

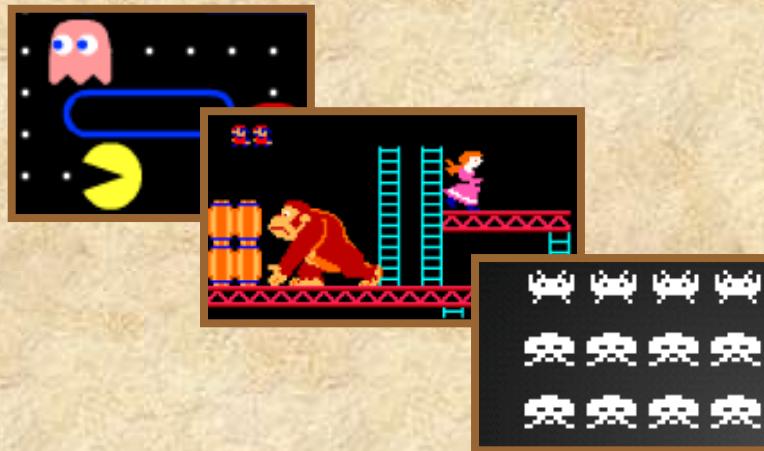


**HY454 : ΑΝΑΠΤΥΞΗ ΕΞΥΠΝΩΝ ΔΙΕΠΑΦΩΝ ΚΑΙ
ΠΑΙΧΝΙΔΙΩΝ**

**ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ,
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ,
ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ**



**ΔΙΔΑΣΚΩΝ
Αντώνιος Σαββίδης**



ΑΝΑΠΤΥΞΗ ΠΑΙΧΝΙΔΙΩΝ, Διάλεξη 5η



Περιεχόμενα

- ***Tile-based games***
- Tiles and tile-bitmaps
- Tile-based terrains
- Basic display method
- Grid motion control
- Grid computation

Tile-based games (1/3)

- Πρόκειται για platform 2D1/2 games, που διαδραματίζονται σε έναν επίπεδο χώρο, μέρος μόνο του οποίου φαίνεται στην οθόνη, με scrolling προς όλες τις κατευθύνσεις.
- Είναι πολύ συνηθισμένο ο χώρος ενός τέτοιου παιχνιδιού να έχει διαστάσεις της τάξης των δεκάδων χιλιάδων pixels, π.χ. 32000x8000.

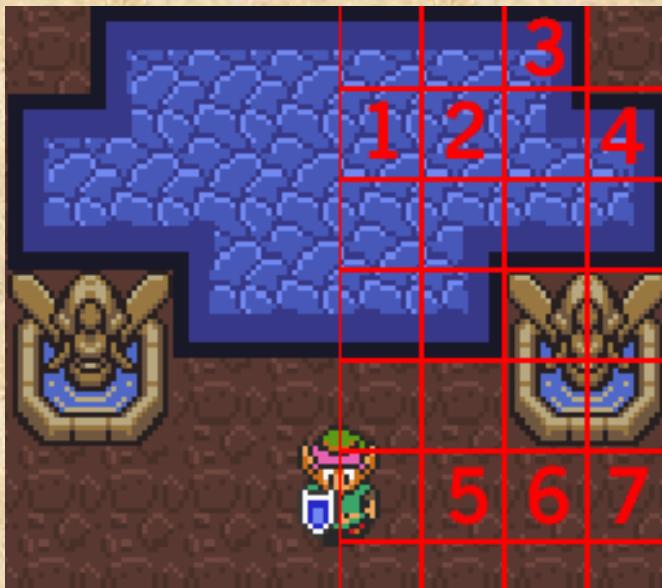


Ο χώρος δράσης είναι πολύ μεγαλύτερος από τον χώρο της οθόνης, ενώ πάντα είναι ορατό το «παράθυρο» (view window) στο οποίο βρίσκεται ο παίκτης (ο βασικός «ήρωας»)

Από το platform game Lomax...

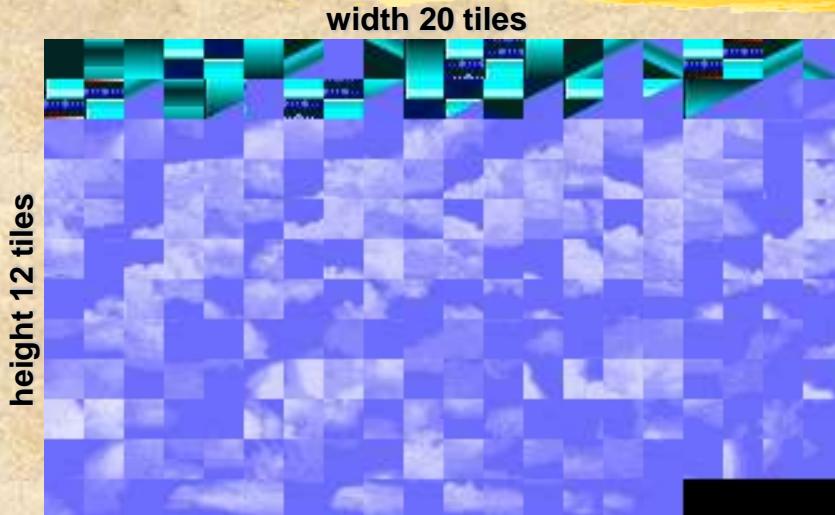
Tile-based games (2/3)

- Ένα από τα προφανή κατασκευαστικά προβλήματα σε τέτοιου είδους παιχνίδια είναι ότι ο χώρος δράσης, εάν αποθηκευτεί ως bitmap, είναι **τρομακτικά μεγάλος** (τουλάχιστον για τα σημερινά δεδομένα).
- Π.χ. για 32000x8000 απαιτούνται 256000000 pixels που στα 24 bits σημαίνει 8192000000 δηλ. 8GB. Ακόμη και αν πετύχουμε compression στο δίσκο, στη μνήμη πρέπει απαραίτητα να καταλαμβάνει τόσο χώρο, κάτι που είναι σχετικά ασύμφορο.



- Η τεχνική που χρησιμοποιείται σε αυτού του είδους τα games είναι ο κατακερματισμός του terrain, δηλ. του χώρου του παιχνιδιού σε μικρό αριθμό από επαναλαμβανόμενα και συνδεόμενα μεταξύ τους bitmaps.
- Αυτά τα bitmaps ονομάζονται tiles, δηλ. «πλακίδια», ή «πλακάκια»
- Στη διπλανή εικόνα, μεταξύ άλλων, τα εξής tiles είναι ίδια μεταξύ τους
 - 1=2, 3=4, 5=6=7

Tile-based games (3/3)



• Η επιλογή των κατάλληλων tiles έχει πάντα σχέση με το είδος του παιχνιδιού, ενώ θα πρέπει να έχουμε σχετικά λίγα tiles (π.χ. 256 επιθυμητό, 512 επαρκές, 1024 αρκετά πλούσιο και πιο αργό).

• Θα δείτε σε λίγο πως ο αριθμός των tiles επηρεάζει τρομακτικά τη μνήμη του παιχνιδιού.

• Το bitmap που περιέχει τα tiles ενός game το ονομάζουμε tiles bitmap και συνήθως θέλουμε να είναι resident στη video RAM.

• Τα tiles πρέπει να σχεδιάζονται έτσι ώστε να μπορούν να συνδυάζονται σε αναδρομικές δομές. Κάτι τέτοιο θέλει πολύ καλούς γραφίστες (υπάρχουν γραφίστες που ειδικεύονται στη σχεδίαση tiles).





