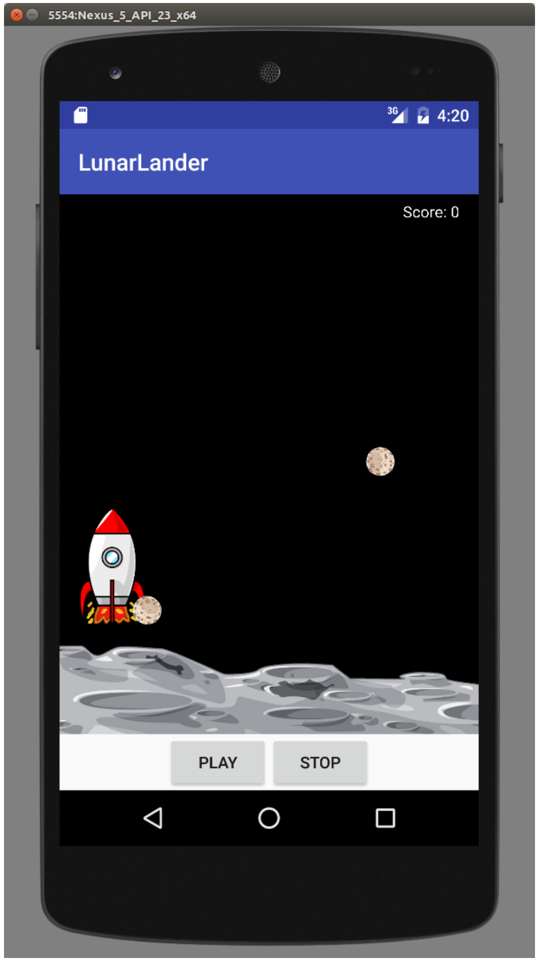
CS 193A

Making Basic 2D Games

Lunar Lander game

- Let's write a **lunar lander** game.
  - Rocket falls with 4 initial velocity, 0.5 downward acceleration.
  - Touch screen and hold to thrust; when thrusting, rocket accelerates upward at 0.3 acceleration.
  - Once per second, an asteroid appears at right edge of screen, going left with 12 velocity. If it hits rocket, game is over.
  - Player earns 1 point per second alive.
  - If rocket can touch the bottom of the screen with a velocity of 7 or less, player wins.



A basic animation loop

- The code to animate a view must do the following in a loop:
  - 1) process any **user input** (mouse touch events, key presses, etc.)
  - 2) **update** the game state (move any sprites, handle collisions, etc.)
  - 3) tell the view to **redraw** itself (which happens on the main UI thread)
  - 4) **pause** for some number of milliseconds

```
// game animation loop pseudo-code
function myAnimationLoop():
    while true:
        1) process user input
        2) update game's state
        3) tell view to redraw self on main UI thread
        4) pause/sleep for some number of MS
```

Animation loop in GCanvas

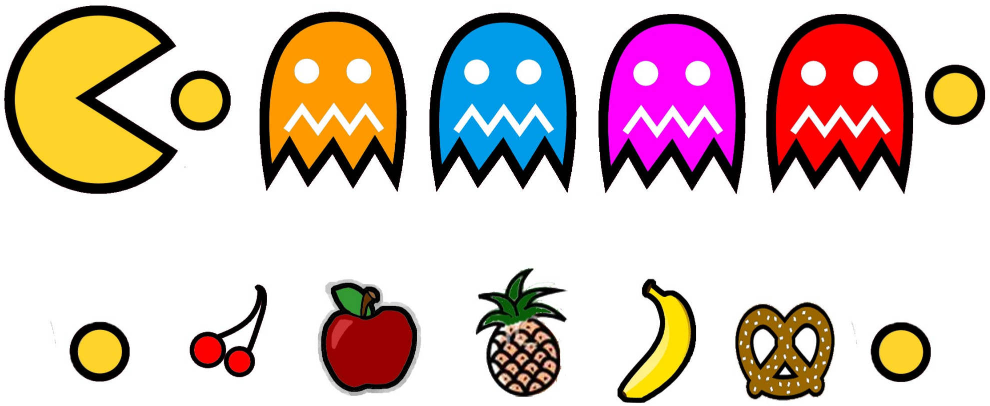
- The library's `GCanvas` has an `animate` method to start animation.
  - It will call your `onAnimateTick` method once per frame.
  - It contains the while loop and the part to invalidate the view.
  - Any `GSprites` in the `GCanvas` will also automatically move themselves.
  - You can just focus on the code to **update the game state**.

```
// in MyCanvas.java
animate(20);   // 20 fps = 50ms
...

// called once per frame of animation
public void onAnimateTick() {
    // process user input
    // update your game's state
    my game update code goes here;
}
```

A Sprite class

- **sprite**: An object of interest in a game.
  - possible data: location, size, velocity, shape/image, points, ...
  - Many games declare some kind of Sprite class to represent the sprites.
  - Useful sprite operations: drawing, movement, visibility, collisions
  - See Stanford library class: `GSprite`

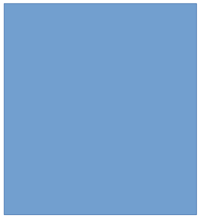
```
// an example sprite class
public class Sprite {
    float x, y, w, h;
    float dx, dy;
    Paint paint;
    ...
}
```



Sprite drawing code

- Sprites can contain code to draw themselves.
  - Game's onDraw tells each sprite to draw itself in a loop.

