

DM 2231 GAMES DEVELOPMENT TECHNIQUES

2015/16 SEMESTER 1

Week 10 – 2D Game Programming #1

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

- Implementing Game Audio
 - Why Audio ?
 - Generation
 - Types
 - Common Formats
 - Incident Sound
 - Music
 - Common Audio Libraries
 - irrKlang
 - How to use irrKlang
 - How to use irrKlang's 3D sound

TABLE OF CONTENT

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

2D GAME BASICS

- First generation of video games were all 2D.
 - The game world have only two axes
 - up-down
 - left-right
 - But some games (Diablo etc) use isometric perspective to give the illusion of depth.
- Hardware running 2D games may be low-spec
 - Arcade machine
 - Mobile phones
 - Handheld game consoles

2D GAME BASICS

- Problem: Hardware may limit the capabilities of the game
 - Some may have 64kb of ram
 - Some may have 8khz of CPU
- Solution: Use less CPU and memory intensive techniques on these hardwares.
 - Use data structure to...
 - encode the character graphics
 - encode background images
 - store the game map

2D GAME BASICS

- Sprite-based Characters
 - is a 2-D image or animation that is integrated into a larger scene.
 - Usually 2D games store each character in a rectangular, bitmapped graphic
 - Benefit: it can be drawn quickly onscreen
 - May include transparency information
 - Sprites can be stored in a variety of ways depending on the hardware



2D GAME BASICS

- How do you store the images for the following?
 - Samsung Galaxy S has 480 x 800 pixels?
 - Images stored as 800x600, 24-bit color with alpha effects, memory space needed is 1,920,000 bytes
 - Images stored as 480x800, 24-bit color with alpha effects, memory space needed is 1,536,000 bytes
 - Nokia 1100 which has 96 x 65 pixels?
 - Images stored as 96x65, 24-bit color, memory space needed is 149,760 bytes
- Down-sampling reduce the image for the smaller display == CPU intensive
- Both memory and CPU intensive == bad for low-spec hardwares

2D GAME BASICS

- The Sinclair ZX Spectrum is an early day computer system.
 - Memory: 48Kb
 - Image resolution: 256×192
 - It does not have a framebuffer
 - To conserve memory, colour is stored separate from the pixel bitmap in a low resolution




2D GAME BASICS

- Can it hold a 128x128 sprite with 24-bit colour depth?
 - $128 * 128 * 24 = 393,216 \text{ bits} = 49,152 \text{ bytes} = 48 \text{ Kb}$
 - NO MEMORY SPACE TO STORE OTHER THINGS!

DATA STRUCTURES FOR 2D GAMES

- For Spectrum system, it uses 8x8 two-colour sprites.
 - Each sprite has 8 pixels by 8 pixels
 - Each pixel has only 1 bit <- SAVES MEMORY SPACE!
 - Each sprite has 2 palette tables to indicate the colour to display for that sprite
 - The palette table has 16 colours, so it takes up 4 bits
 - One table each for foreground and background
 - Overall, 9 bytes ($9 \times 8 = 72$ bits) per sprite are used.
- For Sinclair ZX Spectrum with 48Kb of memory, you can store $49152 \text{ bytes} / 9 \text{ byte} = 5461.33$ images.



How did they work around it?

8x8 pixels @ 1 bit each = 64 bits

4-bit each for
foregrnd+bkgnd = 8
bits

DATA STRUCTURES FOR 2D GAMES

- If each pixel can display 4-bits of data, then an 8x8 sprite can display 16 colours per pixel.
 - Total memory usage = 8×8 pixels @ 4 bits per pixel = 256 bits.
 - Overall, 32 bytes ($32 \times 8 = 256$ bits) per sprite are used.
 - For Sinclair ZX Spectrum with 48Kb of memory, you can store $49152 \text{ bytes} / 32 \text{ bytes} = 1536$ images.

1 pixel @ 4 bit = 4 bits

DATA STRUCTURES FOR 2D GAMES

- What if we want to display our 8x8 sprite as tiles on a 320 x 200 display @ 256 colours per pixel on a personal computer,
 - For 256 colours, we need 8-bits of data per pixel
 - Memory usage per sprite = 8×8 pixels @ 8 bits per pixel = 512 bits == 64 bytes
 - Each colour palette = 256 colours x 24-bit = 6144 bits = 768 bytes
 - TWO colour palettes = 1536 bytes
 - 320 x 200 resolution = 40 x 25 sprites
 - Memory used for sprites = $40 * 25 * 64 = 64000$ bytes
 - Total memory = $64000 + 1536$ bytes = 65536 bytes = 64Kb

DATA STRUCTURES FOR 2D GAMES

- Question: What is the purpose of knowing all these?
- Answer: You may need to develop resource intensive games for low performance hardware.



