อย โดยวินโดวส์จะจัดสรรหน่วยความจำให้ตามที่ขอ แต่จะส่งมาให้เป็นค่าแฮนเดิลของหน่วยความจำนั้นๆ (ถ้ามีการขอจองหน่วยความจำแบบเคลื่อนย้ายได้) จากนั้นเป็นหน้าที่ของผู้เขียนโปรแกรมที่จะทำการ Lock หน่วยความจำที่ได้นั้นไว้เพื่อไม่ให้มีการโยกย้ายข้อมูลในขณะที่กำลังใช้งานอยู่ ค่าที่ได้รับจากการ Lock นี้จะเป็นตำแหน่งในหน่วยความจำที่เก็บข้อมูล เมื่อใช้งานหน่วยความจำส่วนนี้เสร็จแล้วจะต้องปลด Lock ทันทีเพื่อให้วินโดวส์สามารถโยกย้ายข้อมูลไปส่วนอื่นของหน่วยความจำได้ในกรณีที่จำเป็น ดังตัวอย่างต่อไปนี้

```
HANDLE   hMem;
LPSTR    lpName;
hMem   = LocalAlloc ( LMEM_MOVEABLE,  20);
lpName = (LPSTR)  LocalLock (hMem);
// ใช้งานหน่วยความจำได้ตามปรกติ
LocalUnlock (hMem);
// ถ้าไม่ต้องการใช้อีกแล้วให้ยกเลิกไป
LocalFree(hMem);
```

ด้วยเหตุนี้จึงออกแบบให้สมาชิกของโครงสร้างใดก็ตามที่จะใช้เป็นพอยเตอร์ชี้หน่วยความจำจะถูกกำหนดให้เป็นแฮนเดิลของโครงสร้างนั้นๆแทน โดยที่ชื่อของแฮนเดิลของโครงสร้างนั้นจะคล้ายกับชื่อของโครงสร้างแต่จะมีอักษร H นำหน้าไว้เสมอเพื่อสังเกตได้ง่าย เช่นแฮนเดิลโครงสร้างของสไปรต์จะเป็น HSPRITE (โครงสร้างชื่อ SPRITE มี LPSPRITE เป็นพอยเตอร์ชี้ไปยังหน่วยความจำที่เก็บโครงสร้าง SPRITE) เป็นต้น

ต่อไปนี้เป็นการกำหนดชนิดข้อมูลให้กับแฮนเดิลของโครงสร้างต่างๆที่ใช้ในคลังโปรแกรม

```
typedef  HANDLE    HSPRITE;
typedef  HANDLE    HBMTAB;
typedef  HANDLE    HFRAMELIST;
typedef  HANDLE    HSPRTAB;
typedef  HANDLE    HSEQUENCE;
typedef  HANDLE    HENVIRON;
typedef  HANDLE    HSPRPATH;
```

ตัวอย่างการขอหน่วยความจำสำหรับโครงสร้างสไปรต์

```
HSPRITE   hSprite;
LPSPRITE  lpSprite;

hSprite = GlobalAlloc (GHND, sizeof(SPRITE));
// ในขั้นตอนนี้จะได้พื้นที่ของหน่วยความจำขนาดเท่ากับโครงสร้างของสไปรต์
// แต่เป็นหน่วยความจำที่ถูกโยกย้ายได้โดยวินโดวส์
lpSprite  = (LPSPRITE) GlobalLock (hSprite);
// ต้อง Lock หน่วยความจำที่ hSprite เป็นตัวชี้เสียก่อน เพื่อให้วินโดวส์ไม่โยกย้ายไป
// การ Lock ถ้าทำสำเร็จจะให้ค่า address ของหน่วยความจำที่ขอจองไว้
GlobalUnlock (hSprite);
// เมื่อไม่ใช้หน่วยความจำนี้ให้ปลด Lock เพื่อให้วินโดวส์สามารถโยกย้ายไปได้
GlobalFree (hSprite);
// ยกเลิกหน่วยความจำที่จองนี้ คืนให้ระบบ
```

<u>การออกแบบโครงสร้างข้อมูลของสภาพแวดล้อมในการทำงาน</u>

สภาพแวดล้อมของระบบจัดการภาพเคลื่อนไหวที่มีสไปรต์เคลื่อนที่อยู่มีความสำคัญต่อกลไกทำภาพเคลื่อนไหวเป็นอย่างมาก ตัวจักรในการทำภาพเคลื่อนไหวจะเข้าถึงข้อมูลของสไปรต์แต่ละตัวได้โดยผ่านทางโครงสร้างข้อมูลนี้ รายละเอียดของโครงสร้างข้อมูลของสภาพแวดล้อมในระบบมีดังต่อไปนี้

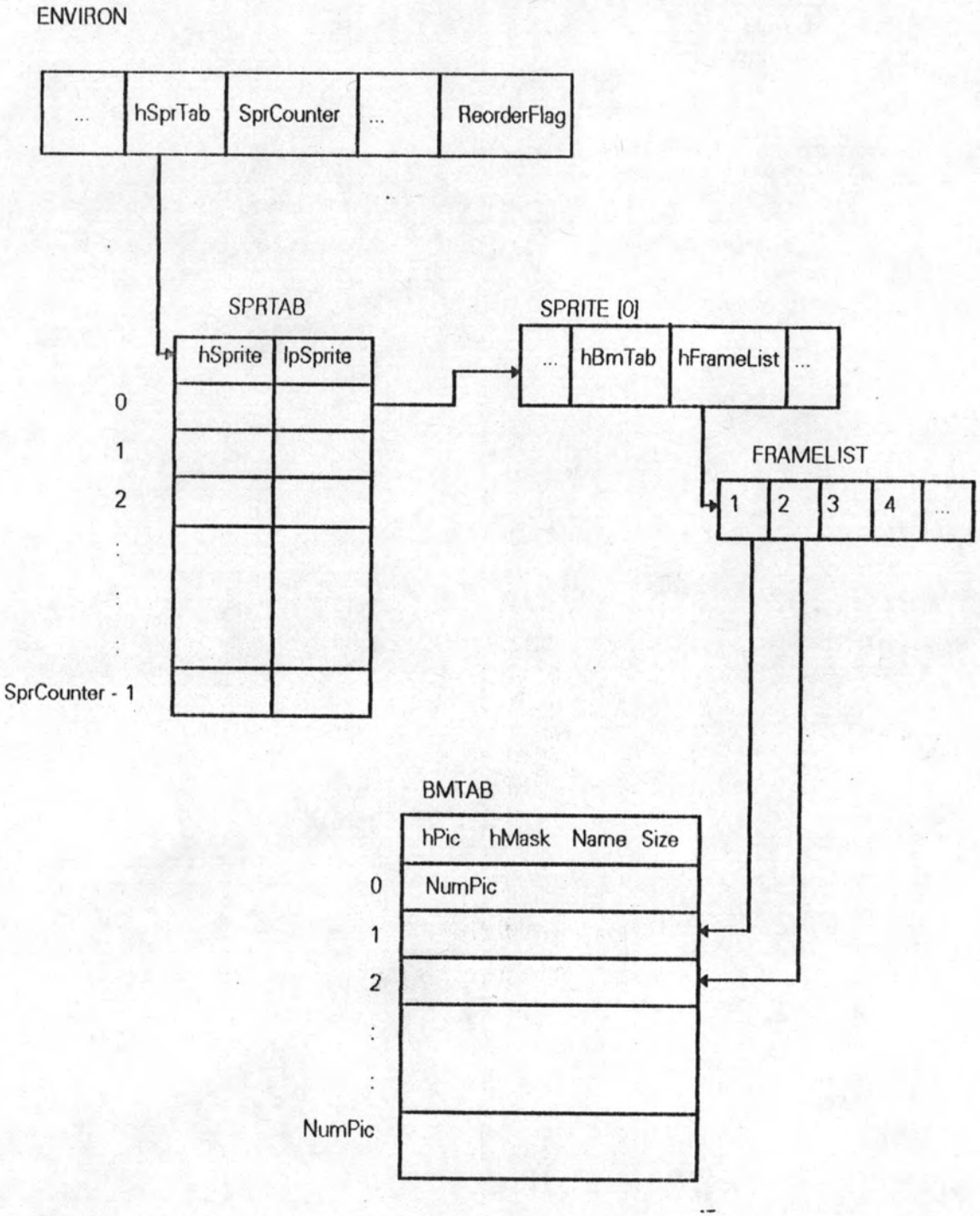
```
typedef struct      tagEnvironment {
    HWND    hDispWnd;           // ตัวนี้ชี้วินโดว์ที่ดำเนินงานอยู่
    HBITMAP hBkGd;              // ตัวนี้ชี้ภาพพื้นหลังบนวินโดว์
    int     left,top,right,bottom;  // ขนาดของวินโดว์
    int     bmWidth, bmHeight;  // ขนาดของภาพพื้นหลังบนวินโดว์
    HBITMAP image;              // ตัวนี้ชี้ภาพบิตแมพของภาพพื้นหลัง
    int     imWidth, imHeight;  // ขนาดของภาพพื้นหลัง
    char    imageFile;          // ภาพพื้นหลังมาจาก image file หรือไม่
    char    BkSize;             // ขนาดของภาพพื้นหลังเท่าเดิมตามในแฟ้ม
                                   ข้อมูลหรือไม่
```

```
    UINT        nRunID;          // หมายเลข Running Number ที่ให้แก่สไปรต์
    HSPRTAB     hSprTab;         // ดัชนีชี้คลังเก็บสไปรต์
    int         SprCounter;      // ตัวนับจำนวนสไปรต์
    char        ReOrderFlag;     // ตัวบ่งชี้การเรียงลำดับสไปรต์ในคลังสไปรต์
} ENVIRON;

typedef struct      tagSprTable {
    HSPRITE     hSprite;         // ดัชนีชี้โครงสร้างข้อมูลของสไปรต์
    LPSPRITE    lpSprite;        // ตำแหน่งที่อยู่ในหน่วยความจำของสไปรต์
} SPRTAB;                        // ตัวจักรทำภาพเคลื่อนไหวเป็นผู้ใช้
```

