ภาคผนวก

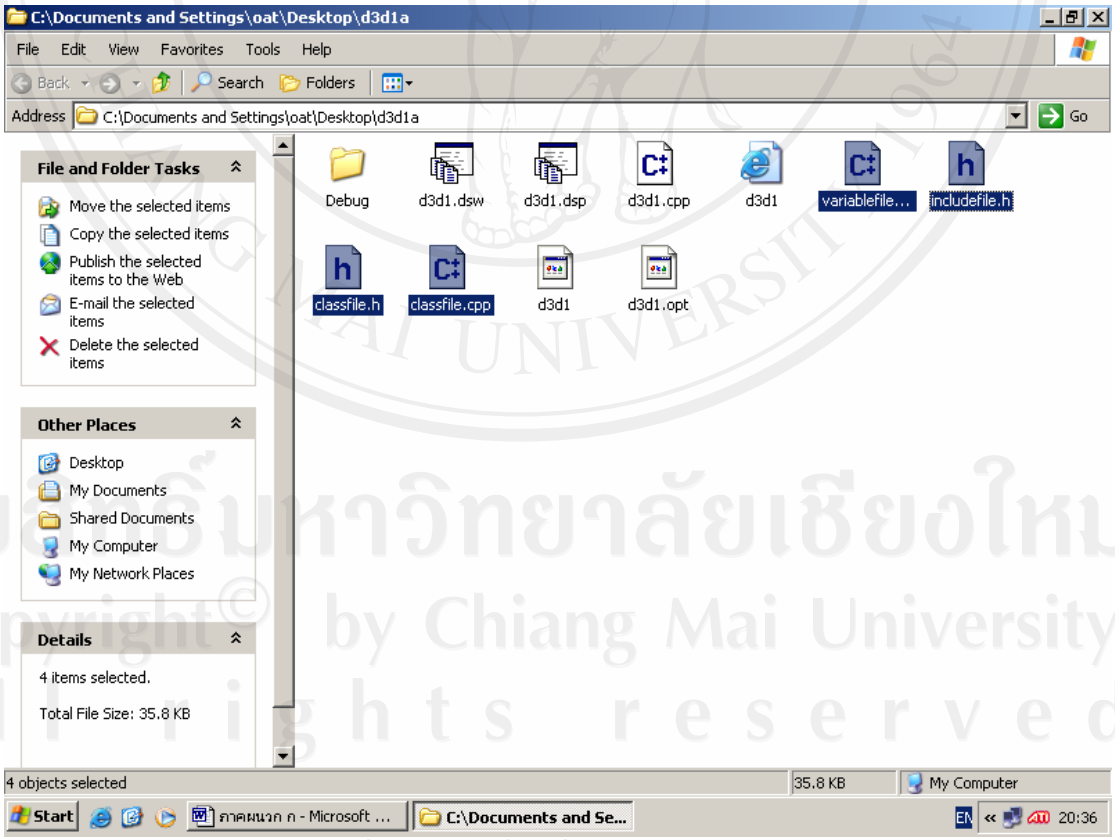
ภาคผนวก ก

การเพิ่มคลาสเข้าไปในโปรเจกต์

การเพิ่มไฟล์ includefile.h , variablefile.cpp , classfile.h , classfile.cpp

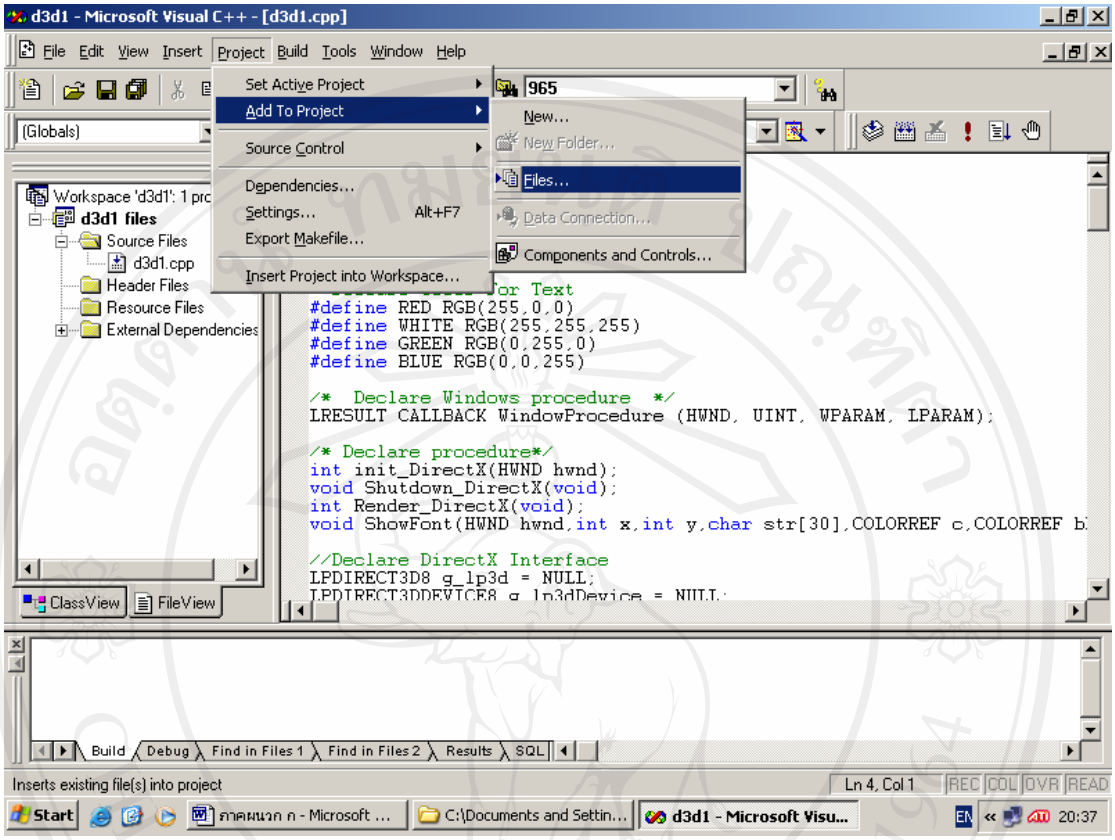
การนำเอาคลาส includefile.h , variablefile.cpp , classfile.h , classfile.cpp ไปใช้ในการสร้างเกม ดังต่อไปนี้

1. การเพิ่ม includefile.h , variablefile.cpp , classfile.h , classfile.cpp เข้าไปในโปรเจ็กต์ทำได้ โดยก็อปปี้ไฟล์ includefile.h , variablefile.cpp , classfile.h , classfile.cpp เข้าไปในไดเร็คทอรีโปรเจ็กต์ ดังรูป ก.1



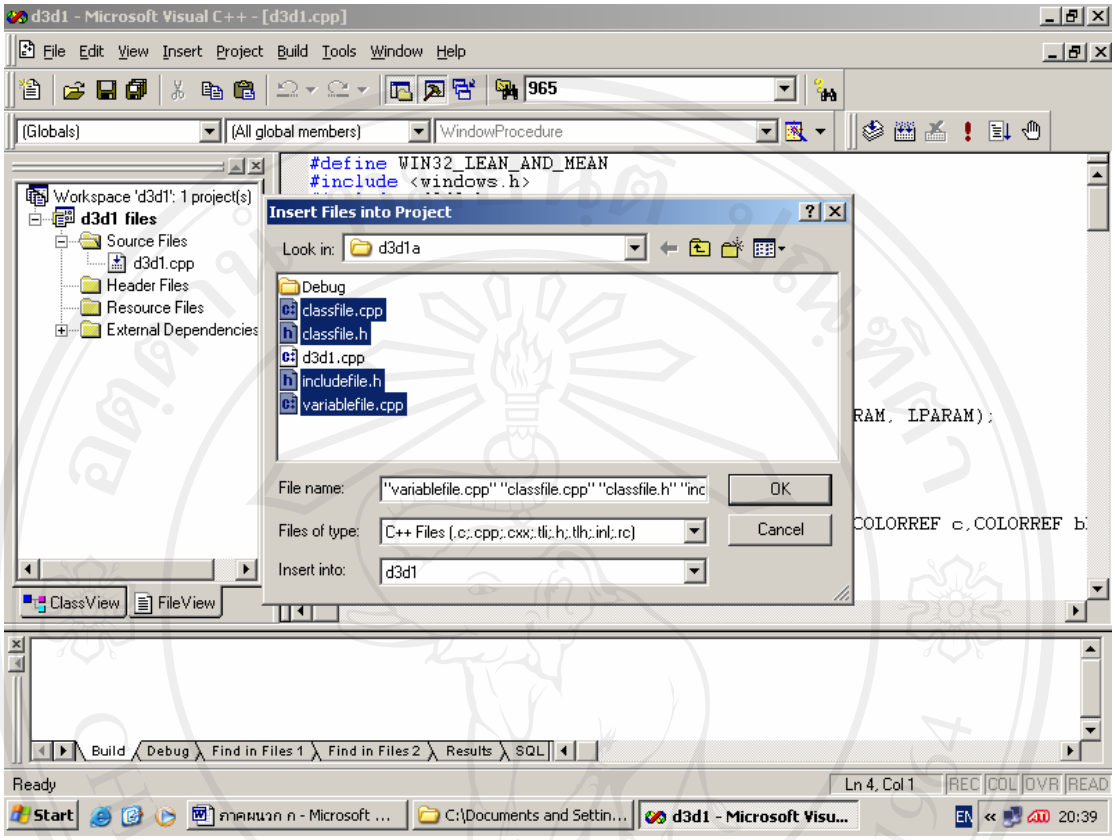
รูปที่ ก.1 ไฟล์ที่ใช้เพิ่มเข้าไปในโปรเจ็กต์

