Swarm by example: `bell`

Ralf Stephan

May 10, 2000

Contents

1 About the first version

This document describes the development of the code of the `bell` Objective-C *swarm* demo program, a toy model that shows a bunch of balls hitting nails and being collected in slots, approximating a bell curve. *swarm*[1] is an Objective-C/Java framework for agent-based modelling (ABM), but not restricted to this use. You should be able to find a Transparent copy of this document at my web page[2] or the swarm.org pages.

This version verbosely introduces the original code in Sections 4 (`Board`) through 12 (`Makefile`) . The Debugging Section 13 deals with debugging of this version but shouldn't provoke general interest. With few exceptions, the code was written in the order you see it. Disclaimer: The order the stuff is presented here should be no dogma for the reader's development work. R.S., April 2000

[1] http://www.swarm.org
[2] http://me.in-berlin.de/~rws/bell-0005.tar.gz

2 First Version Thanks

The idea for the document flew to me from a Starlogo[3] project[4] with the same theme.

Tools used for pictures and text include but were not limited to noweb[5], LaTeX, xfig, The Gimp[6], LaTeX2HTML, and Amaya-2.2[7].

Kudos to all authors of the software I needed for this. R.S., April 2000

3 Introduction

3.1 Imagining the situation

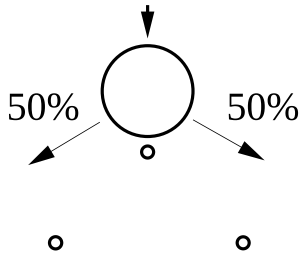

Figure 1: Schema

Picture a small ball on a planar slope, rolling down and hitting a nail of the same size, erectly sticking in the ground. Following that, there is a 50-50 chance that it will roll on down either of the two sides. Soon it will hit the next nail center, with the same chances of moving on to the left or right.

Presume that the nails are standing evenly-spaced in a triangular area, with the first nail the top corner of the triangle, and on the symmetry axis. So you can see rows of 1,2,3,4... nails, all symmetrical to the same axis.

Now, if you let enough balls pass this parcourt, and collect them, where they fall, in tubes, you would get a picture of the so-called gaussian[8] curve (referring to its discoverer) or bell curve (referring to its shape). The bell curve is ubiquitous in nature and/or is used extensively in statistics. Its mathematical formula is derived from the function in Fig. 2.

[3] http://el.www.media.mit.edu/groups/el/Projects/starlogo/
[4] http://el.www.media.mit.edu/groups/el/Projects/starlogo/projects/gaussian.html
[5] http://www.eecs.harvard.edu/~nr/noweb/
[6] http://www.gimp.org
[7] http://www.w3.org/Amaya/
[8] http://www-groups.dcs.st-and.ac.uk/~history/Mathematicians/Gauss.html

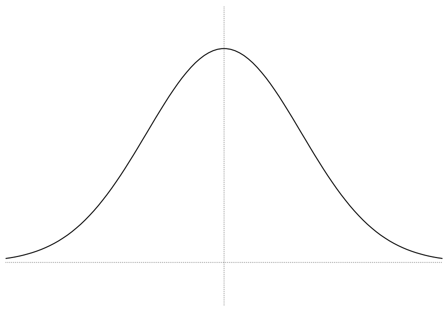

Figure 2: $f(x) = e^{-x^2}$

3.2 More detailed: the first model

Ideally (Fig. 3), the user starts the program, clicks on the Start button and sees balls pouring out of a container, through the nail area, and into the pockets, stopping when all balls are still. The start number/positions of balls/nails are stored in a 2D-string fashion for easy editing.

Graphically, the user will see the whole area as a cartesian plane with pixel chunks replaced by images for nicer look. It suffices to have balls and nails on the board, other agents aren't needed for this demo.

Figure 3: Screen shots

Another issue would be speed: the demonstration should be timed good enough to be realistically and visually pleasing.

3.3 Possible objects/classes

From the above, we have several candidates to be crystallized into classes: the `Board`, the `Balls`, the `Nails`, the `Nail`, the `Ball`:

> Usually, *swarm* code separates the GUI from the model for speed reasons with large models, but this doesn't apply to this demonstration — it could be added later on, if necessary. The swarmapps use an `ObserverSwarm` that can be replaced by a `BatchSwarm`, for example.
>
> So `Board` handles the graphics as well as the geometry here, and for that, it communicates with all other classes.
>
> `Nails` is the class that, from the `Board` viewpoint, contains information about the `Nail` agents.
>
> `Nail` is an agent that does nothing. This means, its methods are mostly empty.
>
> `Ball` is an agent that acts through simple rules ("gravity", "balance", "being stuck" etc.) in the environment.
>
> It was soon obvious that there is a container needed to multiplex requests to the balls, too. Stunningly, our best name for it is `Balls`. We might see other classes popping into existence as necessary.

The latter four classes communicate with `Board`, and to some extent with each other.

4 The Board class and its objects

The first files to be written are `Board.h` and `Board.m`, so the code chunks that follow go into these. In the last subsection, we'll join all the chunks together.

4.1 A class skeleton

The control panel that we need is provided by the *swarm* class `GUISwarm` which itself is a subclass of `Swarm` that represents a general container for agents, giving them common time and memory, so we inherit `Board` in the

5a ⟨*Board declaration head* 5a⟩≡

```
@interface Board: GUISwarm
```

within `Board.h`. To use `GUISwarm` like this, we also need to

5b ⟨*Import the GUISwarm interface* 5b⟩≡

```
#import <simtoolsgui/GUISwarm.h>
```

The following code chunks describe the file `Board.m`. At the top, the class header file is included and the implementation started.

