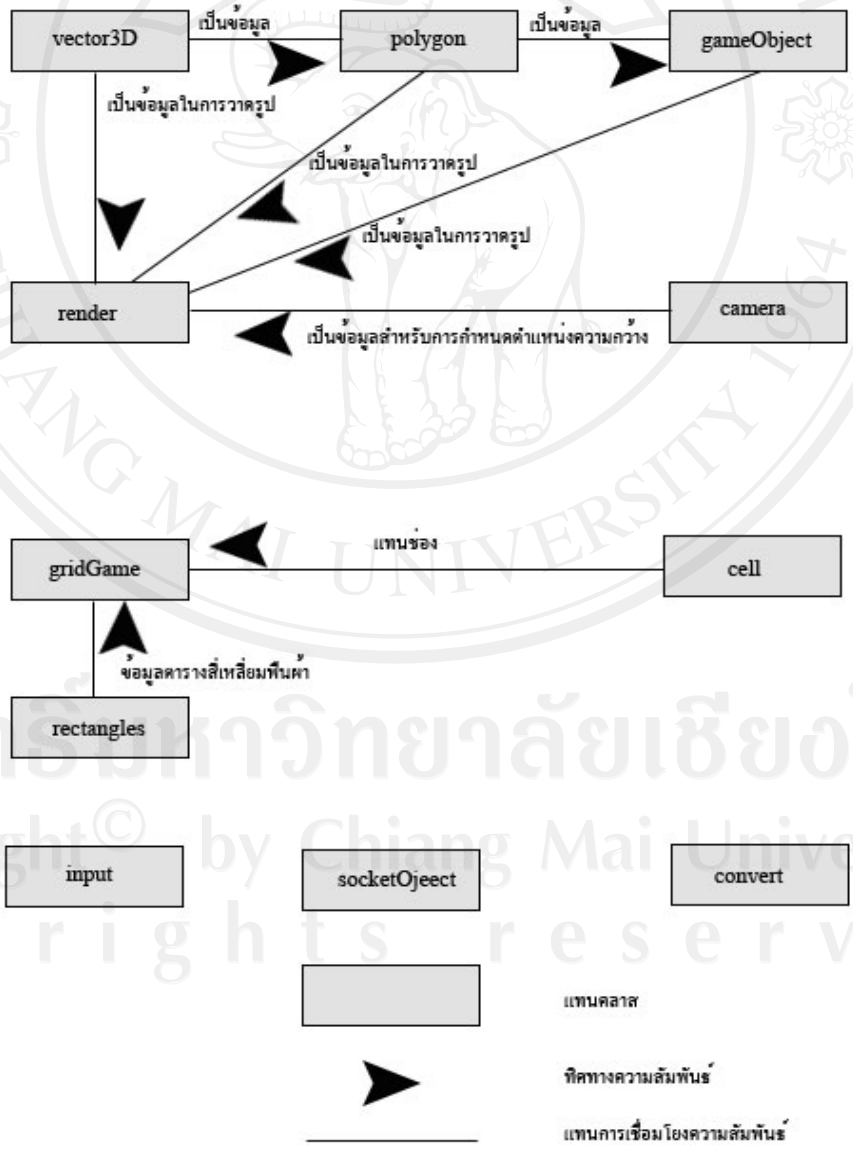
# บทที่ 2
# เฟรมเวิร์คของเกม

บทนี้กล่าวถึงเนื้อหาของเฟรมเวิร์คเกมและโครงสร้างของคลาสต่างๆ ว่ามีอะไรบ้าง

## 2.1 สถาปัตยกรรมของเฟรมเวิร์คเกม




รูปที่ 2.1 สถาปัตยกรรมของเฟรมเวิร์คเกม

จากรูปที่ 2.1

**คลาส vector3D** เป็นคลาสที่ทำหน้าที่เก็บข้อมูล vertex และสามารถเป็นข้อมูลในการวาดรูปของคลาส render

**คลาส polygon** ทำหน้าที่เก็บข้อมูลของแต่ละพื้นผิว (Polygon) ของวัตถุในเกม โดยข้อมูลของคลาส polygon จะเป็นชนิดคลาส vector3D คือจุดเวอร์เท็กซ์ (Vertex) ต่างๆ

**คลาส gameObject** ทำหน้าที่เก็บข้อมูลของวัตถุในเกม โดยข้อมูลเป็นคลาส **polygon**

**คลาส camera** ทำหน้าที่เก็บข้อมูลของกล้องเช่น ที่ตั้ง ตำแหน่งการมอง ใช้แทนตำแหน่งของผู้เล่น ตำแหน่งการมองของผู้เล่นและเก็บข้อมูลของลักษณะการแสดงภาพ เพื่อเป็นข้อมูลให้กับคลาส render ในการ projection

**คลาส render** ทำหน้าที่ในการวาดรูปและการ projection

**คลาส rectangles** ทำหน้าที่เก็บข้อมูลของพื้นที่สี่เหลี่ยมผืนผ้า ข้อมูลคือ จุดเริ่มต้น ความกว้าง ความสูง

**คลาส cell** ทำหน้าที่แทนช่องตารางสำหรับคลาส gridGame

**คลาส gridGame** ทำหน้าที่เหมือนตีกรอบพื้นที่ของเกม เพื่อช่วยลดการวาดรูปและการทดสอบอื่นๆ เช่นทดสอบการชน

**คลาส input** ทำหน้าที่รับค่าการกดคีย์บอร์ดจากผู้เล่น

**คลาส socketObject** ทำหน้าที่เกี่ยวกับการติดต่อสื่อสาร การส่งข้อความ

**คลาส convert** เป็นคลาสสำหรับช่วยแปลงชนิดตัวแปร

## 2.2 การทำงานเรื่องต่างๆ ของเฟรมเวิร์คเกม

การทำงานเรื่องต่างๆ ของเฟรมเวิร์คเกมแบ่งเป็น 7 ส่วน

### 2.2.1 ด้านการรับอินพุตของผู้เล่น (Update Input)

สร้างคลาส input ขึ้นมาโดยคลาสนี้ จะทำการเริ่มต้นการเรียกใช้ ไดเรกอินพุตและจะสร้างฟังก์ชันที่ รีเทิร์นค่า ปุ่มที่กดกลับคืนให้กลับผู้เรียกใช้

