

Object Orientation - Creating an Ant Swarm

Part One - Modeling

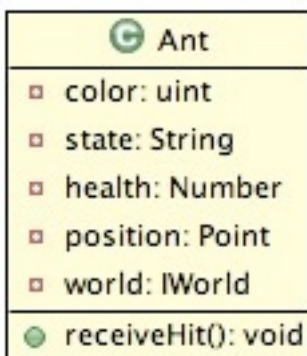
If you have never heard of object-orientation, you should check [sun's summary](#) about object-oriented concepts first. We create a simple but fun simulator simulating the behaviour of a simplified ant swarm.

In an ant swarm there are ants. Lots of ants :) But it's not that simple, there are different types of ants: on the top of the hierarchy, there is the queen. Under her, there are workers and drones. The workers get food from the outside world, and fight ants from other swarms, the drones take the food to the larvas.

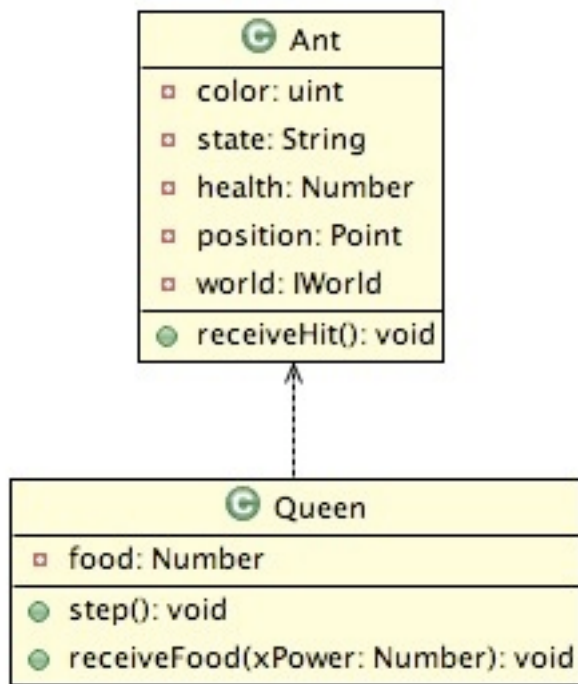
How should we model this? Basicly, there are swarms, ants, and food. Swarms are represented by the queens, so there is no need for a swarm instance. Only ants and food left. What kind of ants should we model? We need a queen, workers, and for a spectacular simulation, let's say that the drones are the fighters, only they can fight other ants, but they can't get food, and only the workers can get food, but they can't fight other ants.

Now we can start object-orienting. Our most important class will be the Ant class. What fields and methods should it contain? What are the common attributes of an ant? Well, an ant surely has a position and health. We want to differ them by color, so we need a color field. We will program them as state-machines, so a state attribute is also needed. And finally, our ants have to interact with the outside world, so a world field is needed.

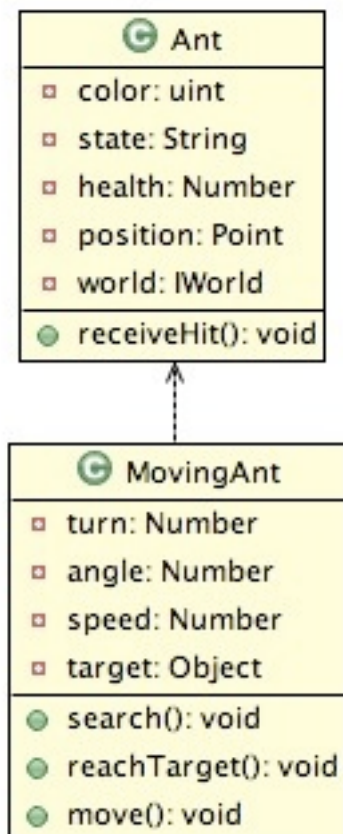
And what methods should we implement? Well, the only method what is common in the three types of ants will be "receiveHit", because all ants can be injured by other ants. And a step function also needed for the state machine. Here is the UML diagram of our Ant class:



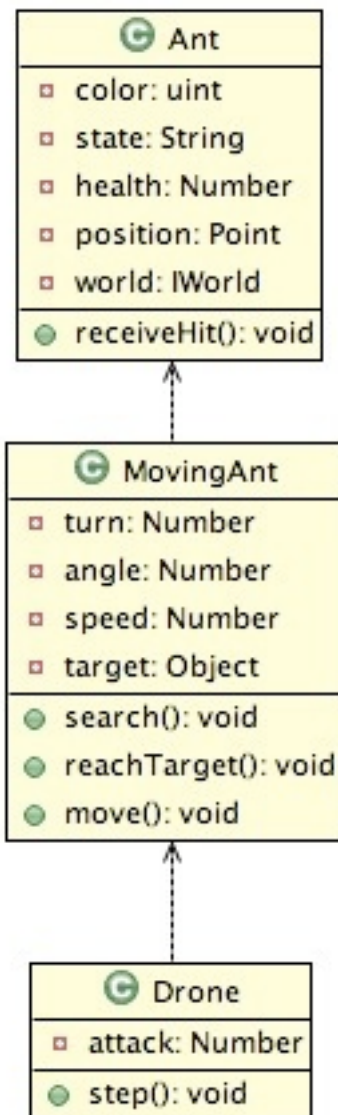
But we need three different types of ants. Queen can be inherited from this class directly, only an additional food field is necessary for her, to store available food mass. We have to override Ant's step function, the different types do different things. And a receiveFood method is also needed.



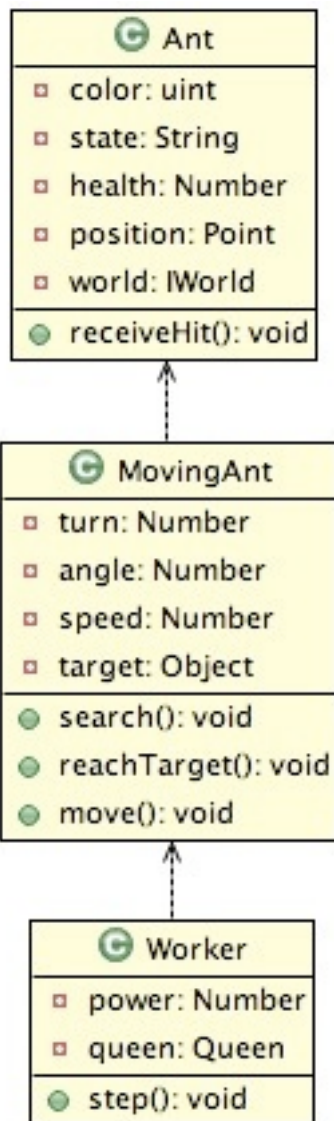
Drone and Worker comes, but we should think a little first. They are different, but they have a few common things what Queen doesn't have : they can move, they can search for food/enemy, they can reach a target. So, mid-level subclass would be a good idea, from which we can inherit these classes. Let's call this class *MovingAnt*. A *MovingAnt* can search, can reachTarget, and can move. Fields needed for this operations: turn, angle, speed, target.



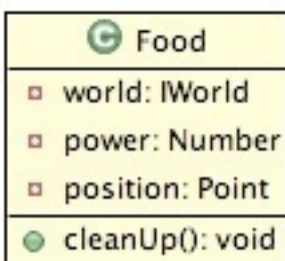
Now we can inherit Drone from it. Drone will be the fighter ant, it has an attack power, and an overridden step function.



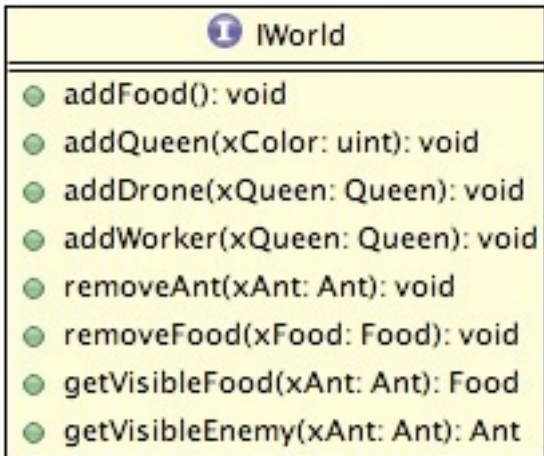
Worker is also simple: it has a power field, which represents the food amount transported by ant, and a queen field, because we need to know which is and where is our queen. And an overridden step function.