5c ⟨*Import header, start implementation* 5c⟩≡

```
#import "Board.h"
@implementation Board
```

To create the `Board` object, a *class method* is needed. Class methods can be invoked by sending a message to the class itself, not to one of the objects instantiated from it, and one of their usages are factory methods. The method is implemented within `Board.m` and it simply invokes the super class' `create:` method. Class methods are prefixed with a plus character.

6a ⟨*Board::create definition* 6a⟩≡
```
+ create: aZone
{
        return [super create: aZone];
}
```

4.2 The buildObjects method

4.2.1 Control panel

Creation of the objects contained by `Board` happens all in two methods named `buildObjects` and `buildActions`, respectively. First, a message is sent to the `GUISwarm` parent class of `Board` to give it the opportunity to build its objects, for example the `controlPanel` which is a globally accessible object. Then we wait for the user to click a button on the control panel, and quit if it's the Quit button.

6b ⟨*Handle control panel* 6b⟩≡
```
[super buildObjects];
[controlPanel setStateStopped];
if ([controlPanel getState] == ControlStateQuit)
        return nil;
```

4.2.2 Balls and nails

At this time, the user has clicked. `Balls` and `Nails` must be instantiated. In a more sophisticated *swarm* simulation, one would create an object (for example, a `ModelSwarm`) that separates the display from the simulated world — here, there is no such object, so imports of the respective header files are necessary.

6c ⟨*Import Balls, Nails headers* 6c⟩≡
```
#import "Balls.h"
#import "Nails.h"
```

Declare a handle at the top of the method for nails:

6d ⟨*nails declaration* 6d⟩≡
```
id nails;
```

And declare a handle as a private member of the `Board` class in `Board.h` because it is needed later when scheduling is done.

6e ⟨*balls declaration* 6e⟩≡
```
id balls;
```

Now set the handles.

7a ⟨*Create Balls, Nails objects* 7a⟩≡
```
balls = [Balls create: [self getZone]];
nails = [Nails create: [self getZone]];
```

4.2.3 The Raster frame

Finally, going on with creation of GUI objects is possible. At this point, usually a color map is built, but, as all objects on the `Board` are shown as pixmaps, this step can be skipped here (in Section 13.0.1, we'll see that this is wrong so expect the code to change a bit later).

So, the next thing to build is a `Raster` object and to draw it. It seems safe to say that the `Raster` represents mainly the simulation window frame and its colormap. The handle declaration goes into the header file.

7b ⟨*worldRaster declaration* 7b⟩≡
```
id <ZoomRaster> worldRaster;
```

The string representation (see Section 9) has the authority for the board dimensions, only the `worldRaster` zoom factor is adapted experimentally in Section 13.0.2. The rest of the code should be self-explanatory.

7c ⟨*Create, set, draw worldRaster* 7c⟩≡
```
worldRaster = [ZoomRaster createBegin: self];
SET_WINDOW_GEOMETRY_RECORD_NAME (worldRaster);
worldRaster = [worldRaster createEnd];
[worldRaster setZoomFactor: 4];
[worldRaster setWidth: [StringRepresentation getSizeX]
             Height: [StringRepresentation getSizeY]];
[worldRaster setWindowTitle: "Balls and Nails"];
[worldRaster pack];                              // draw the window.
```

Here, the first usage of `createBegin:` instead of `create:` can be seen. Also, `self` is given as the Zone. That is possible because `Board` inherits from `Zone` through `GUISwarm` through `Swarm`.

The message pair `createBegin:/createEnd` is used for creating objects whenever a distinction is necessary between messages to objects 'still not scheduled' and those to objects 'in schedule'. Messages in between such pairs help *swarm* with doing simulation efficiently, and might provide the user with improved memory usage stats in the future.

4.2.4 The world grid

As soon as the frame is set, its content comes into focus.

In other *swarm* applications, the grid containing the agents, as well as all other non-graphical objects like values, is inside its own `Swarm` to separate computation and graphics. But, as said, we put it all into `Board` for convenience.

We choose a `Grid2d` that can contain arbitrary int values or pointers. Nails and balls are pointers to the respective agents, and empty space is 0.

The declaration is written at the `buildObject` method's top.

8a ⟨*Local world declaration* 8a⟩≡
```
id <Grid2d> world;
```

In the method's body, the grid is then created and sent to `Balls` where it's needed by the agents.

8b ⟨*Create world grid* 8b⟩≡
```
world = [Grid2d create: self
         setSizeX: [StringRepresentation getSizeX]
         Y: [StringRepresentation getSizeY]];

[balls buildObjects];
[nails buildObjects];
[balls setWorld: world];
```

The `StringRepresentation` has authority of the worlds size and content. Its interface should be imported at the `Board` class implementations top, too.

8c ⟨*Import StringRepresentation interface* 8c⟩≡
```
#import "StringRepresentation.h"
```

4.2.5 Displaying the world

Now that we have the window frame and the grid, we can create the object that displays the agents and put it in a handle.

As the handle is used later when events are scheduled, its declaration as a private member is put into the class header. According to the choice of the grid, this will be an `Object2dDisplay`.

8d ⟨*boardDisplay declaration* 8d⟩≡
```
id <Object2dDisplay> boardDisplay;
```

The interfaces of the display and grid classes reside in `<space.h>` so this header is best imported from the `Board` header, too.

9a ⟨*Import space interfaces* 9a⟩≡

```
#import <space.h>
```

Let's create the display.

