CS 696 Mobile Phone Application Development
Fall Semester, 2009
Doc 16 2D Graphics part 2
Oct 21, 2009

Basic Parts

Bitmap

- Rectangular grid of pixels of an image
- What is displayed on screen
- PNG, JPG are bitmap formats

Canvas

- Knows how to draw on bitmaps

Paint

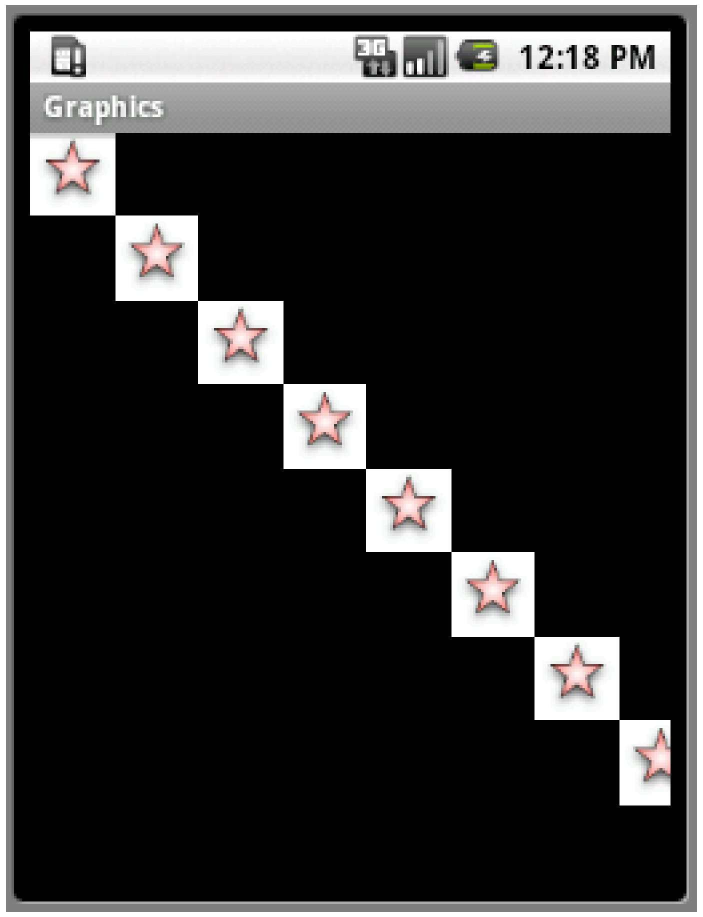
- Style and color information about how to draw things

Drawing primitive

- Rect, text , Path, Bitmap

Using Bitmaps

```
public class GraphicsExamples extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       View shapes = new SimpleDrawing(this);
       setContentView(shapes);
    }
}
```



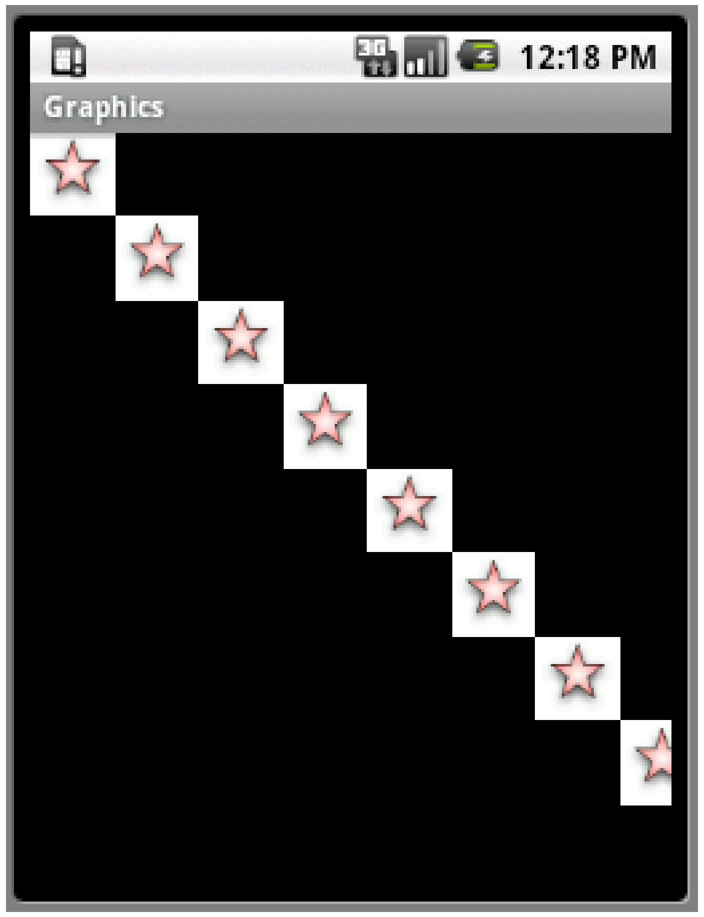
Creating the bitmap

```
public class SimpleDrawing extends View {
    int starSize = 42;
    Bitmap star = Bitmap.createBitmap(this.starSize, this.starSize,
            Bitmap.Config.ARGB_8888);

    private final Paint basicPaint = new Paint();

    public SimpleDrawing(Context context) {
        super(context);
        Resources resource = this.getContext().getResources();
        Drawable image = resource.getDrawable(R.drawable.star);
        Canvas canvas = new Canvas(this.star);
        image.setBounds(0, 0, this.starSize, this.starSize);
        image.draw(canvas);
    }
```

