

keekercdc.com

essays and pithy thoughts

Hexagon grids: coordinate systems and distance calculations

Mar 7th, 2011 in [Statecraft](#)

A big challenge that loomed when I embarked on the Statecraft was proper implementation of a hexagon grid. I felt the move to a hex grid was one of the few things that Civ V actually got right, and had the potential to bring a more interesting military layer to the game. Unfortunately, it was done at the expense of...everything else.

Taking specific gameplay mechanics aside however, a poorly implemented hex grid, at the base technical level, has the potential to create massive headaches for me (the programmer) and millions of wasted CPU cycles. The reason for this is the very nature of trying to assign Cartesian coordinates to a honeycomb grid. Quite frankly, all the existing works browsable through Google searches on the subject of coordinate systems on hex grids are pretty piss poor, needlessly complex, or just flat out wrong; I would have to tackle the topic myself.

First things first, square grids.

Let's step back a moment and consider a normal Cartesian plane, where the x axis is horizontal and the y axis is vertical; the graphing plane familiar to us all from our first algebra class onward. If we lay down a square-based grid on top of this plane, calculating distances is a relatively easy task. You can either go the old trusty Pythagorean route, using:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

...to get the distance 'as the crow flies', or you can calculate distances atomically, using indivisible units of movement 'along the ground' from one tile to another. When considered from a board game context, we're much more interested in the latter.

Civilization games before Civ V used a square grid to divide the play surface, and a move into any of the eight neighboring squares (vertical, horizontal, or along the diagonals) were all considered to take one unit of movement. This was a rather strange way of calculating movement distances, as you could cover roughly 40% more territory by always making diagonal moves, rather than moving horizontally or vertically. Consequently, calculating the distance between two squares 'along the ground', assuming no obstacles, became relatively simple:

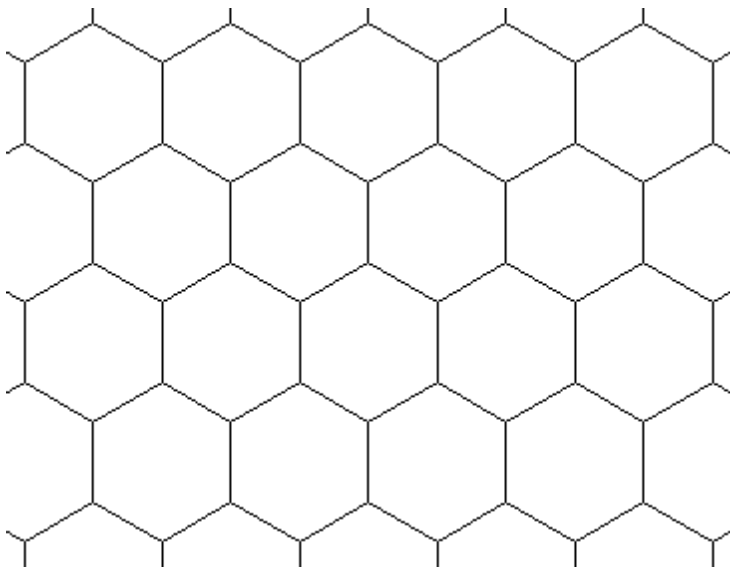
$$d = \max(x_2 - x_1, y_2 - y_1)$$

Since diagonal moves, where both x and y change, still only count as one move, the smaller of the two 'deltas' becomes inconsequential; the number of moves it takes to go from one square to another will always be the largest of the changes in coordinates.

It's easy to assign a consistent coordinate pattern to a square grid: start at $(0,0)$; every square right adds one to the x coordinate, and every square upwards adds one to the y coordinate. Things get a bit more complex when you try to do the same to a grid of hexagons, where there seemingly exists two plausible 'upwards' axes.