2.เลือกเมนู Project > Add To Project > Files… ตามรูป ก.2



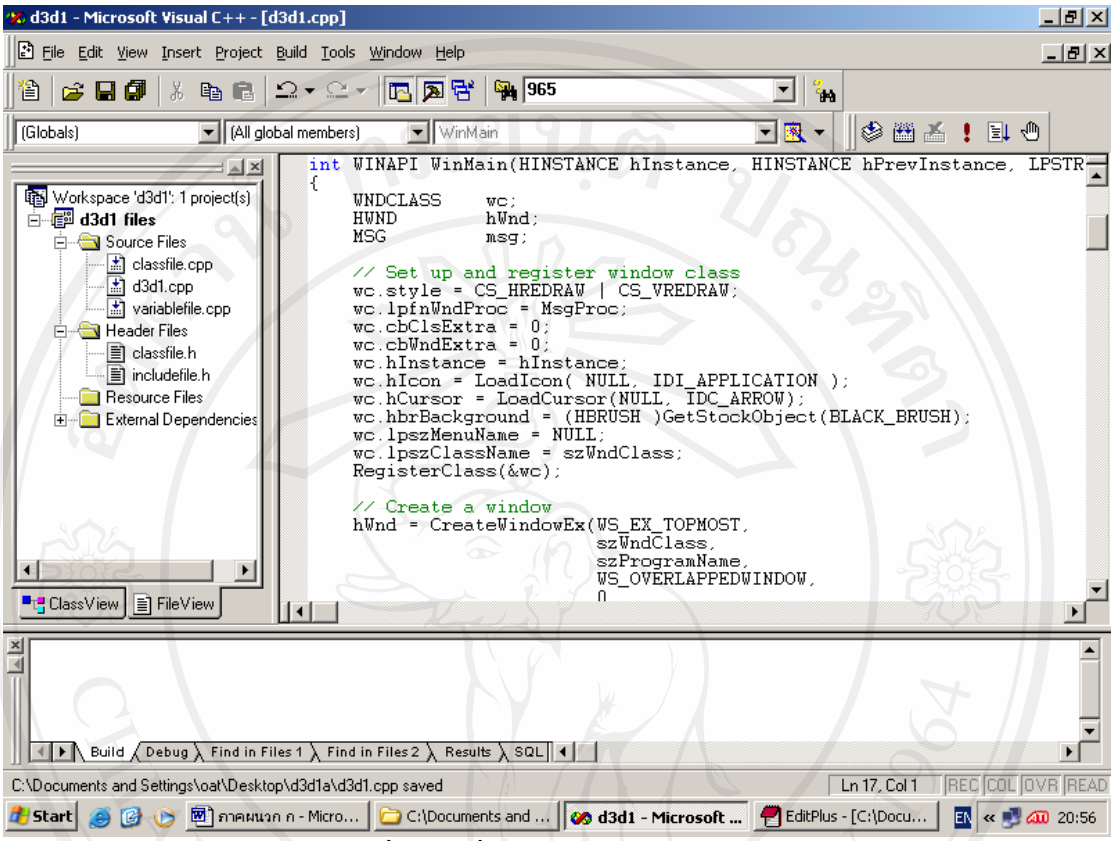
รูปที่ ก.2 การเพิ่มไฟล์เข้าโปรเจ็กต์

เลือกไฟล์ includefile.h , variablefile.cpp , classfile.h , classfile.cpp ตามรูป ก.3



รูปที่ ก.3 เลือกไฟล์เข้าโปรเจ็กต์

เมื่อนำไฟล์เข้าสู่โปรเจ็กต์ แล้ว ดังรูป ก.4



รูปที่ ก.4 เมื่อนำไฟล์เข้าโปรเจ็กต์แล้ว

3. ทำการ include ไฟล์ classfile.h เข้าไปในโปรเจ็กต์

#include “classfile.h”

หลังจากนั้นก็สามารถเรียกใช้งานคลาสต่างๆ ที่ include ได้

ภาคผนวก ข

คู่มือการใช้งาน

เฟรมเวิร์คมีทั้งหมด 11 คลาส คือvector3D, polygon, gameObject, input, rectangles, cell, camera, gridGame, render, socketObject, convert

1. คลาส vector3D

```
class vector3D{
public:
float x;
float y;
float z;
DWORD color;

Vector3D();
Vector3D(DWORD color);
Vector3D(float x, float y, float z);
Vector3D(float x, float y, float z, DWORD color);
Vector3D plus(vector3D v);
Vector3D minus(vector3D v);
float dotProduct(vector3D v);
vector3D crossProduct(vector3D v);
};
```

รายละเอียดของฟังก์ชันในคลาส

```
vector3D::vector3D(){ x=0.0f; y=0.0f; z=0.0f; }

vector3D::vector3D(DWORD color)
{ x=0.0f; y=0.0f; z=0.0f;this->color=color; }

vector3D::vector3D(float x, float y, float z)
{ this->x=x; this->y=y; this->z=z; }

vector3D::vector3D(float x, float y, float z, DWORD color){
this->x=x; this->y=y; this->z=z;
this->color=color;
}
```

```
vector3D vector3D::plus(vector3D v){
float x, y, z;
x=this->x+v.x;
y=this->y+v.y;
z=this->z+v.z;
return vector3D(x, y, z);
}
vector3D vector3D::minus(vector3D v){
float x, y, z;
x=this->x-v.x;
y=this->y-v.y;
z=this->z-v.z;
return vector3D(x, y, z);
}

float vector3D::dotProduct(vector3D v){
float x, y, z, sum;
x=this->x*v.x;
y=this->y*v.y;
z=this->z*v.z;
sum=x+y+z;
return sum;
}

vector3D vector3D::crossProduct(vector3D v){
float x, y, z;
x=(this->y*v.z)-(this->z*v.y);
y=(this->z*v.x)-(this->x*v.z);
z=(this->x*v.y)-(this->y*v.x);
return vector3D(x, y, z);
}
```

ตัวแปรและฟังก์ชัน

float x; เก็บค่าข้อมูลตำแหน่ง x

float y; เก็บค่าข้อมูลตำแหน่ง y

float z; เก็บค่าข้อมูลตำแหน่ง z

DWORD color; เก็บค่าข้อมูลตำแหน่งของสี

vector3D(); ตัวสร้างของคลาส

vector3D(DWORD color); ตัวสร้างของคลาสรับค่าสี

vector3D(float x, float y, float z); ตัวสร้างของคลาสรับค่าพิกัด

vector3D(float x, float y, float z, DWORD color); ตัวสร้างของคลาสรับค่าพิกัดและสี

vector3D plus(vector3D v); ฟังก์ชันสำหรับบวกเวกเตอร์2เวกเตอร์รีเทิร์นค่าเป็นเวกเตอร์

vector3D minus(vector3D v); ฟังก์ชันสำหรับลบเวกเตอร์2เวกเตอร์รีเทิร์นค่าเป็นเวกเตอร์

float dotProduct(vector3D v); ฟังก์ชันสำหรับคูณเชิงสเกลาร์ รีเทิร์นค่าเป็น float

vector3D crossProduct(vector3D v); ฟังก์ชันสำหรับคูณเชิงเวกเตอร์ รีเทิร์นค่าเป็นเวกเตอร์

2. คลาส polygon

```
class polygon{
public:
vector3D *v;
int numVertices;

polygon();
polygon(vector3D v[], int numVertices);
polygon(vector3D v0, vector3D v1, vector3D v2);
void setVector(vector3D v[], int numVertices);
void setVector(vector3D v0, vector3D v1, vector3D v2);
};
```

รายละเอียดของฟังก์ชันในคลาส

```
polygon::polygon(){
v=NULL;
numVertices=0;
}

polygon::polygon(vector3D v[], int numVertices){
this->v=new vector3D[numVertices];
for(int i=0;i<numVertices;i++)
{
this->v[i]=v[i];
}
this->numVertices=numVertices;
}

polygon::polygon(vector3D v0, vector3D v1, vector3D v2){
this->v=new vector3D[3];
this->v[0]=v0;
this->v[1]=v1;
this->v[2]=v2;
this->numVertices=3;
}

void polygon::setVector(vector3D v[], int numVertices){
this->v=new vector3D[numVertices];
for(int i=0;i<numVertices;i++)
{
this->v[i]=v[i];
}
this->numVertices=numVertices;
}
```

```
void polygon::setVector(vector3D v0, vector3D v1, vector3D v2){
this->v=new vector3D[3];
this->v[0]=v0;
this->v[1]=v1;
this->v[2]=v2;
this->numVertices=3;
}
```

ตัวแปรและฟังก์ชัน

vector3D *v; พอยเตอร์ของคลาส vector3D เก็บค่าพิกัดของ polygon

int numVertices; ค่าจำนวนของเวอร์เท็กซ์ (Vertex) ทั้งหมด

polygon(); ตัวสร้างของคลาส

polygon(vector3D v[], int numVertices); ตัวสร้างของคลาสรับค่าอาเรย์ของคลาส vector3D และ จำนวนเวอร์เท็กซ์ (Vertex)

polygon(vector3D v0, vector3D v1, vector3D v2); ตัวสร้างของคลาสรับค่าเวกเตอร์ 3 ค่า

void setVector(vector3D v[], int numVertices); ฟังก์ชันรับค่าอาเรย์ของคลาส vector3D และ จำนวนเวอร์เท็กซ์ (Vertex)

void setVector(vector3D v0, vector3D v1, vector3D v2); ฟังก์ชันรับค่าเวกเตอร์ 3 ค่า

3. คลาส gameObject

```
class gameObject{
public:
string name;
int type;
polygon *p;
int numPolygons;
float top;
float bottom;
float radius;
float radius2D;
float maxX, minX, maxY, minY, maxZ, minZ;
vector3D centerPivotPoint;
vector3D vectorNormal;
polygon polygonNormal;
bool see;
bool hear;
float k;
float distanceEye;
int front, back, collinear;
float radar;

gameObject();
gameObject(polygon p);
gameObject(polygon p[], int numPolygons);
string getName();
void setName(string name);
void setType(int type);
void setDistanceEye(float distanceEye);
void setRadar(float radar);
void setPolygon(polygon p);
void setPolygon(polygon p[], int numPolygons);
void calcCenterPivotPoint();
float getX();
float getY();
float getZ();
bool canSee(vector3D location);
bool canHear(vector3D location);
int getSide(vector3D location);
};
```

