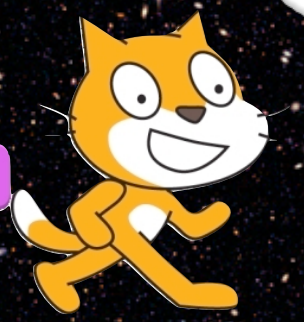
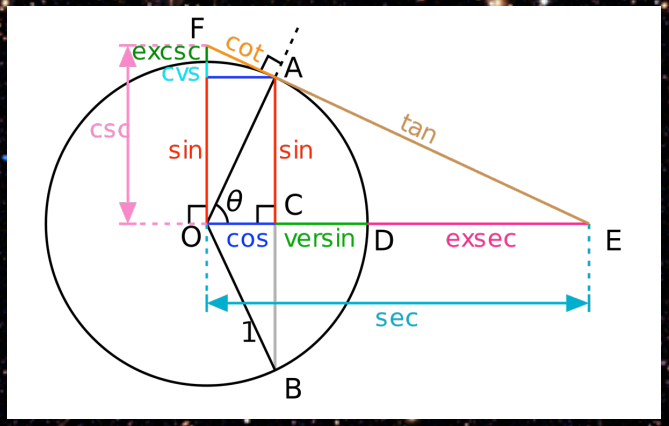
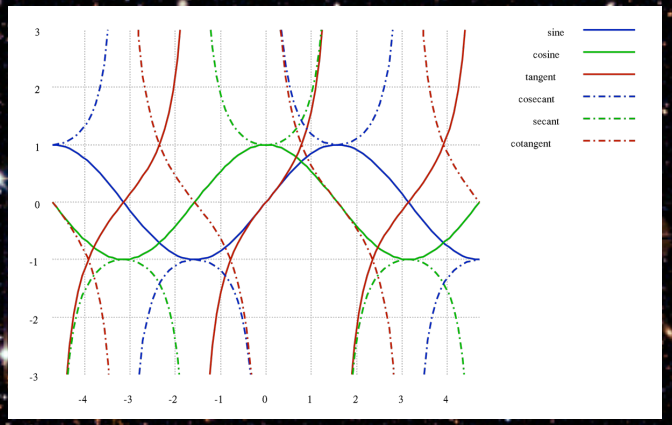


SCRATCH, GREENFOOT, MATH & ASTEROIDS



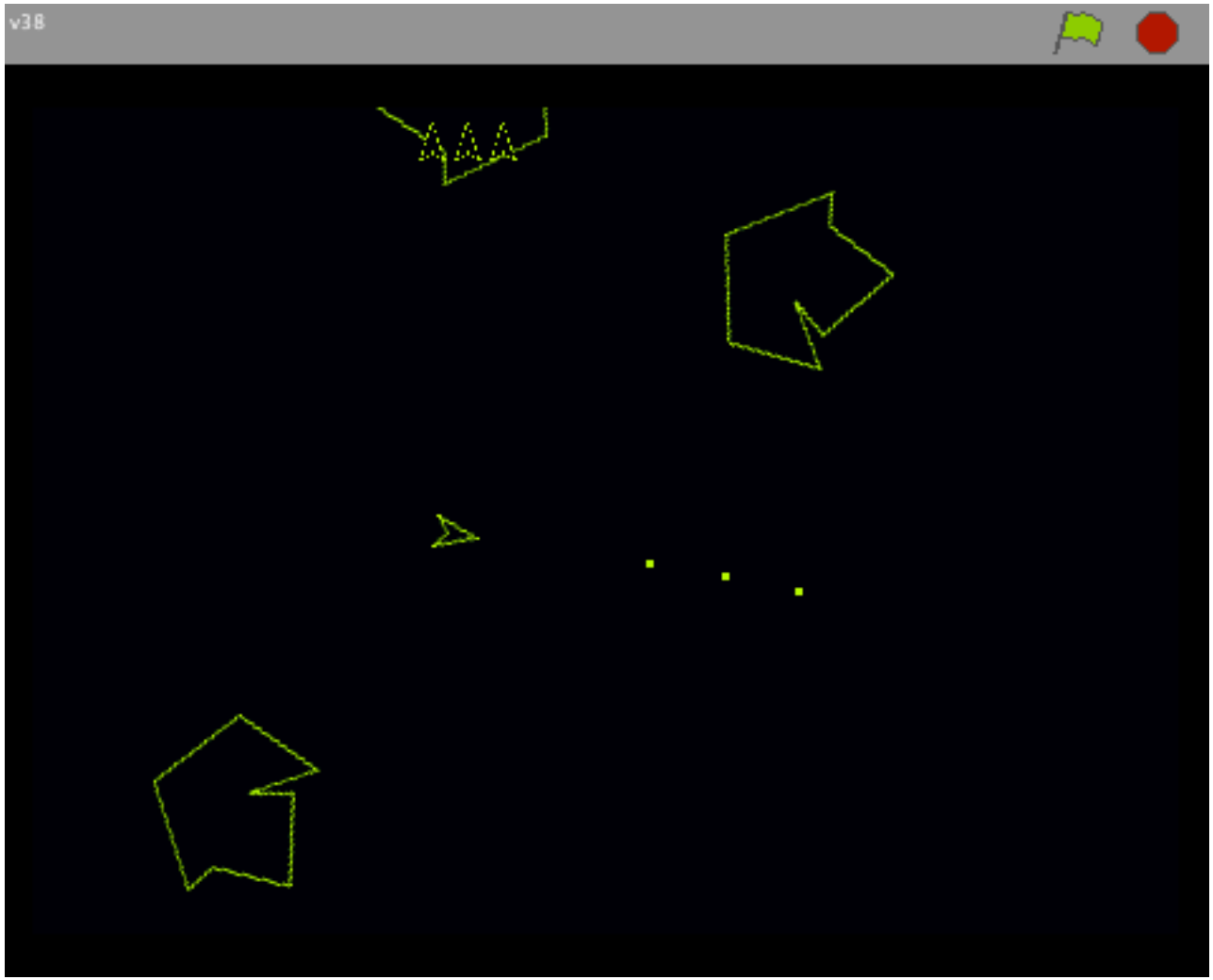
$$x = \frac{\sqrt{4ac + b^2} - b}{2a}.$$

$$\cos\theta \equiv \sin\left(\frac{\pi}{2} - \theta\right) \equiv \frac{1}{\sec\theta}$$



"But when are we going to use this?"

MISSION: Make a Game



- Something the students might already enjoy
- Involves motion, acceleration, momentum
- Involves projectiles

http://scratch.mit.edu/projects/SonicPopsDad/245563

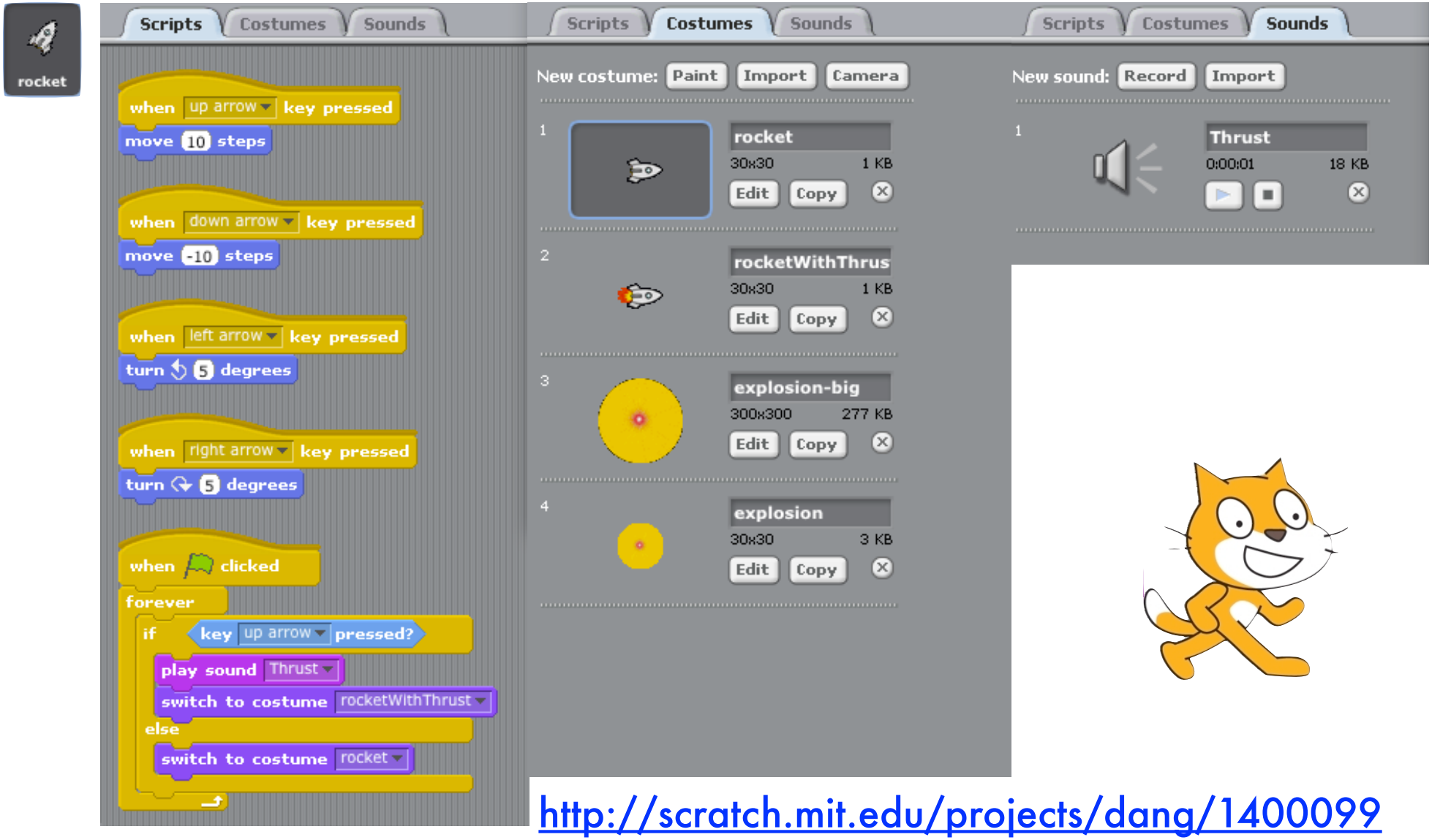
CS MISSION: Involve Programming

	Scratch	Alice	Media Computation / drjava	Greenfoot	BlueJ	NetBeans	Java
FEATURES	**Simplest tool** **Ages 5-15** •Drag and drop code blocks •2D graphics frameworks interaction	**Simpler tool** **Ages 8-22** •Drag and drop code blocks •3D graphics frameworks interaction	**Less simple** **Ages 12-22** •Interactive interpreter code typing •Media computation helper classes	**Less simple** **Ages 13-25** •Type,compile, run, debug •2D gaming framework interaction via 5 Java classes	**Less simple** **Ages 15-25** •Type, compile, run, debug •No default graphics framework environment	**Complex tool** **Ages 15-25** •Type, compile, run, debug •No default graphics framework environment	**No/Any tool** **Ages 16+** •Type, compile, run, debug •No default graphics framework environment
TOOLS							
TEACHING CONCEPTS	• Sequence • Iteration • Conditional Logic • Variables • Data Structures (dynamic lists) • Events Handling • Parallel Execution • Synchronization • Random Numbers • Boolean Logic • Dynamic Interaction • User Interface Design • Publish projects as Java applets on scratch.mit.edu	• All Scratch concepts plus.. • Procedures and Functions • Parameter Passing & Return Values • Recursion • Defining Classes of Objects • Inheritance • Text Input	• All previous Alice concepts, excluding user interface design • Manipulation of audio, images, video through media computation helper classes – see mediacomputation.org • All Java Programming Language Features Available • Interpreted mode is great step from drag and drop code blocks (no errors possible) to type – compile – run – debug (all errors possible)	• All previous Alice concepts plus... • 5 Java classes encapsulate 2D gaming and simulation concepts • All Java Programming Language Features Available	• Simplest Java IDE • All Java Programming Language Features Available	• Complex IDE • Multiple programming languages available (Java, Ruby, Groovy, Python, PHP, JavaScript, etc.) • Tooling covers mobile and embedded, enterprise, all avenues	• Programming Language common to Alice, Dr. Java, Greenfoot, BlueJ and NetBeans environments • All Java Programming Language Features Available • Tool / environment agnostic

Scratch Project: Asteroids

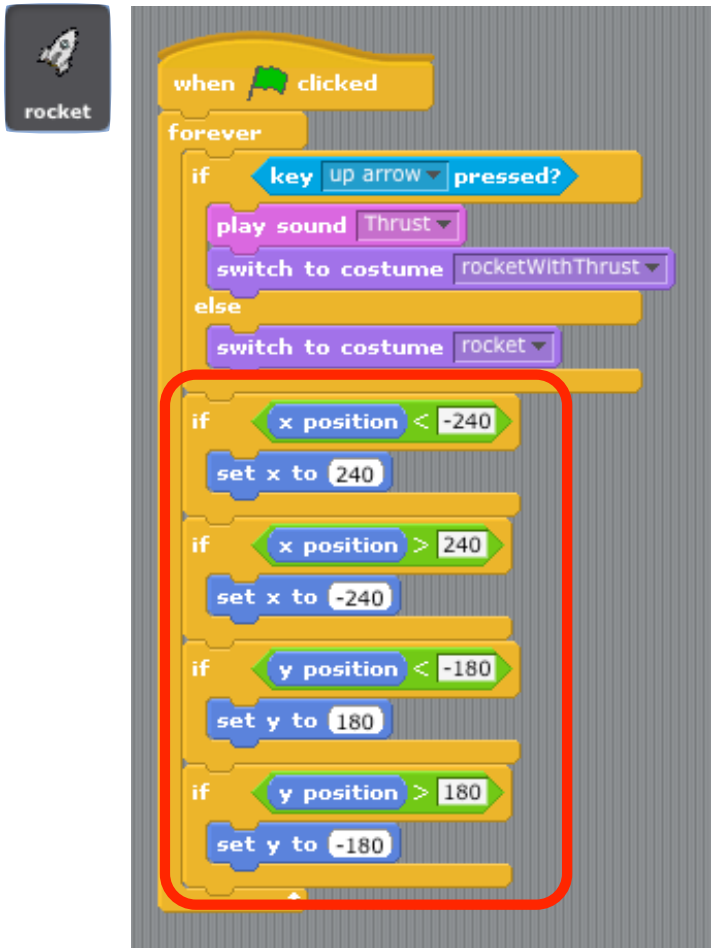
Phase	 Scratch	Math	Asteroids
1. Get Moving	**Motion**, **Looks**, **Sound**, **Control**, **Sensing**	angle of ship turn, steps to move ship = speed	rocket turn, thrust, movement
2. Off Screen	**Motion**, **Control**, **Operators**	x/y coordinates of ship position	screen wrap around
3. Collisions	**Motion**, **Looks**, **Sound**, **Control**, **Sensing**, **Operators**	variables, conditional logic, event handling	ship crashing into asteroids, asteroids crashing into ship
4. Shooting	**Motion**, **Looks**, **Sound**, **Control**, **Sensing**, **Operators**	message passing, relational expressions	ship shooting bullet, asteroids getting hit by bullet
5. Momentum	**Motion**, **Looks**, **Control**, **Sensing**, **Operators**	trigonometric functions, velocity, acceleration	gliding based on momentum and thrust acceleration

