

**Department of Informatics
BSc Computer Science & Artificial Intelligence
2005**

The Multi-Agent Collectives Simulation System

**Written By
Iain Robinson**

Candidate No. 68181

**Supervisor
Sharon Wood**

Statement of Originality

This report is submitted as part requirement for the degree of Computer Science and Artificial Intelligence at the University of Sussex. It is the product of my own labour except where indicated in the text. The report may be freely copied and distributed provided the source is acknowledged.

28th April 2005

Iain P. Robinson

Acknowledgements

I wish to thank all past and current members of the comp.ai.*, comp.theory.* and comp.lang.java.* Usenet newsgroups for providing a seemingly limitless source of knowledge on the subjects involved in this report.

I also wish to thank Kris O'Shea and James Lovett for allowing me to use them as sounding boards for many of my ideas and last (but by no means least) my supervisor Sharon Wood for her excellent advice and guidance throughout the time spent on this project.

Summary

This report discusses the design and implementation of the Multi Agents Collectives Simulator (MACS) System. The MACS system is a design and development environment for the creation of Multi Agent Systems (MAS) and the artificial agents that populate such systems. The system is primarily for the design of situated, embodied, virtual agents but may also find a use in the design of other forms of agent.

This report provides a brief outline of the field of Multi Agent Systems particularly those of relevance to the MACS system such as Particle Based Systems and Individual Based Models – this includes a number of previous implementations of this style of application and comparisons are made between these and the MACS system to draw attention to the differences between them.

A full requirements analysis of the system is conducted and details of the implementation of the MACS system is provided with particular emphasis on how the system addresses each of the requirements found in the analysis stage. Full details of all systems testing is given and suggestions for further work and possible additions or changes to the system are proposed.

The version of the MACS system implemented provides the ability for the user to create and adapt scenarios of their choosing combining any number of agent “species” to study the various interactions that these agents exhibit. The application allows a user to design agents using a built-in Java editor and to compile these agents, ready for use within the application, using the standard Sun Java compiler.

Table of Contents

1. Introduction	6
1.1 The Structure of This Report.....	7
2. Professional & Ethical Issues.....	8
The BCS Code of Conduct	8
BCS Code of Practice.....	8
3. Related Work.....	10
3.1 Theoretical Works.....	10
3.2 Practical Implementations.....	11
3.2.1 Swarm.....	11
3.2.2 Breve.....	12
3.2.3 MatFa's Boids.....	14
4. Requirements Analysis.....	15
4.1 Application Concept.....	15
4.2 Target User Groups.....	15
4.3 Proposed Functional Requirements.....	16
4.4 Data Requirements.....	13
4.4.1 The Species File.....	13
4.4.2 The Scenario File.....	13
4.4.3 The Simulation-Run File.....	15
4.4.4 Import Security.....	16
4.5 Performance Constraints.....	17
4.5.1 Regional Binning.....	17
4.5.2 Graphical / Simulation Decoupling.....	18
4.5.3 Behavioural Algorithm Use Reduction.....	18
4.5.4 Obstacle Avoidance Algorithm Use Reduction.....	18
4.6 Usability Requirements.....	18
5. Implementation.....	19
5.1 Application Packages.....	19
5.2 Application Classes.....	19
5.3 Functional Requirements Implementation.....	22
5.4 System Operation.....	25
5.4.1 Initial System Start-up.....	25
5.4.2 Drag & Drop Interface.....	26
5.4.3 Importing New Species.....	27
5.4.4 Successful Importation.....	28
5.4.4 Creating a New Species.....	29
5.4.5 Species Generation.....	30
5.4.6 Species Compilation.....	31
5.5 Implementation Issues & Solutions.....	32
5.5.1 The Use of Multi-Threading.....	32
5.5.2 Syntax Highlighting.....	32
5.5.3 Automated Windows Installation.....	33
5.6 Testing.....	34
6. Conclusions & Further Work.....	35
6.1 User Feedback Mechanisms.....	35

6.2 Viewer Wizard.....	35
6.3 Code Efficiency.....	35
6.4 Redundant Code Elimination.....	35
7. References & Bibliography.....	36

1. Introduction

The subject of Multi Agent Systems (MAS) spans both the field of Artificial Intelligence and that of Artificial Life. There is no one “definitive” description of what actually constitutes an “agent” or a MAS since the concept is widely used in many different areas. Wooldridge & Jennings’ “Weak Notion of Agency”[16] can be thought of as the *least* that can be said about what constitutes an agent. It states that an agent must exhibit the three following characteristics:-

Autonomy – An agent must operate without human intervention and possess control of its actions and internal state.

