1. [Introduction](http://archive.gamedev.net/reference/articles/article747.asp#INTRO)
2. [What the Hell are Isometric and Hexagonal Maps for?](http://archive.gamedev.net/reference/articles/article747.asp#WHATFOR)
3. [Forcing Iso/Hex Maps onto a Rectangular Grid](http://archive.gamedev.net/reference/articles/article747.asp#RECTGRID)
4. [Plotting Iso/Hex Tiles on the Map](http://archive.gamedev.net/reference/articles/article747.asp#PLOTTING)
5. [Moving Around in Iso/Hex Maps](http://archive.gamedev.net/reference/articles/article747.asp#MOVING)
6. [Facing and Turning](http://archive.gamedev.net/reference/articles/article747.asp#FACING)
7. [Mouse Matters](http://archive.gamedev.net/reference/articles/article747.asp#MOUSE)

# Introduction

WHOA! What do you know, I'm finally doing a tutorial on an actual Programming Topic. I think the temperature in Hell must have dropped below 32.

The source code examples in this tutorial will be both in Pascal, and C (wherever I can translate it). Since the techniques are cross platform, I won't be showing code on how to Blit the images to the screen.

All of the tiles I use are 40x40, and all of the calculations are based on it. Depending on the size of bitmap you use, you may have to scale up or down.

# What the Hell are Isometric and Hexagonal Maps for?

Isometric Maps are maps that use rhombuses instead of squares or rectangles. (A rhombus is a four sided figure, with all sides the same length, but not necessarily 90 degrees at the corners. Yes, a Square is technically a rhombus). Isometric maps give sort of an illusion of being 3d, but without actually being so. Sid Meier's Civilization II uses Isometric Maps.

Here is an Isometric Tile: http://archive.gamedev.net/reference/articles/taniso/GreenIso.gif(This tile is actually 40x40, but the rhombus only takes up the bottom 40x21)

Hexagonal Maps are maps that use Hexagons (6 sided figures) instead of squares or rectangles. Hexagonal maps are used mostly for overhead view strategy games (The use of these dates back to Avalon Hill, and other strategy game companies).

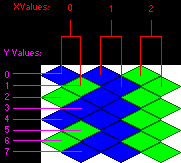
Here is a Hexagonal Tile: http://archive.gamedev.net/reference/articles/taniso/GreenHex.gif

# Forcing Isometric and Hexagonal Maps onto a Rectangular Grid

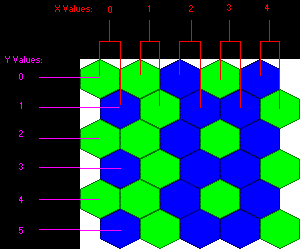
Okay, you can make Chessboard-like maps all day, you just have to use a 2d array. Spiffy. But Isometric and Hexagonal Maps don't work that way. Every other line is offset.

We can still put Iso and Hex maps into a 2d Array, but the WAY in which we map them is different.

Here's an IsoMap:



Here's a HexMap:



As demonstrated in the above pictures, for odd Y values, you shift the line over by half of a tile (20 pixels, in my case).

(The White Spaces are border tiles not on the map. Usually, you would fill these with black.)

# Plotting the Iso/Hex Tiles on the Map

Since both Iso and Hex tiles are contained in overlapping rectangles, you MUST USE BITMASKS!

My Iso Tiles, and the Iso BitMask: http://archive.gamedev.net/reference/articles/taniso/GreenIso.gifhttp://archive.gamedev.net/reference/articles/taniso/BlueIso.gifhttp://archive.gamedev.net/reference/articles/taniso/MaskIso.gif

My Hex Tiles, and the Hex BitMask: http://archive.gamedev.net/reference/articles/taniso/GreenHex.gifhttp://archive.gamedev.net/reference/articles/taniso/BlueHex.gifhttp://archive.gamedev.net/reference/articles/taniso/MaskHex.gif

{The Brief Review of BitMasking: You blit the bitmask using AND, then Blit the Tile using OR.}

**Pixel Coordinates of Iso/Hex Tiles**

When calculating X,Y coordinates for Rectangular tiles, you use the following calculations:

PlotX=MapX\*Width

PlotY=MapY\*Height

For Iso/Hex maps, it's a little trickier, since the bounding rectangles overlap.