MAPPING MATRICES

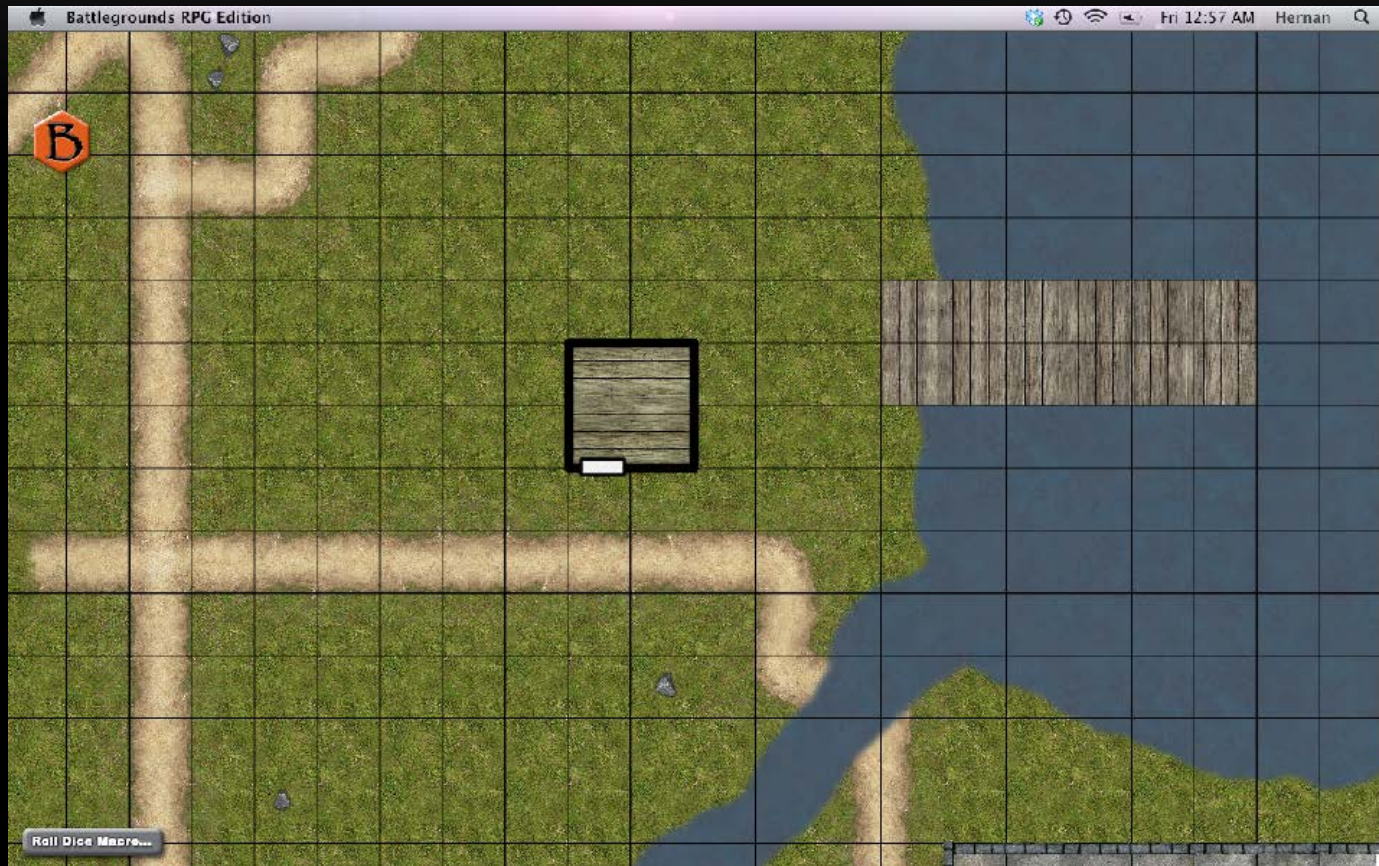
- Many 2D games can present large, varied game worlds on very limited machines.
 - Use compression techniques!
 - Make data fit on the very small memory chips.
 - As an example, let's analyze the memory used by a top-down game such as The Legend of Zelda
 - We assume the entire game level occupies 5x5 television screens across, and that every screen is 256x200 pixels, and each pixel displays 256 colors.
 - Loading a single bitmap with those dimensions will use 1.28MB of memory space, clearly more than a Nintendo console had at that time.
 - How to achieve this?
 - Use mapping!