**โครงสร้างของคลาส input**

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

const int keyA ถึง const int keyZ

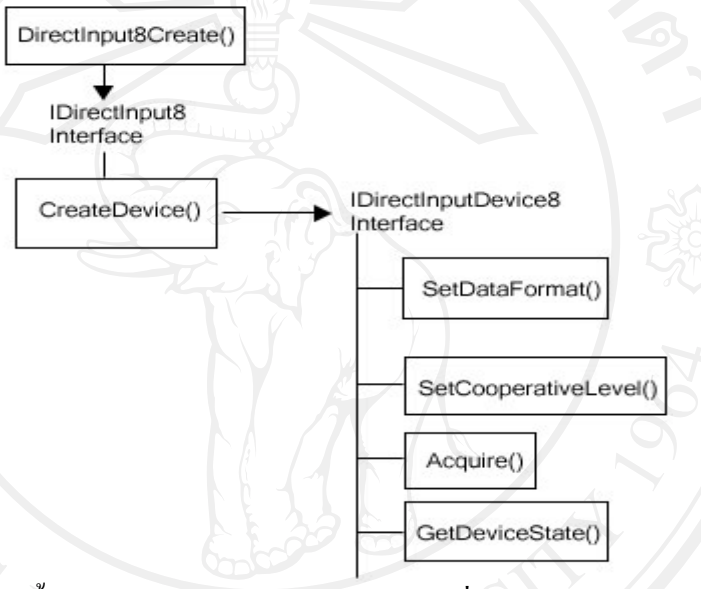
const int keyUP keyDOWN keyLEFT keyRIGHT

และ const ค่า คีย์ อื่นๆ

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

**การสร้างคลาส input จากการใช้งานไดเรกอินพุต (Direct Input)**

มีขั้นตอนการเรียกฟังก์ชันตามรูปที่ 2.2 คือ




รูปที่ 2.2 ขั้นตอนการสร้างตัวแปรไดเรกอินพุตเพื่อใช้งานในคลาส input

1) สร้าง IDirectInput 8 Interface จากฟังก์ชัน DirectInput8Create()

2) สร้าง IDirectInputDecvice 8 Interface จากฟังก์ชัน CreateDevice() ของ IDirectInput 8 Interface ที่ได้จากข้อ 1.

3) เมื่อได้ IDirectInputDecvice8 Interface จากข้อ2. แล้วสามารถเรียกใช้ฟังก์ชันต่างๆของIDirectInputDecvice 8 Interface ได้ เช่น

SetDataFormat() การsetรูปแบบข้อมูลว่าจะมาจาก keyboard mouse หรือ joystick

SetCooperativeLevel() กำหนดรูปแบบของการใช้อุปกรณ์

Acquire() ครอบครอง device

GetDeviceState() รับข้อมูลจาก device

สามข้อที่กล่าวมาเป็นการเรียกใช้ไดเรกอินพุต ทั้งหมดรวมไว้ในคลาส input แล้วผู้ใช้สามารถสร้างตัวแปร input และเรียกใช้ได้

ฟังก์ชันตัวสร้าง (constructor) ของคลาสจะเริ่มต้นติดต่อกับ ไดเรกอินพุต (Direct Input) โดยทำข้อ 1 กับข้อ 2

ฟังก์ชัน getKey() จะรีเทิร์นค่า คีย์ต่างๆมาให้เมื่อเรียกใช้ ให้ผู้พัฒนาเกมนำไปทดสอบว่ากดปุ่มใด

**ตัวอย่างฟังก์ชันที่สำคัญ**

ฟังก์ชัน getKey()

```
int input::getKey(){
keyboard->GetDeviceState( sizeof(buffer) , (LPVOID)&buffer );
if( buffer[DIK_A] & 0x80 )
{
return keyA;
}
หรือรีเทิร์น คีย์อื่นๆแล้วแต่การกด
}
```

รูปแบบเวลาใช้งานจะเป็นการวนลูปเรียกใช้ฟังก์ชัน getKey() เมื่อรีเทิร์นค่ามาเป็นคีย์ใดก็ทำงานตามความต้องการ

### 2.2.2 ด้านการปรับตำแหน่งของกล้อง (Update Camera)

สร้างคลาส camera แทนผู้เล่น

มีข้อมูลหลักๆคือ

angleY มุมที่กระทำกับแกนวาย (y axis)

speed ความเร็วในการเดิน

rotateSpeed ความเร็วในการหมุน

radius ระยะรัศมี

location ตำแหน่งของกล้อง

target ตำแหน่งที่กล้องมองไป

distanceEye ระยะสายตา

ข้อมูลเหล่านี้เป็นข้อมูลของกล้อง การปรับเปลี่ยนตำแหน่งของกล้องก็คือการปรับเปลี่ยนค่าของตัวแปรเหล่านี้ มีฟังก์ชันสำคัญคือ ฟังก์ชันในการปรับตำแหน่งเมื่อเดินหน้า ถอยหลัง หันซ้าย หันขวา