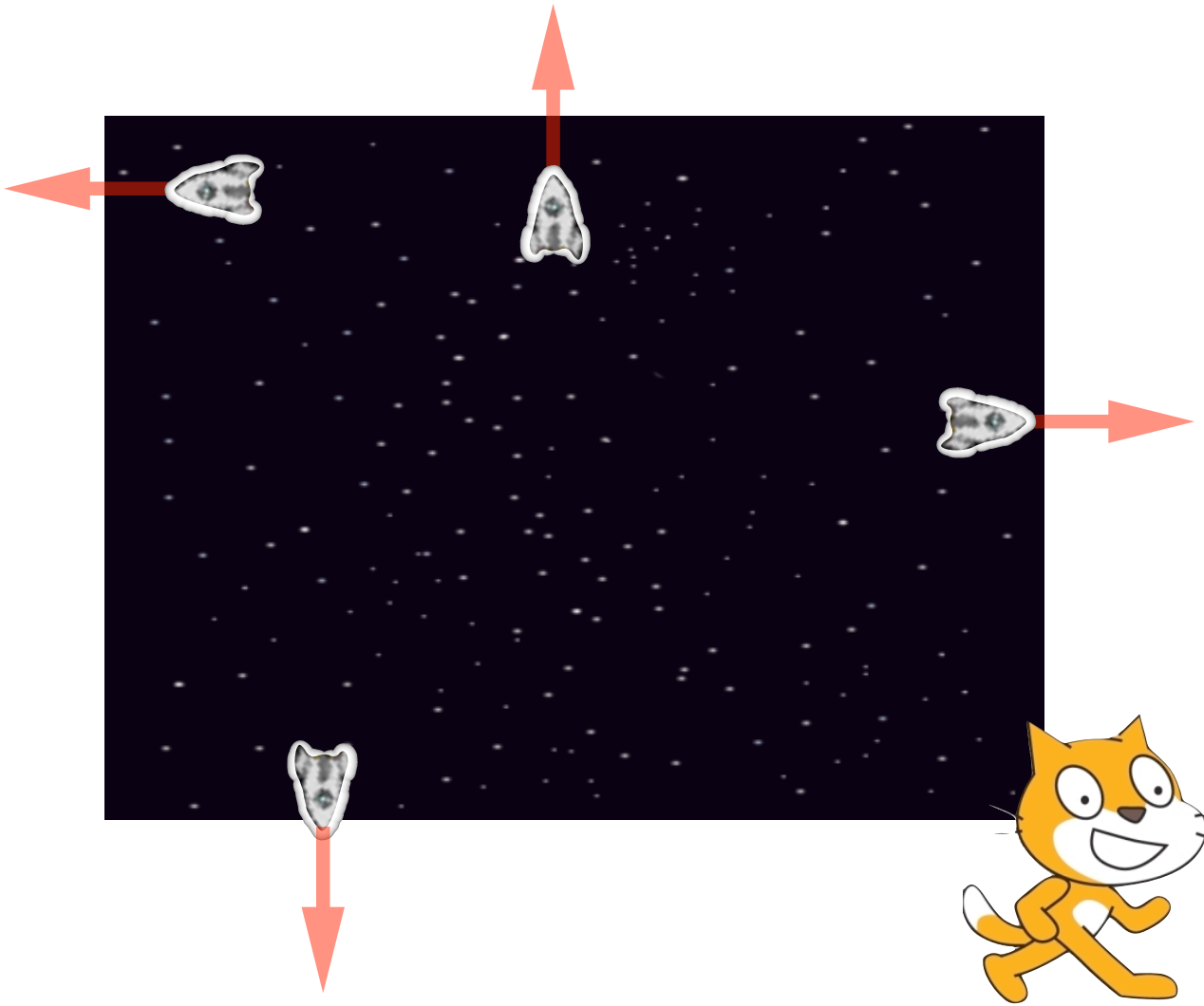
Scratch Phase 1: Get Moving



http://scratch.mit.edu/projects/dang/1400099

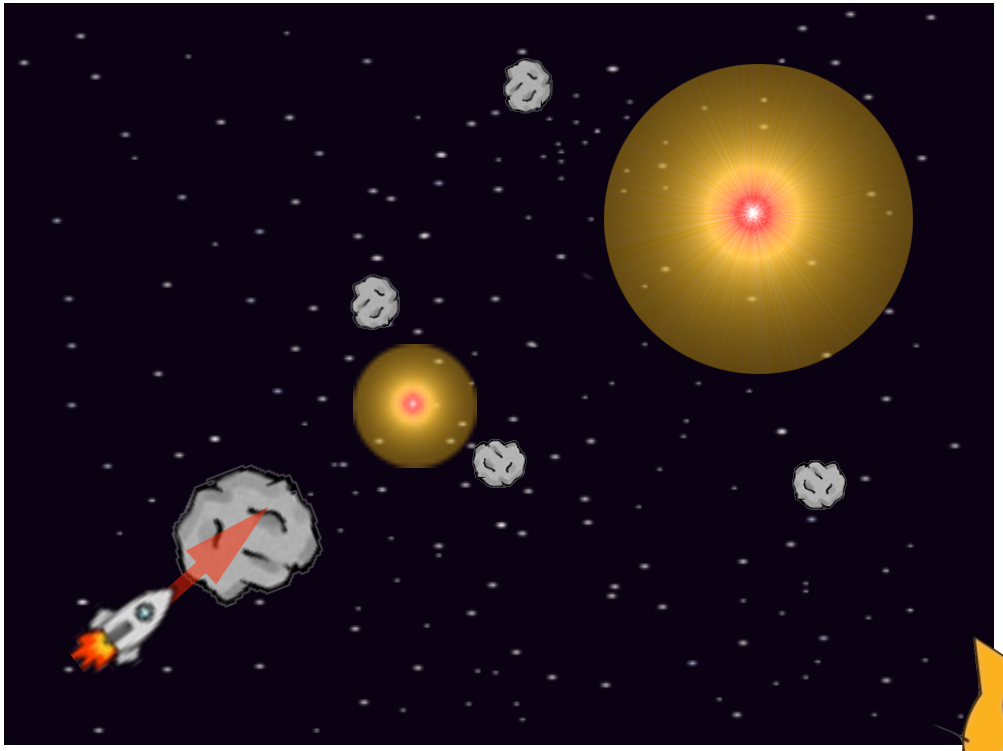
Scratch Phase 2: Off Screen





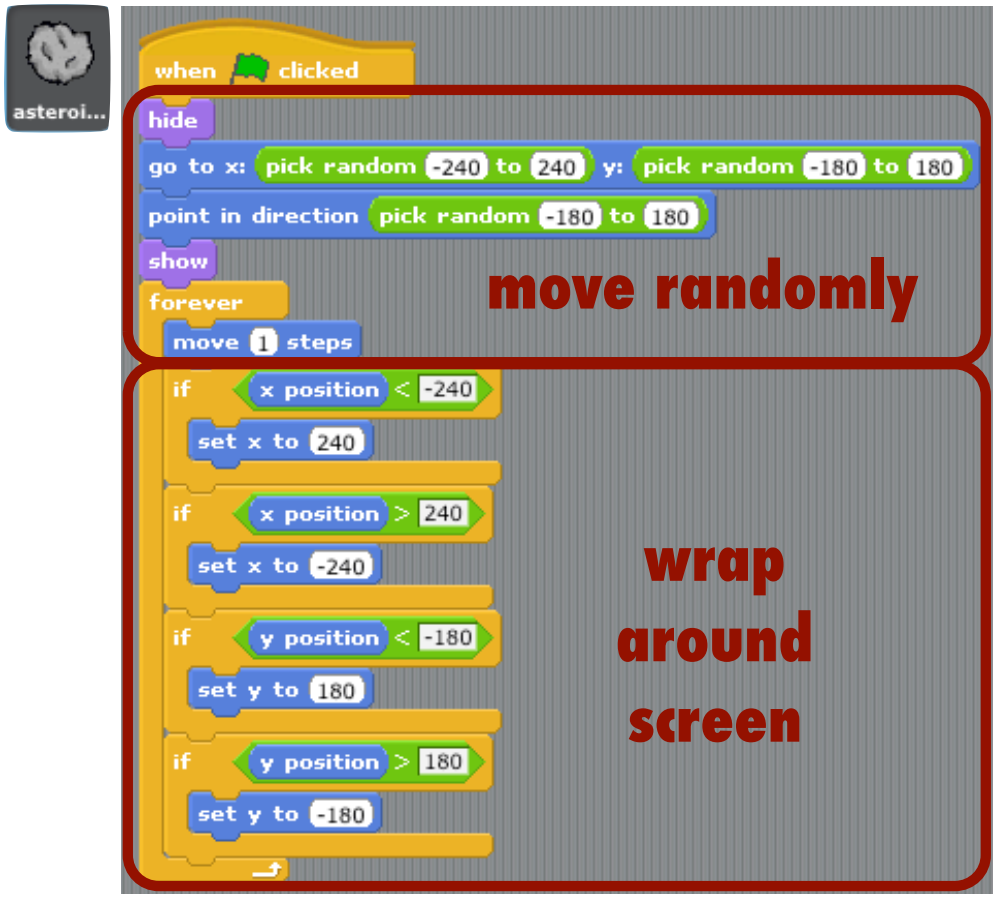
http://scratch.mit.edu/projects/dang/1400601

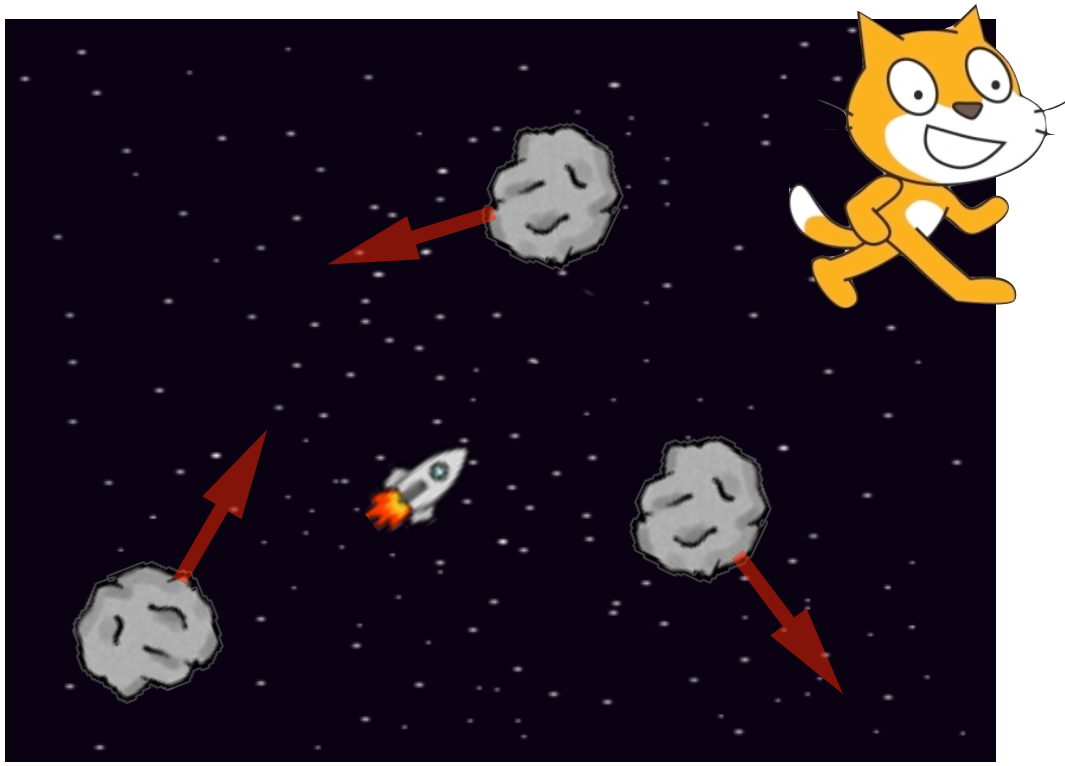
Scratch Phase 3: Collisions



http://scratch.mit.edu/projects/dang/1400765

Scratch Phase 3: Collisions





http://scratch.mit.edu/projects/dang/1400765

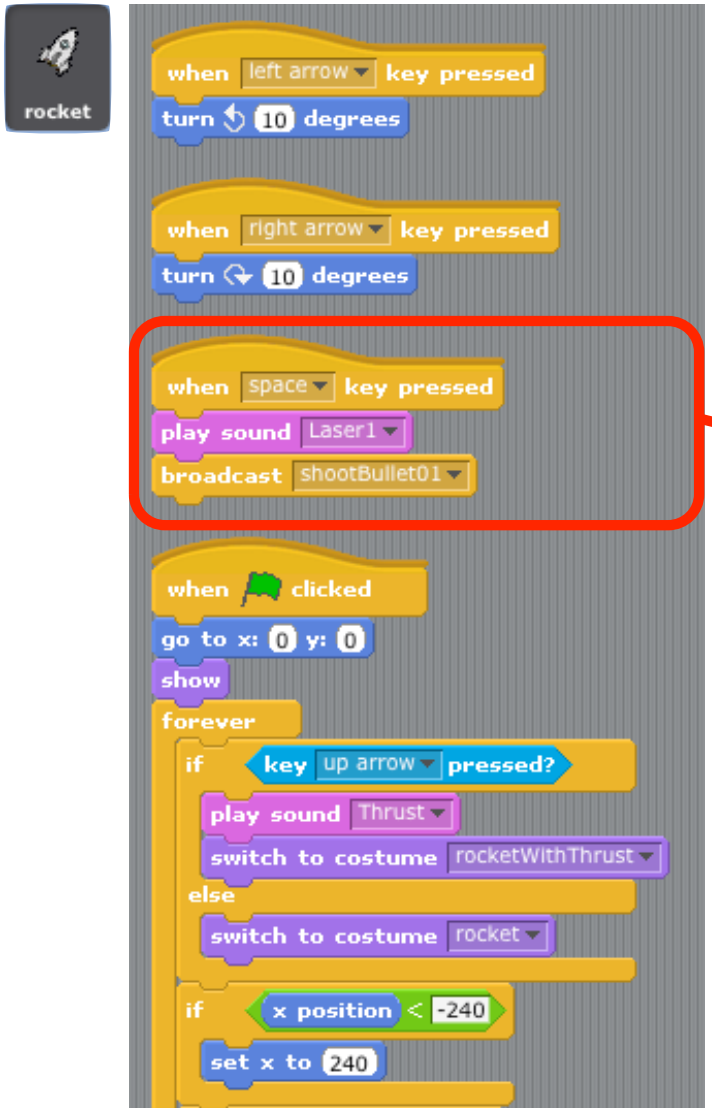
- Asteroids are not yet exploding on impact
- They provide something for the rocket to hit

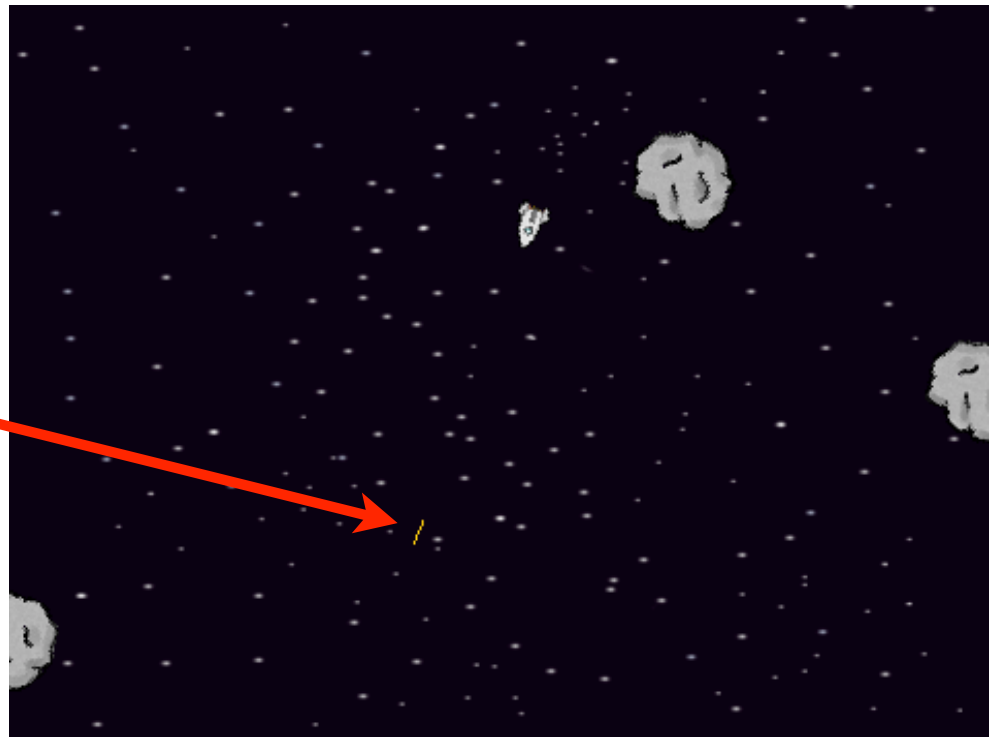
Scratch Phase 3: Collisions

http://scratch.mit.edu/projects/dang/1400765

- `gameOver` message is broadcast from rocket when it collides with an asteroid

Scratch Phase 4: Shooting

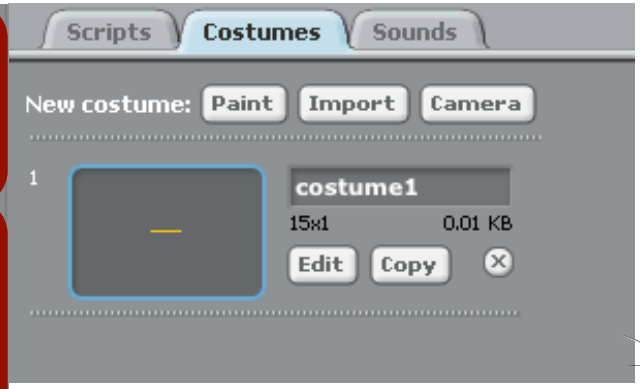


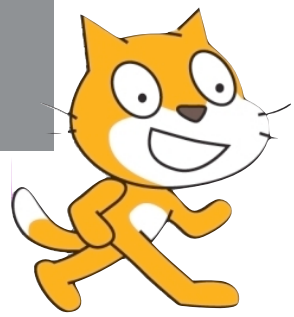


http://scratch.mit.edu/projects/dang/1402543

- `shootBullet01` message is broadcast from rocket when space key is pressed

Scratch Phase 4: Shooting





http://scratch.mit.edu/projects/dang/1402543

- This version only deals with shooting one bullet at a time
- Students can add `bullet02`, `bullet03`, and scripting

Scratch Phase 4: Shooting

big 1

big 2

big 3

http://scratch.mit.edu/projects/dang/1402543

small 1

small 2

small 3

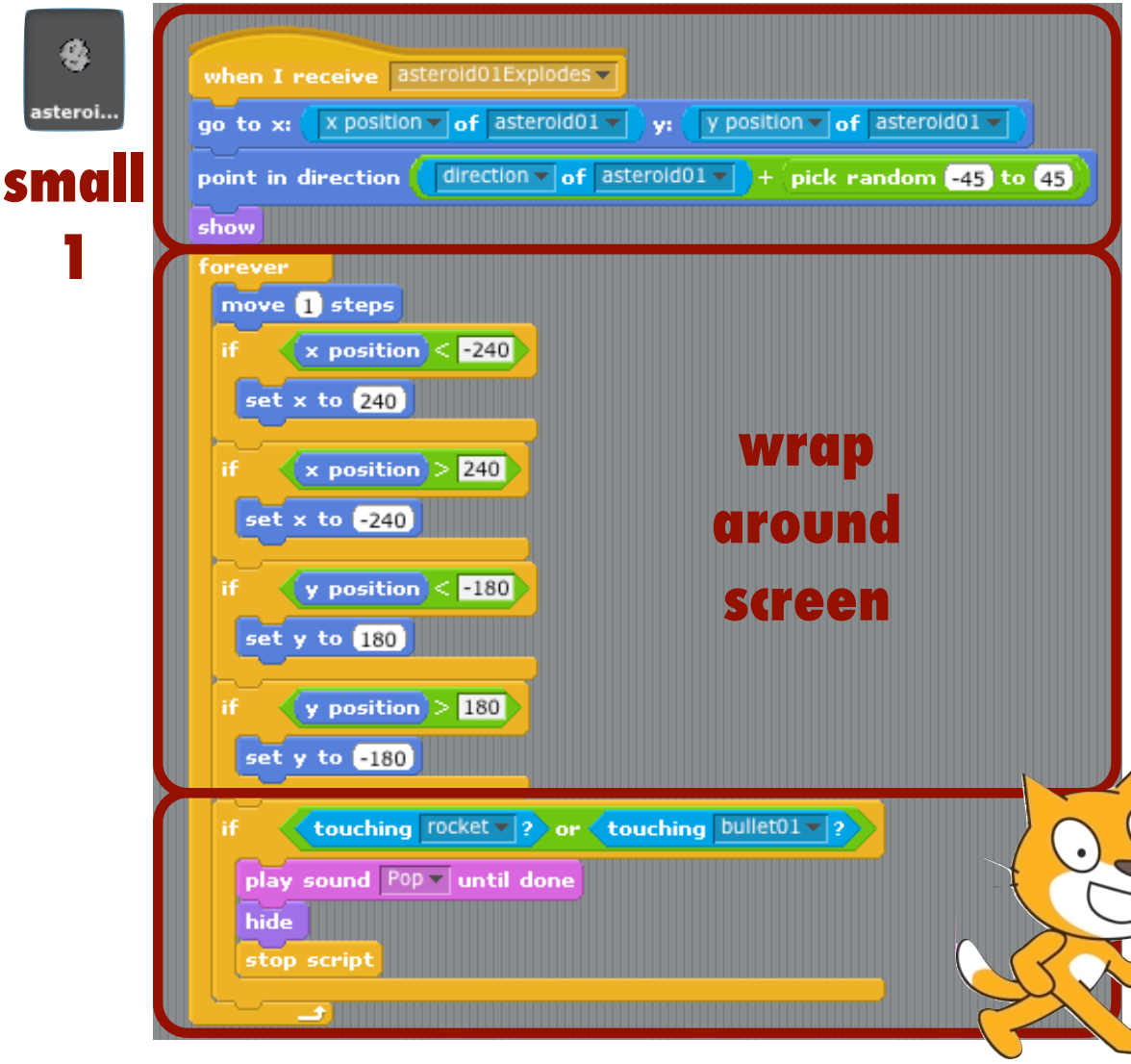
small 4

small 5

small 6

- Listening for `asteroid01Explodes`
- Listening for `asteroid01Explodes`
- Listening for `asteroid02Explodes`
- Listening for `asteroid02Explodes`
- Listening for `asteroid03Explodes`
- Listening for `asteroid03Explodes`

Scratch Phase 4: Shooting



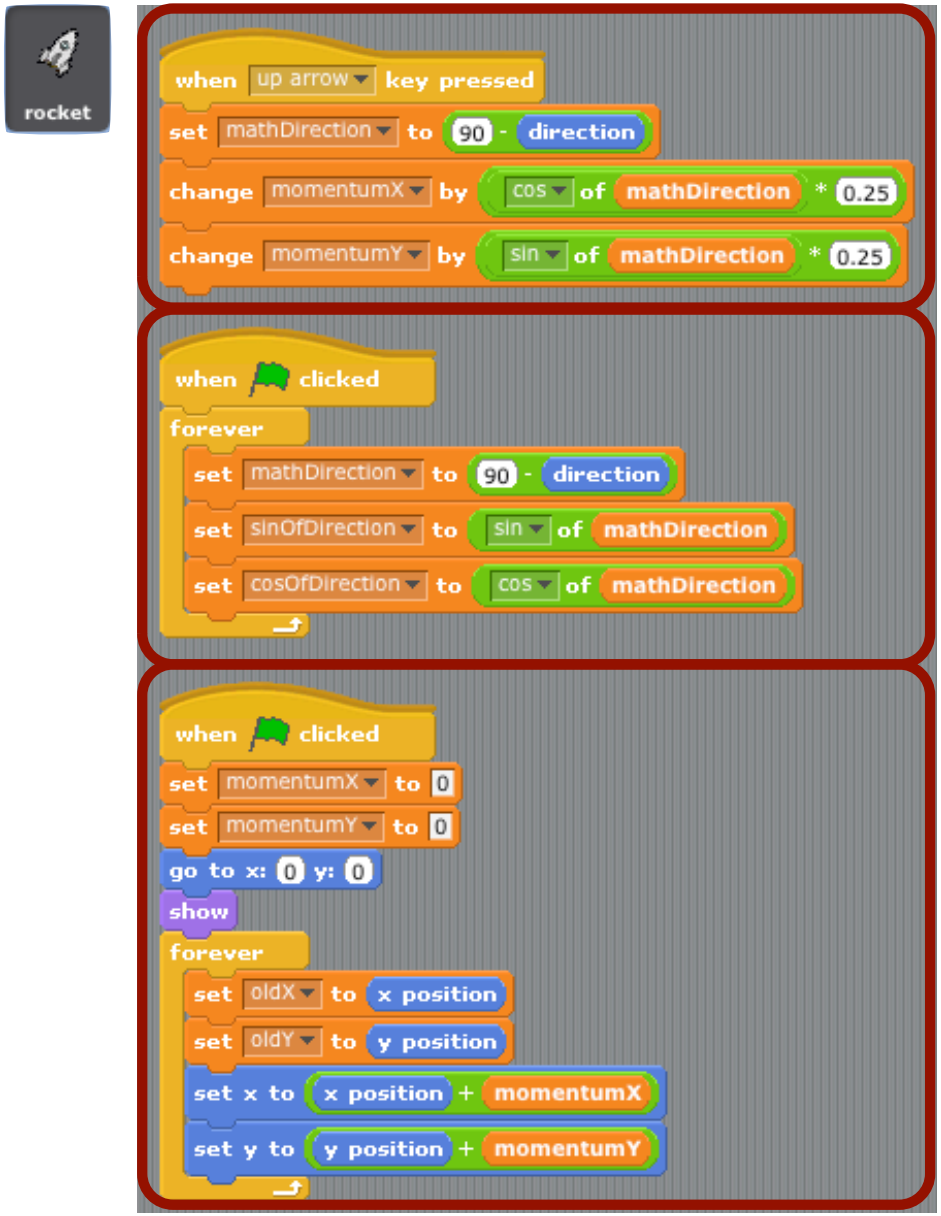
appear near where the bigger asteroid exploded traveling in roughly same direction