Social Ability – An agent must interact with other agents in its environment.

Reactivity – An agent must perceive its environment and be able to respond to changes that occur within it.

It must be noted that this description provides no guidelines for how these characteristics should be achieved and that the designer of such an agent is free to determine exactly how an agent perceives its environment, effects changes in it and communicates with others – these notions being driven by the context of the agent rather than by the notion of agency. It can also be said that the characteristic of “social ability” requires that any agent, to be defined as such, must operate within a MAS.

Introducing any more than the most basic attributes given above tends to divide the notion of an agent along lines based upon the purpose of the agent. This being the case it may seem that creating a development environment for artificial agents maybe a difficult task if one is to make such a system applicable to all areas of use.

The system outlined within the body of this report is intended to provide a development environment for artificial agents using the weak notion of agency as its basis so as to provide the greatest level of adaptation to a users needs. The requirements of the system (in terms of what constitutes an agent) are kept to an absolute minimum in an attempt to make the system applicable to as many of the different types of agent as possible and to allow the user to use the application in a way that suits their own needs.

1.1 The Structure of This Report

Within the body of this report the following topics are covered:-

Ethical Issues – A discussion on the ethical and professional issues that can arise when developing any form of software application and those especially relevant to the MACS system. An outline of how these issues were tackled during the design and development of the MACS systems is also contained within this section.

Background – Provides an introduction to the field of Multi Agent Systems with a brief discussion of the different types of MASs that have been developed – particularly those that are relevant to the design and implementation of the MACS system.

Related Work – This section seeks to examine both the underlying theories of MASs and also discuss related applications that have been previously implemented. Within this section I evaluate the strengths and weaknesses of these related applications and discusses in what way the MACS system differs, and what aspects it borrows from them.

Requirements Analysis – A detailed description of the proposed manner of operation of the MACS system and illustrating the way in which each of the requirements made on the system is addressed.

Implementation – Discusses the implementation process involved in developing the MACS system, gives an overview of the various components of the system, provides graphical examples of the application in operation and points out a number of issues that arose during development and how they were resolved.

Test Results – Reviews the results of the testing scheme used and what aspects of the program design were or could be reconsidered in light of the results.

Conclusions & Further Work – Discusses the possible changes or additions that could be the subject of further work with the application, what alternative methodologies may have proved useful in light of the work already done, and considers the success of the project in terms of its initial objectives.

2. Professional & Ethical Issues

The ethical standards governing the computing profession in Britain are defined by the British Computer Society's (BCS) Code of Conduct[1] and Code of Practice[2]. Within these documents are a number of sections that could pertain to the development of the proposed system, although, since the work is quite abstract in nature, much of these documents do not apply. What follows is a brief outline of the relevant sections of the documents and how the issues are addressed within this project (section numbers in bold indicate the portion of the relevant BCS document the statement refers to).

The BCS Code of Conduct

Section 9 – “You shall not misrepresent or withhold information on the performance of products, systems or services...”

By conducting an accurate and extensive testing regime on the proposed system before submission I can fully evaluate its performance and be in a position to report any possible problems that testing may bring to light.

Section 15 – “You shall not claim any level of competence that you do not possess. You shall only offer to do work or provide a service that is within your professional competence.”

The initial proposal of the system outlined the work to be done (also extended upon in this document) and the different technologies to be used during design and implementation. I have, in my opinion, a high level of competence in all these technologies and view this project to be viable and achievable in the time given.

BCS Code of Practice

3.1.3 – Project Planning

At each stage of the project reasonable outlines of the proposed time scale of the project have been submitted with design goals, aims and objectives clearly stated. Much time has been devoted to the active research of other work in this and related fields of study and to investigating previously implemented software systems that are related to that discussed within this report.

3.1.6 – Progress Tracking

Throughout the time spent on the project an accurate and detailed account of all work done in relation to the project and the time spent on each individual task was kept.

3.1.7 – Project Closing

A good final testing scheme, combined with constant usability testing and ongoing testing during the development process has provided the ability, at the close of the project, to honestly summarise any mistakes made, good fortune and lessons learned from the project - as well as recommend changes and/or further work that may be possible – These can be found towards the end of this document.

4.2.1 – Research

Much research into related topics done by others (much of which is presented here) and has been carried out and any acknowledgements have been indicated where necessary in an attempt to gain as much information as possible into the work already carried out and any useful information that these studies have provided.

5.1.1 – Requirements Analysis

A detailed analysis of the requirements (presented here), was performed using recognised methods, and documented using an appropriate method (the UML) in a bid to assure that the objectives set forth in the initial proposal were technically feasible.

