DM 2231
GAMES DEVELOPMENT TECHNIQUES

2015/16 SEMESTER 1

2D Game Programming #2

MODULE SCHEDULE

Week	Dates	Topic	Remarks	Public Holidays
1	20-Apr-2015 to 24-Apr-2015	Module Introduction / 3D Game Programming	Issue Assignment 1	
2	27-Apr-2015 to 1-May-2015	Game Application		1 May. Labour Day
3	4-May-2015 to 8-May-2015	User Input		
4	11-May-2015 to 15-May-2015	Camera and GUI #1		
5	18-May-2015 to 22-May-2015	Camera and GUI #2		
6	25-May-2015 to 29-May-2015	Basic Game Physics	Submit Assignment 1	
7	1-Jun-2015 to 5-Jun-2015	Implementing Game Audio (E-learning)		1 Jun. Vesak Day
8	8-Jun-2015 to 12-Jun-2015	Mid-Sem Break		
9	15-Jun-2015 to 19-Jun-2015	Mid-Sem Break		
10	22-Jun-2015 to 26-Jun-2015	2D Game Programming #1	Issue Assignment 2	
11	29-Jun-2015 to 3-Jul-2015	2D Game Programming #2		
12	6-Jul-2015 to 10-Jul-2015	2D Game Programming #3		
13	13-Jul-2015 to 17-Jul-2015	Game Data		17 Jul. Hari Raya Puasa
14	20-Jul-2015 to 24-Jul-2015	Design Pattern #1		
15	27-Jul-2015 to 31-Jul-2015	Design Pattern #2		
16	3-Aug-2015 to 7-Aug-2015	Basic Artificial Intelligence (E-learning)		7 Aug. SG50 Public Holiday
17	10-Aug-2015 to 14-Aug-2015	Good Programming Practices	Submit Assignment 2	10 Aug. National Day

RECAP ON LAST WEEK’S LECTURE

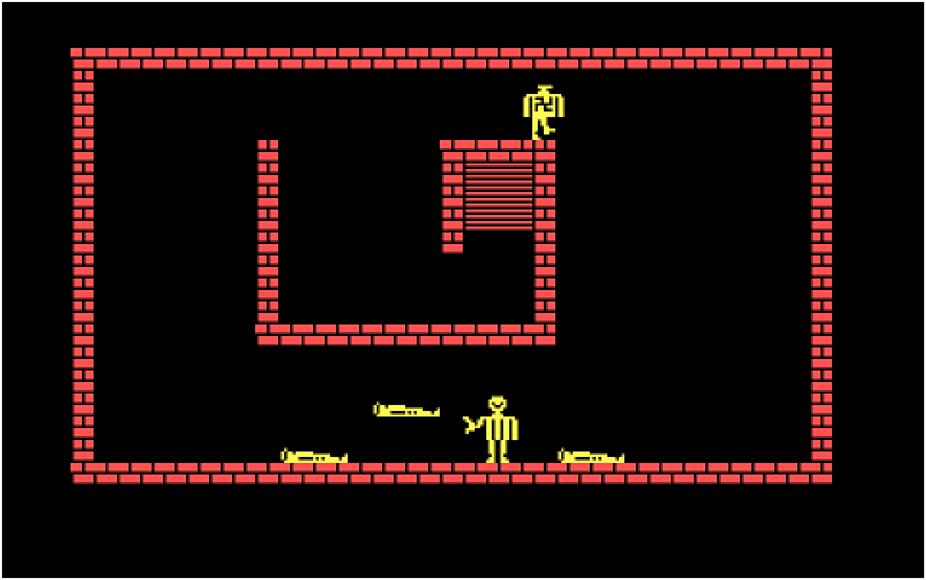
- 2D Game Programming #1
  - 2D Game Basics
  - Data Structures for 2D Games
  - Mapping Matrices
  - Tile Tables
  - 2D Game Algorithms
  - Screen-based Games

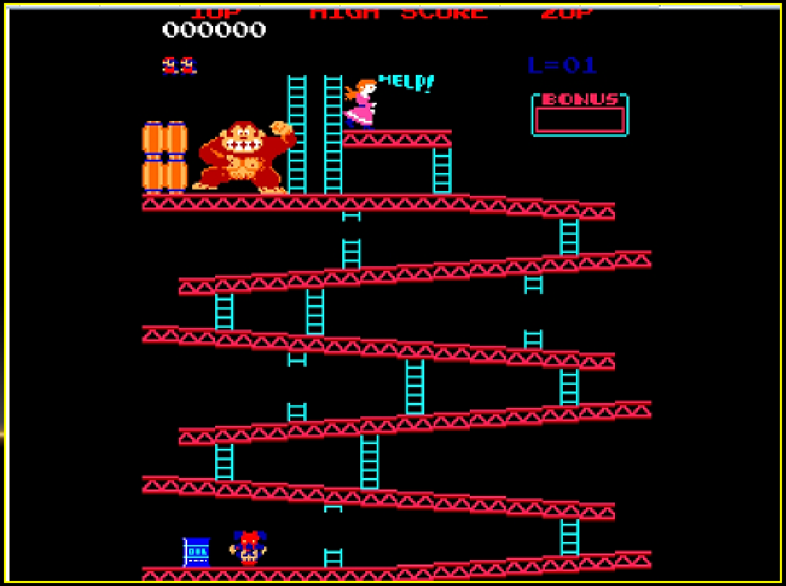
TABLE OF CONTENT

- 2D Game Programming #2
  - Screen-based Games
  - Background Scrolling

2D GAME ALGORITHMS: SCREEN-BASED GAMES

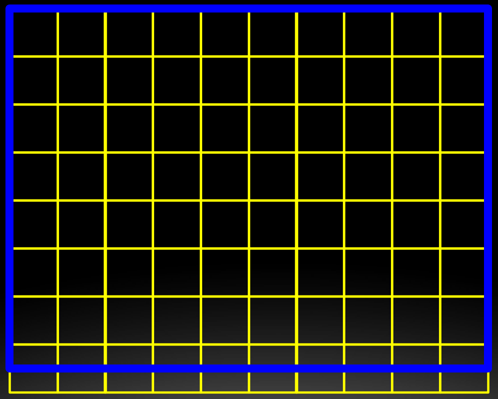
- Simplest mapped game
  - the player confronts a series of screens.
  - When he exits one screen, the graphics are substituted by those in the next screen, and so forth. This means that the player has progressed to the next level.
  - No continuity or transition between screens.
    - Example is Castle Wolfenstein 2D, Donkey Kong.