รายละเอียดของฟังก์ชันในคลาส

```
gameObject::gameObject():front(1)
, back(-1), collinear(0), distanceEye(250.0f){
p=NULL;
```

```
numPolygons=0;
name="NoName";
}

gameObject::gameObject(polygon p):front(1)
, back(-1), collinear(0), distanceEye(250.0f){
int j=0;
this->p=new polygon();
*(this->p)=p;
this->numPolygons=1;
calcCenterPivotPoint();
name="NoName";
}

gameObject::gameObject(polygon p[], int numPolygons):front(1)
, back(-1), collinear(0), distanceEye(250.0f){
int i=0; int j=0;
this->p=new polygon[numPolygons];
for(i=0;i<numPolygons;i++)
{
this->p[i]=p[i];
}
this->numPolygons=numPolygons;
calcCenterPivotPoint();
name="NoName";
}

void gameObject::setName(string name){
this->name=name;
}

string gameObject::getName(){
return this->name;
}

void gameObject::setType(int type){
this->type=type;
}

void gameObject::setDistanceEye(float distanceEye){
this->distanceEye=distanceEye;
}

void gameObject::setRadar(float radar){
this->radar=radar;
}
```

```
void gameObject::setPolygon(polygon p){
int j=0;
this->p=new polygon();
*(this->p)=p;
this->numPolygons=1;
calcCenterPivotPoint();
}

void gameObject::setPolygon(polygon p[], int numPolygons){
int i=0; int j=0;
this->p=new polygon[numPolygons];
for(i=0;i<numPolygons;i++)
{
this->p[i]=p[i];
}
this->numPolygons=numPolygons;
calcCenterPivotPoint();
}

void gameObject::calcCenterPivotPoint(){
int i, j;
float maxX, minX, maxY, minY, maxZ, minZ;
float distanceX, distanceY, distanceZ;
float centerX, centerY, centerZ;
maxX=FLT_MIN; minX=FLT_MAX;
maxY=FLT_MIN; minY=FLT_MAX;
maxZ=FLT_MIN; minZ=FLT_MAX;
for(i=0;i<numPolygons;i++){
        for(j=0;j<p[i].numVertices;j++)
        {
        maxX=__max( maxX , p[i].v[j].x );
        minX=__min( minX , p[i].v[j].x );

        maxY=__max( maxY , p[i].v[j].y );
        minY=__min( minY , p[i].v[j].y );

        maxZ=__max( maxZ , p[i].v[j].z );
        minZ=__min( minZ , p[i].v[j].z );
        }
}

this->maxX=maxX;//MAX MIN
this->minX=minX;
this->maxY=maxY;
this->minY=minY;
this->maxZ=maxZ;
this->minZ=minZ;
```

```
bottom=minY; //BOTTOM
top=maxY; //TOP

distanceX=maxX-minX;
distanceY=maxY-minY;
distanceZ=maxZ-minZ;

centerX=minX+(distanceX / 2);
centerY=minY+(distanceY / 2);
centerZ=minZ+(distanceZ / 2);

centerPivotPoint.x=centerX; // Center
centerPivotPoint.y=centerY;
centerPivotPoint.z=centerZ;

//RADIUS
radius=( (centerX-maxX)*(centerX-maxX) ) +
( (centerY-maxY)*(centerY-maxY) ) +
( (centerZ-maxZ)*(centerZ-maxZ) ) ;
radius=sqrt(radius);

//RADIUS 2D for cylinder
radius2D=( (centerX-maxX)*(centerX-maxX) ) +
+( (centerZ-maxZ)*(centerZ-maxZ) ) ;
radius2D=sqrt(radius2D);

//polygonNormal
vector3D v0(minX, minY, minZ);
vector3D v1(maxX, minY, minZ);
vector3D v2(minX, maxY, minZ);
polygonNormal.setVector(v0, v1, v2);

//vectorNormal = (p1-p2)X(p3-p2)
vector3D temp1;
vector3D temp2;
temp1=polygonNormal.v[0].minus(polygonNormal.v[1]);
temp2=polygonNormal.v[2].minus(polygonNormal.v[1]);
vectorNormal=temp1.crossProduct(temp2);

//k
k=vectorNormal.dotProduct(polygonNormal.v[0]);
radar=250;
}

float gameObject::getX(){
return centerPivotPoint.x;
```

```
}

float gameObject::getY(){
return centerPivotPoint.y;
}

float gameObject::getZ(){
return centerPivotPoint.z;
}

bool gameObject::canSee(vector3D location){//AI
float distance=0.0f;
int side=1000;
distance=((location.x-centerPivotPoint.x)*(location.x-centerPivotPoint.x))
        +((location.y-centerPivotPoint.y)*(location.y-centerPivotPoint.y))
        +((location.z-centerPivotPoint.z)*(location.z-centerPivotPoint.z));
distance=sqrt(distance);
side=getSide(location);
if( (side==front) && (distanceEye>distance) )
return true;
return false;
}

bool gameObject::canHear(vector3D location){//AI
float distance;
distance=((location.x-centerPivotPoint.x)*(location.x-centerPivotPoint.x))
        +((location.y-centerPivotPoint.y)*(location.y-centerPivotPoint.y))
        +((location.z-centerPivotPoint.z)*(location.z-centerPivotPoint.z));
distance=sqrt(distance);
if(distance<radar) return true;
return false;
}

int gameObject::getSide(vector3D location){//AI
float side;
side=location.dotProduct(vectorNormal);
if(side==k) return collinear;
if(side>k) return front;
if(side<k) return back;
return 1000;
}
```


ตัวแปรและฟังก์ชัน

string name; ชื่อของวัตถุ

int type; ชนิดของวัตถุ

polygon *p; ข้อมูลของpolygonในgameObjectนี้

int numPolygons; จำนวนpolygonในgameObjectนี้

float top; จุดสูงสุดของgameObjectนี้

float bottom; จุดสูงสุดของgameObjectนี้

float radius; รัศมีของgameObjectนี้

float radius2D; รัศมีของgameObjectนี้ โดยเก็บแบบ 2 มิติเพื่อทดสอบการชนแบบทรงกระบอก

float maxX, minX, maxY, minY, maxZ, minZ; จุดmax,min ของ x, y, z

vector3D centerPivotPoint; จุดศูนย์กลาง

vector3D vectorNormal; เวกเตอร์Normal ของgameObject

polygon polygonNormal; polygon ที่เป็นตัวแทนในการหา vector Normal

bool see; ค่าบูลีนของการมองเห็น

bool hear; ค่าบูลีนของการได้ยิน

float k; ค่า k ที่คำนวณได้จาก vector Normal

float distanceEye; ระยะของสายตา

int front, back, collinear; ค่าที่ใช้แทนด้านหน้า ด้านหลัง ด้านข้าง

float radar; ระยะการได้ยิน

gameObject(); ตัวสร้างของคลาส

gameObject(polygon p); ตัวสร้างของคลาสรับค่าเป็นpolygon

gameObject(polygon p[], int numPolygons); ตัวสร้างของคลาสอาเรย์ของpolygonและจำนวนของpolygon

string getName(); ฟังก์ชันในการรีเทิร์นชื่อ

void setName(string name); ฟังก์ชันในการตั้งชื่อgameObject

void setType(int type); ฟังก์ชันในการตั้งค่าประเภทของgameObject

void setDistanceEye(float distanceEye); ฟังก์ชันในการตั้งค่าระยะสายตา

void setRadar(float radar); ฟังก์ชันในการตั้งค่าระยะการได้ยิน

void setPolygon(polygon p); ฟังก์ชันใส่ค่าpolygonให้gameObject

void setPolygon(polygon p[], int numPolygons); ฟังก์ชันใส่ค่าpolygonให้gameObject รับค่าอาเรย์ของpolygonและจำนวนของpolygon

void calcCenterPivotPoint(); ฟังก์ชันการคำนวณค่าต่างในคลาส

float getX(); ฟังก์ชันรีเทิร์นค่า x

float getY(); ฟังก์ชันรีเทิร์นค่า y

float getZ(); ฟังก์ชันรีเทิร์นค่า z

bool canSee(vector3D location); ฟังก์ชันรีเทิร์นค่าบูลีนว่ามองเห็นหรือไม่

bool canHear(vector3D location); ฟังก์ชันรีเทิร์นค่าบูลีนว่าได้ยินหรือไม่

int getSide(vector3D location); ฟังก์ชันรีเทิร์นว่าเป็นด้านหน้า ด้านหลัง ด้านข้าง

4. คลาส input

```
class input{
public:
HINSTANCE hInstance;
HWND hWnd;
char buffer[256];
LPDIRECTINPUT inputvar;
LPDIRECTINPUTDEVICE keyboard;

 const int keyA;
 const int keyB;
 const int keyC;
 const int keyD;
 const int keyE;
 const int keyF;
 const int keyG;
 const int keyH;
 const int keyI;
 const int keyJ;
 const int keyK;
 const int keyL;
 const int keyM;
 const int keyN;
 const int keyO;
 const int keyP;
 const int keyQ;
 const int keyR;
 const int keyS;
 const int keyT;
 const int keyU;
 const int keyV;
 const int keyW;
 const int keyX;
 const int keyY;
 const int keyZ;

 const int keyUP;
 const int keyDOWN;
 const int keyLEFT;
 const int keyRIGHT;

 const int keyESCAPE;
 const int keySPACE;
 const int keyRETURN;
 const int keyNULL;

input();
```

```
input(HINSTANCE hInstance, HWND hWnd);
int getKey();
};
```

รายละเอียดของฟังก์ชันในคลาส

input::input():keyA(65), keyB(66), keyC(67), keyD(68), keyE(69), keyF(70), keyG(71), keyH(72), keyI(73), keyJ(74), keyK(75), keyL(76), keyM(77), keyN(78), keyO(79), keyP(80), keyQ(81), keyR(82), keyS(83), keyT(84), keyU(85), keyV(86), keyW(87), keyX(88), keyY(89), keyZ(90), keyUP(38), keyDOWN(40), keyLEFT(37), keyRIGHT(39), keyESCAPE(27), keySPACE(32), keyRETURN(13), keyNULL(4000) {}

input::input(HINSTANCEhInstance, HWND hWnd): keyA(65), keyB(66), keyC(67), keyD(68), keyE(69), keyF(70), keyG(71), keyH(72), keyI(73), keyJ(74), keyK(75), keyL(76), keyM(77), keyN(78), keyO(79), keyP(80), keyQ(81), keyR(82), keyS(83), keyT(84), keyU(85), keyV(86), keyW(87), keyX(88), keyY(89), keyZ(90), keyUP(38), keyDOWN(40), keyLEFT(37), keyRIGHT(39), keyESCAPE(27), keySPACE(32), keyRETURN(13), keyNULL(4000)

```
{//Start Function
this->hInstance=hInstance;
this->hWnd=hWnd;
DirectInput8Create(hInstance, DIRECTINPUT_VERSION, IID_IDirectInput8,
(VOID**)&inputvar, NULL);
inputvar->CreateDevice(GUID_SysKeyboard, &keyboard, NULL);
keyboard->SetDataFormat(&c_dfDIKeyboard);
keyboard->SetCooperativeLevel(hWnd, DISCL_FOREGROUND |
DISCL_NONEXCLUSIVE);
keyboard->Acquire();
}//End Function
int input::getKey(){//Start Function
keyboard->GetDeviceState( sizeof(buffer) , (LPVOID)&buffer );

if( buffer[DIK_A] & 0x80 )
{
return keyA;
}
if( buffer[DIK_B] & 0x80 )
```

```
{
return keyB;
}
if( buffer[DIK_C] & 0x80 )
{
return keyC;
}
if( buffer[DIK_D] & 0x80 )
{
return keyD;
}
if( buffer[DIK_E] & 0x80 )
{
return keyE;
}
if( buffer[DIK_F] & 0x80 )
{
return keyF;
}
if( buffer[DIK_G] & 0x80 )
{
return keyG;
}
if( buffer[DIK_H] & 0x80 )
{
return keyH;
}
if( buffer[DIK_I] & 0x80 )
{
return keyI;
}
if( buffer[DIK_J] & 0x80 )
{
return keyJ;
}
if( buffer[DIK_K] & 0x80 )
{
return keyK;
}
if( buffer[DIK_L] & 0x80 )
{
return keyL;
```

```
}
if( buffer[DIK_M] & 0x80 )
{
return keyM;
}
if( buffer[DIK_N] & 0x80 )
{
return keyN;
}
if( buffer[DIK_O] & 0x80 )
{
return keyO;
}
if( buffer[DIK_P] & 0x80 )
{
return keyP;
}
if( buffer[DIK_Q] & 0x80 )
{
return keyQ;
}
if( buffer[DIK_R] & 0x80 )
{
return keyR;
}
if( buffer[DIK_S] & 0x80 )
{
return keyS;
}
if( buffer[DIK_T] & 0x80 )
{
return keyT;
}
if( buffer[DIK_U] & 0x80 )
{
return keyU;
}
if( buffer[DIK_V] & 0x80 )
{
return keyV;
}
```

```
if( buffer[DIK_W] & 0x80 )
{
return keyW;
}
if( buffer[DIK_X] & 0x80 )
{
return keyX;
}
if( buffer[DIK_Y] & 0x80 )
{
return keyY;
}
if( buffer[DIK_Z] & 0x80 )
{
return keyZ;
}
if( buffer[DIK_UP] & 0x80 )
{
return keyUP;
}
if( buffer[DIK_DOWN] & 0x80 )
{
return keyDOWN;
}
if( buffer[DIK_LEFT] & 0x80 )
{
return keyLEFT;
}
if( buffer[DIK_RIGHT] & 0x80 )
{
return keyRIGHT;
}
if( buffer[DIK_ESCAPE] & 0x80 )
{
return keyESCAPE;
}
if( buffer[DIK_SPACE] & 0x80 )
{
return keySPACE;
}
if( buffer[DIK_RETURN] & 0x80 )
```

```
{
return keyRETURN;
}
return keyNULL;
}//End Function
```