**Iso Maps:**

{(MapY AND 1) tells us if MapY is odd or even, and shifts the tile to the right if it is odd}

PlotX=MapX\*Width+(MapY AND 1)\*(Width/2)

PlotY=MapY\*HeightOverLapping-YOffset

Important: Width should always be an even number, or you wind up with a black zigzag line between rows of tiles

{This assumes you have shaped your rhombus like mine, with one pixel}

{at the left and right corners, and two at the top and bottom.}

HeightOverLapping=(Height of Rhombus)/2+1

{to make the first row flush with the top of the map}

Yoffset=Height-(Height of Rhombus)

**HexMaps:**

PlotX=MapX\*Width+(MapY AND 1)\*(Width/2)

PlotY=MapY\*HeightOverLapping

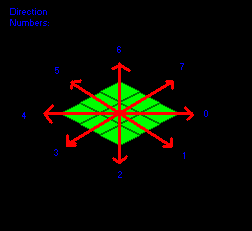
HeightOverLapping=(Height of Hexagon)\*0.75 {Assuming your hexagon looks like mine}

# Moving Around in Iso/Hex Maps

In Rectangular maps, movement from square to square is easy. Just add/subtract 1 to X and/or Y, and you have made the move.

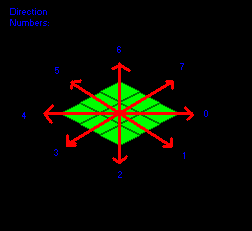
Iso and Hex maps make THAT more difficult, as well. Due to the fact that every other line is offset, there are different calculations, depending if whatever is moving is on an Even Row, or an Odd Row.

Isometric Directions:



**Isometric Directions**

:



For coding purposes, we will give names to these directions:

{Pascal}

Const

IsoEast=0;

IsoSouthEast=1;

IsoSouth=2;

IsoSouthWest=3;

IsoWest=4;

IsoNorthWest=5;

IsoNorth=6;

IsoNorthEast=7;

{C}

#define ISOEAST 0

#define ISOSOUTHEAST 1

#define ISOSOUTH 2

#define ISOSOUTHWEST 3

#define ISOWEST 4

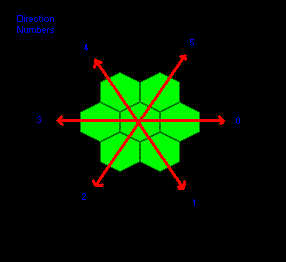
#define ISONORTHWEST 5

#define ISONORTH 6

#define ISONORTHEAST 7

**Hexagonal Directions**

:



The names for these directions:

{Pascal}

Const

HexEast=0;

HexSouthEast=1;

HexSouthWest=2;

HexWest=3;

HexNorthWest=4;

HexNorthEast=5;

{C}

#define HEXEAST 0

#define HEXSOUTHEAST 1

#define HEXSOUTHWEST 2

#define HEXWEST 3

#define HEXNORTHWEST 4

#define HEXNORTHEAST 5

Here is a table of DX(Change In X), and DY(Change in Y) for each direction on the Iso and Hex maps, divided into two lists, one for even Y values, and one for odd Y values.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Direction** | | **Even Y Value** | | **Odd Y Value** | |
| **Isometric** | **Hexagonal** | **DeltaX(DX)** | **DeltaY(DY)** | **DeltaX(DX)** | **DeltaY(DY)** |
| IsoEast | HexEast | +1 | 0 | +1 | 0 |
| IsoSouthEast | HexSouthEast | 0 | +1 | +1 | +1 |
| IsoSouth | N/A | 0 | +2 | 0 | +2 |
| IsoSouthWest | HexSouthWest | -1 | +1 | 0 | +1 |
| IsoWest | HexWest | -1 | 0 | -1 | 0 |
| IsoNorthWest | HexNorthWest | -1 | -1 | 0 | -1 |
| IsoNorth | N/A | 0 | -2 | 0 | -2 |
| IsoNorthEast | HexNorthEast | 0 | -1 | 1 | -1 |

As you can see, DeltaY is the same, no matter what row you are on. Only DeltaX changes.

Also, For the Cardinal Directions (North, East, South, and West), DeltaX is the same no matter what row you are on.

Its only diagonal movement that is tricky.

