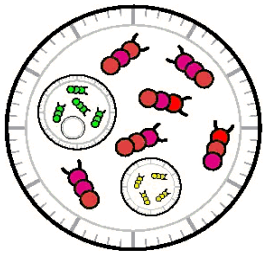
Swarm Tutorial
Java Session



http://www.swarm.org

Marco Lamieri
lamieri@econ.unito.it

Dep. of Statistics and Mathematics
Diego De Castro
University of Turin

Michele Sonnessa
sonnessa@di.unito.it

Dep. of Computer Sciences
University of Turin

Agenda

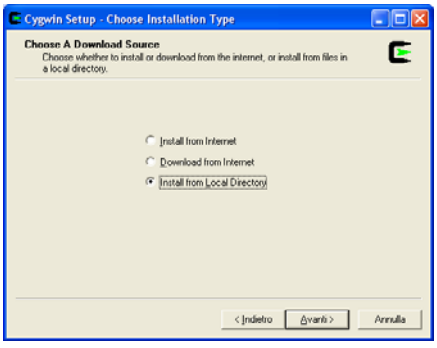
- Installation process
- Java language overiew
- ObjC vs Java
- Classes and Inheritance
- Create agents in Java
- Creating a Swarm
- Creating an Observer Swarm
- Creating a Scheduler
- Swarm libraries overview

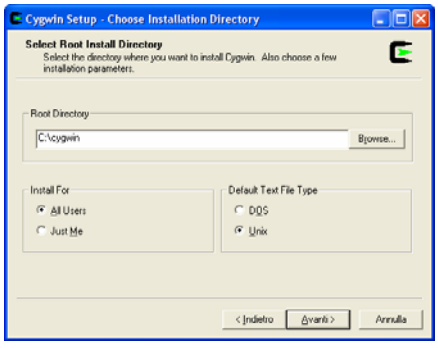
Setup under Windows

- Swarm in ObjC
  [Need Cygwin, a linux emulator]
  1. Install Cygwin/Swarm from CD
  2. Setup Cygwin configuration files
  3. Execute a demo application
- Swarm in Java
  [Do not need cygwin because use MingW compiler]
  1. Copy JavaSwarm files
  2. Set up windows environment variables
  3. Execute a demo application

Install Cygwin

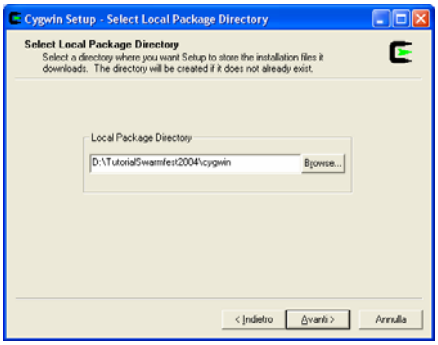
- From folder 'TutorialSwarmfest2005' in the CD execute ***setup.exe***