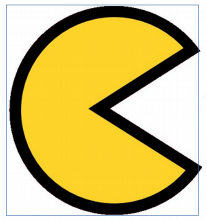
```
public class Sprite {
    float x, y, w, h;
    float dx, dy;
    Paint paint;
    ...
    public void draw(Canvas canvas) {
        canvas.drawRect(x, y, x+w, y+h, paint);
    }
}
```



Sprite with bitmap

- Most games draw their sprites as bitmap images.
  - `GSprite`: setBitmap, constructor

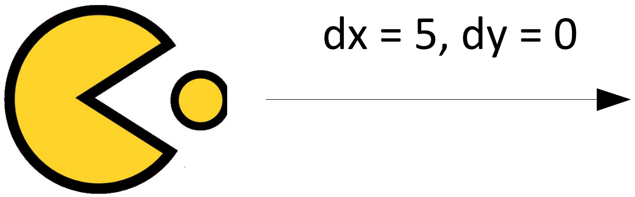
```
public class Sprite {
    float x, y, w, h;
    float dx, dy;
    Bitmap bitmap;
    ...

    public void draw(Canvas canvas) {
        canvas.drawBitmap(bitmap, x, y, paint);
    }
}
```



Moving Sprites

- One way to do movement: Have each sprite store a **velocity**.
  - dx/dy pair, or write a simple 2D Vector class
  - usually write a simple method to tell the sprite to move/update itself
    - this method is called once for each frame in your game's animation loop
  - `GSprite`: setVelocity, rotateVelocity

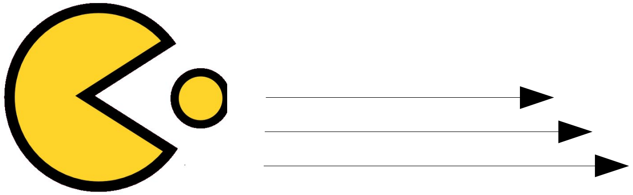
```
public class Sprite {
    float x, y, w, h;
    float dx, dy;
    ...
    public void move() {
        x += dx;
        y += dy;
    }
}
```



Acceleration

- More advanced: You may want to apply **acceleration** to movement.
  - could be a single value (scale up/down), separate ax/ay, or another 2D vector
  - updates velocity when object moves
    - need to watch out for **sign issues** if velocity components are negative
    - GSprite: setAcceleration

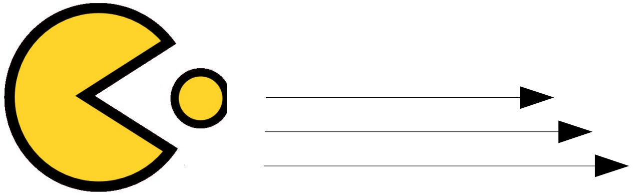
```
public class Sprite {
    float x, y, w, h;
    float dx, dy;
    float ax, ay;
    ...
    public void move() {
        x += dx;
        y += dy;
        dx *= (1.0 + ax);  // accelerate
        dy *= (1.0 + ay);
    }
}
```



Frame rate independence

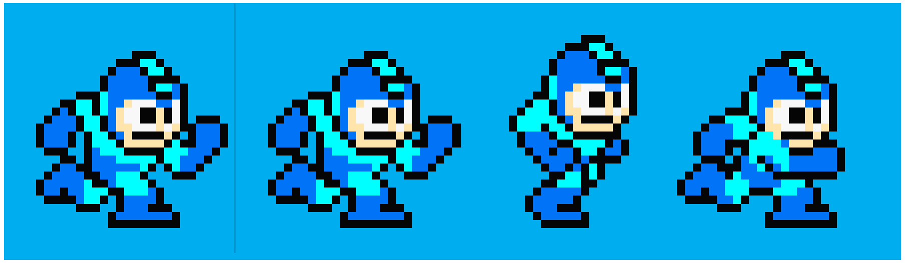
- You may want to move objects at a constant speed regardless of the number of frames/sec used in your animation.
  - To do this, allow sprites to see FPS and weight their velocity accordingly.
  - Now dx/dy means change per second, not per frame of animation.
  - Advanced: On slower/older devices, can implement a *frame skip* .

```
public class Sprite {
    float x, y, w, h;
    float dx, dy;
    ...
    public void move(int fps) {
        x += dx / fps;
        y += dy / fps;
    }
}
```