**โครงสร้างของคลาส 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;
floar 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);
void addHitObjects(gameObject object);
void clearHitObjects();
};
```

**การสร้างคลาส camera**

การสร้างคลาส camera ได้นำสูตรการปรับตำแหน่งของกล้องและการปรับตำแหน่งการมองของกล้องในบทที่1 มาใช้ในฟังก์ชันที่เกี่ยวกับการเคลื่อนที่ของกล้องคือ cameraForward() cameraBackward() cameraTurnLeft() และ cameraTurnRight()

**ตัวอย่างฟังก์ชันการเปลี่ยนตำแหน่ง**

```
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)));
}
```

ในตัวอย่างเป็นฟังก์ชันการเดินหน้า ใช้สูตรการปรับตำแหน่งตามที่กล่าวไว้ในบทที่1 เมื่อเกิดการเปลี่ยนตำแหน่งของกล้องจะปรับข้อมูลและคำนวณไปตามสูตร

รูปแบบในการใช้งานคลาส camera นี้ คือ การเรียกใช้ฟังก์ชัน cameraForward() cameraBackward() cameraTurnLeft() และ cameraTurnRight() เป็นหลัก เมื่อมีการกดคีย์บอร์ดแล้วรับค่าคีย์เข้ามาก็จะเรียกฟังก์ชันเหล่านี้ในการปรับเปลี่ยนตำแหน่งของกล้อง คลาส camera นี้ไม่ได้วาดรูปออกหน้าจอ เป็นเพียงคลาสที่เก็บข้อมูลเกี่ยวกับกล้องไว้เพื่อส่งข้อมูลให้กับการวาดรูปเท่านั้น การเปลี่ยนตำแหน่งของกล้องจึงเป็นแค่การเปลี่ยนตัวเลขที่เก็บไว้เท่านั้น เช่น การเดินหน้าฟังก์ชันจะทำการเปลี่ยนตำแหน่งของกล้อง การหันซ้ายก็ปรับองศาที่หมุนรอบแกน y

### 2.2.3 ด้านการตรวจจับการชน (Collision Detection)

สร้างฟังก์ชันอยู่ใน คลาส camera ตัวแปรหลักๆที่อยู่ในคลาส camera แล้วนำมาใช้ในฟังก์ชันคือ

radius ระยะรัศมี

location ตำแหน่งของกล้อง

```
vector<gameObject> vhitObjects;
gameObject *hitObjects;
int numHitObjects;
```

ฟังก์ชันที่เกี่ยวกับการตรวจจับการชน คือ

```
bool hitObject(gameObject object);
void hitObject(cell c);
```

```
bool hitObject(gameObject object);
void hitObject(cell c);
void addHitObjects(gameObject object);
void clearHitObjects();
```

**การทำงานของฟังก์ชัน**

ฟังก์ชัน hitObject(gameObject object); จะตรวจจับการชนเทียบกับ gameObject ที่เป็นพารามิเตอร์

ฟังก์ชัน hitObject(cell c); จะตรวจจับการชนเทียบกับทุก gameObject ที่อยู่ใน cell ที่เป็นพารามิเตอร์แล้วเก็บส่ง gameObject ที่ชนใส่ไว้ในตัวแปร vhitObjects กับ hitObjects โดย vhitObjects เป็นตัวแปรชนิด vector เหมือนกับอาเรย์ที่มีการเพิ่มขนาดได้ส่วน hitObjects เป็นตัวแปร pointer ชนิด gameObject ข้อมูลในสองตัวแปรนี้เหมือนกันมีสองชนิดเพื่อสะดวกในการเรียกใช้ จำนวนตัวแปร gameObject มีกี่จำนวนเก็บไว้ใน numHitObjects

การใส่ข้อมูลลงใน vhitObjects กับ hitObjects ใช้การส่ง gameObject ที่ชนไปเป็นพารามิเตอร์ในฟังก์ชัน addHitObjects(gameObject object); ฟังก์ชันนี้จะถูกเรียกจากฟังก์ชัน hitObject(cell c);

ฟังก์ชัน clearHitObjects(); ใช้สำหรับลบข้อมูลในตัวแปร vhitObjects กับ hitObjects

รูปแบบการใช้งานคือเมื่อต้องการทดสอบการชนสามารถเรียกใช้ฟังก์ชัน hitObject() โดยเลือกทดสอบเฉพาะ gameObject เดียวหรือทดสอบกับ gameObject ทั้งหมดที่มีอยู่ใน cell เป้าหมายก็ได้แล้วแต่พารามิเตอร์

ฟังก์ชัน hitObjectCylinder() เหมือนกับฟังก์ชัน hitObject() แต่เป็นการทดสอบแบบทรงกระบอกแต่ฟังก์ชัน hitObject() เป็นการทดสอบแบบวงกลม

**ตัวอย่างฟังก์ชัน**

```
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; // ชน
}
return false; // ไม่ชน
}