5.2.2 – Software Design

As explained within the requirements analysis the proposed system was to be usable by experienced and novice users alike. As such the system was designed for use by both user-groups from the outset.

5.2.4 – Good Programming

During the course of project development much emphasis has been placed on producing well structured (Java) code by following the programming guidelines set down by the Sun Java Coding Conventions[13], the use of a number of code style/re-factoring tools and including detailed documentation at all stages. In the initial proposal a need for the code to be platform independent was indicated – hence the choice of Java as the programming language used to produce the MACS system.

5.2.5 – Good Testing

A rigorous and ongoing testing policy using the JUnit testing scheme combined with regular usability tests was enforced to accurately and effectively test the proposed system. The use of JUnit testing allowed tests to be automated, re-run when needed and for a detailed log of all tests to be produced.

5.2.8 – Technical Documentation

All the code that makes up the MACS system is documented as per the Java Coding Conventions cited earlier. This allows maintenance of the code by either myself or others at any time in the future and provides a clear description of the code's functionality.

5.2.9 – User Documentation

It is my intention to provide clear and concise user documentation for the proposed system (containing diagrams and screen-shots where needed) suitable for both novices within the field and those with more experience. It will cover all aspects of how to both install and use the proposed system.

3. Related Work

3.1 Theoretical Works

Possibly the earliest form of MAS were Cellular Automata. These were first investigated by John von Neumann and Stanislaw Ulam during the 1940's[7] and were used to study biological processes such as self-reproduction. Although one of the oldest branches of Artificial Life research into their properties still continues [15], and can be seen as the most basic form of MAS. CA consist of a group of identical "cells" that inhabit a discrete "lattice" environment. They work on similar principles to those of the type of agents considered here but on a much simpler level. The main difference between the two is that CA do not fulfil the definition of agency given above (unless it is considered *very* weakly) since the individual cells involved perform only very limited computation and effect changes in the environment only indirectly (either through changes in their own internal state or through their presence, or absence, in the environment).

A more complex example of a MAS is illustrated by Particle Systems[9]. These type of systems are used a great deal in the field of animation to model "complex" systems such as smoke, fire and clouds. In a similar fashion to CA they use collections of identical, but simple, "points" (i.e. having no geometric structure or volume to speak of) that are designed to exhibit a variety of "behaviours". Particle systems do not require any interaction between particles (i.e. they do not conform to Wooldridge and Jennings' concept of social ability) whereas the "boids" form of MAS requires it.

The Boids Flocking Model[10] is an extension of the typical particle system. The difference between the two systems is small but significant. In the boids model the point based particles are replaced by objects possessing a full geometrical structure, a local coordinate system and as a result an "orientation" within their environment. As noted earlier this model differs from typical particle based systems in that it requires a strong sense of interaction between individuals and that behaviour is governed by both internal and external influences.

Although these systems exhibit quite different forms of behaviour the differences between them are one of degree but not of kind. The boids model exhibits forms of behaviour orders of magnitude greater in complexity than that of a particle based system, which in turn is more complex than a standard CA. All share the same type of behaviour, just to a greater or lesser extent. It is this common behaviour that makes each of these systems suitable for development within the MACS systems.

3.2 Practical Implementations

There are many applications that are available for people wishing to develop multi agent group simulations. Some are aimed at designing agents suited for specific problem areas while others adopt a more general approach to agent development. Here is brief discussion of three different systems and the varying approaches they take to the subject.

3.2.1 Swarm

The Swarm system[14] is one of the most widely used systems for the development of MAS. The package comes as a self-extracting file and once installation is complete the user can view a README file which provides links to on-line user documentation and demo files, although neither come with the package. Each demo file provided with the application opens as an executable file in its own right and provides graphical control of its own parameters. The agent development language of choice within Swarm is Objective C since it is, to quote the documentation, “a very simple extension to the C language...[and] has a high quality, freely distributable implementation in the GNU C compiler”.

It has been noted by others[6] that although Swarm is a very powerful package and is available on a wide range of platforms there are a number of areas in which it is lacking. In particular its usability and use of the Objective C language make it difficult for new users to being to use. After my own investigation of the system I have to agree with these previous findings.