2D GAME ALGORITHMS: SCREEN-BASED GAMES

- In these games, each screen is represented using a different mapping matrix, which represents the screen layout of the different elements.
  - So, for a 320x240 screen using 32x32 tiles, we would store the screen in a 10x8 matrix of bytes.
  - Notice that 240 does not allow for an exact number of tiles to be rendered vertically onscreen (we can fit 7.5 tiles).
  - So, we take the integer excess to ensure the whole screen is mapped.



2D GAME ALGORITHMS: SCREEN-BASED GAMES

- We had seen this in Week 8 lecture.
  - Use a 3D array matrix indexed by room identifier, x value and y value.
  - In this way, a single data structure can hold the whole game map.
  - To use this approach, we would need a line such as this in our code:

```
int tileid=mapping_matrix [roomid][yi][xi];
```

- How about 2D games? Do you use 3D array matrix?

2D GAME ALGORITHMS: SCREEN-BASED GAMES

- We had seen something similar previously in Week 8 lecture:

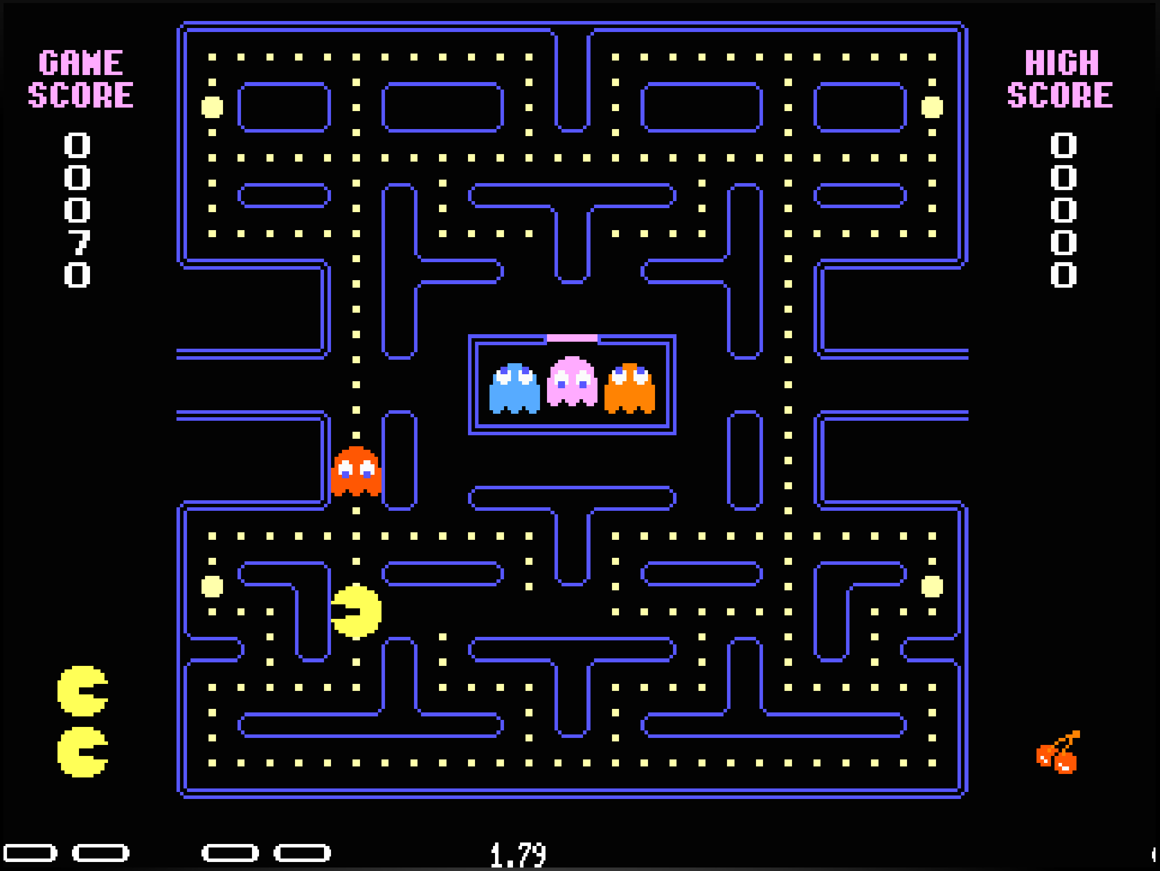
```
#define tile_wide 32
#define tile_high 32
#define screen_wide 320
#define screen_high 240

int roomid = 0;
int xtiles=screen_wide/tile_wide;
int ytiles=screen_high/tile_high;
for (yi=0;yi<ytiles;yi++)
{
    for (xi=0;xi<xtiles;xi++)
    {
          int screex=xi*tile_wide;
          int screey=yi*tile_high;
          int tileid=mapping_matrix [roomid][yi][xi];
          DisplayTile(tile_table[tileid],screenx,screeny);
    }
}
```