Sprite animation / walk cycle

- Make the sprite change image as it animates (a "walk cycle"):
  - Store a list of bitmap images to display.
    - Cycle through the images in the list by remembering a current index.
  - Don't change images every frame; this will be too fast.
  - List of bitmaps may change based on game events.
    - Example: Change direction; get shot; get powerup.
  - `GSprite`: setBitmaps, setFramesPerBitmap



Sprite with walk cycle

```
public class Sprite {
    ArrayList<Bitmap> bitmaps;
    int bitmapIndex = 0;
    int frame = 0;
    int framesPerBitmap = 10;
    ...
    public void move() {
        frame++;
        if (frame % framesPerBitmap == 0) {
            // move to next bitmap in cycle
            bitmapIndex = (bitmapIndex + 1) % bitmaps.size();
        }
    }
    public void draw(Canvas canvas) {
        canvas.drawBitmap(bitmaps.get(bitmapIndex), x, y,
                          null);
    }
}
```

Image strips

- Loading lots of small images can be slow.

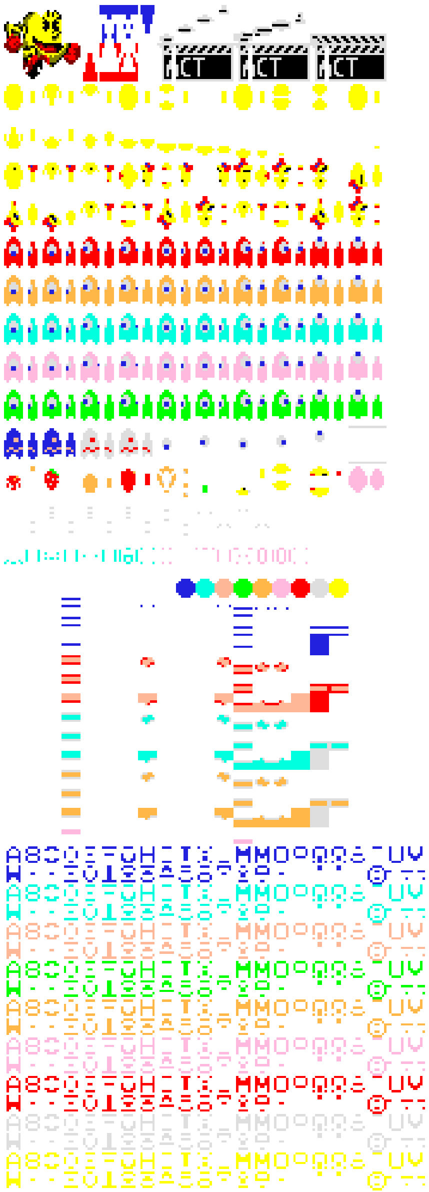
- **image strip**: Many images in one large file.
  - Can load it just once and then chop it apart.
  - Code is a bit more complex,
    but load/run time is very fast.

- In Android's `Bitmap` class:

```
public Bitmap createBitmap(
    Bitmap source,
    int x, int y,
    int width, int height)
```

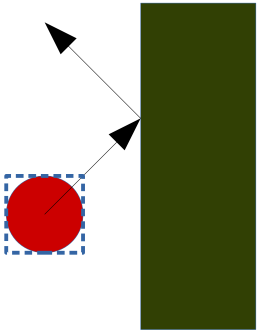
  - Extracts the given sub-range of pixels of this bitmap as its own Bitmap and returns it.

pacman-strip.png

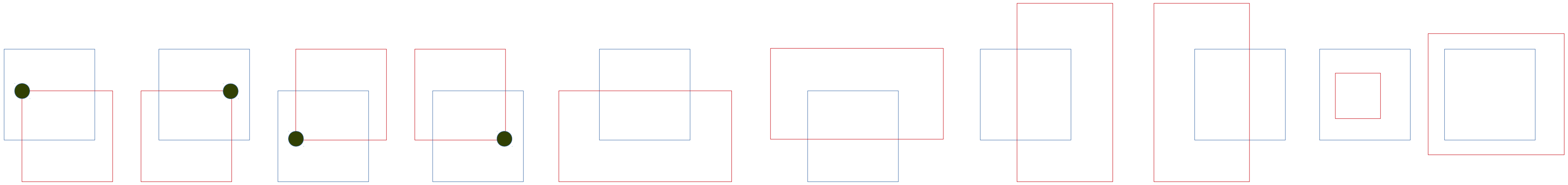


Collision detection

- **collision detection**: Determining whether sprites in the game world are touching each other (and reacting accordingly).
  - You can calculate whether two sprites have collided by seeing whether their bounds overlap.