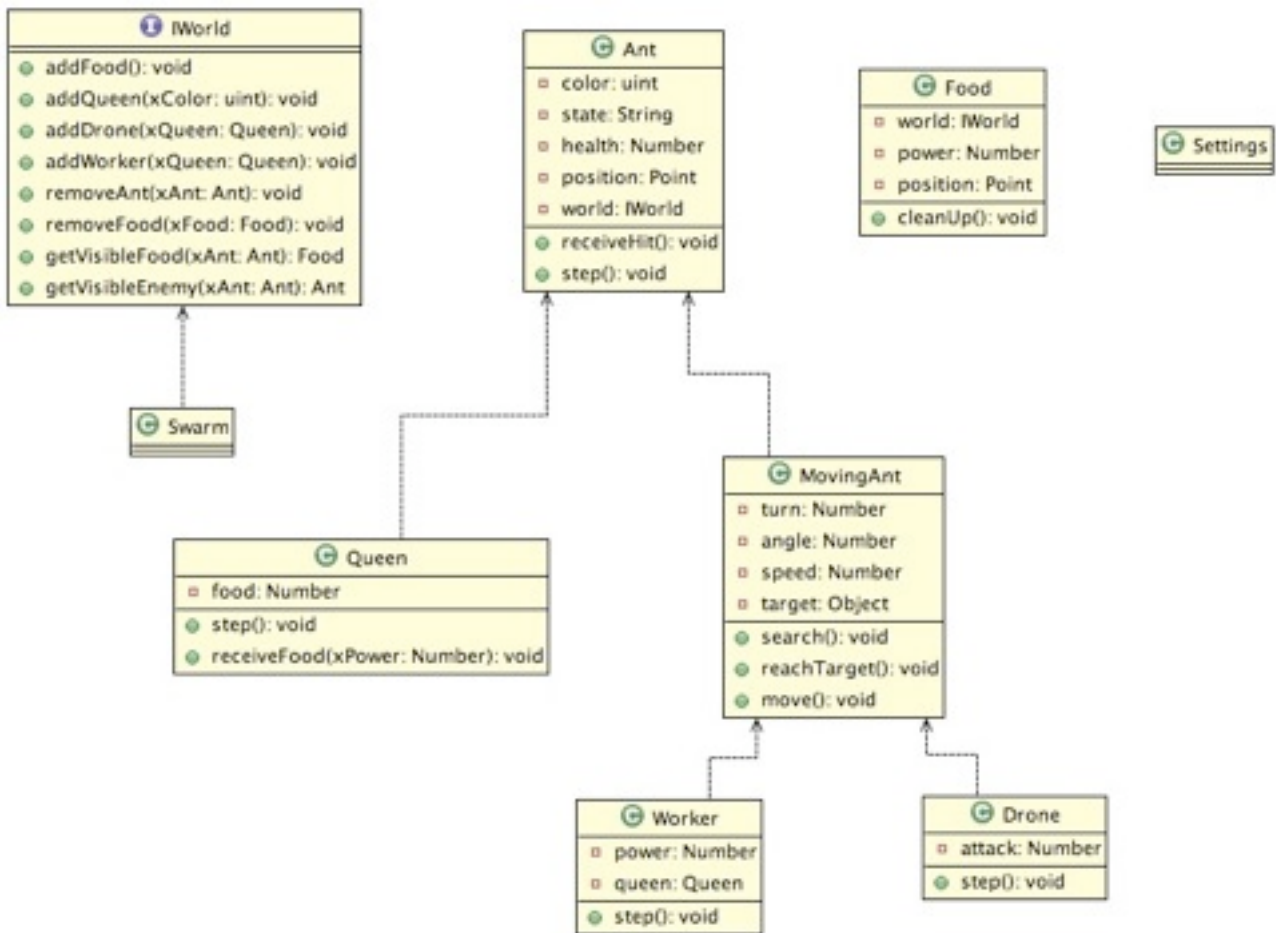
Food will be the next, it is a very simple class, it has a position, power, and the outside world to interact with. And one method: `cleanUp`, if someone picked it up.



