2D Game Algorithms

- 2D Game Algorithms
  - Screen-Based Games
  - Scrolling Game
  - Multilayered Engines
  - Semi-3D approach
    - Parallax Scrollers
    - Isometric Engines
  - Page-Swap Scroller

2D Game Algorithms

- **Screen-Based Games**
  - **The player confronts a series of screens**
    - **screen == gameworld**
  - **No continuity or transition between screens**
    - **Ex) 320x240 screen using 32x32 tiles**

```
#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 screenx = xi * tile_wide;
            int screeny = yi * tile_high;
            int tileid = mapping_matrix [yi][xi];
            blit(tile_table[tileid], screenx, screeny);
      }
}
```

Hold whole game map:
Mapping matrix [roomid] [yi] [xi]

2D Game Algorithms

- Two- and Four-way Scrollers (= Scrolling Game)
  - Create a larger than-screen gameworld that we can continually explore from a sliding camera
  - A continuum, with no screen swapping at all
  - More complex than screen-based game
  - Ex) 1942(2-way top-down), Super Mario Bros(2-way side-scrolling), Zelda(4-way top-down scrolling)

2D Game Algorithms

- **Scrolling Game (Complete rendering loop)**

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

tileplayerx= playerx/tile_wide
tileplayery= playery/tile_high

int xtiles=screen_wide/tile_wide;
int ytiles=screen_high/tile_high;

int beginx= tileplayerx – xtiles/2;
int beginy= tileplayery – ytiles/2;
int endx= tileplayerx + xtiles/2;
int endy= tileplayery + ytiles/2;
```

```
for (yi=beginy;yi<endy;yi++){
  for (xi=beginx;xi<endx;xi++) {
    int screenx=xi*tile_wide -playerx +screenplayerx;
    int screeny=yi*tile_high -playery +screenplayery;
    int tileid=mapping_matrix [yi][xi];
    blit(tile_table[tileid],screenx,screeny);
  }
}
```

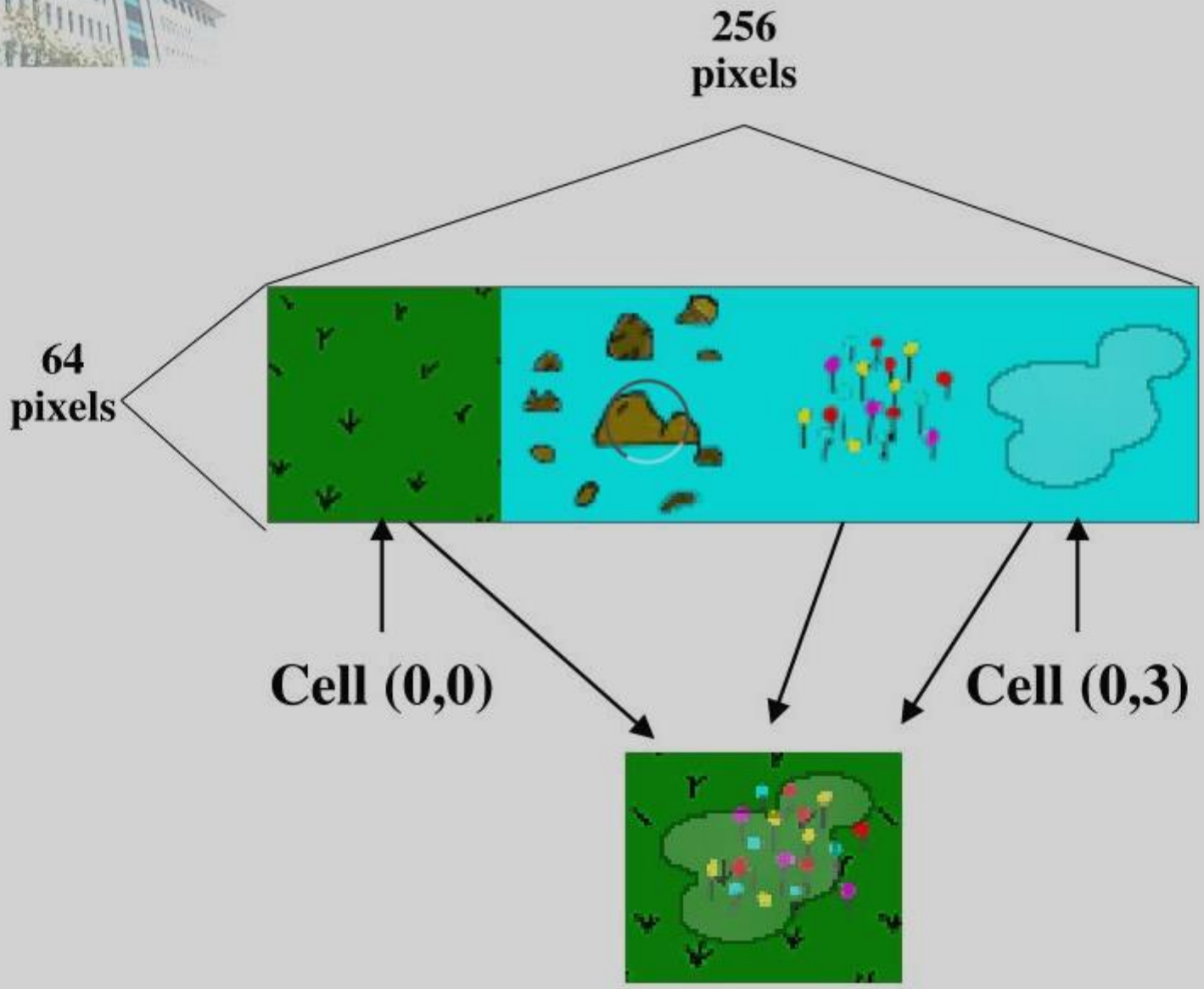
Gameworld

2D Game Algorithms

- **Multilayered Engines**
  - Use several mapping matrices to encode the game map
    - Need to combine tiles
    - Need to move objects over the BG
    - Want to give the illusion of depth
    - Ex) BG: terrains, another: trees