จากโครงสร้างข้อมูลของสภาพแวดล้อมในการทำภาพเคลื่อนไหวจะเห็นว่ามีการเก็บข้อมูลเกี่ยวกับภาพพื้นหลัง เก็บข้อมูลเกี่ยวกับสไปรต์ทุกตัวไว้ในโครงสร้างข้อมูลแบบแถวลำดับโดยใช้ดัชนีชี้แถวลำดับของโครงสร้างข้อมูลสไปรต์และยังมีข้อมูลที่สำคัญอีกตัวหนึ่งคือ ReOrderFlag ซึ่งเป็นข้อมูลที่ใช้บ่งบอกสถานะของระบบว่าต้องมีการจัดเรียงลำดับของสไปรต์ในคลังสไปรต์หรือไม่

กรณีที่สไปรต์มีการเปลี่ยนระดับชั้นในการเคลื่อนที่ (เปลี่ยน plane) จะต้องมีการเรียงลำดับของสไปรต์ในคลังเก็บสไปรต์เสียก่อน โดยจะตรวจสอบที่โครงสร้างข้อมูลสภาพแวดล้อมของระบบ (ENVIRON) ว่าเขตข้อมูลชื่อ ReOrderFlag มีค่าเป็น 1 หรือ 0 ถ้ามีค่าเป็นหนึ่งหมายถึงสไปรต์เหล่านั้นมีการเปลี่ยนระดับชั้นในการเคลื่อนไหว จะต้องทำการเรียงลำดับของสไปรต์ในคลังเก็บสไปรต์เสียก่อนแล้วจึงเข้าสู่วงวนการสร้างภาพเคลื่อนไหว แต่ถ้าเขตข้อมูลชื่อ ReOrderFlag มีค่าเป็น 0 ก็ไม่ต้องเรียงลำดับสไปรต์ การจัดเรียงลำดับสไปรต์นั้นใช้ขั้นตอนวิธีการเรียงลำดับแบบ Insertion Sort โดยใช้ CurZ เป็นกุญแจหลักและ sId เป็นกุญแจรอง

โครงสร้างข้อมูลที่ออกแบบมีความสัมพันธ์กันดังรูปต่อไปนี้



รูปที่ 3.7 แสดงความสัมพันธ์ระหว่างโครงสร้างข้อมูลในระบบ

การสร้างสภาพแวดล้อม

ในแต่ละวินโดว์ที่มีสไปรต์เป็นสมาชิกจะต้องมีการจัดการสภาพแวดล้อมต่างๆซึ่งจะเป็นข้อมูลที่จำเป็นสำหรับกลไกในการทำภาพเคลื่อนไหวตามที่ได้ออกแบบโครงสร้างข้อมูลของสภาพแวดล้อมไปแล้วนั้น การสร้างสภาพแวดล้อมของโปรแกรมจะกระทำในส่วนเริ่มต้นของโปรแกรม โดยมีขั้นตอนดังนี้

การสร้างสภาพแวดล้อม :

1. จองหน่วยความจำ สำหรับโครงสร้างข้อมูล ENVIRON
2. สร้างแถวลำดับของโครงสร้างข้อมูล SPRTAB ขนาด 0 ไบต์
   (เพื่อขอแฮนเดิลสำหรับ SPRTAB)
3. เก็บค่าแฮนเดิลสำหรับวินโดว์นี้
4. ส่งค่าแฮนเดิลสำหรับโครงสร้างข้อมูล ENVIRON กลับไปให้โปรแกรมที่เรียกมา

จบการสร้างสภาพแวดล้อม :

เมื่อจบงานจะได้หน่วยความจำสำหรับเก็บข้อมูลสภาพแวดล้อมซึ่งเป็นค่าโดยปริยาย และเนื่องจากการสร้างสภาพแวดล้อมเป็นงานเริ่มต้นจึงยังไม่มีสไปรต์เป็นสมาชิก (ยังไม่มีการสร้างสไปรต์) ทำให้จำนวนสไปรต์มีค่าเป็น 0 แต่อย่างไรก็ตามจะต้องสร้างแถวลำดับสำหรับเก็บสไปรต์เอาไว้ ในที่นี้ได้ออกแบบให้มีการจองหน่วยความจำขนาด 0 ไบต์ (ซึ่งวินโดวส์อนุญาตให้ทำได้) เมื่อถึงเวลาสร้างสไปรต์ก็จะมาขยายหน่วยความจำที่ใช้แฮนเดิลอันนี้อีกที ดังนั้นเมื่อมีการสร้างสไปรต์แถวลำดับเก็บสไปรต์ (SPRTAB) ในโครงสร้างข้อมูล ENVIRON จะขยายออก ถ้ามีการยกเลิกสไปรต์แถวลำดับนี้ก็จะมีการลดขนาดลง การออกแบบเช่นนี้อาจทำให้เสียเวลาในการขยายและลดขนาดของแถวลำดับ (ต้องขอหน่วยความจำใหม่หรือ ReAllocate memory) แต่จะทำให้การดำเนินงานของกลไกทำภาพเคลื่อนไหวเร็วและไม่ซับซ้อนเมื่อเทียบกับการใช้ Link List เพราะในระบบนี้ให้ความสำคัญกับความเร็วของการทำภาพเคลื่อนไหวมากกว่าการสร้างหรือยกเลิกสไปรต์ซึ่งเป็นงานเริ่มโปรแกรม

การสร้างคลังภาพ

เมื่อมีการสร้างสภาพแวดล้อมของวินโดว์เรียบร้อยแล้วก็จะมีการสร้างสไปรต์ แต่การสร้างสไปรต์นั้นจะต้องดึงภาพจากคลังภาพหรือฐานข้อมูลภาพที่จัดเตรียมไว้ ในโปรแกรมจึงต้องมีการสร้างคลังภาพกันก่อน โดยที่ภาพเหล่านั้นถูกสร้างด้วยโปรแกรมสร้างภาพและมีการบันทึกไว้ในรูปแบบที่ระบบวินโดวส์สามารถอ่านขึ้นมาได้ (Resource compiler รู้จัก) ตามปกติจะอยู่ในรูปแบบ .BMP ในการ