Περιεχόμενα

- Tile-based games
- ***Tiles and Tile bitmaps***
- Tile-based terrains
- Basic display method
- Grid motion control
- Grid computation

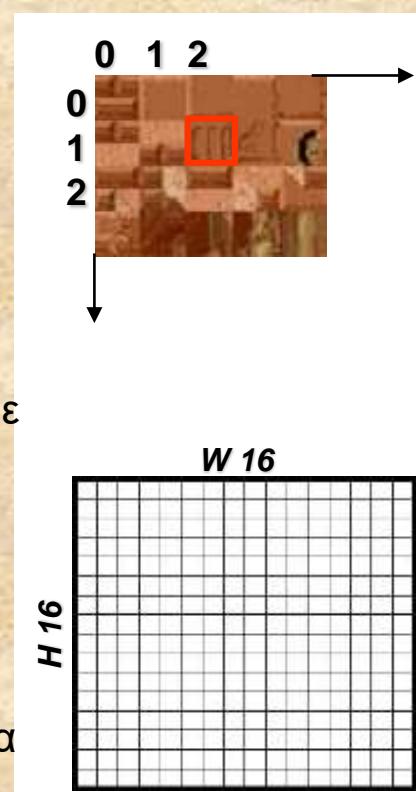


Tiles and tile bitmaps (1/8)

- Η επιλογή του αριθμού των tiles καθώς και της διάταξης μέσα στο bitmap (π.χ. **20x12**) είναι σημαντική απόφαση
 - επηρεάζει τις απαιτήσεις τόσο σε μνήμη όσο και την ταχύτητα
- Συνήθως χρησιμοποιούμε tiles με διαστάσεις 16 ή 32 pixels, όπως **16x16, 16x32, 32x16** και **32x32**
 - Όσο μεγαλύτερα τα tiles, τόσο ταχύτερα μπορεί να γίνει rendered ένα view window, καθώς απαιτούνται λιγότερα *blits*
 - αλλά τόσο καλύτερη πρέπει να είναι η γραφική σχεδίαση καθώς τα μεγάλα tiles δύσκολα συνδέονται
 - ενώ συνήθως απαιτείται μεγαλύτερος αριθμός που σημαίνει ότι έχουμε μεγαλύτερο tile set (bitmap)
- Η χρήση tiles μεταβλητού μεγέθους δε συνίσταται καθώς μπορούν αυτά να εξομοιωθούν με επιμέρους μικρότερα tiles
 - χωρίς να αυξάνουμε την πολυπλοκότητα του game engine

Tiles and tile bitmaps (2/8)

- Η θέση ενός tile μέσα στο **tile set** (bitmap) ορίζεται από έναν μοναδικό αριθμό που λέγεται **tile index**
 - Π.χ. το διπλανό tile, με tile bitmap 20x12 tiles, είναι στην σειρά 1 και στήλη 2, έτσι μπορεί να πάρει τον μοναδικό αριθμό (Index) $1 * 20 + 2 = 22$
 - Καθώς έχουμε 240 tiles το index αυτό μπορεί να αποθηκευτεί σε ένα byte (*unsigned char*)
 - ◆ Ενώ εάν αποθηκεύαμε ξεχωριστά row και column θα θέλαμε 2 bytes
 - ◆ Δυστυχώς δεν μπορούμε να βάλουμε και τα δύο σε ένα byte καθώς θέλουμε 0...19, 5 bits για column και 0...11, 4 bits για row
 - ◆ μπορούμε με διαφορετική διάταξη στο tile set: εάν χρησιμοποιήσουμε 16x16 bitmap, με 256 tiles (περισσότερα από πριν), χρειάζομαι 0...15, 4 bits, για row / column





Tiles and tile bitmaps (3/8)

- Γιατί θέλουμε να κάνουμε το compression αφού έτσι και αλλιώς καταφέραμε μέσω του index να αντιστοιχίσουμε μοναδικά κάθε tile σε ένα byte?
 - Με την μέθοδο του index ως σειρά εμφάνισης από **αριστερά → δεξιά** και **πάνω → κάτω**, πρέπει να κάνω τα εξής για να προσδιορίσω τη θέση μέσα στο bitmap:
 - ◆ $row = index / 20$ και $column = index \% 12$
 - ◆ παραπάνω αυτές είναι χρονοβόρες και έχω δύο επιλογές: pre-caching η χρήση tile set με διαστάσεις που είναι δυνάμεις του 2

```
unsigned char divIndex[240];
unsigned char modIndex[240];
for (i=0; i<240; ++i)
    divIndex[i] = i/20, modIndex[i] = i%12;
row      = divIndex[index];
column  = modIndex[index];
```

Αυτό εκτελείται κατά το start-up του προγράμματος «εφ άπαξ»



Tiles and tile bitmaps (4/8)

- Αλλά με την αποθήκευση σε ένα byte απλώς χρειάζομαι κατάλληλα masks
- Επίσης, πολύ γρήγορος είναι και ο υπολογισμός της θέσης ενός tile μέσα σε ένα bitmap
- Και μπορεί να επιταχυνθεί ακόμη περισσότερο γνωρίζοντας ότι ο πολλαπλασιασμός με δυνάμεις του 2 ισοδυναμεί με shift left (να γιατί προτιμούμε διαστάσεις 16 ή 32)
- και ακόμη περισσότερο με απευθείας εξαγωγή των row / column values και shifts χωρίς κλήση στην GetRow και GetCol
- **Εντάξει, είναι απαραίτητο τέτοιου είδους low-level hacking?**
 - Θα δείτε ότι αυτοί οι υπολογισμοί γίνονται μέσα στο rendering ή display loop συνεχώς, επηρεάζοντας σημαντικά την ταχύτητα

```
#define TILESET_WIDTH      16 // row = 16 tiles
#define TILESET_HEIGHT     16 // col = 16 tiles
#define TILE_WIDTH          16
#define TILE_HEIGHT         16
#define ROW_MASK            0x0F
#define COL_MASK            0xF0
#define COL_SHIFT           4
typedef unsigned char byte;
byte MakeIndex (byte row, byte col)
{ return (col << COL_SHIFT) | row; }
byte GetCol (byte index)
{ return index >> COL_SHIFT; }
byte GetRow (byte index)
{ return index & ROW_MASK; }

Dim TileX (byte index)
{ return GetCol(index) * TILESET_WIDTH; }
Dim TileY (byte index)
{ return GetRow(index) * TILESET_HEIGHT; }

#define MUL_TILE_WIDTH(i) ((i)<<4)
#define MUL_TILE_HEIGHT(i)((i)<<4)
#define DIV_TILE_WIDTH(i) ((i)>>4)
#define DIV_TILE_HEIGHT(i)((i)>>4)
#define MOD_TILE_WIDTH(i) ((i)&15)
#define MOD_TILE_HEIGHT(i)((i)&15)

Dim TileX2 (byte index)
{ return MUL_TILE_WIDTH(GetCol(index)); }

Dim TileY2 (byte index)
{ return MUL_TILE_HEIGHT(GetRow(index)); }
```



Tiles and tile bitmaps (5/5)

- Αν έχουμε περισσότερη μνήμη μπορούμε να αποθηκεύσουμε στο index τη θέση του tile σε pixels
 - Για 16x16 tile set αρκούν 2 bytes, καθώς έχω 256x256 pixels για το tile bitmap
 - Χρειάζομαι έναν *unsigned 16 bits* (π.χ. unsigned short, αλλά να κάνετε ένα check στον compiler που χρησιμοποιείτε)
 - Με τον τρόπο αυτό με ένα tile index μπορώ ταχύτατα να κάνω *blit* από το tile bitmap σε οποιοδήποτε άλλο target bitmap

```
typedef unsigned short Index; // [MSB X][LSB Y]
#define      TILEX_MASK    0xFF00
#define      TILEX_SHIFT   8
#define      TILEY_MASK    0x00FF
Index MakeIndex2 (byte row, byte col)
{ return (MUL_TILE_WIDTH(col) << TILEX_SHIFT) | MUL_TILE_HEIGHT(row); }
Dim TileX3 (Index index) { return index >> TILEX_SHIFT; }
Dim TileY3 (Index index) { return index & TILEY_MASK; }

void PutTile (Bitmap dest, Dim x, Dim y, Bitmap tiles, Index tile) {
    BitmapBlit(
        tiles, Rect { TileX3(tile), TileY3(tile), TILE_WIDTH, TILE_HEIGHT },
        dest, Point{ x, y }
    );
}
```