```
for (yi=beginy; yi<endy; yi++){
    for (xi=beginx; xi<endx; xi++) {
        int screenx=xi*tile_wide-playerx+screenplayerx;
        int screeny=yi*tile_high-playery+screenplayery;
        for (layeri=0;layeri<numlayers;layeri++) {
            int tileid=mapping_matrix [layeri][yi][xi];
            if (tileid>0)   blit(tile_table[tileid],screenx,screeny);
        }
    }
}
```

Ex: 1x4 Bitmap Template



Multi-layering Tiles

- Most worlds require layering. Ex:
  - place grass
  - place flowers on grass
  - place cloud over flowers
- Other common objects:
  - trees
  - rocks
  - treasure
- To edit:
  - use multiple tiles, one for each layer
  - map file may join & order tiles

2D Game Algorithms

- **Semi-3D approach**
  - **Parallax Scrollers**
    - **The illusion of a third dimension by simulating depth**
      - ✓ Storing depth-layered tiles
      - ✓ Moving them at different speeds to convey a sense of depth

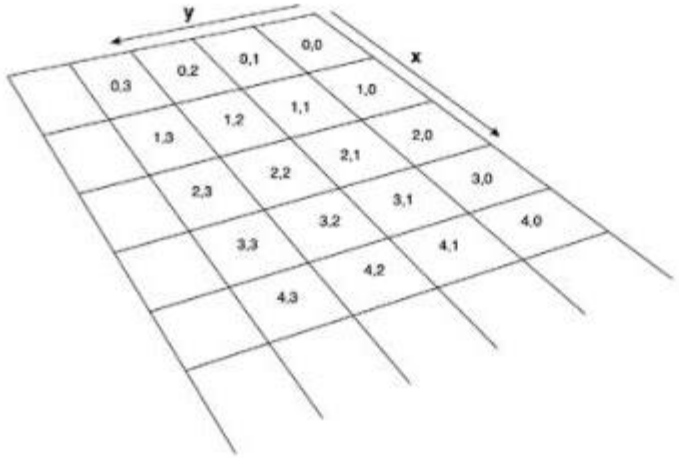
```
if (pressed the right cursor)
                    for (layeri=0;layeri<numlayers;layeri++)
                  playerx[layeri]+=1*(layeri+1);

for (layeri=0;layeri<numlayers;layeri++) {
    for (yi=beginy; yi<endy; yi++){
        for (xi=beginx; xi<endx; xi++) {
            int screenx=xi*tile_wide-playerx[layeri]-screenplayerx;
            int screeny=yi*tile_high-playery[layeri]-screenplayery;
            int tileid=mapping_matrix [layeri][yi][xi];
            if (tileid>0) blit(tile_table[tileid],screenx,screeny);
        }
    }
```

2D Game Algorithms

- **Isometric Engines**
  - **Representing an object from raised viewpoint (rotate 45)**
    - ✓ **Parallel projection ➔ do not suffer from distortion**
  - **Tiles for an isometric(같은크기) are rhomboids (평행사변형)**
    - ✓ **Tend to be wider than they are high**

- **Ex) Diablo**



2D Game Algorithms

- **Page-Swap Scroller**
  - Without being restricted to a closed set of tiles
    - Sector should be loaded into main memory
    - The rest are stored secondary media
      - ✓ Will be swapped into MM as needed
  - The mapper resembles a cache memory
  - Improve performance
    - The velocity of the player ??

Special Effects

- Palette Effects
- Stippling Effects
- Glenzing
- Fire

Palette Effects

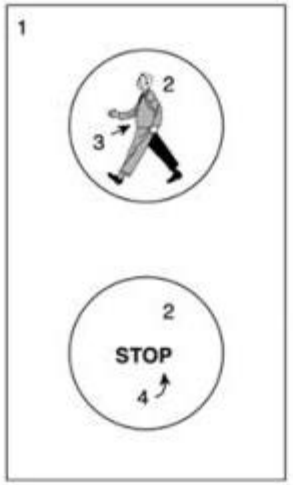
- **Palette Effects**
  - Implemented by manipulating, not the screen itself, but the hardware color palette
    - Altering the palette was much faster than having to write to the frame buffer (not depend on the screen resolution)
- **fade in/out**