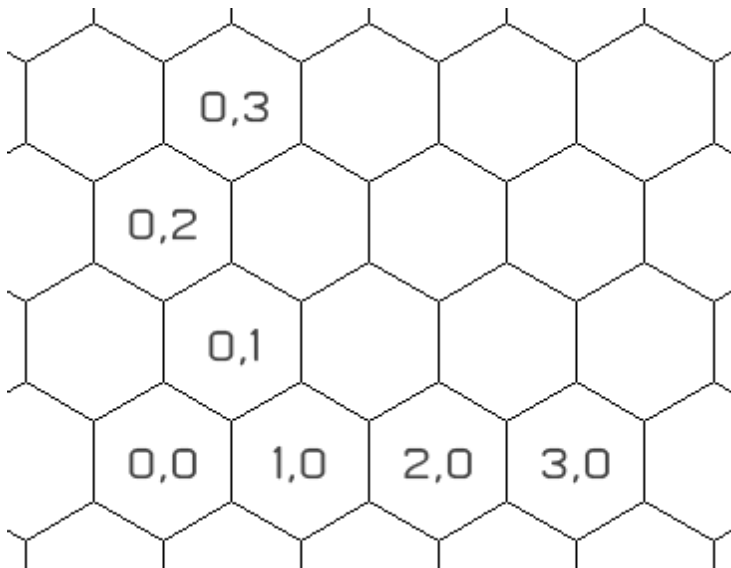
Some have taken the 'squiggly' y -axis approach to force a Cartesian feel into a hex grid.

Let's consider a small portion of a hex grid:

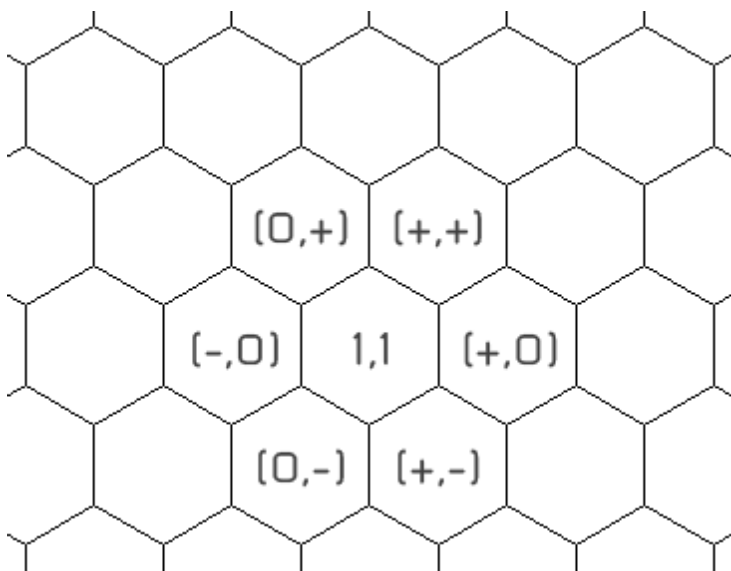


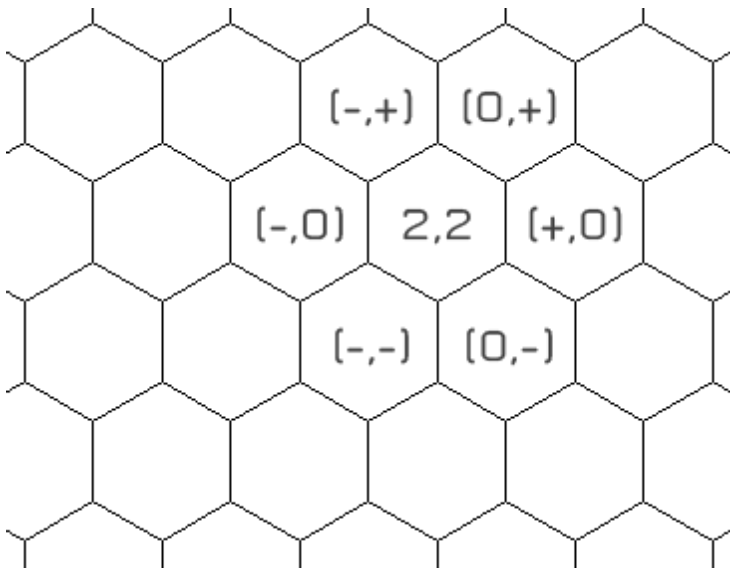
The x -axis here is an easy one to assign. You can start from any tile and go straight to the right, incrementing the value of x along the way. But in which direction is the upward y -axis?

