

# Programación Gráfica 2D ( VI )

## Mousemapping y optimizaciones .

Autor: Sergio Hidalgo  
[serhid@wired-weasel.com](mailto:serhid@wired-weasel.com)

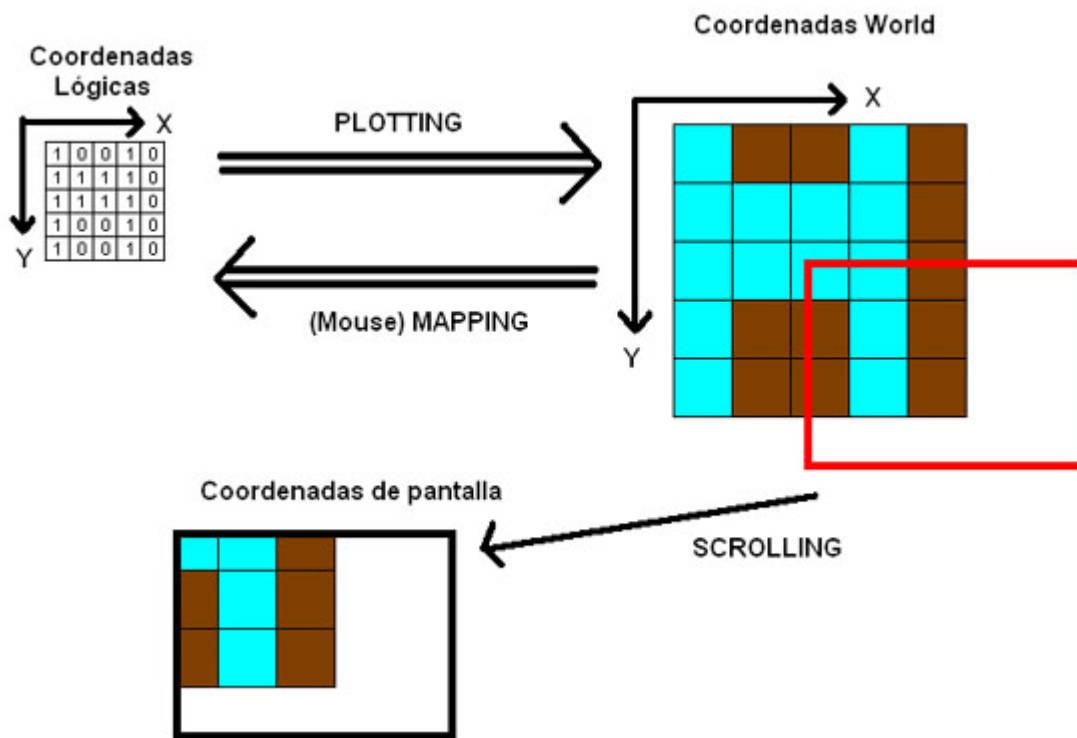
### Introducción

En el último tutorial explicaba como dibujar mapas isométricos, y como realizar el proceso de “plotting”. El proceso inverso, de convertir las coordenadas de pantalla a coordenadas lógicas (mapping), lo dejaba aparcado porque era bastante más complicado.

Pues bueno, ha llegado el momento de explicarlo:

### Recordatorio

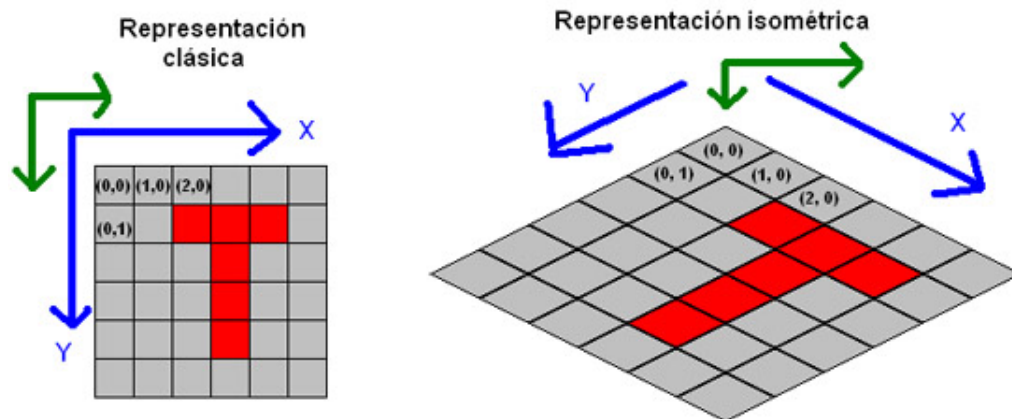
Antes de nada, vamos a ver en qué consistía el “mouse mapping” en los mapas 2D clásicos:



