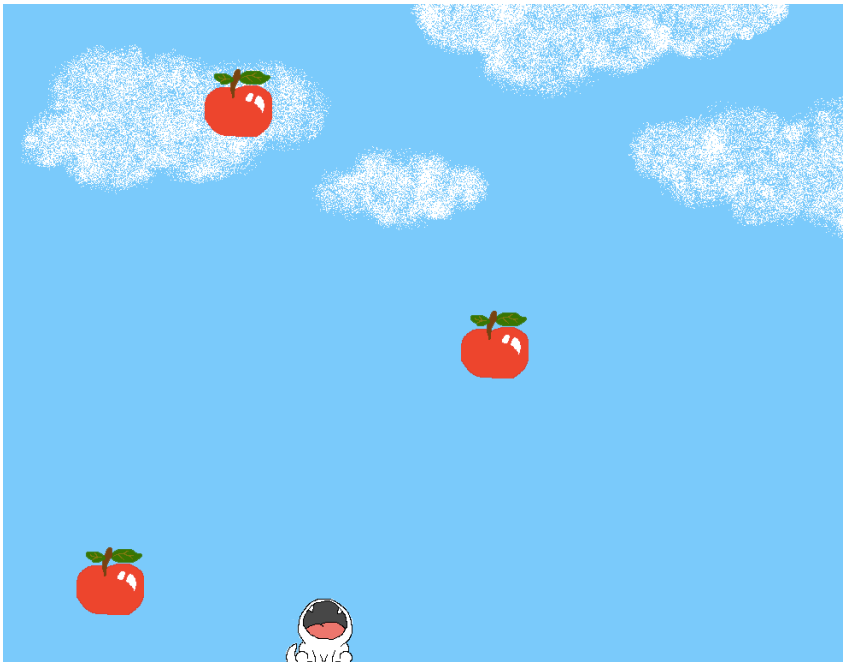
The Feed Me Game

This activity focuses on a more interesting program – a greedy apple-eating monster dog running around and eating apples that fall out of the sky!



Demo – you are the apple-eating monster dog! Catch as many apples as you can!

There are six coding files in this program, along with many image and sound files used in the game. Here's a summary of the files used in this game:

- `Game.java` – initializes a new game and runs it
- `SoundEffect.java` – manages the sound files used in the game
- `Board.java` – coordinates all the game pieces from start to end
- `Sprite.java` – manages the monster dog, how fast it moves, which direction it moves to when a specific arrow key is pressed
- `Drop.java` – manages a collection of falling apples and determines where and how many apples are dropped from the sky
- `Apple.java` – manages the falling object

- `Laser-01.wav, alien-noise-01.wav, cheer-01.wav, Move Forward86.wav` – audio files for sound effects and background music

- `Apple.png, doghappy.png, dogopen.png, sky.png` – image files for the objects and background in the game

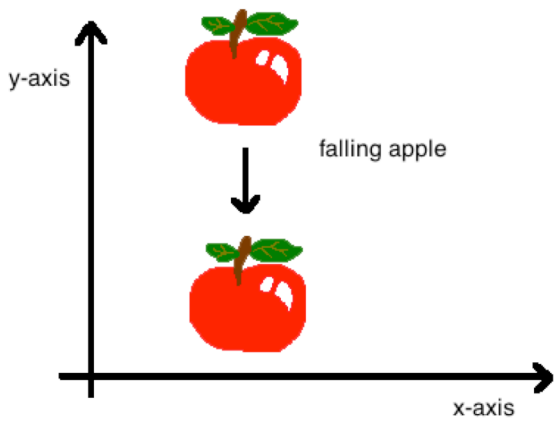
What You Have To Do

The focus of the activities in this game is to help you understand how 2D graphics (in order words, "images") and animation work in a software program. Two small activities are designed for this purpose.

Activity 1: Falling Apples

In this activity, we will concentrate on the mechanics of a single falling apple. First, open `Apple.java` file.

In simple animation programs, what we usually have is an image and the program controls when the image is added to the view and when it is removed from it. In Feed Me, that means creating an instance of the Apple object that has an apple image and making that *visible* on the screen. When the program needs to remove the Apple object, the program will first make the object *invisible* and then delete it altogether.



The program is also in charge of managing how fast an image moves on the screen. Since the screen is actually made up of pixels, that means our program has to specify how many pixels an object moves each time.

This is the part that we will focus on, and it is in the `move` method at the end of the file.

Two things need to happen in the `move` method:

1. First, the apple needs to fall. That means you will need to change its `y` position by the `FALL_SPEED` (of 2 pixels). These variables are already defined in the file, you just have to put them together into an equation.

2. Second, if the apple keeps falling, it will eventually fall outside of the boundaries of the screen. In that case, we will change the apple's visibility to make it invisible. (Another part of the program will then remove the apple altogether.)

   Here, the "screen size" of the program is determined by a variable in the `Board` class called `BOARD_HEIGHT`. From the `Apple.java` file that we are in, you need to refer to this variable as `Board.BOARD_HEIGHT`.

Complete the following worksheet by filling in the blanks to accomplish these two steps. When you are done, copy the code into the `move` method (with correct capitalization and punctuation).