So now, let's build a few functions: IsoDeltaX, IsoDeltaY, HexDeltaX, and HexDeltaY.

{Pascal}

{To find out how we should modify X in order to move a given direction.}

{Dir is direction of intended movement, and OddRow is whether or not the}

{current Y position is odd or even. you can feed the expression ((Y and 1)=1}

Function IsoDeltaX(Dir:byte;OddRow:boolean):integer;

Var Temp:integer;

Begin

Temp:=0; {The default change in X is 0. We'll only modify it if we have to}

Case Dir of

IsoEast: Temp:=1;

IsoWest: Temp:=-1;

IsoNorth: Temp:=Temp-2;

IsoSouth: Temp:=Temp+2;

IsoSouthEast, IsoNorthEast: If OddRow then Temp:=1;{If Not OddRow, then leave as 0}

IsoSouthWest, IsoNorthWest: If Not OddRow then Temp:=-1; {If OddRow, the leave as 0}

End;

IsoDeltaX:=Temp;{Return the Value}

End;

{To find out how we should modify Y in order to move in a given direction.}

{Dir is the direction of intended movement}

Function IsoDeltaY(Dir:byte):integer;

Var Temp:integer;

Begin

Temp:=0;{Default Value of 0. We will change it only if we have to}

Case Dir of

IsoNorth: Temp:=-2;

IsoSouth: Temp:=2;

IsoNorthWest, IsoNorthEast: Temp:=-1;

IsoSouthWest,IsoSouthEast: Temp:=1;

End;

IsoDeltaY:=Temp;{Return the value}

End;

Function HexDeltaX(Dir:byte;OddRow:boolean):integer;

Var Temp:integer;

Begin

Temp:=0;

Case Dir of

HexEast: Temp:=1;

HexWest: Temp:=-1;

HexSouthEast, HexNorthEast: If OddRow then Temp:=1;

HexSouthWest, HexNorthWest: If Not OddRow then Temp:=-1;

End;

HexDeltaX:=Temp;

End;

Function HexDeltaY(Dir:byte):integer;

Var Temp:integer;

Begin

Temp:=0;

Case Dir of

HexNorthWest, HexNorthEast: Temp:=-1;

HexSouthWest,HexSouthEast: Temp:=1;

End;

HexDeltaY:=Temp;

End;

{C}

int isodeltax(unsigned char dir, BOOL oddrow)

{

int temp=0;

switch(dir)

{

case ISOEAST: temp=1; break;

case ISOWEST: temp=-1;break;

case ISOSOUTHEAST:

case ISONORTHEAST: if (oddrow==TRUE) temp=1; break;

case ISOSOUTHWEST:

case ISONORTHWEST: if (oddrow==FALSE) temp=-1;break;

}

return(temp);

}

int isodeltay(unsigned char dir)

{

int temp=0;

switch(dir)

{

case ISONORTH: temp=-2;break;

case ISOSOUTH: temp=2;break;

case ISOSOUTHEAST:

case ISOSOUTHWEST: temp=1;break;

case ISONORTHEAST:

case ISONORTHWEST: temp=-1;break;

}

return(temp);

}

int hexdeltax(unsigned char dir, BOOL oddrow)

{

int temp=0;

switch(dir)

{

case HEXEAST: temp=1; break;

case HEXWEST: temp=-1;break;

case HEXSOUTHEAST:

case HEXNORTHEAST: if (oddrow==TRUE) temp=1; break;

case HEXSOUTHWEST:

case HEXNORTHWEST: if (oddrow==FALSE) temp=-1;break;

}

return(temp);

}

int hexdeltay(unsigned char dir)

{

int temp=0;

switch(dir)

{

case HEXSOUTHEAST:

case HEXSOUTHWEST: temp=1;break;

case HEXNORTHEAST:

case HEXNORTHWEST: temp=-1;break;

}

return(temp);

}

# Facing and Turning

In some games, like strategy games, as well as others, the direction that something on a tile is facing is just as important as what tile they are on. (for things like arc fire, etc.)

Keeping track of facing is no big deal. It's just a byte (char) that keeps track of the unit's direction (0 to 7 for Iso, 0 to 5 for Hex)

For Turning the unit, we may want to have a function or two, as well as some turning constants.