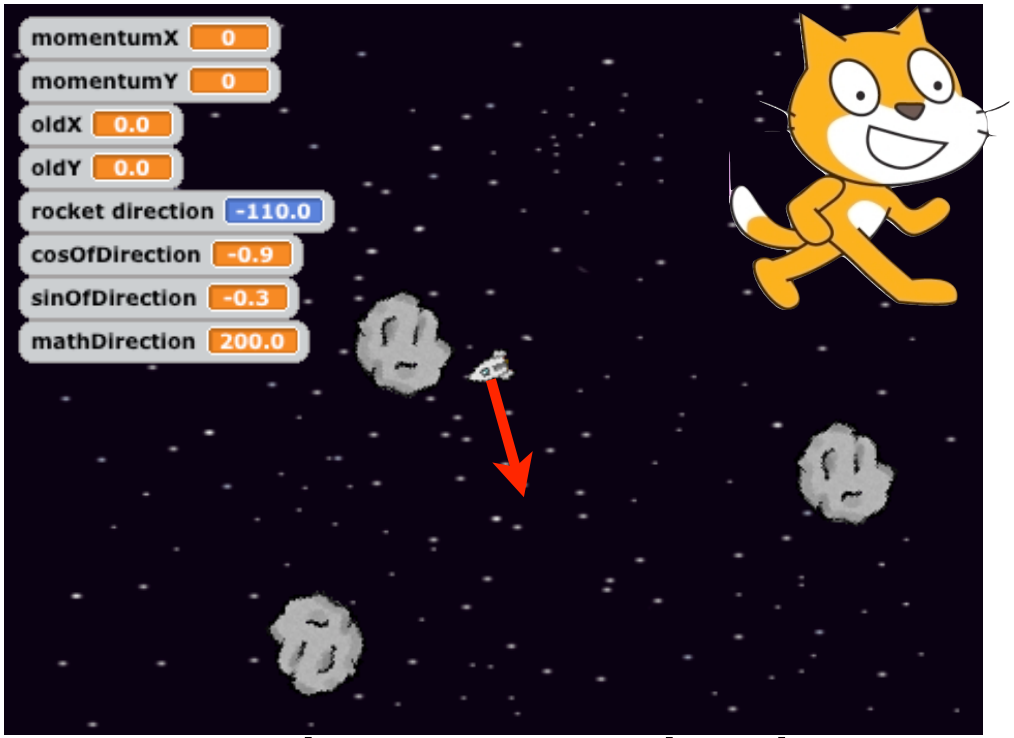
- Each smaller asteroid belongs to a larger asteroid
- Each smaller asteroid sprite has the same script except for responding to `asteroid01`, `asteroid02`, or `asteroid03`

if hit by rocket or bullet, explode and don't worry about smaller chunks

http://scratch.mit.edu/projects/dang/1402543

Scratch Phase 5: Momentum

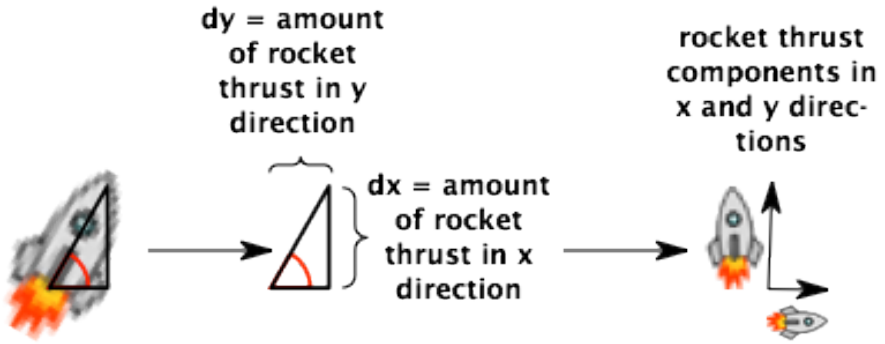




- Major switch in moving the ship from stop and go to maintaining directional momentum
- Adjusted `mathDirection` from Scratch built in `direction` variable

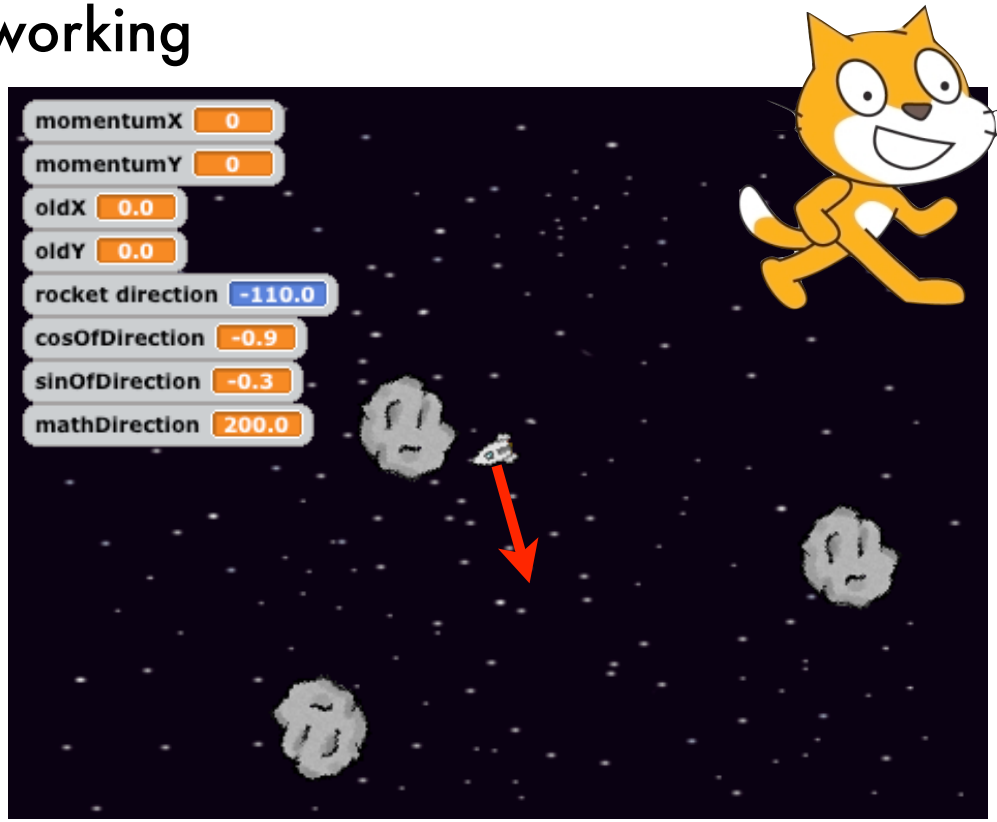
http://scratch.mit.edu/projects/dang/1423848

Momentum - Part 1



- In order to keep your rocket ship from the Asteroids game flying around in the right direction and maintaining momentum, there are **two things** you'll need to do get that working

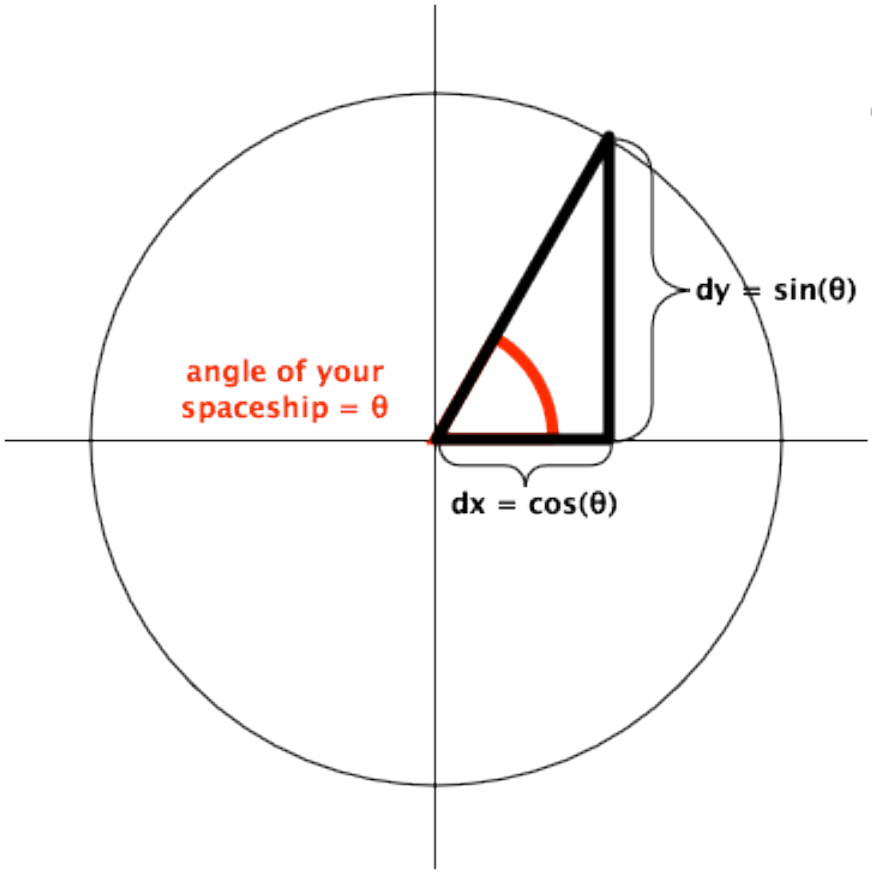
- **THING 1** - We need to know how big or small of change in both the x and y direction to apply as you zoom around the screen with your rocket turned on

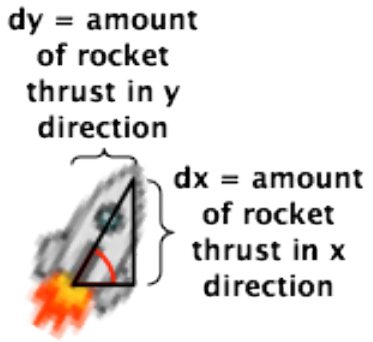


http://scratch.mit.edu/projects/dang/1423848

Momentum - Part 2

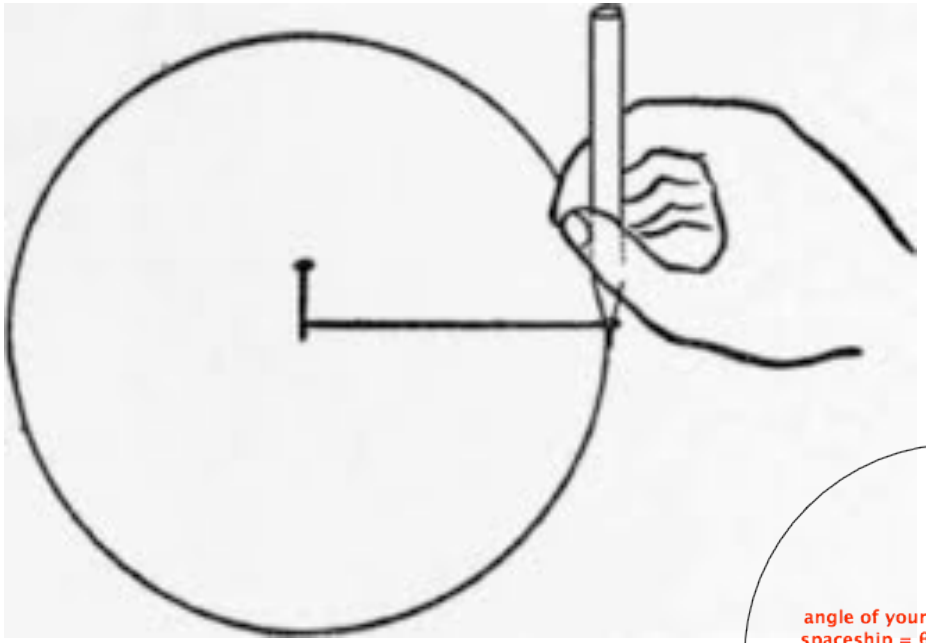
- **THING 2** - We need to know how big or small of a change in momentum. If you liken this triangle of rocket tilt to a circle, you can imagine your rocket is the radius of the circle and for the purposes of telling Scratch how much or little to move us along the x axis and y axis, we're interested in getting the x and y component of the triangle formed at every position around the circle as the rocket is rotating as illustrated in the figure here

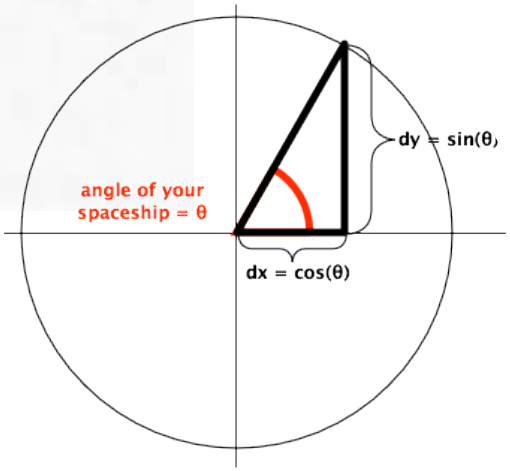




Momentum - Part 3

- **TRIG FUNCTIONS** -

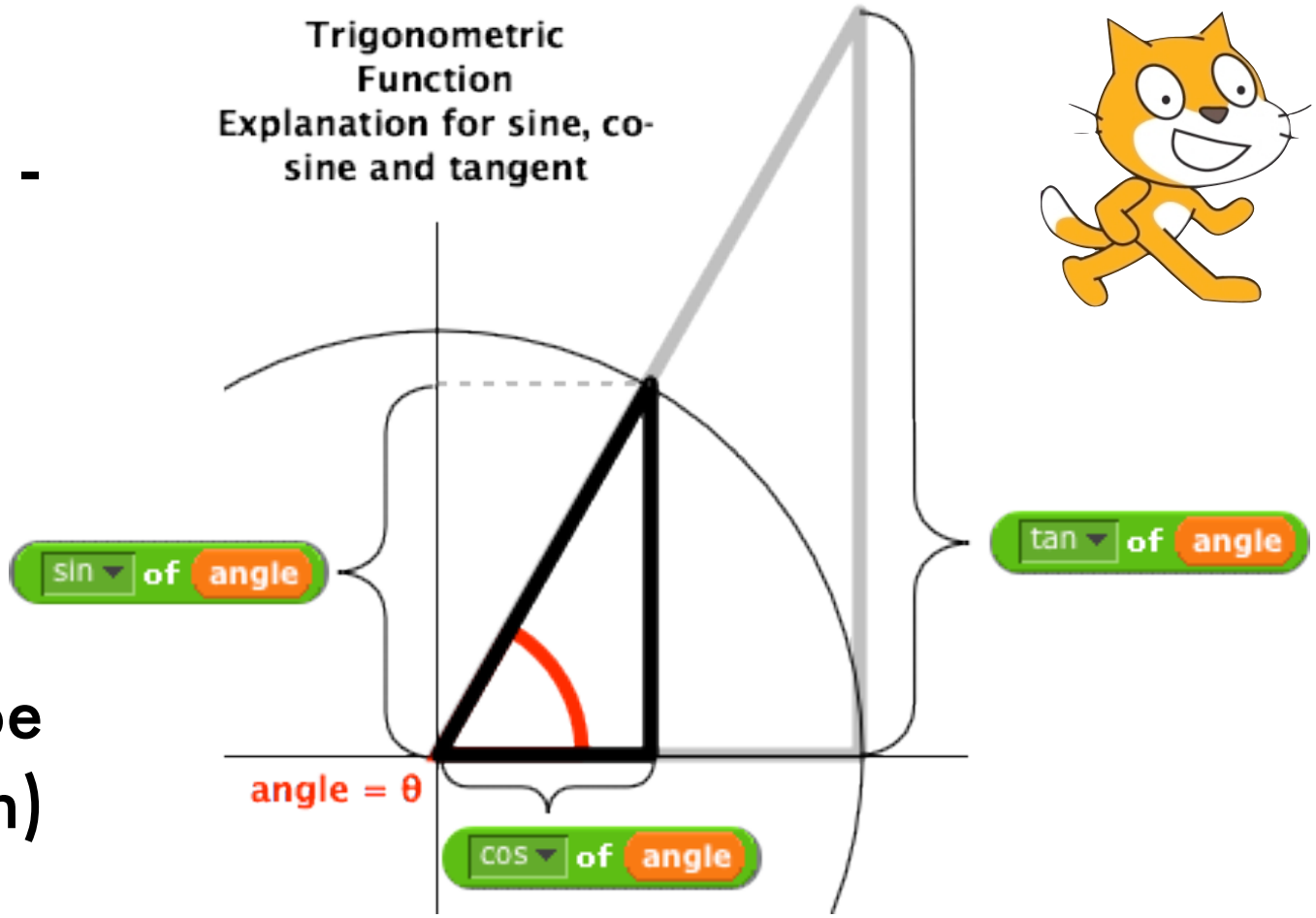




Thankfully there are handy trigonometric functions available to us that give us exactly this, the x and y component of a triangle in this way. You may have already hit this in school, known as the formula for a circle, where, placing a circle with origin at x=0, y=0, all points along the circle can be described as the radius squared is equal to the sum of the x positon squared plus the y position squared, or r^2 = x^2 + y^2. An easy way to think about this is drawing a circle with a pencil, a piece of string, and a pin, it would look like the figure above

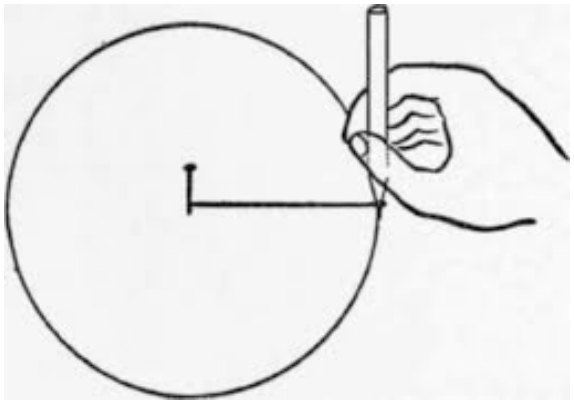
Momentum - Part 4

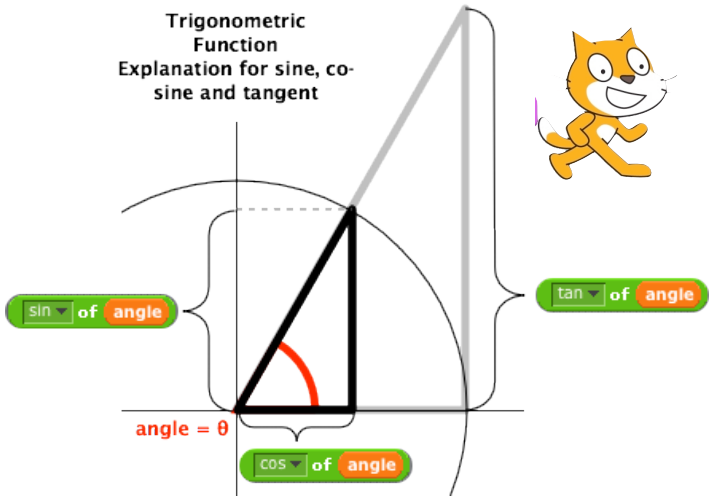
- **TRIG & SCRATCH** - The relationship of functions we'll use and which Scratch blocks you'll need are illustrated here, note that we won't be needing tangent (tan) but it's here for completeness, almost any discussion including sine and cosine will also mention tangent.



Momentum - Part 5

- **FROM ZERO TO ONE AND BACK AGAIN** - Rather than draw lots of different triangles representing you spinning your rocket ship around, here are the values these functions return given the angle you put in, the table below shows values for a few different angles. For our purposes, you'll be using the built in `direction` variable in scratch for your angle.