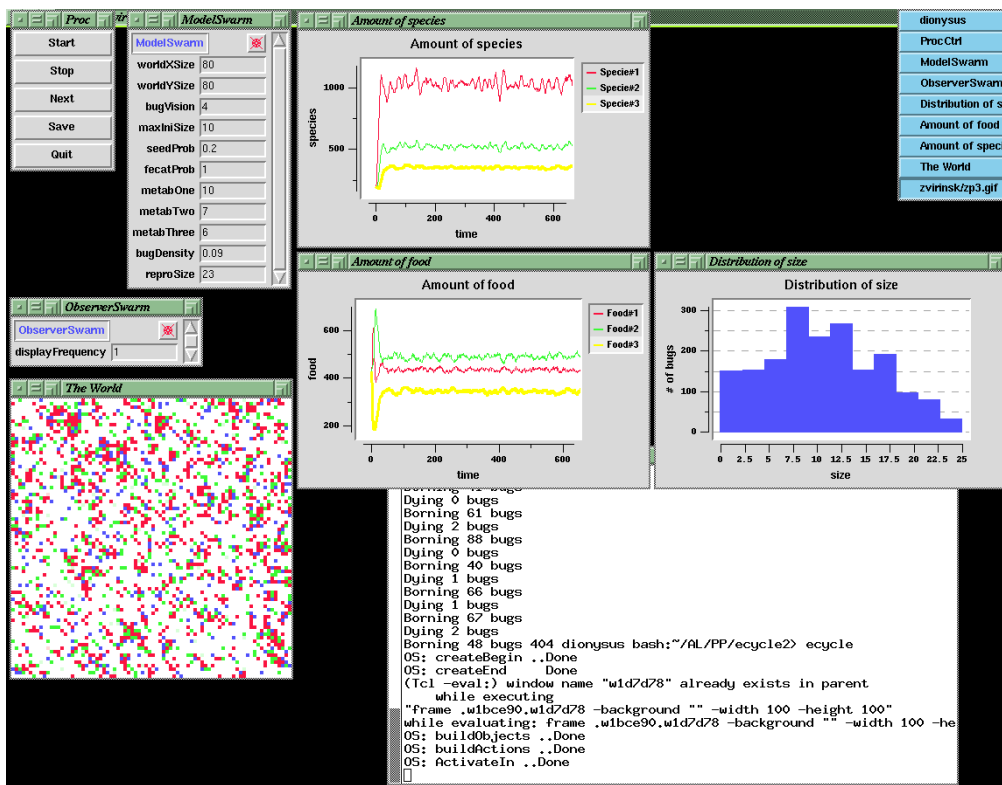


Figure 1 - Swarm Application Screenshot

3.2.2 Breve

“Breve”[6] is an open-source application developed primarily to aid the creation of 3D multi-agent and physically based simulations and has a built-in physics engine with features like collision detection especially for this purpose. The package is available for a number of platforms (Mac-OSX being the most widely supported) and comes as a self extracting ZIP. The program requires Windows users to download an Open-GL library separately to use the program and users are also required to set up class-paths – a feature familiar to users of Java but not necessarily to all users.

The application has only a command line interface within Windows, although a GUI is available on Mac-OSX. It comes with a number of demo programs that illustrate its range of applications. The graphical output is reasonable but highly processor intensive and for all but the most simple the output is not “real-time” (at least not on the machine used for this test – a 1.6Ghz PC running Windows XP). While a simulation is running the user has a number of functions available to them through a right-click “context menu” and these are to some degree definable by a simulation's creator. The application uses its own declarative language, called “Steve”, allowing users to create their own simulations.

Overall the system is very similar to the Swarm system discussed earlier although is directed towards an area not addressed by Swarm (i.e. 3D simulation). Its underlying program architecture is very good but this is not apparent from the command-line GUI. It has a number of features not available within other applications of this type such as the physics engine. It is the opinion of the author that the learning curve for a novice would still be quite steep, but probably not as steep as with Swarm.

In terms of its underlying architecture it is this program that most resembles that planned for the proposed system. The application has a basic Object class that is extended by 2 classes – AbstractObject (that have no representation in the environment) and RealObject (specific physical objects) and this RealObject class has its analogue in the BaseEntity class in the MACS system (see later class diagrams for details). In the “breve” system there is a distinction in the RealObject class hierarchy between static objects (e.g. obstacles) and dynamic objects (agents) – at the current time this differs from the design of the MACS system although this may change in the future where the distinction between dynamic and static objects is not made.