The drawing

```
    @Override
    protected void onDraw(Canvas canvas) {
        canvas.drawColor(Color.BLACK);
        for (int topRight = 0; topRight < 3000; topRight += this.starSize)
            canvas.drawBitmap(this.star, topRight, topRight, this.basicPaint);
    }
}
```

What is with the white rectangles?

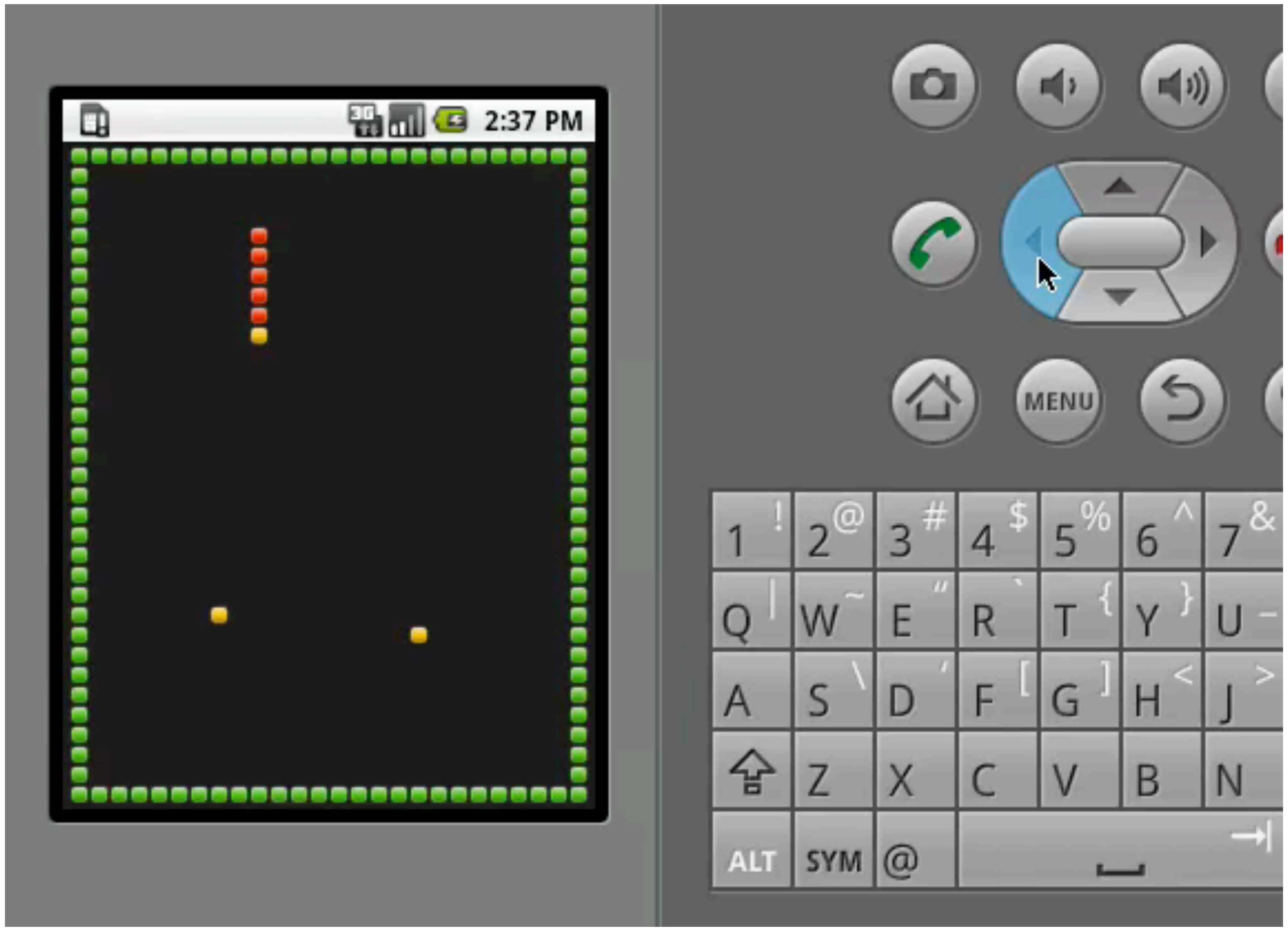


Bitmap.Config

ALPHA_8
ARGB_4444
ARGB_8888
RGB_565

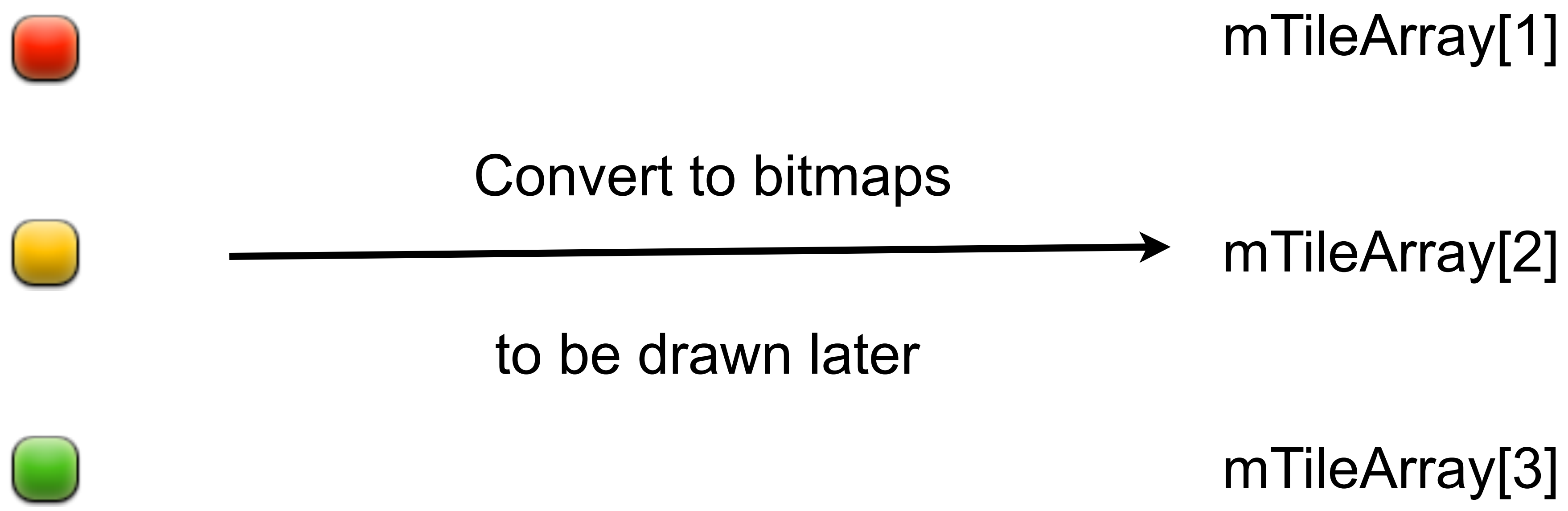
ARGB_XXXX
X = bits used to store given channel (alpha, color)