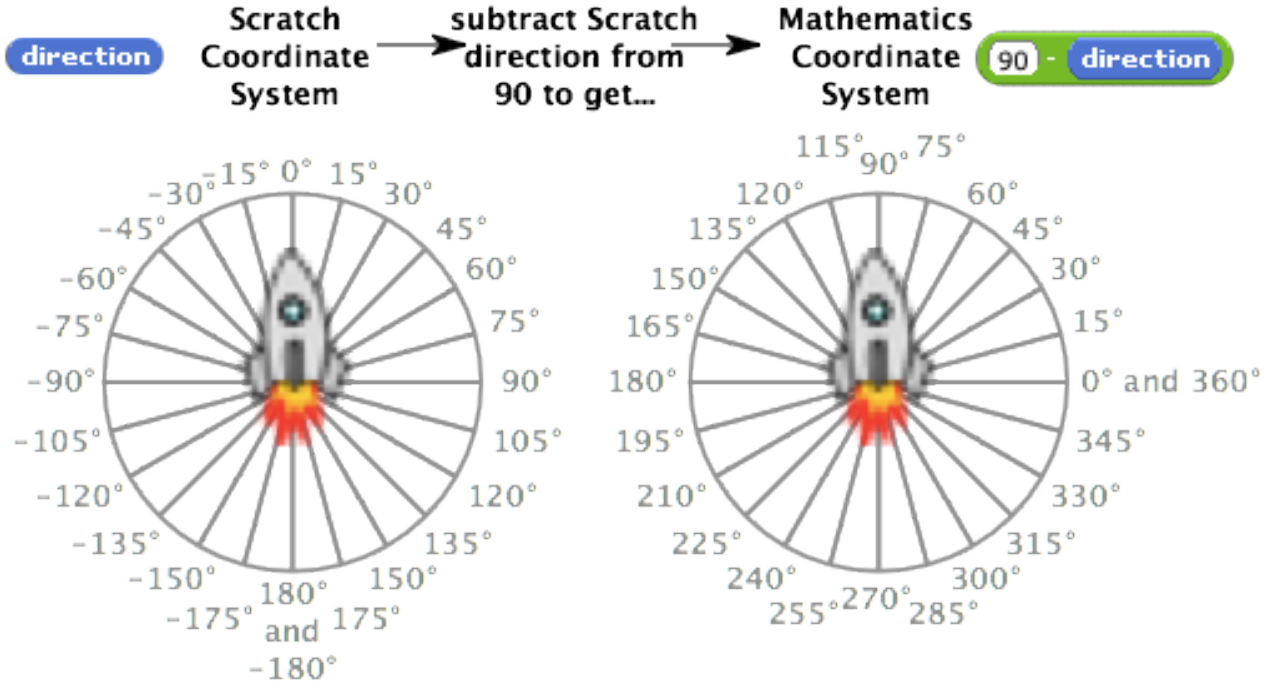




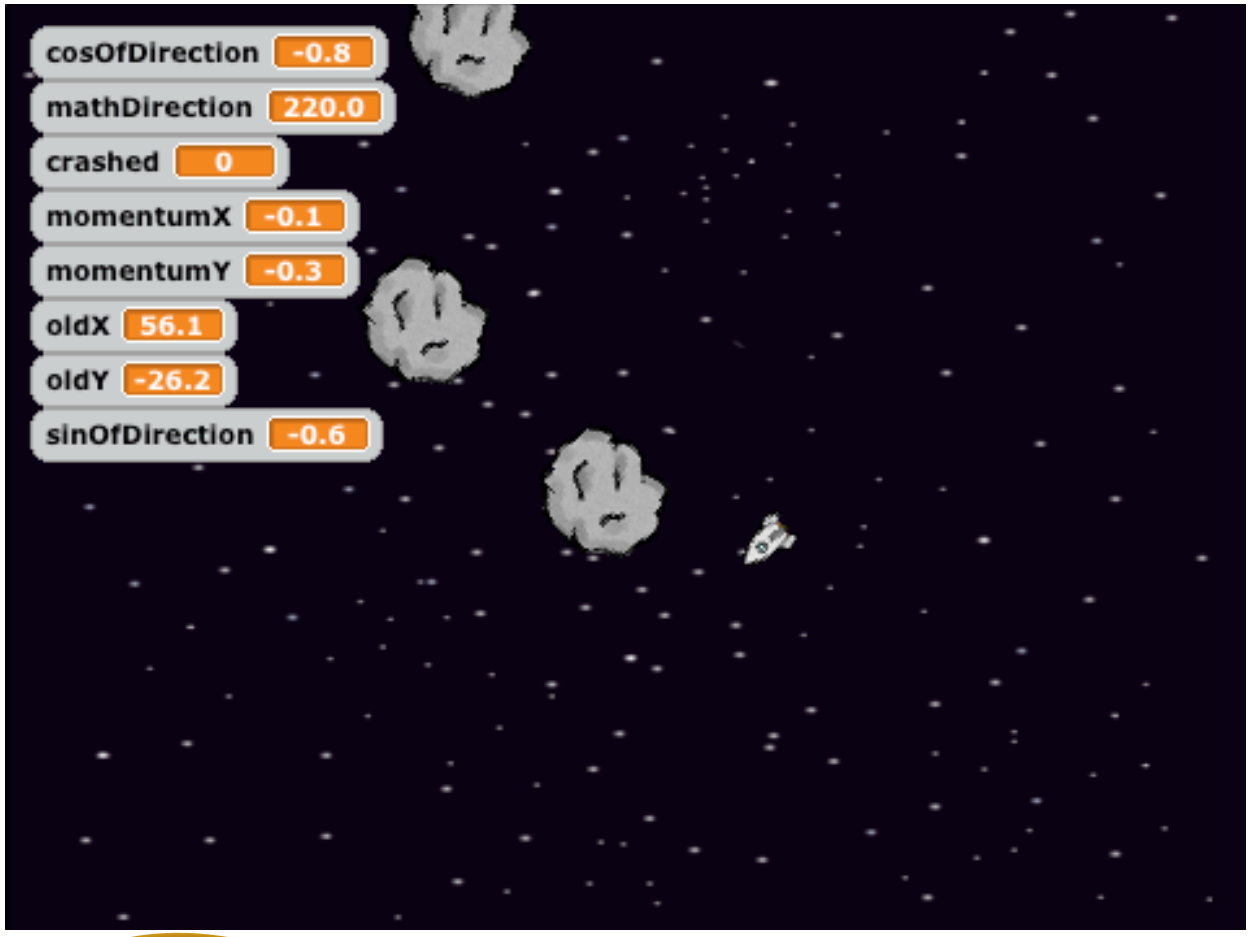
angle=θ	cos(θ)	sin(θ)	tan(θ)
0	1.0	0.0	0.0
15	0.9659	0.2588	0.2679
30	0.8660	0.5	0.5773
45	0.7071	0.7071	1.0
60	0.5	0.8660	1.7320
75	0.2588	0.9659	3.7320
90	0.0	1.0	infinity
105	-0.2588	0.9659	-3.7320
120	-0.5	0.8660	-1.7320
135	-0.7071	0.7071	-1.0
150	-0.8660	0.5	-0.5773
165	-0.9659	0.2588	-0.2679
180	-1.0	0.0	0.0
195	-0.9659	-0.2588	0.2679
210	-0.8660	-0.5	0.5773
225	-0.7071	-0.7071	1.0
240	-0.5	-0.8660	1.7320
255	-0.2588	-0.9659	3.7320
270	0.0	-1.0	infinity
285	0.2588	-0.9659	-3.7320
300	0.5	-0.8660	-1.7320
315	0.7071	-0.7071	-1.0
330	0.8660	-0.5	-0.5773
345	0.9659	-0.2588	-0.2679
360	1.0	0.0	0.0

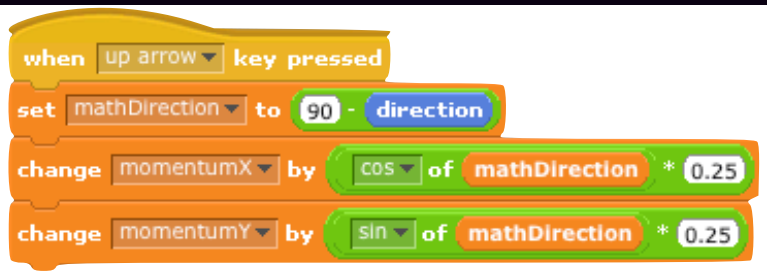
Momentum - Part 6

- **BUT IT DOESN'T WORK???** - If you're following along so far and have tried all this, you'll notice it doesn't work correctly, the ship flies off in directions you don't expect. Well, here the reason: the coordinate system for direction used by scratch is not the same as the coordinate system we typically see in math examples, namely, most math examples assume angle=0 points to the right and Scratch assumes angle=0 points up. Neither is right or wrong, you can spin your coordinate system any way you'd like. And that's exactly what we'll do as shown in the diagram here.

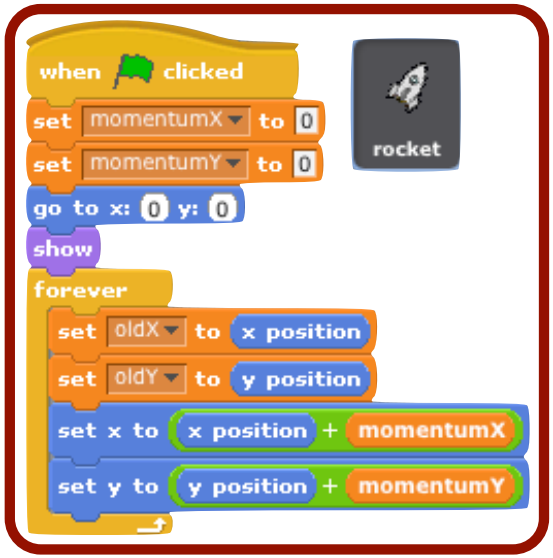


Momentum - Part 7





- **MOVE** - repeatedly move ship around based on momentum in the x and y direction

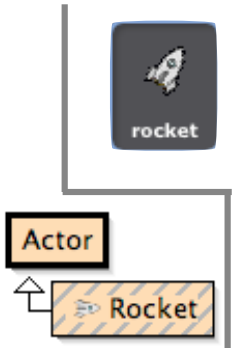


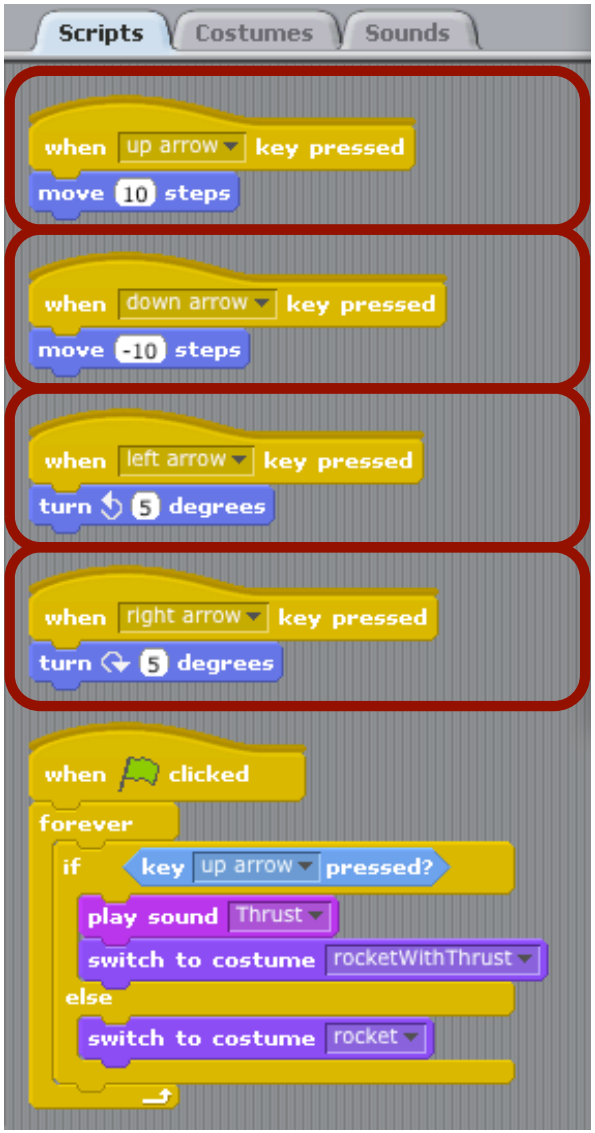
Greenfoot Project: Asteroids

Phase	Greenfoot	Math	Asteroids
1. Get Moving	World, Actor, act() / run / reset framework	angle of ship turn, steps to move ship = speed	rocket turn, thrust, movement
2. Off Screen	to be completed	x/y coordinates of ship position	screen wrap around
3. Collisions	to be completed	variables, conditional logic, event handling	ship crashing into asteroids, asteroids crashing into ship
4. Shooting	to be completed	message passing, relational expressions	ship shooting bullet, asteroids getting hit by bullet
5. Momentum	to be completed	trigonometric functions, velocity, acceleration	gliding based on momentum and thrust acceleration

Greenfoot Phase 1: Get Moving

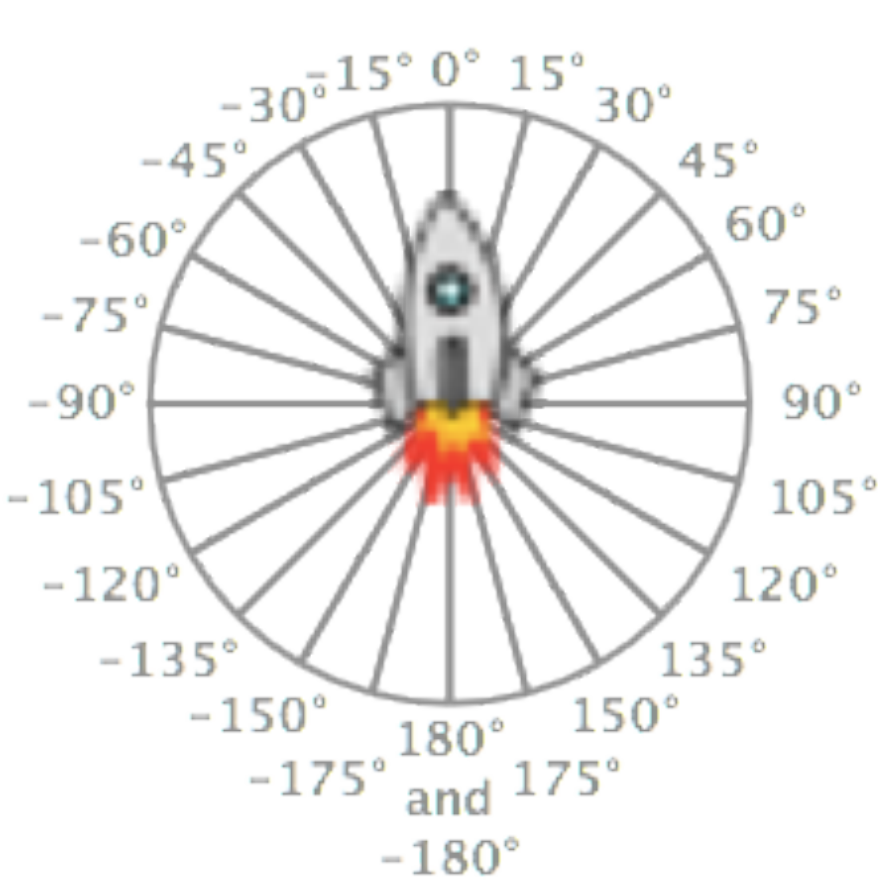
```
public class Rocket extends Actor
{
    /**
     * Act - do whatever the Rocket wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        checkKeys();
    }

    /**
     * Check whether there are any key pressed and react to them.
     */
    private void checkKeys()
    {
        if(Greenfoot.isKeyDown("up")) {
            moveSteps(10);
        }
        if(Greenfoot.isKeyDown("down")) {
            moveSteps(-10);
        }
        if(Greenfoot.isKeyDown("left")) {
            turnLeftDegrees(5);
        }
        if(Greenfoot.isKeyDown("right")) {
            turnRightDegrees(5);
        }
    }
```

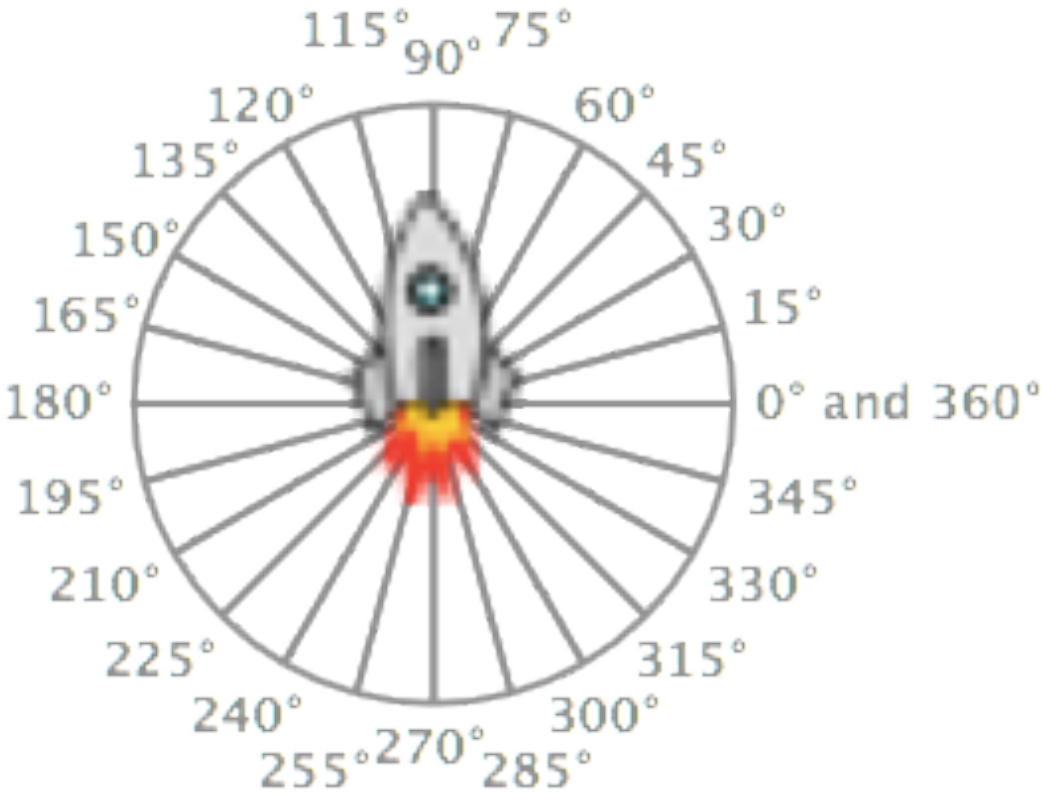




Scratch Movement (review)

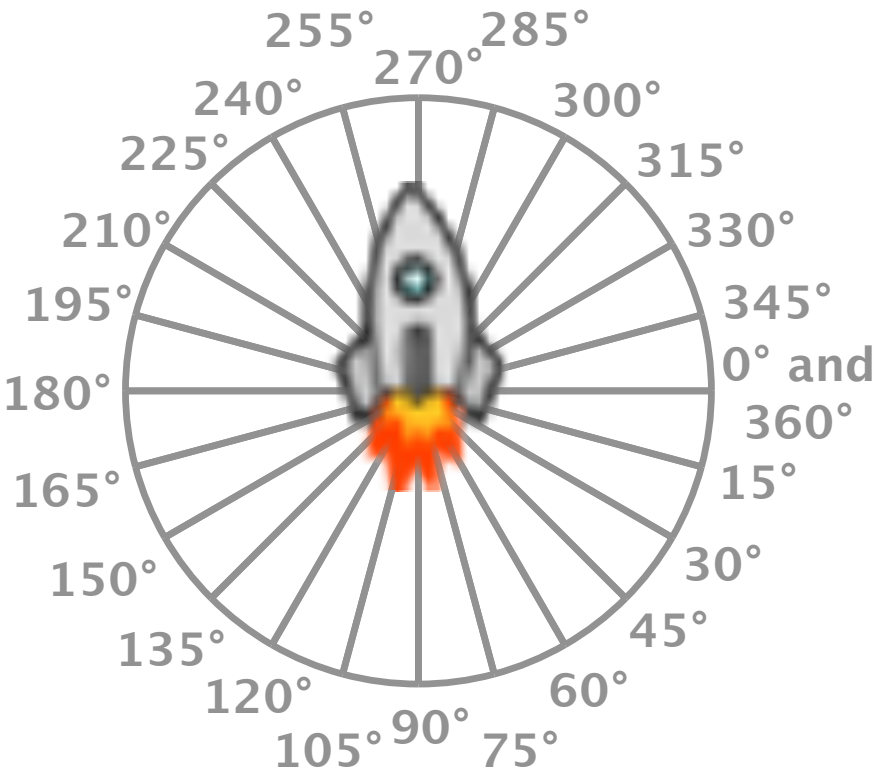
direction → Scratch Coordinate System → subtract Scratch direction from 90 to get... → Mathematics Coordinate System → 90 - direction

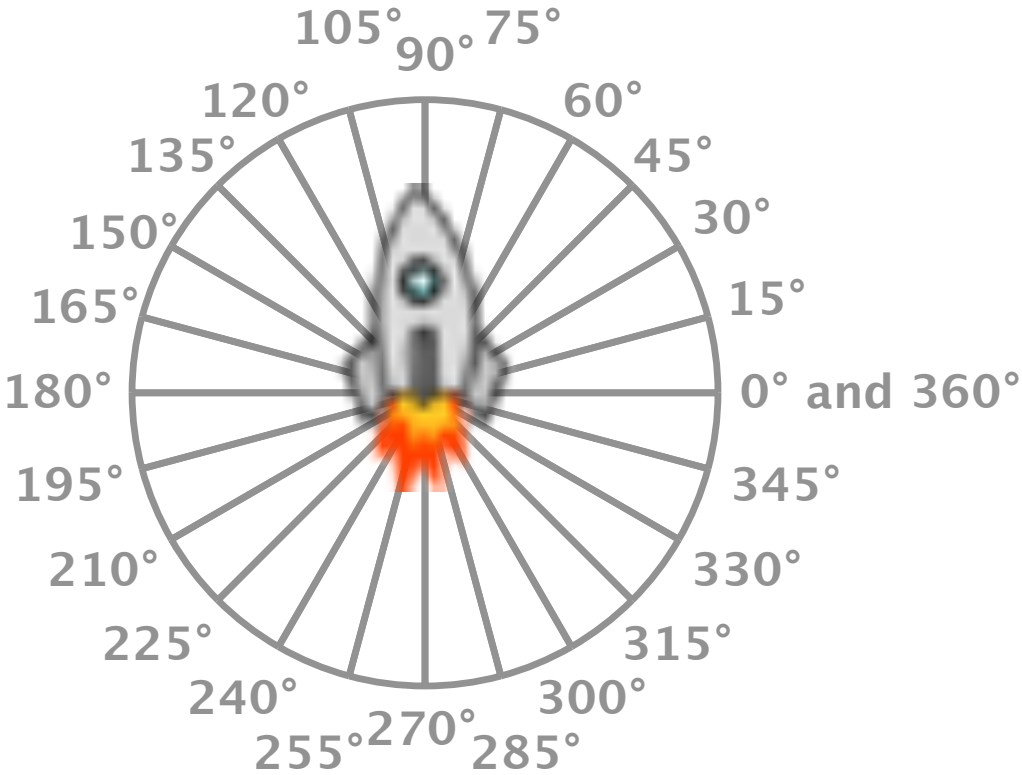




Greenfoot Movement - Part 1

`Actor.getRotation()` → Greenfoot Coordinate System → subtract Greenfoot direction from 360 to get... → Mathematics Coordinate System → `(360 - Actor.getRotation())`