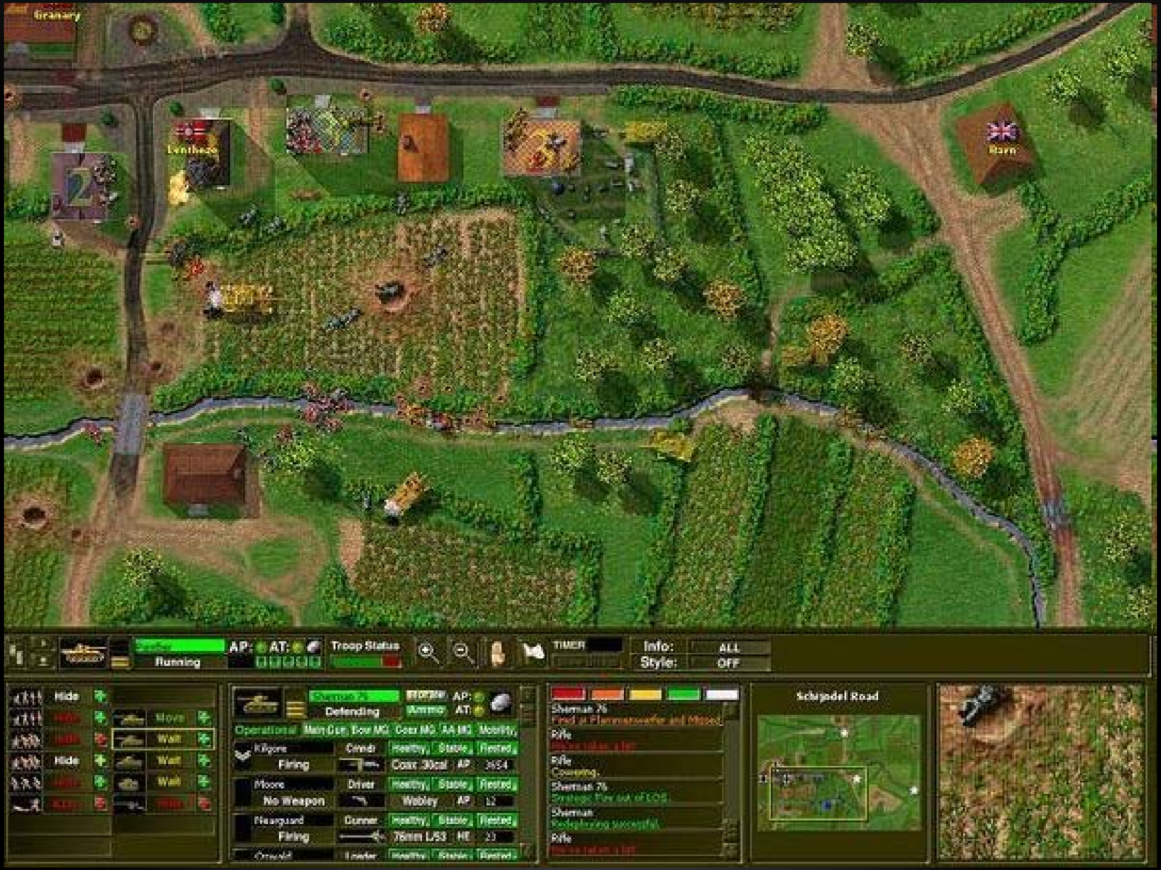
SCREEN-BASED GAMES

- Screen-based Games use the screen as a single room.
  - Also called room-based game
  - Each level is played in one screen
  - When the avatar reaches a certain location, he may go to
    - another room (a.k.a. screen), or
    - pop up elsewhere in the room.
    - There is no scrolling of the room/screen.
- Examples:
  - Pac-man
  - Close-combat
  - Gauntlet

SCREEN-BASED GAME: PAC-MAN

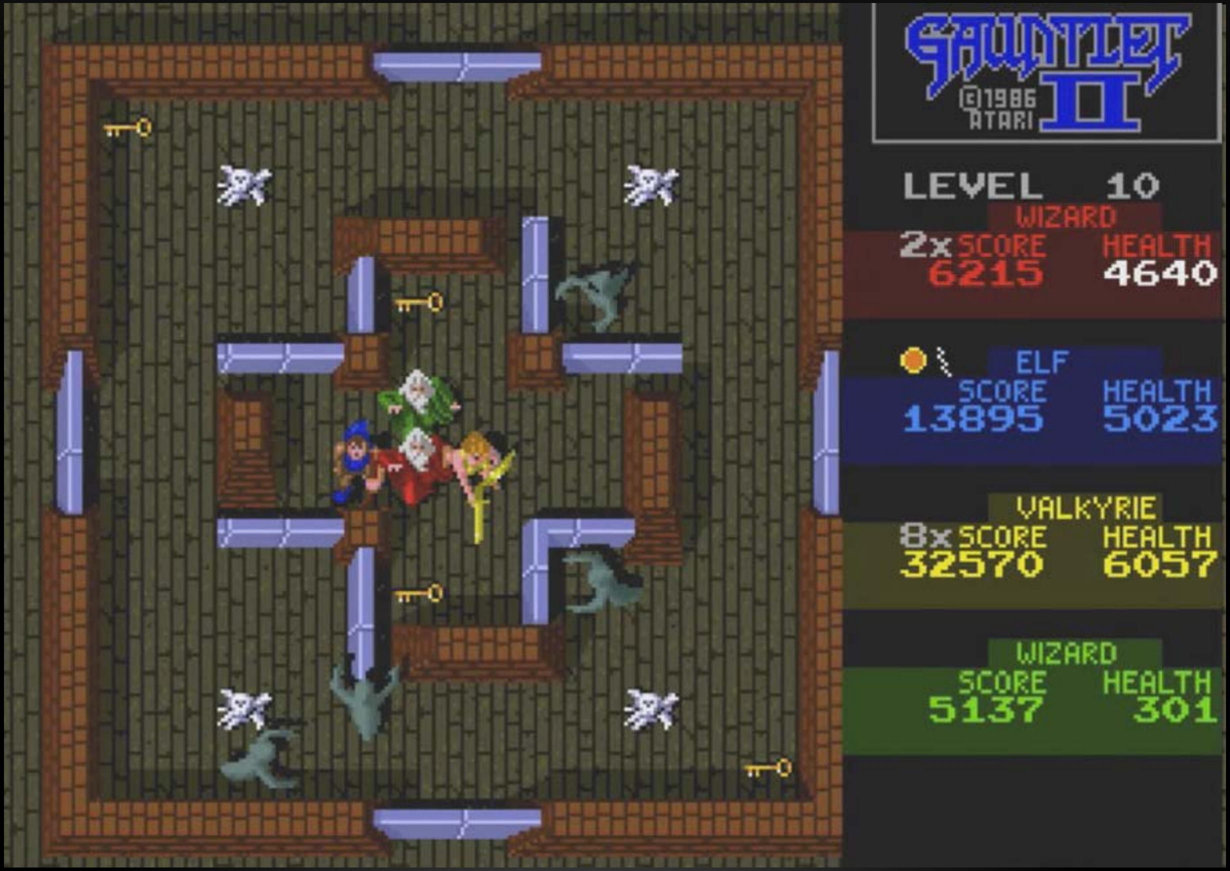


SCREEN-BASED GAME: CLOSE COMBAT



Sherman 76
Fired at Flammenwerfer and Missed
Rifle
Rifle
Covering.
Sherman 76
Strategic Fire out of LOS
Sherman
Redeploying successful.
Rifle