9b ⟨*Create world display* 9b⟩≡

```
boardDisplay = [Object2dDisplay create: self
                 setDisplayWidget: worldRaster
                 setDiscrete2dToDisplay: world
                 setDisplayMessage: M(drawSelfOn:)];
```

Usually, the display will look through its grid and if there is an object, it will be sent the `drawSelfOn:` message to display itself. Another strategy would have been to give the grid a collection of to-be-displayed objects.

4.2.6 Summary of buildObjects

Finally, let's have a look at the overall structure of the `buildObjects` method:

9c ⟨*Board::buildObjects definition* 9c⟩≡

```
- buildObjects
{
  ⟨nails declaration 6d⟩
  ⟨Local world declaration 8a⟩

  ⟨Handle control panel 6b⟩
  ⟨Create Balls, Nails objects 7a⟩
  ⟨Create, set, draw worldRaster 7c⟩
  ⟨Create world grid 8b⟩
  ⟨Create world display 9b⟩

  return self;
}
```

The method has to return an `id` because it has to conform to this interface which is inherited from the `Swarm` superclass. It doesn't seem a bad idea to generally `return self` so that messages could be chain grouped.

4.3 The buildActions method

In `buildActions`, the main event loop of the simulation is scheduled and, for that, an `ActionGroup` object is needed.

One can say that this is like a list of tasks to be worked on, with their own schedule, e.g. every timestep of the simulation. It is possible to randomize tasks in an `ActionGroup` by allowing them random order within one step, but this is not useful with the main schedule. But `ActionGroups` can be nested, and later some agent schedules can be randomized if necessary.

The other object to be created is the `displaySchedule` itself; it needs to know the frequency it's running on, and the `ActionGroup` to schedule.

The handles of both objects are declared in the `Board.h` header file:

10a ⟨*Action, schedule declarations* 10a⟩≡
```
id displayActions;
id displaySchedule;
```

The reader might note that it's not strictly necessary to give a protocol that the handle conforms to. Known messages to known objects might be handled differently by *swarm* than those to unknown ones but there doesn't seem much overhead. A side effect of this is that the necessary imports can be kept out of the header file and put on top of the implementation file:

10b ⟨*Import ActionGroup, Schedule interfaces* 10b⟩≡
```
#import <activity.h>
```

First, the method sends a message to the super class of `Board` to build the control panel and balls actions. Then it creates the first object and adds the members of the action list. The first three actions cause the content of the frame to be displayed efficiently — no actual erase happens but positional changes of the agents should be handled.

The last action consists of sending a message to the Tk library to render all graphical changes. Obviously, these actions have to happen in the right order on every step so they can't be randomized.

The setting up of the schedule object completes this method.

11a ⟨*Board::buildActions definition* 11a⟩≡

```
- buildActions
{
  [super buildActions];
  [balls buildActions];

  displayActions = [ActionGroup create: self];
  [displayActions createActionTo: worldRaster  message: M(erase)];
  [displayActions createActionTo: boardDisplay  message: M(display)];
  [displayActions createActionTo: worldRaster  message: M(drawSelf)];
  [displayActions createActionTo: actionCache  message: M(doTkEvents)];

  displaySchedule = [Schedule create: self setRepeatInterval: 1];
  [displaySchedule at: 0 createAction: displayActions];

  return self;
}
```

4.4 The activateIn: method

In this method, the already created and set `Schedule` is activated, as well as those of other objects that have one, including the super class. The swarm context comes from the `main` function. `Nails` don't need a schedule, they don't move.

11b ⟨*Board::activateIn definition* 11b⟩≡

```
- activateIn: swarmContext
{
  [super activateIn: swarmContext];
  [balls activateIn: self];
  [displaySchedule activateIn: self];

  return [self getSwarmActivity];
}
```

The returned `Activity` object isn't used in this simulation.

4.5 Summary of the Board class files

So, to put it all together, there is the implementation file:

12a ⟨*Board.m* 12a⟩≡

```
⟨Import ActionGroup, Schedule interfaces 10b⟩
⟨Import StringRepresentation interface 8c⟩
⟨Import header, start implementation 5c⟩

⟨Board::create definition 6a⟩
⟨Board::buildObjects definition 9c⟩
⟨Board::buildActions definition 11a⟩
⟨Board::activateIn definition 11b⟩
@end
```

and, from pieces all over the section, the interface file.

12b ⟨*Board.h* 12b⟩≡

```
⟨Import the GUISwarm interface 5b⟩
⟨Import space interfaces 9a⟩
⟨Import Balls, Nails headers 6c⟩
⟨Board declaration head 5a⟩
{
⟨balls declaration 6e⟩
⟨worldRaster declaration 7b⟩
⟨boardDisplay declaration 8d⟩
⟨Action, schedule declarations 10a⟩
}
+ create: aZone;
- buildObjects;
- buildActions;
- activateIn: swarmContext;
@end
```

This completes the description of the `Board` class.

5 The Balls class

This class contains a List of Balls, and activates their scheduling. It handles messages from Board and uses the StringRepresentation of the model start positions to set the Ball positions.

5.1 Balls.h

The header file Balls.h looks like this:

13a ⟨*Balls.h* 13a⟩≡

```
#import <objectbase/Swarm.h>
#import <collections.h>

@interface Balls: Swarm
{
  id <List> theBalls;
  id ballsActions, ballsSchedule;
}
+ create: aZone;
- buildObjects;
- setWorld: aWorld;
- buildActions;
- activateIn: swarmContext;
@end;
```

5.2 Balls.m

The Balls implementation draws resources from several other classes. The file has the following structure[9]:

13b ⟨*Balls.m* 13b⟩≡

```
#import <activity.h>
#import "Balls.h"
#import "Ball.h"
#import "Position.h"
#import "StringRepresentation.h"

@implementation Balls

⟨Balls::create definition 14a⟩
⟨Balls::buildObjects definition 14b⟩
⟨Balls::setWorld definition 14c⟩
⟨Balls::buildActions definition 15a⟩
⟨Balls::activateIn definition 15b⟩
@end
```

[9] Here, the whole file is presented first, to the contrary with the previous section.