ตัวแปรและฟังก์ชัน

HINSTANCE hInstance; ตัวแปร instanceของวินโดส์

HWND hWnd; ตัวแปร handle ของวินโดส์

char buffer[256]; ตัวแปรอาเรย์ของchar ใช้รับค่าการกดคีย์

LPDIRECTINPUT inputvar; ตัวแปรของไดเรกอินพุต

LPDIRECTINPUTDEVICE keyboard; ตัวแปรไดเรกอินพุตดีไวซ์

ค่าคงที่สำหรับการกดคีย์ต่างๆ

```
const int keyA;
const int keyB;
const int keyC;
const int keyD;
const int keyE;
const int keyF;
const int keyG;
const int keyH;
const int keyI;
const int keyJ;
const int keyK;
const int keyL;
const int keyM;
const int keyN;
const int keyO;
const int keyP;
const int keyQ;
const int keyR;
const int keyS;
const int keyT;
const int keyU;
const int keyV;
const int keyW;
const int keyX;
const int keyY;
const int keyZ;
const int keyUP;
const int keyDOWN;
const int keyLEFT;
const int keyRIGHT;
```

const int keyESCAPE;
const int keySPACE;
const int keyRETURN;
const int keyNULL;

input(); ตัวสร้าง

input(HINSTANCE hInstance, HWND hWnd); ตัวสร้างรับค่าinstanceและhandle

int getKey(); ฟังก์ชันรับการกดของคีย์

5. คลาส rectangles

```
class rectangles{
public:
int x, y, width, height;

rectangles();
rectangles(int x, int y, int width, int height);
};
```

รายละเอียดของฟังก์ชันในคลาส

```
rectangles::rectangles(){}

rectangles::rectangles(int x, int y, int width, int height){
this->x=x;
this->y=y;
this->width=width;
this->height=height;
}
```

ตัวแปรและฟังก์ชัน

int x; ค่าตำแหน่งเริ่มต้นของ x

int y; ค่าตำแหน่งเริ่มต้นของ y

int width; ค่าความกว้าง

int height; ค่าความสูง

rectangles(); ตัวสร้างของคลาส

rectangles(int x, int y, int width, int height); ตัวสร้างของคลาสรับค่า x y ความกว้าง ความสูง

6. คลาส cell

```
class cell{
public:
gameObject *objects;
vector<gameObject> vobjects;
int numObjects;
bool visible;
bool inGrid;

cell();
cell(bool b);
void addObject(gameObject object);
};
```

รายละเอียดของฟังก์ชันในคลาส

```
cell::cell(){
objects=NULL;
numObjects=0;
visible=false;
inGrid=true;
}

cell::cell(bool b){
inGrid=false;
}

void cell::addObject(gameObject object){
int i;
gameObject *temp;
vobjects.push_back(object);
numObjects=vobjects.size();
temp=objects;
objects=new gameObject[numObjects];
for(i=0;i<numObjects;i++){
objects[i]=vobjects.at(i);
}
delete [] temp;
}
```


ตัวแปรและฟังก์ชัน

gameObject *objects; พอยเตอร์gameObjectในคลาสcell

vector<gameObject> vobjects; ตัวแปรvectorชนิดgameObjectในคลาสcell

int numObjects; จำนวนตัวแปรgameObjectในคลาส

bool visible; ค่าบูลีนแทนว่าcellนี้มองเห็นได้หรือไม่

bool inGrid; ค่าบูลีนแทนว่าcell นี้อยู่ใน gridGameหรือไม่

cell(); ตัวสร้างของคลาส

cell(bool b); ตัวสร้างของคลาสรับค่าเป็นบูลีน

void addObject(gameObject object); ฟังก์ชันการเพิ่มgameObjectเข้าไปในcell

7. คลาส camera

```
class camera{
public:
float angleX, angleY, angleZ;
float speed;
float rotateSpeed;
vector3D location;
vector3D oldLocation;
vector3D target;
vector3D oldTarget;
float fovy, aspect, zNear, zFar;
float radius;
float radius2D;
float distanceEye;
vector<gameObject> vhitObjects;
gameObject *hitObjects;
int numHitObjects;

camera();
camera(float x, float y, float z);
void setHeight(float y);
void setfovy(float fovy);
void setaspect(float aspect);
void setzNear(float zNear);
void setzFar(float zFar);
void rotateAngleX(float fRotateSpeed);
void rotateAngleY(float fRotateSpeed);
void rotateAngleZ(float fRotateSpeed);
void adjustSpeed(float speed);
void adjustRotateSpeed(float rotateSpeed);
void setSpeed(float speed);
void setRotateSpeed(float rotateSpeed);
void setDistanceEye(float distanceEye);
void setRadius(float radius);
void setRadius2D(float radius2D);
void cameraForward();
void cameraBackward();
void cameraTurnLeft();
void cameraTurnRight();
float getX();
float getY();
float getZ();
bool hitObject(gameObject object);
void hitObject(cell c);
bool hitObjectCylinder(gameObject object);
void hitObjectCylinder(cell c);
void addHitObjects(gameObject object);
```

```
void clearHitObjects();
};
```

รายละเอียดของฟังก์ชันในคลาส

```
camera::camera(){
angleX=0.0f;
angleY=0.0f;
angleZ=0.0f;
radius=1.0f;radius2D=1.0f;
speed=0.1f;
rotateSpeed=0.1f;
location.x=0.0f;
location.y=0.0f;
location.z=0.0f;
oldLocation.x=0.0f;
oldLocation.y=0.0f;
oldLocation.z=0.0f;
distanceEye=965.5f;
target.x = location.x-(distanceEye*sin(D3DXToRadian(angleY)));
target.y = 2.5f;
target.z = location.z+(distanceEye*cos(D3DXToRadian(angleY)));
fovy=D3DX_PI/4;
aspect=800.0f/600.0f;
zNear=1.0f;
zFar=1000.0f;
hitObjects=NULL;
}

camera::camera(float x, float y, float z){
angleX=0.0f;
angleY=0.0f;
angleZ=0.0f;
radius=1.0f;radius2D=1.0f;
speed=0.1f;
rotateSpeed=0.1f;
location.x=x;
location.y=y;
location.z=z;
oldLocation.x=x;
oldLocation.y=y;
oldLocation.z=z;
distanceEye=965.5f;
target.x=location.x-(distanceEye*sin(D3DXToRadian(angleY)));
target.y=2.5f;
target.z=location.z+(distanceEye*cos(D3DXToRadian(angleY)));
fovy=D3DX_PI/4;
aspect=800.0f/600.0f;
```

```
zNear=1.0f;
zFar=1000.0f;
hitObjects=NULL;
}

void camera::setHeight(float y){
location.y=y;
target.y=y;
}

void camera::setfovy(float fovy){
this->fovy=fovy;
}

void camera::setaspect(float aspect){
this->aspect=aspect;
}

void camera::setzNear(float zNear){
this->zNear=zNear;
}

void camera::setzFar(float zFar){
this->zFar=zFar;
}

void camera::rotateAngleX(float fRotateSpeed){
angleX=angleX+fRotateSpeed;
if(angleX>360.0f) angleX=0.0f;
if(angleX<0.0f) angleX=360.0f;
}

void camera::rotateAngleY(float fRotateSpeed){
angleY=angleY+fRotateSpeed;
if(angleY>360.0f) angleY=0.0f;
if(angleY<0.0f) angleY=360.0f;
}
void camera::rotateAngleZ(float fRotateSpeed){
angleZ=angleZ+fRotateSpeed;
if(angleZ>360.0f) angleZ=0.0f;
if(angleZ<0.0f) angleZ=360.0f;
}

void camera::adjustSpeed(float speed){
this->speed+=speed;
if(this->speed<0.0f) this->speed=0.0f;
```

```
}

void camera::adjustRotateSpeed(float rotateSpeed){
this->rotateSpeed+=rotateSpeed;
if(this->rotateSpeed<0.0f) this->rotateSpeed=0.0f;
}

void camera::setSpeed(float speed){
this->speed=speed;
if(this->speed<0.0f) this->speed=0.0f;
}

void camera::setRotateSpeed(float rotateSpeed){
this->rotateSpeed=this->rotateSpeed;
if(this->rotateSpeed<0.0f) this->rotateSpeed=0.0f;
}

void camera::setDistanceEye(float distanceEye){
this->distanceEye=distanceEye;
if(this->distanceEye < 1.0f) this->distanceEye=1.0f;
}

void camera::setRadius(float radius){
this->radius=radius;
if(this->radius<0) this->radius=0.0f;
}

void camera::setRadius2D(float radius2D){
this->radius2D=radius2D;
if(this->radius2D<0) this->radius2D=0.0f;
}

void camera::cameraForward(){
oldLocation=location;
oldTarget=target;
location.x -= speed * sin(D3DXToRadian(angleY));
location.z += speed * cos(D3DXToRadian(angleY));
target.x=location.x - (distanceEye*sin(D3DXToRadian(angleY)));
target.z=location.z + (distanceEye*cos(D3DXToRadian(angleY)));
}

void camera::cameraBackward(){
oldLocation=location;
oldTarget=target;
location.x += speed * sin(D3DXToRadian(angleY));
location.z -= speed * cos(D3DXToRadian(angleY));
target.x=location.x - (distanceEye*sin(D3DXToRadian(angleY)));
```

```
target.z=location.z + (distanceEye*cos(D3DXToRadian(angleY)));
}

void camera::cameraTurnLeft(){
oldLocation=location;
oldTarget=target;
angleY=angleY+rotateSpeed;
if (angleY>360.0f) angleY=0.0f;
target.x=location.x - (distanceEye*sin(D3DXToRadian(angleY)));
target.z=location.z + (distanceEye*cos(D3DXToRadian(angleY)));
}

void camera::cameraTurnRight(){
oldLocation=location;
oldTarget=target;
angleY=angleY-rotateSpeed;
if (angleY<0.0f) angleY=360.0f;
target.x=location.x - (distanceEye*sin(D3DXToRadian(angleY)));
target.z=location.z + (distanceEye*cos(D3DXToRadian(angleY)));
}

float camera::getX(){
return location.x;
}

float camera::getY(){
return location.y;
}

float camera::getZ(){
return location.z;
}

bool camera::hitObject(gameObject object){
float distance;
float minimumDistance;
distance=(location.x-object.centerPivotPoint.x)*(location.x-object.centerPivotPoint.x)
      +(location.y-object.centerPivotPoint.y)*(location.y-object.centerPivotPoint.y)
         +(location.z-object.centerPivotPoint.z)*(location.z-object.centerPivotPoint.z);
distance=sqrt(distance);
minimumDistance=radius+object.radius;
if(distance<minimumDistance){
return true;// Collide
}
return false;
}
```

```
void camera::hitObject(cell c){
int i;
bool boolean;
for(i=0;i<c.numObjects;i++){
        boolean=hitObject(c.objects[i]);
        if(boolean==true) addHitObjects(c.objects[i]);
        }
}

bool camera::hitObjectCylinder(gameObject object){
float distance;
float minimumDistance;
distance=(location.x-object.centerPivotPoint.x)*(location.x-object.centerPivotPoint.x)
      +(location.z-object.centerPivotPoint.z)*(location.z-object.centerPivotPoint.z);
distance=sqrt(distance);
minimumDistance=radius2D+object.radius2D;
if(distance<minimumDistance){
return true;// Collide
}
return false;
}

void camera::hitObjectCylinder(cell c){
int i;
bool boolean;
for(i=0;i<c.numObjects;i++){
        boolean=hitObjectCylinder(c.objects[i]);
        if(boolean==true) addHitObjects(c.objects[i]);
    }
}

void camera::addHitObjects(gameObject object){
int i;
vhitObjects.push_back(object);
numHitObjects=vhitObjects.size();
delete [] hitObjects;
hitObjects=new gameObject[numHitObjects];
for(i=0;i<numHitObjects;i++){
        hitObjects[i]=vhitObjects.at(i);
    }
}

void camera::clearHitObjects(){
numHitObjects=0;
vhitObjects.clear();
delete [] hitObjects;
hitObjects=NULL;
```