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;// ชน
}
return false; // ไม่ชน
}
```

ตัวอย่างเป็นฟังก์ชัน hitObject() และ hitObjectCylinder() ที่อยู่ในคลาส camera ฟังก์ชันจะรีเทิร์นค่าบูลีน ถ้าคำนวณตามสูตรแล้วเกิดการชนจะรีเทิร์น true ถ้าไม่รีเทิร์น false

จะเห็นว่าระยะทางที่ใกล้ที่สุดของ hitObject() เป็น 3 แกนคือ x, y, z แต่ hitObjectCylinder() ใช้ 2 แกนคือ x, z ทำให้ hitObject() เหมือนเป็นทรงกลมแต่ hitObjectCylinder() เป็นทรงกระบอก

**2.2.4 ด้านปัญญาประดิษฐ์ของเกม (Artificial Intelligence)**

สร้างเป็นฟังก์ชันอยู่ในคลาส gameObject
ข้อมูลหลักๆคือ
vectorNormal เวกเตอร์ที่ตั้งฉากกับ พื้นผิว (Polygon) ตัวแทน
polygonNormal พื้นผิว (Polygon) ตัวแทน
distanceEye ระยะทางการมองเห็น
centerPivotPoint จุดศูนย์กลางของวัตถุ
radius รัศมี
ข้อมูลเหล่านี้ใช้ในฟังก์ชันที่เกี่ยวกับเรื่องนี้คือ canSee() canHear() getSide()

**โครงสร้างของคลาส gameObject**

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

gameObject();
gameObject(polygon p);
gameObject(polygon p[], int numPolygons);
void setName(string name);
string getName();
void setType(int type);
void setDistanceEye(float distanceEye);
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);
};
```



**การสร้างฟังก์ชัน**

การสร้างฟังก์ชัน canSee() การจะรู้ว่าวัตถุมองเห็นสิ่งที่ต้องการทดสอบหรือไม่ ต้องรู้ว่าสิ่งที่ต้องการทดสอบนั้นอยู่ด้านหน้าของวัตถุหรือไม่แล้วอยู่ในระยะสายตาหรือไม่ด้วย การทดสอบทั้งหมดนี้ทำตามหลักการของเรื่องปัญญาประดิษฐ์ของเกมที่กล่าวไว้ในบทที่1

การสร้างฟังก์ชัน canHear() การรู้ว่าวัตถุได้ยินใช้ระยะทางเพื่อวัดว่าไกลหรือใกล้เกินกว่าการได้ยิน ทั้งหมดนี้ทำตามหลักการของเรื่องปัญญาประดิษฐ์ของเกมที่กล่าวไว้ในบทที่1

**ตัวอย่างฟังก์ชัน**

```
bool gameObject::canSee(vector3D location){
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) && (distance<distanceEye) )
return true;
return false;
}

bool gameObject::canHear(vector3D location){
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<radius) return true;
return false;
}
```

ฟังก์ชัน canSee เป็นฟังก์ชันทดสอบการมองเห็น ดูว่าอยู่ด้านหน้าและระยะทางเกินไม่เกินระยะสายตา ถือว่ามองเห็น

ฟังก์ชัน canHear เป็นฟังก์ชันทดสอบการได้ยิน อยู่ในรัศมี ถือว่าได้ยิน

**2.2.5 ด้านการจัดการกับวัตถุในเกม (Game Object Management)**

สร้างคลาส cell คลาส gridGame เพื่อเป็นคลาสสำหรับเก็บข้อมูลของวัตถุในเกม

คลาส cell

หน้าที่หลักๆคือ มีอาเรย์ของ gameObject เก็บ gameObject ที่อยู่ใน cell นี้ การวัดว่าวัตถุอยู่ที่ cell ไหนวัดจากจุดศูนย์กลางของวัตถุว่าตกอยู่ cell ไหน

คลาส gridGame

หน้าที่หลักๆคือ มีอาเรย์ของ cell แต่ละ cell ก็มีข้อมูลของ gameObject ที่อยู่ใน cell นั้นๆ

**โครงสร้างของคลาส cell**

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

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

};
```

**โครงสร้างของคลาส gridGame**

```
class gridGame{
public:
int GRID_SIZE_BITS;
int GRID_SIZE;

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();
```

};

**การสร้างคลาส gridGame**

การสร้างคลาสนี้แบ่งเป็นหลายฟังก์ชัน ตัวสร้างมี 4 ฟังก์ชันเพื่อให้สะดวกในการส่งพารามิเตอร์ให้กับฟังก์ชัน มีฟังก์ชัน getCell() 4 ฟังก์ชันเพื่อรับ cell ที่ต้องการโดยคำนวณจากพารามิเตอร์ที่รับเข้ามา การรับค่า cell ต้องคำนวณตำแหน่งของสิ่งที่เป็นพารามิเตอร์ว่าอยู่ใน cell ไหน โดยใช้ฟังก์ชัน convertMapXtoGridX() และ convertMapYtoGridY() เมื่อคำนวณได้ว่าสิ่งที่ส่งมาเป็นพารามิเตอร์นั้นอยู่ใน cell ไหนแล้วก็สามารถจำกัดการกระทำเช่นการตรวจจับการชนให้อยู่เฉพาะใน cell นั้นๆ การได้ cell เป้าหมายนี้ถือเป็นการจัดการบริหารวัตถุในเกม การคำนวณต่างๆ ทำตามหลักการที่กล่าวไว้ในบทที่1

**ตัวอย่างฟังก์ชัน**