Some folks around the internet have decided to answer this question with a 'squiggly' y -axis, wherein you retain the same value for x first up and to the right, then up and to the left, then up and to the right again, and so on. So a sample grid under this paradigm looks something like this:



Doing so, however, creates a needlessly awkward problem when trying to determine an atomic distance 'along the ground' between two tiles in this grid, because any move from one row of tiles to another has to take into account whether you're moving into a 'left-handed' column or a 'right-handed' one - the change in the x coordinate is different for a move in any of these directions based on what kind of row you're moving from.

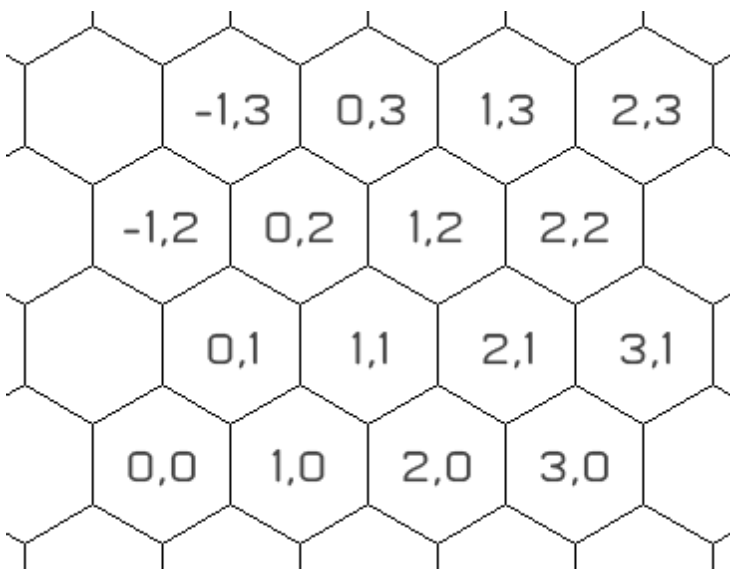




Thus, any formula you might try to put together to calculate atomic distances between two tiles, simply from their coordinates, has to take this strange 'ripple' effect you've introduced into the grid, since a movement in a particular direction can mean something different for the change in coordinates based on whether you're in an even or odd row. This is not to say that it can't be done, since it's screwed up in a consistent manner, but it's a complication that's unnecessary.

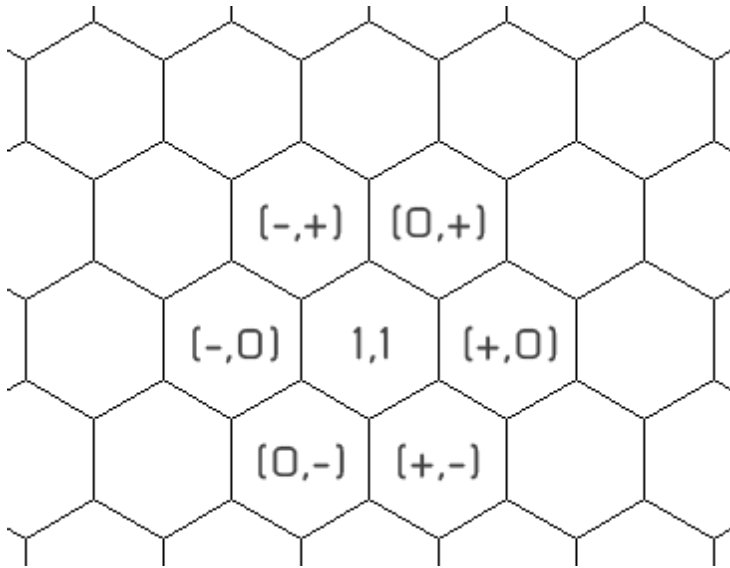
A better course is to straighten out the axes.