}

ตัวแปรและฟังก์ชัน

float angleX, angleY, angleZ; ค่าของมุมแกน x y z

float speed; ค่าความเร็วในการเดิน

float rotateSpeed; ค่าความร็วในการหมุนรอบแกน

vector3D location; ค่าตำแหน่งของกล้อง

vector3D oldLocation; ค่าตำแหน่งเก่าของกล้อง

vector3D target; ค่าตำแหน่งการมองของกล้อง

vector3D oldTarget; ค่าตำแหน่งการมองเก่าของกล้อง

float fovy, aspect, zNear, zFar; ค่าสำหรับการกำหนดค่าให้ Projection ในไดเรกเอกซ์ (DirectX)

float radius; ค่ารัศมีการชน

float radius2D; ค่ารัศมีการชน 2 มิติสำหรับทดสอบการชนแบบทรงกระบอก

float distanceEye; ค่าระยะสายตา

vector<gameObject> vhitObjects; ตัวแปรvectorชนิดgameObject

gameObject *hitObjects; พอยเตอร์gameObject

int numHitObjects; จำนวนgameObject

camera(); ตัวสร้างของคลาส

camera(float x, float y, float z); ตัวสร้างของคลาสรับค่าเป็นตำแหน่ง x y z

void setHeight(float y); ฟังก์ชันตั้งค่าความสูงของกล้อง(ตั้งค่าแกน y)

void setfovy(float fovy); ฟังก์ชันตั้งค่า field of view y

void setaspect(float aspect); ฟังก์ชันตั้งค่าอัตราส่วนความกว้าง ความสูง

void setzNear(float zNear); ฟังก์ชันตั้งค่าระยะที่ใกล้ที่สุดที่สามารถมองเห็นได้

void setzFar(float zFar); ฟังก์ชันตั้งค่าระยะที่ไกลที่สุดที่สามารถมองเห็นได้

void rotateAngleX(float fRotateSpeed); ฟังก์ชันการหมุนองศารอบแกน x

void rotateAngleY(float fRotateSpeed); ฟังก์ชันการหมุนองศารอบแกน y

void rotateAngleZ(float fRotateSpeed); ฟังก์ชันการหมุนองศารอบแกน z

void adjustSpeed(float speed); ฟังก์ชันการปรับความเร็วในการเดิน

void adjustRotateSpeed(float rotateSpeed); ฟังก์ชันการปรับความเร็วในการหมุน

void setSpeed(float speed); ฟังก์ชันการตั้งค่าความเร็วการเดิน

void setRotateSpeed(float rotateSpeed); ฟังก์ชันการตั้งค่าความเร็วในการหมุน

void setDistanceEye(float distanceEye); ฟังก์ชันการตั้งค่าระยะสายตา

void setRadius(float radius); ฟังก์ชันการตั้งค่ารัศมีการชน

void setRadius2D(float radius2D); ฟังก์ชันการตั้งค่ารัศมีการชนแบบ 2 มิติ

void cameraForward(); ฟังก์ชันการเดินไปข้างหน้าของกล้อง

void cameraBackward(); ฟังก์ชันการเดินไปข้างหลังของกล้อง

void cameraTurnLeft(); ฟังก์ชันการหมุนไปทางซ้ายของกล้อง

void cameraTurnRight(); ฟังก์ชันการเดินไปทางขวาของกล้อง

float getX(); ฟังก์ชันการรับค่าตำแหน่ง x

float getY(); ฟังก์ชันการรับค่าตำแหน่ง y

float getZ(); ฟังก์ชันการรับค่าตำแหน่ง z

bool hitObject(gameObject object); ฟังก์ชันทดสอบการชนรับค่าGameObject

void hitObject(cell c); ฟังก์ชันทดสอบการชนรับค่าcell

bool hitObjectCylinder(gameObject object); ฟังก์ชันทดสอบการชนรับค่าGameObject แบบทรงกระบอก

void hitObjectCylinder(cell c); ฟังก์ชันทดสอบการชนรับค่าcell แบบทรงกระบอก

void addHitObjects(gameObject object); ฟังก์ชันการเพิ่มข้อมูลgameObjectในอาเรย์ของวัตถุที่ชน

void clearHitObjects(); ฟังก์ชันลบในอาเรย์ของgameObject

8. คลาส gridGame

```
class gridGame{
public:
int GRID_SIZE_BITS;// = 9;
int GRID_SIZE;// = 1 << GRID_SIZE_BITS;
cell *grid;  //cell[] grid;
rectangles mapBound;
int gridWidth;
int gridHeight;
int numCells;
vector<gameObject> vobjects;
gameObject *objects;
int numObjects;

gridGame();
gridGame(gameObject object);
gridGame(vector<gameObject> vobjects);
gridGame(gameObject objects[], int numObjects);
int convertMapXtoGridX(int x);
int convertMapYtoGridY(int y);
cell* getCell(camera c);
cell* getCell(gameObject object);
cell* getCell(vector3D v);
cell* getCell(float x, float z);
cell* getCell(int x, int y);
void updateVisible(camera c);
rectangles calcBound();
};
```

รายละเอียดของฟังก์ชันในคลาส

```
gridGame::gridGame(){
GRID_SIZE_BITS = 9;
GRID_SIZE = 1 << GRID_SIZE_BITS; // 512
this->objects=NULL;
}
gridGame::gridGame(gameObject object){
GRID_SIZE_BITS = 9;
GRID_SIZE = 1 << GRID_SIZE_BITS;
int i;
this->objects=new gameObject();
this->numObjects=1;
this->vobjects.push_back(object);
```

```
*(this->objects)=object;
calcBound();
gridWidth=(mapBound.width>>GRID_SIZE_BITS)+1;
gridHeight=(mapBound.height>>GRID_SIZE_BITS)+1;
grid=new cell[gridWidth*gridHeight];
for(i=0;i<this->numObjects;i++){
cell *c=getCell(this->objects[i]);
if(c->inGrid!=false) { c->addObject(this->objects[i]); }
}
}

gridGame::gridGame(vector<gameObject> vobjects){// <-- Start Function
GRID_SIZE_BITS = 9;
GRID_SIZE = 1 << GRID_SIZE_BITS;
int i;
this->objects=new gameObject[vobjects.size()];
this->numObjects=vobjects.size();
for(i=0;i<vobjects.size();i++){
this->vobjects.push_back(vobjects.at(i));
this->objects[i]=vobjects.at(i);
}
calcBound();
gridWidth=(mapBound.width>>GRID_SIZE_BITS)+1;
gridHeight=(mapBound.height>>GRID_SIZE_BITS)+1;
grid=new cell[gridWidth*gridHeight];
for(i=0;i<this->numObjects;i++){
cell *c=getCell(this->objects[i]);
if(c->inGrid!=false) { c->addObject(this->objects[i]); }
}
}// <-- End Function

gridGame::gridGame(gameObject objects[], int numObjects){// <-- Start Function
GRID_SIZE_BITS = 9;
GRID_SIZE = 1 << GRID_SIZE_BITS;
int i;
objects=new gameObject[numObjects];
this->numObjects=numObjects;
for(i=0;i<this->numObjects;i++){
this->vobjects.push_back(objects[i]);
this->objects[i]=objects[i];
```

```
}
calcBound();
gridWidth=(mapBound.width>>GRID_SIZE_BITS)+1;
gridHeight=(mapBound.height>>GRID_SIZE_BITS)+1;
grid=new cell[gridWidth*gridHeight];
for(i=0;i<this->numObjects;i++){
cell *c=getCell(this->objects[i]);
if(c->inGrid!=false) { c->addObject(this->objects[i]); }
}
}// <-- End Function

int gridGame::convertMapXtoGridX(int x){
return (x-mapBound.x)>>GRID_SIZE_BITS;
}

int gridGame::convertMapYtoGridY(int y){
return (y-mapBound.y)>>GRID_SIZE_BITS;
}

cell* gridGame::getCell(camera c){// <-- Start Function
int x=convertMapXtoGridX((int)c.location.x);
int z=convertMapYtoGridY((int)c.location.z);
return getCell(x, z);
}// <-- End Function

cell* gridGame::getCell(gameObject object){// <-- Start Function
int x=convertMapXtoGridX((int)object.centerPivotPoint.x);
int z=convertMapYtoGridY((int)object.centerPivotPoint.z);
return getCell(x, z);
}// <-- End Function

cell* gridGame::getCell(vector3D v){// <-- Start Function
int x=convertMapXtoGridX((int)v.x);
int z=convertMapYtoGridY((int)v.z);
return getCell(x, z);
}// <-- End Function

cell* gridGame::getCell(float x, float z){// <-- Start Function
int pointx=convertMapXtoGridX((int)x);
int pointz=convertMapYtoGridY((int)z);
return getCell(pointx, pointz);
}// <-- End Function

cell* gridGame::getCell(int x, int y){// <-- Start Function
if (x < 0 || y < 0 || x >= gridWidth || y >= gridHeight) {
```

```
return &(cell(false));
}
return &(grid[x + y * gridWidth]);
}// <-- End Function


void gridGame::updateVisible(camera c){// <-- Start Function
int i, j;
int x=convertMapXtoGridX((int)c.location.x);
int z=convertMapYtoGridY((int)c.location.z);
for(i=0;i<gridWidth*gridHeight;i++){
grid[i].visible=false;
}
for(i=x-1;i<=x+1;i++){
        for(j=z-1;j<z+1;j++){
        grid[i+j*gridWidth].visible=true;
        }
}
}// <-- End Function

rectangles gridGame::calcBound(){ // <-- Start Function
int i, j, k;
int maxX, minX, maxZ, minZ;
int x, z;
maxX=INT_MIN; maxZ=INT_MIN;
minX=INT_MAX; minZ=INT_MAX;
for(i=0;i<vobjects.size();i++){
        for(j=0;j<vobjects.at(i).numPolygons;j++){
                for(k=0;k<vobjects.at(i).p[j].numVertices;k++){
                        x=(int)floor(vobjects.at(i).p[j].v[k].x);
                        z=(int)floor(vobjects.at(i).p[j].v[k].z);
                        minX=__min(minX, x);
                        maxX=__max(maxX, x);
                        minZ=__min(minZ, z);
                        maxZ=__max(maxZ, z);
                }
        }
}
mapBound.x=minX;
mapBound.y=minZ;
mapBound.width=maxX-minX;
mapBound.height=maxZ-minZ;
return rectangles(mapBound.x, mapBound.y, mapBound.width, mapBound.height);
} // <-- End Function
```