```
void FadeOut()
{
        unsigned char r,g,b;
        for (int isteps=0;isteps<64;isteps++)
      {
         WaitVSync();
         for (int ipal=0;ipal<256;ipal++)  {
             GetPaletteEntry(ipal,r,g,b);
             if (r>0) r--;
             if (g>0) g--;
             if (b>0) b--;
             SetPaletteEntry(ipal,r,g,b);
         }
       }
}
```

```
void FadeIn()
{
        unsigned char r, g, b;
        for (int isteps=0;isteps<64;isteps++)
      {
         WaitVSync();
         for (int ipal=0;ipal<256;ipal++)  {
             GetPaletteEntry(ipal,r,g,b);
             if (r<palette[ipal].r) r++;
             if (g<palette[ipal].g) g++;
             if (b<palette[ipal].b) b++;
             SetPaletteEntry(ipal, r, g, b);
         }
       }
}
```

palette rotation

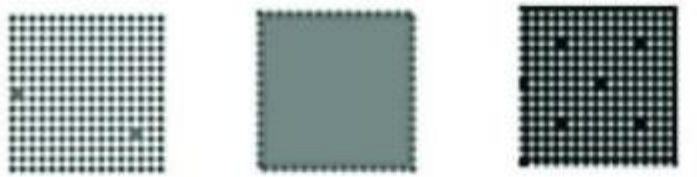
- if we change some palette entries, we can produce color changes in sprites that look like real animation.
  - Ex) water, lava, fire, neon glows
    - Semaphore(신호등)
      - ✓ four palette entries
      - ✓ 1: Yellow
      - ✓ 2: Black
      - ✓ 3: Green walking char.
      - ✓ 4: Red stop sign



    - Animated water(물결)
      - ✓ Reserve six palettes
      - ✓ store different hue of water color (Deep blue ➔ light blue)

Stippling Effects

- **Stipple**
  - A simple patterned texture that combines one color (generally black or grey) with the transparency index
- **Illusion of shadow(그림자 표현)**
  1. Render the background
  2. Using the transparency index, render the stipple
  3. Render the character
- **Fog(안개)**
  1. Render the background.
  2. Render the character.
  3. Using the transparency index, render the stipple.



- **illuminate parts of the scene**
  - Stippling pattern must be colored as the light source (yellow, orange)
- **fog-of-war techniques**
  - Where only the part of the map where the player has actually explored is shown, and the rest is covered in fog
    - The closer the area, the less dense the fog

Glenzing

- Stippling
  - Nothing but a poor man's transparency
- Glenzing
  - Really mix colors as if we were painting a partially transparent object
  - Convert a color interpolation into a palette value interpolation
  - Better than those achieved by simple stippling

Color = Color_transparent*opacity + Color_opaque*(1-opacity)

Fire Effect

- **Fire Effect**
  - Can be an animated sprite
  - Using 2D particle system
  - Using a **cellular automata** on the frame buffer
    - Automata consisting of a number of cells running in parallel whose behavior is governed by neighboring cells
    - Ex) simulate life, create fire
      - ✓ Fire emitter
        - pure white fire color ➔ yellow, orange, red, black

$$color(x,y) = (color(x,y+1) + color(x+1,y+1) + color(x-1,y+1))/3$$

Expensive effect: need the whole screen to be recalculated at each frame
➔ Confine to a specific area

Fire Effect

```
// generate new sparks
for (int i=0;i<SCREENX/2;i++)  {
        int x=rand()%SCREENX;
        int col=rand()%25;
        PutPixel(x,SCREENY-1,col);    // emitted by the bottom of the screen
}

// recompute fire
for (int ix=0;ix<SCREENX;ix++) {
   for (int iy=0;iy<SCREENY;iy++){
      unsigned char col;
      col =  (GetPixel(ix-1,iy+1) + GetPixel(ix,iy+1) + GetPixel(ix+1,iy+1)) / 3;
      PutPixel (ix,iy,col);
   }
}
```

Sprite Data

- Suppose we wanted to draw an animated Mario, what data might we need?
  - position
  - z-order (huh?)
  - speed
  - direction
  - Texture(s)
    - array of Textures if using individual images
    - each index represents a frame of animation
  - possible states of sprite
  - current state of sprite (standing, running, jumping, dying, etc.)
  - animation sequences for different states. Huh?
  - current frame being displayed (an index)
  - animation speed