สร้างคลังภาพจะระบุชื่อของภาพบิตแมพไว้ในแถวลำดับของโครงสร้างข้อมูล PICMASK ซึ่งมีรายละเอียดดังนี้

```
typedef  struct  tagPicMask  {
        LPSTR    PicName;              // ชื่อแฟ้มข้อมูลภาพ
        LPSTR    MaskName;             // ชื่อแฟ้มข้อมูลภาพเงา
} PICMASK;
```

ชื่อของภาพบิตแมพนั้นจะต้องตรงกับที่ระบุไว้ใน Resource File (.RC) เพื่อให้ resource compiler สามารถสร้างโปรแกรมสำหรับอ่านแฟ้มข้อมูลภาพได้ เมื่ออ่านเข้ามาแล้วจะเก็บแฮนเดิลสำหรับภาพบิตแมพไว้ในโครงสร้างข้อมูล BMTAB ขั้นตอนในการสร้างคลังภาพมีดังต่อไปนี้

การสร้างคลังภาพ :

1. นับจำนวนแถวลำดับของโครงสร้างข้อมูล PICMASK ที่ได้รับมา
2. จองหน่วยความจำสำหรับแถวลำดับของ BMTAB เท่าจำนวนที่นับได้บวก 1
3. โหลดภาพบิตแมพตามชื่อใน PicName
4. โหลดภาพเงาตามชื่อใน MaskName
   ถ้าโหลดไม่ได้ ให้เรียกฟังก์ชันสร้างภาพเงา จากคลังโปรแกรม
5. วนทำข้อ 3 ถึงข้อ 4 จนครบทุกภาพ
6. นำจำนวนภาพในคลังภาพไปใส่ในแถวลำดับของ BMTAB ตัวแรก (BMTAB[0])
7. นำแฮนเดิลของภาพบิตแมพทั้งหมดไปใส่ในแถวลำดับ BMTAB จากตัวที่ 1 ถึงตัวสุดท้าย

จบงาน :

<u>การสร้างสไปรต์</u>

เมื่อมีการจัดเตรียมสภาพแวดล้อมและคลังภาพเรียบร้อยแล้ว ขั้นต่อมาจึงจะสร้างสไปรต์ได้ การสร้างสไปรต์นั้นจะจองหน่วยความจำสำหรับเก็บข้อมูลตามโครงสร้าง SPRITE โดยภาพของสไปรต์จะเลือกจากคลังภาพที่มีอยู่ การเลือกคลังภาพมาใช้กับสไปรต์นั้นได้ออกแบบให้ระบุชื่อของภาพ (PicName) ลงไปในแถวลำดับของโครงสร้างข้อมูล FRAME ซึ่งมีรายละเอียดดังนี้

```
typedef  struct  tagFrame  {
        LPSTR    Frame;          // ชื่อภาพที่ต้องการ
        char     Status;         // รูปแบบของภาพที่จะนำมาใช้เช่นกลับหัว, กลับด้าน
} FRAME;
```

การเลือกภาพจากคลังภาพทำได้โดยใส่ชื่อภาพที่ต้องการ จะเป็นภาพเดียวหรือหลายภาพก็ได้ จากนั้นจึงนำไปใช้กับฟังก์ชันการสร้างสไปรต์ ซึ่งมีขั้นตอนดังนี้

การสร้างสไปรต์ :

1. จองหน่วยความจำสำหรับโครงสร้างข้อมูล SPRITE
2. เลือกภาพจากคลังภาพตามที่ระบุในแถวลำดับของโครงสร้าง FRAME
3. กำหนดค่าโดยปริยายและค่าที่ระบุมา ให้กับโครงสร้างข้อมูลสไปรต์นี้
4. เพิ่มจำนวนสไปรต์ในโครงสร้างข้อมูล ENVIRON
5. ขยายแถวลำดับของโครงสร้างข้อมูล SPRTAB สำหรับเก็บแฮนเดิลของสไปรต์
6. ให้กำเนิดวินโดว์ลูกสำหรับสไปรต์ โดยไม่ต้องแสดงวินโดว์
7. ใช้ฟังก์ชัน SetWindowWord() ใน API ของวินโดวส์ เพื่อนำค่าของสไปรต์ไปใส่ ทำให้ฟังก์ชันประจำสไปรต์อ้างอิงค่าแฮนเดิลของสไปรต์ได้
8. ส่งข้อความ SPR_MSG_CREATE ไปให้ฟังก์ชันประจำสไปรต์รับรู้

จบงาน :

<u>การออกแบบตัวจักรทำภาพเคลื่อนไหว</u>

ขั้นตอนวิธีในการทำภาพเคลื่อนไหวเป็นสิ่งสำคัญมากพอๆกับการออกแบบโครงสร้างข้อมูลต่างๆของระบบการทำภาพเคลื่อนไหว ถ้าในระบบมีสไปรต์เพียงตัวเดียวขั้นตอนวิธีจะไม่ยุ่งยาก เมื่อมีสไปรต์หลายตัวเกิดขึ้นในระบบขั้นตอนวิธีในการจัดแสดงภาพสไปรต์จะค่อนข้างยาก และจะวุ่นวายมากเมื่อสไปรต์ทั้งหลายเหล่านั้นมีการชนกัน เคลื่อนที่ผ่านกัน สไปรต์ตัวนั้นเคลื่อนที่มาบังตัวนี้ หรือแม้กระทั่งการเคลื่อนที่วนรอบสไปรต์ตัวอื่นๆ ตัวจักรทำภาพเคลื่อนไหวจะต้องมีวิธีแก้ปัญหาเหล่านี้

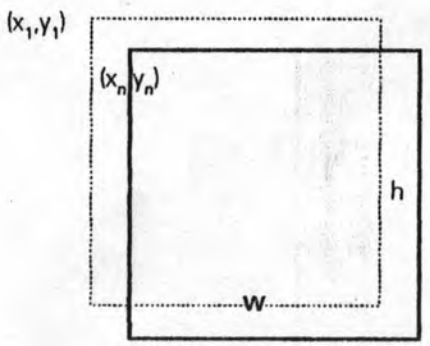
การออกแบบตัวจักรทำภาพเคลื่อนไหวนี้พยายามใช้ขั้นตอนวิธีที่มีการทำสำเนาข้อมูลจากจอภาพให้น้อยที่สุด โดยการจัดการภาพบิตแมพต่างๆในหน่วยความจำหลักที่เตรียมไว้ให้เรียบร้อยก่อนแล้วจึงทำสำเนาไปที่จอภาพ วิธีการนี้เรียกว่า Offscreen Bitmap Technique ซึ่งสามารถลดการกระพริบ

ของภาพที่เกิดขึ้นบนวินโดว์ได้ (Barkakati, Nabajyoti 1993) หลักการดำเนินการระดับบิตและขั้นตอนวิธีในการจัดเรียงลำดับข้อมูลถูกนำมาใช้ในการทำงานของตัวจักรทำภาพเคลื่อนไหวด้วย

ขั้นตอนวิธีของตัวจักรทำภาพเคลื่อนไหว

เมื่อสไปรต์มีการเคลื่อนที่จากตำแหน่งหนึ่ง (x_1,y_1) ไปยังอีกตำแหน่งหนึ่ง (x_n,y_n) วิธีการพื้นฐานในการทำให้เห็นว่าสไปรต์เคลื่อนที่ไปได้นั้นคือการลบภาพที่ตำแหน่งเดิมแล้ววาดภาพที่ตำแหน่งใหม่ ในกรณีที่เคลื่อนที่ไปบนบิตแมพของภาพพื้นหลังนั้น การลบภาพของสไปรต์ที่ตำแหน่งเดิมจะทำให้ภาพของพื้นหลังถูกลบไปด้วย ดังนั้นจึงต้องมีการทำสำเนาบิตแมพของพื้นหลังเก็บไว้ก่อน เมื่อใดก็ตามที่ต้องการลบภาพของสไปรต์ก็นำส่วนของพื้นหลังที่เก็บไว้นั้นมาปะ (blitting) ลงไปในตำแหน่งที่ต้องการจะลบสไปรต์ โดยระบุตำแหน่งและขนาดกว้างxยาวของพื้นที่ที่ต้องการลบ (ในกรณีนี้คือตำแหน่งเดิมและความกว้างxสูงของสไปรต์นั่นเอง)