Snake



The Images

Hand created png files

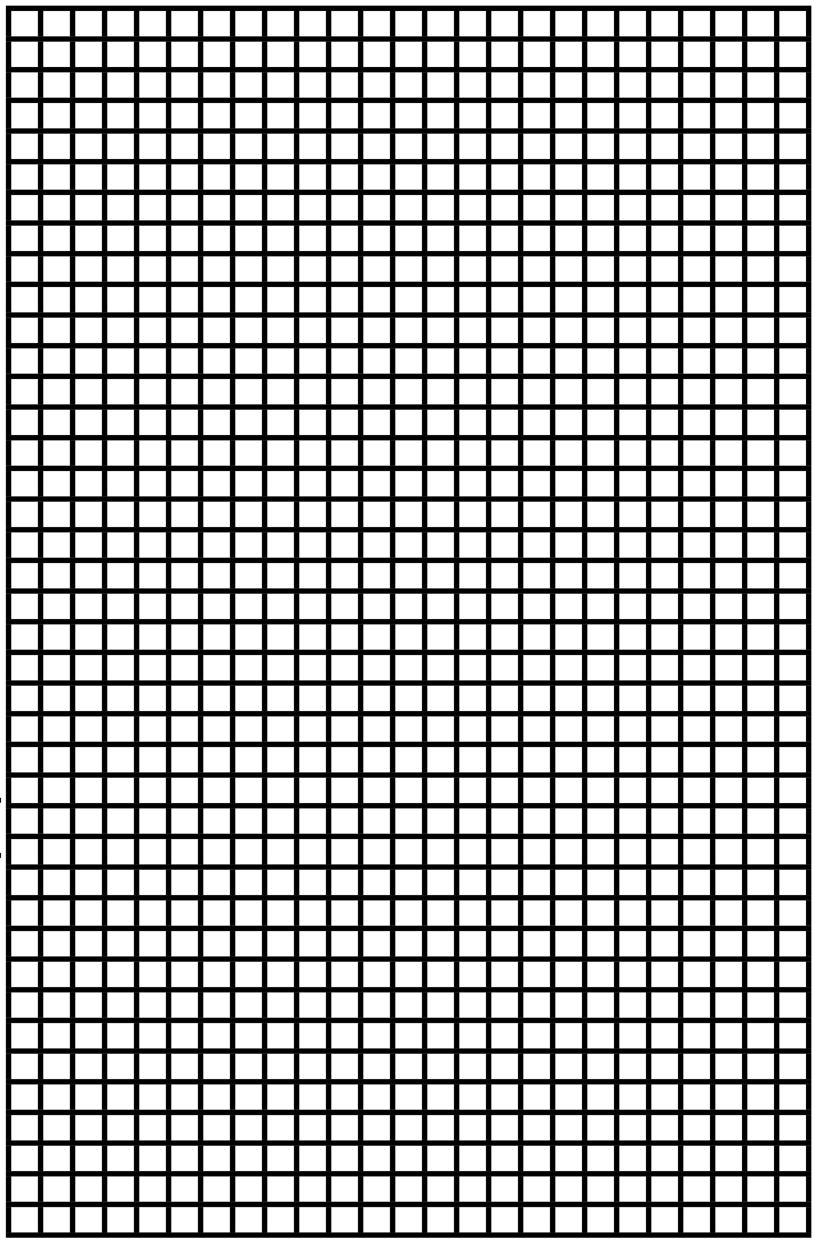


```
public void loadTile(int key, Drawable tile) {
        Bitmap bitmap = Bitmap.createBitmap(mTileSize, mTileSize,
                        Bitmap.Config.ARGB_4444);
        Canvas canvas = new Canvas(bitmap);
        tile.setBounds(0, 0, mTileSize, mTileSize);
        tile.draw(canvas);

        this.mTileArray[key] = bitmap;
}
```

The Grid

n*m array of ints

mTileGrid



Computing n & m

```
@Override
protected void onSizeChanged(int w, int h, int oldw, int oldh) {
    mXTileCount = (int) Math.floor(w / mTileSize);
    mYTileCount = (int) Math.floor(h / mTileSize);

    mXOffset = (w - mTileSize * mXTileCount) / 2;
    mYOffset = (h - mTileSize * mYTileCount) / 2;

    this.mTileGrid = new int[mXTileCount][mYTileCount];
    clearTiles();
}
```

onSizeChanged is a method in the view class

Drawing Grid

mTileGrid[n][m]