ตัวแปรและฟังก์ชัน

int GRID_SIZE_BITS; ตัวแปรค่าขนาดของbitที่ใช้คำนวณขนาดของกริด

int GRID_SIZE; ตัวแปรค่าขนาดของกริด

cell *grid; ตัวแปรอาเรย์ของcell

rectangles mapBound; ตัวแปรrectangles

int gridWidth; ตัวแปรจำนวนกริดด้านกว้าง

int gridHeight; ตัวแปรจำนวนกริดด้านสูง

int numCells; ตัวแปรจำนวนcellในกริด

vector<gameObject> vobjects; ตัวแปรvectorชนิดgameObject

gameObject *objects; ตัวแปรอาเรย์ของ gameObject

int numObjects; ตัวแปรจำนวนgameObject

gridGame(); ตัวสร้างของคลาส

gridGame(gameObject object); ตัวสร้างของคลาสรับค่าgameObject

gridGame(vector<gameObject> vobjects); ตัวสร้างของคลาสรับค่าvectorของ gameObject

gridGame(gameObject objects[], int numObjects); ตัวสร้างของคลาสรับค่าอาเรย์ของ gameObjectและจำนวนgameObject

int convertMapXtoGridX(int x); ฟังก์ชันการปรับตำแหน่งของxเป็นตำแหน่งของกริดด้าน x

int convertMapYtoGridY(int y); ฟังก์ชันการปรับตำแหน่งของyเป็นตำแหน่งของกริดด้าน y

cell* getCell(camera c); ฟังก์ชันรีเทิร์นcellในกริดรับค่าเป็นcamera

cell* getCell(gameObject object); ฟังก์ชันรีเทิร์นcellในกริดรับค่าเป็นgameObject

cell* getCell(vector3D v); ฟังก์ชันรีเทิร์นcellในกริดรับค่าเป็นvector3D

cell* getCell(float x, float z); ฟังก์ชันรีเทิร์นcellในกริดรับค่าเป็นfloat2ค่า

cell* getCell(int x, int y); ฟังก์ชันรีเทิร์นcellในกริดรับค่าเป็นint2ค่า

void updateVisible(camera c); ฟังก์ชันการปรับให้cellเป็นvisible

rectangles calcBound(); ฟังก์ชันการคำนวณขอบเขตของกริด

9. คลาส render

```
class render{
public:
HWND hwnd;
LPDIRECT3D8 pD3D;
LPDIRECT3DDEVICE8 pd3dDevice;
D3DXMATRIX matCameraView;    // Camera Data
D3DXMATRIX matProj;          // Projection Data
D3DXVECTOR3 d3dxVecLocation;
D3DXVECTOR3 d3dxVecTarget;
LPDIRECT3DVERTEXBUFFER8 lpVertexBuffer;// = NULL;
D3DPRESENT_PARAMETERS d3dpp;
D3DDISPLAYMODE       d3ddm;
RECT rcWindowBounds;    // Saved window bounds for mode switches
RECT rcWindowClient;    // Saved client area size for mode switches
void *lpVertices;
HDC hdc;

render();
render( HWND hWnd, camera c );
HRESULT hrInit3D( HWND hWnd, camera c ) ;
void setCamera(camera c);
void setProjection();
void setProjection(float aspect, float zNear, float zFar);
void setProjection(camera c);
void clearScene();
void clearScene(int red, int green, int blue);
void drawtoBackBuffer(vector3D v0);
void drawtoBackBuffer(vector3D v0, vector3D v1);
void drawtoBackBuffer(polygon p);
void drawtoBackBuffer(gameObject go);
void drawtoBackBuffer(cell c);
void drawtoBackBuffer(gridGame gg);
void draw(camera c, polygon p);
void draw(camera c, gameObject go);
void draw(camera c, cell cl);
void draw(camera c, gridGame gg);
void drawText(HWND hwnd, int x, int y, char str[]);
void drawText(HWND hwnd, int x, int y, char str[], COLORREF c, COLORREF bk,
bool transparent);
void drawText(HWND hwnd, int x, int y, string str, COLORREF c, COLORREF bk,
bool transparent);
void present();
void vCleanup(void);
};
```

รายละเอียดของฟังก์ชันในคลาส

```
render::render(){}

render::render( HWND hWnd, camera c ) {
this->hwnd=hWnd;
hrInit3D( hWnd, c ) ;
}

HRESULT render::hrInit3D( HWND hWnd, camera c )
{
// Create the D3D object.
if( NULL == ( pD3D = Direct3DCreate8( D3D_SDK_VERSION ) ) )
return E_FAIL;
// Get adapter settings
if( FAILED( pD3D->GetAdapterDisplayMode( D3DADAPTER_DEFAULT,
&d3ddm ) ) )
return E_FAIL;
// Get the size of the window
GetWindowRect( hWnd, &rcWindowBounds );
GetClientRect( hWnd, &rcWindowClient );
// Clear out device parameters
ZeroMemory( &d3dpp, sizeof(d3dpp) );
// Setup device parameters
d3dpp.hDeviceWindow             = hWnd;
d3dpp.BackBufferWidth           = (rcWindowClient.right -
rcWindowClient.left);
d3dpp.BackBufferHeight          = (rcWindowClient.bottom -
rcWindowClient.top);
d3dpp.BackBufferFormat          = d3ddm.Format;
d3dpp.BackBufferCount           = 1;
d3dpp.SwapEffect                = D3DSWAPEFFECT_FLIP;
d3dpp.MultiSampleType           = D3DMULTISAMPLE_NONE;
d3dpp.EnableAutoDepthStencil    = TRUE;
d3dpp.AutoDepthStencilFormat    = D3DFMT_D16;
d3dpp.Windowed                  = TRUE;
// Create the D3DDevice
if( FAILED( pD3D->CreateDevice( D3DADAPTER_DEFAULT,
                    D3DDEVTYPE_HAL,
                    hWnd,
                    D3DCREATE_SOFTWARE_VERTEXPROCESSING,
                    &d3dpp,
                    &pd3dDevice ) ) )
{
// Try 32-bits
d3dpp.AutoDepthStencilFormat    = D3DFMT_D32;
if( FAILED( pD3D->CreateDevice( D3DADAPTER_DEFAULT,
                D3DDEVTYPE_HAL,
```

```
                hWnd,
                D3DCREATE_SOFTWARE_VERTEXPROCESSING,
                &d3dpp,
                &pd3dDevice ) ) )
{
return E_FAIL;
}
}
// Turn on the zbuffer
pd3dDevice->SetRenderState( D3DRS_ZENABLE, TRUE );
// Turn on ambient lighting
pd3dDevice->SetRenderState( D3DRS_AMBIENT, 0xffffffff );
// Turn off D3D lighting
pd3dDevice->SetRenderState( D3DRS_LIGHTING, FALSE );
// Setup Camera View
d3dxVecLocation.x=c.location.x;
d3dxVecLocation.y=c.location.y;
d3dxVecLocation.z=c.location.z;
d3dxVecTarget.x=c.target.x;
d3dxVecTarget.y=c.target.y;
d3dxVecTarget.z=c.target.z;
D3DXMatrixLookAtLH( &matCameraView,
                    &d3dxVecLocation,
                    &d3dxVecTarget,
                    &D3DXVECTOR3( 0.0f, 1.0f, 0.0f ) );
// Tell the device to use the camera view for the viewport
pd3dDevice->SetTransform( D3DTS_VIEW, &matCameraView );
// Setup Projection
D3DXMatrixPerspectiveFovLH( &matProj, D3DX_PI/4, 800.0f/600.0f, 1.0f, 1000.0f
);
// Tell the device to use the above matrix for projection
pd3dDevice->SetTransform( D3DTS_PROJECTION, &matProj );
// Create the vertex buffer
if(FAILED(pd3dDevice->CreateVertexBuffer(3*sizeof(vector3D),
0, D3DFVF_MYVERTEX, D3DPOOL_DEFAULT, &lpVertexBuffer))) {
return E_FAIL;
}
return S_OK;
}

void render::setCamera(camera c){
d3dxVecLocation.x=c.location.x;
d3dxVecLocation.y=c.location.y;
d3dxVecLocation.z=c.location.z;
d3dxVecTarget.x=c.target.x;
d3dxVecTarget.y=c.target.y;
d3dxVecTarget.z=c.target.z;
```

```
D3DXMatrixIdentity(&matCameraView);
D3DXMatrixLookAtLH( &matCameraView,
                    &d3dxVecLocation,
                    &d3dxVecTarget,
                    &D3DXVECTOR3( 0.0f, 1.0f, 0.0f ) );
// Tell the device to use the camera view for the viewport
pd3dDevice->SetTransform( D3DTS_VIEW, &matCameraView );

// Setup Projection
D3DXMatrixPerspectiveFovLH( &matProj, D3DX_PI/4, 800.0f/600.0f, 1.0f, 1000.0f
);
// Tell the device to use the above matrix for projection
pd3dDevice->SetTransform( D3DTS_PROJECTION, &matProj );
}

void render::setProjection(){
D3DXMatrixIdentity(&matProj);
// Setup Projection
D3DXMatrixPerspectiveFovLH( &matProj, D3DX_PI/4, 800.0f/600.0f, 1.0f, 1000.0f
);
// Tell the device to use the above matrix for projection
pd3dDevice->SetTransform( D3DTS_PROJECTION, &matProj );
}

void render::setProjection(float aspect, float zNear, float zFar){
D3DXMatrixIdentity(&matProj);
// Setup Projection
D3DXMatrixPerspectiveFovLH( &matProj, D3DX_PI/4, aspect, zNear, zFar );
// Tell the device to use the above matrix for projection
pd3dDevice->SetTransform( D3DTS_PROJECTION, &matProj );
}

void render::setProjection(camera c){
D3DXMatrixIdentity(&matProj);
// Setup Projection
D3DXMatrixPerspectiveFovLH( &matProj, c.fovy, c.aspect, c.zNear, c.zFar );
// Tell the device to use the above matrix for projection
pd3dDevice->SetTransform( D3DTS_PROJECTION, &matProj );
}

void render::clearScene(){
// Clear the backbuffer and the zbuffer
pd3dDevice->Clear( 0, NULL, D3DCLEAR_TARGET|D3DCLEAR_ZBUFFER,
D3DCOLOR_XRGB(0, 0, 0), 1.0f, 0 );
//pd3dDevice->Present( NULL, NULL, NULL, NULL );
}
```

```
void render::clearScene(int red, int green, int blue){
// Clear the backbuffer and the zbuffer
pd3dDevice->Clear( 0, NULL, D3DCLEAR_TARGET|D3DCLEAR_ZBUFFER,
D3DCOLOR_XRGB(red, green, blue), 1.0f, 0 );
//pd3dDevice->Present( NULL, NULL, NULL, NULL );
}

void render::drawtoBackBuffer(vector3D v0){
vector3D v[1];
v[0]=v0;
polygon p(v, 1);
pd3dDevice->BeginScene();
pd3dDevice->SetRenderState( D3DRS_CULLMODE, D3DCULL_NONE );
lpVertexBuffer->Lock(0, 0, (BYTE**)&lpVertices, 0);
memcpy(lpVertices, p.v, 1*sizeof(vector3D));
lpVertexBuffer->Unlock();

// Set the vertex buffer to render
pd3dDevice->SetStreamSource(0, lpVertexBuffer, sizeof(vector3D));
// Set the vertex format
pd3dDevice->SetVertexShader(D3DFVF_MYVERTEX);
// Draw the primitive D3DPT_LINELIST  D3DPT_TRIANGLELIST
pd3dDevice->DrawPrimitive(D3DPT_POINTLIST, 0, 1);
// Stop Rendering
pd3dDevice->EndScene();
}

void render::drawtoBackBuffer(vector3D v0, vector3D v1){
vector3D v[2];
v[0]=v0;
v[1]=v1;
polygon p(v, 2);
pd3dDevice->BeginScene();
pd3dDevice->SetRenderState( D3DRS_CULLMODE, D3DCULL_NONE );
lpVertexBuffer->Lock(0, 0, (BYTE**)&lpVertices, 0);
memcpy(lpVertices, p.v, 2*sizeof(vector3D));
lpVertexBuffer->Unlock();
// Set the vertex buffer to render
pd3dDevice->SetStreamSource(0, lpVertexBuffer, sizeof(vector3D));
// Set the vertex format
pd3dDevice->SetVertexShader(D3DFVF_MYVERTEX);
// Draw the primitive D3DPT_LINELIST  D3DPT_TRIANGLELIST
pd3dDevice->DrawPrimitive(D3DPT_LINELIST, 0, 1);
// Stop Rendering
pd3dDevice->EndScene();
}
```

```
void render::drawtoBackBuffer(polygon p){
if(p.numVertices==1){
drawtoBackBuffer(p.v[0]);
}
else if(p.numVertices==2){
        drawtoBackBuffer(p.v[0], p.v[1]);
        }
        else{
        pd3dDevice->BeginScene();
        pd3dDevice->SetRenderState( D3DRS_CULLMODE, D3DCULL_NONE );
        lpVertexBuffer->Lock(0, 0, (BYTE**)&lpVertices, 0);
        memcpy(lpVertices, p.v, 3*sizeof(vector3D));
        lpVertexBuffer->Unlock();
        // Set the vertex buffer to render
        pd3dDevice->SetStreamSource(0, lpVertexBuffer, sizeof(vector3D));
        // Set the vertex format
        pd3dDevice->SetVertexShader(D3DFVF_MYVERTEX);
        // Draw the primitive D3DPT_LINELIST  D3DPT_TRIANGLELIST
        pd3dDevice->DrawPrimitive(D3DPT_TRIANGLELIST, 0, 1);
        // Stop Rendering
        pd3dDevice->EndScene();
        }
}

void render::drawtoBackBuffer(gameObject go){
int i=0;
polygon p;
for(i=0;i<go.numPolygons;i++){
        p=go.p[i];
        drawtoBackBuffer(p);
}
}

void render::drawtoBackBuffer(cell c){
int i=0;
gameObject go;
for(i=0;i<c.numObjects;i++){
        go=c.objects[i];
        drawtoBackBuffer(go);
}
}

void render::drawtoBackBuffer(gridGame gg){
int i=0;
cell c;
for(i=0;i<gg.numCells;i++){
        if(gg.grid[i].visible==true){
```

```
        c=gg.grid[i];
        drawtoBackBuffer(c);
        }
}
}
void render::draw(camera c, polygon p){
setCamera(c);
setProjection();
clearScene();
drawtoBackBuffer(p);
// Output the scene
pd3dDevice->Present( NULL, NULL, NULL, NULL );
}

void render::draw(camera c, gameObject go){
setCamera(c);
setProjection();
clearScene();
drawtoBackBuffer(go);
// Output the scene
pd3dDevice->Present( NULL, NULL, NULL, NULL );
}

void render::draw(camera c, cell cl){
setCamera(c);
setProjection();
clearScene();
drawtoBackBuffer(cl);
// Output the scene
pd3dDevice->Present( NULL, NULL, NULL, NULL );
}

void render::draw(camera c, gridGame gg){
setCamera(c);
setProjection();
clearScene();
drawtoBackBuffer(gg);
// Output the scene
pd3dDevice->Present( NULL, NULL, NULL, NULL );
}

void render::drawText(HWND hwnd, int x, int y, char str[])
{
hdc=GetDC(hwnd);
SetTextColor(hdc, RGB(255, 0, 0));
SetBkColor(hdc, RGB(0, 255, 0));
//SetBkMode(hdc, TRANSPARENT);
```

```
TextOut(hdc, x, y, str, strlen(str));
ReleaseDC(hwnd, hdc);
}

void render::drawText(HWND hwnd, int x, int y, char str[], COLORREF c,
COLORREF bk, bool transparent)
{
HDC hdc;
hdc=GetDC(hwnd);
SetTextColor(hdc, c);
SetBkColor(hdc, bk);
if(transparent==true){
SetBkMode(hdc, TRANSPARENT);
}
TextOut(hdc, x, y, str, strlen(str));
ReleaseDC(hwnd, hdc);
}

void render::drawText(HWND hwnd, int x, int y, string str, COLORREF c,
COLORREF bk, bool transparent)
{
char* ch;
HDC hdc;
hdc=GetDC(hwnd);
SetTextColor(hdc, c);
SetBkColor(hdc, bk);
if(transparent==true){
SetBkMode(hdc, TRANSPARENT);
}
ch = new char[str.length() + 1];
strcpy(ch, str.c_str());
TextOut(hdc, x, y, ch, str.length() + 1);
delete [] ch;
ReleaseDC(hwnd, hdc);
}

void render::present(){
pd3dDevice->Present( NULL, NULL, NULL, NULL );
}

void render::vCleanup(void)
{
if( lpVertexBuffer != NULL )
lpVertexBuffer->Release();
if( pd3dDevice != NULL )
pd3dDevice->Release();
if( pD3D != NULL )
```

```
pD3D->Release();
}
```