การวาดภาพของสไปรต์ลงไปยังตำแหน่งใหม่ทำได้โดย blitting บิตแมพที่เป็นเงาของสไปรต์ (mask) ลงไปที่ตำแหน่งนั้นโดยใช้ Raster operator แบบ AND และตามด้วย blitting บิตแมพที่เป็นรูปของสไปรต์บนพื้นดำลงไปที่ตำแหน่งเดียวกันด้วย Raster operator แบบ OR ผลลัพธ์ที่ได้ทำให้ภาพของสไปรต์มีลักษณะเป็น Transparent



รูปที่ 3.8 แสดงการเปลี่ยนตำแหน่งของสไปรต์

จากรูปที่ 3.8 สไปรต์ขนาดกว้าง w และสูง h เดิมอยู่ที่ตำแหน่ง (x_1,y_1) ต่อมาเคลื่อนที่ไปยังตำแหน่งใหม่ (x_n,y_n) วิธีการขั้นพื้นฐานที่ทำให้สไปรต์เคลื่อนที่จากตำแหน่ง (x_1,y_1) ไปยังตำแหน่ง (x_n,y_n) ได้แก่

1. ลบภาพที่ตำแหน่ง (x_1,y_1)

   โดยนำบิตแมพของพื้นหลัง (ที่เก็บไว้ก่อนแล้ว) ที่ตำแหน่งและขนาดเท่ากับสไปรต์มาปะลงไป

2. blitting บิตแมพที่เป็นเงาของสไปรต์ (mask) ลงไปยังตำแหน่งใหม่ (x_n,y_n)

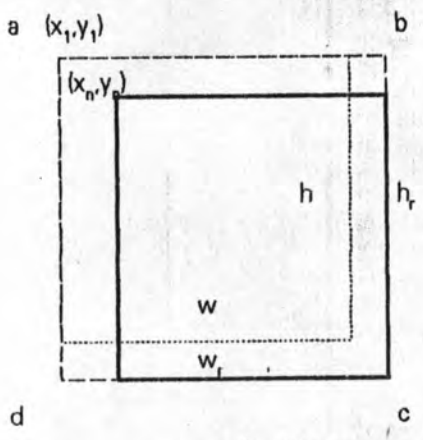
   โดยใช้ Raster operator แบบ AND

3. blitting บิตแมพที่เป็นภาพของสไปรต์บนพื้นดำลงไปที่ตำแหน่งเดียวกันกับเงา (x_n,y_n)

   โดยใช้ Raster operator แบบ OR

จากวิธีการขั้นพื้นฐานนี้เป็นแบบลบแล้ววาดใหม่ มีข้อดีคือง่าย แต่มีข้อเสียคือเกิดการกระพริบเนื่องจากการลบภาพที่ตำแหน่งเดิมแล้ววาดสไปรต์ที่ตำแหน่งใหม่ซึ่งจะต้องมีการ update บนจอภาพถึง 3 ครั้ง (blitting พื้นหลัง,mask และภาพสไปรต์ โดยเฉพาะเมื่อ blitting mask ลงไปจะเห็นเป็นสีขาว) ปัญหาการกระพริบนั้นสามารถแก้ไขได้โดยนำเทคนิค Offscreen Bitmap มาประยุกต์ใช้ หลักการพื้นฐานคือ การจัดสรรหน่วยความจำขึ้นมาทำหน้าที่แทนจอภาพ จากนั้นทำการ update รายละเอียดต่างๆที่ต้องการแสดงบนจอภาพลงบนหน่วยความจำส่วนนี้ให้เรียบร้อยแล้วจึงทำสำเนาไปยังจอภาพ ด้วยหลักการนี้จะเห็นว่าเป็นการ update ข้อมูลบนจอภาพเพียงครั้งเดียวและขจัดปัญหาเกิดการกระพริบของภาพสไปรต์ที่มีการเปลี่ยนแปลงได้

จากเทคนิคแบบ Offscreen Bitmap ดังกล่าว ถ้าเป็นการสำเนาข้อมูลทั้งจอภาพอาจจะทำให้การเปลี่ยนแปลงรายละเอียดต่างๆบนจอภาพเป็นไปได้ช้า (เนื่องจากต้อง blitting ข้อมูลจำนวนมาก) ดังนั้นจึงนำวิธีการนี้มาประยุกต์ใช้โดยจะทำการ update ข้อมูลเฉพาะพื้นที่ที่มีการเปลี่ยนแปลงเท่านั้น ซึ่งพื้นที่ที่มีการเปลี่ยนแปลงคือพื้นที่ที่ต้องการลบและพื้นที่ที่ต้องวาดใหม่ ด้วยวิธีการนี้จะช่วยให้การ update ข้อมูลบนจอภาพทำได้เร็วขึ้น โดยมีวิธีการดังต่อไปนี้



รูปที่ 3.9 แสดงการคำนวณพื้นที่ที่ใช้ในการ update ในหน่วยความจำ

จากรูปที่ 3.9 สไปรต์มีขนาดกว้าง w สูง h เดิมอยู่ที่ตำแหน่ง (x_1,y_1) ต้องการย้ายไปยังตำแหน่งใหม่คือ (x_n,y_n) มีขั้นตอนจัดการดังนี้

1. คำนวณหาพื้นที่ที่ครอบสไปรต์ทั้งตำแหน่งเดิมและตำแหน่งใหม่ จากในรูปที่ 3.9 คือหาพื้นที่ของสี่เหลี่ยม abcd ดังนี้

x_r = min (x, x_n);

y_r = min (y, y_r);

w_r = max (x, x_n) + w;

h_r = max (y, y_n) + h;

2. จัดสรรหน่วยความจำแทนจอภาพ (hidden page) ขนาดกว้าง w_r สูง h_r ตามที่คำนวณได้ดังนี้

```
hDC = GetDC (NULL);
hmDC = CreateCompatibleDC (hDC);
hBmp = CreateCompatibleBitmap (hDC, w_r, h_r);
SelectObject (hmDC, hBmp);
ReleaseDC (NULL, hDC);
```

3. Blitting บิตแมพของภาพพื้นหลังที่เก็บไว้ ณ ตำแหน่ง (x_r,y_r) กว้าง w_r สูง h_r ไปยังหน่วยความจำแทนจอภาพ (hmDC)

```
lpEnv = (LPENVIRON) GlobalLock(hEnv);
hBkDC = CreateCompatibleDC (hDC);
SelectObject (hBkDC, lpEnv->hBKGD);
BitBlt (hmDC, 0,0,w_r,h_r, hBkDC, x_r,y_r, SRCCOPY);
```

4. Blitting บิตแมพ mask ของสไปรต์ไปยัง hmDC ณ ตำแหน่งใหม่ ดังนี้

x_m = x_n - x_r;

y_m = y_n - y_r;

```
BitBlt (hmDC, x_m, y_m, w, h,  hMask, 0, 0, SRCAND);  // AND Operation
```

5. Blitting บิตแมพภาพของสไปรต์ (hPic) ไปยัง hmDC ณ ตำแหน่งเดียวกันกับ mask

```
BitBlt (hmDC, x_m, y_m, w,h,  hPic, 0, 0, SRCPAINT);   // OR Operation
```

เมื่อถึงขั้นตอนนี้จะได้ hmDC ที่มีสไปรต์อยู่บนภาพพื้นหลังที่สมบูรณ์

6. Blitting หน่วยความจำ hmDC ที่ได้ไปยังจอภาพ ณ ตำแหน่ง (x_r , y_r) ขนาดกว้าง w_r สูง h_r

```
BitBlt (hDC, x_r, y_r, w_r, h_r,  hmDC, 0, 0, SRCCOPY);
```

7. ยกเลิกหน่วยความจำ hmDC เพื่อคืนทรัพยากรให้แก่ระบบ

```
DeleteObject ( SelectObject(hmDC, hBmp) );
DeleteDC(hmDC);
```