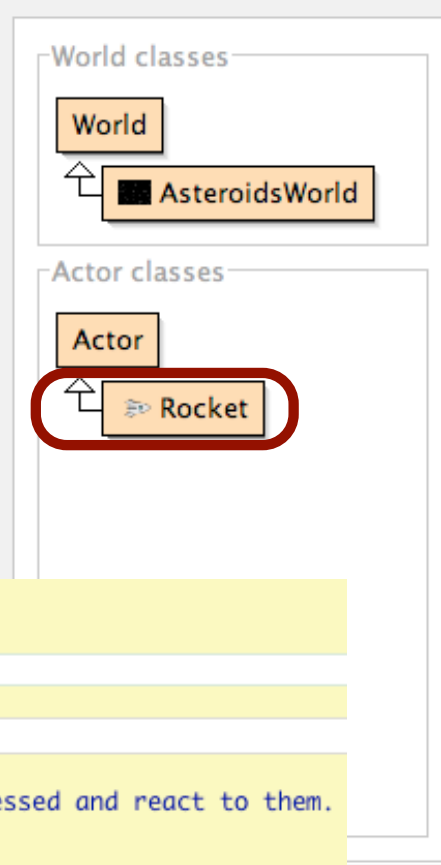




Greenfoot Movement - Part 1

```
public void turnLeftDegrees(int degrees)
{
    int rotation1;
    int rotation2;

    // current degrees of rotation of the ship
    rotation1 = getRotation();
    // new degrees of rotation of the ship
    // SUBTRACT to go left
    rotation2 = rotation1 - degrees;

    setRotation(rotation2);
}
```

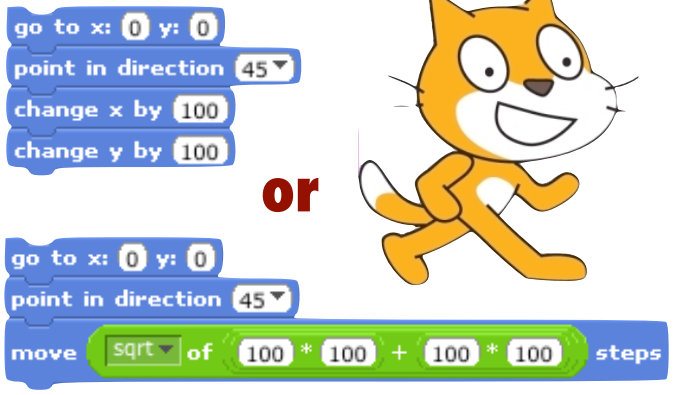
```
public void turnRightDegrees(int d)
{
    int rotation1;
    int rotation2;

    // current degrees of rotation of the ship
    rotation1 = getRotation();
    // new degrees of rotation of the ship
    // ADD to go right
    rotation2 = rotation1 + d;

    setRotation(rotation2);
}
```

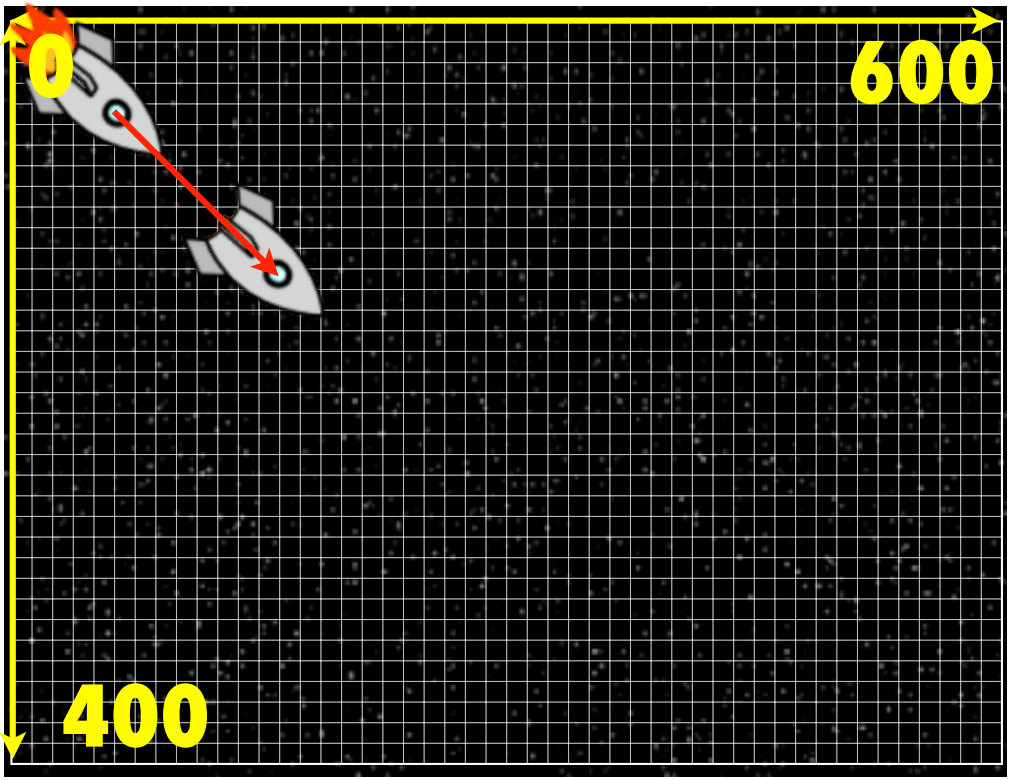


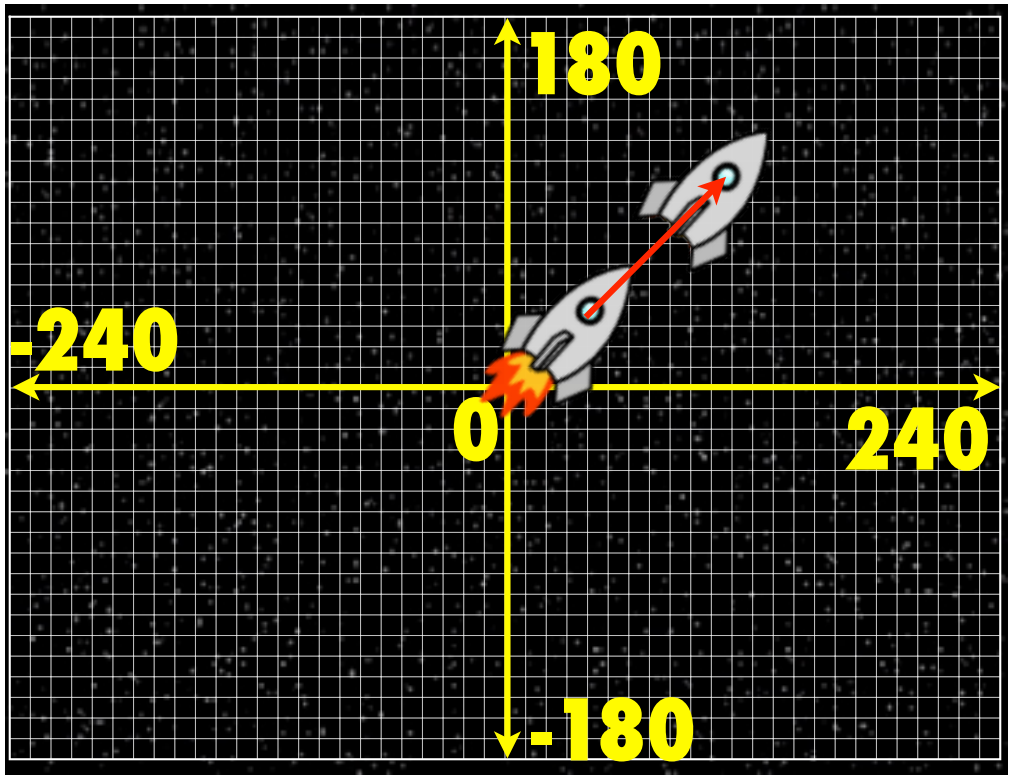

```
public void act()
{
    checkKeys();
}
```

```
/**
 * Check whether there are any key pressed and react to them.
 */
private void checkKeys()
{
    if(Greenfoot.isKeyDown("left")) {
        turnLeftDegrees(5);
    }

    if(Greenfoot.isKeyDown("right")) {
        turnRightDegrees(5);
    }
}
```

Greenfoot Movement - Part 1

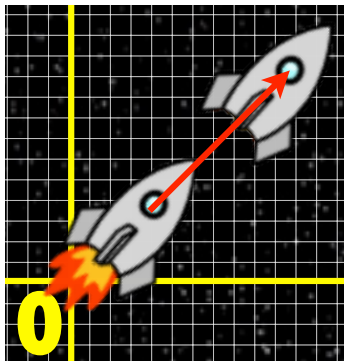






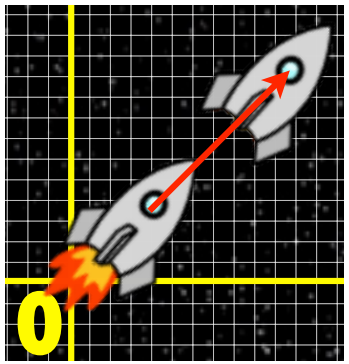

```
{
    setLocation(0, 0);
    setRotation(45);
    setLocation(100,100);
}
```

Greenfoot Movement - Part 1



- **WHAT, NO `move()`???** - There is no "move forward in the current direction" method in the Actor class, so we'll need to make our own
- Thankfully, we already have what we need to do this, which is:
  - the current direction
    - `int getRotation()`
  - a way to get the x and y position
    - `int getX()`
    - `int getY()`
  - a way to set the x and y position
    - `setLocation (int x, int y)`
  - Math functions for x and y portion of a vector
    - `Math.cos(double angle)`
    - `Math.sin(double angle)`

Scratch - Reinventing move



- If you had no 

block in Scratch, you could do the same thing, because in Scratch you also have:
  - the current direction
    - 

  - a way to get the x and y position
    - 

    - 

  - a way to set the x and y position
    - 

  - Math functions for x and y portion of a vector
    - 

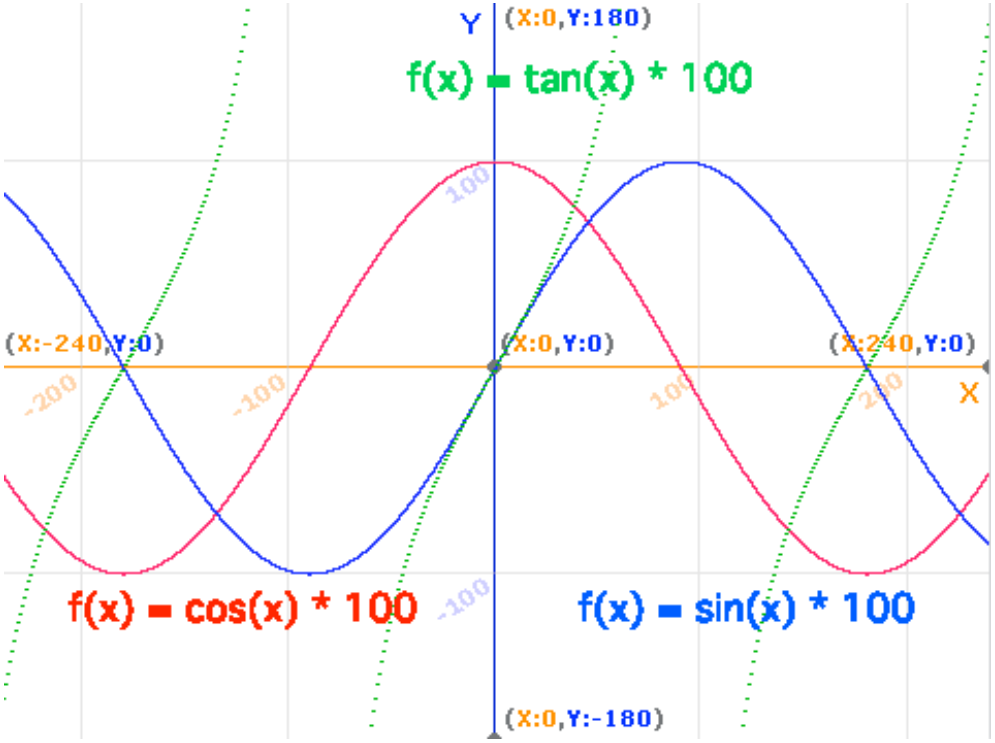
    - 

Scratch - Reinventing move

- Math functions for x and y portion of a vector tell us how much to change x and y by if we're not using 

  - 

  - 

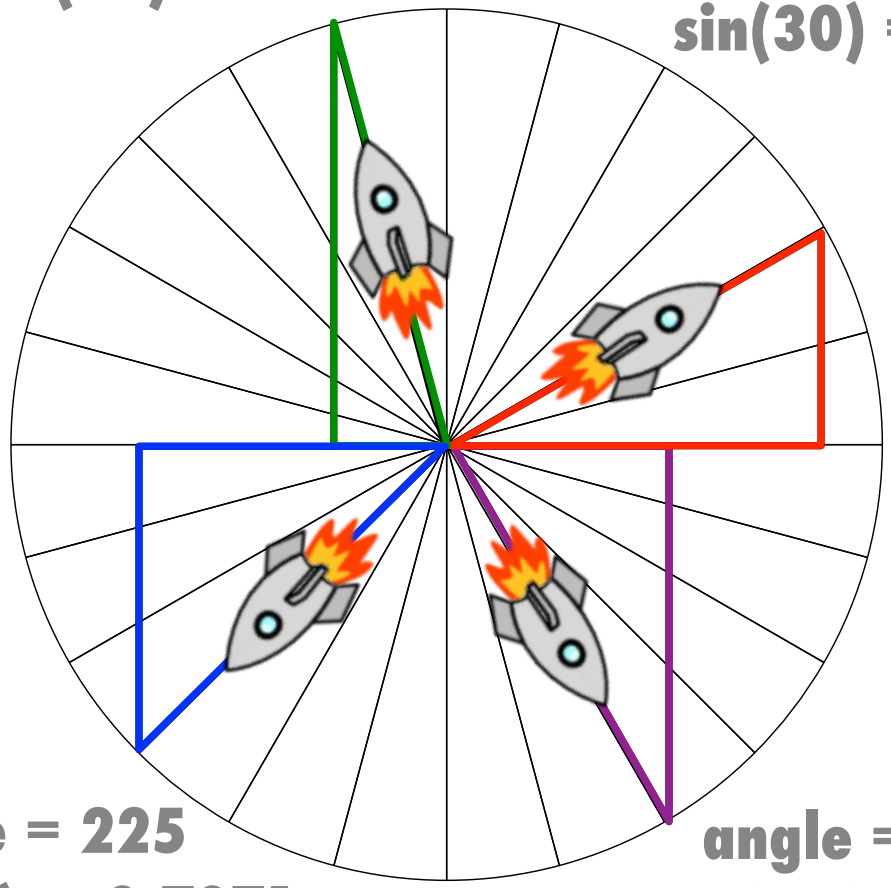


angle=θ	cos(θ)	sin(θ)	tan(θ)
0	1.0	0.0	0.0
15	0.9659	0.2588	0.2679
30	0.8660	0.5	0.5773
45	0.7071	0.7071	1.0
60	0.5	0.8660	1.7320
75	0.2588	0.9659	3.7320
90	0.0	1.0	infinity
105	-0.2588	0.9659	-3.7320
120	-0.5	0.8660	-1.7320
135	-0.7071	0.7071	-1.0
150	-0.8660	0.5	-0.5773
165	-0.9659	0.2588	-0.2679
180	-1.0	0.0	0.0
195	-0.9659	-0.2588	0.2679
210	-0.8660	-0.5	0.5773
225	-0.7071	-0.7071	1.0
240	-0.5	-0.8660	1.7320
255	-0.2588	-0.9659	3.7320
270	0.0	-1.0	infinity
285	0.2588	-0.9659	-3.7320
300	0.5	-0.8660	-1.7320
315	0.7071	-0.7071	-1.0
330	0.8660	-0.5	-0.5773
345	0.9659	-0.2588	-0.2679
360	1.0	0.0	0.0

Scratch - Reinventing move

angle = 105
cos(30) = -0.2588
sin(30) = 0.9659

angle = 30
cos(30) = 0.866
sin(30) = 0.5

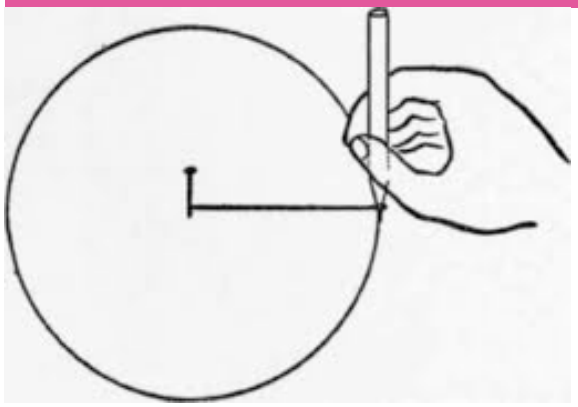


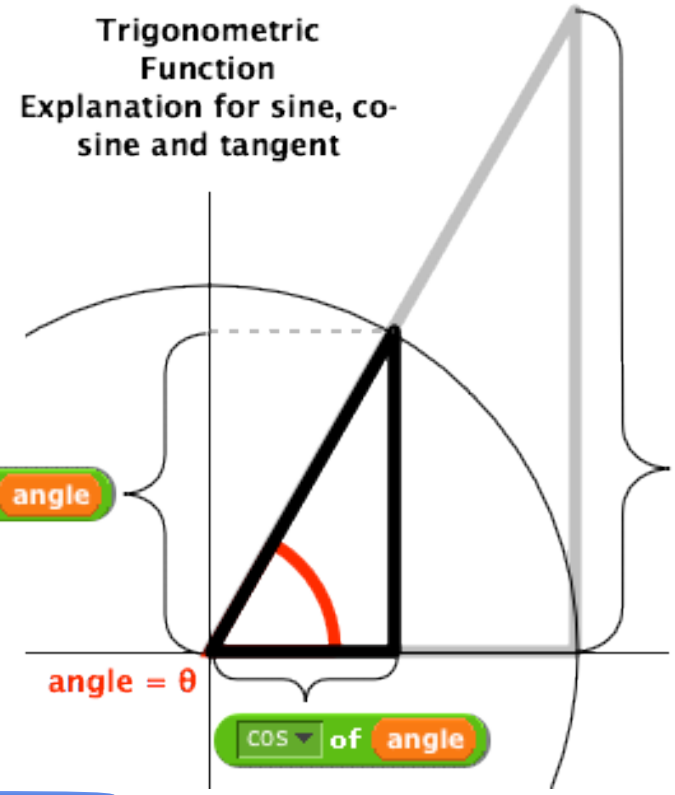
angle = 225
cos(30) = -0.7071
sin(30) = -0.7071

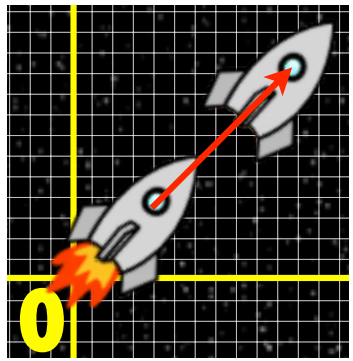
angle = 300
cos(30) = 0.5
sin(30) = -0.866

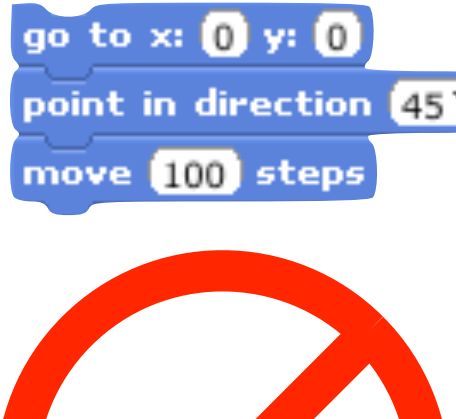
angle=θ	cos(θ)	sin(θ)	tan(θ)
0	1.0	0.0	0.0
15	0.9659	0.2588	0.2679
30	0.8660	0.5	0.5773
45	0.7071	0.7071	1.0
60	0.5	0.8660	1.7320
75	0.2588	0.9659	3.7320
90	0.0	1.0	infinity
105	-0.2588	0.9659	-3.7320
120	-0.5	0.8660	-1.7320
135	-0.7071	0.7071	-1.0
150	-0.8660	0.5	-0.5773
165	-0.9659	0.2588	-0.2679
180	-1.0	0.0	0.0
195	-0.9659	-0.2588	0.2679
210	-0.8660	-0.5	0.5773
225	-0.7071	-0.7071	1.0
240	-0.5	-0.8660	1.7320
255	-0.2588	-0.9659	3.7320
270	0.0	-1.0	infinity
285	0.2588	-0.9659	-3.7320
300	0.5	-0.8660	-1.7320
315	0.7071	-0.7071	-1.0
330	0.8660	-0.5	-0.5773
345	0.9659	-0.2588	-0.2679
360	1.0	0.0	0.0