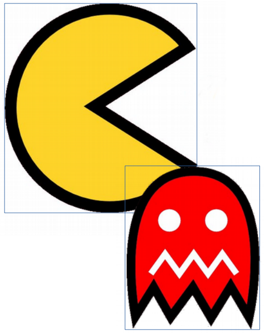
- Android's `RectF` (link) and other shapes have methods to check whether they touch:
  - ***`rect1`***`.contains(`***`x`***`, `***`y`***`)`
  - ***`rect1`***`.contains(`***`rect2`***`)`
  - `RectF.intersects(`***`rect1`***`, `***`rect2`***`)`
  - `GSprite`: collidesWith



Sprite with collision detection

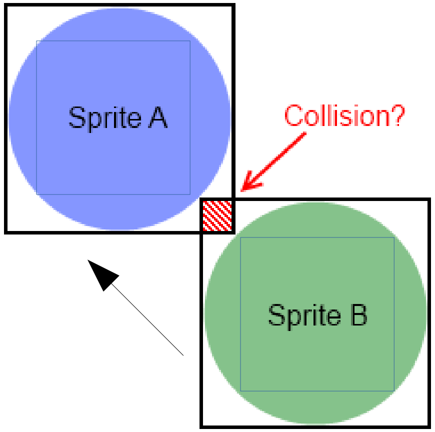
- Suggested: Have sprite represent its bounds as a rectangle.
  - The rectangle object will know if it hits another sprite.

```
public class Sprite {
    RectF bounds;
    float dx, dy;
    ...
    public void move() {
        bounds.left += dx;
        bounds.top += dy;
    }

    public boolean collides(Sprite other) {
        return RectF.intersects(bounds, other.bounds);
    }
}
```

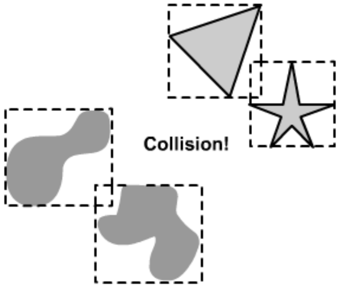


Collision margin

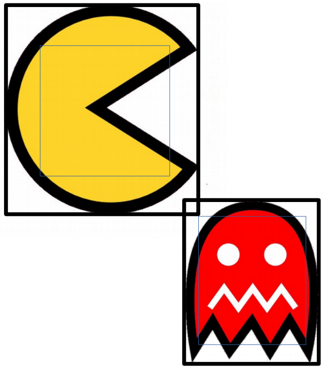
- Collisions are harder to compute for non-rectangular sprites.
  - Don't want the empty edges to collide.
  - Even for rectangular shapes, it can be preferable to have a bit of collision "slack".



- Some games use a **collision rectangle** smaller than the overall bounding box to give the collisions a bit of lenience.



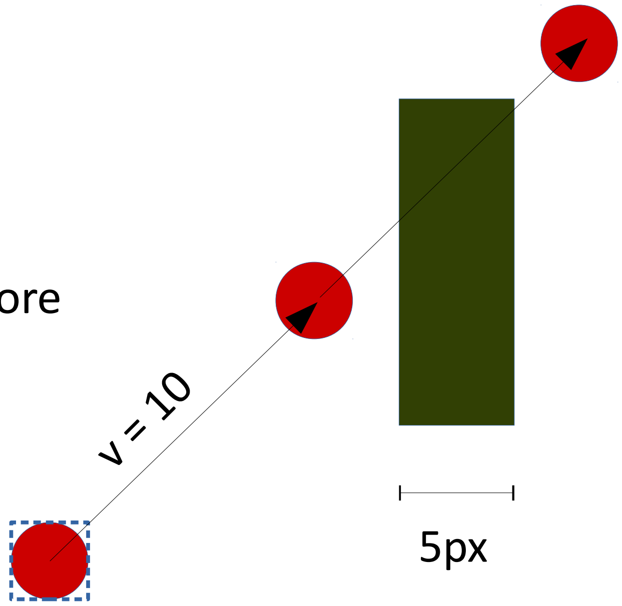
  - `GSprite`: setCollisionMargin(...)



Common collision bug

- When an object moves at high velocity, it may wrongly "jump through" a sprite it ought to collide with.