จากหลักการนี้ใช้ได้ดีกับระบบที่จัดการกับสไปรต์จำนวนไม่มาก กรณีที่มีสไปรต์หลายตัวและมีการเคลื่อนที่สวนกันซึ่งจะต้องมีการวาดภาพของสไปรต์ทับกัน (overlapping) จะเห็นได้ว่าเกิดการวนรอบการ update บนจอภาพในตำแหน่งเดียวกันมากกว่า 1 ครั้ง (เนื่องจากสไปรต์ทับกัน) ดังนั้นจึงปรับปรุงวิธีการที่ผ่านมาอีกเล็กน้อยคือ มีการวาดสไปรต์ตามลำดับการขั้นเคลื่อนที่ของสไปรต์ (สามารถเปลี่ยนแปลงได้) และหาวิธีการลดจำนวนการ update บนจอภาพเมื่อมีสไปรต์มากกว่า 1 ตัวเคลื่อนที่ทับกัน (จากวิธีที่ผ่านมาจะ update จอภาพเท่ากับจำนวนสไปรต์ที่มี ไม่ว่าจะเคลื่อนที่มาทับกันหรือไม่ก็ตาม) โดยใช้หลักการดังนี้

1. นำสไปรต์ที่อยู่ในแถวลำดับ (อยู่ในโครงสร้าง ENVIRON) มาตรวจสอบกับสไปรต์ตัวอื่นๆว่ามีการซ้อนทับกันหรือไม่ ถ้ามีการทับกันให้กำหนดค่าสถานภาพของสไปรต์เหล่านั้นเป็น Overlapped (เพื่อใช้ในการวาดลงหน่วยความจำ) และคำนวณหาพื้นที่ที่ครอบสไปรต์ที่ทับกันทั้งหมด (เพื่อใช้ในการ update บิตแมพของภาพพื้นหลัง)

2. ทำสำเนาบิตแมพของภาพพื้นหลัง ณ ตำแหน่งและขนาดตามที่คำนวณได้ในข้อที่ 1 ไปยังหน่วยความจำแทนจอภาพ

3. วาดสไปรต์ทุกตัวที่มีสถานภาพเป็น Overlapped ลงบนหน่วยความจำแทนจอภาพ

4. ทำสำเนาหน่วยความจำแทนจอภาพไปยังจอภาพ ณ ตำแหน่งและขนาดที่คำนวณได้ในข้อที่ 1

5. วนทำข้อ 1 - 4 จนครบทุกตัว

จากวิธีการดังกล่าวนี้ กรณีที่ใช้ได้ดีที่สุดเมื่อมีสไปรต์มากกว่า 1 ตัว อยู่ในตำแหน่งที่ซ้อนทับกันหมดทุกตัว

ต่อไปนี้เป็นขั้นตอนการทำงานของตัวจักรทำภาพเคลื่อนไหวที่ประยุกต์จากหลักการที่กล่าวมาแล้ว

Function amAnimate ()

Parameter hEnvironment

Begin

ตรวจสอบสภาพแวดล้อมต่างๆ

- วินโดว์ที่แสดงสไปรต์เป็นไอคอนหรือไม่

ถ้าเป็นไอคอนให้กลับไปฟังก์ชันที่เรียกมา

- ต้องการเรียงลำดับของสไปรต์หรือไม่

ถ้าต้องการ ให้เรียกฟังก์ชันการเรียงลำดับสไปรต์ (amSortSprTab)

LOOP (1) : สำหรับสไปรต์ทุกตัว

ตรวจสอบสถานภาพของสไปรต์

- ถ้าเป็น Invisible (ไม่ปรากฏตัว)

กลับไปวนที่ LOOP (1)

- ถ้าเป็น Update (มีการเคลื่อนที่ หรือ เปลี่ยนภาพ)

1. คำนวณพื้นที่ที่ครอบสไปรต์ทั้งตำแหน่งเดิมและตำแหน่งใหม่

```
Bx = min (CurX, LastX);
By = min (CurY, LastY);
Ex = max (CurX, LastX) + bmWidth;
Ey = max (CurY, LastY) + bmHeight;
Wr = Ex - Bx + 1;
Hr = Ey - By + 1;
```

2. เปลี่ยนสถานภาพของสไปรต์เป็น Overlapped เพื่อใช้ในการวาดต่อไป

3. นำพื้นที่ที่คำนวณได้จากข้อ 1 ไปตรวจสอบการซ้อนทับกับสไปรต์ตัวอื่น

LOOP(2) : สำหรับสไปรต์ทุกตัว

ตรวจสอบสถานภาพของสไปรต์

- ถ้า สถานภาพเป็น Update และ ไม่ Overlapped

- คำนวณพื้นที่ที่ครอบสไปรต์ทั้งตำแหน่งเดิม

และตำแหน่งใหม่ ด้วยวิธีการเดียวกับข้อ 1

- นำพื้นที่นี้ไปตรวจสอบว่ามีการทับกับพื้นที่ (Bx,By,Ex,Ey) หรือไม่ ถ้าทับกัน ให้ขยายพื้นที่ (Bx,Bx,Ex,Ey) ให้ครอบทั้ง 2 พื้นที่คือครอบทั้ง(Bx,By,Ex,Ey)เดิมและสไปรต์ตัวที่ตรวจสอบ

End LOOP (2)

4. นำพื้นที่ที่คำนวณได้จากข้อ 3 มาตรวจสอบว่า อยู่ในพื้นที่ของวินโดว์ หรือไม่
   - ถ้าไม่อยู่ในวินโดว์เลย ให้ไปวน LOOP (1) ต่อไป
   - ถ้าอยู่ในวินโดว์ ให้คำนวณพื้นที่เฉพาะที่จะต้องวาดภาพใน วินโดว์เท่านั้น

```
Wr = min (Bx+Wr, Env->left+Env->Width) -
     max (Bx, Env->left);
Hr = min (By+Hr, Env->top+Env->Height) -
     max (By, Env->top);
Bx = max (Bx, Env->left);
By = max (By, Env->top);
```

5. นำพื้นที่ที่คำนวณได้จากข้อ 2.4 มาตรวจสอบกับสไปรต์ที่ไม่ต้องการ Update (หรือสไปรต์ที่อยู่เฉยๆไม่เคลื่อนที่ไม่เปลี่ยนภาพ) ถ้ามีการซ้อนทับกัน ให้กำหนดสถานภาพของสไปรต์ตัวนั้นเป็น Overlapped เพื่อใช้ในการวาดใหม่ (เนื่องจากถูกพื้นที่ดังกล่าวทับบางส่วนหรืออาจถูกทับทั้งตัวก็ได้)
6. สำเนาบิตแมพของภาพพื้นหลัง (ที่เก็บไว้แล้ว) ณ ตำแหน่งและขนาด กว้างxสูง ตามที่คำนวณได้มาที่หน่วยความจำแทนจอภาพ (hmDC)
7. วาดสไปรต์ทุกตัวที่มีสถานภาพเป็น Overlapped ลงไปที่ hmDC ที่ตำแหน่งของสไปรต์โดยเทียบกับพื้นที่ของ hmDC

```
Bxo = CurX - Bx;
Byo = CurY - By;
BitBlt (hmDC,  Bxo, Byo, bmWidth, bmHeight,
```

```
                    hMask,  0, 0,  SRCAND);
                BitBlt (hmDC,  Bxo, Byo, bmWidth, bmHeight,
                    hPic,    0, 0,  SRCPAINT);
```

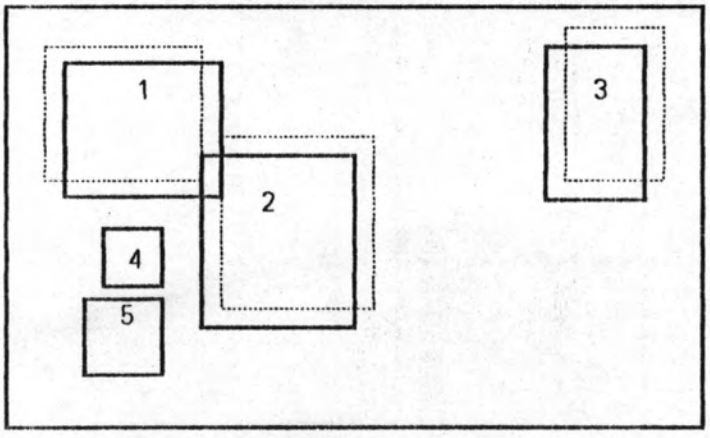
8. สำเนา hmDC ที่ได้ไปยังจอภาพที่ตำแหน่ง Bx, By

```
                BitBlt (hDC, Bx, By, Wr, Hr,  hmDC, 0, 0, SRCCOPY);
End LOOP (1)
```