MAPPING MATRICES

- **Mapping** is a **compression technique** that will allow us to create good-looking game worlds at a fraction of the memory footprint.
- We lose some visual variety in the process, but our dream game will fit on our target platform.
- It is an extremely popular technique.
 - Used in thousands of games for classic consoles and arcades, and is still used today, even in some 3D games.
 - The key idea is to divide our game world into a set of tiles or basic primitives.
 - Each tile will represent a rectangular pattern, which we will combine with other tiles to represent the level.
 - So, if our game world must represent grass, stone, snow, and sand, we will use four tiles, and then map them as if we were putting tiles on the floor.

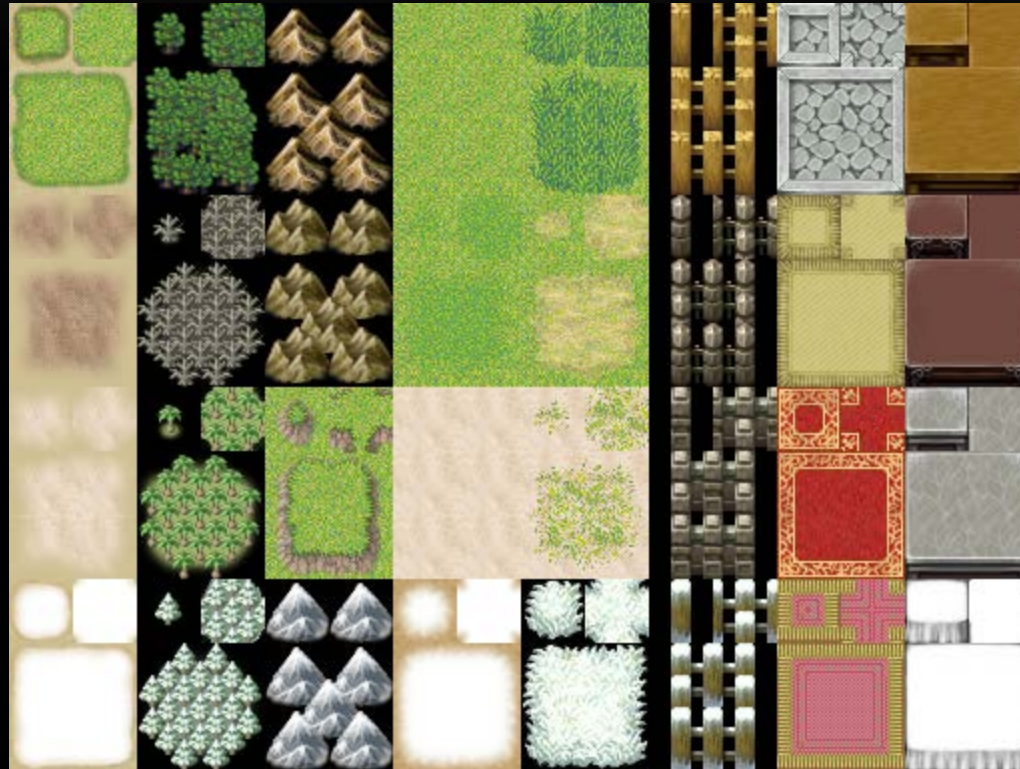
MAPPING MATRICES

A map which uses tiles



MAPPING MATRICES

A tile map



MAPPING MATRICES

- The compression comes from the fact that, if we repeat the same pattern frequently, we will only have it once in memory.
- To prove this, let's assume we represent our initial game with a tile set.
 - We will use 256 different tiles (more than enough to provide richness to the game world).
 - Each tile will be 8x8 pixels. So, each tile occupies
 - $8 \times 8 = 64 = 32$ bytes (we are using 16 colors only)
 - The whole tile set will require
 - $32 \times 256 = 8\text{KB}$
- We will call this structure the tile table (more on tile tables in the next section).
- We will then need a second data structure, called the mapping matrix.

MAPPING MATRICES

- This structure stores the layout information: how the tiles should be arranged in game level. From the sprite size, we know the size of our mapping matrix will be
 - $256 \times 5 / 8 = 160$
 - $200 \times 5 / 8 = 125$
- Because each entry will be a byte-sized value so we can index the 256 possible tiles, the whole table will require
 - $160 \times 125 = 20000$ bytes
- So, our compressed map will take up around 19.5KB, with 8 extra kilobytes dedicated to the tile list.
- On the whole, that is 27.5KB, down from **1.28MB**. That means dividing by a factor of 50 approximately.
- Mapping can help to reduce memory usage, and that's the reason mapping was extremely popular in the '80s.
 - Games such as Mario Bros, Zelda, 1942, and many others use variants of the mapped scheme.