Ένθετο

[Pre-caching (or memoization)]

The row caching function is:

```
unsigned char(byte) f_row(index i:[0,240))  
|Domain(index)=240| x sizeof(Result(byte)) = Mem(f_row),  
=> is low thus acceptable for precaching
```

In general, let f function in inner loops where:

T f(T1 a1,...Tn an)

- 1) make sure we have discrete domains for T_i
- 2) alternatively make sure quantization is possible without loss of precision
- 3) drop all values or ranges from $D(T_i)$ that never occur in game
- 4) verify that $|D(T_i)|$ is small enough
- 5) verify that $|D(T_1)| \times \dots \times |D(T_n)|$ is acceptable
- 6) in case the $D(T_i)$ is only determined at runtime then use runtime caching



Περιεχόμενα

- Tile-based games
- Tiles and tile-bitmaps
- ***Tile-based terrains***
- Basic display method
- Grid motion control
- Grid computation

Tile-based terrains (1/5)

- To terrain είναι ένας ορθογώνιος χώρος φτιαγμένος από tiles
- Η απλούστερη δομή που μπορεί κάποιος να φανταστεί είναι ένας πίνακας από tile indices που συνήθως λέγεται **tile map**



Tile map editor (*drop pane*)



Tile selector (*pick pane*)



Tile-based terrains (2/5)

A first code design trial to handle tile maps, is not final

```
// fixed size, game requirements lock these values
#define      MAX_WIDTH     1024
#define      MAX_HEIGHT    256

typedef Index TileMap[MAX_WIDTH][MAX_HEIGHT];
static TileMap map; // example of a global static map

void SetTile (TileMap* m, Dim col, Dim row, Index index)
{ (*m)[row][col] = index; }

Index GetTile (const TileMap* m, Dim col, Dim row)
{ return (*m)[row][col]; }

void WriteBinMap (const TileMap* m, FILE* fp)
{ fwrite(m, sizeof(TileMap), 1, fp); } // simplistic...
bool ReadBinMap (TileMap* m, FILE* fp)
{ /* binary formatted read, like descent parsing */ }

void WriteTextMap (const TileMap*, FILE* fp)
{ /* custom write in text format */ }
bool ReadTextMap (TileMap* m, FILE* fp)
{ /* parsing... */ }
```



Tile-based terrains (3/5)

- Θα δούμε ότι χρειάζεται να μπορούμε να ορίσουμε πως σε κάποια θέση του terrain έχουμε «κενό» tile
- Για το λόγο αυτό το **index 0** συνήθως χρησιμοποιείται
 - Θα μπορούσε να είναι οποιοδήποτε άλλο index αλλά η σύγκριση με το 0 είναι η ταχύτερη που υπάρχει και επίσης **0=null=none**
- Σχεδόν πάντα κατασκευάζουμε και tile editor με τον οποίο δημιουργούμε το terrain και το αποθηκεύουμε σε μορφή κατάλληλη για το παιχνίδι
 - Απλούστερη περίπτωση: binary read / write χωρίς ελέγχους
 - Ο tile editor είναι απαραίτητο εργαλείο που θα πρέπει να φτιάξετε ή να βρείτε έτοιμο για τη δημιουργία του κόσμου του παιχνιδιού



Tile-based terrains (4/5)

- Μία τακτική που των tile-based terrains και εφαρμόζεται σήμερα με παραλλαγές είναι η εξής:
 - Η **οπτική δομή** του terrain οργανώνεται με τα tiles
 - Πλέον απαιτητικά τμήματα σε γραφικά γίνονται αντικείμενα οποιονδήποτε διαστάσεων και σε οποιαδήποτε θέση
 - Η **λογική γεωμετρία** του terrain οργανώνεται ως ένα extra grid και χρησιμοποιείται αποκλειστικά για να αποτυπώνονται τα μηδιαβατά (solid) τμήματα του terrain
 - ◆ Με grid elements ίδιων διαστάσεων με τα tiles
- ☞ **Ta δύο παραπάνω συγχρονίζονται σωστά μέσω ενός editor πρακτικά τα grid tiles δεν έχουν ρόλο στο display**
- ☞ **Ενώ τα bitmaps δεν έχουν ρόλο στους ελέγχους γεωμετρίας κατά τη διάρκεια της δράσης**

Tile-based terrains (5/5)



- The scene is fully composed of tiles (visual structure)
- The grid contains empty cells except the designated ones to bound motion



Περιεχόμενα

- Tile-based games
- Tiles and tile-bitmaps
- Tile-based terrains
- *Basic display method*
- Grid motion control
- Grid computation



Basic display method (1/9)

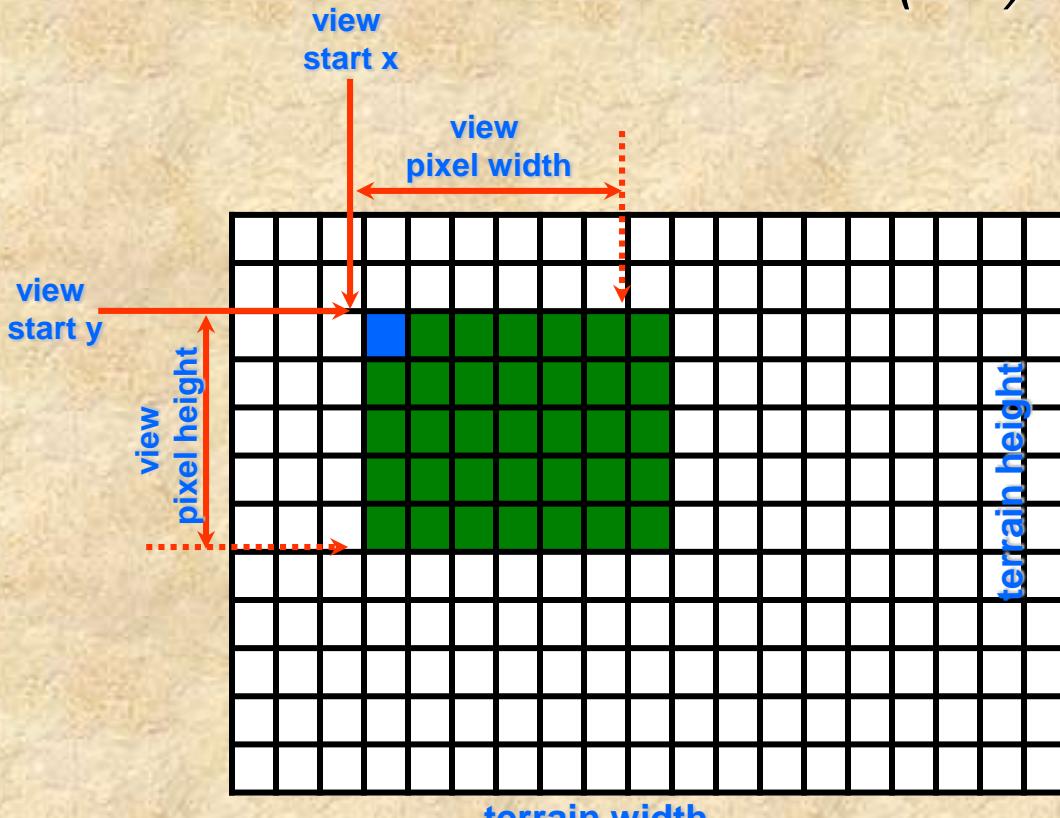
■ *View window (1/4)*

- Ένα terrain αποτελούμενο από tiles (ένα tile map) συνιστά το συνολικό χώρο του παιχνιδιού
- Το μέγεθος ενός terrain είναι κατά πολύ μεγαλύτερο της οθόνης
 - ◆ π.χ. το προηγούμενο που ορίσαμε είναι 16384x3840 pixels δηλ. εμβαδόν 132 οθονών σε 800x600 resolution
- Καθώς η «δράση» διαδραματίζεται στο terrain, πρέπει να υπάρχει ένα τμήμα του το οποίο είναι ανά πάση στιγμή το «κέντρο» της δράσης και ταυτόχρονα αυτό που παρουσιάζεται στην οθόνη
- Το τμήμα αυτό είναι ορθογώνιο, είναι πάντα μικρότερο σε εμβαδόν (pixels) από το resolution της οθόνης, ενώ ονομάζεται *terrain view window*