คืนทรัพยากรให้แก่ระบบ

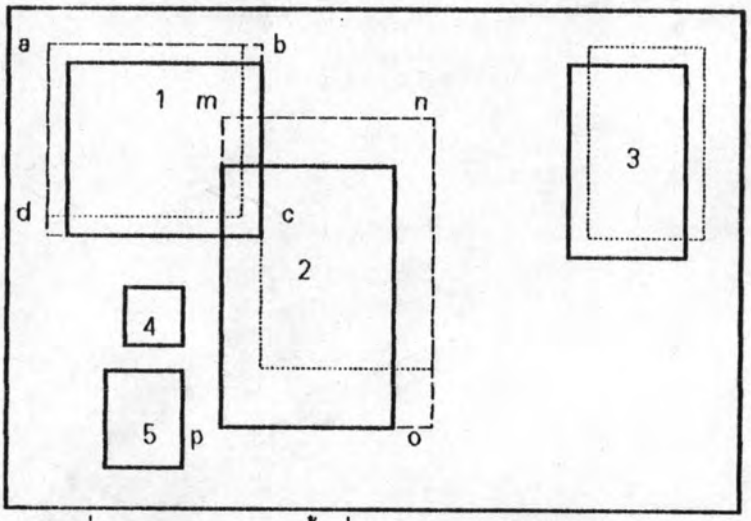
End Function (amAnimate)

รูปต่อไปนี้แสดงตัวอย่างการหาพื้นที่ในการ Update จอภาพ



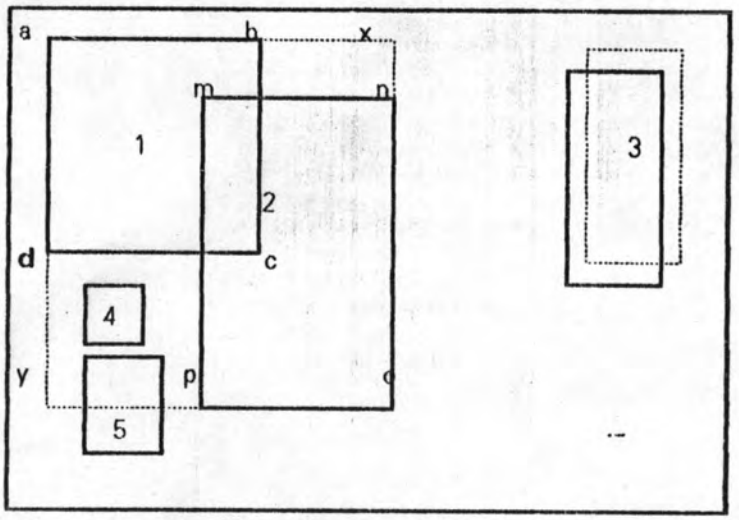
รูปที่ 3.10 แสดงตัวอย่างสไปรต์ที่ซ้อนทับกัน

จากรูปที่ 3.10 มีสไปรต์ 5 ตัว ตามหมายเลข 1,2,3,4,5 โดยที่ สไปรต์ 1,2,3 มีการเคลื่อนที่ เส้นประคือตำแหน่งเดิมของสไปรต์ ส่วนสไปรต์ 4,5 ไม่มีการเคลื่อนที่



รูปที่ 3.11 แสดงการหาพื้นที่ Update ของสไปรต์

จากรูปที่ 3.11 เป็นการหาพื้นที่ในการ update จอภาพสำหรับสไปรต์หมายเลข 1 พื้นที่นี้ได้แก่ สี่เหลี่ยม abcd จากนั้นนำสี่เหลี่ยม abcd ไปตรวจสอบว่ามีการทับกับสไปรต์ตัวอื่นหรือไม่ ในรูปที่ 3.11 เป็นสไปรต์หมายเลข 2 โดยหาพื้นที่ในการ update ของสไปรต์หมายเลข 2 ได้สี่เหลี่ยม mnop จากนั้นนำสี่เหลี่ยมทั้ง 2 คือ สี่เหลี่ยม abcd และ สี่เหลี่ยม mnop มาตรวจสอบว่าทับกันหรือไม่ ในรูปที่ 3.11 จะเห็นว่ามีการทับกัน ดังนั้นจึงต้องขยายสี่เหลี่ยม abcd ให้ครอบสี่เหลี่ยม mnop ซึ่งจะได้ผลลัพธ์ดังนี้



รูปที่ 3.12 แสดงการหาพื้นที่ที่ต้อง update เมื่อมีสไปรต์ทับกันหลายตัว

จากรูปที่ 3.12 สี่เหลี่ยม abcd เป็นพื้นที่ที่ต้อง update ของ สไปรต์หมายเลข 1 และสี่เหลี่ยม mnop เป็นพื้นที่ที่ต้อง update ของสไปรต์หมายเลข 2 เมื่อนำไปตรวจสอบปรากฏว่ามีการซ้อนทับกัน จึงต้องคำนวณพื้นที่ที่ครอบพื้นที่การ update ของทั้ง 2 สไปรต์ตามในรูปจะได้สี่เหลี่ยม axoy

เมื่อได้สี่เหลี่ยม axoy แล้วนำสี่เหลี่ยมนี้ไปตรวจสอบกับสไปรต์ตัวอื่นต่อไปถ้าทับกันอีกก็ใช้วิธีการเดียวกันในการขยายพื้นที่ของสี่เหลี่ยม จากรูปจะเห็นว่า สไปรต์ที่มีการเคลื่อนที่คือ 1,2,3 มีเพียง 1 และ 2 เท่านั้นที่ทับกัน ดังนั้นจึงไม่ขยายสี่เหลี่ยมไปครอบสไปรต์หมายเลข 3 สำหรับสไปรต์หมายเลข 4 และ 5 นั้นถูกสี่เหลี่ยม axoy ทับ แต่สไปรต์ทั้งสองไม่มีการเคลื่อนที่และไม่เปลี่ยนภาพ จึงมีความจำเป็นเพียงวาดใหม่ที่ตำแหน่งเดิมเท่านั้น

จากขั้นตอนวิธีดังกล่าวนี้ จะเห็นว่าการตรวจสอบการซ้อนทับกันของสไปรต์เพื่อคำนวณหาพื้นที่ที่ต้องการ update บนจอภาพนั้น ทำการตรวจสอบกับสไปรต์ทุกตัวในระบบซึ่งอาจทำให้ประสิทธิภาพของระบบลดลงไปได้ ดังนั้นผู้วิจัยจึงได้ปรับปรุงขั้นตอนวิธีดังกล่าวนี้ให้มีประสิทธิภาพขึ้นโดยอาศัยวิธีการเรียงลำดับสไปรต์ตามตำแหน่งในแนวนอนหรือเรียงตาม CurX จากนั้นจึงทำการตรวจสอบการซ้อนทับกันของสไปรต์ในช่วงที่เป็นไปได้เท่านั้น ด้วยวิธีการเช่นนี้สามารถลดจำนวนการตรวจสอบสไปรต์ในระบบได้

ขั้นตอนวิธีต่อไปนี้ได้ทำการปรับปรุงจากวิธีเดิมตามแนวทางที่ได้ออกแบบใหม่

Function amAnimate ()

Parameter hEnvironment

Begin

ตรวจสอบสภาพแวดล้อมต่างๆ

- วินโดว์ที่แสดงสไปรต์เป็นไอคอนหรือไม่

ถ้าเป็นไอคอนให้กลับไปฟังก์ชันที่เรียกมา

- ต้องการเรียงลำดับของสไปรต์ตามระดับการเคลื่อนที่ (CurZ) หรือไม่

ถ้าต้องการ ให้เรียกฟังก์ชันการเรียงลำดับสไปรต์ (amSortSprTab)