TILE TABLES

- 2D scrolling games have background graphics or tiles.
- The data structure used is the tile table (TT).
 - A list of background images that can be tiled and combined using a mapping matrix to create a complete game map.
- Some platforms, such as the NES, had this structure defined in the hardware's specifications.
- Others, like PCs, allowed the developer to specify its own.
- In both cases, the TT is used to store unique tiles that will later be mapped to the screen.
- There are a number of decisions involved in creating an efficient TT.
 - Format of the tiles
 - Number of tiles

TILE TABLES: FORMAT OF THE TILES

- Different formats affect the table's size.
- Traditionally, tile sizes used to be powers of 2, as this allowed some optimization in the blitting routines used to transfer them to the screen.
 - Instead of transferring bytes, we can use words or even 32-bit values for increased efficiency.
- Are all tiles of the same size or different?
 - Classic games used equal-sized tiles for easier screen rendering.
 - For a real-time strategy games uses an isometric view with buildings of different sizes, wouldn't it be wiser to allow for different tile sizes?

TILE TABLES: FORMAT OF THE TILES

- In addition, we must decide the color format of the tiles.
 - In the old days, tiles were palletized, so each pixel was encoded in one byte that indexed the color palette.
 - However, more recent games have used high-color modes (16 bits encoding RGB in 6-5-5) or even true color (24 bits encoding RGB as 8-8-8).
 - Clearly, the more colors the better, but more colors means more memory, more bus usage, and thus less performance.
- However, there is an equation that will give us the memory size of a single tile, as a function of several parameters.

$$size = bitsPerPixel \times width \times height$$

TILE TABLES: NUMBER OF TILES

- The number of tiles held in the TT will be as important as their format.
 - On one hand, more tiles means nicer graphics.
 - On the other hand, memory use will increase dramatically in more ways than you think.
- Imagine that our game needs to hold 256 different tiles.
 - The TT will encode them in positions 0 to 255.
 - Thus, each position in the mapping matrix will need to index that table using an unsigned, 8-bit number.
 - However, imagine that our artist raises the bar to 300 tiles.
 - The TT will grow accordingly, but the mapping matrix will also undergo some changes. We cannot encode 300 values in a byte.

TILE TABLES: NUMBER OF TILES

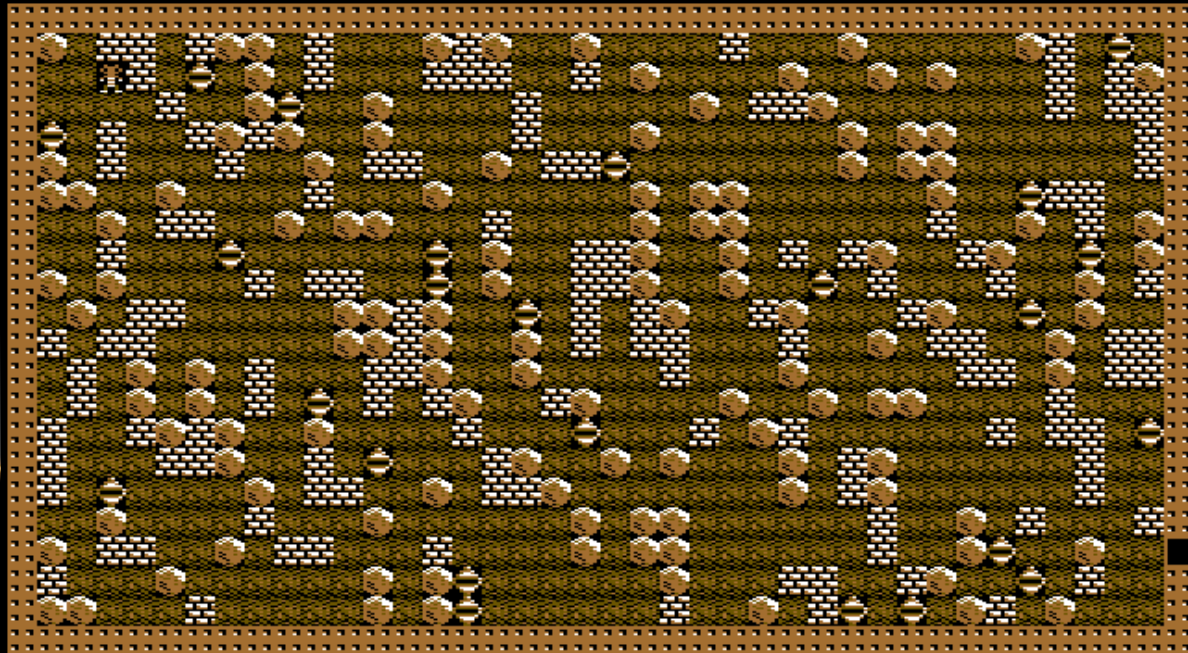
- We have two options:
 - Use 9 bits, which allow 512 values, but require some rather obfuscated code to access.
 - Use a 16-bit value, which will take up double the memory but give us simple access.
- An alternative to allow this in platforms that have some sort of file system is to ensure that only 256 tiles are used per level, so each level has a different TT. This way we can preserve the variety without the memory hit.
 - But some platforms require the full program (including data) to be in main memory at the same time, so there is no way to select the TT.