```
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]); }
}
}
```

ในตัวอย่างเป็นฟังก์ชันตัวสร้าง (constructor) ฟังก์ชันรับค่าวัตถุเข้ามาแล้วเรียก ฟังก์ชัน calcBound() เพื่อทำการคำนวณตำแหน่งเพื่อสร้างกรอบสี่เหลี่ยมเหมือนแผนที่แล้วทำการแบ่งเป็นช่องๆ

**2.2.6 ด้านการวาดรูป (Render)**

สร้างเป็นคลาส render มีข้อมูลสำคัญเกี่ยวกับ ตัวแปรไดเรกเอกซ์ (DirectX) และดีไวซ์คอนเท็กซ์

โครงสร้างคลาส render

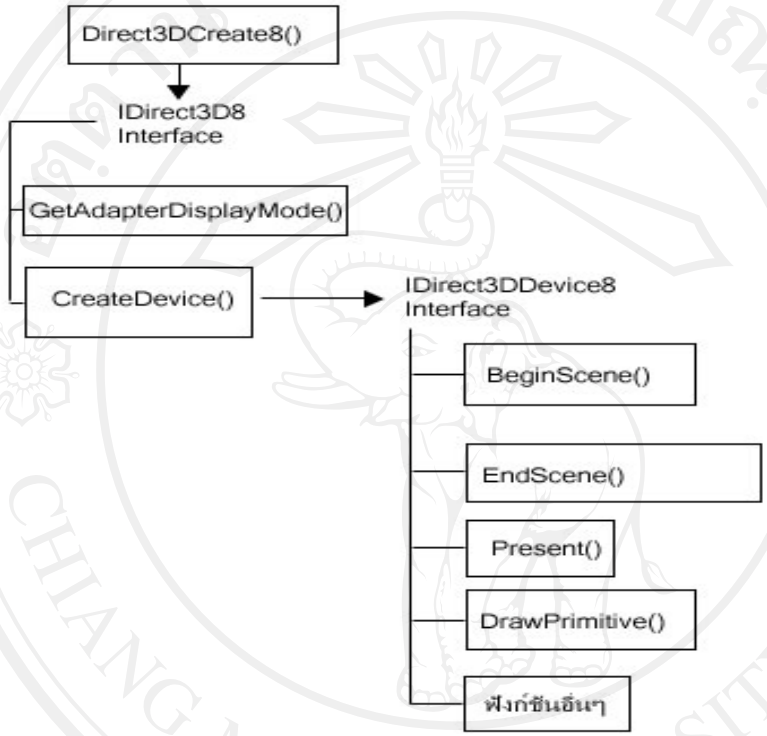
```
class render{
public:
HWND hwnd;
LPDIRECT3D8 pD3D;
LPDIRECT3DDEVICE8 pd3dDevice;
D3DXMATRIX matCameraView;    // Camera Data
D3DXMATRIX matProj;          // Projection Data
D3DXVECTOR3 d3dxVecLocation;
D3DXVECTOR3 d3dxVecTarget;
LPDIRECT3DVERTEXBUFFER8 lpVertexBuffer;// = NULL;
D3DPRESENT_PARAMETERS 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);
};
```

**การสร้างคลาสและฟังก์ชัน**

มีขั้นตอนดังรูปที่ 2.3




รูปที่ 2.3 ขั้นตอนการสร้างตัวแปรไดเรกสามดีเพื่อใช้ในคลาส render

1) ฟังก์ชันDirect3DCreate8 สร้างIDirect3D8 ออบเจ็กต์

2)ใช้ฟังก์ชัน GetAdapterDisplayMode ของออบเจ็กต์IDirect3D8 เพื่อรับค่าเกี่ยวกับหน้าจอ

3)ใช้ฟังก์ชัน CreateDevice ของออบเจ็กต์IDirect3D8 เพื่อแทน diosplay adapter ได้ออบเจ็กต์IDirect3DDevice8

4)ใช้ออบเจ็กต์ IDirect3DDevice8 ซึ่งได้จากข้อ 3) เรียกฟังก์ชันของออบเจ็กต์ IDirect3DDevice8 ที่เกี่ยวกับการวาดรูปในฟังก์ชันต่างๆของคลาส render

ส่วนการสร้างฟังก์ชันเกี่ยวกับการวาดข้อความใช้ฟังก์ชัน

SetTextColor() ตั้งค่าสี

SetBkColor() ตั้งค่าสีของพื้นหลัง

TextOut() ฟังก์ชันการวาดข้อความของระบบปฏิบัติการวินโดส์

**ตัวอย่างฟังก์ชัน**

ฟังก์ชัน draw ใช้ในการวาดรูป

```
void render::draw(camera c, polygon p){
        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);
        D3DXMatrixIdentity(&matProj);
        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 );

        // Clear the backbuffer and the zbuffer
        pd3dDevice->Clear( 0, NULL,
D3DCLEAR_TARGET|D3DCLEAR_ZBUFFER, D3DCOLOR_XRGB(0, 0, 255),
1.0f, 0 );
        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
        pd3dDevice->DrawPrimitive(D3DPT_TRIANGLELIST, 0, 1);
        // Stop Rendering
        pd3dDevice->EndScene();
        // Output the scene
        pd3dDevice->Present( NULL, NULL, NULL, NULL );
}
```

ฟังก์ชัน drawText ใช้ในการวาดตัวอักษร

```
void render::drawText(HWND hwnd, int x, int y, char str[30])
{
        hdc=GetDC(hwnd);
        SetTextColor(hdc, RGB(255, 0, 0));
        SetBkColor(hdc, RGB(0, 255, 0));
        TextOut(hdc, x, y, str, strlen(str));
        ReleaseDC(hwnd, hdc);
}
```

**2.2.7 ด้านการติดต่อสื่อสาร (Network Communication)**

สร้างเป็นคลาส socketObject มีฟังก์ชันสำคัญเกี่ยวกับการติดต่อสื่อสาร

โครงสร้างคลาส 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);
void Disconnect();
bool Connect(char* chpServerAddress, int iPort);
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();
};
```