- Several possible **fixes** to this issue:
  - perform more/smaller "updates" per frame of animation to effectively reduce velocity
    - e.g. update (move 5px), update (move 5px), redraw;
    - works well with FPS-independent movement shown before
  - temporarily enlarge collision rectangles for some fast-moving sprites
    - enlarge; do collision detection; shrink
  - use a proper physics engine, vectors, etc. for game movement
    - most professional-quality game engines will help you address this bug in some way



GSprite methods

Method	Description
GSprite(...)	initializes a new sprite with given shape, coords
canvas: addTo(GCanvas), remove()	adds/removes sprite from screen
collisions: collidesWith(sprite), is/setCollidable(), get/setCollisionMarginTop/Bottom/Left/Right/X/Y()	returns true if two sprites touch each other; sets collision margin to reduce collisions
location: get/setX/Y, get/setLocation(), bound(), boundHorizontal/Vertical(), isInBounds(), moveTo(x, y), translate(dx, dy)	where the sprite is on the screen
velocity: get/setVelocity, flipVelocity/X/Y, isMoving, moveBy(dx, dy)	whether the sprite will move on each update()
acceleration: get/setAcceleration/X/Y()	whether velocity will change on each update()
bitmaps: get/setBitmaps, scale, get/setFramesPerBitmap, isSetLoopBitmaps	images to draw on the sprite; if multiple images are passed, can cycle between them/animate
extras: get/setExtra(s), hasExtra, clearExtras	stuff "extra" data inside a sprite for convenience
size: get/setWidth/Height/Size/Bounds	size of sprite on screen
color: get/setColor, Paint, get/setFillColor, is/setFilled	color for sprites that are shapes

Different screen sizes

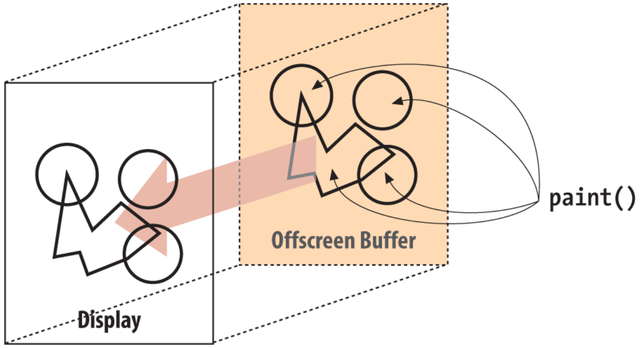
- Android devices come in a variety of screen sizes and shapes.
  - Your game should run on a variety of device sizes.

- Some ways to handle device sizing:
  - **Scale** your bitmaps/coordinates relative to getWidth(), getHeight()

    Bitmap.createScaledBitmap(***bitmap***, ***width***, ***height***)

  - Draw onto a **backing buffer** and then scale the buffer to fit the screen

    Bitmap.createBitmap(***width***, ***height***,
    Bitmap.Config.ARGB_8888)



SimpleBitmap methods

Method	Description
SimpleBitmap.with(*activity*)	get a SimpleBitmap instance

get(*id*)
get(*id, width, height*)
	load a bitmap from a resource ID (possibly scaling it)

getAll(*ids*)	load a list of bitmaps from resource IDs
rotate(*bmp or id, degrees*) rotate(*bmp or id, degrees, rx, ry*)	rotate counter-clockwise about a given point

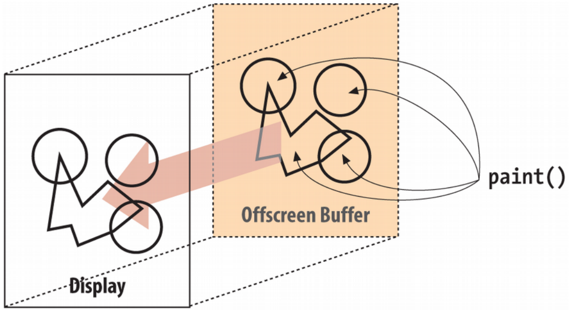
rotateLeft(*bmp or id*)
rotateRight(*bmp or id*)
	rotate 90 degrees counter-clockwise or clockwise

scale(*bmp or id, scaleFactor*) scale(*bmp or id, sfX, sfY*)	scale size of bitmap by given ratio
scaleToFit(*bmp or id, canvas*)	scale size of bitmap to largest size to fit within given canvas (maintain aspect ratio)
scaleToWidth(*bmp or id, width*) scaleToHeight(*bmp or id, height*)	scale size of bitmap to given width or height (maintain aspect ratio)
setFiltered(*bool*)	whether to anti-alias/smooth pixels (default true)

Double buffering

- **double buffering**: Drawing all individual shapes/sprites onto an auxiliary image first, then drawing that image onto the screen.

```
// hypothetical code to draw onto buffer
Bitmap bmp = Bitmap.createBitmap(
    width, height, Bitmap.Config.ARGB_8888);
Canvas bmpCanvas = new Canvas(bmp);
for (Sprite sprite : mySprites) {
    sprite.draw(bmpCanvas);
}
...

protected void onDraw(Canvas canvas) {
    // scale the buffer and draw it onto the screen
    Rect src = new Rect(0, 0, bmp.getWidth(), bmp.getHeight());
    RectF dst = new RectF(0, 0, getWidth(), getHeight());
    canvas.drawBitmap(bmp, src, dst, /* paint */ null);
```