เรียงลำดับสไปรต์ตามแนวนอน min (CurX, LastX)

LOOP (1) สำหรับสไปรต์ทุกตัว

ตรวจสอบสถานภาพของสไปรต์

- ถ้าเป็น Invisible (ไม่ปรากฏตัว)

กลับไปวนที่ LOOP (1)

- ถ้าเป็น Update (มีการเคลื่อนที่ หรือ เปลี่ยนภาพ)

1. คำนวณพื้นที่ที่ครอบสไปรต์ทั้งตำแหน่งเดิมและตำแหน่งใหม่

```
Bx = min (CurX, LastX);
By = min (CurY, LastY);
Ex = max (CurX, LastX) + max (bmWidth, LastWidth);
Ey = max (CurY, LastY) + max (bmHeight, LastHeight);
Wr = Ex - Bx + 1;
Hr = Ey - By + 1;
```

2. เปลี่ยนสถานภาพของสไปรต์เป็น Overlapped เพื่อใช้ในการวาดต่อไป
3. นำพื้นที่ที่คำนวณได้จากข้อ 1 ไปตรวจสอบการซ้อนทับกับสไปรต์ตัวอื่น

```
LOOP(2) : สำหรับสไปรต์ทุกตัว
        ถ้า Bx > กรอบภาพด้านขวาของสไปรต์
           วน LOOP(2) ต่อไป
        ถ้า Ex < กรอบภาพด้านซ้ายของสไปรต์
           ออกจาก LOOP(2)
        ตรวจสอบสถานภาพของสไปรต์
        - ถ้า สถานภาพเป็น Update และ ไม่ Overlapped
          - คำนวณพื้นที่ที่ครอบสไปรต์ทั้งตำแหน่งเดิม
            และตำแหน่งใหม่ ด้วยวิธีการเดียวกับข้อ 1
          - นำพื้นที่นี้ไปตรวจสอบว่ามีการทับกับพื้นที่
            (Bx,By,Ex,Ey) หรือไม่ ถ้าทับกัน
            ให้ขยายพื้นที่ (Bx,Bx,Ex,Ey) ให้ครอบทั้ง 2
            พื้นที่คือครอบทั้ง(Bx,By,Ex,Ey)เดิมและ
            สไปรต์ตัวที่ตรวจสอบ --
End LOOP (2)
```

4. นำพื้นที่ที่คำนวณได้จากข้อ 3 มาตรวจสอบว่า อยู่ในพื้นที่ของวินโดว์ หรือไม่
   - ถ้าไม่อยู่ในวินโดว์เลย ให้ไปวน LOOP (1) ต่อไป

- ถ้าอยู่ในวินโดว์ ให้คำนวณพื้นที่เฉพาะที่จะต้องวาดภาย ใน วินโดว์เท่านั้น

```
Wr = min (Bx+Wr, Env->left+Env->Width) -
     max (Bx, Env->left);
Hr = min (By+Hr, Env->top+Env->Height) -
     max (By, Env->top);
Bx = max (Bx, Env->left);
By = max (By, Env->top);
```

5. นำพื้นที่ที่คำนวณได้จากข้อ 2.4 มาตรวจสอบกับสไปรต์ที่ไม่ต้องการ Update (หรือสไปรต์ที่อยู่เฉยๆไม่เคลื่อนที่ไม่เปลี่ยนภาพ) ถ้ามีการซ้อนทับกัน ให้กำหนดสถานภาพของสไปรต์ตัวนั้นเป็น Overlapped เพื่อใช้ในการวาดใหม่ (เนื่องจากถูกพื้นที่ดังกล่าวทับบางส่วนหรือถูกทับทั้งตัวก็ได้)
6. สำเนาบิตแมพของภาพพื้นหลัง (ที่เก็บไว้แล้ว) ณ ตำแหน่งและขนาด กว้างxสูง ตามที่คำนวณได้มาที่หน่วยความจำแทนจอภาพ (hmDC)
7. วาดสไปรต์ทุกตัวที่มีสถานภาพเป็น Overlapped ลงไปที่ hmDC ที่ตำแหน่งของสไปรต์โดยเทียบกับพื้นที่ของ hmDC

```
Bxo = CurX - Bx;
Byo = CurY - By;
BitBlt (hmDC,  Bxo, Byo, bmWidth, bmHeight,
        hMask,  0, 0,  SRCAND);
BitBlt (hmDC,  Bxo, Byo, bmWidth, bmHeight,
        hPic,   0, 0,  SRCPAINT);
```

8. สำเนา hmDC ที่ได้ไปยังจอภาพที่ตำแหน่ง Bx, By

```
BitBlt (hDC, Bx, By, Wr, Hr,  hmDC, 0, 0, SRCCOPY);
```

End LOOP (1)

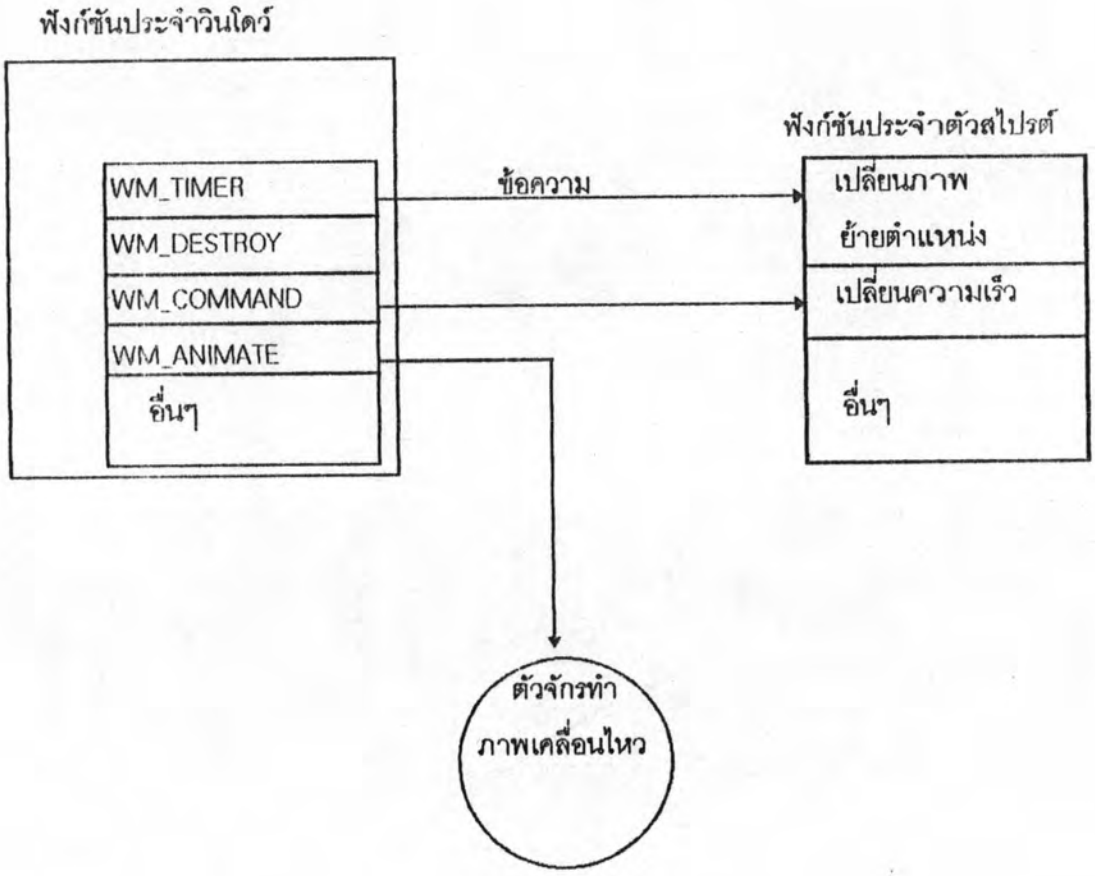
คืนทรัพยากรให้แก่ระบบ

End Function (amAnimate)