ในหัวข้อนี้จะอธิบายการสร้างคลาสแยกเป็นฟังก์ชันดังนี้

**1) การสร้างฟังก์ชัน Bind ของคลาส socketObject**

ใช้การเรียกใช้ฟังก์ชัน bind ของ socket API มี prototype คือ

**int bind( SOCKET** s, **const struct sockaddr FAR*** name, **int** namelen **);**

มีตัวแปรโครงสร้างอยู่ 2 ชนิดที่เกี่ยวข้องคือ sockaddr_in กับ sockaddr ตัวแปรโครงสร้างสองชนิดประกาศไว้ใน winsock2.h

โครงสร้างของทั้งสองตัวแปรโครงสร้างมีดังนี้

```
struct sockaddr {
    ushort  sa_family;
    char    sa_data[14];
};

struct sockaddr_in{
  short sin_family;
  unsigned short sin_port;
  struct in_addr sin_addr;
  char sin_zero[8];
};
```

ตัวแปรตัวที่3 ของ sockaddr_in มีชนิดเป็น in_addr

```
struct in_addr {
```

```
    union {
        struct { unsigned char s_b1, s_b2, s_b3, s_b4; } S_un_b;
        struct { unsigned short s_w1, s_w2; } S_un_w;
        unsigned long S_addr;
    } S_un;
};
```

การใส่ค่าเข้าไปในตัวแปรเพื่ออ้างอิง address ของ server จะใช้คำสั่ง

```
saServerAddress.sin_family = AF_INET;
saServerAddress.sin_addr.s_addr = htonl(INADDR_ANY);
saServerAddress.sin_port = htons(iPort);
```

ในฟังก์ชัน Bind ของคลาส socketObject เรียกฟังก์ชัน bind ของ socket API โดยใช้คำสั่ง

```
bind(skSocket, (sockaddr*) &saServerAddress, sizeof(sockaddr));
```

จะเห็นว่าในคำสั่งพารามิเตอร์ตัวที่2 ใช้วิธีเปลี่ยนชนิดจาก sockaddr_in มาเป็น sockaddr ตัวแปรสองชนิดนี้มีขนาดเท่ากัน

ฟังก์ชัน Bind ของคลาส socketObject จะรีเทิร์น true ถ้าไม่มี error ใดๆแต่ถ้ามี error จะรีเทิร์น false

**2) การสร้างฟังก์ชัน Listen ของคลาส socketObject**

ใช้การเรียกใช้ฟังก์ชัน listen ของ socket API มี prototype คือ

**int listen( SOCKET** s, **int** backlog **);**

พารามิเตอร์ตัวแรกคือตัวแปร socket ตัวที่สองคือ จำนวนในการเชื่อมต่อ ในฟังก์ชัน Listen ของคลาส socketObject เรียกใช้ฟังก์ชัน listen โดยใช้คำสั่ง

```
listen( skSocket, 32 );
```

ฟังก์ชัน Listen ของคลาส socketObject จะรีเทิร์น 0 ถ้าไม่มี error ใดๆ แต่ถ้ามี error จะรีเทิร์น SOCKET_ERROR

**3) การสร้างฟังก์ชัน Accept ของคลาส socketObject**

ใช้การเรียกใช้ฟังก์ชัน accept ของ socket API มี prototype คือ

**SOCKET accept( SOCKET** s, **struct sockaddr FAR*** addr, **int FAR*** addrlen **);**

พารามิเตอร์ที่1 คือ ตัวแปร socket ที่ทำการ accept โดยมากเป็น socket ที่เป็น server

พารามิเตอร์ที่2 คือ ตัวแปร pointer ชนิด sockaddr ที่กล่าวถึงในหัวข้อฟังก์ชัน bind
ฟังก์ชัน accept นี้จะทำการคืนค่าให้กับตัวแปร sockaddr ที่ส่งเข้ามาให้ฟังก์ชัน
พารามิเตอร์ที่3 คือ ขนาดของ address ก็คือขนาดของตัวแปร sockaddr
ในคลาสเรียกใช้ฟังก์ชัน accept นี้โดยใช้คำสั่ง

```
int iClientSize = sizeof(sockaddr_in);
skAcceptSocket.skSocket = accept(
        skSocket,
        (struct sockaddr*)&skAcceptSocket.
        sockaddr_inIP, &iClientSize
        );
```

ฟังก์ชัน Accept ของคลาส socketObject ถ้า accept สำเร็จรีเทิร์น true ถ้าไม่สำเร็จรีเทิร์น false

**4) การสร้างฟังก์ชัน Connect ของคลาส socketObject**

ใช้การเรียกใช้ฟังก์ชัน connect ของ socket API มี prototype คือ
**int connect( SOCKET** s, **const struct sockaddr FAR*** name, **int** namelen **);**

พารามิเตอร์ที่1 คือ ตัวแปร socket ที่ทำการ connect
พารามิเตอร์ที่2 คือ ตัวแปร pointer ชนิด sockaddr ที่กล่าวถึงในหัวข้อฟังก์ชัน bind ตัวแปรนี้จะเก็บ address ของเครื่องที่จะ connect ไป
พารามิเตอร์ที่3 คือ ขนาดของตัวแปร sockaddr
ในคลาสเรียกใช้ฟังก์ชัน connect คำสั่ง

```
connect(skSocket, (struct sockaddr*)&serv_addr, sizeof(sockaddr));
```

ฟังก์ชัน Connect ของคลาส socketObject ถ้า connect ได้รีเทิร์น true ถ้า connect ไม่ได้รีเทิร์น false