And only on main class left. The outside world. As we see two different classes will "use" it, or more, if we extend our code a little, and it can change rapidly during development, so we better define it as an interface. What methods should it contain? We will surely need an `addQueen`, `addDrone`, `addWorker` and `addFood` method to construct our scene. That's why a `removeAnt` and `removeFood` method will also be needed. And to help our ants exploring the world, a `getVisibleFood` and `getvisibleEnemy` methods are necessary.



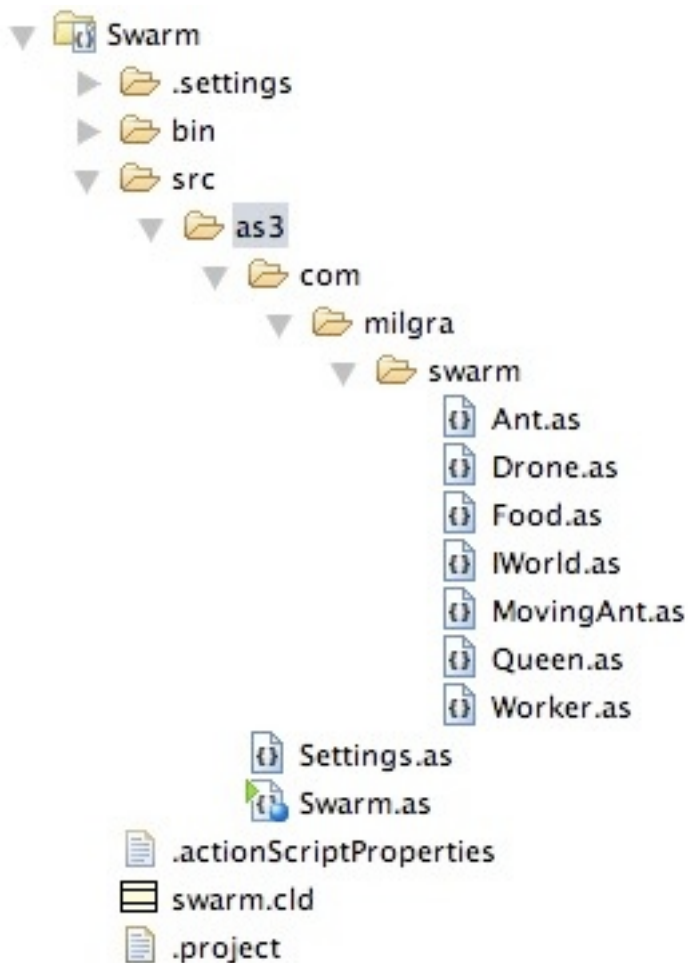
And our model is ready. For our program two additional classes will be needed: the main class, which controls and starts the game, and also acts as World, so it must implement the IWorld interface, and a class with static properties to store game constants. Final UML diagram will look like this:



Part Two - Realization

Now we can start typing. Create a new actionscript project in Eclipse/Flex called swarm. Create a directory structure src/as3/com/milgra/swarm, and set src/as3 as the main source folder. (Project menuitem - Properties - Actionscript Build Path - Main

source folder: src/as3. In folder src/as3/com/milgra/swarm create the model-related classes Ant, MovingAnt, Drone, Worker, Queen, Food, IWorld. Form the root, move Swarm.as to src/as3, and create a Settings class here also. Navigator view should look like this:



Class Ant

```
package com.milgra.swarm
{

    import flash.geom.Point;
    import flash.display.Sprite;

    // Sprite extension is needed because we want avatars representing ants

    public class Ant extends Sprite
    {

        // all variable should be public in this class, because subclasses
        // use them

        public var world : IWorld;           // world
        public var state : String;           // actual state
        public var color : uint;
        public var health : Number;
        public var position : Point;

        public function Ant ( xWorld : IWorld )
        {

            // storing world
```

```

        world = xWorld;
        // initializing position
        position = new Point( );
    }
    // it's an empty function, subclasses will override it
    public function step ( ):void
    {

    }
    // receiving hit from an enemy
    public function receiveHit ( hitStrength : Number ):void
    {
        // subtract hit strength from health
        health -= hitStrength;
        // if we are dead, removing ourself
        if ( health < 0 ) world.removeAnt( this );
        // do a little zig-zag when hit
        rotation = Math.random( ) * 360;
    }
}
}
}