Creating the object consists of creating its superclass, a minimal create method.

14a ⟨*Balls::create definition* 14a⟩≡
```
+ create: aZone
{
  return [super create: aZone];
}
```

The list is created and filled within the `buildObjects` method. The information is readily available from the `StringRepresentation` of the initial board.

As the `Position` objects (see Section10) are never dropped (to free their memory etc.) elsewhere, this has to be done here.

14b ⟨*Balls::buildObjects definition* 14b⟩≡
```
- buildObjects
{
  id pos;

  theBalls = [List create: [self getZone]];

  [StringRepresentation resetToChar: '0'];
  while ((pos = [StringRepresentation getNextPos]) != nil)
  {
    Ball* ball = [Ball create: [self getZone]];
    [ball setX: [pos getX] Y: [pos getY]];
    [theBalls addFirst: ball];
    [pos drop];
  }
  return self;
}
```

Readers might note that `StringRepresentation` behaves like an iterator, returning all `Position` objects that can be linked to the `0` character. It's also a class that consists entirely of class methods. On another eMail tip by Paul E Johnson, the `List::addFirst` method is used instead of `addLast`, since the order of balls in the list doesn't matter.

The `setWorld` message is received from the board when `world` creation is finished. Every single ball needs a handle to it because they are assumed to access `world` independently. So we send it with the `setWorld` message to all of them.

14c ⟨*Balls::setWorld definition* 14c⟩≡
```
- setWorld: aWorld
{
  [theBalls forEach: M(setWorld:) : aWorld];
  return self;
}
```

Like in the Board implementation, there are Actions and a Schedule to be built[10].

15a ⟨*Balls::buildActions definition* 15a⟩≡

```
- buildActions
{
  ballsActions = [ActionGroup create: self];
  [ballsActions createActionForEach: theBalls message: M(step)];

  ballsSchedule = [Schedule createBegin: self];
  [ballsSchedule setRepeatInterval: 1];
  ballsSchedule = [ballsSchedule createEnd];
  [ballsSchedule at: 0 createAction: ballsActions];

  return self;
}
```

In the activateIn method, the schedule of all balls is added to the caller's activity.

15b ⟨*Balls::activateIn definition* 15b⟩≡

```
- activateIn: swarmContext
{
  [ballsSchedule activateIn: swarmContext];
  return self;
}
```

[10] The reader deeper into automata such as this should note that the balls aren't scheduled randomly. For the step method being realistic with a randomized schedule, we would have to add security that balls don't interfere in the sieve part, which was too much for this little demo.

6 The Nails class

This class contains a `List` of `Nails`, and sets their positions. It handles messages from `Board` and uses the `StringRepresentation` of the model start positions to set the `Nail` positions.

The difference to `Balls` is that nails are never put into the simulation schedule since they don't move, so there are no `buildActions` and `activateIn:` methods needed. For the same reason, nails need not access the world — except for drawing the first time but then they get it as parameter with the `drawSelfOn:` message —, and so `setWorld:` isn't needed, as well.

The header file `Nails.h` looks like this:

16 ⟨*Nails.h* 16⟩≡

```
#import <objectbase/SwarmObject.h>
#import <collections.h>

@interface Nails: SwarmObject
{
  id <List> theNails;
}
+ create: aZone;
- buildObjects;
@end;
```

And the implementation file can be adapted from `Balls.m` with the above-mentioned changes:

17 ⟨*Nails.m* 17⟩≡

```
#import "Nails.h"
#import "Nail.h"
#import "Position.h"
#import "StringRepresentation.h"

@implementation Nails

+ create: aZone
{
  return [super create: aZone];
}

- buildObjects
{
  id pos;

  theNails = [List create: [self getZone]];

  [StringRepresentation resetToChar: '1'];
  while ((pos = [StringRepresentation getNextPos]) != nil)
  {
    id nail = [Nail create: [self getZone]];
    [nail setX: [pos getX] Y: [pos getY]];
    [theNails addLast: nail];
    [pos drop];
  }
  return self;
}
@end
```

7 Ball agents

Balls know their position and the world grid where it all happens. Also, balls are scheduled to get the `step` message each turn. So, the header looks like this:

18a ⟨*Ball.h* 18a⟩≡

```
#import <objectbase/SwarmObject.h>
#import <space.h>       // for Raster protocol

@interface Ball: SwarmObject
{
  unsigned x,y;
  id <Grid2d> world;
}
+ create: aZone;
- (void) setX: (unsigned) X Y: (unsigned) Y;
- setWorld: aWorld;
- (void) step;
- drawSelfOn: (id <Raster>) aRaster;
@end;
```

The implementation is straightforward but there is more to say about the `step` and `drawSelfOn:` methods.

18b ⟨*Ball.m* 18b⟩≡

```
#import <random.h>
#import "Ball.h"

@implementation Ball

+ create: aZone
{
  return [super create: aZone];
}

- (void) setX: (unsigned) X Y: (unsigned) Y
{
  x = X;
  y = Y;
}

- setWorld: aWorld
{
  world = aWorld;
  return self;
}

⟨Ball::step definition 19⟩
⟨Ball::drawSelfOn definition 20a⟩

@end
```

7.1 The step method

In a certain way, an agent's `step` method is the heart of its behaviour. Balls are presented with a cartesian world consisting of other balls, nails, and empty space. The `world` can (and should only) be queried and set through the `getObjectAtX:Y:` and `putObject:atX:Y:` methods.