SCREEN-BASED GAME: GAUNTLET



SCREEN-BASED GAMES

- One screen may not be enough to show the entire level
  - Example: RPG games have stories and items to collect, so one screen is not enough to show all.
- Can we show that over a few screens?
  - Yes, but it may affect gameplay if the player has to jump to another screen and back to complete one level.

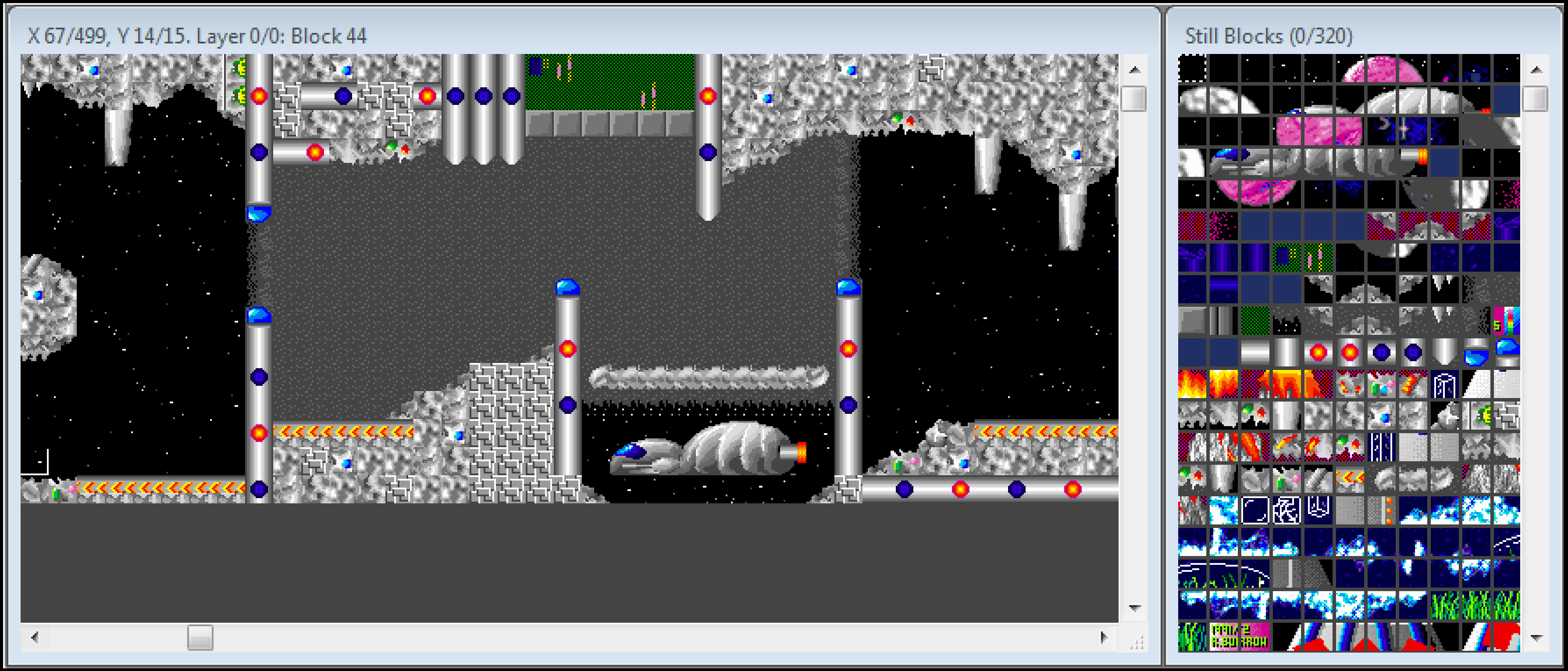
SCROLLING GAMEWORLDS

- Many games have a gameworld larger than the visible area on screen
  - Permits level design
- Level design
  - Pacing of challenge
  - Creation of environment where gameplay takes place
- Usually convenient to use a level design tool
  - Time consuming to write one of these
- One approach
  - Use tiles to construct gameworld

Make your own Editor!