Simplemente, se trataba de coger unas coordenadas de pantalla, en pixeles, y obtener las coordenadas lógicas de la casilla que ocupa ese lugar. Esto era muy útil, por ejemplo, para averiguar sobre qué casilla está el ratón, o para realizar optimizaciones más adelante.

Con los mapas rectangulares, esto era muy fácil, bastaba con hacer una división. Como los tiles eran rectangulares también, no había problemas.

Sin embargo, ahora eso no es tan simple. Los tiles tienen forma de rombo, y los ejes de coordenadas lógicas y absolutas son completamente distintos:



Las flechas azules indican las coordenadas lógicas de las casillas  
 Las flechas verdes indican las coordenadas absolutas en pixels  
 Las coordenadas que aparecen en la casillas son las coordenadas lógicas

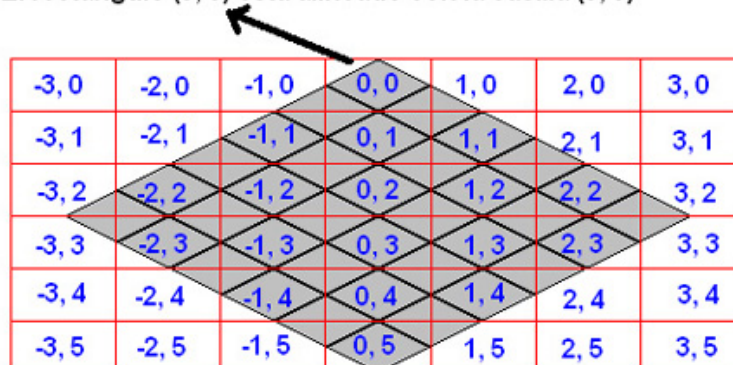
Así que hacer el mapping va a ser más difícil. Empezaré explicando el método tradicional que se suele usar, basado en una imagen de referencia. Este método funciona con cualquier tipo de mapa y cualquier relación de ancho / alto. Después comentaré un segundo método, más sencillo pero también más limitado.

## La rejilla

Hemos dicho que calcular coordenadas en mapas rectangulares es muy fácil. ¿Por qué no probamos a “transformar” el problema en algo así?. Podemos dividir el mapa en una “rejilla” rectangular, que es bastante sencilla de calcular:

División del mapa mediante una "rejilla" rectangular

El rectángulo (0, 0) está alineado con la casilla (0, 0)



En azul se indican las coordenadas de cada rectángulo o "coordenadas de rejilla"

Cada rectángulo de la rejilla tiene exactamente el tamaño de un tile, y el rectángulo (0, 0) está alineado con la casilla (0, 0). Hay que recordar que la casilla (0, 0) también hace de centro de origen de las coordenadas absolutas.

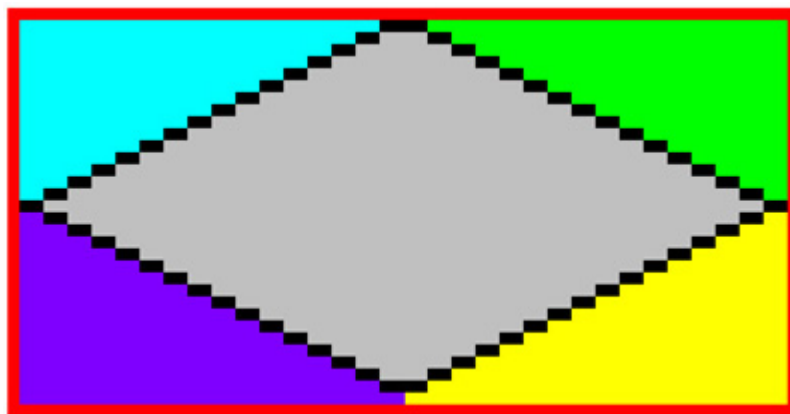
Con esto, averiguar las coordenadas del rectángulo (coordenadas de rejilla) en el que está el cursor es tan fácil como lo era en los mapas rectangulares. Simplemente basta con dividir.

```
rejilla.x = mouse.x / TILE_ANCHO;  
rejilla.y = mouse.y / TILE_ALTO;
```