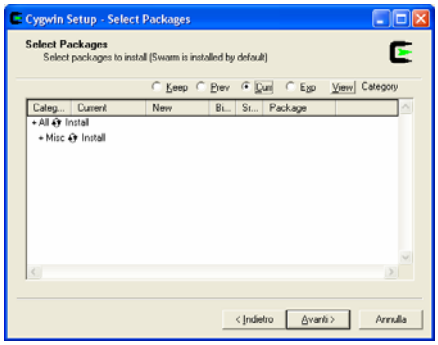




- Install from a **local directory**
- Root ***C:\cygwin***

Install Cygwin

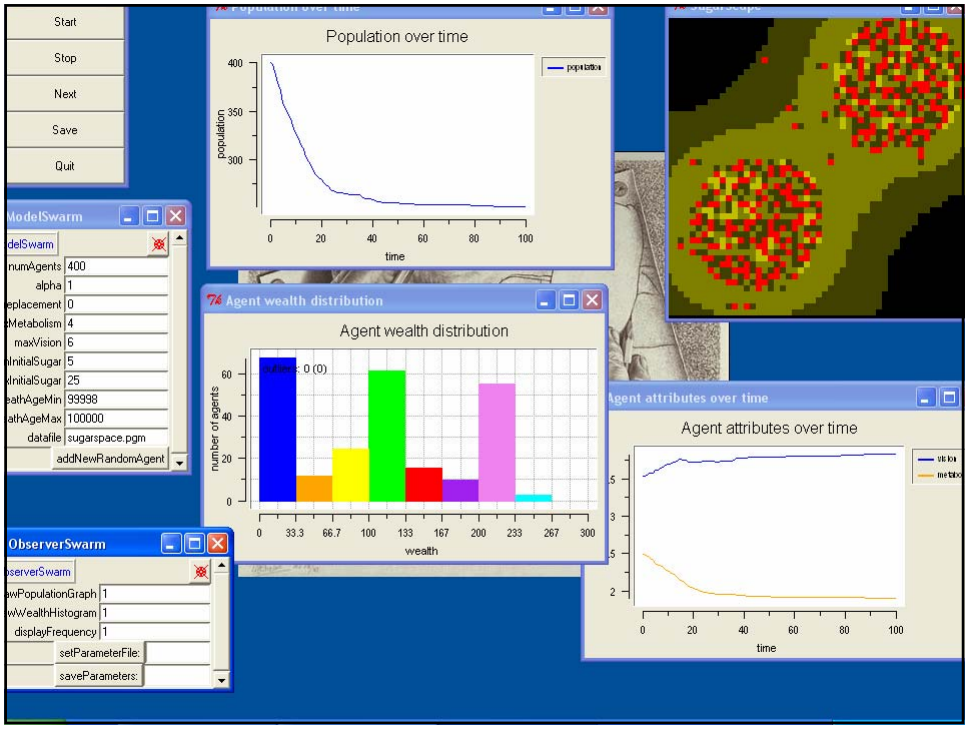




- Local package directory: ***D:\TutorialSwarmfest2004\cygwin***
- Install **all** packages

ObjC Swarm

- Copy from D:\TutorialSwarmfest2005 to C:\cygwin\home*username*
  .bashrc
- Create folder
  C:\SwarmApp
- Copy from CD in C:\SwarmApp
  swarmapps-objc-2.2-2.tar.gz
- Extract files to here
- Launch Cygwin bash and execute:
  - ***cd /cygdrive/c/SwarmApp/swarmapps-objc-2.2-2/sss***
  - ***make***
  - ***./sss.exe***