Scratch - Reinventing move





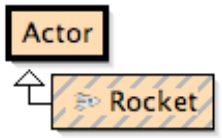


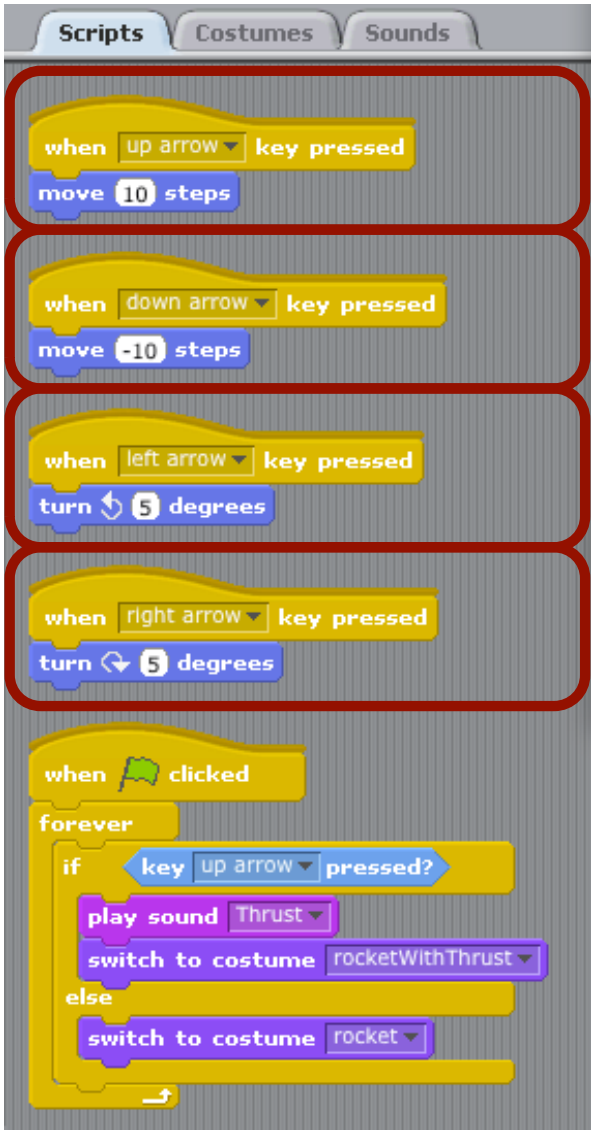


- Since cos() and sin() return a value beteween 0 and 1, multiply the result by the steps you would have used in the move block to get the x and y component

Greenfoot Phase 1: Get Moving

```
public class Rocket extends Actor
{
    /**
     * Act - do whatever the Rocket wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        checkKeys();
    }

    /**
     * Check whether there are any key pressed and react to them.
     */
    private void checkKeys()
    {
        if(Greenfoot.isKeyDown("up")) {
            moveSteps(10);
        }
        if(Greenfoot.isKeyDown("down")) {
            moveSteps(-10);
        }
        if(Greenfoot.isKeyDown("left")) {
            turnLeftDegrees(5);
        }
        if(Greenfoot.isKeyDown("right")) {
            turnRightDegrees(5);
        }
    }
```



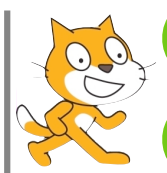


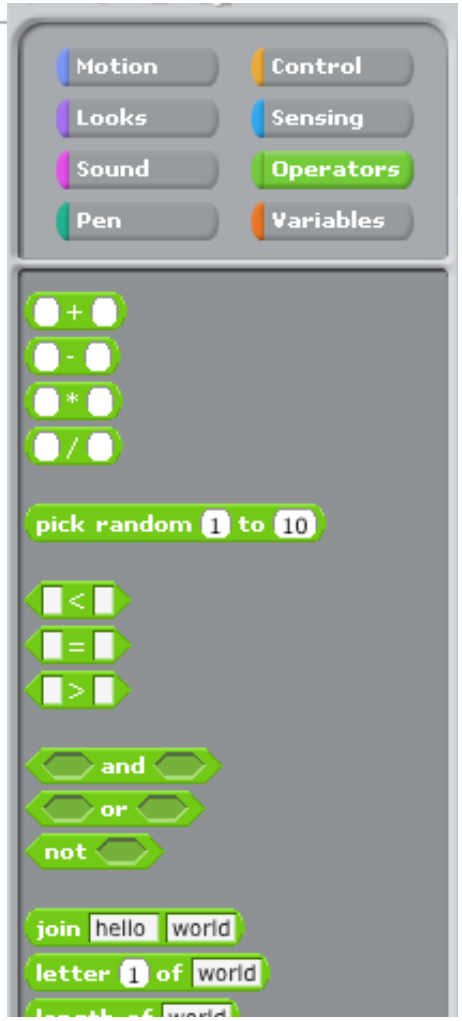
java.lang.Math

java.lang

Class Math

java.lang.Object
└ java.lang.Math





`public final class` **`Math`**
`extends` Object

The class `Math` contains methods for performing basic numeric operations such as the elementary exponential, logarithm, square root, and trigonometric functions.

Method Detail

sin

`public static double` **`sin`**`(double a)`

Returns the trigonometric sine of an angle.

Parameters:
`a` - an angle, in radians.
Returns:
the sine of the argument.

cos

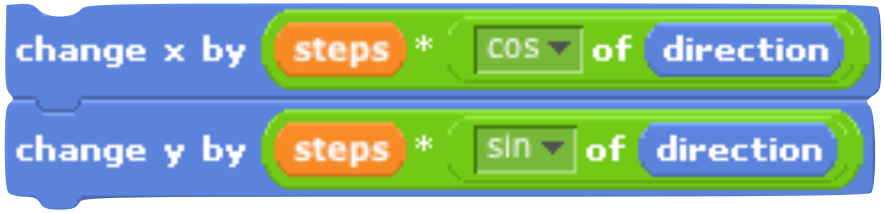
`public static double` **`cos`**`(double a)`

Returns the trigonometric cosine of an angle.

Parameters:
`a` - an angle, in radians.
Returns:
the cosine of the argument.

Greenfoot - Inventing move()

```
public void moveSteps(int steps)
{
    int x1;
    int x2;
    int y1;
    int y2;
    int direction;

    x1 = getX();
    y1 = getY();
    direction = getRotation();
    // new x value is old x value plus the x component of current direction
    x2 = x1 + (steps * (Math.cos(direction)) );
    // new y value is old y value minus the y component of current direction
    y2 = y1 + (steps * (Math.sin(direction)) );

    setLocation(x2, y2);
}
```



TAKE 1

- Try compiling this, does everything come out OK?

Greenfoot - Inventing move()

```
public void moveSteps(int steps)
{
    int x1;
    int x2;
    int y1;
    int y2;
    int direction;

    x1 = getX();
    y1 = getY();
    direction = getRotation();
    // new x value is old x value plus the x component of current direction
    x2 = x1 + (steps * (Math.cos(direction)) );
    // new y value is old y value minus the y component of current direction
    y2 = y1 + (steps * (Math.sin(direction)) );

    setLocation(x2, y2);
}
```

FAIL!

```
possible loss of precision
found   : double required: int
```

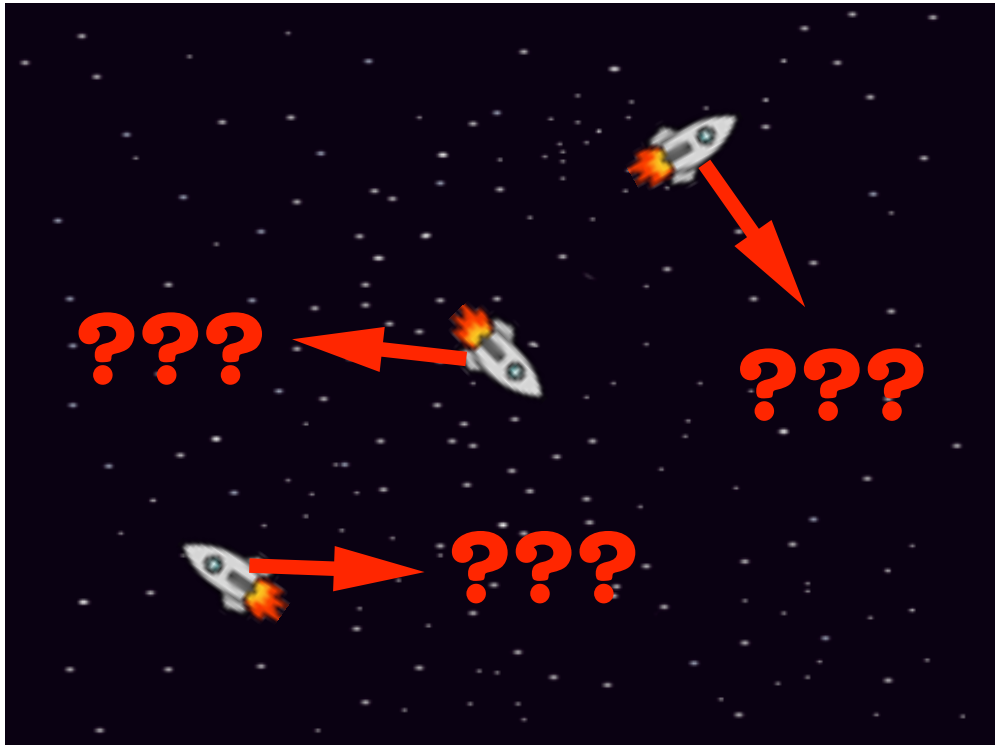
Greenfoot - Inventing move()

```
public void moveSteps(int steps)
{
    int x1;
    int x2;
    int y1;
    int y2;
    int direction;

    x1 = getX();
    y1 = getY();
    direction = getRotation();
    // new x value is old x value plus the x component of current direction
    x2 = x1 + (steps * (int) (Math.cos(direction)) );
    // new y value is old y value minus the y component of current direction
    y2 = y1 + (steps * (int) (Math.sin(direction)) );

    setLocation(x2, y2);
}
```

TAKE 2

- It now compiles but… does it look right?

Greenfoot - Inventing move()



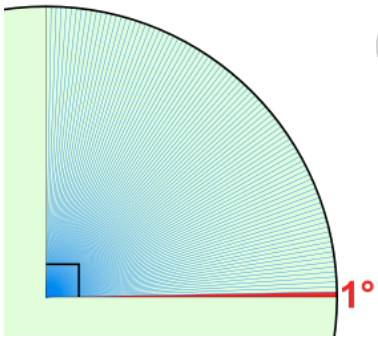
FAIL!

cos

```
public static double cos(double a)
```

Returns the trigonometric cosine of an angle.

Parameters:
a - an angle, in radians.
Returns:
the cosine of the argument.

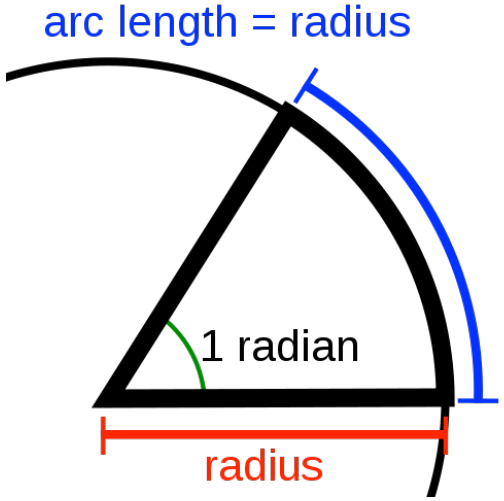


sin

```
public static double sin(double a)
```

Returns the trigonometric sine of an angle.

Parameters:
a - an angle, in radians.
Returns:
the sine of the argument.



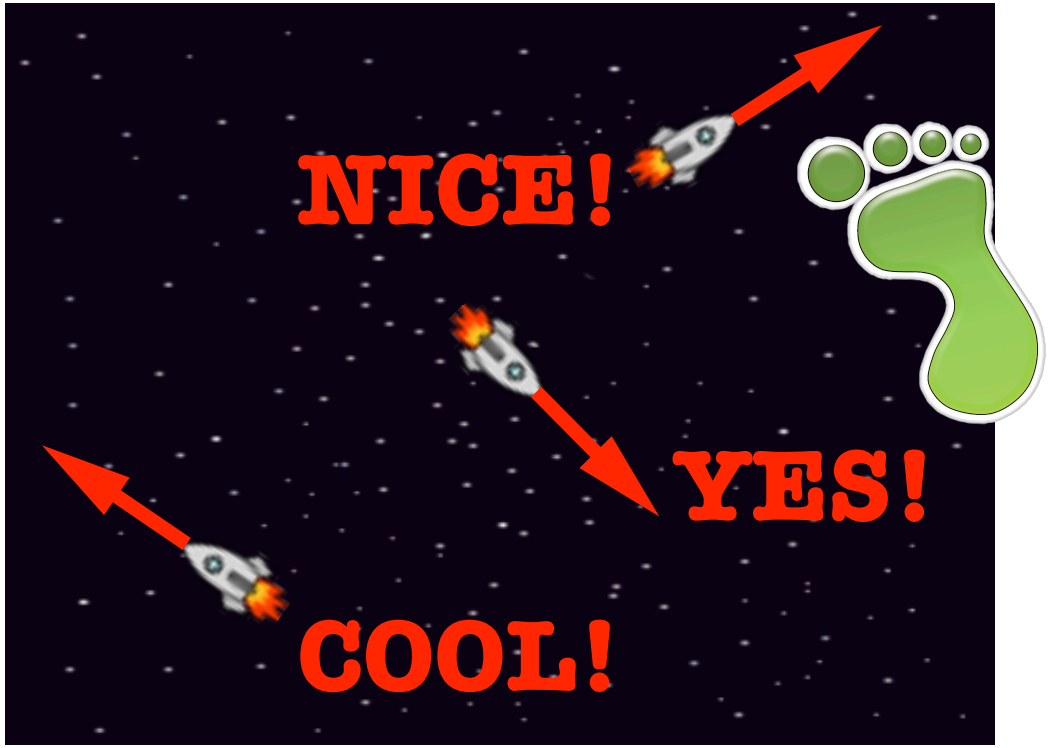
Greenfoot - Inventing move()

```
public void moveSteps(int steps)
{
    int x1;
    int y1;
    int x2;
    int y2;
    int direction;

    // get the status of the rocket - x value, y value and direction
    x1 = getX();
    y1 = getY();
    direction = getRotation();
    // new x value is old x value plus the x component of current direction
    x2 = x1 + (int)(steps * ( Math.cos( Math.toRadians( direction ) ) ) );
    // new y value is old y value minus the y component of current direction
    y2 = y1 + (int)(steps * ( Math.sin( Math.toRadians( direction ) ) ) );

    setLocation(x2, y2);
}
```

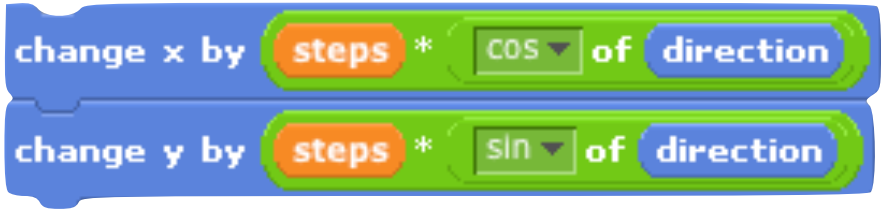
TAKE 3

- Does our ship behave now?

Greenfoot - Inventing move()



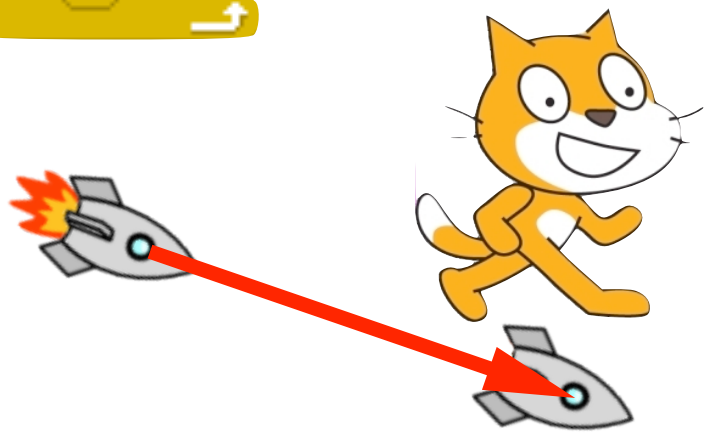
- We now have a working `moveSteps(int steps)` for our Rocket Actor class in Greenfoot
  - Which behaves like this



- Which does the same thing as this

YEAH!

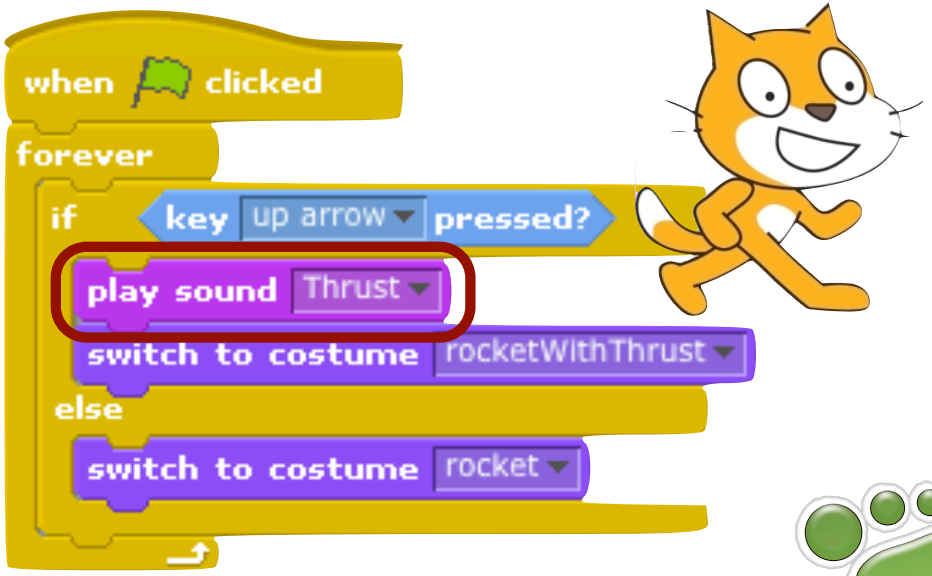
Greenfoot - change costumes

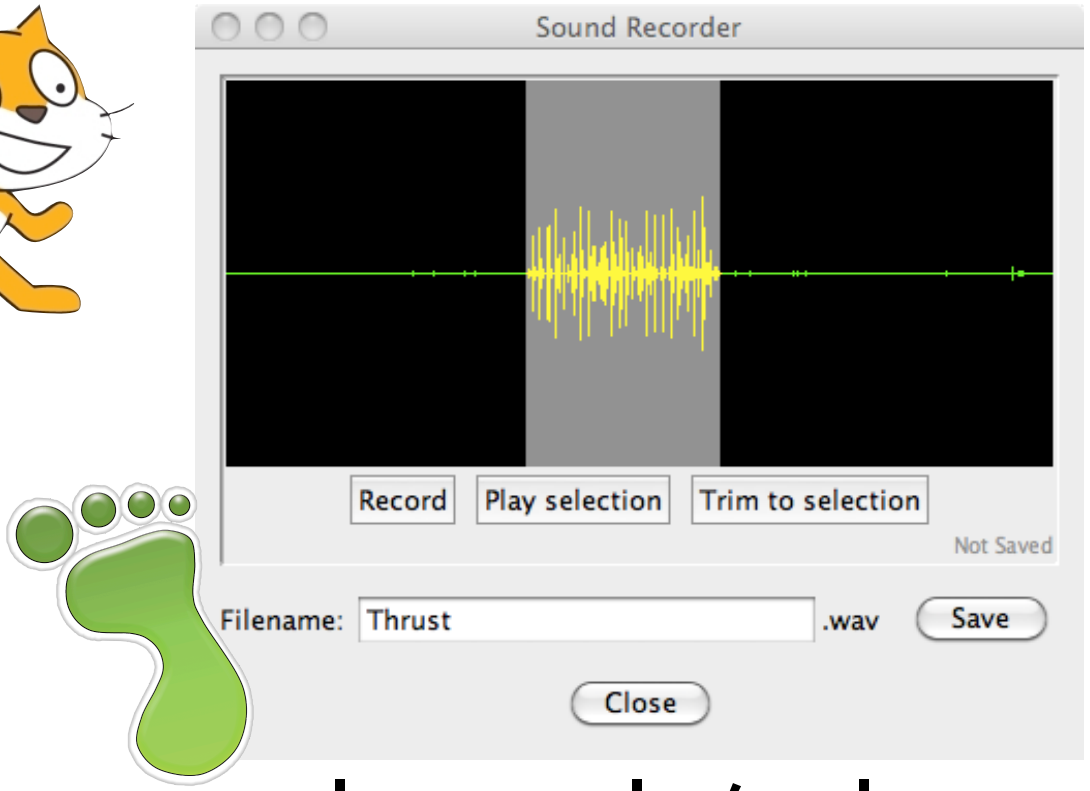


- Since we're moving correctly now, let's add image changes
- use **`Actor.setImage(String)`**

```
private void checkKeys()
{
    if(Greenfoot.isKeyDown("up")) {
        setImage("rocketWithThrust.png");
        moveSteps(10);
    }
    else {
        setImage("rocket.png");
    }
```

Greenfoot - make noises





- Since we're moving correctly now, let's play a sound when we **`moveSteps(int)`**
- Create a new sound and use **`GreenfootSound.play()` to play it**

Greenfoot - GreenfootSound

greenfoot

Class GreenfootSound

```
java.lang.Object
  greenfoot.GreenfootSound
```

```
public class GreenfootSound
extends java.lang.Object
```

Represents audio that can be played in Greenfoot. A GreenfootSound loads the audio from a file. The sound cannot be played several times simultaneously, but can be played several times sequentially. Most files of the following formats are supported: AIFF, AU, WAV, MP3 and MIDI.