จากขั้นตอนวิธีนี้ทำให้การตรวจสอบการซ้อนทับกันของสไปรต์อยู่ในช่วงที่เป็นไปได้ จากรูปที่ 3.12 จะทำการเรียงลำดับสไปรต์ตามแนวนอนหรือตามแกน X จะได้ลำดับของสไปรต์ในการตรวจสอบ คือ สไปรต์หมายเลข 1,4,5,2,3 และเมื่อนำสไปรต์หมายเลข 1 มาทำการตรวจสอบการซ้อนทับกันปรากฏว่า สไปรต์หมายเลข 3 จะไม่ถูกตรวจสอบเลยเนื่องจากพื้นที่ที่จะ update ไม่ทับกับพื้นที่ update ของสไปรต์หมายเลข 3 (กรอบด้านขวาของพื้นที่ update หรือ Ex น้อยกว่า กรอบด้านซ้ายของสไปรต์หมายเลข 3) ดังนั้นถ้ามีสไปรต์อีกหลายตัวอยู่ด้านขวาของกรอบ update (หรือ Ex) และกรอบซ้ายของสไปรต์เหล่านั้นมากกว่ากรอบด้านขวาของพื้นที่ update จะไม่มีการตรวจสอบกับสไปรต์เหล่านั้น (เนื่องจากมีการเรียงลำดับสไปรต์ตามแนวนอนอยู่)

<u>การออกแบบข้อความที่ใช้สื่อสารภายในระบบ</u>

ระบบจัดการทำภาพเคลื่อนไหวนี้ได้ออกแบบให้มีการสื่อสารกันระหว่างสไปรต์กับฟังก์ชันประจำวินโดว์ด้วยการใช้ข้อความซึ่งเป็นวิธีเดียวกันกับที่สภาพปฏิบัติการวินโดวส์ใช้ นอกจากนี้ยังอนุญาตให้มีการใช้ฟังก์ชันตามแบบปกติได้ด้วย เหตุผลที่ออกแบบให้ฟังก์ชันประจำวินโดว์มีการสั่งงานหรือสื่อสารกับสไปรต์ด้วยวิธีการส่งข้อความนั้น เนื่องจากต้องการให้สไปรต์แต่ละตัวมีพฤติกรรมเฉพาะตัวได้ โดยทำงานตามเหตุการณ์ที่เกิดขึ้นหรือตามข่าวสารที่ได้รับ จึงทำให้การเขียนโปรแกรมแบ่งเป็นส่วนของฟังก์ชันประจำวินโดว์กับส่วนที่เป็นฟังก์ชันประจำตัวสไปรต์และมีการสื่อสารกันด้วยข้อความ



รูปที่ 3.13 แสดงการสื่อสารกันระหว่างฟังก์ชันประจำวินโดว์กับสไปรต์

ข้อความหรือข่าวสารที่ใช้ควบคุมการทำงานถูกกำหนดให้มีรูปแบบต่างจากข้อความของระบบวินโดวส์ เพื่อให้จำได้ง่ายและไม่ปะปนกับข้อความของระบบวินโดวส์ โดยข้อความที่กำหนดไว้มีรูปแบบดังนี้

1. เป็นอักษรใหญ่ทั้งหมด
2. ขึ้นต้นด้วย SPR_ ตามด้วยชนิดของข้อความเช่น MSG_ และตามด้วยคำสั่ง เช่น CREATE

ตัวอย่างเช่น SPR_MSG_CREATE SPR_REQ_XYZ

การส่งข้อความไปยังสไปรต์หรือจากสไปรต์ไปยังระบบนั้น จะส่งตามพื้นฐานของฟังก์ชันส่งข้อความของวินโดวส์คือ SendMessage() สำหรับในคลังโปรแกรมได้ออกแบบให้เหมาะสมกับการใช้งานมากขึ้น นั่นคือ SendMessage() จะส่งให้กับฟังก์ชันประจำวินโดว์ แต่ในคลังโปรแกรมมีฟังก์ชันที่ทำหน้าที่นี้ชื่อ amMessageToSprite() จะส่งข้อความไปยังตัวสไปรต์ไม่ใช่ฟังก์ชันประจำตัวสไปรต์ ในกรณี

ที่ต้องการกระจายข้อความนี้ไปให้สไปรต์ทุกตัวในโปรแกรมประยุกต์ก็ใช้ฟังก์ชัน amBroadCastMessage() ได้โดยไม่ต้องใช้วงวนในการส่งข้อความ

วิธีการส่งข้อความจะใช้ข้อมูลในการส่งได้ดังนี้

```
amMessageToSprite(hSprite, Message, wParam, lParam);
```

hSprite = ค่าดัชนีประจำตัวสไปรต์ (ได้จากฟังก์ชันสร้างสไปรต์)

Message = ข้อความที่ต้องการส่ง

wParam และ lParam = ข้อมูลเพิ่มเติมที่ใช้ร่วมกับข้อความที่ส่ง

ตัวอย่างเช่น ต้องการส่งข้อความให้สไปรต์เคลื่อนที่ไปข้างหน้าจำนวน x , y ตำแหน่ง และให้เปลี่ยนภาพไปด้วย มีวิธีส่งดังนี้

amMessageToSprite(สไปรต์, ข้อความให้เคลื่อนที่, คำสั่งให้เปลี่ยนภาพ, จำนวน x ,y จุด);

ข้อความที่ใช้สื่อสารกันภายในระบบได้ออกแบบไว้เป็น 3 ชนิด ได้แก่

1. ข้อความควบคุมการทำงานภายในระบบ
2. ข้อความเปลี่ยนแปลงคุณสมบัติของสไปรต์
3. ข้อความร้องขอข้อมูลต่างๆ

<u>ข้อความควบคุมการทำงานภายในระบบ</u>

ในส่วนของข้อความที่ใช้ควบคุมการทำงานนั้นจะมีรูปแบบ SPR_CTL_command ใช้ในการควบคุมการเคลื่อนไหวของสไปรต์เช่น คำสั่งให้เคลื่อนที่ไปข้างหน้า สั่งให้หยุดเคลื่อนที่ คำสั่งให้เปลี่ยนภาพ ของสไปรต์ เป็นต้น ข้อความชนิดนี้จะใช้ภายในระบบหรือใช้เป็นส่วนหนึ่งของข้อมูลในการเรียกฟังก์ชัน ตัวอย่างเช่น SPR_CTL_FRMSTOP, SPR_CTL_MOVELEFT, SPR_CTL_FRMLOOP เป็นต้น

<u>ข้อความเปลี่ยนแปลงคุณสมบัติของสไปรต์</u>

คุณสมบัติต่างๆของสไปรต์เช่น ตำแหน่งบนจอภาพ เปลี่ยนระดับการเคลื่อนที่ เป็นต้น มีรูปแบบดังนี้ SPR_MSG_setprop ตัวอย่างเช่น SPR_MSG_SETPROP , SPR_MSG_SETFRAME, SPR_MSG_SETPLANE

ข้อความร้องขอข้อมูล

ข้อความชนิดนี้ใช้ในการขอข้อมูลเกี่ยวกับสไปรต์ที่ระบุ เช่น ตำแหน่งของสไปรต์ สถานภาพของสไปรต์ อายุของสไปรต์ ลำดับความสำคัญของสไปรต์ เป็นต้น มีรูปแบบดังนี้ SPR_REQ_request ตัวอย่างเช่น SPR_REQ_XYZ, SPR_REQ_STATUS, SPR_REQ_CURPRIORITY

นอกจากนี้ยังมีข้อความที่ใช้เป็นค่าเริ่มต้นของข้อความอื่นๆในระบบคือ WM_TAMMESSAGE โดยกำหนดไว้ดังนี้

```
#define  WM_TAMMESSAGE     WM_USER+200
```

โดยที่ค่าของ WM_TAMMESSAGE สามารถเปลี่ยนได้ตามความเหมาะสมของผู้ใช้เนื่องจากข้อความที่ใช้ในระบบทำภาพเคลื่อนไหวไม่เกี่ยวกับข้อความที่ใช้ในระบบวินโดวส์