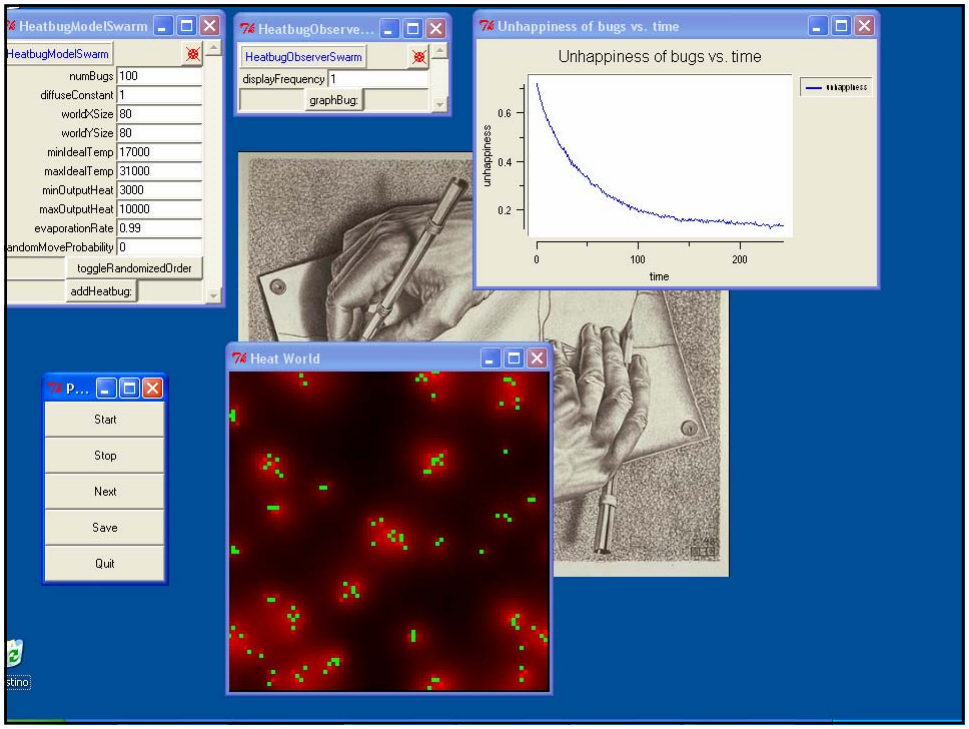


Install Java Swarm

1. Install from CD
   j2sdk-1_4_2_08-windows-i586-p.exe
2. Copy from CD into C:\
   Swarm-2.2-java.tar.gz
3. Extract files to here
4. Copy from CD into C:\
   StartJavaSwarm.bat
5. Copy from CD into C:\SwarmApp
   StartJavaSwarm.bat

Execute Java Swarm Apps

1. Copy from CD into C:\SwarmApp
   jheatbugs-2001-03-28.tar.gz and ***jmousetrap-2001-09-13.tar.gz***
2. Extract files to here
3. Double-click on
   C:\SwarmApp\StartJavaSwarm.bat
4. Execute:
   1. **cd jheatbugs-2001-03-28**
   2. **javac *.java**
   3. **java StartHeatbugs**



Java

- Pure objected-oriented created by Sun
- Independent of the underlying platform
- Large community
- Swarm Java layer mirrors the protocols of Swarm libraries as Java libraries
- Swarm community is more experienced with Objective C

Classes

- A class is a blueprint for making an object.
- Here's an example of a simple class:

```
public class Position {
   public int x; public int y;
   public Position (int x, int y) {
       this.x = x; this.y = y;
   }
   public int getX () { return x; }
   public int getY () { return y; }
}
```

  - Two ***member variables*** x , y
  - Two ***methods*** Position(), and getX()
  - The modifier ***public*** means that an item is visible outside of the class.

Inheritance

- In Java, inheritance lets you take one blueprint and extend it to more complicated blueprints.
- For example, here the Position class is extended to have three dimensions:

```
public class Position3D extends Position {
  public int z;
  public Position3D (int x, int y, int z) {
   super(x,y)
   this.z = z;
  }
   public int getZ() { return z; }
}
```

Message in Java

- The syntax for sending a message is:

```
g.setColor(Color.red);
```

- Java has *polymorphism* (you can have methods with the same name that accept different parameters). e.g. the following is possible:

```
void setColor(float red, float green, float blue) {
   Color g = new Color();
   g.setColor(red);
   g.setColor(green);
   g.setColor(blue);
}
```

Agents: Every agent is a Java object

- Inheriting from the Object class, Agents know:
  - create: Allocate memory
  - drop: Deallocate and die
- To create instance of class customer we call its constructor passing the required parameters:

```
Agent a =  new Agent();
```

- When Java objects are no longer used their memory is freed automatically by Java garbage collector

Where Java finds classes defintion

- While ObjC links the libraries within a unique executable file,
- Java compiles each class separately and load the required class into memory only when necessary.
- The classes are found by the JVM in the special path specification: the CLASSPATH

The import keyword

- A class is located in a particular folder into the file system.
- When you put an import call like

```
import java.util.ArrayList
```

- the JVM searches for the ArrayList binary code into the java/util folder
- Usually the both java folder and the current user code folders are part of the standard CLASSPATH definition.
- Other special libaries (like Swarm) must be defined by the user both at compilation and execution time

The Swarm $ notation

- While in Swarm-ObjC you have the following notation

```
[agent setX: 0 Y: 0]
```

- Swarm's authors used the corresponding Java version adopting the $ notation

```
agent.setX$Y(0,0);
```

where the $ separates the name of the expected parameters.