So how to organize the logic within? Easy: get information, decide, move. For the balls, there seem to be several cases:

> If the way 'down' is clear, move there.
>
> If the place 'under' the ball is occupied, check if the positions left/right of that are occupied.
>
> > If yes, don't move.
> >
> > If one is empty, move there.
> >
> > If both are empty, draw 50-50 lots, move.

That's all. Here is the

19 ⟨*Ball::step definition* 19⟩≡

```
- (void) step
{
  id pos1Object, pos2Object;

  pos1Object = [world getObjectAtX: x Y: y+1];
  if (pos1Object == nil)
  {
    [world putObject: nil atX: x Y: y];
    [world putObject: self atX: x Y: y+1];
        return;
  }

  pos1Object = [world getObjectAtX: x-1 Y: y+1];
  pos2Object = [world getObjectAtX: x+1 Y: y+1];

  if (pos1Object && pos2Object) return;

  [world putObject: nil atX: x Y: y];
  if (pos1Object)
    [world putObject: self atX: x+1 Y: y+1];
  else if (pos2Object)
    [world putObject: self atX: x-1 Y: y+1];
  else
    if ([uniformIntRand getIntegerWithMin: 0 withMax: 1])
      [world putObject: self atX: x-1 Y: y+1];
        else
      [world putObject: self atX: x+1 Y: y+1];
}
```

7.2 The drawSelfOn: method

As it was planned to use pixmaps for the agents, a simple `drawPoint` message to the raster doesn't suffice. The pixmap has to be loaded from a file, and it should be taken care that this happens only once. So, the static handle `ballPixmap` is used for holding it. Remember, all static variables exist only once in memory, and are initialized to 0/nil, so the following idiom works nicely.

20a ⟨*Ball::drawSelfOn definition* 20a⟩≡

```
- drawSelfOn: (id <Raster>) aRaster
{
  static id ballPixmap;

  if (ballPixmap == nil)
  {
    ballPixmap = [Pixmap createBegin: [self getZone]];

    [ballPixmap setDirectory: [arguments getAppDataPath]];
    [ballPixmap setFile: "ball.png"];
    ballPixmap = [ballPixmap createEnd];
    [ballPixmap setRaster: aRaster];
  }

  [aRaster draw: ballPixmap X: x Y: y];

  return self;
}
```

8 Nail agents

Nails are much simpler than balls: only adaptation of three of `Balls` methods is needed. This is the header:

20b ⟨*Nail.h* 20b⟩≡

```
#import <objectbase/SwarmObject.h>

@interface Nail: SwarmObject
{
  unsigned x,y;
}
+ create: aZone;
- (void) setX: (unsigned) x Y: (unsigned) y;
- drawSelfOn: aWorld;
@end;
```

and here goes the implementation:

21 ⟨*Nail.m* 21⟩≡

```
#import <gui.h>
#import "Nail.h"

@implementation Nail

+ create: aZone
{
  return [super create: aZone];
}

- (void) setX: (unsigned) X Y: (unsigned) Y
{
  x = X;
  y = Y;
}

- drawSelfOn: (id <Raster>) aRaster
{
  static id nailPixmap;

  if (nailPixmap == nil)
  {
    nailPixmap = [Pixmap createBegin: [self getZone]];

    [nailPixmap setDirectory: [arguments getAppDataPath]];
    [nailPixmap setFile: "nail.png"];
    nailPixmap = [nailPixmap createEnd];
    [nailPixmap setRaster: aRaster];
  }

  [aRaster draw: nailPixmap X: x Y: y];

  return self;
}
@end
```

9 StringRepresentation of the initial world

There are no static member variables in Objective-C, but this is no loss as C-like statics fit the same niche perfectly. The header is empty except for the four method declarations.

22a ⟨*StringRepresentation.h* 22a⟩≡

```
#import "Position.h"

@interface StringRepresentation
{}
+ (unsigned) getSizeX;
+ (unsigned) getSizeY;
+ (void) resetToChar: (char) aChar;
+ (Position*) getNextPos;
@end;
```

The implementation file starts with the actual string array which serves as the database for the class, and can be easily changed with any fixed-width font text editor.

22b ⟨*SR database* 22b⟩≡

```
#import "StringRepresentation.h"
#include <string.h>

@implementation StringRepresentation

static char *str[] = {
"  100000000000001  ",
"  110000000000011  ",
"   1100000000011   ",
"    11000000011    ",
"     110000011     ",
"      1100011      ",
"       11011       ",
"                   ",
"         1         ",
"        1 1        ",
"       1 1 1       ",
"      1 1 1 1      ",
"     1 1 1 1 1     ",
"    1 1 1 1 1 1    ",
"   1 1 1 1 1 1 1   ",
"  1 1 1 1 1 1 1 1  ",
"                   ",
"1 1 1 1 1 1 1 1 1 1",
"1 1 1 1 1 1 1 1 1 1",
"1 1 1 1 1 1 1 1 1 1",
"1 1 1 1 1 1 1 1 1 1",
"1 1 1 1 1 1 1 1 1 1",
"1 1 1 1 1 1 1 1 1 1",
"1111111111111111111"};
```

Outside of the `resetToChar` method, there are additional statics declared, and set within for later use.