TILE-BASED GAMEWORLD

- A small number of tiles
- Combine to create game levels

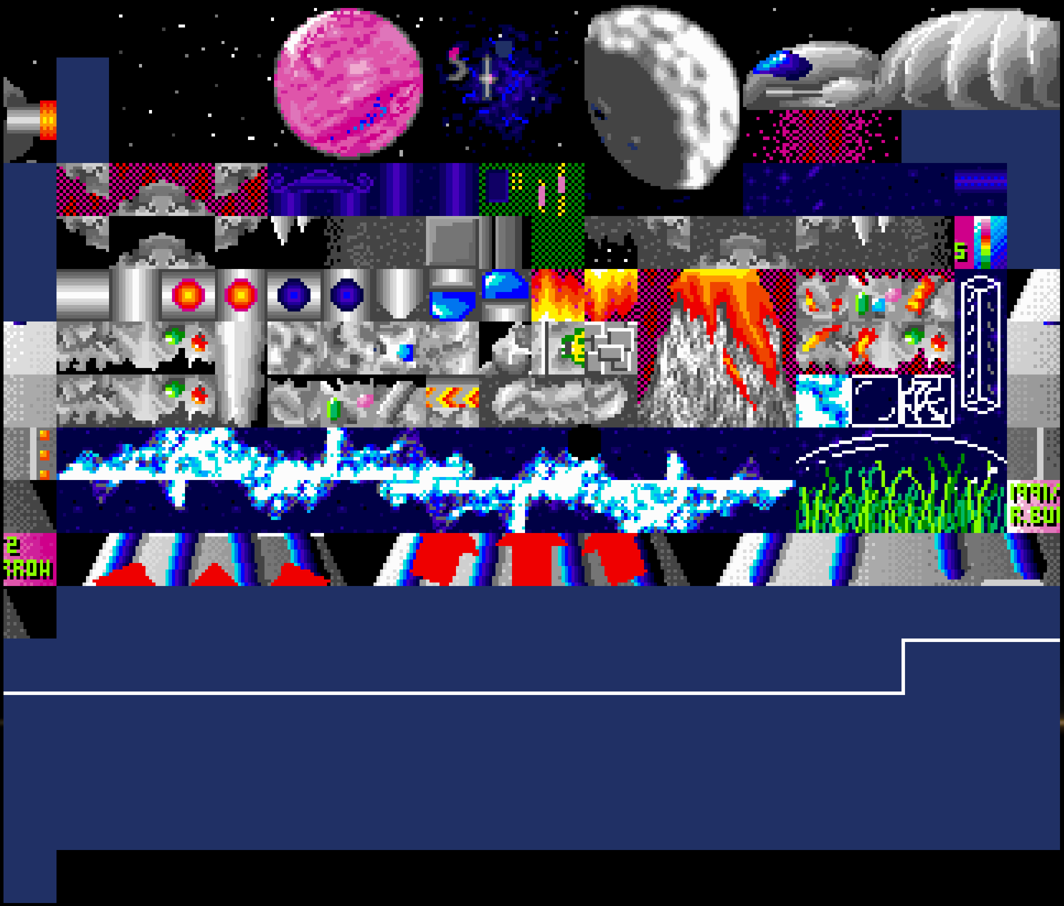


REPRESENTING TILE LEVELS

- A bitmap image represents the tiles

0, 1, 2, 3, … … 18, 19

0
20
40
…



REPRESENTING TILE LEVELS

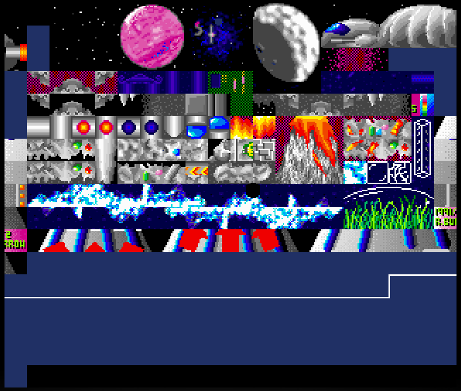
- A two dimensional array
  - Represents an entire game level
  - worldInTiles[y, x] contains the number of a tile
  - Look up tile in bitmap array
- A three dimensional array
  - Represents an entire game level
  - mapping_matrix[roomid][y][x] contains the number of a tile
  - Look up tile in bitmap array

REPRESENTING TILE LEVELS

- Can use a tile editor to create a level
  - Mappy editor
  - Demonstration of mappy

```
{ {5, 6, 7},
  { 25, 26, 27},
  { 45, 46, 47} }
```

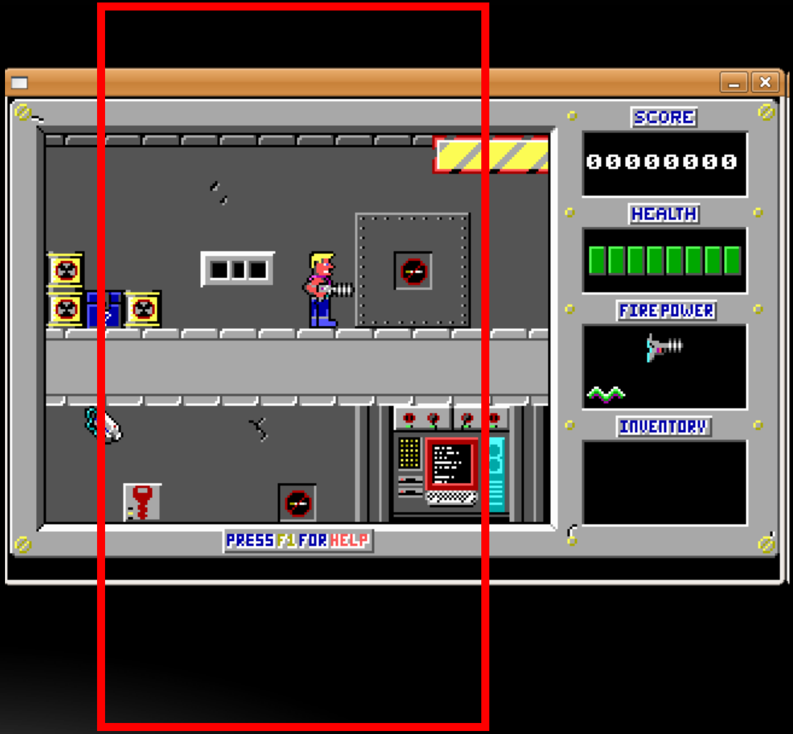
+



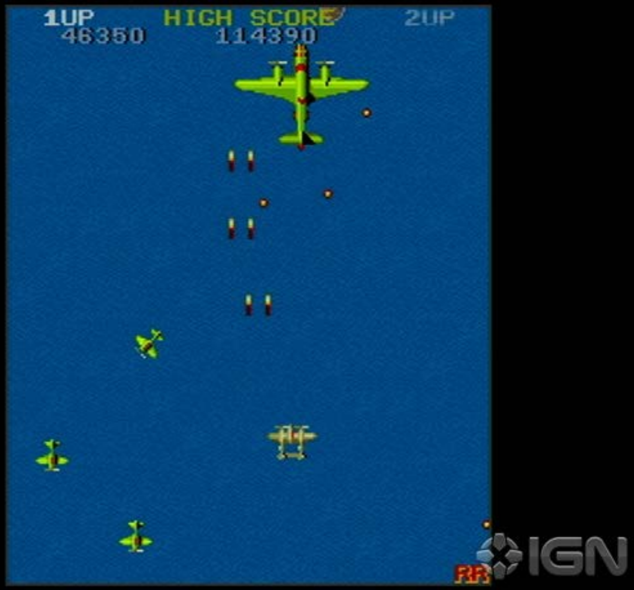
→

BACKGROUND SCROLLING

- The Avatar moves within an area without the background scrolling.
- Once the avatar tries to move out of the area, the background will scroll to show another part of the game level.