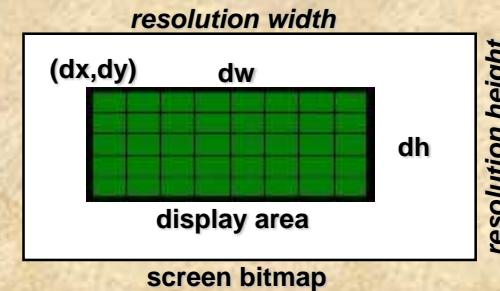


Basic display method (2/9)

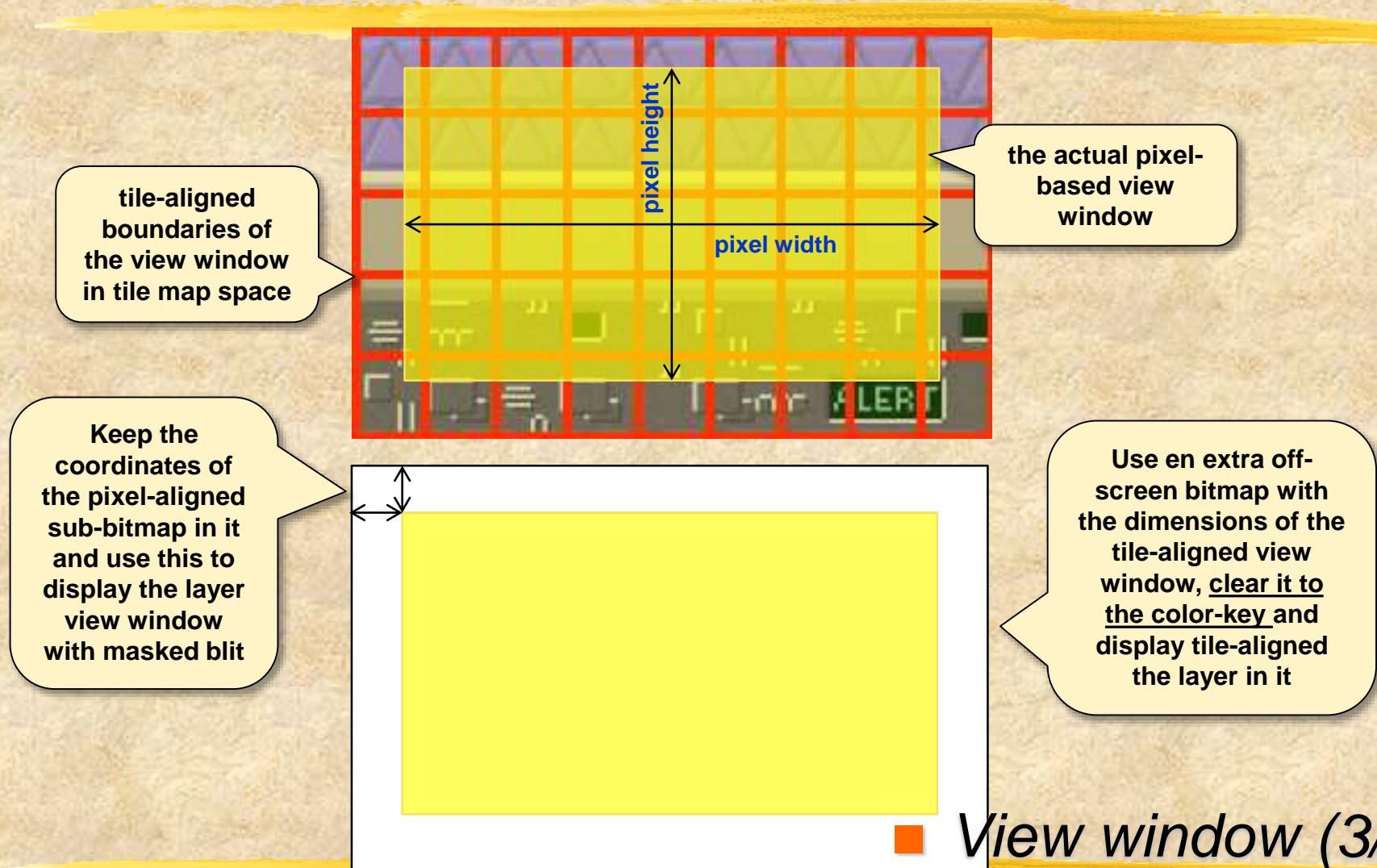
View window (2/4)



- Το view window ορίζεται πάντα σε pixel coordinates και dimensions
- Το origin του view window δε χρειάζεται να είναι σε tile boundaries (δηλ. μη σας ξεγελάει το σχήμα στα αριστερά)
- Το view window είναι ίδιων διαστάσεων με το χώρο που του δίνεται στην οθόνη (display area), που δεν είναι πάντα όλη η οθόνη
- Για ένα terrain, γενικά το display area είναι $\{dx, dy, dw, dh\}$ το οποίο είναι τμήμα του bitmap της οθόνης και ορίζει $vw=dw$, και $vh=dh$



Basic display method (3/9)





Basic display method (4/9)

```
Bitmap dpyBuffer = nullptr;
Point viewPosCached { -1, -1 };
Dim dpyX = 0, dpyY = 0;
void TileTerrainDisplay (TileMap* map, Bitmap dest, const Rect& viewWin, const Rect& displayArea) {
    if (viewPosCached.x != viewWin.x || viewPosCached.y != viewWin.y) {
        auto startCol = DIV_TILE_WIDTH(viewWin.x);
        auto startRow = DIV_TILE_HEIGHT(viewWin.y);
        auto endCol = DIV_TILE_WIDTH(viewWin.x + viewWin.w - 1);
        auto endRow = DIV_TILE_HEIGHT(viewWin.y + viewWin.y - 1);
        dpyX = MOD_TILE_WIDTH(viewWin.x);
        dpyY = MOD_TILE_WIDTH(viewWin.y);
        viewPosCached.x = viewWin.x, viewPosCached.y = viewWin.y;
        for (Dim row = startRow; row <= endRow; ++row)
            for (Dim col = startCol; col <= endCol; ++col)
                PutTile(
                    dpyBuffer,
                    MUL_TILE_WIDTH(col - startCol),
                    MUL_TILE_HEIGHT(row - startRow),
                    tiles,
                    GetTile(map, col, row)
                );
    }
    BitmapBlit(
        dpyBuffer,
        { dpyX, dpyY, viewWin.w, viewWin.h },
        dest,
        { displayArea.x, displayArea.y }
    );
}
```

← *display caching*

■ *View window (4/4)*



Basic display method (5/9)

■ *Scrolling* (1/3)

- Η μετακίνηση του view window πάνω στο terrain ουσιαστικά μετακινεί την «κάμερα» πάνω στο σκηνικό του terrain
- Αυτή η μετακίνηση, που γίνεται σε μικρά διαδοχικά βήματα με αλλαγή απλώς του origin του view window, είναι γνωστή ως scrolling
 - ◆ έτσι τα tiled-based games είναι γνωστά ως **side scrollers**
- Η μετακίνηση γίνεται σε pixels και σχεδόν πάντα προκαλείται από την μετακίνηση του player character

Basic display method (6/9)