```

Class Queen

```

package com.milgra.swarm
{
    import Settings;
    import flash.geom.Point;

    public class Queen extends Ant
    {
        private var food : Number;           // food amount

        public function Queen ( xPos:Point , xColor:uint , xWorld : IWorld )
        {
            // call superclass constructor
            super( xWorld );
            // set attributes based on Settings class
            food = Settings.QUEEN_FOOD;
            color = xColor;
            health = Settings.QUEEN_HEALTH;

            // set position
            x = position.x = xPos.x;
            y = position.y = xPos.y;
        }
    }
}

```

```

        // draw avatar
        graphics.beginFill( xColor , 1 );
        graphics.drawCircle( 0 , 0 , Settings.QUEEN_SIZE );
    }

    // override Ant's step function
    public override function step ( ):void
    {
        // if we have enough amount of food
        if ( food > Settings.ENTITY_FOOD )
        {
            // adding new drone and worker
            world.addDrone( this );
            world.addWorker( this );

            // subtracting food
            food -= Settings.ENTITY_FOOD;
        }
    }

    // receive food from worker
    public function receiveFood ( xPower:Number ):void
    {
        food += xPower;
    }
}
}

```

Class MovingAnt

```

package com.milgra.swarm
{
    public class MovingAnt extends Ant
    {
        public var turn : Number;           // maximal turning in radian
        public var angle : Number;         // actual angle
        public var speed : Number;         // speed
        public var target : Object;        // target to reach

        public function MovingAnt ( xWorld : IWorld )
        {
            // call superclass constructor
            super( xWorld );

            // set starting angle
            angle = 0;

            // set maximal turn

```

```

    turn = Settings.MAX_TURN;
}

protected function search ( ):void
{
    // maka move in with random turning angle
    angle = angle - turn / 2 + Math.random( ) * turn;
    // do movement
    move( );
}

// protected function : subclasses see them, external classes don't
protected function reachTarget ( ):void
{
    var dx : Number = ( target.position.x - position.x );
    var dy : Number = ( target.position.y - position.y );

    // calculate direction to target
    angle = Math.atan2( dy , dx );
    // do movement
    move( );
}

// private function : can be accessed only from this class
private function move ( ):void
{
    // calcualte new position
    position.x += Math.cos( angle ) * speed;
    position.y += Math.sin( angle ) * speed;

    var dx:Number = position.x - x;
    var dy:Number = position.y - y;

    // calculate rotation
    rotation = Math.atan2( dy , dx ) * 180 / Math.PI;

    // set own position
    x = position.x;
    y = position.y;

    // check borders
    if ( position.x < 0 ) position.x = stage.stageWidth + position.x;
    else if ( position.x > stage.stageWidth ) position.x = 0;

    if ( position.y < 0 ) position.y = stage.stageHeight + position.y;
    else if ( position.y > stage.stageHeight ) position.y = 0;
}
}
}