Creating a Swarm

I. create
- Initialize memory and parameters

II. buildObjects
- Build all the agents and objects in the model

III. buildActions
- Define order and timing of events

IV. activate
- Merge into top level swarm or start Swarm running

I: Initializing

```
import swarm.objectbase.Swarm;
import swarm.objectbase.SwarmImpl;

public class HeatbugModelSwarm extends SwarmImpl {

public SimpleModel(Zone aZone){

   super (aZone);

   numAgents = 100;

} }
```

II: Building Agents

```
public Object buildObjects (){

     super.buildObjects();

      world = new Grid2dImpl(getZone (), worldXSize,
  worldYSize);

      agent = new Agent();
  agent.setX$Y(0,0);
      world.putObject$atX$Y(agent, 0, 0);
  }
```

III: Building schedules

```
 public Object buildActions () {

     super.buildActions();

     Selector sel = new Selector (Agent.getClass (),
  “step", false);

     modelSchedule = new ScheduleImpl (getZone (), 1);
     modelSchedule.at$createActionTo$Message (0, agent,
  sel);
}
```

IV: Activating the Swarm

```
public Activity activateIn (Swarm swarmContext) {

    super.activateIn (swarmContext);

    modelSchedule.activateIn (this);

    return getActivity ();
}
```

Creating an ObserverSwarm

I. create

Initialize memory and parameters

II. buildObjects

Build ModelSwarm

Build graphs, rasters and probes

III. buildActions

Define order and timing of GUI events

IV. activate

start Swarm running

I: Initializing

```
import swarm.simtoolsgui.GUISwarm;
import swarm.simtoolsgui.GUISwarmImpl;

public class MyObserverSwarm extends GUISwarmImpl {

public ObserverSwarm (Zone aZone) {

    super(aZone);
    displayFrequency = 1;

} }
```

II: Creating objects

```
 public Object buildObjects () {

    super.buildObjects ();

    ModelSwarm mo = new ModelSwarm (getZone ());
    getControlPanel ().setStateStopped ();
    mo.buildObjects ();
```

III: Building schedules

```
public Object buildActions () {

    super.buildActions();

    mo.buildActions();

    displayActions = new ActionGroupImpl (getZone());


  displayActions.createActionTo$message(Globals.env.probeD
  isplayManager,
        new Selector
  (Globals.env.probeDisplayManager.getClass (),"update",
  true));
```

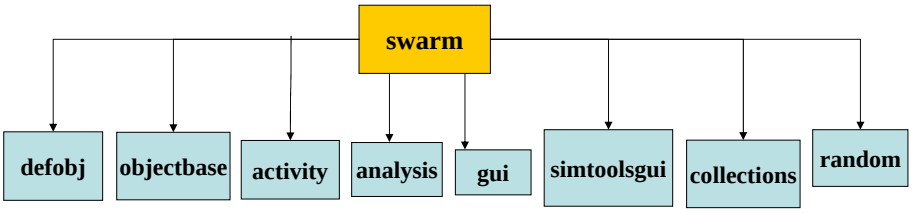
III: Building schedules

```
displayActions.createActionTo$message(getActionCache (),
  new Selector(
        getActionCache ().getClass (), "doTkEvents", true)
  );

displaySchedule = new ScheduleImpl (getZone (),
  displayFrequency);
   displaySchedule.at$createAction (0, displayActions);

    return this;
}
```

IV: Activating the Swarms

```
public Activity activateIn (Swarm swarmContext) {

    super.activateIn (swarmContext);

    mo.activateIn (this);

    displaySchedule.activateIn (this);

    return getActivity();
}
```

The libraries



Tanks to Hala Al-Bakour (University of Essex) for inspiring some slides of this tutorial