CGT 353: Principles of Interactive and Dynamic Media
Intro to Flash Gaming and Basic Physics

Many Types of Games You Can Create:

- **Action** – ex. Space Invaders, Half-Life
- **Adventure –** action + story (different from RPG in that player actions do not affect characters overall abilities) – ex. Super Mario Bros
- **Casino –** gambling games – ex. Poker, Roulette
- **Educational** – learn something as you go
- **First-Person Shooter**
- **Puzzle**
- **Sports**
- **Role Playing Game (RPG) –** main distinction is that character attributes change as you play the game
- **Strategy** – try to build or run something – ex. Sim City

Game Views:

- 3D
- Chase
- First person
- Isometric
- Side
- Third Person
- Top Down

General Terminology:

- **Algorithm -** logical process by which a problem can be solved or a decision made
- **Artificial Intelligence -** set of algorithms that can make decisions in a logical way
- **Avatar -** graphical representations of people in a game or chat room
- **Collision detection -** also called a hit detection
- **Collision reaction -** what happens after a collision has been detected
- **Console -** computer designed for the sole purpose of playing video games
- **Map -** area that defines the world of the game
- **Real-time -**

- **Render -** process of drawing an object to the screen
- **Source Code -** original work created by the developer
- **Turn-based -** the restriction on which the player can make a move (you have to wait for your turn)
- **Vector graphics -** duh…
- **World -** the environment of the game

Flash Pros in Gaming:

1. Web Deployment
2. Small File size
3. Plug-in Penetration
4. Server-side integration
5. File sharing between programmer and designers
6. Ease of Use

Flash Cons in Gaming:

1. Performance
2. Lack of 3D Support
3. Lack of Operating System Integration

Difficult Game Features:

3D Games:

- True 3D Flash games exist, but not as common as 2D games

- **Most aren't true 3D** - Usually involve a "rig" that simulates 3D or other technology such as the Shockwave player

Three basic limitations to 3D engines in Flash gaming:

1. **Texture mapping** - cannot map textures well in Flash
2. **Z-sorting** - limited to sorting at the movie-clip level
3. **Speed** - usually can only handle simple shapes

Some "3D" Flash Game Resources:

- http://www.gameshot.org/?search=3D&cat=games
- http://www.gskinner.com/games/puki
- http://www.albinoblacksheep.com/games/3d
- http://www.dabontv.com/3dgames.html
- http://www.desq.co.uk/braincell/braincell.htm

Real-Time Multiplayer Games:

- Possible... but much more challenging.
- Requires the use of a real-time interactive media server.
- Becoming more common....
- Graphics have to be relatively simple....

Intense Real-Time Calculation:

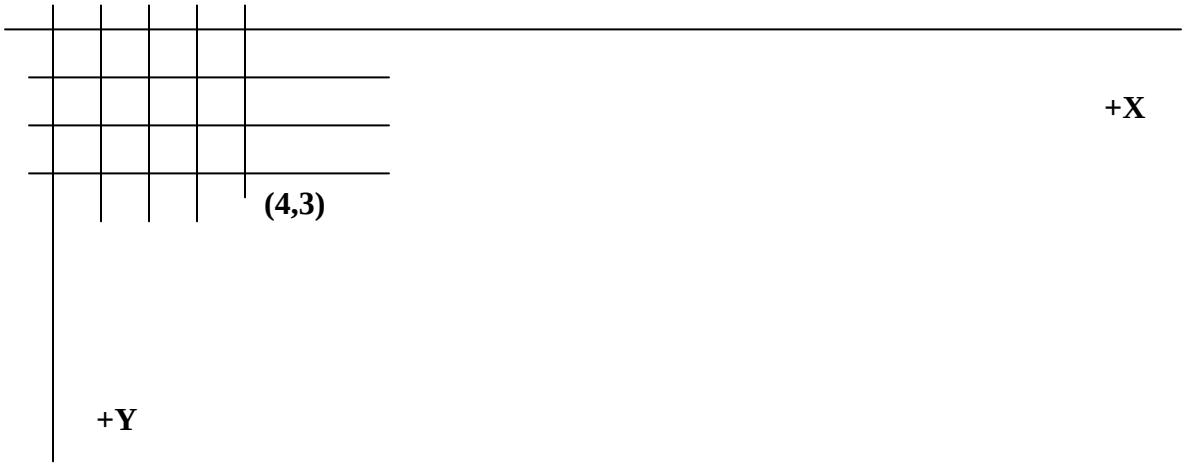
- Limitations to Flash player as a Web plugin....

Game Mathematics:

- Math is vital to creating games in Flash, specifically trigonometry....
- Useful for:
  - Rotating objects
  - Calculating trajectories
  - Collision detection and reaction

The Flash Coordinate System:

- Uses a version of the Cartesian Coordinate System



- The **origin** of every Flash movie is in the upper left-hand corner, and the registration point of every movie clip is it's own separate origin...