**NOTA:** Antes de esto, hay que ajustar mouse para tener en cuenta el scroll. Basta con sumar el desplazamiento a las coordenadas de pantalla.

## La imagen de referencia

Bien, ahora que sabemos qué rectángulo de la rejilla contiene a la casilla pulsada, vamos a echar un vistazo más de cerca a ese rectángulo:



Vemos que hay 5 casillas en el rectángulo que han podido ser pulsadas. La del centro en gris, y las 4 “vecinas”, cada una marcada en un color.

¿Como saber cual de todas es la que ha sido pulsada?

Bueno, en primer lugar, podemos averiguar cual es la posición del cursor DENTRO del rectángulo (es decir, el pixel exacto del rectángulo que se pulsó), esto es tan simple como hacer un módulo:

```
indice.x = mouse.x % TILE_ANCHO;  
indice.y = mouse.y % TILE_ALTO;
```

En el caso de estar usando coordenadas negativas, indice.x (o indice.y) serán negativos, lo que significa que nos estamos saliendo del rectángulo de la rejilla. En ese caso, nos “desplazamos” al nuevo rectángulo en el que está el índice, y ajustamos los nuevos valores:

```
if (indice.x < 0) {  
    indice.x += TILE_ANCHO;  
    --rejilla.x;  
}
```

Y lo mismo para indice.y

Y ahora, conociendo el pixel, ¿por qué no mirar el color en la imagen que he puesto antes?. Si el color en ese pixel es verde, entonces sabemos que la casilla pulsada es la de arriba a la derecha.

Usaremos una imagen de referencia exactamente igual que la que he puesto antes, donde cada casilla viene en un color distinto. Mirando el color del pixel pulsado, podemos saber a qué casilla pertenece:

```
switch (imagenReferencia[indixe.x][indice.y].colorPixel) {  
  
case gris:  
    casilla = CENTRO;  
    break;  
  
case azul claro:  
    casilla = NO;  
    break;  
  
case verde:  
    casilla = NE;  
    break;  
  
case azul oscuro:  
    casilla = SO;  
    break;  
  
case amarillo:  
    casilla = SE;  
    break;  
}
```

**Nota:** En realidad, usaremos un array bidimensional con las dimensiones del tile, pero con números en lugar de colores. Por ejemplo, el número 3 significaría la casilla al SO, o el 4 la del SE. Esto es mejor porque acceder a un array es más rápido que bloquear una superficie, leer un pixel, y decodificar el color.

## Walking

Con todo esto ya tenemos lo suficiente para saber qué casilla fue pulsada. ¿Pero como calculamos sus coordenadas? Para eso usaremos el “tileWalker” que se vió en el otro tutorial.

Básicamente, el proceso completo es:

- Averiguar las coordenadas de rejilla y el índice.
- Calcular las coordenadas lógicas de la casilla del centro del rectángulo pulsado.
- Mirar en la imagen de referencia. Si el color es amarillo, dar un paso al SE, si es verde, dar un paso al NE, si es azul claro, dar un paso al NO, si es azul oscuro, dar un paso al SO. Si es gris, no dar ningún paso.