23a ⟨*SR resetToChar definition* 23a⟩≡
```
static char activeChar = '0';
static int currentX, currentY;

+ (void) resetToChar: (char) aChar
{
  activeChar = aChar;
  currentX = currentY = 0;
}
```

`getNextPos` searches through the string array to find the position of the next occurence of `activeChar`, returning a `Position` object or `nil`.

23b ⟨*SR getNextPos definition* 23b⟩≡
```
+ (Position*) getNextPos
{
  char c;

  while (currentY < sizeof(str)/sizeof(char*))
  {
    while ((c = str[currentY][currentX++]) != 0)
      if (c == activeChar)
        return [Position create: scratchZone
                withX: currentX Y: currentY];

    currentX = 0;
    ++currentY;
  }

  return nil;
}
```

The last two methods calculate the dimensions of the representation.

23c ⟨*SR dimensions calculation* 23c⟩≡
```
- (unsigned) getSizeX
{
  int i, max=0;

  for (i=0; i<sizeof(str)/sizeof(char*); i++)
  {
    int len = strlen (str[i]);
    if (len > max) max = len;
  }
  return max;
}

- (unsigned) getSizeY { return sizeof(str)/sizeof(char*); }
```

At last, the whole file:

24a ⟨*StringRepresentation.m* 24a⟩≡
⟨*SR database* 22b⟩
⟨*SR resetToChar definition* 23a⟩
⟨*SR getNextPos definition* 23b⟩
⟨*SR dimensions calculation* 23c⟩

10 The Position class

Following, without comment, the two respective files for this class.

24b ⟨*Position.h* 24b⟩≡
```
#import <objectbase/SwarmObject.h>

@interface Position: SwarmObject
{
  unsigned theX, theY;
}
+ create: aZone withX: (unsigned) x Y: (unsigned) y;
- (unsigned) getX;
- (unsigned) getY;
@end;
```

24c ⟨*Position.m* 24c⟩≡
```
#import "Position.h"

@implementation Position

+ create: aZone withX: (unsigned) x Y: (unsigned) y
{
  Position* obj;
  obj = [super create: aZone];
  obj->theX = x;
  obj->theY = y;
  return obj;
}

- (unsigned) getX { return theX; }

- (unsigned) getY { return theY; }
```

11 The file main.m

`main.m` is the entrance into the simulation, and the only thing to do here is to `initSwarm` and create the `Board`.

25a ⟨*main.m* 25a⟩≡

```
#import <simtools.h>  // for initSwarm
#import "Board.h"

int main (int argc, const char **argv)
{
  Board *theBoard;

  initSwarm (argc, argv);

  theBoard = [Board create: globalZone];
  [theBoard buildObjects];
  [theBoard buildActions];
  [theBoard activateIn: nil];
  [theBoard go];

  return 0;
}
```

12 The Makefile

The `Makefile` consists of the variable part and several targets.

25b ⟨*Makefile* 25b⟩≡

⟨*Makefile variables* 26a⟩
⟨*all target* 26b⟩
⟨*src target* 26c⟩
⟨*dvi target* 26d⟩
⟨*html target* 27a⟩
⟨*tarball target* 27b⟩

The first part defines variables to be used later. This code could be adapted easily for other apps by replacing variable content — it suffices to set the application name, and its objects.

26a ⟨*Makefile variables* 26a⟩≡

```
ifeq ($(SWARMHOME),)
# please set your SWARMHOME in your environment or put it here:
SWARMHOME=
endif
BUGADDRESS=ralf@ark.in-berlin.de

APPLICATION=bell
OBJECTS=main.o Board.o Balls.o Nails.o Ball.o Nail.o Position.o StringRepresentation.o
APPLIBS=
APPDIR=$(APPLICATION)

SOURCES=$(addsuffix .m, $(basename $(OBJECTS)))
HEADERS=$(addsuffix .h, $(basename $(OBJECTS)))
HEADERS:=$(filter-out main.h,$(HEADERS))
PATCHES:=$(shell noroots $(APPLICATION).nw | grep patch | sed 's/[<>]//g'\
| sort +1 -n -t-)
```

After that, the generic *swarm* Makefile is included and the `all` target set.

26b ⟨*all target* 26b⟩≡

```
include $(SWARMHOME)/etc/swarm/Makefile.appl

all: $(APPLICATION)
```

For the `src` target, several `noweb` tools and `patch` are used to extract the Objective-C source.

26c ⟨*src target* 26c⟩≡

```
src:
        @echo extracting original source files
        @for i in $(SOURCES) $(HEADERS) $(PATCHES);\
                do notangle -R$$i $(APPLICATION).nw |cpif $$i;\
                done
        @for i in $(PATCHES); do echo $$i:; patch <$$i; done
        @echo done
```

To produce the dvi file, the LaTeX source is extracted and compiled, and everything needed put into the DVI directory.

26d ⟨*dvi target* 26d⟩≡

```
dvi:
        @mkdir -p DVI
        rm -f $(APPLICATION).aux $(APPLICATION).toc
        noweave -n -x -latex $(APPLICATION).nw >body.tex
        latex $(APPLICATION).tex
        latex $(APPLICATION).tex
        latex $(APPLICATION).tex
        cp $(APPLICATION).dvi *.ps.gz *.ps.bb DVI
```

The HTML page is produced with LaTeX2HTML, put into the HTML directory, and made HTML-4.0 (transitional) conformant with Amaya-2.2.