In order to produce a consistent pattern to movement between tiles, and make this grid truly 'flat', we need to straighten out the axes. We can do so by letting this *y-axis*, where $x=0$, lie along one of the diagonals, essentially shifting every row in the same direction. I found it simplest to tilt it up and to the right, like this:



This creates a consistent pattern for what a move from one tile to another means for your new coordinates. A move directly right or left means a change

in the x coordinate only. A move along either of the diagonals will now *always* mean a consistent change in coordinates; this is an important thing to keep in mind as we try to find a reliable means of determining atomic distances between tiles. The change in coordinates, from any tile, now looks like this:



Now we have a consistent 'grain' to our grid. If you're going 'with the grain,' either along the x - or y -axes, it means a change in that coordinate. Going 'against the grain' means a change in *both* coordinates, in opposite directions. Now, all we need is a formula, in which we can plug in two sets of coordinates and get back the distance 'along the ground' between the two. Once we have this, we've a solid foundation for solving other gameplay problems, such as pathfinding.

Problem is, a means of finding a consistent formula that produces the proper distance no matter which direction you're moving in proves illusive; you can find a formula that works in two directions, but not the third. What's actually needed is a third axis.

Hexes as three-dimensional cubes

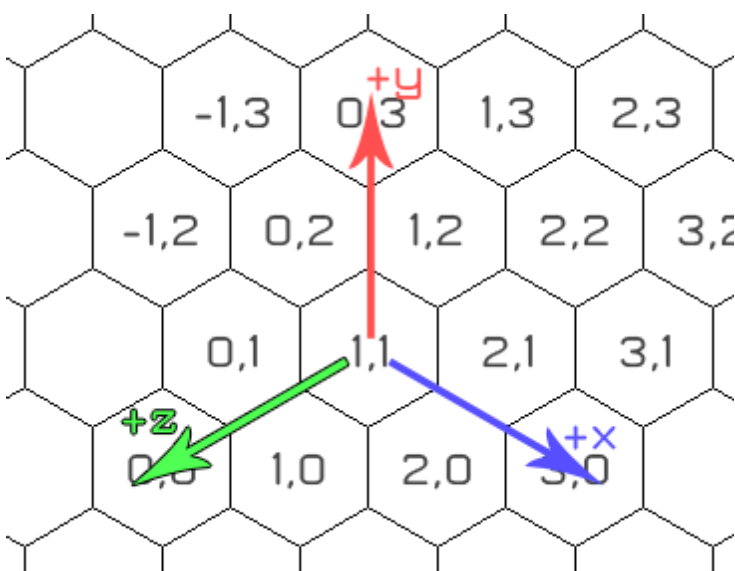
We get that third axis by adding...a third dimension.

Did anybody figure at the top of this discussion that Q*Bert had the answer all along?



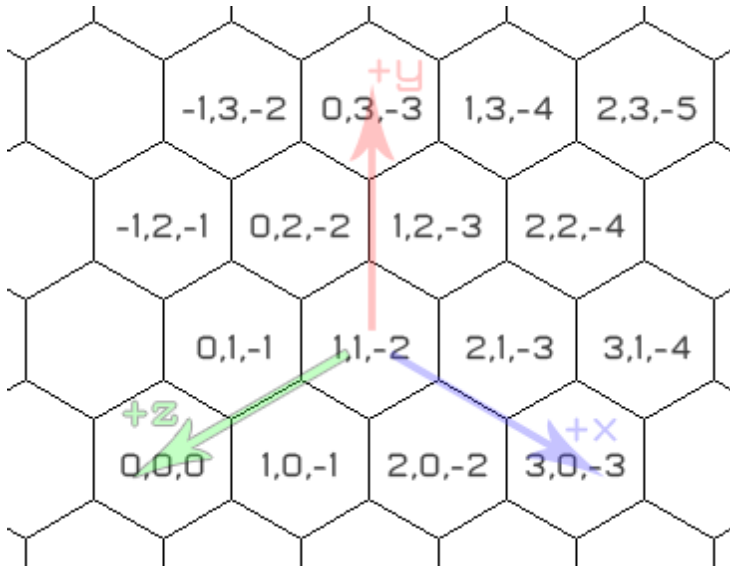
Check it. Each cube actually forms a hexagon as rendered in two dimensions – from any given cube that Q*Bert is standing on, you have 6 different one-step moves you can make. If we fill in our missing third axis on the hex grid we’ve already been working with, so that you have three axes all offset by 120 degrees, an interesting mathematical phenomenon occurs.