```

Class Drone

```
package com.milgra.swarm
{
    import Settings;
    import flash.geom.Point;

    public class Drone extends MovingAnt
    {
        public var attack : Number;           // attack strength

        public function Drone ( xQueen:Queen , xWorld : IWorld )
        {
            // call superclass constructor
            super( xWorld );

            // set starting position to queens position
            position.x = xQueen.position.x;
            position.y = xQueen.position.y;

            // starting state is searching state
            state = "search";

            // our color is our queens color
            color = xQueen.color;

            // get constants
            speed = Settings.DRONE_SPEED;
            attack = Settings.DRONE_ATTACK;
            health = Settings.DRONE_HEALTH;

            // draw avatar
            graphics.beginFill( color );
            graphics.lineStyle( 1 , 0xff0000 , 1 );
            graphics.drawEllipse( - Settings.DRONE_WIDTH / 2 ,
                                  - Settings.DRONE_HEIGHT / 2 ,
                                  Settings.DRONE_WIDTH ,
                                  Settings.DRONE_HEIGHT );
        }

        // override Ant's step
        public override function step ( ):void
        {
            // check state
            switch ( state )
            {
                case "search" :
                    // do a search movement ( inherited from MovingAnt )
                    search( );

                    // check nearby enemies
                    var enemy:Ant = world.getVisibleEnemy( this );
            }
        }
    }
}
```



```

public function Worker ( xQueen:Queen , xWorld : IWorld )
{
    // call superclass constructor
    super( xWorld );

    // our position is our queens position
    position.x = xQueen.position.x;
    position.y = xQueen.position.y;

    // starting state is searching state
    state = "search";

    // set queen and color
    queen = xQueen;
    color = xQueen.color;

    // set constants
    speed = Settings.WORKER_SPEED;
    health = Settings.WORKER_HEALTH;

    // draw avatar
    graphics.beginFill( color );
    graphics.drawEllipse( - Settings.WORKER_WIDTH / 2 ,
                          - Settings.WORKER_HEIGHT / 2 ,
                          Settings.WORKER_WIDTH ,
                          Settings.WORKER_HEIGHT );
}

// override ant's step
public override function step ( ):void
{
    // check state
    switch ( state )
    {
        case "search" :
            // do a search movement ( inherited from MovingAnt )
            search( );

            // check nearby food
            var food:Food = world.getVisibleFood( this );

            // if got food
            if ( food != null )
            {
                // set food as target
                target = food;

                // set new state
                state = "reach";
            }
    }
}

```

```

        break;
    case "reach" :
        // if someone was faster
        if ( target.power == 0 )
        {
            // set searching state
            state = "search";
            return;
        }
        // do reach movement ( inherited from MovingAnt )
        reachTarget( );
        // check distance
        var distance:Number = Point.distance( position , target.position );
        // if we are close enough, get food
        if ( distance < Settings.CONTACT_DISTANCE )
        {
            // get food power
            power = target.power;
            // cleanup food
            target.cleanUp( );
            // set new state
            state = "transfer";
            // queen is our new target
            target = queen;
        }
        break;
    case "transfer" :
        // do reach movement ( inherited from MovingAnt )
        reachTarget( );
        // check distance
        var distance:Number = Point.distance( position , target.position );
        // if we are close enough
        if ( distance < Settings.CONTACT_DISTANCE )
        {
            // give food to queen
            queen.receiveFood( power );
            // reset food power
            power = 0;
        }
    }
}

```



```

import Settings;
import com.milgra.swarm.Ant;
import com.milgra.swarm.Food;
import com.milgra.swarm.Queen;
import com.milgra.swarm.Drone;
import com.milgra.swarm.Worker;
import com.milgra.swarm.IWorld;
import flash.events.MouseEvent;

// we implement IWorld interface, because no individual world class needed yet,
// all functions can be created in the main class

public class Swarm extends Sprite implements IWorld
{
    public var fps:Number;           // actual fps
    public var stamp:Number;        // timestamp for fps measurement

    public var antList:Array;       // active ants
    public var foodList:Array;      // active food

    public var reset:Boolean;       // have to wait one frameevent before reset
because of thread asynchronity
    public var scene:Sprite;        // main display
    public var fpsField:TextField;  // textfield showing fps
    public var countField:TextField;// textfield showing active objects

    public function Swarm ( )
    {
        // set framerate

        stage.frameRate = 30;

        // wait until stage.stageWidth

        addEventListener( Event.ENTER_FRAME , init );
    }

    public function init ( event:Event ):void
    {
        removeEventListener( Event.ENTER_FRAME , init );

        // instance creation

        scene = new Sprite( );
        fpsField = new TextField( );
        countField = new TextField( );

        antList = [ ];
        foodList = [ ];

        stamp = ( new Date( ) ).time;

        // adding displayobjects

        addChild( scene );
        addChild( fpsField );
        addChild( countField );

        // set up textfields

        fpsField.textColor = 0xffffffff;
        countField.textColor = 0xffffffff;
        countField.y = 20;

        // create starting configuration

        for ( var a:int = 0 ; a < Settings.STARTING_FOODS ; a++ ) addFood( );