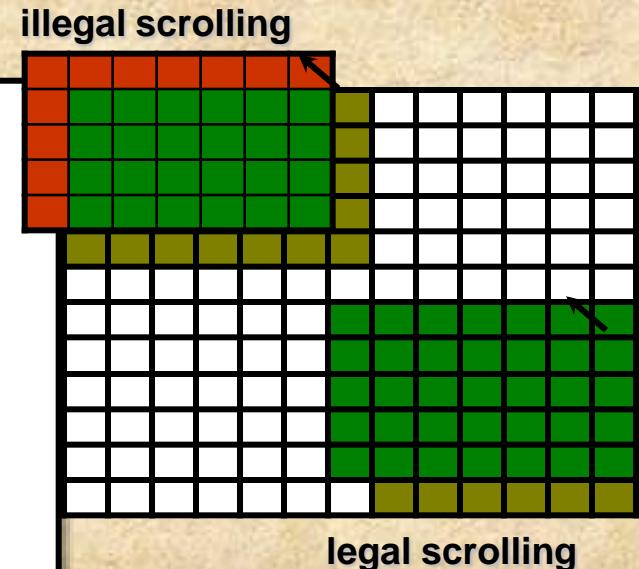
■ Scrolling (2/3)

```

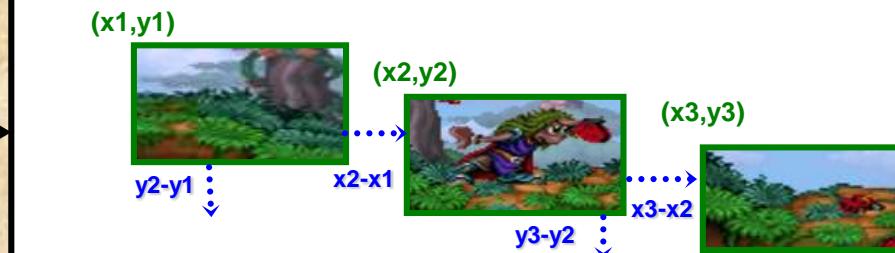
int GetMapPixelWidth (void);
int GetMapPixelHeight (void);

void Scroll (Rect* viewWin, int dx, int dy)
{ viewWin->x += dx; viewWin->y += dy; }
bool CanScrollHoriz (const Rect& viewWin, int dx) {
    return viewWin.x >= -dx &&
           (viewWin.x + viewWin.w + dx) <= GetMapPixelWidth();
}
bool CanScrollVert (const Rect& viewWin, int dy) {
    return viewWin.y >= -dy &&
           (viewWin.y + viewWin.h + dy) <= GetMapPixelHeight();
}

```



Πρέπει να ελέγχουμε ανά πάσα στιγμή ότι το νέο view window, μετά το scrolling, είναι πάντα εντός των ορίων του terrain (μην ξεχνάμε ότι το view window είναι απλώς το πλαίσιο ορατότητας πάνω από το terrain)





Basic display method (7/9)

■ Scrolling (3/3)

```
static void FilterScrollDistance (
    int      viewStartCoord,          // x or y
    int      viewSize,               // w or h
    int*    d,                      // dx or dy
    int      maxMapSize             // w or h
) {
    auto val = *d + viewStartCoord;
    if (val < 0)
        *d = viewStartCoord; // cross low bound
    else
        if (viewSize >= maxMapSize)// fits entirely
            *d = 0;
        else
            if ((val + viewSize) >= maxMapSize) // cross upper bound
                *d = maxMapSize - (viewStartCoord + viewSize);
}
void FilterScroll (const Rect& viewWin, int* dx, int* dy) {
    FilterScrollDistance(
        viewWin.x, viewWin.w, dx, GetMapPixelWidth()
    );
    FilterScrollDistance(
        viewWin.y, viewWin.h, dy, GetMapPixelHeight()
    );
}
```

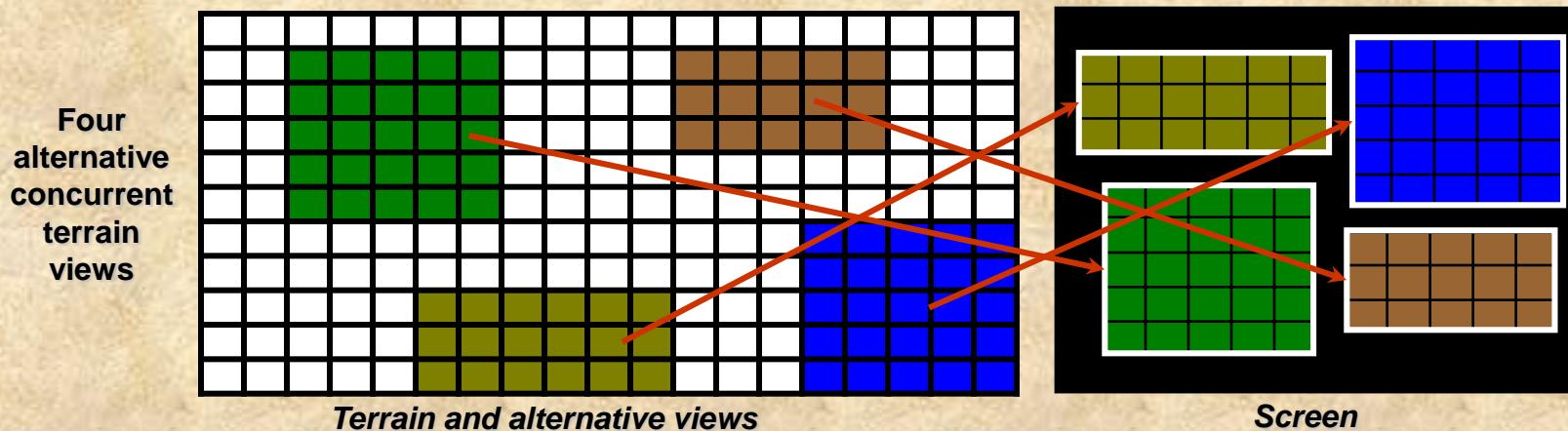
```
void ScrollWithBoundsCheck (
    Rect* viewWin,
    int dx,
    int dy
) {
    FilterScroll(*viewWin, &dx, &dy);
    Scroll(viewWin, dx, dy);
}
```

Αυτό το function
χρησιμοποιούμε
τελικά για scrolling

Basic display method (8/9)

■ *Multiple views and split screens (1/2)*

- Υπάρχουν περιπτώσεις στις οποίες θέλουμε στην οθόνη να εμφανίζονται πολλαπλές εικόνες του terrain, η κάθε μία με διαφορετικό view window
 - ◆ split screens για παιχνίδι δύο παικτών
 - ◆ views από περιοχές του terrain όπου υπάρχει κάποια δράση που ενδιαφέρει τον παίκτη
- Αυτό απαιτεί την ύπαρξη παράλληλα διαφορετικών view windows και αντίστοιχα και διαφορετικών αντίστοιχων display areas





Basic display method (9/9)

■ Multiple views and split screens (2/2)

```
struct ViewData {
    Bitmap     dpyBuffer = nullptr;
    Point      viewPosCached {-1, -1};
    Dim        dpyX = 0, dpyY = 0;
    Rect       viewWin;
    Rect       displayArea;
};

#define MAX_VIEWS 4
ViewData views[MAX_VIEWS];

// refined to accept everything from 'view' parameter
void TileTerrainDisplay (TileMap* map, Bitmap dest, ViewData& view);

void DisplayTerrain (TileMap* map) {
    for (auto i = 0; i < MAX_VIEWS; ++i)
        TileTerrainDisplay(
            map,
            GetBackBuffer(),
            views[i]
        );
}
```

The code defines a `ViewData` struct containing pointers to `Bitmap`, `Point`, `Dim`, `Rect`, and `Rect`. It also includes a macro `#define MAX_VIEWS 4` and a variable `ViewData views[MAX_VIEWS];`. The `DisplayTerrain` function iterates through `MAX_VIEWS` and calls `TileTerrainDisplay` for each view.

Annotations:

- A brace `{}` groups `dpyBuffer`, `viewPosCached`, and `dpyX`, `dpyY` under the label **caching**.
- A brace `{}` groups `viewWin` and `displayArea` under the label **data**.