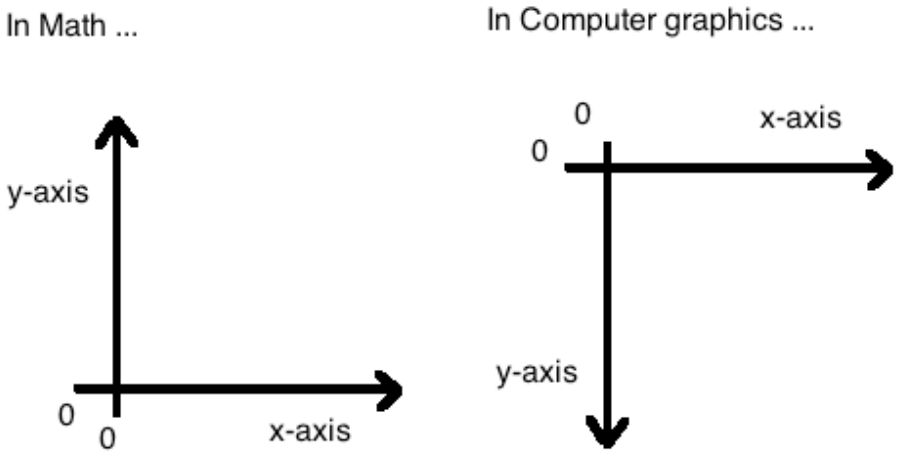
Coding Worksheet #1 for the Feed Me Game

```
public void move()
{
  // 1. change the y value so that it has an added FALL_SPEED

  y = _____________________________ ;

  // 2. when the apple moves beyond the board, it becomes invisible

  if( _____________________________ )

    visible = false;
}
```

Common Pitfalls To Avoid

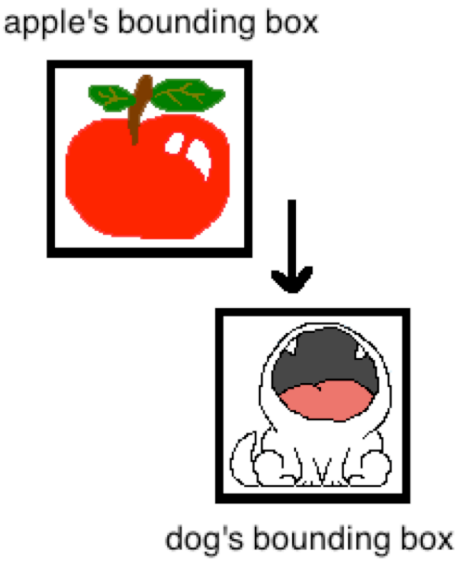
- Remember to spell out the variables exactly the way they are written elsewhere in the code

- In Math, we are used to having the geometry planes oriented with the origin at the bottom left corner and the y-axis increasing upwards. In computer graphics, the origin is oriented at the top left corner and the y-axis increases *down*wards. Therefore, when the apple falls, the `FALL_SPEED` must be added to `y` (not subtracted).



Activity 2: Eating Apples

In this activity, we will concentrate on the mechanics of two objects colliding with each other. First, open `Board.java` file and focus on the `checkCollision` method near the end of the file.



Every image in the program has a *bounding box* around it. The size of the bounding box is defined by the dimensions of the image. In an animation, when two objects collide, we would like the program to respond with some reactive behavior. In this case, when an apple collides with the dog, we want to show the effect of the dog eating the apple – thus, the apple disappears and the dog becomes happy. This logic is handled by the `checkCollision` method.

Q: How do we express when two objects collide?
A: When two bounding boxes intersect!
(Yay! Geometry at work!)

Inside the `checkCollision` method, the program is responsible of obtaining the monster dog's bounding box (call this `r1`) and an apple's bounding box (call this `r2`). If `r1` intersects with `r2`, then do the following:

- Make the apple disappear (because it got eaten)
- Play the "eat" sound
- Show that the monster dog is now happy
- Increase the score

Otherwise, nothing happens and the game continues. This logic is repeatedly applied to each apple that is available in the game.

Your job is to complete the second worksheet on the next page by filling in the blanks. A few points to note:

- In addition to numbers and letters, Java can store a basic type of information called *booleans*. For numbers, the values could be 0, 0.5, 9214, etc. For letters, the values could be a, b, c, etc. For booleans, the values must be either `true` or `false`. Booleans are useful when you know that something has only two outcomes (e.g. visible vs. not visible, happy vs. not happy).

- You don't have to do the math for calculating the intersection of two rectangles! Java has the mechanisms available for bounding boxes. In the program, you need to express as a condition whether `r1` intersects with `r2`.

Complete the following worksheet by filling in the blanks to accomplish these two steps. When you are done, copy the code into the `move` method (with correct capitalization and punctuation).

Coding Worksheet #2 for the Feed Me Game

```
private void checkCollision()
{
  // get bounding box of monster dog
  Rectangle r1 = monster.getBounds();

  // get bounding box of each individual apple
  ArrayList<Apple> food = dropper.getTargets();
  for( int i=0; i<food.size(); i++ )
  {
    Apple a = food.get( i );
    Rectangle r2 = a.getBounds();

    // check intersection – if so, monster dog eats apple

    if( _____________________________ )
    {
      // make the apple invisible

      a.setVisible( _______________ );

      // play the sound effect corresponding to the apple being eaten

      SoundEffect.EAT.play();

      // indicate that the monster dog is now happy because he ate

      monster.setHappy( _______________ );

      // increase the score by 1 point

      score = _____________________________ ;
    }
  }
}
```

Common Pitfalls To Avoid

- Each rectangle has a method called `intersects` that takes another rectangle and checks if the two intersect or not. If you have rectangles `X` and `Y`, you can accomplish this by writing: `X.intersects( Y )` or `Y.intersects( X )`. The result will be either `true` or `false`, depending on whether there is any intersection.