Games Development

Jim Holmström
2D Graphics – Screen Layout

- Coordinate system
- Upper left corner is (0,0)
- Each point is a pixel (picture element)
- Pixel Color – Bit depth
  - 3 parts RGB (red, green, blue)
  - 16 bit, $2^{16}$ colors, 5bit red & blue, 6bit green
  - 32 bit, 8 bits each (RGB), $2^{24}$ colors.

Pixel at (2,5)
2D Graphics – Resolution

- Screen Resolution – a measure
  - 800x600 – width and height, x and y
  - The number of pixels
- 800x600 in 24bit color is about 1.37MB of memory
- Refresh rate Hz (hertz) – how often the screen is refreshed.
import java.awt.*;
import javax.swing.JFrame;

public class FullScreenTest2 extends JFrame {
    private static final long DEMO_TIME = 5000;
    
    public static void main(String[] args) {
        FullScreenTest2 test = new FullScreenTest2();
        test.go();
    }

    public void go() {
        DisplayMode displayMode = new DisplayMode(800, 600, 16, DisplayMode.REFRESH_RATE_UNKNOWN);
        setBackground(Color.blue);
        setForeground(Color.white);
        setFont(new Font("Dialog", 0, 24));

        SimpleScreenManager screen = new SimpleScreenManager();
        try {
            screen.setFullScreen(displayMode, this);
            try { Thread.sleep(DEMO_TIME); } catch (InterruptedException ex) {} 
            
        } finally { screen.restoreScreen(); }

        public void paint(Graphics g) {
            g.drawString("Hello World!", 20, 50);
        }
    }
}

(DEMO : FullScreenTest2)
2D Graphics – Images

- Image formats
  - E.g. JPEG, BMP, GIF, PNG
  - Some formats store images with no loss of information others loose quality while gaining compression.
2D Graphics – Images

- **Loading image in java**
  ```java
  ImageIcon icon = new ImageIcon("img.jpg");
  Image image = icon.getImage();
  ```

- **Painting an image on the screen**
  ```java
  Graphics g
  g.drawImage(image, X, Y, null);
  ```

- Image is painted, top left corner starting at (X,Y)
2D Graphics – Images

- (DEMO : ImageTest)
  - Opaque Image – draw the entire box.
  - Transparent – background is transparent
  - Translucent – image is see-through
  - Anti-alised – edges of image are translucent
  - Translucent type images take more time to draw.
2D Graphics – Images

Transformations – rotate, scale and translate

```java
AffineTransform t = new AffineTransform();
t.translate(35, 35);
t.scale(-1, 1);
t.rotate(25π/180);
graphics.drawImage(someImage, t, null);
```

- Translate(35, 35)
- Scale(-1, 1)  “Flip Horizontal”
- Scale(1.4, 1)
- Rotate(25π/180)  Translate(35, 35)
2D Graphics – Rendering

- Passive rendering
  - Use the paint(...) and repaint() methods.
    - May cause delays, when the AWT thread is busy.

- Active rendering
  - Draw directly to the screen, no delays.
    ```java
    Graphics g =
    screenManager.getFullScreenWindow().getGraphics();
    draw(g);
    g.dispose();
    ```
  - Must call dispose() to release resources.
2D Graphics – Animation

- Animated images
  - A collection of images associated with durations, shown in succession

- Java Classes
  - AnimationFrame – an image and the ending time
  - Animation – a list of AnimationFrames and methods for updating and adding.

```
code\Animation.java
```
2D Graphics – Animation

DEMOS: AnimationTest1

- Why is it flickering (SimpleScreenManager)
  - We are drawing the background over the image
  - The screen is updated before the image is drawn

- How do we fix this?
  - Dubbel buffering, used in ScreenManager.java
  - DEMO: AnimationTest2

code\AnimationTest1.java
code\AnimationTest2.java
2D Graphics – Double Buffer

- Double Buffering
  - Draw in a back buffer and copy to screen
  - Expensive, copy a large amount of data.

![Double Buffer Diagram]
2D Graphics – Page Flipping

- Maintain 2 buffers, and switch display pointer.
- Draw in the back buffer, then switch pointer.
- Fast.
2D Graphics – Tearing

- Tearing
  - Part of the new buffer is displayed at the same time as the old buffer.
2D Graphics – Example

- For page flipping or Double Buffering
  ```java
  frame.createBufferStrategy(2);
  ```

- Example – how to get current drawing buffer
  ```java
  BufferStrategy s = frame.getBufferStrategy();
  Graphics g = s.getDrawGraphics();
  drawSomething(g);
  g.dispose();
  s.show();
  ```

- This is implemented in `ScreenManager.java`
2D Graphics – Sprites

- What is a sprite and why do we need it
  - An “image” that moves independently on the screen.
  - Can be animated.
  - Practical abstraction - a class containing:
    - An (animated) image
    - A velocity
    - A position

- See Sprite.java

- (DEMO: SpriteTest1) (DEMO: SpriteTest2)
2D Graphics – Tile World

- Tile Based Maps
  - Grid based
  - Reusable building blocks
  - Reduced memory cost
  - Normal size: power of 2 (16, 32 or 64 pixels wide)
2D Graphics – Tile World

- Drawing Tile Maps (ch 5)
  - Need only draw tiles that are visible
2D Graphics – Tile World

- Parallax Scrolling
  - Background moves slower than foreground
  - Illusion of depth

(DEMO: Parallax)
Collision Detection

Basic techniques in 2D
Collision Detection – Why?

- Prevent player from passing through walls
- Bullets, power-ups, creatures…

How?

- Bounding objects, circles, polygons
- Pixel test
- Divide into regions and test
Collision Detection

Goals

- Eliminate as many tests as possible
- Efficiently decide if there is a collision
- Provide detection with the necessary accuracy
  - Some objects require more accurate detection, others perhaps just a box.
- Handle collisions in a non-distracting way.
  - E.g. don’t stop a player completely when they bump into a wall, allow them to slide.
Collision Detection

- **Procedure**
  - Update the object’s position
  - Check for collisions between this object and all relevant objects.
  - If there was a collision, handle it.
Basic Tile Collision

- Prevent player from passing through walls
  - Split into two tests, horizontal and vertical.
  - Correct the position of the sprite after the move.
Collision Detection

- Eliminate unnecessary tests.
  - Saves time, don’t need to test objects that are far away.
  - Maintain data-structures for which objects are in the same grid square.
Collision Detection

- Bounding Objects
  - Circle or Rectangle
    - Fast, sometimes inaccurate, circles intersect but not the objects.
    - Use more than one circle for more accurate test.
    - Combine different bounding objects.
Collision Detection – Time

- Discrete Time causes problems
  - May “pass through” objects.
  - A moving object can be treated as a new type of bounding object.