Constructor Summary

`GreenfootSound(java.lang.String filename)`
Creates a new sound from the given file.

Method Summary

`boolean isPlaying()`
True if the sound is currently playing.

`void pause()`
Pauses the current sound if it is currently playing.

`void play()`
Start playing this sound.

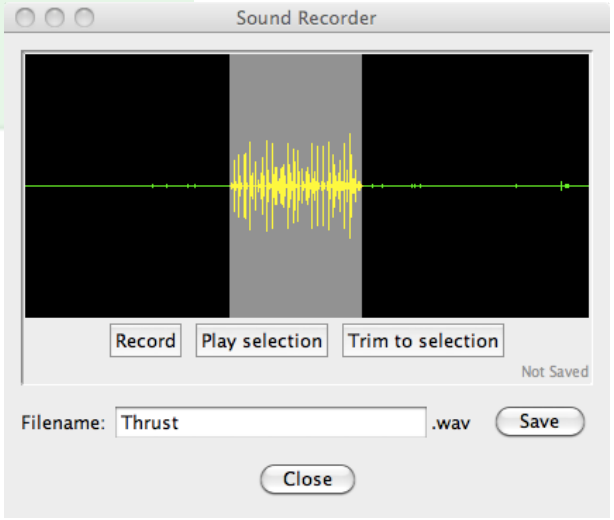
`void playLoop()`
Play this sound repeatedly in a loop.

`void stop()`
Stop playing this sound if it is currently playing.

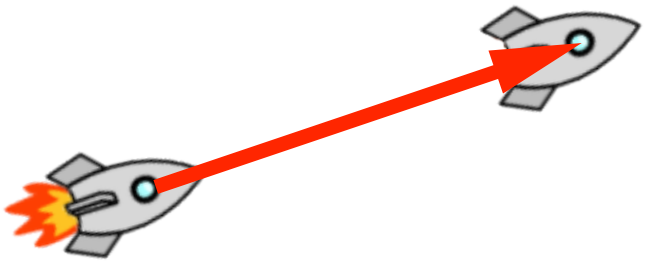
`java.lang.String toString()`
Returns a string representation of this sound containing the name of the file and whether it is currently playing or not.

Greenfoot - make noises

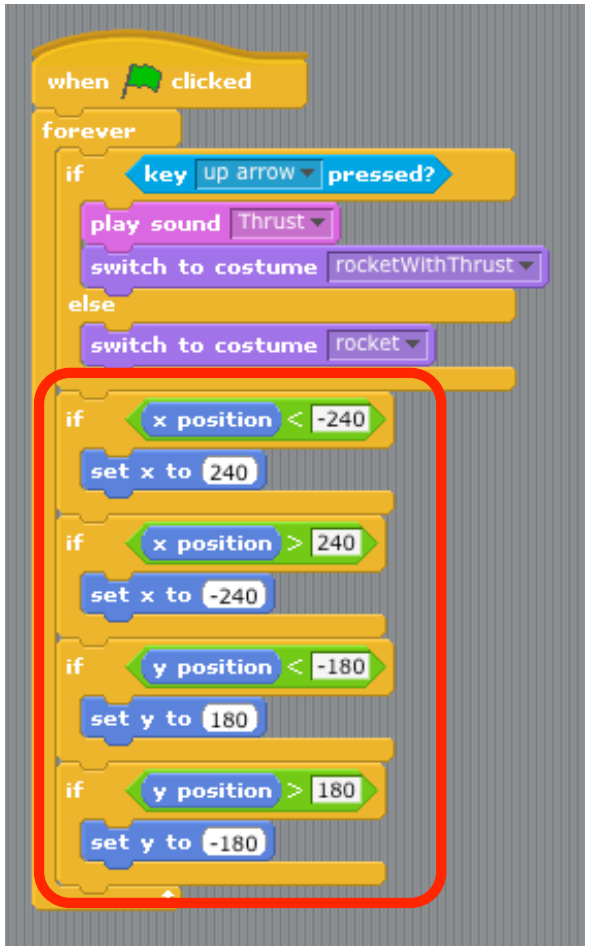
```
public class Rocket extends Actor
{
    GreenfootSound thrustSound;

    /**
     * Rocket constructor
     */
    {
        thrustSound = new GreenfootSound("Thrust.wav");
    }
```

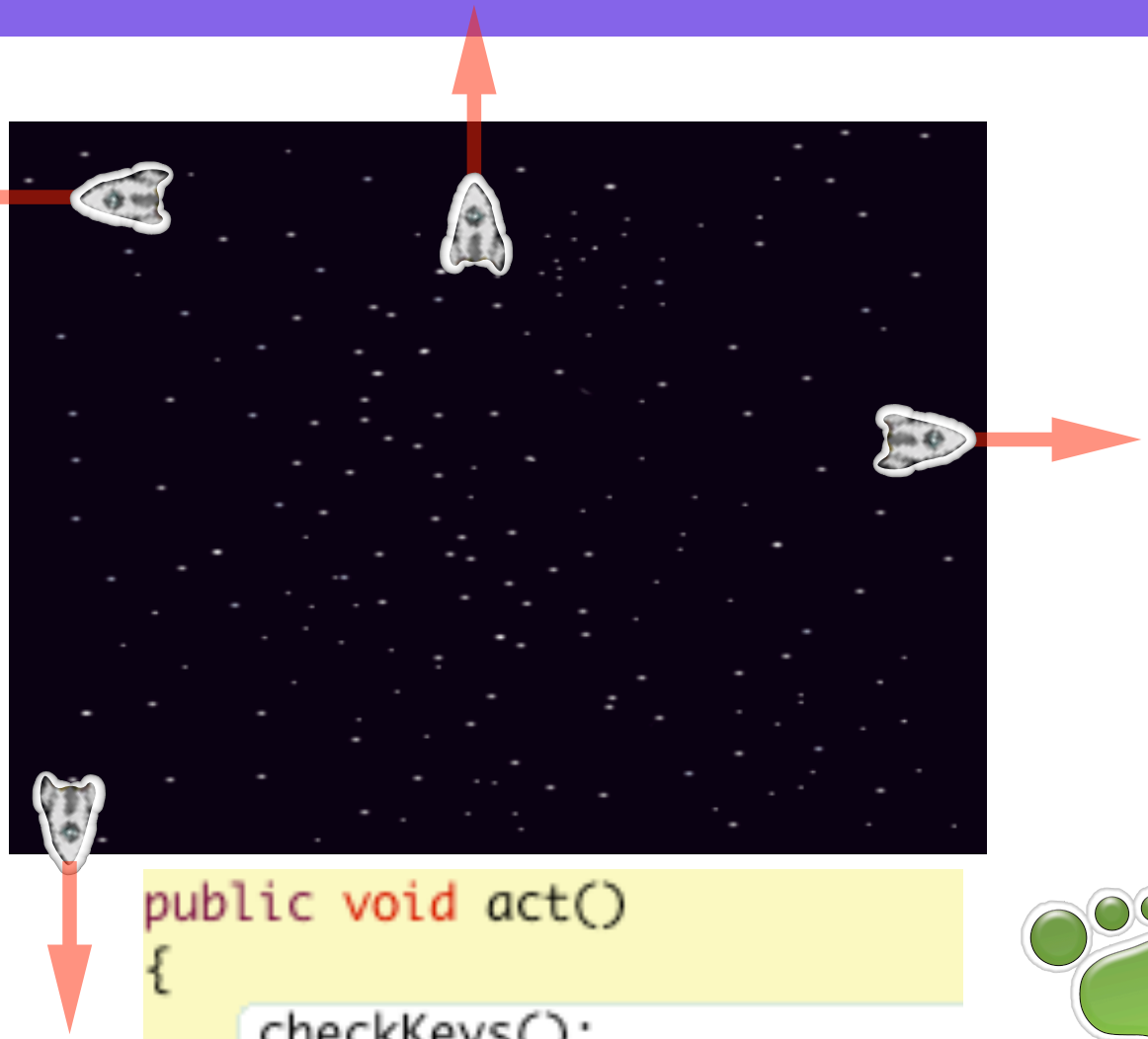



```
private void checkKeys()
{
    if(Greenfoot.isKeyDown("up")) {
        setImage("rocketWithThrust.png");
        thrustSound.play();
        moveSteps(10);
    }
```



Greenfoot Phase 2: Off Screen

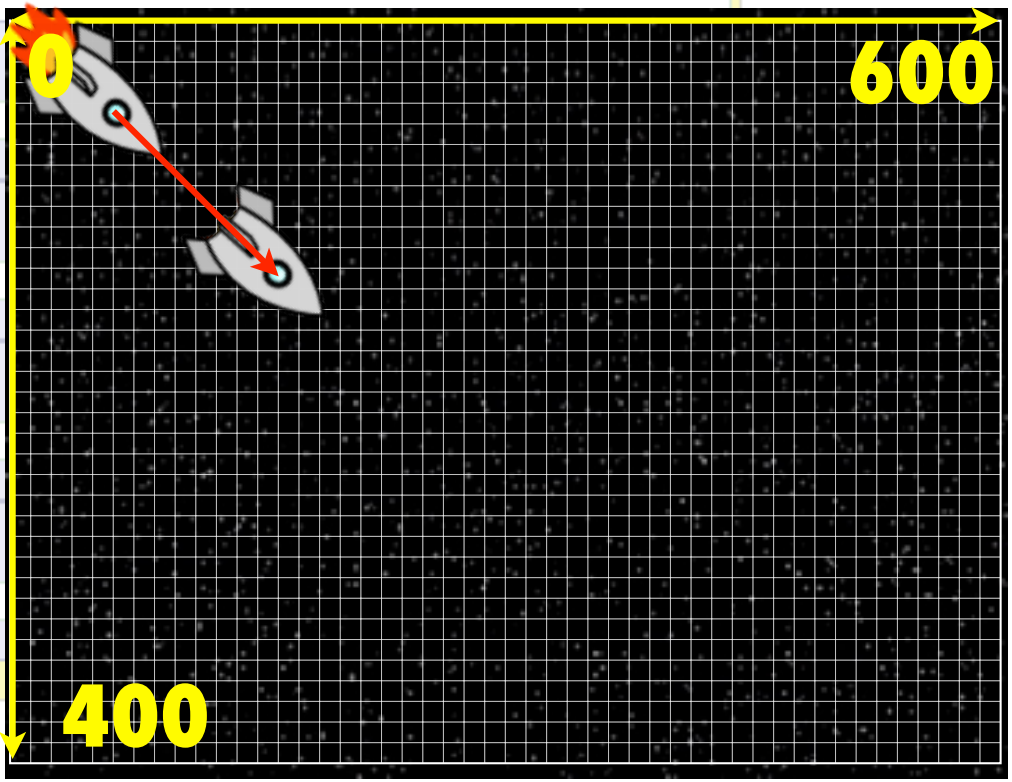




```
public void act()
{
    checkKeys();
    checkOffScreen();
}
```

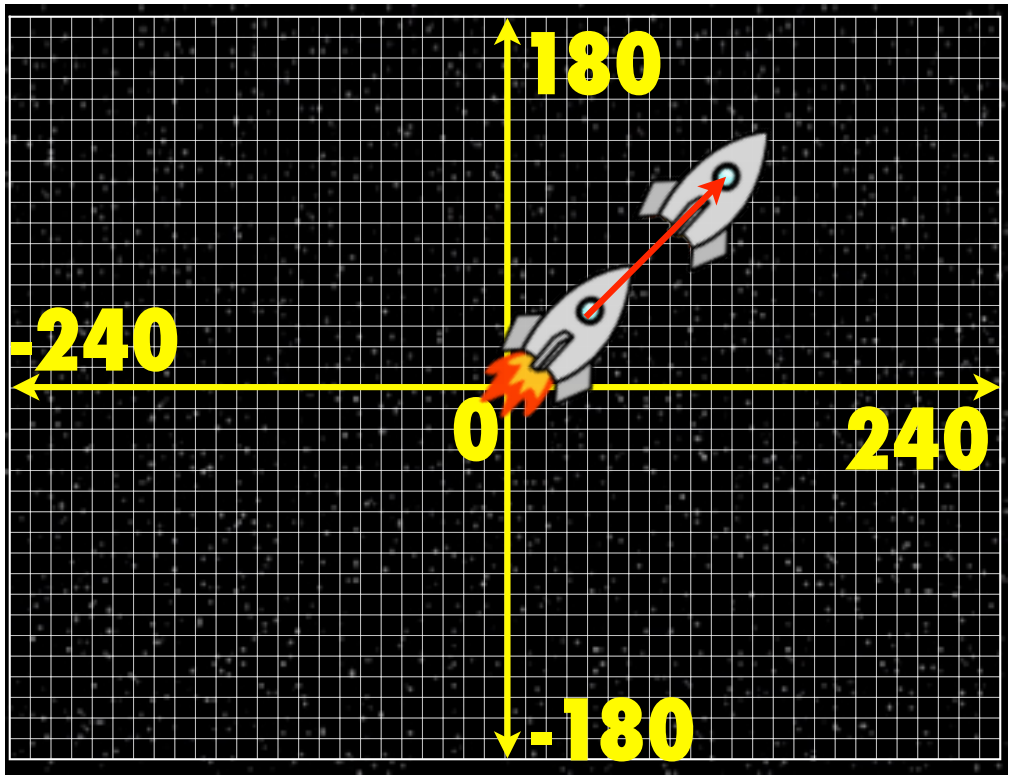
Greenfoot Phase 2: Off Screen

```
/**
 * Check whether the rocket is headed off screen and put it on the other side of the screen
 */
public void checkOffScreen()
{
    int x;
    int y;

    x = getX();
    y = getY();

    if (/* off left side of screen*/)
    {
        /* do something */
    }
    if (/* off right side of screen */)
    {
        /* do something */
    }
    if (/* off top of screen */)
    {
        /* do something */
    }
    if (/* off bottom of screen */)
    {
        /* do something */
    }
}
```



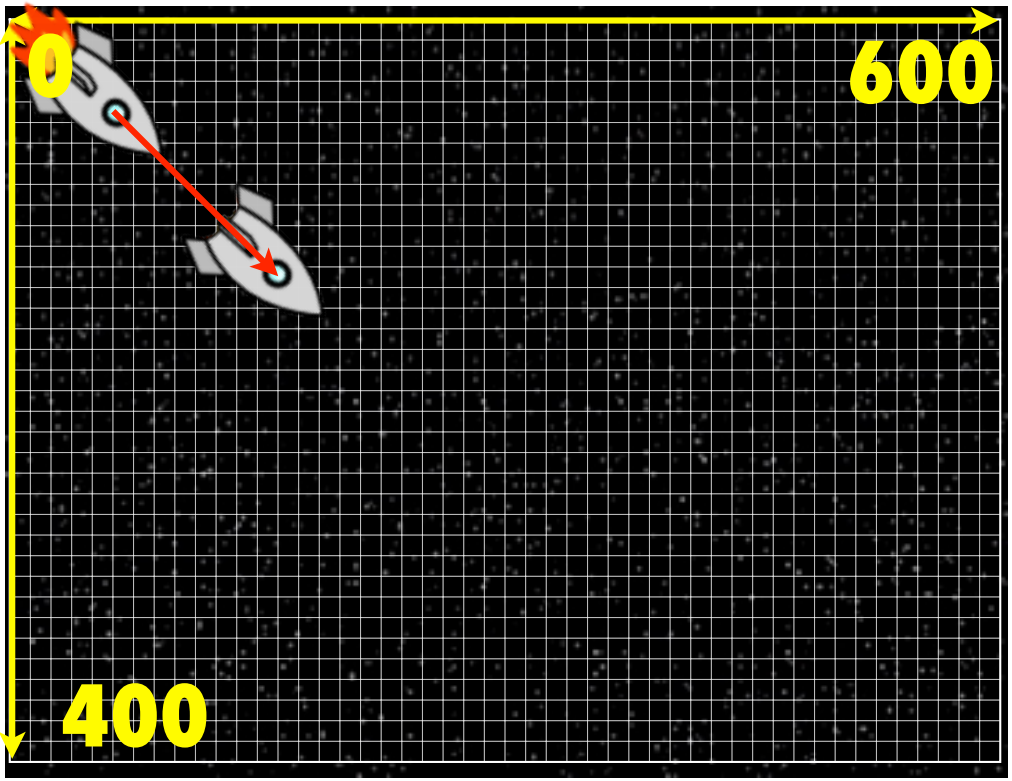
Scratch - how far is too far?

- If the Scratch screen boundary is 240 left and right, can x be > 240 or < -240?
- If the Scratch screen boundary is 180 up and down, can y be > 180 or < -180?



Greenfoot - how far is too far?

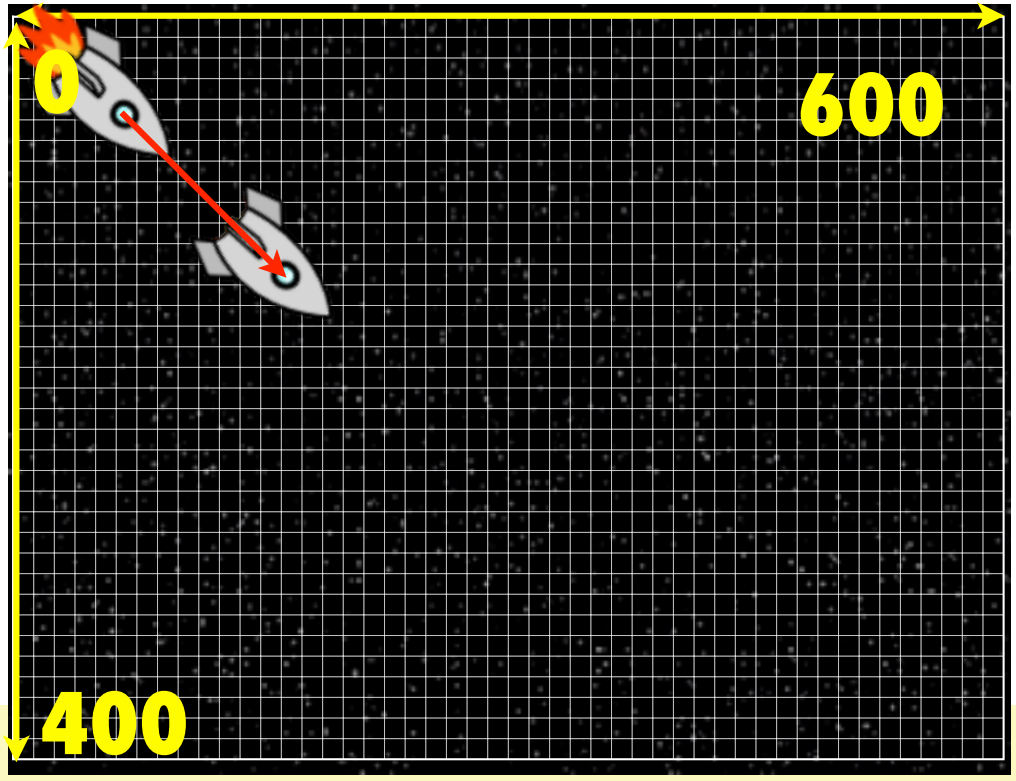
- If the Greenfoot screen boundary is set to 0 to 600 left and right, can x be > 600 or < 0?
- If the Greenfoot screen boundary is set to 0 to 400 up and down, can y be > 400 or < 0?




```
private void checkKeys()
{
    if(Greenfoot.isKeyDown("up")) {
        setLocation(getX(), getY() + 1);
    }
    if(Greenfoot.isKeyDown("down")) {
        setLocation(getX(), getY() - 1);
    }

    if(Greenfoot.isKeyDown("left")) {
        setLocation(getX() - 1, getY());
    }

    if(Greenfoot.isKeyDown("right")) {
        setLocation(getX() + 1, getY());
    }
}
```

Greenfoot Phase 2: Off Screen

```
public class Rocket extends Actor
{

    GreenfootSound thrustSound;
    int screenLeft;
    int screenRight;
    int screenUp;
    int screenDown;
```




```
public void addedToWorld(World w) {
    screenLeft = 0;
    screenRight = w.getWidth();
    screenUp = 0;
    screenDown = w.getHeight();
    System.out.println("screenRight=" + screenRight + ", screenDown=" +
}
```

Greenfoot Phase 2: Off Screen

```
public void checkOffScreen()
{
    int x;
    int y;

    x = getX();
    y = getY();

    if (x <= screenLeft)
    {
        x = screenRight - 1;
    } else
    if (x >= screenRight - 1)
    {
        x = screenLeft;
    }
    if (y <= screenUp)
    {
        y = screenDown - 1;
    } else
    if (y >= screenDown - 1)
    {
        y = screenUp;
    }
    setLocation(x, y);
}
```

- This can be tricky to get right and not have things "stuck" on a side of the screen or the corner
- Greenfoot 2.0 also includes a "border-less" option for Worlds:

`public abstract class` **`World`**
`extends java.lang.Object`

World is the world that Actors live in. It is a two-dimensional grid of cells.