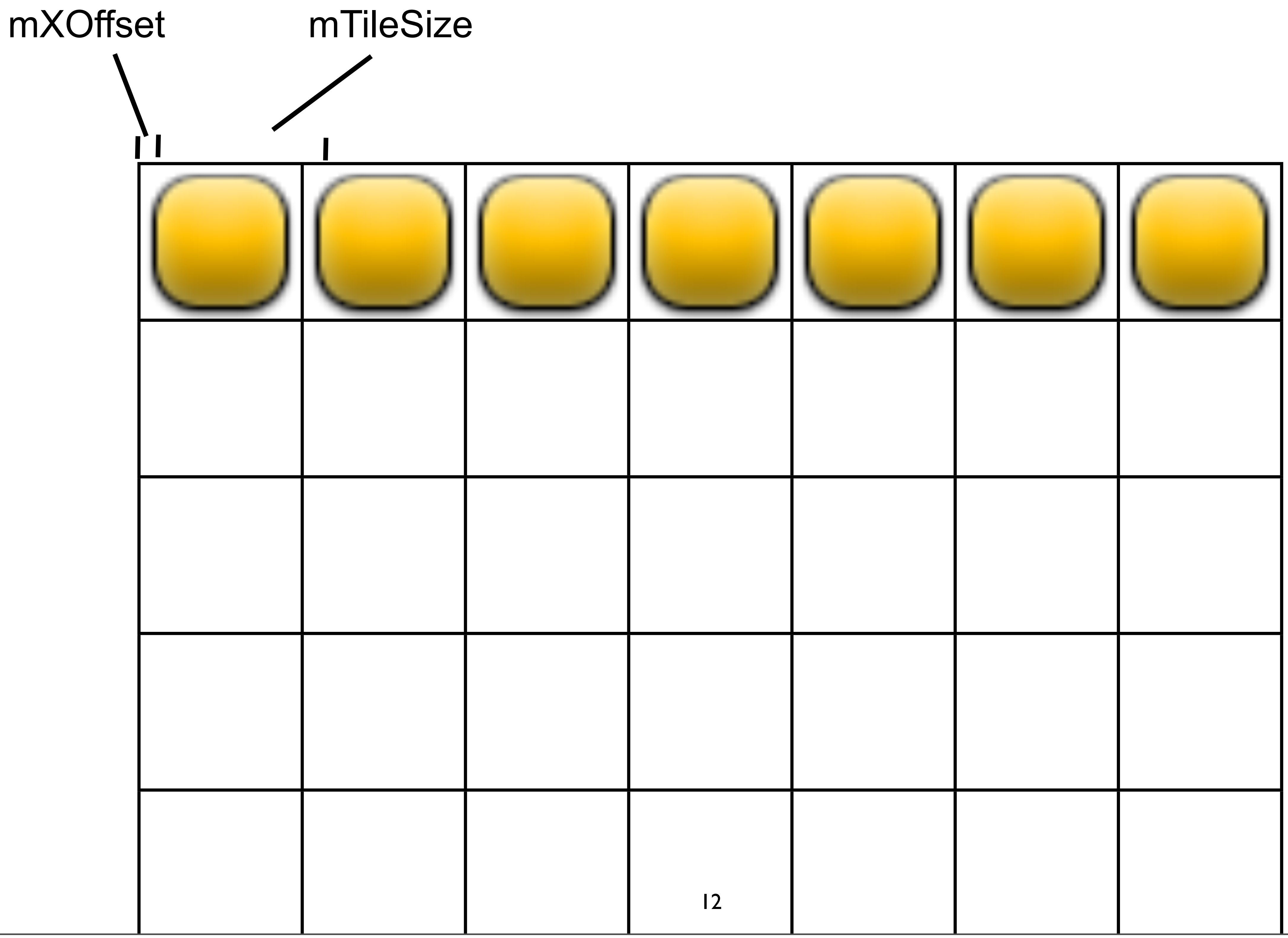
if 0 draw nothing

else draw mTileArray[this.mTileGrid[n][m]]

mTileArray[1]	🟥
mTileArray[2]	🟨
mTileArray[3]	🟩

```
public void onDraw(Canvas canvas) {
    super.onDraw(canvas);
    for (int x = 0; x < mXTileCount; x += 1) {
        for (int y = 0; y < mYTileCount; y += 1) {
            if (this.mTileGrid[x][y] > 0) {
                canvas.drawBitmap(this.mTileArray[this.mTileGrid[x][y]],
                        mXOffset + x * mTileSize, mYOffset + y * mTileSize,
                        this.mPaint);
            }
        }
    }
```

Where to Draw



The Snake

Just a list of points

```
private ArrayList<Coordinate> mSnakeTrail = new ArrayList<Coordinate>();
```

```
mSnakeTrail.add(new Coordinate(7, 7));
mSnakeTrail.add(new Coordinate(6, 7));
mSnakeTrail.add(new Coordinate(5, 7));
mSnakeTrail.add(new Coordinate(4, 7));
mSnakeTrail.add(new Coordinate(3, 7));
mSnakeTrail.add(new Coordinate(2, 7));
```

```
private class Coordinate {
    public int x;
    public int y;
```

How to Draw the board

mTileGrid holds what to draw at each location

Fill boundary with wall tiles

Fill snake locations

Fill apple locations

draw

```
private void updateWalls() {
    for (int x = 0; x < mXTileCount; x++) {
        setTile(GREEN_STAR, x, 0);
        setTile(GREEN_STAR, x, mYTileCount - 1);
    }
    for (int y = 1; y < mYTileCount - 1; y++) {
        setTile(GREEN_STAR, 0, y);
        setTile(GREEN_STAR, mXTileCount - 1, y);
    }
```

The outline

Draw all the game elements
Wait 600 milliseconds
Move the snake
Check for collisions
Draw all the game elements
etc.

How to wait?

```
private RefreshHandler mRedrawHandler = new RefreshHandler();


  class RefreshHandler extends Handler {

      @Override
      public void handleMessage(Message msg) {
          SnakeView.this.update();
          SnakeView.this.invalidate();
      }

      public void sleep(long delayMillis) {
              this.removeMessages(0);
          sendMessageDelayed(obtainMessage(0), delayMillis);
      }
```

update

```
public void update() {
    if (mMode == RUNNING) {
        long now = System.currentTimeMillis();

        if (now - mLastMove > mMoveDelay) {
            clearTiles();
            updateWalls();
            updateSnake();
            updateApples();
            mLastMove = now;
        }
        mRedrawHandler.sleep(mMoveDelay);
    }

}
```

How to force a redraw of a View

View methods

invalidate(int l, int t, int r, int b)

Mark the the area defined by the rect (l,t,r,b) as needing to be drawn.

invalidate()

Invalidate the whole view.

invalidate(Rect dirty)

Mark the the area defined by dirty as needing to be drawn.