"Mouse" (touch) events

- **old mouse events**: a physical mouse attached to a device
  - *types:* button press, release; cursor moved; drag; enter/exit; hover

- **new touch events**: a finger touching the screen
  - *types:* button press, release; drag
  - mouse movement, hovering largely absent
  - multi-touch input now possible (not covered today)

Mouse touch events (link)

- To handle finger presses from the user, write an onTouch method in your GCanvas or custom View class.
  - actions: ACTION_DOWN, ACTION_UP, ACTION_MOVE, ...
  - caution: don't write confusingly-similar onTouchEvent method

```
@Override
public boolean onTouch(View v, MotionEvent event) {
    float x = event.getX();
    float y = event.getY();
    if (event.getAction() == MotionEvent.ACTION_DOWN) {
        // code to run when finger is pressed

    }

    return super.onTouch(v, event);
}
```

Mouse event handling

- Typically you don't draw sprites or handle much in a mouse event handler.
  - Instead, **remember** the user 's action; use it in your next animation "tick" frame update.

```
private GSprite car;
...

public boolean onTouchEvent(MotionEvent event) {
    float x = event.getX();
    if (event.getAction() == MotionEvent.ACTION_DOWN) {   // finger press
        if (x < getWidth() / 2) {
            car.setVelocityX(-10);   // will move left
        } else {
            car.setVelocityX(10);    // will move right
        }
    } else if (event.getAction() == MotionEvent.ACTION_UP) { // finger lift
        car.setVelocityX(0);
    }
    return super.onTouchEvent(event);
}
```

Keyboard events

- Most Android devices do not have physical keyboards!
  - If they do, it's likely an external device and not always attached.
  - Onscreen keyboard generates events, but it's flaky and usually hidden.

- Therefore, no app should ever use keyboard input for a critical part of its UI.
  - Should always provide a mouse/touch / other method of input.
  - Keyboard events are mostly for testing on a PC or dev machine.

Keyboard events (link)

If you want to handle key presses (if the device has a keyboard):

- set your app to receive keyboard "focus" in View constructor:

```
requestFocus();
setFocusableInTouchMode(true);
```

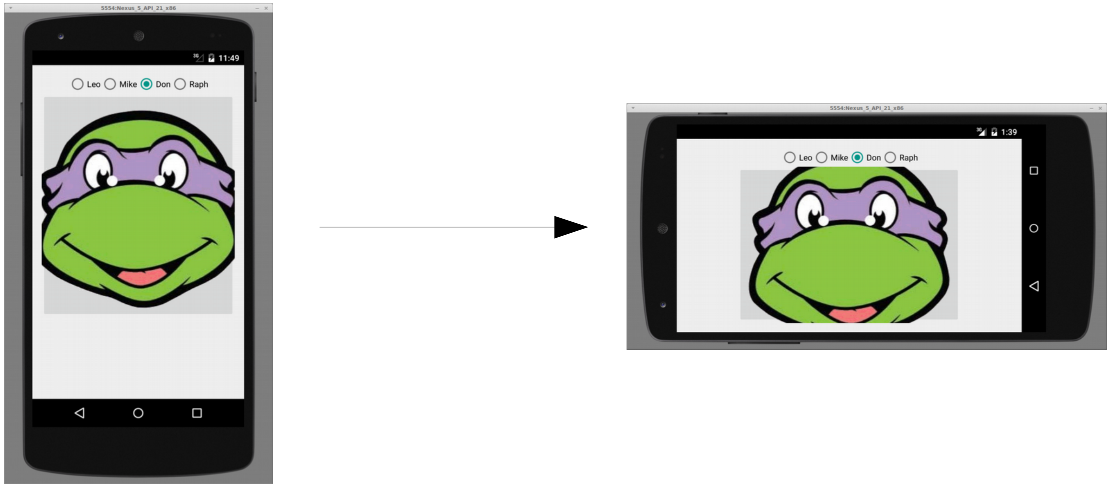
- write onKeyDown/Up methods in your custom View class.
  - each key has a "code" such as `KeyEvent.KEYCODE_ENTER`
  - *or call setOnKeyListener and pass an OnKeyListener*

```
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
    if (keyCode == KeyEvent.KEYCODE_X) {
        // code to run when user presses the X key
    }
    return super.onKeyDown(keyCode, event);
}
```

Recall: Keeping state on orientation

- By default, rotating your app nukes your activity, reloads it, and loses any unsaved instance state.
  - e.g. private fields, some GUI widget state information