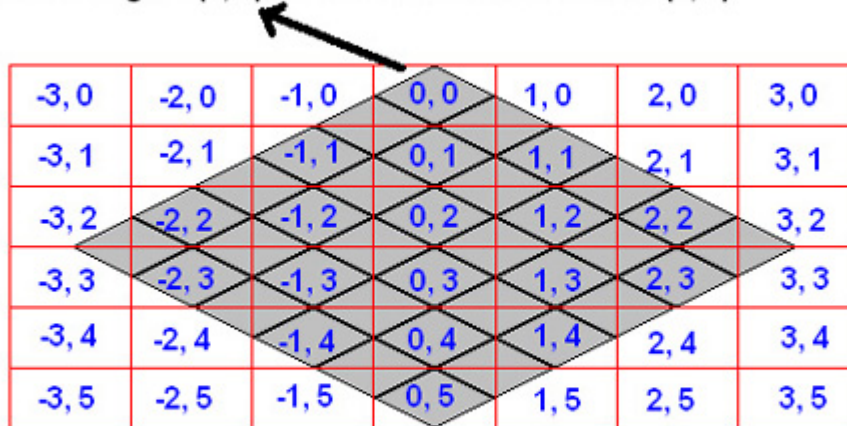
Las coordenadas resultantes son las de la casilla pulsada.

¿Pero como calcular las coordenadas de la casilla centro del rectángulo?

Echemos un vistazo a la imagen anterior:

## División del mapa mediante una "rejilla" rectangular

El rectángulo (0, 0) está alineado con la casilla (0, 0)



-3, 0	-2, 0	-1, 0	0, 0	1, 0	2, 0	3, 0
-3, 1	-2, 1	-1, 1	0, 1	1, 1	2, 1	3, 1
-3, 2	-2, 2	-1, 2	0, 2	1, 2	2, 2	3, 2
-3, 3	-2, 3	-1, 3	0, 3	1, 3	2, 3	3, 3
-3, 4	-2, 4	-1, 4	0, 4	1, 4	2, 4	3, 4
-3, 5	-2, 5	-1, 5	0, 5	1, 5	2, 5	3, 5

En azul se indican las coordenadas de cada rectángulo o "coordenadas de rejilla"

Vemos que, gracias a que los recuadros se alinean con la casilla (0, 0), podemos ir “caminando” con el tileWalker a cualquiera de ellos.

Por ejemplo, para ir al rectángulo (1, 2), desde la casilla (0, 0), podemos movernos un paso al Este, y dos al Sur.

Para ir al rectángulo (3, 2), nos movemos 3 al Este, y 2 al Sur.

Para ir al rectángulo (x, y), nos movemos x al Este, y y al Sur.

```
casilla = tileWalk (ISOD_E, Punto(0,0), rejilla.x);  
casilla = tileWalk (ISOD_S, casilla, rejilla.y);
```

**Nota:** También funciona con valores negativos.

Y después simplemente daremos el último paso que nos indique la tabla de referencia dependiendo del color. Y el resultado serán las coordenadas lógicas de la casilla pulsada, que será el valor que devuelva la función.

¿Facil, verdad? xD

## Otro método

Como he comentado antes, este método de la imagen de referencia funciona con cualquier mapa y cualquier relación de aspecto. Pero si sabemos que siempre vamos a usar mapas con forma de rombo, y que la relación siempre va a ser 2:1, podemos simplificarlos bastante la vida, usando el siguiente código:

```

x0 = mouse.x - (TILE_ANCHO/2);
y0 = mouse.y;

casilla.x = y0 + (x0 / 2);
casilla.y = y0 - (x0 / 2);
casilla.x /= TILE_ALTO;
casilla.y /= TILE_ALTO;

```

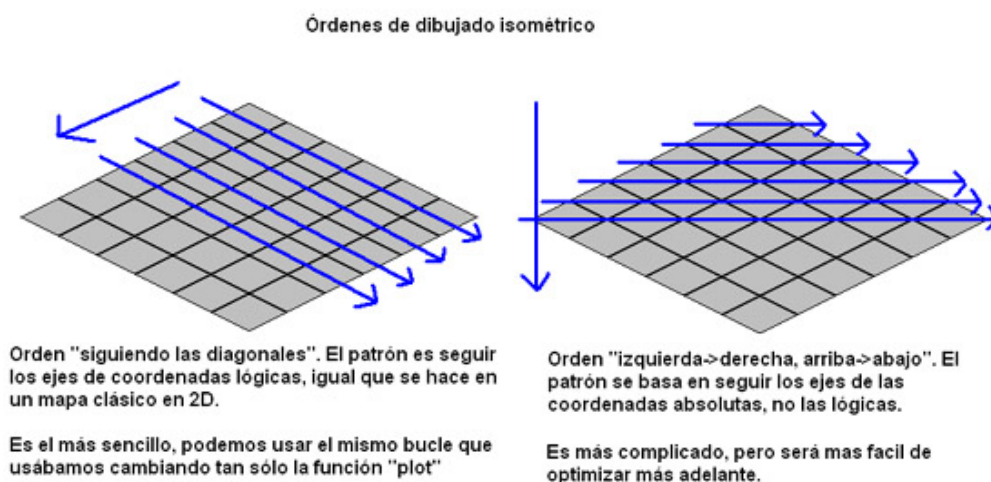
Para saber de donde sale esto os recomiendo mirar [este tutorial](#).