2D GAME ALGORITHMS

- We have reviewed the constituent elements of traditional 2D games.
- We will now move on to specific game techniques, reviewing the algorithms behind popular classics such as Zelda and Mario Bros.
- All the algorithms use the same data structures we discussed previously:
 - tile tables,
 - sprites, and
 - mapping matrices.

2D GAME ALGORITHMS: SCREEN-BASED GAMES

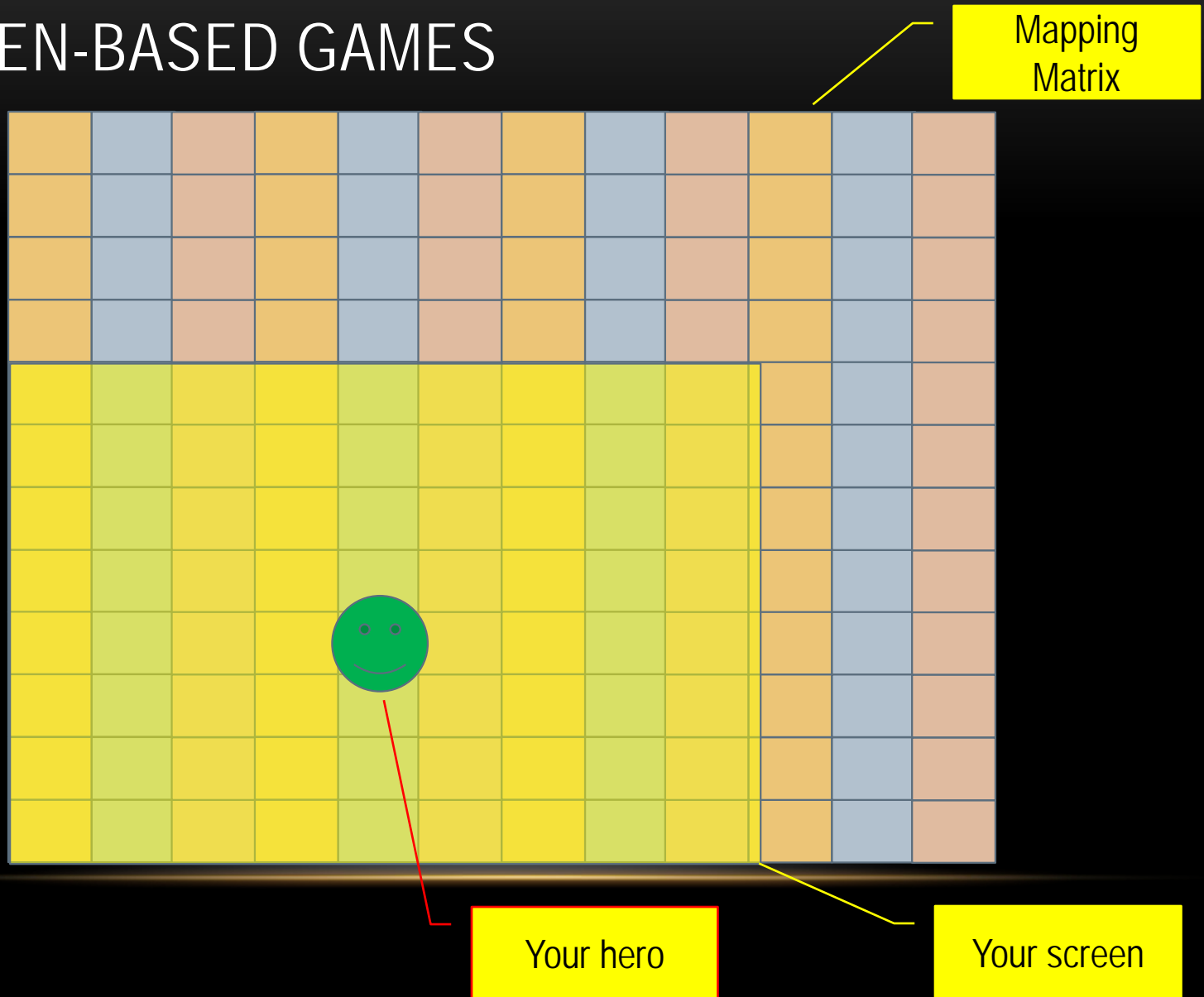
- The simplest mapped game is the screen-based game, in which the player confronts a series of screens.
- When he exits one screen through its edge, the graphics are substituted by those in the next screen, and so forth.
- There is no continuity or transition between screens.
 - Example is Boulder Dash.