Table 1 Examples of Breve Output

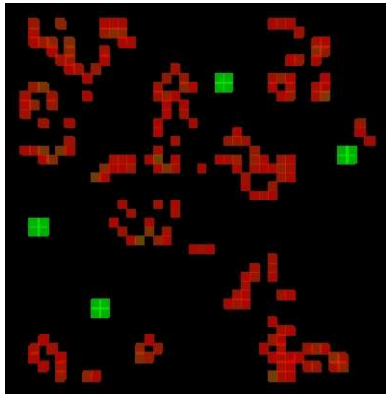


Figure 2 - 2D Game of Life

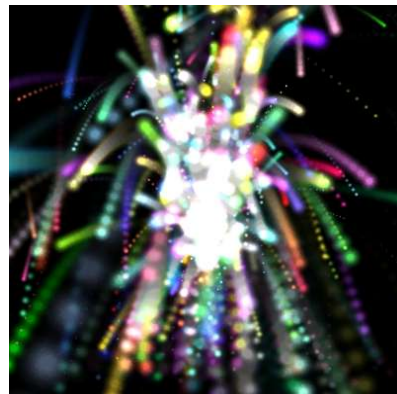


Figure 3 - Particle Based Animation

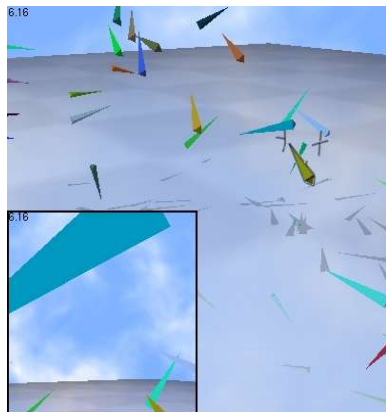


Figure 4 - 3D Boids Model

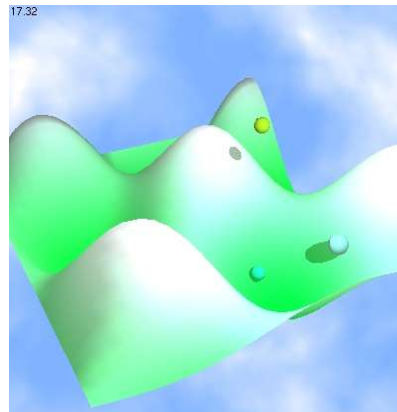


Figure 5 - Spheres & Terrain Model

3.2.3 MatFa's Boids

Mattias Fagerlund's application[3] is a good example of an implementation of a multi-agent system. Although the system is solely devoted to the Boids Model discussed earlier it provides a good illustration of exactly how diverse such a model can be and the range of behaviour such a dynamic system can exhibit. All the program's graphical output is 2D in nature (using a point rather than geometry based implementation) and the user interface is simple enough to be used by novices while still retaining enough diversity of control to be of some interest to more experienced users. The application can illustrate graphically the triggering of the individual "rules" that control boid behaviour in real-time. Users cannot add new rules by which boid behaviour is governed but it is possible to switch on/off the rules that each boid uses – thus allowing a small amount of "customisation" by the user.

Overall this application is easy to use and illustrates the concepts and results of one particular type of MAS in a concise manner. While the amount of customisation a user can perform is minimal the program is usable even by novices with instant results using a simple but informative GUI.