Nosotros usamos como valor de ajuste horizontal en  $x0$  la mitad del ancho del tile, dado que nuestro origen de coordenadas no está en el centro de la casilla, sino en la esquina superior izquierda del rectángulo del tile. Comentaré mas sobre anclas y desplazamientos en futuros tutoriales.

## Optimizaciones

Hay que recordar que estamos dibujando el mapa COMPLETO en cada fotograma. Esto simplemente es inaceptable. En cuanto el mapa sea un poco grande, irá lentísimo. Lo que nos interesa es hacer como hacíamos con los mapas rectangulares, y dibujar solamente el “trozo” del mapa que está en la pantalla.

Pero para eso hay que cambiar el orden de dibujado:



A partir de ahora usaremos el orden “izquierda->derecha, arriba->abajo”.

El algoritmo de dibujado se puede dividir en varias fases:

- En primer lugar, calculamos las “esquinas” del area que dibujaremos. Esto es parecido a lo que hacíamos para los mapas rectangulares (si no os acordais, mirad el tutorial IV, por favor).
- Después, ampliaremos el area de dibujado, para asegurarnos que todas las casillas que pueden aparecer en pantalla se dibujan siempre.
- En tercer lugar, recorreremos este area en el orden indicado con dos bucles, dibujando cada casilla.

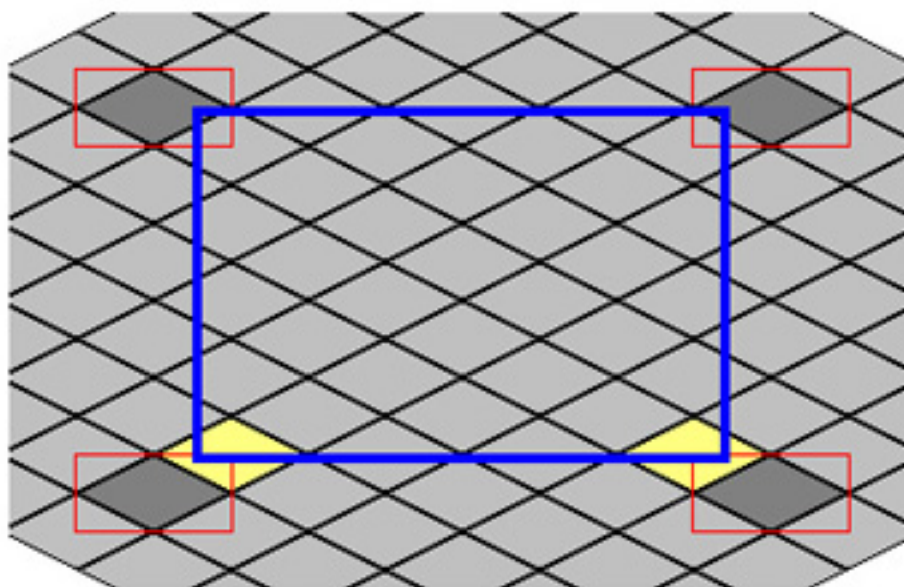
Bueno, pues empecemos. Esto lo voy a explicar rapidito xD



## Fase 1

Para calcular las casillas sobre las que están cada esquina, usaremos el mismo proceso de “mouse mapping” que hemos usado antes (el primero). Sólo que con una variación. Nos detendremos cuando hayamos calculado las coordenadas de la casilla “centro” del rectángulo. No nos interesa calcular la casilla exacta.