27a ⟨*html target* 27a⟩≡

```
html:
        rm -rf $(APPLICATION) HTML
        mkdir HTML
        noweave $(NOWEBOPTS) -latex+html $(APPLICATION).nw >body.tex
        latex2html -split 0 -no_navigation $(APPLICATION).tex
        cp $(APPLICATION)/$(APPLICATION).html $(APPLICATION)/*.png \
                $(APPLICATION)/$(APPLICATION).css HTML
        rm -rf $(APPLICATION)
        amaya HTML/$(APPLICATION).html
```

For the tarball, a helper directory is created and the wanted files copied into that before archiving.

27b ⟨*tarball target* 27b⟩≡

```
tarball:
        rm -rf $(APPDIR)
        mkdir $(APPDIR)
        cp *.h *.m *.png *.ps.* Makefile $(APPLICATION).tex\
                $(APPLICATION).nw README $(APPDIR)
        cp -dpR HTML DVI $(APPDIR)
        tar cvfz $(APPDIR).tar.gz $(APPDIR)
        rm -rf $(APPDIR)
```

13 Debugging

At this time, the source compiles but, of course, there are still bugs and, from here on, changes were added as patch files named `patches-1` and so on. Patches are presented in unified format which should be the easiest to read. You can get yourself such patches by using revision control systems like `cvs` or `rcs`, or by giving the command `diff -u oldfile newfile`. In the `src` target of Section 12 (`Makefile`) the corresponding steps are shown to apply the patches to the hitherto written first version files.

13.0.1 Coredump: no colormap

In the first run, the simulation went on and produced a core dump. Looking at the backtrace with gdb (the `bt` command) reveals it happened when trying to `setRaster` in `Ball::drawSelfOn:`, and further examination shows that, inside swarm, a colormap is accessed but we never set the handle. This shows the argumentation on page 7 was flawed, the reason being that there is always a colormap needed (in other cases, swarm issues a warning but it did not in this case so a bug report with patch was sent to the bug address).

The latter means we have to patch the start of `Board::buildObjects` and include code for creating the `colormap`, setting a number of colors, and giving the colormap to `worldRaster`.

28 ⟨*patch-1* 28⟩≡

```
--- Board.m
+++ Board.m
@@ -11,11 +11,16 @@
 {
   id nails;
   id <Grid2d> world;
-
+  id colormap;
+  int i;
+
   [super buildObjects];
   [controlPanel setStateStopped];
   if ([controlPanel getState] == ControlStateQuit)
           return nil;
+  colormap = [Colormap create: self];
+  for (i=0; i<32; i++)
+    [colormap setColor: i ToRed: 0 Green: 0 Blue: 0];
   balls = [Balls create: [self getZone]];
   nails = [Nails create: [self getZone]];
   worldRaster = [ZoomRaster createBegin: self];
@@ -24,6 +29,7 @@
   [worldRaster setZoomFactor: 4];
   [worldRaster setWidth: [StringRepresentation getSizeX]
                Height: [StringRepresentation getSizeY]];
+  [worldRaster setColormap: colormap];
```

```
[worldRaster setWindowTitle: "Balls and Nails"];
[worldRaster pack];                                  // draw the window.
world = [Grid2d create: self
```

13.0.2 Window too small: zoom factor

Then, it comes to notice that the simulation window can't know its element size beforehand so the zoom factor is adjusted to 16 to make the window big enough.

29 ⟨*patch-2* 29⟩≡

```
--- Board.m
+++ Board.m
@@ -26,7 +26,7 @@
   worldRaster = [ZoomRaster createBegin: self];
   SET_WINDOW_GEOMETRY_RECORD_NAME (worldRaster);
   worldRaster = [worldRaster createEnd];
-  [worldRaster setZoomFactor: 4];
+  [worldRaster setZoomFactor: 16];
   [worldRaster setWidth: [StringRepresentation getSizeX]
                Height: [StringRepresentation getSizeY]];
   [worldRaster setColormap: colormap];
```

13.0.3 World has no nails: putObject

At this point, the simulation window looks like Fig. 4, and there are several things wrong.

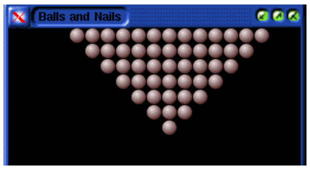

Figure 4: screen shot

We see the nail pixmaps are missing, but the balls are at the right places. Including `printf` statements into the `Balls/Nails` code reveals the nails are created with the correct positions. Switching the graphics file doesn't matter: no nail ever gets the `drawSelfOn:` message but for what reason?

It turns out that nail objects are never put into the world, which is the case with balls accidentally, in their `step` method. A solution is to add a `setWorld:` message to `Nails` as well, and call it from the `Board` code, so we have the following patches.

30a ⟨*patch-3* 30a⟩≡

```
--- Board.m
+++ Board.m
@@ -39,6 +39,7 @@
   [balls buildObjects];
   [nails buildObjects];
   [balls setWorld: world];
+  [nails setWorld: world];
   boardDisplay = [Object2dDisplay create: self
                    setDisplayWidget: worldRaster
                    setDiscrete2dToDisplay: world
```

30b ⟨*patch-4* 30b⟩≡

```
--- Nails.m
+++ Nails.m
@@ -25,4 +25,10 @@
   }
   return self;
 }
+
+- setWorld: aWorld
+{
+  [theNails forEach: M(setWorld:) : aWorld];
+  return self;
+}
 @end
```