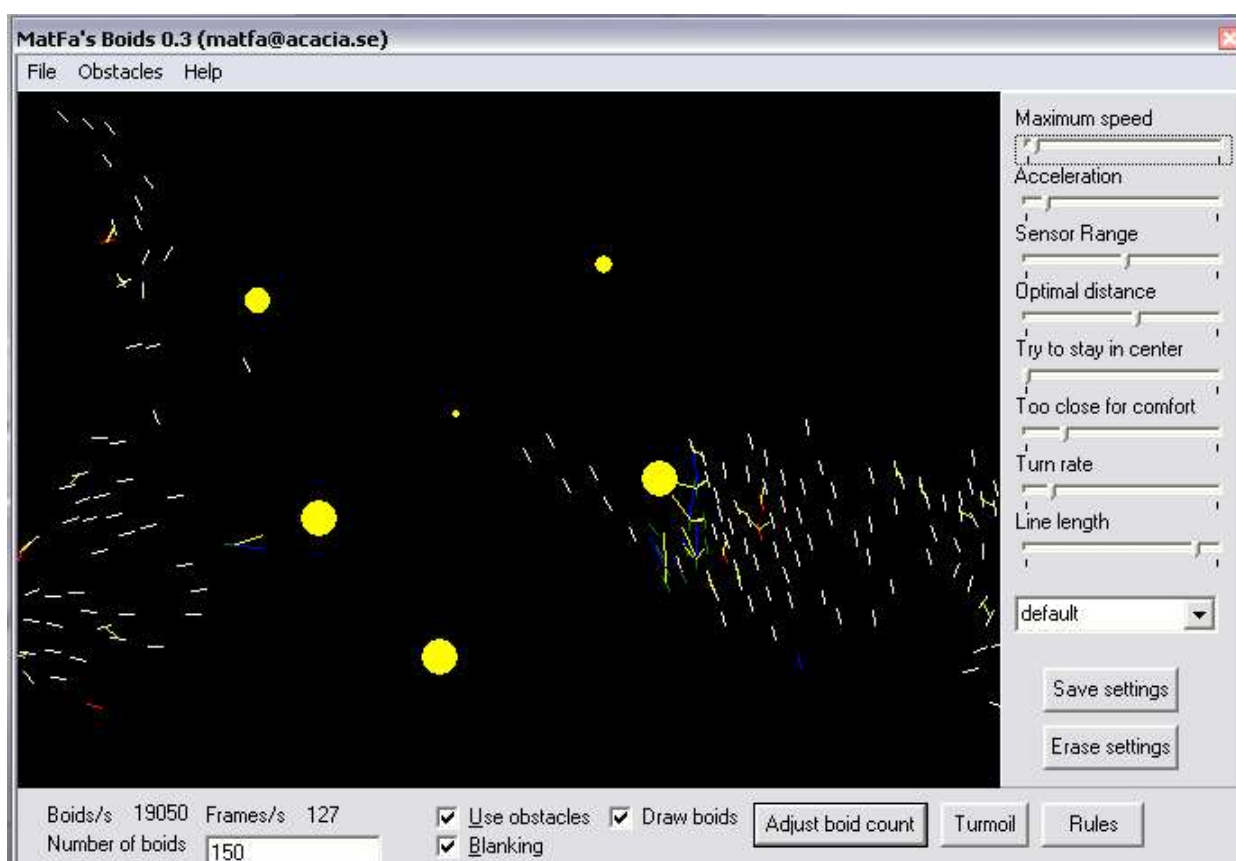


Figure 6 - MatFa's Boid Screenshot

4. Requirements Analysis

4.1 Application Concept

This project aims to produce a software application suitable for simulating the intra-group and inter-group interactions that occur between any number of user specified agent groups of any (reasonable) size. The three fields of related work discussed earlier exemplify the evolution of the type of agent that the proposed system is to be concerned with. These are, to paraphrase one researcher in the field, “situated, embodied, reactive [and deliberative], virtual agents” (Reynolds[11] pp.2).

The application is to be primarily concerned with the simulation of inter/intra “species” actions within a given environment and, although important, the quality of the graphical output of such a simulation is secondary. This being the case the option to represent the simulation (at least initially) in a 2 dimensional display was taken as it was felt that: -

- a) This would be less difficult to achieve within the given time than a 3D alternative.
- b) The quality of the results given by a 3D simulation would improve only in terms of data visualisation, rather than numerical accuracy, and it was felt that the overall amount of time that would have to be devoted to developing a 3D environment of reasonable quality was excessive for any benefit that would have been added.

4.2 Target User Groups

The proposed system is aimed at two groups of users in particular: -

The “developer” – This user will typically have a background in, or a thorough understanding of, the field of MAS or an application area. The user will use the system to build scenarios and incorporate both species developed by others and personally. In this case the system needs to be as flexible as possible so as to allow full creative influence on the outcome. This user may be a University Lecturer designing examples for students, a researcher in the field or an “viewer” level user wishing to start developing scenarios.

The “viewer” – This user typically will have no need to design agents or scenarios and will use the software only to view the results of a “developer's” work. They may be a casual browser, a person developing an interest in the subject or a student of the subject wishing to view scenarios developed by a tutor for educational purposes. In this case little or no previous knowledge with regards to the design of a MAS should be assumed.

4.3 Proposed Functional Requirements

The proposed application needs to allow the user to perform the following tasks:-

1. Write the code (in Java) for a “species” of agent. This code must extend from a supplied “base” class so as to ensure it conforms to a prescribed standard. This will allow agents designed elsewhere to effectively inhabit the same environment as those designed by the user – thus promoting extensibility, code-sharing and portability.
2. Import species of agent written by the user or others into the application with an aim to including them within a particular scenario.
3. Specify the parameters of the scenario that have a direct influence on real-time performance. Examples of these include (but are not limited to), the obstacle avoidance rate, the amount of Simulation/Rendering “de-coupling” and the amount of Behavioural Minimisation (more on these terms follows).
4. Create a “scenario” file that contains all data regarding a particular “world set-up” including all environment parameters, locations of “species” members and simulation engine parameters.
5. Specify the conditions applied within a given virtual environment, for example, whether the environment is infinite in extent, has a defined limit or boundary and if so what conditions are enforced at those extremes (so called “boundary conditions”).
6. Position individual species members within the chosen environment.
7. Choose either a real-time simulation (shown graphically) or a deferred-time simulation (written to a file)
8. Control both the temporal and spatial aspects of the real-time simulation through an intuitive set of controls.