### 1ª Fase - Determinar las casillas de las esquinas



**En rojo, las rejillas correspondientes a las esquinas de la pantalla**

**En azul, los bordes de la pantalla**

**En gris oscuro, las casillas "centro" de cada rejilla**

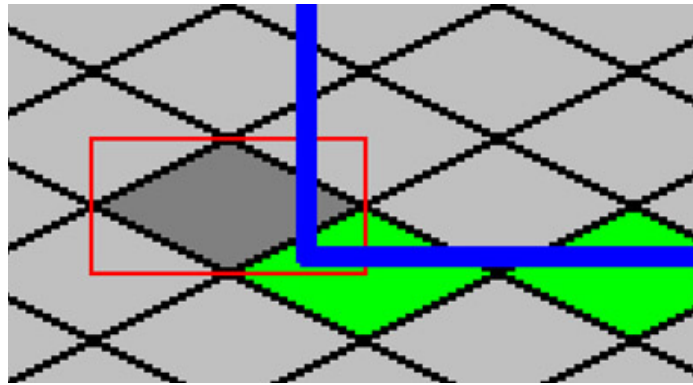
**En amarillo, las casillas en las que "realmente" se encuentran las esquinas**

La razón de eso es porque las casillas “centro” siempre están alineadas. Mientras que con las casillas exactas no ocurre eso. Fijaros en la imagen como las casillas amarillas de abajo no están en la misma columna que las superiores.

Para que el algoritmo funcione, necesitamos que las casillas “límite” estén siempre alineadas.

## Fase 2

Al usar únicamente las casillas centro, pueden darse situaciones como esta:



En este caso, las casillas verdes nunca se dibujarían, aunque parte de ellas caen dentro de la pantalla. Por lo tanto, tenemos un problema.

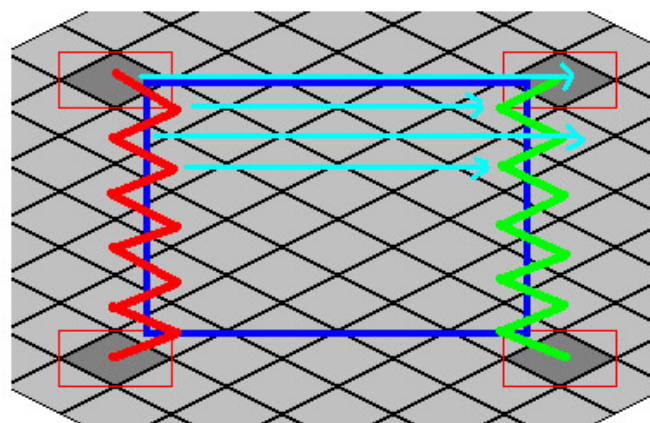
La solución es simple, basta con ampliar el marco de dibujado, alejándonos de los bordes de la pantalla un paso:

```
supIzq = tileWalk (ISOD_NO, supIzq, 1);
supDch = tileWalk (ISOD_NE, supDch, 1);
infIzq = tileWalk (ISOD_SO, infIzq, 1);
infDch = tileWalk (ISOD_SE, infDch, 1);
```

**NOTA:** En el caso de dibujar objetos, tendremos que desplazar los límites inferiores hacia abajo, para asegurarnos de que aunque la casilla en la que está el objeto caiga fuera de la pantalla, la parte superior del mismo se dibuje correctamente. Si la altura máxima de los posibles objetos es  $h$ , desplazaremos hacia abajo ( $h / \text{TILE\_ALTO}$ ) veces.

### Fase 3

Una vez hemos hecho todo lo anterior, la tercera fase es bastante simple. Tendremos dos marcadores que nos indicaran la casilla inicial y final de cada fila de casillas. Iremos moviendo esos marcadores hacia abajo, y por cada fila, dibujamos todas las casillas de izquierda a derecha.



Marcador de inicio de la fila

Marcador de fin de la fila

Orden de dibujo de casillas



Un par de cosas a tener en cuenta es que los marcadores no los movemos al sur, porque nos saltaríamos filas, sino que los movemos alternativamente al SO, y SE.