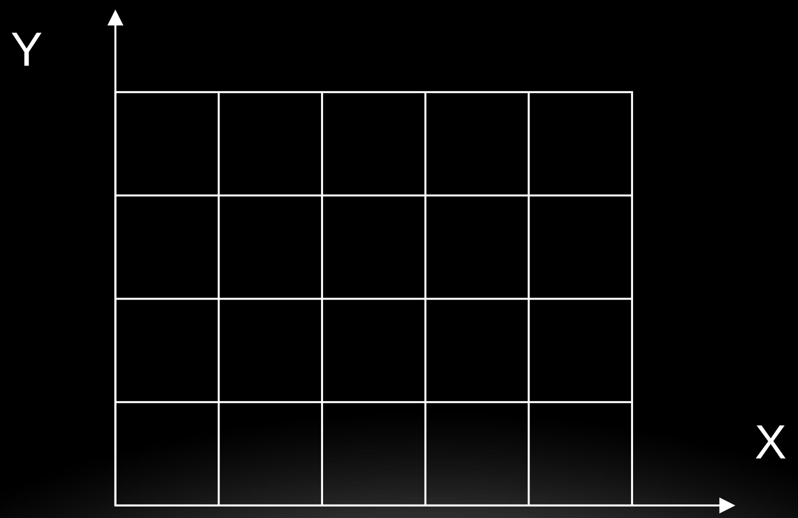
BACKGROUND SCROLLING



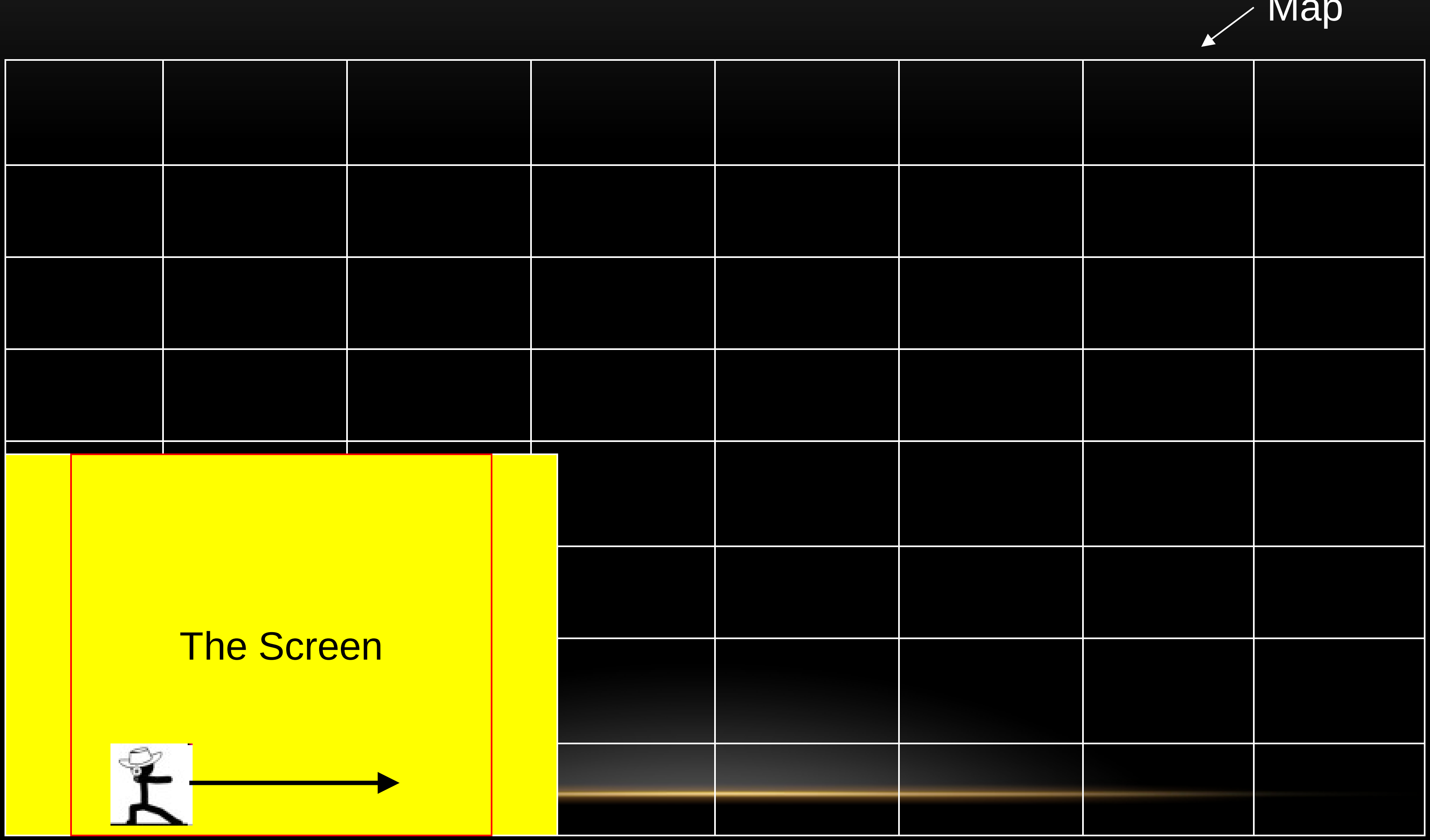
- Not all background scrolling games allow the avatar to dictate the movement of the background
- Capcom’s 1942 has a background scrolling which is controlled by the game