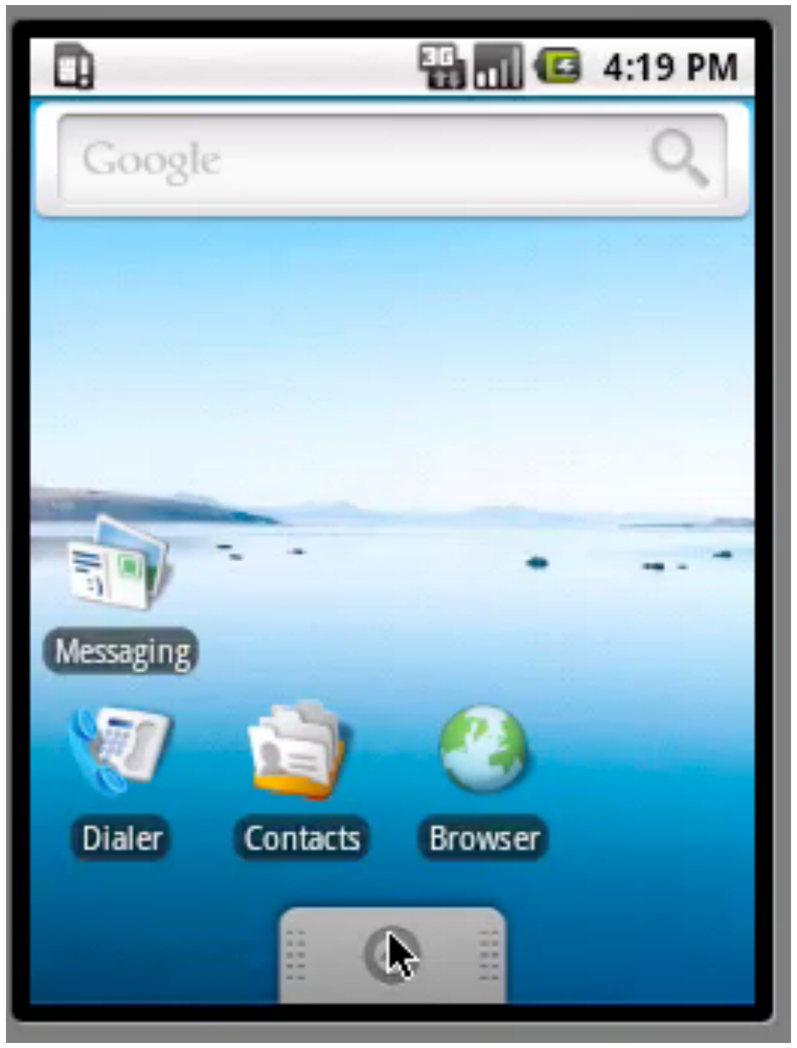
invalidateDrawable(Drawable drawable)

Invalidates the specified Drawable.

Don't forget onPause() etc.

```
protected void onPause() {
      super.onPause();
      mSnakeView.setMode(SnakeView.PAUSE);
  }
```

Finger Paint



Basic Idea

Current path - one user is currently drawing

- Store in a Path object as it is being created

Old paths - ones drawn before

- Store all in a bitmap

Basic Algorithm

```
On ACTION_DOWN (mouse down, user pressed)
    Create new Path
    Path starts where user pressed
    redraw Screen

On ACTION_MOVE
    Get current location
    Add "line" from old location to current location in path
    redraw screen

On ACTION_UP
    Finish path
    Add path to bitmap of previous paths
    Reset path
    redraw screen
```

How get thick red line

```
this.mPaint = new Paint();
        this.mPaint.setAntiAlias(true);
        this.mPaint.setDither(true);
        this.mPaint.setColor(0xFFFF0000);
        this.mPaint.setStyle(Paint.Style.STROKE);
        this.mPaint.setStrokeJoin(Paint.Join.ROUND);
        this.mPaint.setStrokeCap(Paint.Cap.ROUND);
        this.mPaint.setStrokeWidth(12);
```

Creating the Bitmap and Path

```
public MyView(Context c) {
        super(c);

        this.mBitmap = Bitmap.createBitmap(320, 480,
                Bitmap.Config.ARGB_8888);
        this.mCanvas = new Canvas(this.mBitmap);
        this.mPath = new Path();
        this.mBitmapPaint = new Paint(Paint.DITHER_FLAG);
    }
```