Περιεχόμενα

- Tile-based games
- Tiles and tile-bitmaps
- Tile-based terrains
- Basic display method
- ***Grid motion control***
- Grid computation



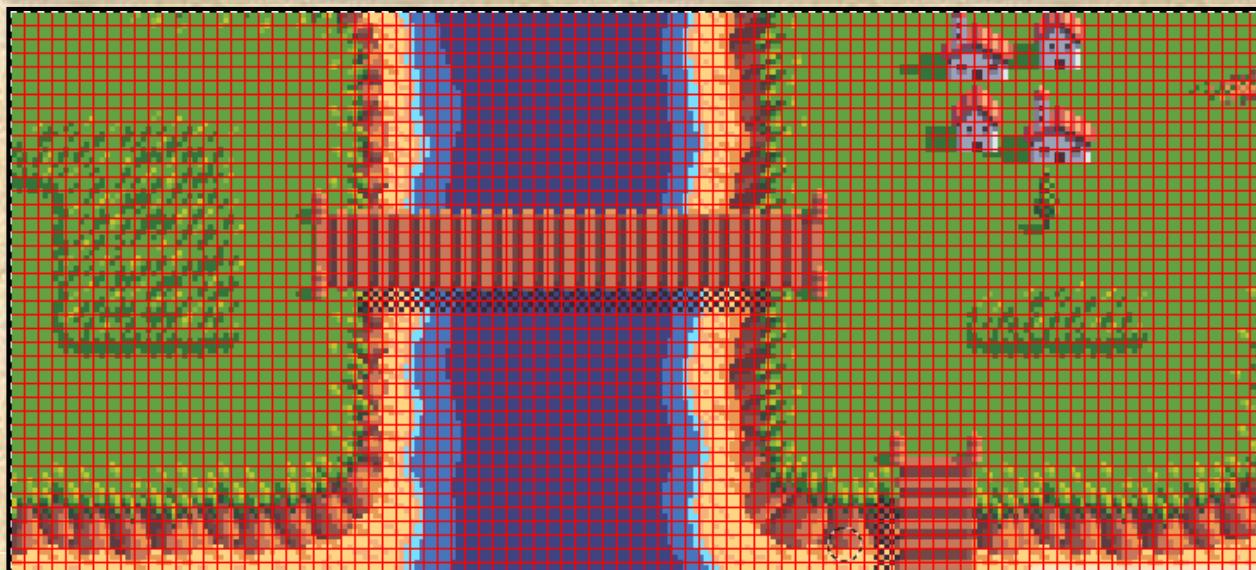
Grid motion control (1/8)

- We assume rectangular areas for moving objects
- Besides denying motion, we compute the maximum allowed offsets:
 - This means the **(dx,dy)** quantities can be altered (*filtered*) by the ***grid motion checker***
- We limit the **(dx,dy)** offsets to grid element size, meaning larger offsets are handled as a sequence of motions
 - Some kind of an outer motion loop, whatever the responsible subsystem for this is

Grid motion control (2/8)



Original tile map,
visual terrain
structure, using
16x16 tiles



The respective
grid map, using
8x8 grid
elements, thus
have 4 elements
(2x2 resolution)
per tile - usually
a 4x4 resolution
suffices



Grid motion control (3/8)

```
#define GRID_ELEMENT_WIDTH      4
#define GRID_ELEMENT_HEIGHT      4

#if TILE_WIDTH % GRID_ELEMENT_WIDTH != 0
#error "TILE_WIDTH % GRID_ELEMENT_WIDTH must be zero!"
#endif

#if TILE_HEIGHT % GRID_ELEMENT_HEIGHT != 0
#error "TILE_HEIGHT % GRID_ELEMENT_HEIGHT must be zero!"
#endif

#define   GRID_BLOCK_COLUMNS      (TILE_WIDTH / GRID_ELEMENT_WIDTH)
#define   GRID_BLOCK_ROWS          (TILE_HEIGHT / GRID_ELEMENT_HEIGHT)
#define   GRID_ELEMENTS_PER_TILE   (GRID_BLOCK_ROWS * GRID_BLOCK_COLUMNS)
#define   GRID_MAX_HEIGHT          (MAX_HEIGHT * GRID_BLOCK_ROWS)
#define   GRID_MAX_WIDTH           (MAX_WIDTH * GRID_BLOCK_COLUMNS)

using GridIndex = byte;
typedef GridIndex GridMap[GRID_MAX_WIDTH][GRID_MAX_HEIGHT];
static GridMap grid; // example of a global static grid

void SetGridTile (GridMap* m, Dim col, Dim row, GridIndex index)
{ (*m)[row][col] = index; }

GridIndex GetGridTile (const GridMap* m, Dim col, Dim row)
{ return (*m)[row][col]; }
```

Definitions required for a 4x4 grid per tile



Grid motion control (4/8)

```
#define GRID_THIN_AIR_MASK      0x0000      // element is ignored
#define GRID_LEFT_SOLID_MASK     0x0001      // bit 0
#define GRID_RIGHT_SOLID_MASK    0x0002      // bit 1
#define GRID_TOP_SOLID_MASK      0x0004      // bit 2
#define GRID_BOTTOM_SOLID_MASK   0x0008      // bit 3
#define GRID_GROUND_MASK         0x0010      // bit 4, keep objects top / bottom (gravity)
#define GRID_FLOATING_MASK       0x0020      // bit 5, keep objects anywhere inside (gravity)

#define GRID_EMPTY_TILE GRID_THIN_AIR_MASK
#define GRID_SOLID_TILE \
(GRID_LEFT_SOLID_MASK | GRID_RIGHT_SOLID_MASK | GRID_TOP_SOLID_MASK | GRID_BOTTOM_SOLID_MASK)

void SetSolidGridTile (GridMap* m, Dim col, Dim row)
{ SetGridTile(m, col, row, GRID_SOLID_TILE); }

void SetEmptyGridTile (GridMap* m, Dim col, Dim row)
{ SetGridTile(m, col, row, GRID_EMPTY_TILE); }

void SetGridTileFlags (GridMap* m, Dim col, Dim row, GridIndex flags)
{ SetGridTile(m, col, row, flags); }

void SetGridTileTopSolidOnly (GridMap* m, Dim col, Dim row)
{ SetGridTileFlags(m, row, col, GRID_TOP_SOLID_MASK); }

bool CanPassGridTile (GridMap* m, Dim col, Dim row, GridIndex flags) // i.e. checks if flags set
{ return GetGridTile(m, row, col) & flags != 0; }
```



Grid motion control (5/8)

```
#define MAX_PIXEL_WIDTH          MUL_TILE_WIDTH(MAX_WIDTH)
#define MAX_PIXEL_HEIGHT          MUL_TILE_HEIGHT(MAX_HEIGHT)
#define DIV_GRID_ELEMENT_WIDTH(i)  ((i)>>2)
#define DIV_GRID_ELEMENT_HEIGHT(i) ((i)>>2)
#define MUL_GRID_ELEMENT_WIDTH(i)  ((i)<<2)
#define MUL_GRID_ELEMENT_HEIGHT(i) ((i)<<2)

void FilterGridMotion (GridMap* m, const Rect& r, int* dx, int* dy) {
    assert(
        abs(*dx) <= GRID_ELEMENT_WIDTH && abs(*dy) <= GRID_ELEMENT_HEIGHT
    );

    // try horizontal move
    if (*dx < 0)
        FilterGridMotionLeft(m, r, dx);
    else
        if (*dx > 0)
            FilterGridMotionRight(m, r, dx);

    // try vertical move
    if (*dy < 0)
        FilterGridMotionUp(m, r, dy);
    else
        if (*dy > 0)
            FilterGridMotionDown(m, r, dy);
}
```