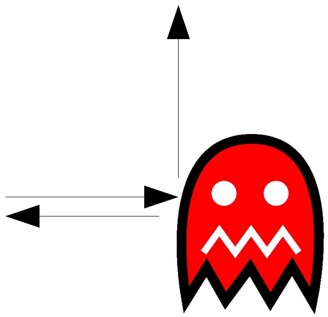
```
<!-- add the following in AndroidManifest.xml -->
<activity android:name=".MainActivity"
    android:configChanges="orientation|screenSize"
    ...>
```



○ Leo ○ Mike ● Don ○ Raph

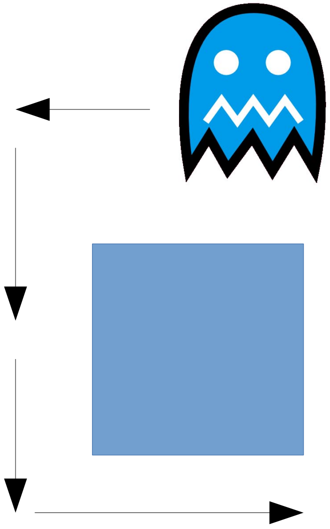
Enemy AI

- Many games have enemy characters with some kind of behavior or AI.
  - Can be implemented in several ways.



- **strategy pattern**: Making small objects to represent different AI styles/strategies.

```
public class AggressiveStrategy extends GhostStrategy {
    public void decideMove() { ... }
}
...
myGhost.setStrategy(new AggressiveStrategy());
```



OpenGL ES (link)

- **OpenGL**: Open standard graphics package supported on many computers and devices. Commonly used for 3D graphics.
  - hardware-accelerated (uses GPU, not CPU)
  - widely supported, freely available

- Android devices include a version of OpenGL for embedded systems ("ES").
  - public class MyGLRenderer implements GLSurfaceView.Renderer { ...

- Used to provide big performance boost, even in 2D.
  - In more recent Android versions, standard Canvas uses OpenGL, too.
  - Now Canvas is often faster than explicitly using OpenGL ES for 2D.
  - Therefore, OpenGL ES is not covered here.

Android Game Libraries

- **Unity**: Popular cross-platform game library.
  - http://unity3d.com/
  - (comprehensive, deploys to many platforms)
- **libgdx**: Another cross-platform game lib based on OpenGL.
  - https://github.com/libgdx/libgdx/
  - (in my opinion, a bit bare)
- **Google Play Games Services**: Set of libraries made by Google for social gaming features.
  - Achievements, High Scores/Leaderboard, Network Multiplayer
  - https://developers.google.com/games/services/

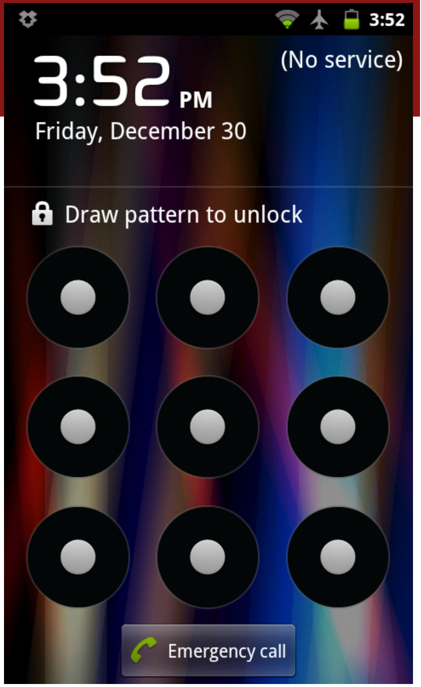
Pros/cons of game frameworks

- You don't even code in Java!
  - Cocos2d-x: C++, JavaScript
  - Unity: C# and others
  - Skia: Python
  - Xamarin: C#

- Have their own editing software (not Android Studio)

- Many are cross-platform and deploy (mostly) the same game code to multiple platforms
  - code game once and deploy it on web, Android, iOS, etc.

WakeLock



- To prevent screen from blanking, use a **wake lock**.
- in `AndroidManifest.xml`:

```
<uses-permission
 android:name="android.permission.WAKE_LOCK" />
```