Y cuando un marcador se mueve en una dirección, el otro lo hace en la contraria. Es decir, si el marcador de inicio se mueve al SO, el de fin lo hace al SE, y viceversa.

Teneis los detalles implementados en el siguiente código:

```
void MotorGrafico::dibujar (Rectangulo area) {

    //Coordenadas de las esquinas
    Punto supIzq, supDch, infIzq, infDch;
    Punto rejilla;
    //Variables para recorrer filas y casillas
    Punto filaInicio, filaFin, filaActual;
    int contadorFilas = 0;
    //Centinelas
    bool terminado, terminadoFila;

    /*****
    /* FASE 1 - Calculamos las coordenadas de las casillas en cada esquina */
    *****/
    //Esquina superior izquierda:
    rejilla = calcularRejilla (Punto (area.x, area.y));
    supIzq = tileWalk (ISOD_E, Punto (0, 0), rejilla.x);
    supIzq = tileWalk (ISOD_S, supIzq, rejilla.y);
    //Esquina superior derecha:
    rejilla = calcularRejilla (Punto (area.x + area.w, area.y));
    supDch = tileWalk (ISOD_E, Punto (0, 0), rejilla.x);
    supDch = tileWalk (ISOD_S, supDch, rejilla.y);
    //Esquina inferior izquierda:
    rejilla = calcularRejilla (Punto (area.x, area.y + area.h));
    infIzq = tileWalk (ISOD_E, Punto (0, 0), rejilla.x);
    infIzq = tileWalk (ISOD_S, infIzq, rejilla.y);
    //Esquina inferior derecha:
    rejilla = calcularRejilla (Punto (area.x + area.w, area.y + area.h));
    infDch = tileWalk (ISOD_E, Punto (0, 0), rejilla.x);
    infDch = tileWalk (ISOD_S, infDch, rejilla.y);

    /*****
    /* FASE 2 - Ampliamos el marco de dibujado */
    *****/
    //Desplazamos cada esquina para alejarnos de la pantalla
    supIzq = tileWalk (ISOD_NO, supIzq, 1);
    supDch = tileWalk (ISOD_NE, supDch, 1);
    infIzq = tileWalk (ISOD_SO, infIzq, 1);
    infDch = tileWalk (ISOD_SE, infDch, 1);
    //Desplazamos las esquinas inferiores 2 pasos al sur para
    //compensar por los objetos altos
    infIzq = tileWalk (ISOD_S, infIzq, 2);
    infDch = tileWalk (ISOD_S, infDch, 2);

    /*****
    /* FASE 3 - Bucle de dibujado */
    *****/
    terminado = false;
    filaInicio = supIzq;
    filaFin = supDch;
```

```

//Para cada fila
while (!terminado) {
    terminadoFila = false;
    //Seleccionamos la primera casilla
    filaActual = filaInicio;
    //Para cada casilla
    while (!terminadoFila) {
        //Dibujamos la casilla
        dibujarCasilla(filaActual);
        //Comprobamos si hemos llegado al final de la fila y si no,
        //nos movemos a la siguiente casilla
        if (filaActual == filaFin)
            terminadoFila = true;
        else
            filaActual = tileWalk (ISOD_E, filaActual, 1);
    }
    //Comprobamos si la fila recorrida era la ultima
    if ((filaInicio == infIzq) && (filaFin == infDch))
        terminado = true;
    else {
        //Si no lo era, movemos las casillas de inicio y fin
        //hacia abajo para comenzar con la siguiente
        if (contadorFilas & 1) {
            //Fila impar

            filaInicio = tileWalk (ISOD_SO, filaInicio, 1);
            filaFin = tileWalk (ISOD_SE, filaFin, 1);
        } else {
            //Fila par
            filaInicio = tileWalk (ISOD_SE, filaInicio, 1);
            filaFin = tileWalk (ISOD_SO, filaFin, 1);
        }
        ++contadorFilas;
    }
}
}

```

**NOTA:** Como veis en el código, no dibujamos la pantalla, sino un rectángulo llamado “area”. Por lo general, area abarcará toda la pantalla, así que dará igual. Pero al hacer esto tenemos la opción de dibujar regiones más pequeñas si nos hace falta mas adelante.

Y esto es todo, un saludo.