ตัวแปรและฟังก์ชัน

HWND hwnd; ตัวแปรhandle

LPDIRECT3D8 pD3D; ตัวแปรไดเรกสามดี

LPDIRECT3DDEVICE8 pd3dDevice; ตัวแปรไดเรกสามดีดีไวซ์

D3DXMATRIX matCameraView; ตัวแปรเมตริกซ์

D3DXMATRIX matProj; ตัวแปรเมตริกซ์

D3DXVECTOR3 d3dxVecLocation; ตัวแปรเวกเตอร์ใช้แทนตำแหน่งlocationของกล้อง

D3DXVECTOR3 d3dxVecTarget; ตัวแปรเวกเตอร์ใช้แทนตำแหน่งจุดที่มองของกล้อง

LPDIRECT3DVERTEXBUFFER8 lpVertexBuffer; ตัวแปรvertexBuffer

D3DPRESENT_PARAMETERS d3dpp; ตัวแปรใช้แทนการตั้งค่าของไดเรกสามดี

D3DDISPLAYMODE d3ddm; ตัวแปรข้อมูลของการแสดงผล

RECT rcWindowBounds; ตัวแปรเก็บข้อมูลของความกว้าง สูงทั้งหมดของวินโดส์

RECT rcWindowClient; ตัวแปรเก็บข้อมูลของความกว้าง สูงด้านในของวินโดส์

void *lpVertices; พอยเตอร์voidสำหรับรับค่าvertexbuffer

HDC hdc; ดีไวซ์คอนเทกซ์

render(); ตัวสร้างของคลาส

render(HWND hWnd, camera c); ตัวสร้างของคลาสรับค่าhandleและcamera

HRESULT hrInit3D(HWND hWnd, camera c) ; ฟังก์ชันเริ่มต้นไดเรกสามดี

void setCamera(camera c); ฟังก์ชันการสร้างมุมกล้อง

void setProjection(); ฟังก์ชันการสร้างprojection

void setProjection(float aspect, float zNear, float zFar); ฟังก์ชันการสร้างprojection รับค่าอัตราส่วนกว้างสูง ระยะใกล้สุดของสายตาและระยะไกลสุด

void setProjection(camera c); ฟังก์ชันการสร้างprojectionรับค่าcamera

void clearScene(); ฟังก์ชันล้างหน้าจอ

void clearScene(int red, int green, int blue); ฟังก์ชันล้างหน้าจอรับค่าสี3สี

void drawtoBackBuffer(vector3D v0); ฟังก์ชันวาดรูปลงไปในbackbufferรับค่า vector3D

void drawtoBackBuffer(vector3D v0, vector3D v1); ฟังก์ชันวาดรูปลงไปใน backbufferรับค่าvector3D 2 ค่า

void drawtoBackBuffer(polygon p); ฟังก์ชันวาดรูปลงไปในbackbufferรับค่าpolygon

void drawtoBackBuffer(gameObject go); ฟังก์ชันวาดรูปลงไปในbackbufferรับค่า gameObject

void drawtoBackBuffer(cell c); ฟังก์ชันวาดรูปลงไปในbackbufferรับค่าcell

void drawtoBackBuffer(gridGame gg); ฟังก์ชันวาดรูปลงไปในbackbufferรับค่า polygon

void draw(camera c, polygon p); ฟังก์ชันวาดรูปรับค่าcamera, polygon

void draw(camera c, gameObject go); ฟังก์ชันวาดรูปรับค่าcemera, gameObject

void draw(camera c, cell cl); ฟังก์ชันวาดรูปรับค่าcamera, cell

void draw(camera c, gridGame gg); ฟังก์ชันวาดรูปรับค่าcamera, gridGame

void drawText(HWND hwnd, int x, int y, char str[]); ฟังก์ชันวาดข้อความรับค่า handle ตำแหน่ง x y ข้อความ

void drawText(HWND hwnd, int x, int y, char str[], COLORREF c, COLORREF bk, bool transparent); ฟังก์ชันวาดข้อความรับค่าhandle ตำแหน่ง x y ข้อความ สีตัวอักษร สีพื้น บูลีนค่าพื้นหลังโปร่งใส

void drawText(HWND hwnd, int x, int y, string str, COLORREF c, COLORREF bk, bool transparent); ฟังก์ชันวาดข้อความรับค่าhandle ตำแหน่ง x y ข้อความ สีตัวอักษร สีพื้น บูลีนค่าพื้นหลังโปร่งใส

void present(); ฟังก์ชันการเรียกโชว์รูปออกหน้าจอ

void vCleanup(void); ฟังก์ชันการปล่อยตัวแปร

10. คลาส socketObject

```
class socketObject
{
public:
HWND hwnd;
SOCKET skSocket;
int iStatus;
sockaddr_in saServerAddress;
sockaddr sockaddrIP;
sockaddr_in sockaddr_inIP;

socketObject();
socketObject(HWND hwnd);
~socketObject();
void setHWND(HWND hwnd);
bool Accept(socketObject& skAcceptSocket);
int Listen(void);
int Bind(int iPort);
bool Connect(char* chpServerAddress, int iPort);
void Disconnect();
int Recv(char *chpBuffer, int iBufLen, int iFlags);
int Send(char *chpBuffer, int iBufLen, int iFlags);
int getIPb1();
int getIPb2();
int getIPb3();
int getIPb4();
void getIPArrayChar(char **szIP);
string getIPString();
};
```