How do we get Motion Events

```
@Override
        public boolean onTouchEvent(MotionEvent event) {
                float x = event.getX();
                float y = event.getY();

                switch (event.getAction()) {
                case MotionEvent.ACTION_DOWN:
                        touch_start(x, y);
                        invalidate();
                        break;
                case MotionEvent.ACTION_MOVE:
                        touch_move(x, y);
                        invalidate();
                        break;
                case MotionEvent.ACTION_UP:
                        touch_up();
                        invalidate();
                        break;
                }
                return true;
        }
```

onTouchEvent is a method in the View class

Starting a path

```
private void touch_start(float x, float y) {
        this.mPath.reset();
        this.mPath.moveTo(x, y);
        this.mX = x;
        this.mY = y;
    }
```

Handling the Move

```
private void touch_move(float x, float y) {
        float dx = Math.abs(x - this.mX);
        float dy = Math.abs(y - this.mY);
        if (dx >= TOUCH_TOLERANCE || dy >= TOUCH_TOLERANCE) {
                this.mPath.quadTo(this.mX, this.mY, (x + this.mX) / 2,
                                (y + this.mY) / 2);
                this.mX = x;
                this.mY = y;
        }
}
```

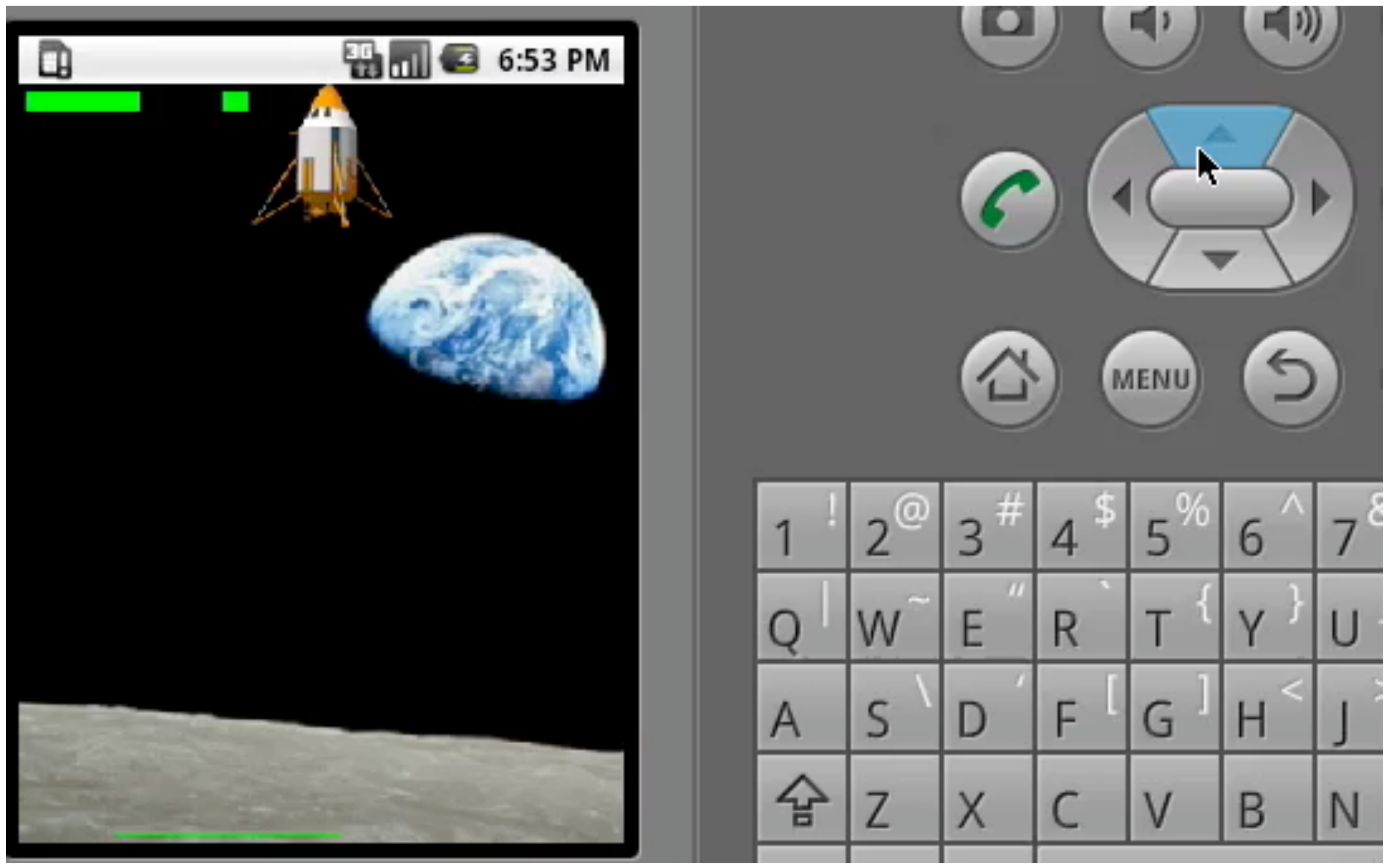
Ending the path

```
private void touch_up() {
        this.mPath.lineTo(this.mX, this.mY);

        // commit the path to our offscreen
        this.mCanvas.drawPath(this.mPath, FingerPaint.this.mPaint);

        // kill this so we don't double draw
        this.mPath.reset();
}
```