31a ⟨*patch-5* 31a⟩≡

```
--- Nail.m
+++ Nail.m
@@ -1,4 +1,5 @@
 #import <gui.h>
+#import <space.h>
 #import "Nail.h"

 @implementation Nail
@@ -30,6 +31,12 @@

   [aRaster draw: nailPixmap X: x Y: y];

+  return self;
+}
+
+- setWorld: (id <Grid2d>) aWorld
+{
+  [aWorld putObject: self atX: x Y:y];
   return self;
 }
 @end
```

Fortunately, this bug leads us to the fact that `Ball` doesn't put itself into `world`, too, so finding the solution for the next bug at once with the patch:

31b ⟨*patch-6* 31b⟩≡

```
--- Ball.m
+++ Ball.m
@@ -1,4 +1,5 @@
 #import <random.h>
+#import <space.h>
 #import "Ball.h"

 @implementation Ball
@@ -17,6 +18,7 @@
 - setWorld: aWorld
 {
   world = aWorld;
+  [world putObject: self atX: x Y:y];
   return self;
 }
```

13.0.4 Board width: unknown cause

Now we can see the nails, and it looks like *swarm* adds two columns at the left side of the grid, so a quick hack makes the picture symmetrical again.

32 ⟨*patch-7* 32⟩≡

```
--- StringRepresentation.m
+++ StringRepresentation.m
@@ -4,7 +4,7 @@
 @implementation StringRepresentation

 static char *str[] = {
+"  100000000000001    ",
-"  100000000000001  ",
 "  110000000000011  ",
 "   1100000000011   ",
 "    11000000011    ",
```

13.0.5 No go: Ball x,y not set

Now appears the problem of balls not moving, but they get their `step` message, so it turns out the position that `ball` knows of itself isn't changed in `step`. The patch is easy.

33 ⟨*patch-8* 33⟩≡

```
--- Ball.m
+++ Ball.m
@@ -30,7 +30,7 @@
   if (pos1Object == nil)
   {
     [world putObject: nil atX: x Y: y];
+    [world putObject: self atX: x Y: ++y];
-    [world putObject: self atX: x Y: y+1];
         return;
   }

@@ -41,14 +41,14 @@

   [world putObject: nil atX: x Y: y];
   if (pos1Object)
+    [world putObject: self atX: ++x Y: ++y];
-    [world putObject: self atX: x+1 Y: y+1];
   else if (pos2Object)
+    [world putObject: self atX: --x Y: ++y];
-    [world putObject: self atX: x-1 Y: y+1];
   else
     if ([uniformIntRand getIntegerWithMin: 0 withMax: 1])
+      [world putObject: self atX: --x Y: ++y];
-      [world putObject: self atX: x-1 Y: y+1];
         else
+      [world putObject: self atX: ++x Y: ++y];
-      [world putObject: self atX: x+1 Y: y+1];
 }
 - drawSelfOn: (id <Raster>) aRaster
 {
```

13.0.6 Ball overflow!

The first run shows everything working nicely except the middle slots are filled to the brim causing havoc. Accordingly, we patch the starting board to give the slots more capacity.

34 ⟨*patch-9* 34⟩≡

```
--- StringRepresentation.m
+++ StringRepresentation.m
@@ -27,6 +27,13 @@
 "1 1 1 1 1 1 1 1 1 1",
 "1 1 1 1 1 1 1 1 1 1",
 "1 1 1 1 1 1 1 1 1 1",
+"1 1 1 1 1 1 1 1 1 1",
+"1 1 1 1 1 1 1 1 1 1",
+"1 1 1 1 1 1 1 1 1 1",
+"1 1 1 1 1 1 1 1 1 1",
+"1 1 1 1 1 1 1 1 1 1",
+"1 1 1 1 1 1 1 1 1 1",
+"1 1 1 1 1 1 1 1 1 1",
 "1111111111111111111"};
 static char activeChar = '0';
 static int currentX, currentY;
```

14 Conclusions about the first version

Several tries with the now debugged program confirm everything works as envisioned. Different random number behaviour can be seen when invoking `bell` with the `-s` option. Even the speed of the simulation is nice, on faster machines one would manually step through it.

What remains? At this point in time, there are still no illustrations done, so we'll finish the documentation with that and a thorough review. But the code flow that has kept us in a linear motivation, same as with the patient reader's attention hopefully, has crystallized into a small demo application that works with swarm-2 (other versions not tested). Not more did we want. R.S., April 2000

15 GNU Free Documentation License

35 ⟨*fdl.txt* 35⟩≡

```
                GNU Free Documentation License
                   Version 1.1, March 2000

 Copyright (C) 2000  Free Software Foundation, Inc.
     59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.


0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other
written document "free" in the sense of freedom: to assure everyone
the effective freedom to copy and redistribute it, with or without
modifying it, either commercially or noncommercially.  Secondarily,
this License preserves for the author and publisher a way to get
credit for their work, while not being considered responsible for
modifications made by others.

This License is a kind of "copyleft", which means that derivative
works of the document must themselves be free in the same sense.  It
complements the GNU General Public License, which is a copyleft
license designed for free software.

We have designed this License in order to use it for manuals for free
software, because free software needs free documentation: a free
program should come with manuals providing the same freedoms that the
software does.  But this License is not limited to software manuals;
it can be used for any textual work, regardless of subject matter or
whether it is published as a printed book.  We recommend this License
principally for works whose purpose is instruction or reference.
```

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
D. Preserve all the copyright notices of the Document.
E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
H. Include an unaltered copy of this License.
I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as

given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http:///www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (c)  YEAR  YOUR NAME.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.1
or any later version published by the Free Software Foundation;
with the Invariant Sections being LIST THEIR TITLES, with the
Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.
A copy of the license is included in the section entitled "GNU
Free Documentation License".
```

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being LIST"; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.