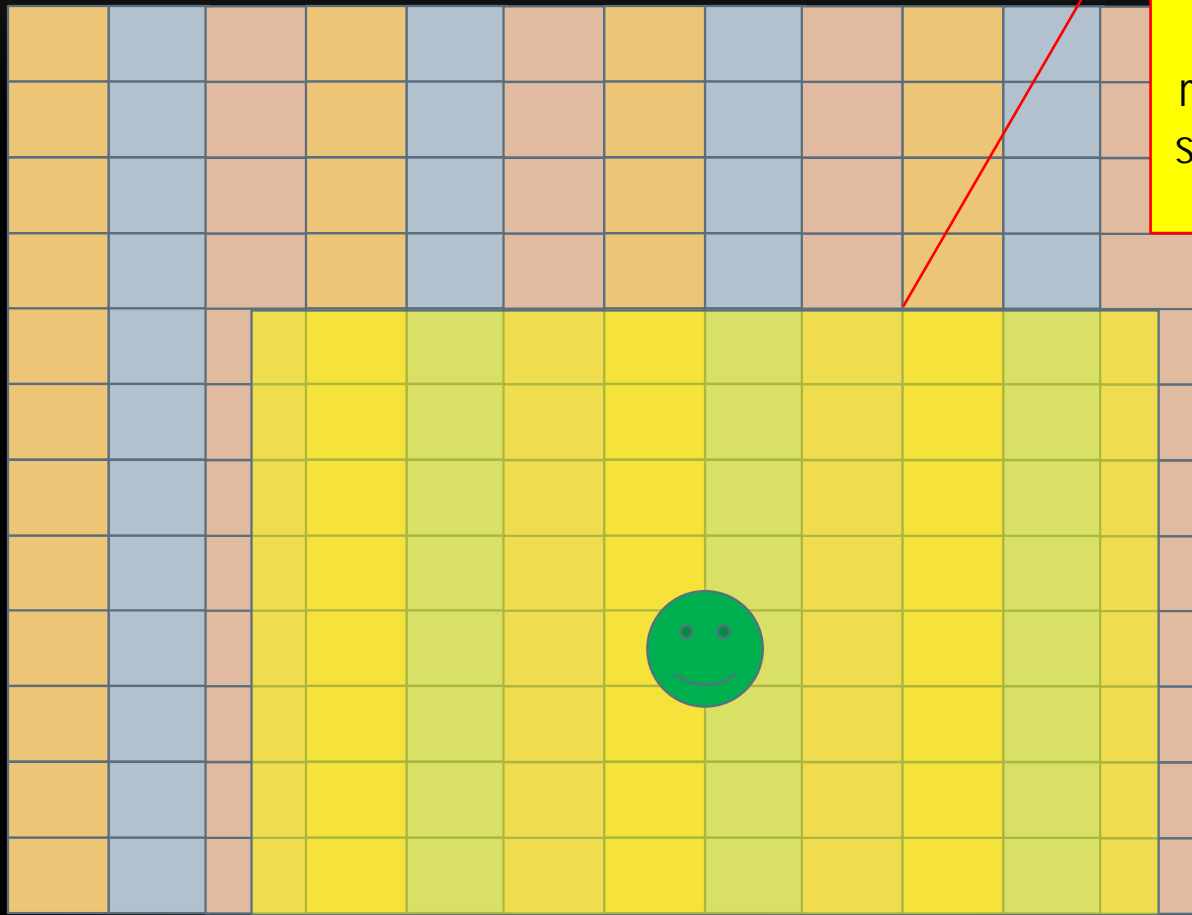
2D GAME ALGORITHMS: SCREEN-BASED GAMES

- 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