Here’s our grid with the new *z-axis* plotted, in terms of the direction in which each coordinate slot increases:



This way, every hex will actually have three coordinates. It’s less complicated

than you might think, however, when we actually look at this section of grid we've been working with, with the z values plotted. Have a look and see if you can pick out the interesting relationship that appears between the coordinate values within each tile:



That's right kids; the coordinates of any given tile add up to 0. $x + y + z = 0$ no matter what direction in which you traverse the grid or how far you go. Yes, that also means that the z -axis is technically a bit redundant in some situations, such as determining where to render each tile on the screen, however its advantage is clear when it comes to pathfinding and distance calculation. Now, we have a consistent coordinate system in all directions on the grid – a truly flat system – and so we can arrive upon this formula for determining the distance between any two tiles, which should look pretty similar to a distance formula we examined earlier, in the context of Civ games:

$$d = \max(x_2 - x_1, y_2 - y_1, z_2 - z_1)$$

Just as in the example of the square grid, in which a move to any of the eight neighboring tiles counts as the same unit of movement; we arrive at a similar equation with a three-coordinate hex grid in which a move to any of the six neighboring tiles counts as the same unit of movement. This is why a hex grid is preferable to a square grid when considered from the context of constraining movement to a grid – its the more accurate means of modeling movement that reaches further than one tile since there's three axes to move along instead of two, and you can use it without bending Pythagoras completely out of shape.

Applying it to gameplay

Just some quick implementation notes (mostly for me, but maybe for you too):

Tiles can still be stored in the database with x and y coordinates only, as the z coordinate is redundant in terms of uniquely identifying tiles by coordinates, as well as in terms of positioning relative to each other on render. I can figure


out where to render a tile relative to (0,0,0) simply based on x and y.

The z coordinate can always be programatically generated since $x + y + z = 0$, so I'll be writing code so that z is always generated on-demand when needed.

The orientation of the axes is relatively arbitrary. The diagrams above could be easily flipped vertically, so that y increases downward and x increases up and to the right. This would create a (0,0,0) tile in the upper left instead of the lower left. Either way works, and doesn't have any impact on the distance formula.

[← Statecraft: the simultaneous turn paradigm of 'Diplomacy' Twitter should charge users for API access. →](#)

19 Comments keekercdc.com

 Juan Ignacio Rodr... ▾ Recommend 24 Share

Sort by Oldest ▾



Join the discussion...

**Stephen** • 4 years ago

After having searched for hours on al gore's interwebs, I had not found an explanation as succinct as this. Thank you. Its actually quite simple once you put it into an intelligible perspective without all the extra fluff. +10 REP

^ | ▾ • Reply • Share >

**Ben** • 4 years ago

Very easy to understand, and makes complete sense. Nice job! Very helpful!

^ | ▾ • Reply • Share >

**Joaquin** • 4 years ago

This is awesome.

^ | ▾ • Reply • Share >

**Erwaitin** • 4 years ago

Thank you!! Great work :)

^ | ▾ • Reply • Share >

**LearningHex** • 4 years ago

Consider checking out <http://www.redblobgames.com...> as well.

^ | ▾ • Reply • Share >

**keekercdc** Author → LearningHex • 4 years ago

This post is linked there. :) Thanks, though.

^ | ▾ • Reply • Share >

**kdb** • 4 years ago

thank you for motivating the z coordinate by pointing that q*bert-style 3d environments are equivalent to 2D hexagonal grids. that, together with your discussion of the zero sum property and the z coordinate's redundancy, added a lot of completeness to my understanding.

this is a very high quality educational article.

1 ^ | ▾ • Reply • Share >

**Justin Hampton** • 4 years ago

Been struggling with a hexagon distance problem that stumped me for over a month, but

About the author

kee • kur • dee • see

I'm Chris Schetter; this is my blog.

Most of my days are spent tapping at keyboards.

I call D.C. home with my wife, daughter, and cats.

When I root, I root for [United!](#)

keekerd.c@gmail.com

[the Archives](#)

[ATOM feed](#)

[on Twitter](#)

[on Github](#)

[on LinkedIn](#)

© 2006-2016

keekerd.c.com