รายละเอียดของฟังก์ชันในคลาส

```
socketObject::socketObject()
{
WSADATA wsaData;
WORD wVersionRequested;
wVersionRequested = MAKEWORD( 2, 0 );
skSocket = INVALID_SOCKET;
iStatus = WSAStartup(wVersionRequested, &wsaData);
}

socketObject::socketObject(HWND hwnd)
{
this->hwnd=hwnd;
WSADATA wsaData;
WORD wVersionRequested;
```

```
wVersionRequested = MAKEWORD( 2, 0 );
skSocket = INVALID_SOCKET;
iStatus = WSAStartup(wVersionRequested, &wsaData);
}

socketObject::~socketObject()
{
Disconnect();
}

void socketObject::setHWND(HWND hwnd)
{
this->hwnd=hwnd;
}

bool socketObject::Accept( socketObject &skAcceptSocket )
{
int iClientSize = sizeof(sockaddr_in);
skAcceptSocket.skSocket = accept( skSocket, (struct
sockaddr*)&skAcceptSocket.sockaddr_inIP, &iClientSize );
if( skAcceptSocket.skSocket == INVALID_SOCKET )
{
return false;
}
else
{
memcpy(&skAcceptSocket.sockaddrIP, &skAcceptSocket.sockaddr_inIP,
sizeof(sockaddr_inIP));
return true;
}
}

int socketObject::Listen( void )
{
int i;
i=listen( skSocket, 32 );
WSAAsyncSelect(skSocket, hwnd, 3000, FD_READ | FD_WRITE | FD_CONNECT |
FD_CLOSE | FD_ACCEPT);
return i;
}

int socketObject::Bind(int iPort)
{
//sockaddr_in saServerAddress;
```

```
skSocket = socket(AF_INET, SOCK_STREAM, 0);
if(skSocket == INVALID_SOCKET)
{
return false;
}
memset(&saServerAddress, 0, sizeof(sockaddr_in));
saServerAddress.sin_family = AF_INET;
saServerAddress.sin_addr.s_addr = htonl(INADDR_ANY);
saServerAddress.sin_port = htons(iPort);
if( bind(skSocket, (sockaddr*) &saServerAddress, sizeof(sockaddr)) ==
SOCKET_ERROR)
{
Disconnect();
return false;
}
else
return true;
}

bool socketObject::Connect(char* chpServerAddress, int iPort)
{
struct sockaddr_in serv_addr;
LPHOSTENT lphost;
memset(&serv_addr, 0, sizeof(sockaddr_in));
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = inet_addr(chpServerAddress);
if (serv_addr.sin_addr.s_addr == INADDR_NONE)
{
lphost = gethostbyname(chpServerAddress);
if (lphost != NULL)
serv_addr.sin_addr.s_addr = ((LPIN_ADDR)lphost->h_addr)->s_addr;
else
{
WSASetLastError(WSAEINVAL);
return FALSE;
}
}
serv_addr.sin_port = htons(iPort);
// Open the socket
skSocket = socket(AF_INET, SOCK_STREAM, 0);
if(skSocket == INVALID_SOCKET)
{
return false;
}
```

```
int err = connect(skSocket, (struct sockaddr*)&serv_addr, sizeof(sockaddr));
WSAAsyncSelect(skSocket, hwnd, 3000, FD_READ | FD_WRITE | FD_CONNECT
| FD_CLOSE | FD_ACCEPT);
if(err == SOCKET_ERROR)
{
Disconnect();
return false;
}
return true;
}

void socketObject::Disconnect()
{
if(skSocket != INVALID_SOCKET)
{
closesocket(skSocket);
skSocket = INVALID_SOCKET;
}
}

int socketObject::Recv( char *chpBuffer, int iBufLen, int iFlags)
{
return recv(skSocket, chpBuffer, iBufLen, iFlags);
}

int socketObject::Send(char *chpBuffer, int iBufLen, int iFlags)
{
return send(skSocket, chpBuffer, iBufLen, iFlags);
}

int socketObject::getIPb1(){
int i;
i=(int)sockaddr_inIP.sin_addr.S_un.S_un_b.s_b1;
return i;
}

int socketObject::getIPb2(){
int i;
i=(int)sockaddr_inIP.sin_addr.S_un.S_un_b.s_b2;
return i;
}

int socketObject::getIPb3(){
int i;
```

```
i=(int)sockaddr_inIP.sin_addr.S_un.S_un_b.s_b3;
return i;
}

int socketObject::getIPb4(){
int i;
i=(int)sockaddr_inIP.sin_addr.S_un.S_un_b.s_b4;
return i;
}

void socketObject::getIPArrayChar(char **szIP){
*szIP=new char[16];
sprintf(*szIP, "%d.%d.%d.%d", getIPb1(), getIPb2(), getIPb3(), getIPb4());
}

string socketObject::getIPString(){
char szIP[16];
string str;
sprintf(szIP, "%d.%d.%d.%d", getIPb1(), getIPb2(), getIPb3(), getIPb4());
szIP[15]='\0';
str=szIP;
return str;
}
```

ตัวแปรและฟังก์ชัน

HWND hwnd; ตัวแปร handle ของ windows

SOCKET skSocket; ตัวแปรซ็อกเก็ต

int iStatus; ตัวแปรint ใช้รับค่าการส่งคืนมาจากฟังก์ชันการเริ่มต้น winsock

sockaddr_in saServerAddress; ตัวแปรโครงสร้าง sockaddr_in

sockaddr sockaddrIP; ตัวแปรโครงสร้าง sockaddr

sockaddr_in sockaddr_inIP; ตัวแปรโครงสร้าง sockaddr_in

socketObject(); ตัวสร้างของคลาส

socketObject(HWND hwnd); ตัวสร้างของคลาสรับค่าเป็นตัวแปร handle ของ windows

~socketObject(); ตัวทำลายของคลาส

void setHWND(HWND hwnd); ฟังก์ชันใช้ตั้งค่า handle ของ windows

bool Accept(socketObject& skAcceptSocket); ฟังก์ชัน accept

int Listen(void); ฟังก์ชัน listen

int Bind(int iPort); ฟังก์ชัน bind

bool Connect(char* chpServerAddress, int iPort); ฟังก์ชัน connect

void Disconnect(); ฟังก์ชัน disconnect

int Recv(char *chpBuffer, int iBufLen, int iFlags); ฟังก์ชัน recv

int Send(char *chpBuffer, int iBufLen, int iFlags); ฟังก์ชัน send

int getIPb1(); ฟังก์ชันรับค่า ip ส่วนที่1

int getIPb2(); ฟังก์ชันรับค่า ip ส่วนที่1

int getIPb3(); ฟังก์ชันรับค่า ip ส่วนที่3

int getIPb4(); ฟังก์ชันรับค่า ip ส่วนที่4

void getIPArrayChar(char **szIP); ฟังก์ชันรับค่า ip ใส่ตัวแปร char* ที่ส่งเข้ามา

string getIPString(); ฟังก์ชันรับค่า ip รีเทิร์นเป็น string

ในฟังก์ชันlistenกับconnectจะมีการเรียกฟังก์ชัน WSAAsyncSelect ของ winsock ฟังก์ชันนี้จะบอกให้ WS2_32.DLL ส่งแมสเสจสู่ windows เจ้าของแมสเสจ โดยในเฟรมเวิร์คนี้ให้ส่งเป็นแมสเสจ 3000 และมี low word ของ lparam แสดงเหตุการณ์ ในฟังก์ชันขอให้แสดงเหตุการณ์ 4 เหตุการณ์คือ FD_READ FD_WRITE FD_CLOSE FD_ACCEPT ความหมายคือ

FD_READ socket พร้อมที่จะรับข้อมูลแล้ว

FD_WRITE มีการส่งข้อมูลมาทาง socket

FD_CLOSE socket ปิด

FD_ACCEPT มีการรับการเชื่อมต่อ

11. คลาส convert

```
class convert{
public:
void threeFloattoArrayChar(float f1, float f2, float f3, char buffer[]);
void fourFloattoArrayChar(float f1, float f2, float f3, float f4, char buffer[]);
int arrayChartoInt(char buffer[]);
float arrayChartoFloat(char buffer[]);
void stringtoArrayChar(char **buffer, string str);
};
```

รายละเอียดของฟังก์ชันในคลาส

```
void convert::threeFloattoArrayChar(float f1, float f2, float f3, char buffer[]){
sprintf(buffer, "%.4f, %.4f, %.4f, ", f1, f2, f3);
}

void convert::fourFloattoArrayChar(float f1, float f2, float f3, float f4, char buffer[]){
sprintf(buffer, "%.4f, %.4f, %.4f, %.4f, ",f1, f2, f3, f4);
}

int convert::arrayChartoInt(char buffer[]){
int i;
i= atoi (buffer);
return i;
}

float convert::arrayChartoFloat(char buffer[]){
double d;
float f;
d = atof ( buffer );
f=(float)d;
return f;
}

void convert::stringtoArrayChar(char **buffer, string str)
{
*buffer = new char[str.length() + 1];
strcpy(*buffer, str.c_str());
}
```


ตัวแปรและฟังก์ชัน

void threeFloattoArrayChar(float f1, float f2, float f3, char buffer[]); ฟังก์ชันรับค่า float 3 ค่าแล้วแปลงเป็นข้อความใส่ลงในตัวแปร buffer

void fourFloattoArrayChar(float f1, float f2, float f3, float f4, char buffer[]); ฟังก์ชันรับค่าfloat 4 ค่าแล้วแปลงเป็นข้อความใส่ลงในตัวแปร buffer

int arrayChartoInt(char buffer[]); ฟังก์ชันการแปลงข้อความเป็น int

float arrayChartoFloat(char buffer[]); ฟังก์ชันการแปลงข้อความเป็น float

void stringtoArrayChar(char **buffer, string str); ฟังก์ชันตัวแปรประเภทstring เป็น arrayของchar

ภาคผนวก ค

ค่าคงที่ในเฟรมเวิร์ค

ค่าคีย์ ต่างๆ อยู่ในไฟล์ variablefile.cpp

keyA=65
keyB=66
keyC=67
keyD=68
keyE=69
keyF=70
keyG=71
keyH=72
keyI=73
keyJ=74
keyK=75
keyL=76
keyM=77
keyN=78
keyO=79
keyP=80
keyQ=81
keyR=82
keyS=83
keyT=84
keyU=85
keyV=86
keyW=87
keyX=88
keyY=89
keyZ=90
keyUP=38
keyDOWN=40
keyLEFT=37
keyRIGHT=39
keyESCAPE=27
keySPACE=32
keyRETURN=13
keyNULL=4000

ประวัติผู้เขียน

ชื่อ	นายภูมิภากร มาโนช
วัน เดือน ปี เกิด	17 มิถุนายน 2522
ประวัติการศึกษา	สำเร็จการศึกษาปริญญาวิทยาศาสตรบัณฑิต สาขาวิชาวิทยาการคอมพิวเตอร์ มหาวิทยาลัยพายัพ ปีการศึกษา 2545