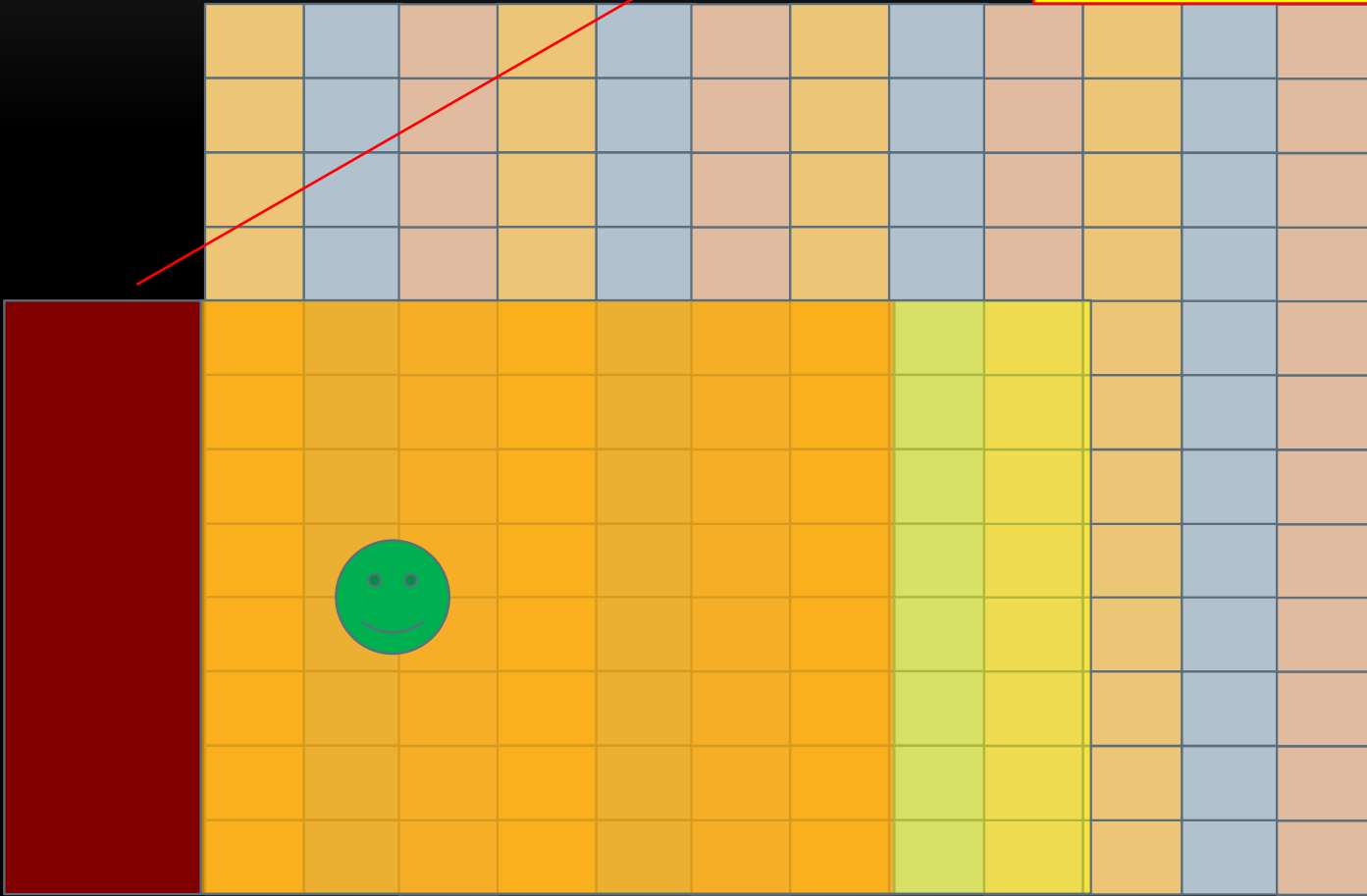
2D GAME ALGORITHMS: SCREEN-BASED GAMES



When your hero moves across the map, the screen will show a different part of it.