```

```

        for ( var b:int = 0 ; b < Settings.STARTING_QUEENS ; b++ ) addQueen
( Math.round( Math.random( ) * 0xffffffff ) );

        // step world on enterframe events
        addEventListener( Event.ENTER_FRAME , stepWorld );
        // reset and generate new instances on mouse click
        stage.addEventListener( MouseEvent.MOUSE_DOWN , resetWorld );
    }

    // create new configuration
    public function resetWorld ( event:MouseEvent ):void
    {

        // removing displayobjects
        for ( var a:* in antList )
            scene.removeChild( antList[a] );

        for ( var b:* in foodList )
            scene.removeChild( foodList[b] );

        // reset arrays
        antList = [ ];
        foodList = [ ];

        // generating random instance numbers
        var food:Number = Math.round( Math.random() * Settings.STARTING_FOODS );
        var queens:Number = Math.round( Math.random() * Settings.STARTING_QUEENS );

        // creating configuration
        for ( var c:int = 0 ; c < food ; c++ ) addFood( );
        for ( var d:int = 0 ; d < queens ; d++ ) addQueen( Math.round( Math.random
( ) * 0xffffffff ) );
    }

    public function stepWorld ( event:Event ):void
    {

        // call all ants' step function
        for ( var a:* in antList ) antList[a].step( );

        // generating new food
        if ( Math.random() < Settings.NEW_FOOD_POSSIBILITY ) addFood( );

        // calculating fps
        var now:Number = ( new Date( ) ).time;
        var delay:Number = ( now - stamp ) / 1000;
        fps = 1 / delay;
        stamp = now;

        // updating textfields
        fpsField.text = String( Math.round( fps ) ) + " fps";
        countField.text = String( antList.length + foodList.length + " objects");
    }

    public function addQueen ( xColor : uint ):void

```

```

{
    // creating new queen at a random position
    var queen : Queen = new Queen( randomPos( ) , xColor , this );
    // adding queen to display list
    scene.addChild( queen );
    // pushing queen in antList
    antList.push( queen );
}

public function addDrone ( xQueen:Queen ):void
{
    var drone : Drone = new Drone( xQueen , this );
    scene.addChild( drone );
    antList.push( drone );
}

public function addWorker ( xQueen:Queen ):void
{
    var worker : Worker = new Worker( xQueen , this );
    scene.addChild( worker );
    antList.push( worker );
}

public function removeAnt ( argAnt:Ant ):void
{
    // remove ant from displaylist and from antList
    for ( var a:* in antList )
        if ( antList[a] == argAnt )
        {
            scene.removeChild( antList[a] );
            antList.splice( a , 1 );
        }
}

public function addFood ( ):void
{
    // add new food
    var food:Food = new Food( randomPos( ) , this );
    scene.addChild( food );
    foodList.push( food );
}

public function removeFood ( argFood:Food ):void
{
    // remove food from displaylist and from foodList
    for ( var a:* in foodList )
        if ( foodList[a] == argFood )

```

```

        {
            scene.removeChild( foodList[a] );
            foodList.splice( a , 1 );
        }
    }

    public function getVisibleFood ( argAnt:Ant ):Food
    {
        // loop through every food
        for ( var a:* in foodList )
        {
            // calculate distance
            var distance : Number = Point.distance( foodList[a].position ,
argAnt.position );
            // if food is in our sight return it
            if ( distance < Settings.SIGHT_DISTANCE ) return foodList[a ];
        }
        return null;
    }

    public function getVisibleEnemy ( argAnt:Ant ):Ant
    {
        // loop through every ant
        for ( var a:* in antList )
        {
            // if ant's color is not the same, it is enemy
            if ( antList[a].color != argAnt.color )
            {
                // calcualte distance
                var distance : Number = Point.distance( antList[a].position ,
argAnt.position );
                // if enemy is in our sight
                if ( distance < Settings.SIGHT_DISTANCE ) return antList[a];
            }
        }
        // else return null
        return null;
    }

    // generate random position
    public function randomPos ( ):Point
    {

```

```
var pos : Point = new Point( );  
pos.x = Math.random( ) * stage.stageWidth;  
pos.y = Math.random( ) * stage.stageHeight;  
  
return pos;  
  
}  
  
}  
  
}  
  
(swarm.swf)  
(swarm.zip)
```

and thats all.

[Link to the published article on Actionsript.org](#)

2007.06.