**5) การสร้างฟังก์ชัน Send ของคลาส socketObject**

ใช้การเรียกใช้ฟังก์ชัน send ของ socket API มี prototype คือ
**int send( SOCKET** s, **const char FAR ***buf, **int** len, **int** flags **);**
พารามิเตอร์ที่1 คือ ตัวแปร socket ที่ทำการส่งข้อมูล
พารามิเตอร์ที่2 คือ ข้อความที่ส่ง
พารามิเตอร์ที่3 คือ ขนาดของข้อความที่ส่ง

พารามิเตอร์ที่4 คือ flags ที่ใช้ในการส่ง ในที่นี้ให้ใส่ 0
ในคลาสเรียกใช้ฟังก์ชัน send คำสั่ง

**send(skSocket, chpBuffer, iBufLen, iFlags);**

ฟังก์ชัน Send ของคลาส socketObject ถ้าไม่มี error ใดๆ ฟังก์ชันจะรีเทิร์นความยาวของข้อความที่ส่งสำเร็จ ถ้ามี error ฟังก์ชันจะรีเทิร์น SOCKET_ERROR

**6) การสร้างฟังก์ชัน Recv ของคลาส socketObject**

ใช้การเรียกใช้ฟังก์ชัน recv ของ socket API มี prototype คือ
**int recv( SOCKET** s, **char FAR ***buf, **int** len, **int** flags **);**
พารามิเตอร์ที่1 คือ ตัวแปร socket ที่ทำการรับข้อมูล
พารามิเตอร์ที่2 คือ buffer ที่ใช้รับข้อความ
พารามิเตอร์ที่3 คือ ขนาดของ buffer ที่ใช้รับข้อความ
พารามิเตอร์ที่4 คือ flags ที่ใช้ในการรับข้อความ ในที่นี้ให้ใส่ 0
ในคลาสเรียกใช้ฟังก์ชัน recv คำสั่ง

**recv(skSocket, chpBuffer, iBufLen, iFlags);**

ฟังก์ชัน Recv ของคลาส socketObject ถ้าไม่มี error ใดๆ ฟังก์ชันจะรีเทิร์นความยาวของข้อความที่รับสำเร็จ ถ้ามี error ฟังก์ชันจะรีเทิร์น SOCKET_ERROR

**7) การใช้ฟังก์ชัน WSAAsyncSelect ในคลาส socketObject**

ฟังก์ชัน WSAAsyncSelect() ของ winsock เป็นฟังก์ชันที่ใช้เพื่อขอให้WS2_32.DLLส่งแมสเสจมาที่วินโดส์ที่เป็นโปรแกรมหลักของ socket นั้นๆ มี prototype คือ

```
int WSAAsyncSelect (
  SOCKET s,
  HWND hWnd,
  unsigned int wMsg,
  long lEvent
);
```

พารามิเตอร์ที่1 คือ ตัวแปร socket ที่ต้องการรับ event
พารามิเตอร์ที่2 คือ ตัวแปร handle ของวินโดส์
พารามิเตอร์ที่3 คือ ตัวเลขแมสเสจที่ต้องการให้ส่ง
พารามิเตอร์ที่4 คือ event ใดบ้างที่ต้องการให้ส่งมา

ในคลาส socketObject มีการเรียกใช้ฟังก์ชัน WSAAsyncSelect() ในฟังก์ชัน Connect() กับ Listen() ในคลาสเรียกใช้โดยใช้บรรทัดคำสั่ง

**WSAAsyncSelect(**
skSocket,
hwnd,
3000,
FD_READ | FD_WRITE | FD_CLOSE | FD_ACCEPT
**);**

หมายความว่าเมื่อมี event ตามพารามิเตอร์ตัวที่4 WS2_32.DLLจะส่งแมสเสจมาที่วินโดส์ที่เป็นโปรแกรมหลักของ socket ที่เป็นฟารามิเตอร์ตัวที่1