Angles:

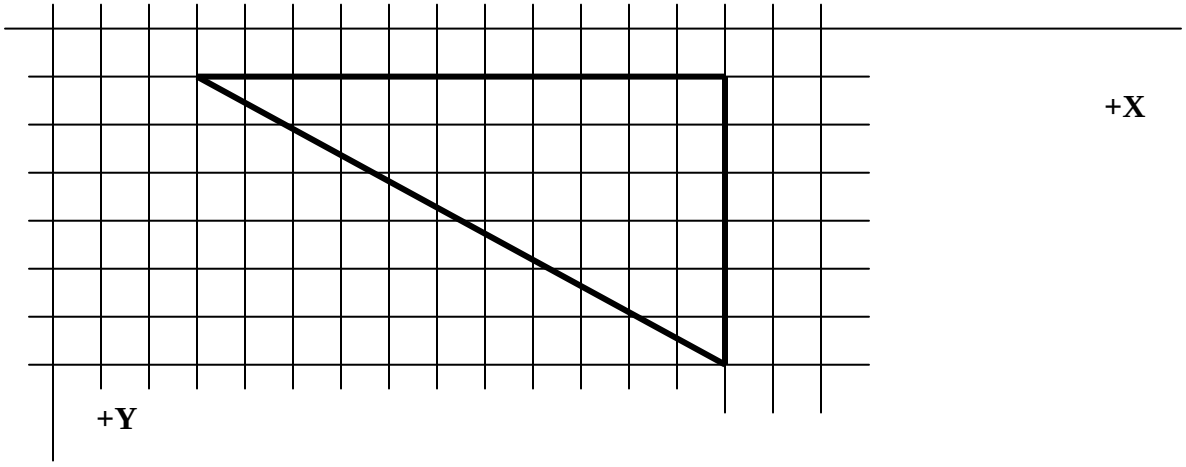
- Of course, the ability to calculate angles is vital...
- Must be measured in **radians** rather than degrees in Flash....
- Only time you use degrees directly is when you're changing the _rotation property of a movie clip...
- Can work with degree in ActionScript but have to be converted to radians:

angle in radians = angle in degrees * (Math.PI/180)

Using a Triangle:

- Sounds simple, but vital to gaming…
- Triangles made of three lines joined by three **vertices.**
- Three angles of a triangle must always equal 180 degrees or PI radians.
- With a right triangle, you can calculate the hypotenuse of a triangle using the Pythagorean theorem ($a^2 + b^2 = c^2$)
- By doing so, you can calculate the distance between two points.

$$\mathbf{c = distance = square\ root\ of\ (x_2-x_1)^2+(y_2-y_1)^2}$$



Or in ActionScript:

```
var Distance:Number = Math.sqrt((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1));
```

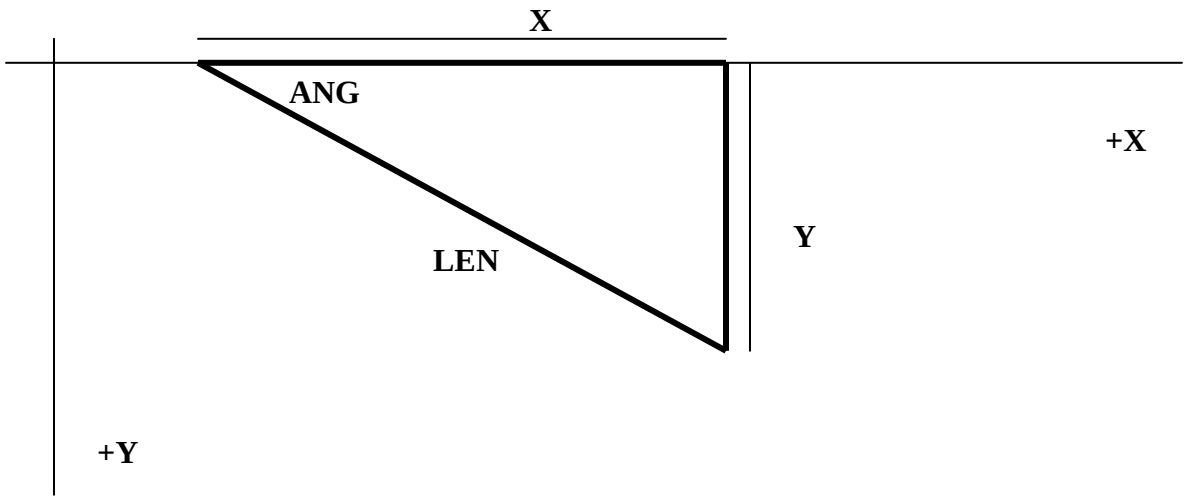
Sine, Cosine, and Tangent:

- Trigonometric functions that use various ratios of triangle side lengths to give results
- Found in the Math class

```
var angle:Number = 45;
var radians:Number = angle*Math.PI/180;
trace (Math.sin(radians));
```

Projection:

- **Projection** refers to the methods of projecting a quantity such as distance or velocity onto the x or y axis



X = LEN*COS(ANG)

or with AS:

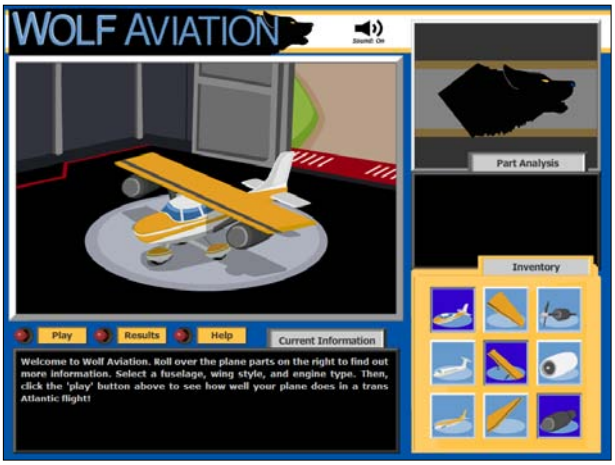
var x:Number = len*Math.cos(ang);

(See Shooter.swf)

Basic Physics:

- Physical properties such as speed velocity, and acceleration are vital to programming games...

- Book makes an excellent point, which is the difference between **real physics and "good-enough" physics...**



Speed and Velocity:

- A **vector** is a mathematical object that has both a) magnitude (numeric value) and b) direction...

- Speed is normally measured in units of distance/ time, but in Flash we use units/frames...

- Velocity is a vector, whereas speed is a magnitude of that vector:

speed = distance/time
acceleration = (velocity2 – velocity1)/ (time2-time1)

velocity_future = velocity_now + acceleration*time
xspeed_future = xspeed_now = x acceleration*time

(See car.fla)

Acceleration:

- If you know the acceleration and current velocity of an object, you can predict the velocity of that object at any point in the future.

- To use acceleration in programming:

    1. Create variable to contain the acceleration

```
var kelvar:Number1 = 2
```

    2. Create initial velocity variables for x and y directions

```
var xmov:Number = 0;
var ymov:Number = 0;
```

    3. Modify the speed when acceleration should be applied

```
xmov += accel;
ymov += accel;
```

    4. For every frame, set the new position of the object

```
car._x += xmov;
car._y += ymov;
```

(See car4.fla)

Newton's Laws of Motion:

First Law: *The velocity of a system will not change unless it experiences a net external force.*

Second Law: *The acceleration of an object is inversely proportional to it's mass and proportional to the net external force applied.*

net force = mass*acceleration
or

F = m*a

This is a very helpful equation, because you can sum all of the forces acting on an object (net force) and from that sum determine its acceleration.

(show balloon.fla)

Balloon mass = 1

Force 1: gravitational force = 30 (its weight)

Force 2: buoyant force = - 31 (rising force of helium)

- Negative number of the buoyant force means that the force is going in the –y direction….or "up".

- Note that this balloon does not take into account the notion of **terminal velocity,** which is a maximum speed of acceleration caused by external factors such as atmosphere and wind

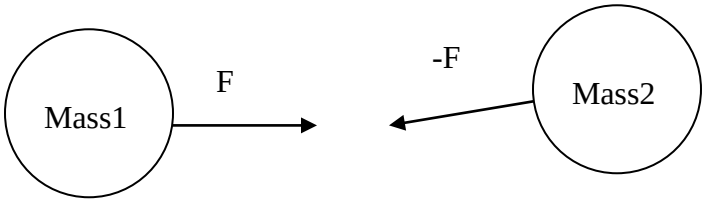
Gravity: "Real" vs "Good Enough"

Gravitational force experienced by two objects is calculated by:

F = G*(mass1*mass2)/distance

G is the constant of universal gravitation

distance is the distance between the centers of two objects



- Will almost never need to apply this realistic treatment of gravity to your games...
- (see Real Gravity.fla and good enough.fla)
- **With GE gravity, the trick is simply to come up with a value for gravity and add that value to your y velocity in every frame.**

(see bounce.fla)

```
var ymov:Number = 0;

var gravity:Number = 2; // set gravity

_root.onEnterFrame = function(){
        ymov += gravity;
        ball._y += ymov;
        if (ball._y>400){
                ball._y = 400;
                ymov *= -1; reverse the velocity
        }
}
```

Real Friction vs Good Enough Friction:

Friction is the force that opposes the direction of motion and is caused by the interaction of two materials.

Sliding Friction = F = u*mass*gravity

Mass*gravity = weight of the object

u = frictional coefficient = numerical value between 0 and 1 that is different for each object-object interaction

To Apply Friction:

1. Find the acceleration due to friction (accel = u*gravity)

2. Apply the accel value to the velocity in every frame until velocity reaches 0

(See roll.fla)

Good Enough Friction:

- The difference between real and "good enough" friction is really not worth coding for our purposes.

- "Real" friction decreases a velocity linearly whereas "GE" decreases it by a percentage of the current velocity (nonlinearly)

To Apply "GEF":

1. Choose a number between 0 and 1. Call it **decay.**

2. Multiply the decay by the velocity in every frame

(see roll2.fla)

```
var xmov:Number = 10;
var decay:Number = .95;

_root.onEnterFrame = function(){
        xmov *= decay;
        ball._x += xmov;
}
```