All Actor are associated with a World and can get access to the world object. The size of cells can be specified at world creation time, and is constant after creation. Simple scenarios may use large cells that entirely contain the representations of objects in a single cell. More elaborate scenarios may use smaller cells (down to single pixel size) to achieve fine-grained placement and smoother animation.

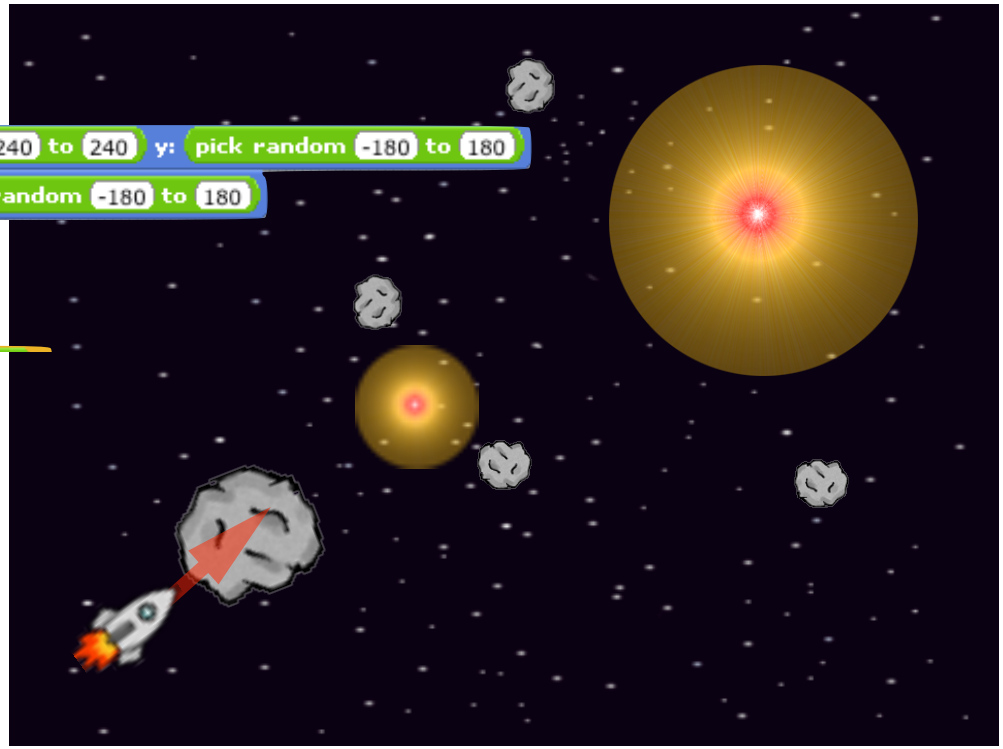
The world background can be decorated with drawings or images.

Constructor Summary

`World(int worldWidth, int worldHeight, int cellSize)`
Construct a new world.

`World(int worldWidth, int worldHeight, int cellSize,` **`boolean bounded`**`)`
Construct a new world.

Greenfoot Phase 3: Collisions



- To provide something to collide with, our Asteroid Actors will be random
- The Greenfoot version of 

will use the `java.util.Random.nextInt(int max)` method

java.util.Random

java.util

Class Random

[java.lang.Object](#)
  java.util.Random

public class **Random** extends [Object](#)

An instance of this class is used to generate a stream of pseudorandom numbers. The class uses a 48-bit seed, which is modified using a linear congruential formula.

Method Detail

int [**nextInt**](#)(int n)

Returns a pseudorandom, uniformly distributed int value between 0 (inclusive) and the specified value (exclusive), drawn from this random number generator's sequence.

Greenfoot Phase 3: Collisions

```
public class Asteroid extends Actor
{
    GreenfootSound explodeSound;
    int screenLeft;
    int screenRight;
    int screenUp;
    int screenDown;
    Random random;
```

```
public void act()
{
    checkOffScreen();
    checkForCollision();
    moveSteps(2);
}
```

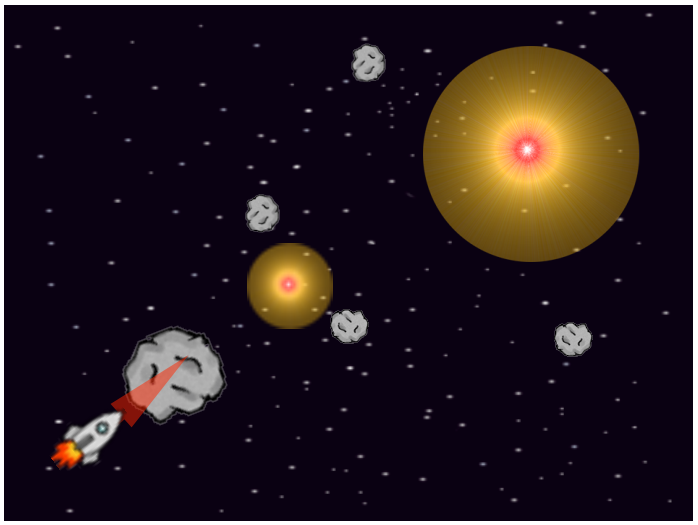
```
public Asteroid() {
    explodeSound = new GreenfootSound("Explosion.wav");
    random = new Random();
}
```

```
public void addedToWorld(World w) {
    screenLeft = 0;
    screenRight = w.getWidth();
    screenUp = 0;
    screenDown = w.getHeight();
    setRotation(random.nextInt(360));
}
```

Actor - Collision Detection

greenfoot

Class Actor



```
java.lang.Object
  greenfoot.Actor
public abstract class Actor
extends java.lang.Object
```

An Actor is an object that exists in the Greenfoot world.

Every Actor has a location in the world, and an appearance

(that is: an icon).

Method Summary

java.util.List `getIntersectingObjects(java.lang.Class cls)`
Return all the objects that intersect this object.

java.util.List `getNeighbours(int distance, boolean diagonal, java.lang.Class cls)`
Return the neighbours to this object within a given distance.

java.util.List `getObjectsAtOffset(int dx, int dy, java.lang.Class cls)`
Return all objects that intersect the center of the given location (relative to this object's location).

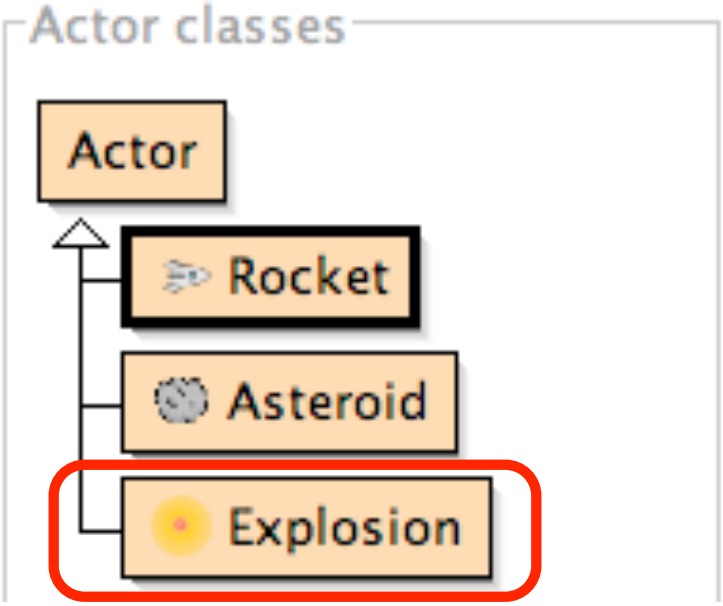
java.util.List `getObjectsInRange(int radius, java.lang.Class cls)`
Return all objects within range 'radius' around this object.

Actor `getOneIntersectingObject(java.lang.Class cls)`
Return an object that intersects this object.

Actor `getOneObjectAtOffset(int dx, int dy, java.lang.Class cls)`
Return one object that is located at the specified cell (relative to this objects location).

Greenfoot Phase 3: Collisions

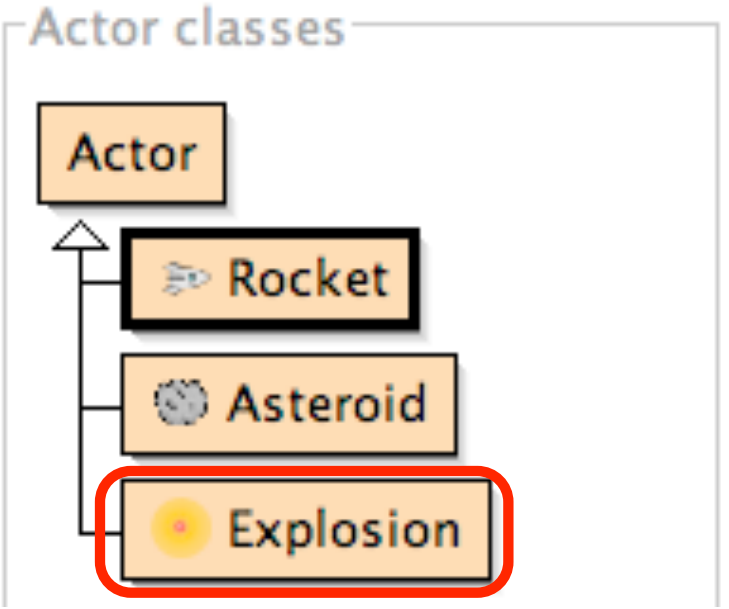
```
public class Rocket extends Actor
{
public void act()
{
    checkKeys();
    checkOffScreen();
    checkCollision();
}
```




```
private void checkCollision() {
    Asteroid a;

    a = (Asteroid) getOneIntersectingObject(Asteroid.class);

    // any object returned means there's a collision, no need to determine how man
    // if there's a collision, replace the rocket with an explosion
    if(a != null) {
        getWorld().addObject(new Explosion(), getX(), getY());
        getWorld().removeObject(this);
    }
}
```

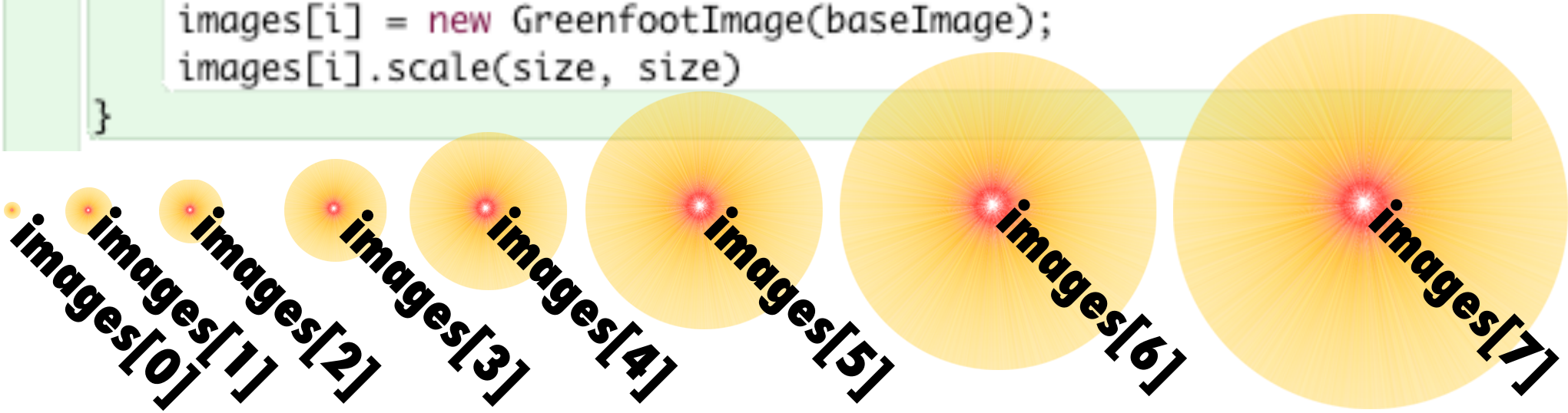
Greenfoot Phase 3: Collisions

```
public class Explosion extends Actor
{
    /** How many images should be used in the animation of the explosion */
    private static int IMAGE_COUNT = 8;
    /**
     * The images in the explosion. This is static so the images are not
     * recreated for every object (improves performance significantly).
     */
    private static GreenfootImage[] images;
    /** Current size of the explosion */
    private int imageNo;
    /** How much do we increment the index in the explosion animation. */
    private int increment;


    public Explosion() {
        IMAGE_COUNT = 8;
        imageNo = 0;
        increment = 1;

        initialiseImages();
        setImage(images[0]);
        Greenfoot.playSound("Explosion.wav");
    }
```



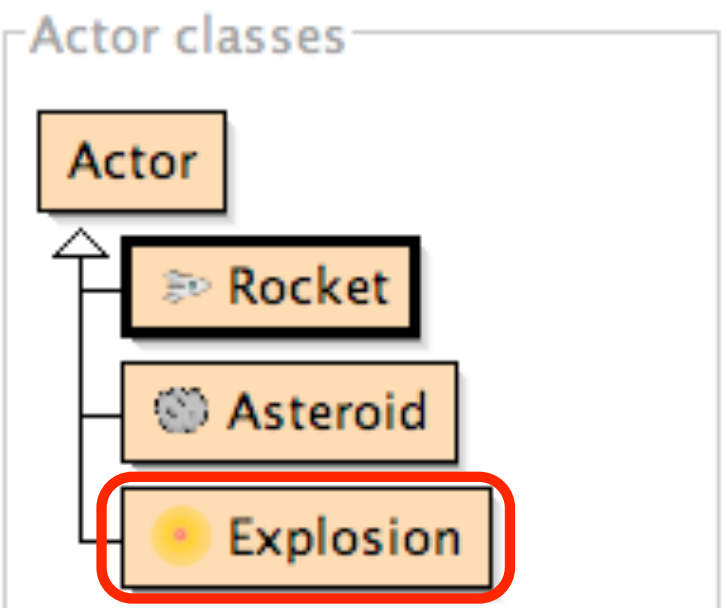
array[] of explosion images

```
public class Explosion extends Actor
{
    private static GreenfootImage[] images;
    GreenfootImage baseImage = new GreenfootImage("explosion-big.png");
    int maxSize = baseImage.getWidth();
    int delta = maxSize / IMAGE_COUNT;
    int size = 0;
    images = new GreenfootImage[IMAGE_COUNT];
    for(int i=0; i < IMAGE_COUNT; i++) {
        size = size + delta;
        images[i] = new GreenfootImage(baseImage);
        images[i].scale(size, size)
    }
```



Greenfoot Phase 3: Collisions

```
public void act()
{
    setImage(images[imageNo]);

    imageNo += increment;
    if(imageNo >= IMAGE_COUNT) {
        increment = -increment;
        imageNo += increment;
    }

    if(imageNo < 0) {
        getWorld().removeObject(this);
    }
}
```

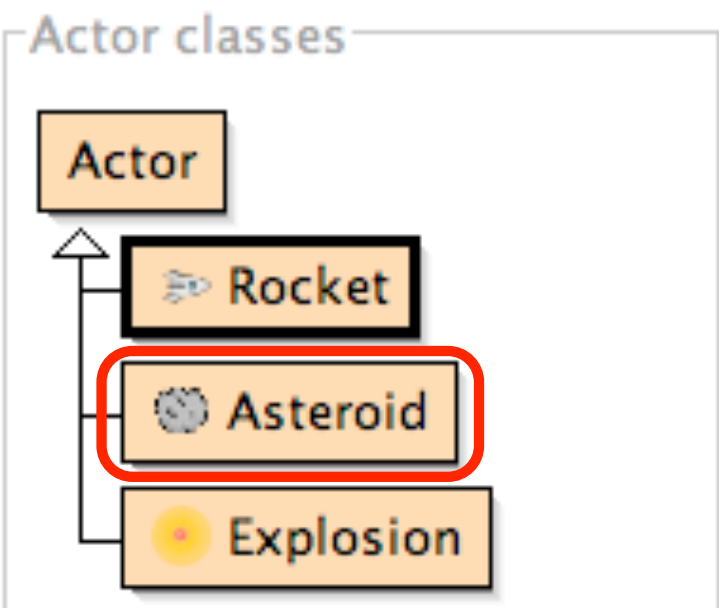


- Resulting order of `images[]` elements set by `act()` method:

0 1 2 3 4 5 6 7 6 5 4 3 2 1 0

Greenfoot Phase 3: Collisions

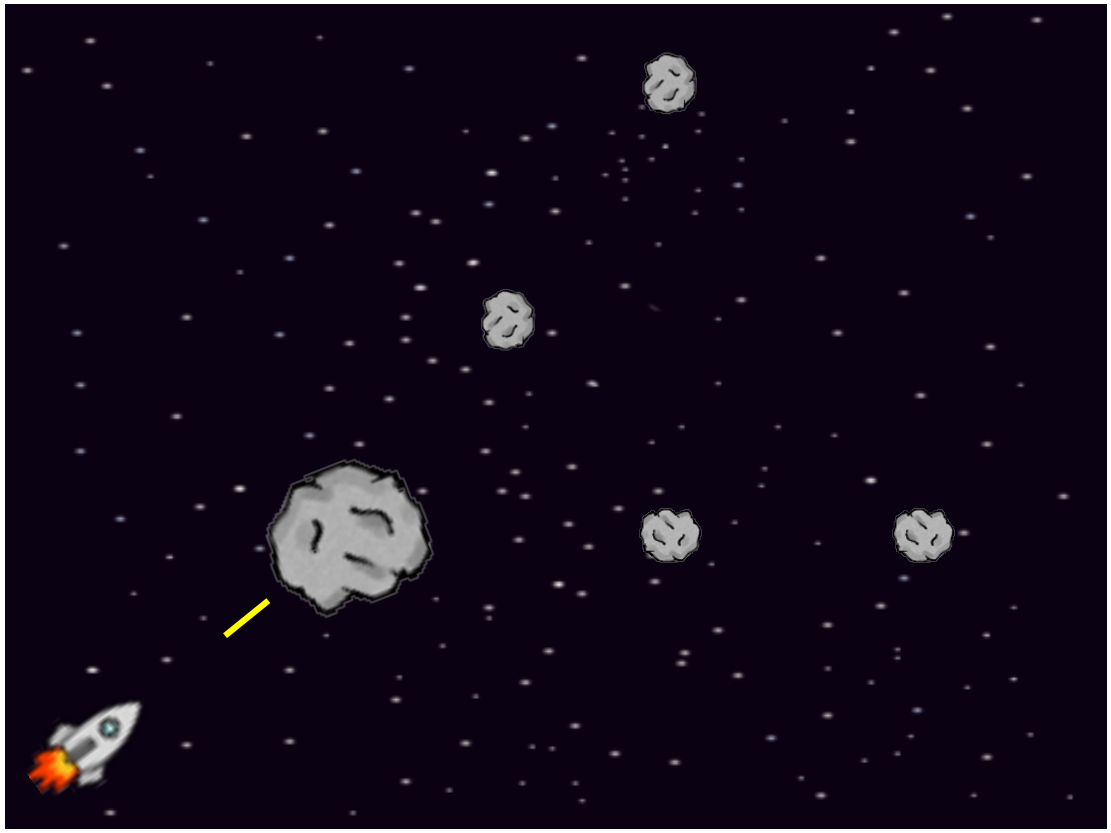
```
public void act()
{
    setImage(images[imageNo]);

    imageNo += increment;
    if(imageNo >= IMAGE_COUNT) {
        increment = -increment;
        imageNo += increment;
    }

    if(imageNo < 0) {
        getWorld().removeObject(this);
    }
}
```



- Resulting order of `images[]` elements set by `act()` method:

0 1 2 3 4 5 6 7 6 5 4 3 2 1 0

Scratch Phase 4: Shooting



Scratch Phase 5: Momentum

TO BE COMPLETED "REAL SOON NOW"(C)(R)(Pat. Pending)