ความหมายของแต่ละ event ในพารามิเตอร์ตัวที่4

FD_READ socket ในพารามิเตอร์ที่1 พร้อมรับข้อมูล

FD_WRITE socket ในพารามิเตอร์ที่1 พร้อมส่งข้อมูล

FD_CLOSE socket ในพารามิเตอร์ที่1 ปิด

FD_ACCEPT socket ในพารามิเตอร์ที่1 accept การ connect

**ตัวอย่างฟังก์ชัน**

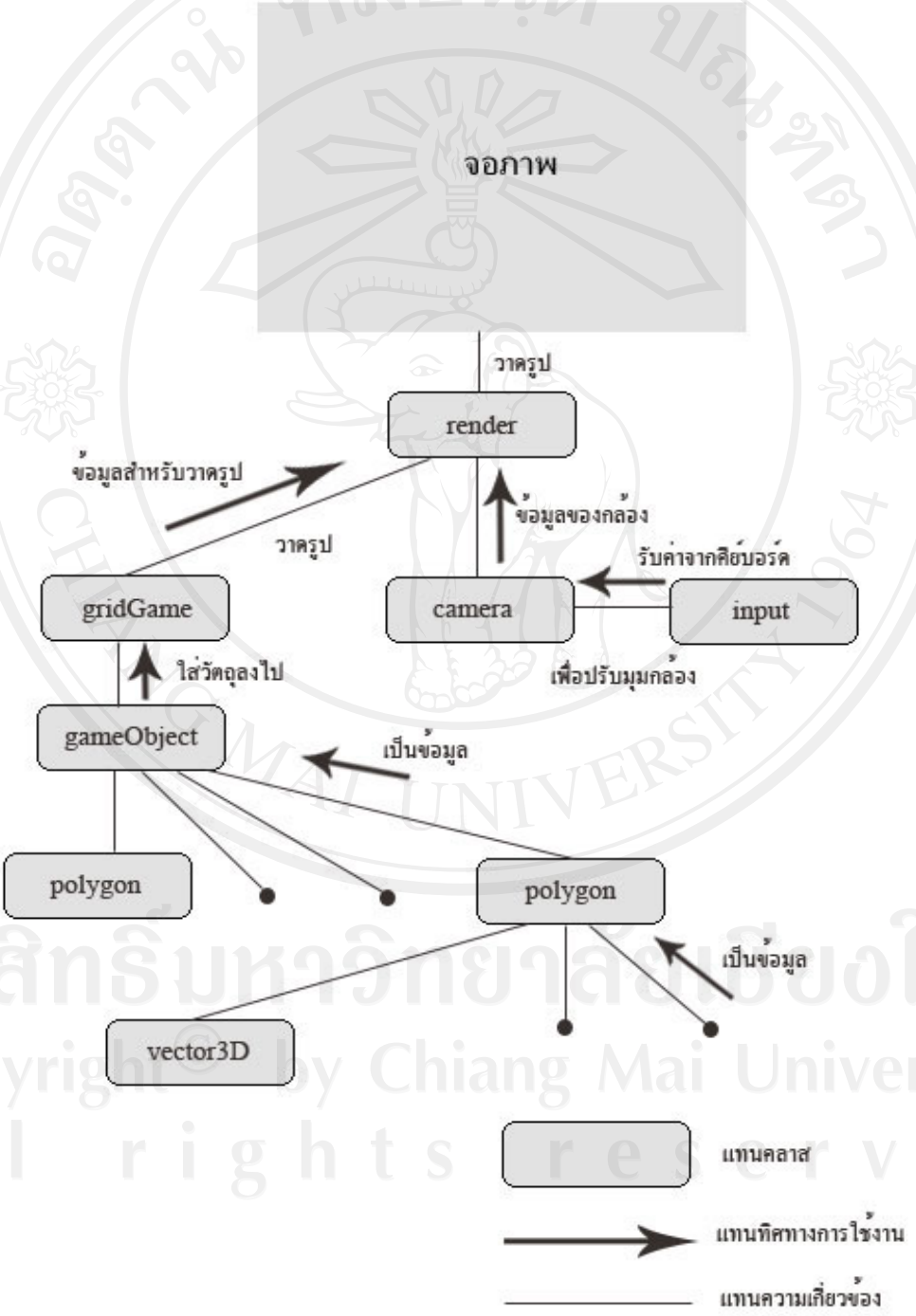
```
int socketObject::Bind(int iPort)
{
	skSocket = socket(AF_INET, SOCK_STREAM, 0);

	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;
}
```

ในตัวอย่างเป็นฟังก์ชัน Bind ของคลาส socketObject ฟังก์ชันนี้จะเรียกใช้ฟังก์ชัน bind ของ socket API ทำหน้าที่ในการจัดตั้งportในการเชื่อมต่อ เป็นฟังก์ชันในส่วนของ server

## 2.3 แนวทางการใช้งานเฟรมเวิร์คเกม




รูปที่ 2.4 แนวทางการใช้งาน

จากรูปที่ 2.4 แนวทางการทำงานคือ

**2.3.1 วัตถุ**

เริ่มต้นต้องมีตัวเลข vertex เพื่อใส่เข้าไปในคลาส vector3D เพื่อแทนจุดต่างๆ ใน 3 มิติ นำ vertex เหล่านี้มาเป็นข้อมูลให้กับคลาส polygon ซึ่งเหมือนกับเป็นพื้นผิวและนำพื้นผิว polygon มาประกอบกันโดยใส่เป็นข้อมูลให้กับคลาส gameObject ได้เป็นวัตถุในเกม นำวัตถุในเกมใส่ลงไปในคลาส gridGame เพื่อสามารถที่จะเป็นการตีกรอบวัตถุว่าอยู่บริเวณใด

**2.3.2 กล้อง**

กำหนดตัวแปรคลาส camera เพื่อเป็นข้อมูลของกล้องและใช้แทนตำแหน่งของผู้เล่น ตำแหน่งการมองของกล้องแทนตำแหน่งการมองของผู้เล่น

**2.3.3 การรับอินพุต**

กำหนดตัวแปรคลาส input เพื่อรับข้อมูลการกดคีย์บอร์ด ทุกครั้งที่มีการกดคีย์บอร์ดจากผู้เล่น เรียกใช้ฟังก์ชันการปรับเปลี่ยนตำแหน่งของกล้องจากคลาส camera การทดสอบการชนเรียกใช้จากฟังก์ชันของคลาส camera ทดสอบและปรับเปลี่ยนสิ่งแวดล้อมเสร็จต้องวาดรูปออกหน้าจอ

**2.3.4 การวาดรูป**

จากรูปที่ 2.4 สี่เหลี่ยมสีเทาด้านบนสุดคือจอภาพที่แสดงรูปที่วาด การวาดรูปใช้ฟังก์ชันของคลาส render โดยเอาข้อมูลของวัตถุในเกมมาจากคลาส vector3D, polygon, gameObject, gridGame ถ้าเป็นคลาส vector3D การวาดรูปจะเป็นการวาดจุดใน 3 มิติ คลาส polygon จะเป็นรูปพื้นผิว คลาส gameObject จะเป็นการวาดรูปวัตถุ คลาส gridGame จะเป็นการวาดรูปวัตถุในแต่ละช่องที่ส่งมาให้วาด และใช้ข้อมูลเกี่ยวกับกล้องจากคลาส camera