BACKGROUND SCROLLING: THE MAP

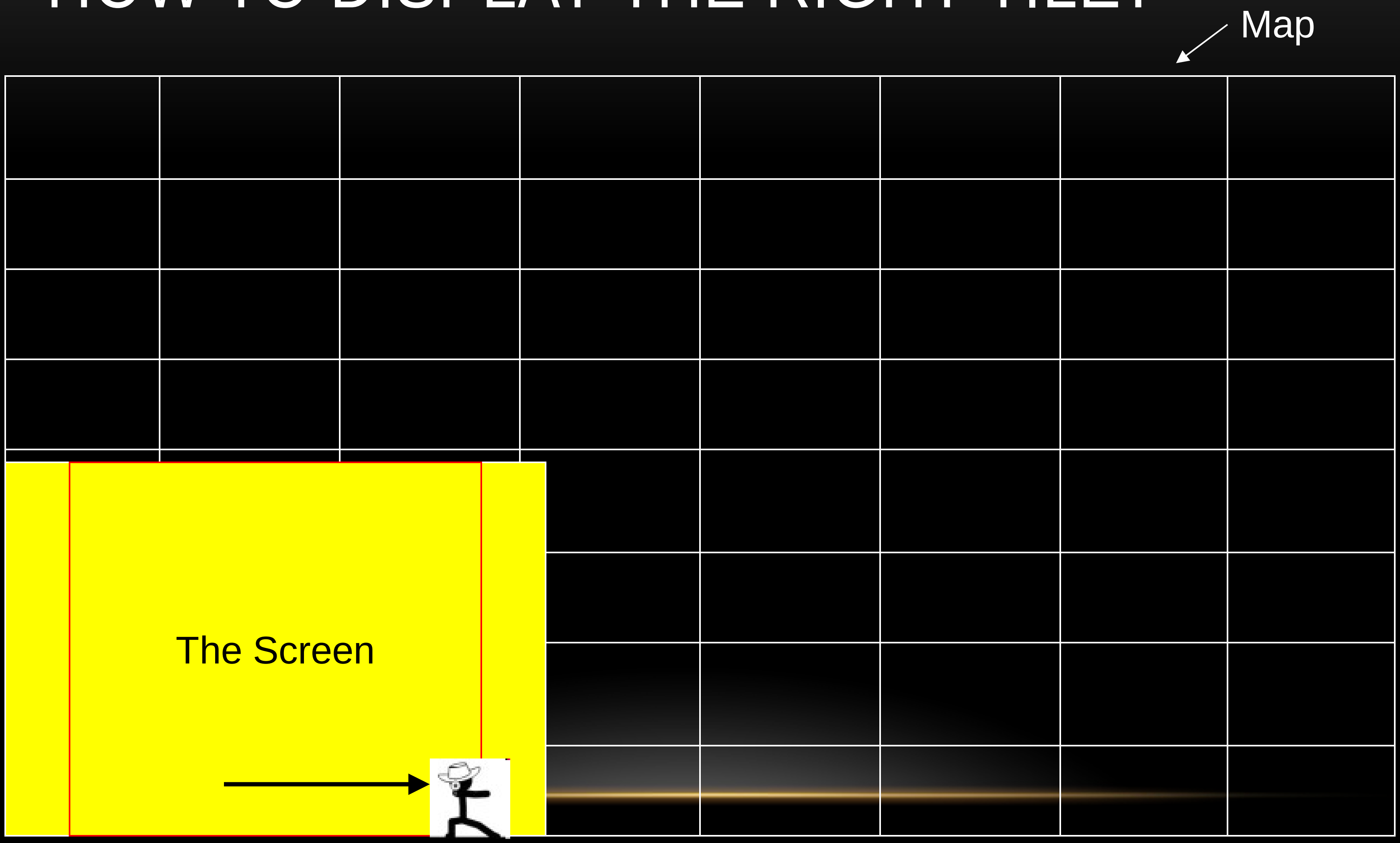
- We use a PCX file to store the map.
- Think of an image file which is of X by Y pixels.
  - Every pixel has a number
  - This number is the index number of the tile to display



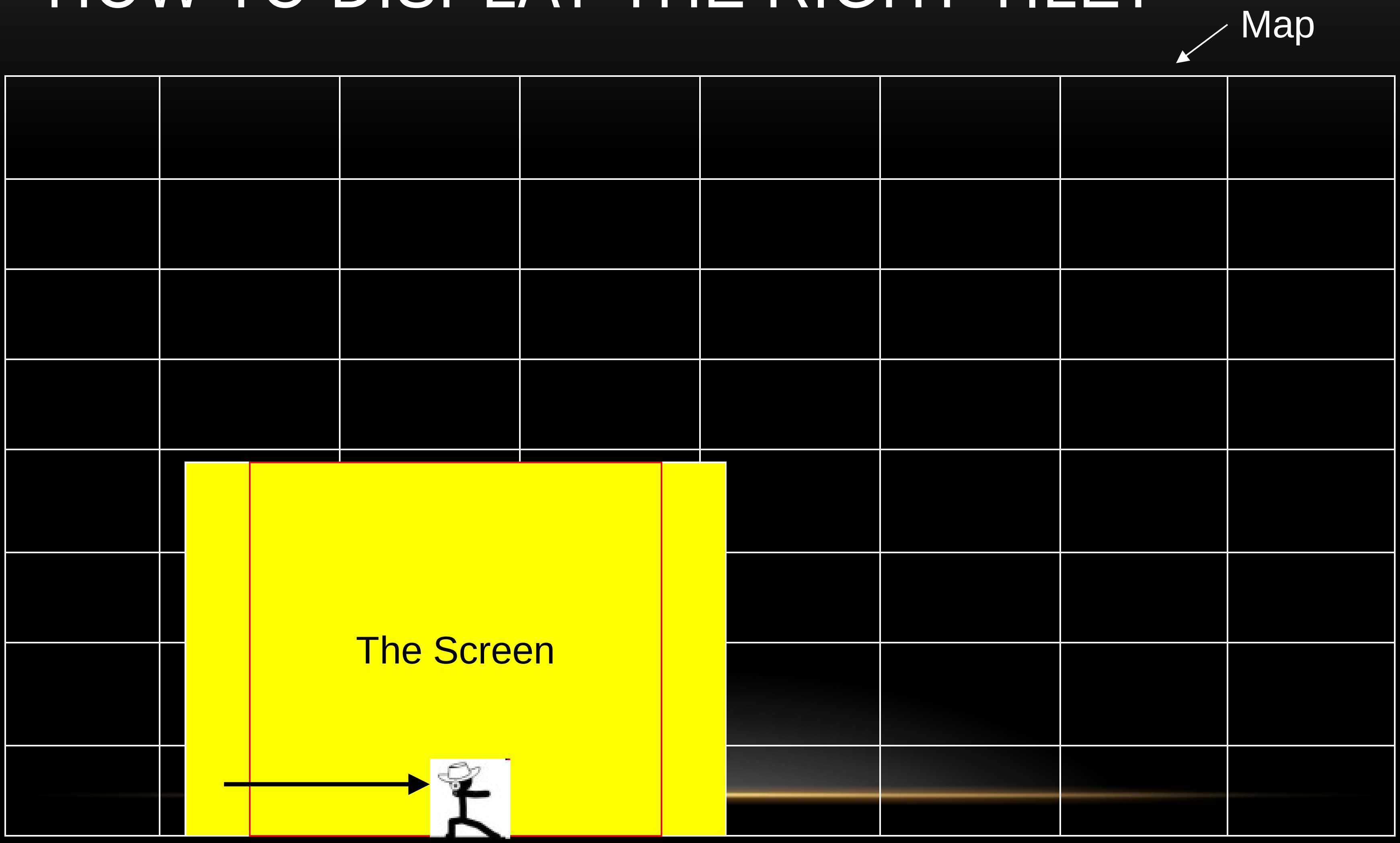
BACKGROUND SCROLLING: HOW TO DISPLAY THE RIGHT TILE?