Grid motion control (6/8)

```
void FilterGridMotionLeft (GridMap* m, const Rect& r, int* dx) {  
    auto x1_next = r.x + *dx;  
    if (x1_next < 0)  
        *dx = -r.x;  
    else {  
        auto newCol  = DIV_GRID_ELEMENT_WIDTH(x1_next);  
        auto currCol = DIV_GRID_ELEMENT_WIDTH(r.x);  
  
        if (newCol != currCol) {  
  
            assert(newCol + 1 == currCol); // we really move left  
  
            auto startRow      = DIV_GRID_ELEMENT_HEIGHT(r.y);  
            auto endRow        = DIV_GRID_ELEMENT_HEIGHT(r.y + r.h - 1),  
  
            for (auto row = startRow; row <= endRow; ++row)  
                if (!CanPassGridTile(m, newCol, row, GRID_RIGHT_SOLID_MASK)) {  
                    *dx = MUL_GRID_ELEMENT_WIDTH(currCol) - r.x; ←  
                    break;  
                }  
        }  
    }  
}
```

crossing side

attach right



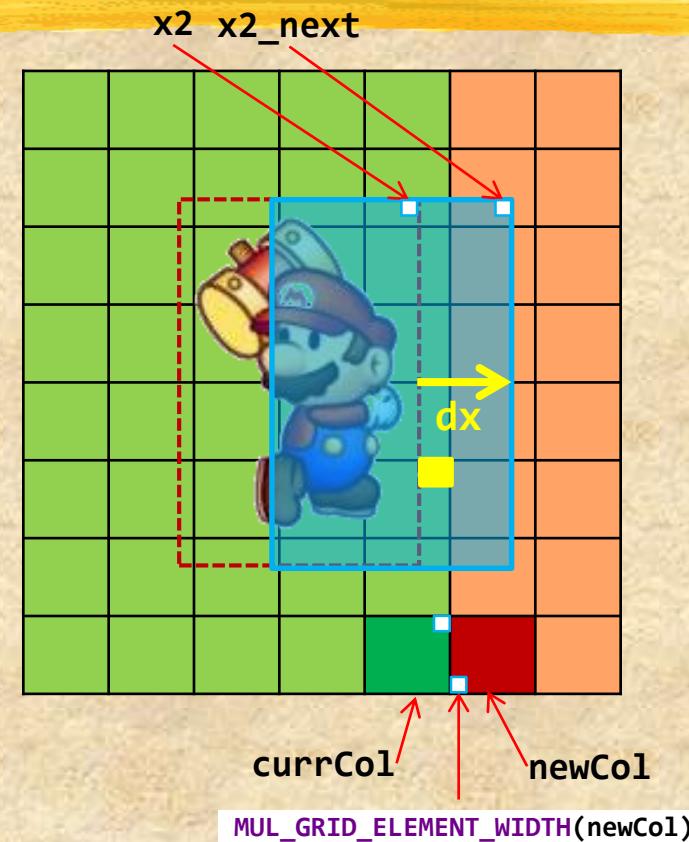
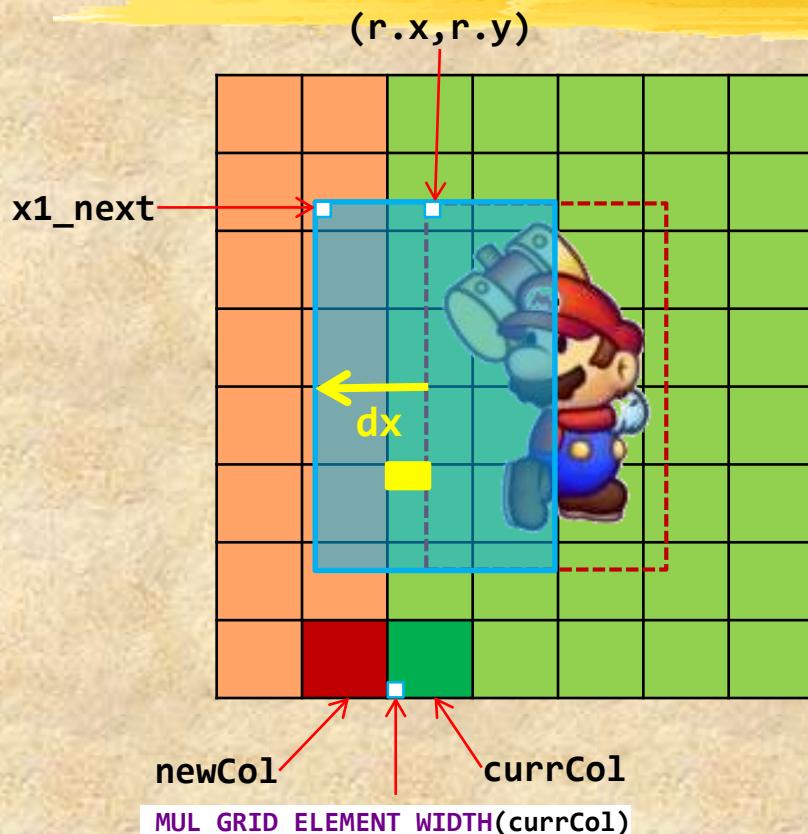
Grid motion control (7/8)

```
void FilterGridMotionRight (GridMap* m, const Rect& r, int* dx) {  
    auto x2 = r.x + r.w - 1;  
    auto x2_next = x2 + *dx;  
    if (x2_next >= MAX_PIXEL_WIDTH)  
        *dx = (MAX_PIXEL_WIDTH - 1) - x2;  
    else {  
        auto newCol = DIV_GRID_ELEMENT_WIDTH(x2_next);  
        auto currCol = DIV_GRID_ELEMENT_WIDTH(x2);  
  
        if (newCol != currCol) {  
  
            assert(newCol - 1 == currCol); // we really move right  
  
            auto startRow = DIV_GRID_ELEMENT_HEIGHT(r.y);  
            auto endRow = DIV_GRID_ELEMENT_HEIGHT(r.y + r.h - 1);  
  
            for (auto row = startRow; row <= endRow; ++row)  
                if (!CanPassGridTile(m, newCol, row, GRID_LEFT_SOLID_MASK)) {  
                    *dx = (MUL_GRID_ELEMENT_WIDTH(newCol) - 1) - x2; ←  
                    break;  
                }  
        }  
    }  
}
```

crossing side

attach left

Grid motion control (8/8)





Περιεχόμενα

- Tile-based games
- Tiles and tile-bitmaps
- Tile-based terrains
- Basic display method
- Grid motion control
- ***Grid computation***



Grid computation (1/8)

- When grid elements subdivide tiles we need a custom **GridMap** structure and respective multiplication / division macros
- We can produce the grid map automatically (*low res*) and then we can fine tune per case (*editor*)
- Firstly we characterize visual tile indices as ***solid*** or ***empty***
- Then we iterate and per tile we produce the respective grid block as follows →

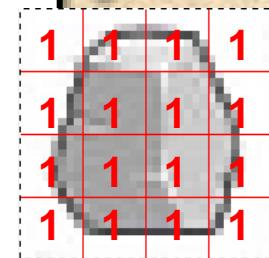


Grid computation (2/8)

```
extern bool IsTileIndexAssumedEmpty (Index index);

void ComputeTileGridBlocks1 (const TileMap* map, GridIndex* grid) {
    for (auto row = 0; row < MAX_HEIGHT; ++row)
        for (auto col = 0; col < MAX_WIDTH; ++col) {
            memset(
                grid,
                IsTileIndexAssumedEmpty(GetTile(map, col, row)) ?
                    GRID_EMPTY_TILE :
                    GRID_SOLID_TILE,
                GRID_ELEMENTS_PER_TILE
            );
            grid += GRID_ELEMENTS_PER_TILE;
        }
}
```

This is treated as full solid



This approach is naïve, and does not exploit the grid resolution (a block of grid elements represents precisely a tile)

But may work for all games where the grid element is of the same size as tiles (usual)



Grid computation (3/8)

```
class TileColorsHolder final {
    private:
        std::set<Index> indices;
        std::set<Color> colors;

    public:
        void Insert (Bitmap bmp, Index index) {
            if (indices.find(index) == indices.end()) {
                indices.insert(index);
                BitmapAccessPixels(
                    bmp,
                    [this](PixelMemory mem)
                    { colors.insert(GetPixel32(mem)); }
                );
            }
        }
        bool In (Color c) const
        { return colors.find(c) != colors.end(); }
};

// keeps colors that are assumed to be empty
static TileColorsHolder emptyTileColors;
bool IsTileColorEmpty (Color c)
{ return emptyTileColors.In(c); } // return false to disable
```

When we assume some tiles as empty, it means they include non-transparent colors

Then, it is probable that these colors are used within other tiles to also represent empty space

What we do is to collect all such colors and assume also that they belong to empty colors

If this gives good results in terms of grid computation we keep it, else we disable it.