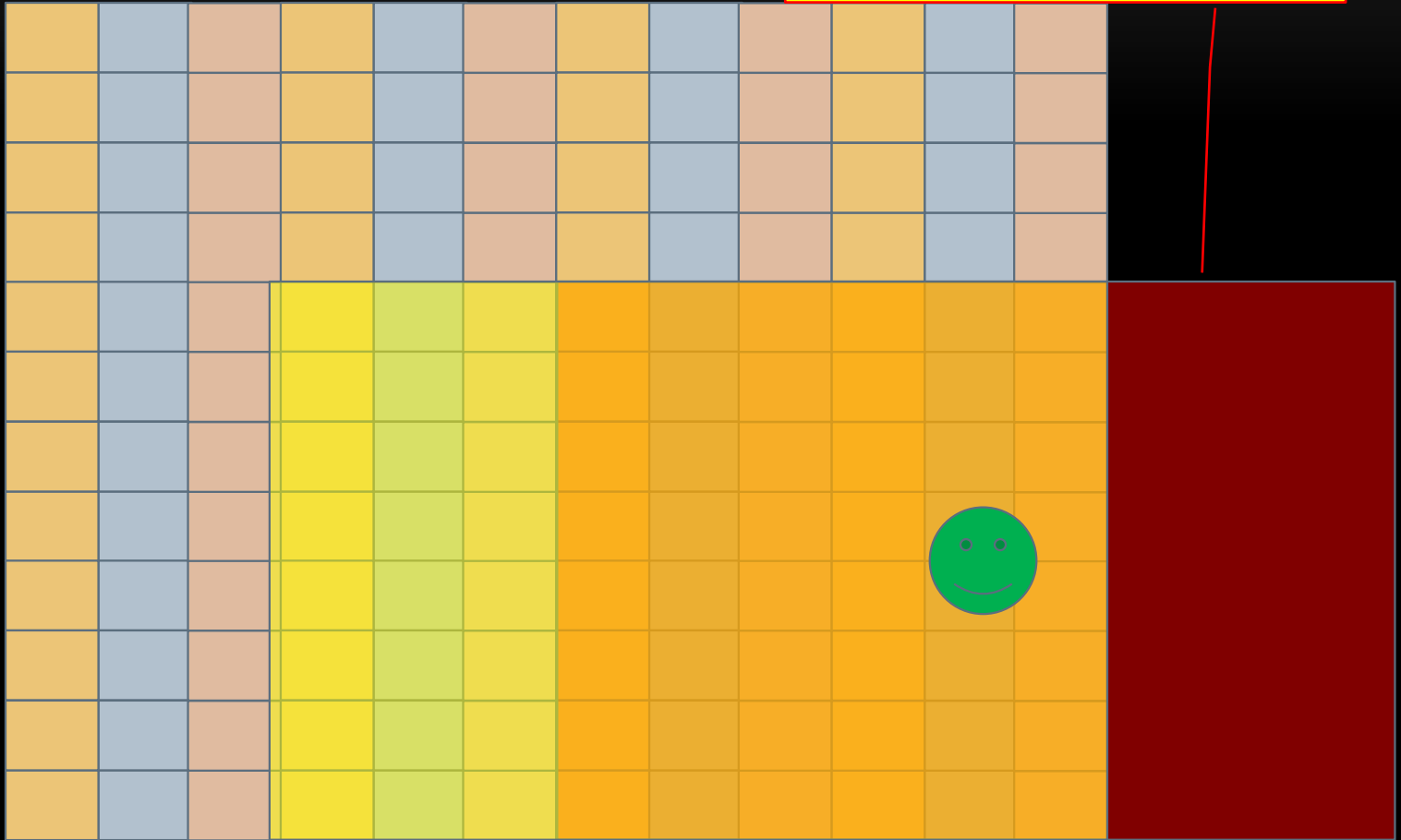
2D GAME ALGORITHMS: SCREEN-BASED GAMES

When the hero move out of the map, there are no tiles to display in the red region!



2D GAME ALGORITHMS: SCREEN-BASED GAMES

When the hero move out of the map, there are no tiles to display in the red region!



Solution: Constrain the screen within the map and move the hero toward the edges of the map. You need to stop the hero if he reaches the edge of the screen!

2D GAME ALGORITHMS: SCREEN-BASED GAMES

- Its code would look something like this:

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

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 [yi][xi];
        DisplayTile(tile_table[tileid],screex,screey);
    }
}
```

More details will be available in the lab session!

2D GAME ALGORITHMS: SCREEN-BASED GAMES

- An interesting variant of the preceding code generalizes the mapping matrix to a 3D matrix indexed by room identifier, x value and y value.
 - room identifier == level?
- 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];
```

SUMMARY

- We had discussed about these topics today
 - 2D Game Basics
 - Data Structures for 2D Games
 - Mapping Matrices
 - Tile Tables
 - 2D Game Algorithm