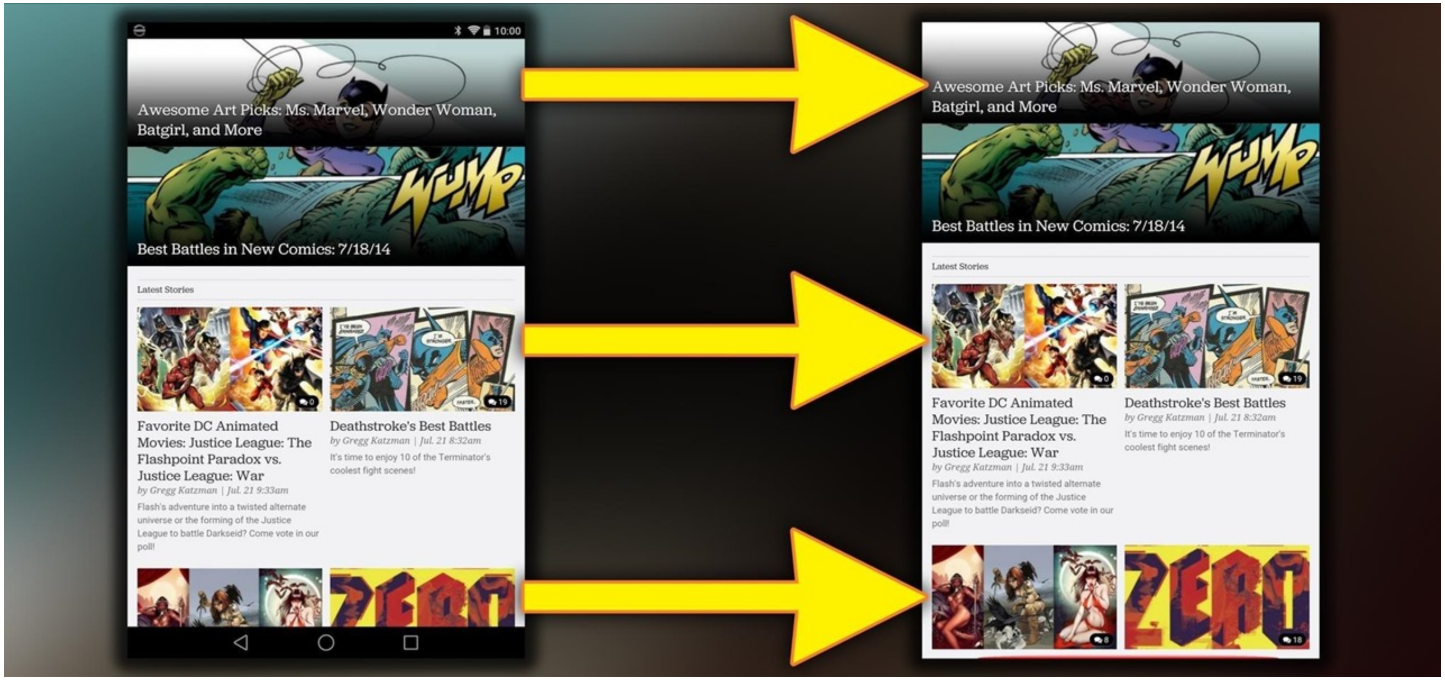
- in app's activity Java code:

```
// create the lock (probably in onCreate)
PowerManager pwr = (PowerManager) getSystemService(POWER_SERVICE);
PowerManager.WakeLock lock =
             pwr.newWakeLock(PowerManager.PARTIAL_WAKE_LOCK,
                             "my lock");

// turn on the lock (in onResume)
lock.acquire();

// turn off the lock (in onPause)
lock.release();
```

Full screen mode

- To put an app (e.g. a game) into full screen mode, which hides the notifications and status bar, put the following in your activity's onCreate method:

```
requestWindowFeature(Window.FEATURE_NO_TITLE);
getWindow().setFlags(
        WindowManager.LayoutParams.FLAG_FULLSCREEN,
        WindowManager.LayoutParams.FLAG_FULLSCREEN);
```



Awesome Art Picks: Ms. Marvel, Wonder Woman, Batgirl, and More

Best Battles in New Comics: 7/18/14

Latest Stories

Favorite DC Animated Movies: Justice League: The Flashpoint Paradox vs. Justice League: War

by Gregg Katzman | Jul. 21 9:33am

Flash's adventure into a twisted alternate universe or the forming of the Justice League to battle Darkseid? Come vote in our poll!

Deathstroke's Best Battles

by Gregg Katzman | Jul. 21 8:32am

It's time to enjoy 10 of the Terminator's coolest fight scenes!

Awesome Art Picks: Ms. Marvel, Wonder Woman, Batgirl, and More

Best Battles in New Comics: 7/18/14

Latest Stories

Favorite DC Animated Movies: Justice League: The Flashpoint Paradox vs. Justice League: War

by Gregg Katzman | Jul. 21 9:33am

Flash's adventure into a twisted alternate universe or the forming of the Justice League to battle Darkseid? Come vote in our poll!

Deathstroke's Best Battles

by Gregg Katzman | Jul. 21 8:32am

It's time to enjoy 10 of the Terminator's coolest fight scenes!

SimpleActivity game methods

Method	Description
`setWakeLock(`***boolean***`);`	set whether wake lock should be on/off
`wakeLockIsEnabled()`	returns true if you called setWakeLock(true); before
`setFullScreenMode(`***boolean***`);`	set whether app should go into full screen mode