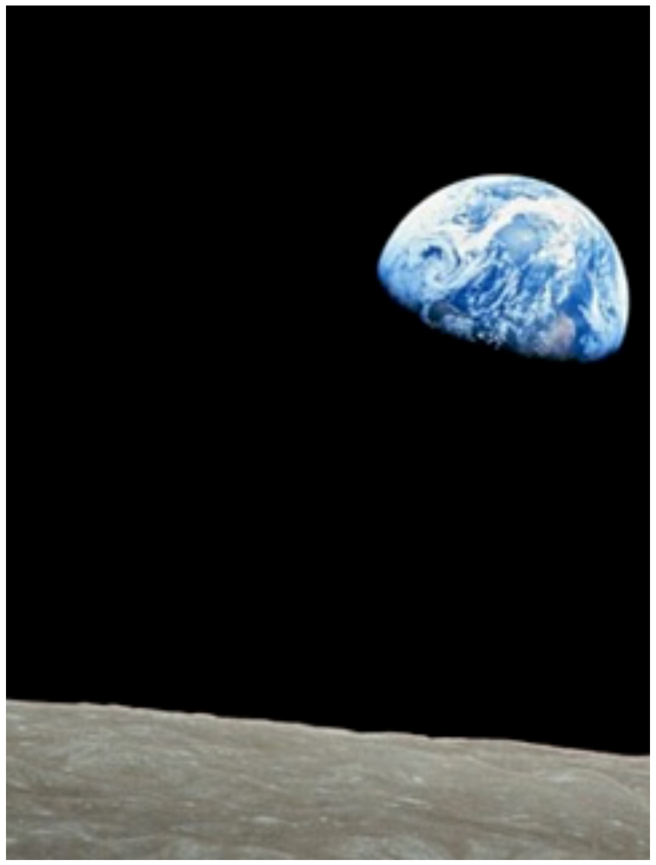
The Actual Drawing

```
protected void onDraw(Canvas canvas) {
	canvas.drawColor(0xFFAAAAAA);

	canvas.drawBitmap(this.mBitmap, 0, 0, this.mBitmapPaint);

	canvas.drawPath(this.mPath, FingerPaint.this.mPaint);
}
```

Lunar Lander



res/drawables



The Physics

We are in the physics building so I am sure some would be will to explain it

SurfaceView

For graphics needing separate thread

getHolder()
  Returns the surface holder for the view's surface

SurfaceHolder.Callback interface methods

surfaceChanged(...)
  Called when the surface dimensions change

surfaceCreated(SurfaceHolder holder)
  Called when surface is ready to use

surfaceDestroyed(SurfaceHolder holder)

SurfaceHolder

Manages the canvas that draws on SurfaceView bitmap

public abstract Canvas lockCanvas ()

    Returns canvas you can draw on in any thread

    Canvas is locked

public abstract void unlockCanvasAndPost (Canvas canvas)

    Contents of canvas are displayed on the screen

Basic Algorithm

```
public void run() {
    while (mRun) {
        Canvas c = null;
        try {
            c = mSurfaceHolder.lockCanvas(null);
            synchronized (mSurfaceHolder) {
                if (mMode == STATE_RUNNING) updatePhysics();
                doDraw(c);
            }
        } finally {
            if (c != null) {
                mSurfaceHolder.unlockCanvasAndPost(c);
            }
        }
    }
}
```

The rotation

```
canvas.save();
canvas.rotate((float) mHeading, (float) mX, mCanvasHeight - (float) mY);
if (mMode == STATE_LOSE) {
    mCrashedImage.setBounds(xLeft, yTop, xLeft + mLanderWidth, yTop + mLanderHeight);
    mCrashedImage.draw(canvas);
} else if (mEngineFiring) {
    mFiringImage.setBounds(xLeft, yTop, xLeft + mLanderWidth, yTop + mLanderHeight);
    mFiringImage.draw(canvas);
} else {
    mLanderImage.setBounds(xLeft, yTop, xLeft + mLanderWidth, yTop + mLanderHeight);
    mLanderImage.draw(canvas);
}
canvas.restore();
```