In Iso, we turn in increments of 45 degrees, in Hex, we turn in increments of 60.

{Pascal}

Const

{Iso Turning Constants}

IsoTurnNone=0;

IsoTurnRight45=1;

IsoTurnRight90=2;

IsoTurnRight135=3;

IsoTurnAround=4;

IsoTurnLeft135=5;

IsoTurnLeft90=6;

IsoTurnLeft45=7;

{Hex Turning Constants}

HexTurnNone=0;

HexTurnRight60=1;

HexTurnRight120=2;

HexTurnAround=3;

HexTurnLeft120=4;

HexTurnLeft60=5;

Function IsoTurn(Dir,Turn:byte):byte;

Begin

IsoTurn:=(Dir+Turn) AND 7;

End;

Function HexTurn(Dir,Turn:byte):byte;

Begin

HexTurn:=(Dir+Turn) MOD 6;

End;

{C}

/\*Iso Turn Constants\*/

#define ISOTURNNONE 0

#define ISOTURNRIGHT45 1

#define ISOTURNRIGHT90 2

#define ISOTURNRIGHT135 3

#define ISOTURNAROUND 4

#define ISOTURNLEFT135 5

#define ISOTURNLEFT90 6

#define ISOTURNLEFT45 7

/\*Hex Turn Constants\*/

#define HEXTURNNONE 0

#define HEXTURNRIGHT60 1

#define HEXTURNRIGHT120 2

#define HEXTURNAROUND 3

#define HEXTURNLEFT120 4

#define HEXTURNLEFT60 5

unsigned char isoturn(unsigned char dir, unsigned char turn)

{

return((dir+turn) & 7);

}

unsigned char hexturn(unsigned char dir, unsigned char turn)

{

return((dir+turn) % 6);

}

# Mouse Matters

Another major difficulty of Iso/Hex mapping is the mouse cursor. This was one of my difficulties for a long time. Then, I took a look at one of the GIFs that shipped with Civilization II.

It had a little picture, kind of like this:

http://archive.gamedev.net/reference/articles/taniso/MouseMapIso.gif

AHA! I said. Then I understood. We don't have to do bizarre calculations in order to figure out what tile we're on! We just divide the screen (or map) into little rectangles like the one above, figure out where in a given rectangle our mouse is, and find the color on the picture above that corresponds! This will allow us to figure out which tile our mouse is hovering over. (After stumbling on to this epiphany, I promptly smacked myself in the forehead and said "DUH!")

I call the above picture the Isometric MouseMap. Here's how to use it.

(For Hex Maps, use the same algorithm, but with the following MouseMap: http://archive.gamedev.net/reference/articles/taniso/MouseMapHex.gif)

* First Step: Find out what region of the map the mouse is in.

RegionX=int(MouseX/MouseMapWidth)   
RegionY=int(MouseY/MouseMapHeight)\*2 {The multiplying by two is very important}

* Second Step: Find out WHERE in the mousemap our mouse is, by finding MouseMapX and MouseMapY.

MouseMapX=MouseX MOD MouseMapWidth   
MouseMapY=MouseY MOD MouseMapHeight

* Third Step: Determine the color in the MouseMap at (MouseMapX,MouseMapY).
* Fourth Step: Find RegionDX and RegionDY in the following table:

|  |  |  |
| --- | --- | --- |
| **Color** | **RegionDX** | **RegionDY** |
| RED | -1 | -1 |
| YELLOW | 0 | -1 |
| WHITE | 0 | 0 |
| GREEN | -1 | 1 |
| BLUE | 0 | 1 |

* Fifth Step: Use RegionX,RegionY, RegionDX, and RegionDY to find out TileX and TileY

TileX=RegionX+RegionDX   
TileY=RegionY+RegionDY

Next Time: I will discuss putting Objects onto our Iso/Hex tiles, with a minimum of muss and fuss, and proper screen updating, so you don't have to draw the whole map every time.

[**Discuss this article in the forums**](http://archive.gamedev.net/community/forums/topic.asp?key=featart&uid=747&forum_id=35&Topic_Title=Isometric+%27n%27+Hexagonal+Maps+Part+I)

Date this article was posted to GameDev.net: **9/16/1999**  
(Note that this date does not necessarily correspond to the date the article was written)