Grid computation (4/8)

```
void ComputeTileGridBlocks2 (
    const TileMap*           map,
    GridIndex*                grid,
    Bitmap                   tileSet,
    Color                     transColor,
    byte                      solidThreshold
) {
    auto tileElem = BitmapCreate(TILE_WIDTH, TILE_HEIGHT);
    auto gridElem = BitmapCreate(GRID_ELEMENT_WIDTH, GRID_ELEMENT_HEIGHT);
    for (auto row = 0; row < MAX_HEIGHT; ++row)
        for (auto col = 0; col < MAX_WIDTH; ++col) {
            auto index = GetTile(map, col, row);
            PutTile(tileElem, 0, 0, tileSet, index);
            if (IsTileIndexAssumedEmpty(index)) {
                emptyTileColors.Insert(tileElem, index); // assume tile colors to be empty
                memset(grid, GRID_EMPTY_TILE, GRID_ELEMENTS_PER_TILE);
                grid += GRID_ELEMENTS_PER_TILE;
            }
            else
                ComputeGridBlock("// auto increments grid as T*&
                    grid, index, tileElem, gridElem,
                    tileSet, transColor, solidThreshold
                );
        }
    BitmapDestroy(tileElem);
    BitmapDestroy(gridElem);
}
```



Grid computation (5/8)

```
void ComputeGridBlock (
    GridIndex*& grid,
    Index index,
    Bitmap tileElem,
    Bitmap gridElem,
    Bitmap tileSet,
    Color transColor,
    byte solidThreshold
) {
    for (auto i = 0; i < GRID_ELEMENTS_PER_TILE; ++i) {
        auto x = i % GRID_BLOCK_ROWS;
        auto y = i / GRID_BLOCK_ROWS;
        BitmapBlit(
            tileElem,
            { x * GRID_ELEMENT_WIDTH, y * GRID_ELEMENT_HEIGHT, GRID_ELEMENT_WIDTH, GRID_ELEMENT_HEIGHT },
            gridElem,
            { 0, 0 }
        );
        auto isEmpty = ComputeIsEmpty(gridElem, transColor, solidThreshold);
        *grid++ = isEmpty ? GRID_EMPTY_TILE : GRID_SOLID_TILE;
    }
}
```



Grid computation (6/8)

```
Color GetPixel32 (PixelMemory mem) {  
    RGBA c;  
    ReadPixelColor32(mem, &c, &c.a);  
    return MakeColor32(c.r, c.g, c.b, c.a);  
}  
  
bool ComputeIsGridIndexEmpty (  
    Bitmap gridElement,  
    Color transColor,  
    byte solidThreshold  
) {  
    auto n = 0;  
    BitmapAccessPixels(  
        gridElement,  
        [transColor, &n](PixelMemory mem) {  
            auto c = GetPixel32(mem);  
            if (c != transColor && !IsTileColorEmpty(c))  
                ++n;  
        }  
    );  
    return n <= solidThreshold;  
}
```

This approach is more detailed than tiles and assumes visual structure reflects solidity

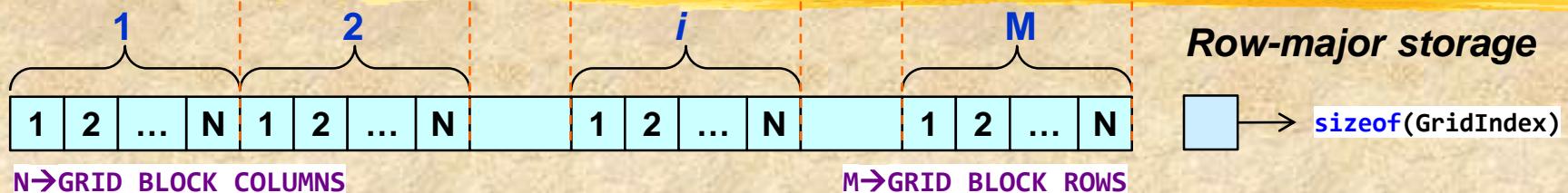
Using the threshold, we can tune the grid resolution to be more rigid or more loose depending on the game

0	1	1	0
0	1	1	1
1	1	1	1
0	1	1	0

0	1	1	0
0	1	1	0
0	1	1	0
0	1	1	0



Grid computation (7/8)



```
#define GRID_BLOCK_SIZEOF \
    (GRID_ELEMENTS_PER_TILE * sizeof(GridIndex))

GridIndex* GetGridTileBlock (Dim colTile, Dim rowTile, Dim tileCols, GridIndex* grid) {
    return grid + (rowTile * tileCols + colTile) * GRID_BLOCK_SIZEOF;
}

void SetGridTileBlock (Dim colTile, Dim rowTile, Dim tileCols, GridIndex* grid, GridIndex flags) {
    memset(
        GetGridTileBlock(colTile, rowTile, tileCols, grid),
        flags,
        GRID_BLOCK_SIZEOF
    );
}

#define SetGridTileBlockEmpty(col, row, cols, grid) \
    SetGridTileBlock(col, row, cols, grid, GRID_EMPTY_TILE)

#define SetGridTileBlockSolid(col, row, cols, grid) \
    SetGridTileBlock(col, row, cols, grid, GRID_SOLID_TILE)
```



Grid computation (8/8)

```
// use this to render grid (toggle on / off), used only for development time testing -  
// a tile grid block is consecutive GRID_BLOCK_ROWS x GRID_BLOCK_COLUMNS block of grid indices  
  
template <typename Tfunc>  
void DisplayGrid (Bitmap dest, const Rect& viewWin, GridIndex* grid, Dim tileCols, const Tfunc& display_f) {  
  
    auto startCol      = DIV_TILE_WIDTH(viewWin.x);  
    auto startRow      = DIV_TILE_HEIGHT(viewWin.y);  
    auto endCol        = DIV_TILE_WIDTH(viewWin.x + viewWin.w - 1);  
    auto endRow        = DIV_TILE_HEIGHT(viewWin.y + viewWin.h - 1);  
  
    for (Dim rowTile = startRow; rowTile <= endRow; ++rowTile)  
        for (Dim colTile = startCol; colTile <= endCol; ++colTile) {  
  
            auto sx = MUL_TILE_WIDTH(colTile - startCol);  
            auto sy = MUL_TILE_HEIGHT(rowTile - startRow);  
            auto* gridBlock = GetGridTileBlock(rowTile, colTile, tileCols, grid);  
  
            for (auto rowElem = 0; rowElem < GRID_BLOCK_ROWS; ++rowElem)  
                for (auto colElem = 0; colElem < GRID_BLOCK_COLUMNS; ++colElem)  
                    if (*gridBlock++ & GRID_SOLID_TILE) {  
                        auto x = sx + MUL_GRID_ELEMENT_WIDTH(colElem);  
                        auto y = sy + MUL_GRID_ELEMENT_HEIGHT(rowElem);  
                        auto w = GRID_ELEMENT_WIDTH - 1;  
                        auto h = GRID_ELEMENT_HEIGHT - 1;  
                        display_f(dest, x, y, w, h);  
                    }  
            }  
        }  
    }  
}
```