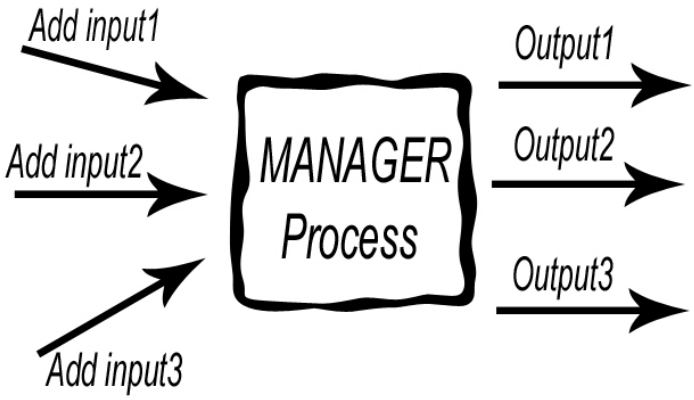
บทที่ 19
ทฤษฎี Manager ในเกม 3 มิติ

ทฤษฎี Manager คือหลักการควบคุมวัตถุที่มีจำนวนมาก เช่น เกมยานยิง สามารถปล่อยมิสไซส์ออกมาได้จำนวนมาก เกมวางแผนการรบ ผู้เล่นสามารถสร้างจำนวนยูนิตของตัวเองได้ตามต้องการ โดยการควบคุมวัตถุที่มีมากมายขนาดนี้ หากเป็นการเขียนโปรแกรมแบบโครงสร้างทั่วไป เขียนไปจนกระทั่งมียูนิตที่หลากหลาย ผลที่ตามคือการควบคุมโค้ดโปรแกรมจะเป็นไปด้วยความยากลำบาก ซับซ้อนและวุ่นวาย เมื่อมีการกลับมาแก้ไขภายหลัง จึงเป็นสาเหตุที่มาของการนำทฤษฎี Managerมาใช้ในเกม 3 มิติ

19.1 การสร้าง Manager

การสร้าง Manager ขึ้นมาใช้งานมีหลักการ 3 ขั้นตอนคือ (1.) Add Input (2.) Manager Process (3.) Output ตามภาพด้านล่าง



ภาพการทำงานของทฤษฎี Manager

ในส่วนของ Add Input จะมีจำนวนเท่าไหร่ก็ได้เท่าที่ต้องการ โดยมาแล้วจะเป็นข้อมูลแบบมีโครงสร้าง ถูกส่งเข้าไปยังขั้นตอน Manager Process เพื่อประมวลผลตามข้อมูลที่ได้รับมา จากนั้นส่งข้อมูลออกไปเป็น Output แสดงออกทางหน้าจอ การสร้าง Manager ขึ้นมาจำเป็นต้องอ้างอิงกับตัวอย่าง ซึ่งตัวอย่างนี้จะเลือกเกมยานยิง สามารถปล่อยมิสไซส์ออกมาได้ตามจำนวนที่ผู้เล่นกดปุ่ม มีขั้นตอนดังนี้ (ก่อนลงมือเขียนโค้ด ผู้อ่านต้องสร้างโปรแกรมโหลดโมเดล 3 มิติและทำการเคลื่อนที่โมเดลด้วยลูกศรขึ้น ลง ซ้าย ขวาให้ได้ก่อน จากนั้นจึงมาเขียนโค้ดในตัวอย่างนี้)

Step 1 of 5

```
typedef struct {
        IAnimatedMesh*  misslie;
        IAnimatedMeshSceneNode* misslie_node;
        vector3df pos, di, vel;
}Missile_info;
array<Missile_info*> mlist;
bool AddMis( char* name, const vector3df& pos, const vector3df& di, const vector3df& vel );
void UpdateMis(void);
void RemoveMis(void);
```

- นิยามโครงสร้างชื่อ Missile_Info เพื่อเก็บข้อมูลของมิสไซส์ 1 ลูก มีข้อมูลคือ IAnimatedMesh* และ IAnimatedMeshSceneNode* สำหรับแสดงโมเดล 3 มิติ ตัวแปรเวคเตอร์ 3 มิติชื่อ pos, di และ vel คือตำแหน่ง ทิศทาง และความเร็วของมิสไซส์ตามลำดับ ตัวแปร di จะมีค่าเป็น 1

และ -1 เท่านั้น ส่วนค่า vel มีค่าตั้งแต่ 0 ขึ้นไป ค่ายิ่งมากจะยิ่งเร็ว

- นิยามตัวแปรชนิดอาร์เรย์ชื่อ mlist เพื่อเก็บข้อมูลของโครงสร้าง Missile*
- สร้างโปรโตไทพ์ฟังชั่นชื่อ AddMis(), UpdateMis() และ RemoveMis() โดย AddMis() ส่งพารามิเตอร์ 4 ตัวคือ ชื่อโมเดลที่นำมาเป็นมิสไซส์ ตำแหน่งเกิด ทิศทางและความเร็วของมิสไซส์ ตามลำดับ