BACKGROUND SCROLLING: HOW TO DISPLAY THE RIGHT TILE?



BACKGROUND SCROLLING:
HOW TO DISPLAY THE RIGHT TILE?



BACKGROUND SCROLLING:
THE CODES

- Let's say you have 256 tiles, numbered 0-255.

```
typedef struct {
   unsigned char tile_number;
} MAP_INFO;
```

- And the map structure array that actually holds the map. Let’s say we have 128x128.

```
#define  MAP_HEIGHT     128
#define  MAPWIDTH       128

MAP_INFO map[MAP_HEIGHT][MAP_WIDTH];
```

BACKGROUND SCROLLING: THE CODES

- Load the map from a PCX file where a 128x128 image represents it. Each color palette entry (0-255) would represent the tile number to draw. If you have a standard PCX reader, just load a map like:

```
signed short map_load(char *pcx_filename)
{
   char *pcxmapbuf;
   short i, j;

   if(pcx_loadimage(pcx_filename, pcxmapbuf) == -1)
   return -1;

   for(i = 0; i < 128; i++) {
   for(j = 0; j < 128; j++)
              map[i][j] = get_pixel(pcxmapbuf, j, i);
   }
   free(pcxmapbuf);

   return 1;
}
```

BACKGROUND SCROLLING: THE CODES

- Store the tiles in memory:

```
#define NUM_TILES       256

TILE_INFO tiles[NUM_TILES];
```

- To render a simple view, with no scrolling, we start at the top-left of the screen and render across and down until the screen is full. Let's say we're using 16x16 tiles and a screen res of 320x240. That means we will fit (320/16) 20 tiles across and (240/16) 15 vertically. That means we will draw 300 tiles a screen.:

```
void map_draw(short mapx, short mapy)
{
   short i, j;

   for(i = 0; i < 15; i++) {
     for(j = 0; j < 20; j++)
          tile_draw(map[mapy + i][mapx + j].tile_number, j * 16, i * 16);
   }
}
```

BACKGROUND SCROLLING: HOW TO SCROLL?

- To do smooth scrolling, we need the tile sizes for easier calculations.

- We need a fine coordinate system for the map.
  - The fine coordinates are simply the size of the tiles, or 16x16.
  - So each map coordinate now is 16x16 in size.
    - That means a map 128x128 is 2048x2048 in fine coordinates.

BACKGROUND SCROLLING:
HOW TO SCROLL?

- When we render a map from now on, we tell it to draw from those coordinates.
- For the rendering function to deal with it, it must decide how much to scroll all the tiles we draw left and up.
- So basically, we're now making the map seem like a huge bitmap.
  - We pick an exact coordinate from that to start drawing in the top-left of the screen.
  - To do that from tiles, we have to decided how the tiles align into that large area.

BACKGROUND SCROLLING: HOW TO SCROLL?

- That's why we use tile sizes with the power of 2.
  - First of all, when we want to render the map, say at 100,126 , we first find the coordinate in the map array.

- Since the fine coordinates run off the tile size, we just divide the numbers by 16, coming up with (100/16) 6, (126/16) 7. That tile will be first draw at the top-left of the screen.

```
    mapx = map_drawx / 16;
    mapy = map_drawy / 16;
```

BACKGROUND SCROLLING:
HOW TO SCROLL?

- What about the remainder?
  - 100/16 = 6.25 and 126/16 = 7.88.
  - Those are the fine scroll parts.

- If we take the coordinates to draw and AND them by the size of the (tile -1), we have those remainders. In our case that's (100 & 15) = 4, (126 & 15) = 14.

```
map_xoff = map_drawx & 15;
map_yoff = map_drawy & 15;
```

BACKGROUND SCROLLING: HOW TO SCROLL?

- So we now start drawing the tiles 4 pixels to the left and 14 pixels up from where you normally would.
- Note that because of this, we will have to draw one more row and column of tiles or the right and bottom edge
  - Because we drew 1 tile less in each direction.

BACKGROUND SCROLLING: THE CODES

```
void map_draw(short map_drawx, short map_drawy)
{
   short i, j;
   short mapx, mapy;
   short map_xoff, map_yoff;

   mapx = map_drawx / 16;
   mapy = map_drawy / 16;

   map_xoff = map_drawx & 15;
   map_yoff = map_drawy & 15;

   for(i = 0; i < 16; i++) {
        for(j = 0; j < 21; j++)
        tile_draw(map[mapy + i][mapx + j].tile_number, j *
                          16 - map_xoff, i * 16 - map_yoff);
   }
}
```

SUMMARY

- We had discussed about these topics today
  - Screen-based Games
  - Background Scrolling