Step 2 of 5

```
bool AddMis( char* name, const vector3df& pos, const vector3df& di, const vector3df& vel ) {
        Missile_info* minfo = new Missile_info( );
        minfo->misslie = g_engine.smgr->getMesh( name );
        minfo->misslie_node = g_engine.smgr->addAnimatedMeshSceneNode( minfo->misslie );
        minfo->misslie_node->setMaterialFlag( EMF_LIGHTING, false );
        minfo->pos = pos;
        minfo->di = di;
        minfo->vel = vel;
        mlist.push_back( minfo );
        return true;
}
```

- เขียนรายละเอียดฟังก์ชั่น AddMis() เริ่มด้วยจองหน่วยความจำของโครงสร้าง Missile_Info โดยนิยามตัวแปร minfo มารองรับที่อยู่ของหน่วยความจำ

- เพิ่มโมเดล 3 มิติด้วยคำสั่ง getMesh(), addAnimatedMeshSceneNode() และสั่งให้ไม่เกิดผลกระทบต่อแสง
- แทนค่าตัวแปรตำแหน่งเกิด ทิศทาง ความเร็วและเพิ่มข้อมูลลงในอาร์เรย์ mlist ที่สร้างขึ้น

Step 3 of 5

```
void UpdateMis(void) {
        for( u32 i = 0; i != mlist.size(); i++ ) {
         mlist[i]->pos += mlist[i]->di * mlist[i]->vel;
         mlist[i]->misslie_node->setPosition( mlist[i]->pos );
        }
}
```

ประมวลผลมิสไซส์ตามที่ได้รับข้อมูลมา โดยมิสไซส์จะมีสมการเคลื่อนที่คือ

ตำแหน่งใหม่ = ตำแหน่งเดิม x ทิศทาง x ความเร็ว

วนลูปให้ครบสมาชิกทุกตัวของอาร์เรย์ จากนั้นนำสมการเคลื่อนที่มิสไซส์มาใช้ เมื่อได้ตำแหน่งใหม่ เซ็ทค่าลงในมิสไซส์ลูกนั้น เพื่อให้เคลื่อนที่ตามข้อมูลที่ได้รับมา

Step 4 of 5

```
void RemoveMis(void) {
        for( u32 i = 0; i != mlist.size(); i++ ) {
                if( mlist[i]->pos.Z > 2000 ) {
```

```
                    mlist[i]->misslie_node->remove( );
                    SAFEDELETE( mlist[i] );
                    mlist.erase( i );
                    return;
                }
        }
}
```

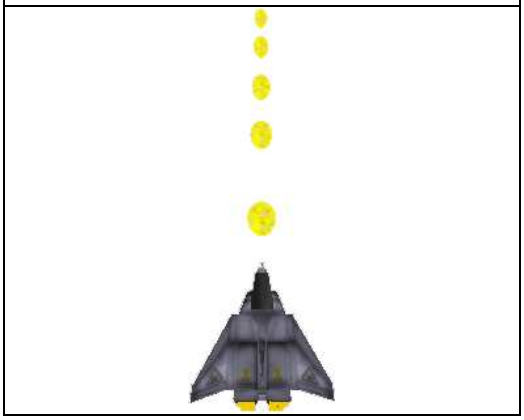
ลบมิสไซส์ออกจาก Manager โดยกำหนดเงื่อนไขในการลบมิสไซส์ หากแกน Z มีค่ามากกว่า 2000 หน่วยให้ลบมิสไซส์ออกจากระบบ

Step 5 of 5

```
if( g_input.keyPressed(KEY_SPACE) )
        AddMis( "pacman.b3d", pac_node->getPosition(), vector3df(0,0,1), vector3df(0,0,10) );
UpdateMis();
RemoveMis();
```

เมื่อได้ Manager ครบทั้ง 3 ส่วนคือ AddMis(), UpdateMis() และ RemoveMis() ต่อไปคือการเรียกใช้ ให้เขียนโค้ดต่อจากบรรทัดที่ควบคุมการเคลื่อนที่ของโมเดล โดยกำหนดให้ปุ่ม SapceBar เป็นการยิงมิสไซส์ ส่งค่าโมเดลชื่อ pacman.b3d ตำแหน่งเกิดของมิสไซส์คือตำแหน่งของโมเดลอ่านค่ามาจากคำสั่ง getPosition() ทิศทางกำหนดเป็น 0,0,1 แกน Z เป็นบวกคือพุ่งไปข้างหน้า ความเร็วเป็น 0,0,10 และสุดท้ายให้เรียกใช้งานคำสั่ง UpdateMis() และ RemoveMis() ตามลำดับ เมื่อรันจะเห็นผลลัพธ์ดังภาพด้านล่าง

ภาพผลลัพธ์การยิงมิสไซส์



สรุปการสร้าง Manager ขึ้นมาใช้งานมีหลักการ 3 ขั้นตอนคือ (1.) Add Input (2.) Manager Process (3.) Output เขียนฟังก์ชั่นขึ้นมาเพื่อให้ควบคุมการทำงานในแต่ละส่วน จากตัวอย่างข้างต้นจะมีฟังก์ชั่น AddMis(), UpdateMis() และ RemoveMis() มีหน้าที่ควบคุมการทำงาน และสุดท้ายคือการเรียกใช้งาน มีการกำหนดต้องการเรียกใช้งานในส่วนใดของเกม เพียงการทำงานไม่กี่คำสั่งทำให้สามารถควบคุมวัตถุที่มีจำนวนมากได้อย่างมีระบบและแก้ไขได้สะดวกภายหลัง