These functional requirements are outlined, as UML Case diagrams, in the figures on the following pages. Each diagram indicates the functionality that each type of user will require from the system. The functionality is divided between two diagrams only to aid clarity.

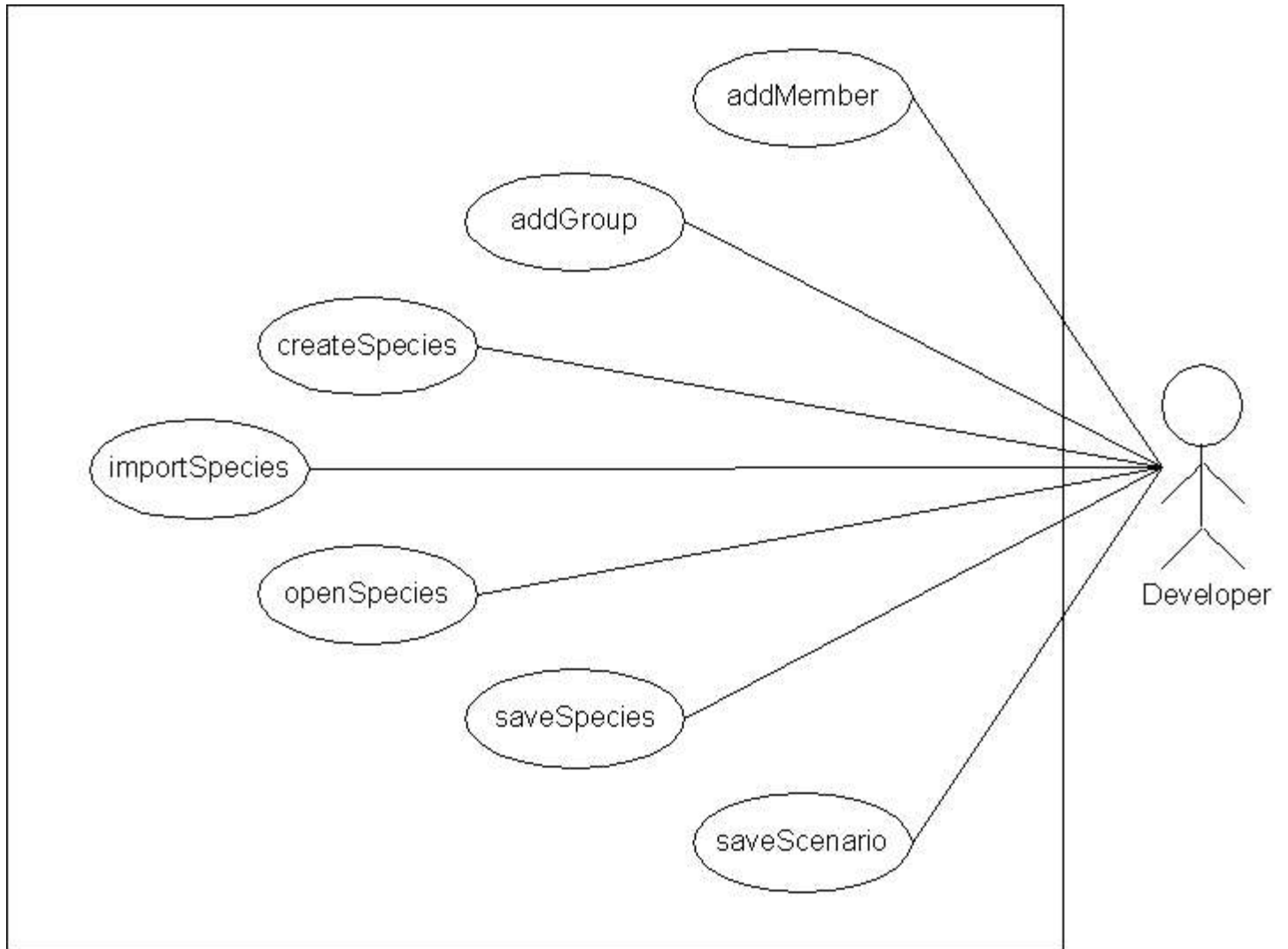


Figure 7 - Developer Functionality

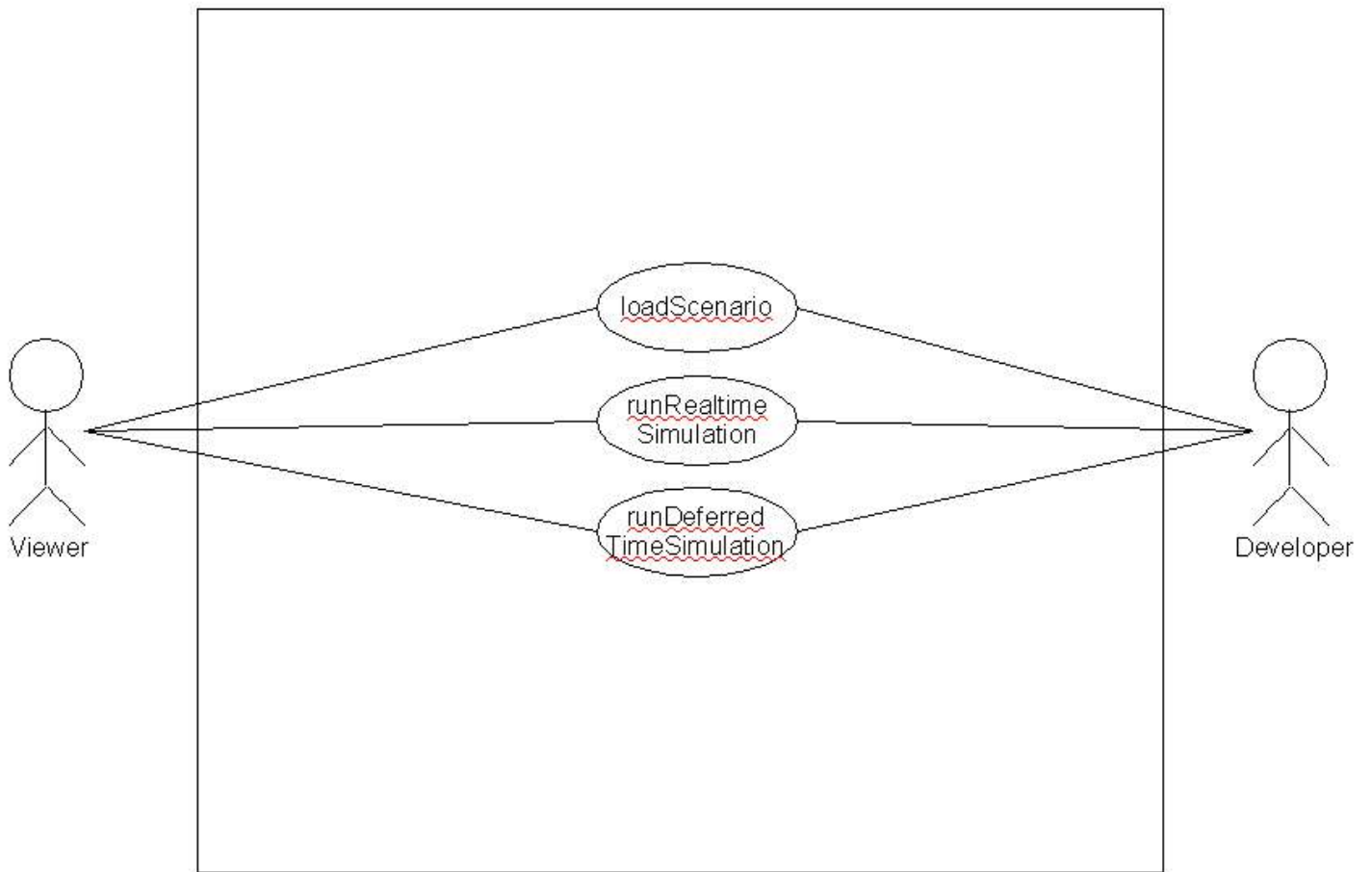


Figure 8 - Developer & Viewer Functionality

The above list of functional requirements should indicate to the reader that the proposed system will, by necessity, be quite a “loosely coupled” system since its main purpose is to facilitate the wishes of the user. This being the case the application must be a highly customisable structure whilst ensuring that the amount of freedom given to a user cannot “destabilise” the application and cause it to malfunction. It is for this reason that the proposed application will communicate with external (i.e. user made) objects via a set of interfaces that will require certain elements to be present in the user made objects so that application stability can be guaranteed.

The following figure illustrates the main components of the system and the main flow of data between the two interface classes BaseEntity and Viewer. These classes will be those responsible for ensuring that a user object is valid.

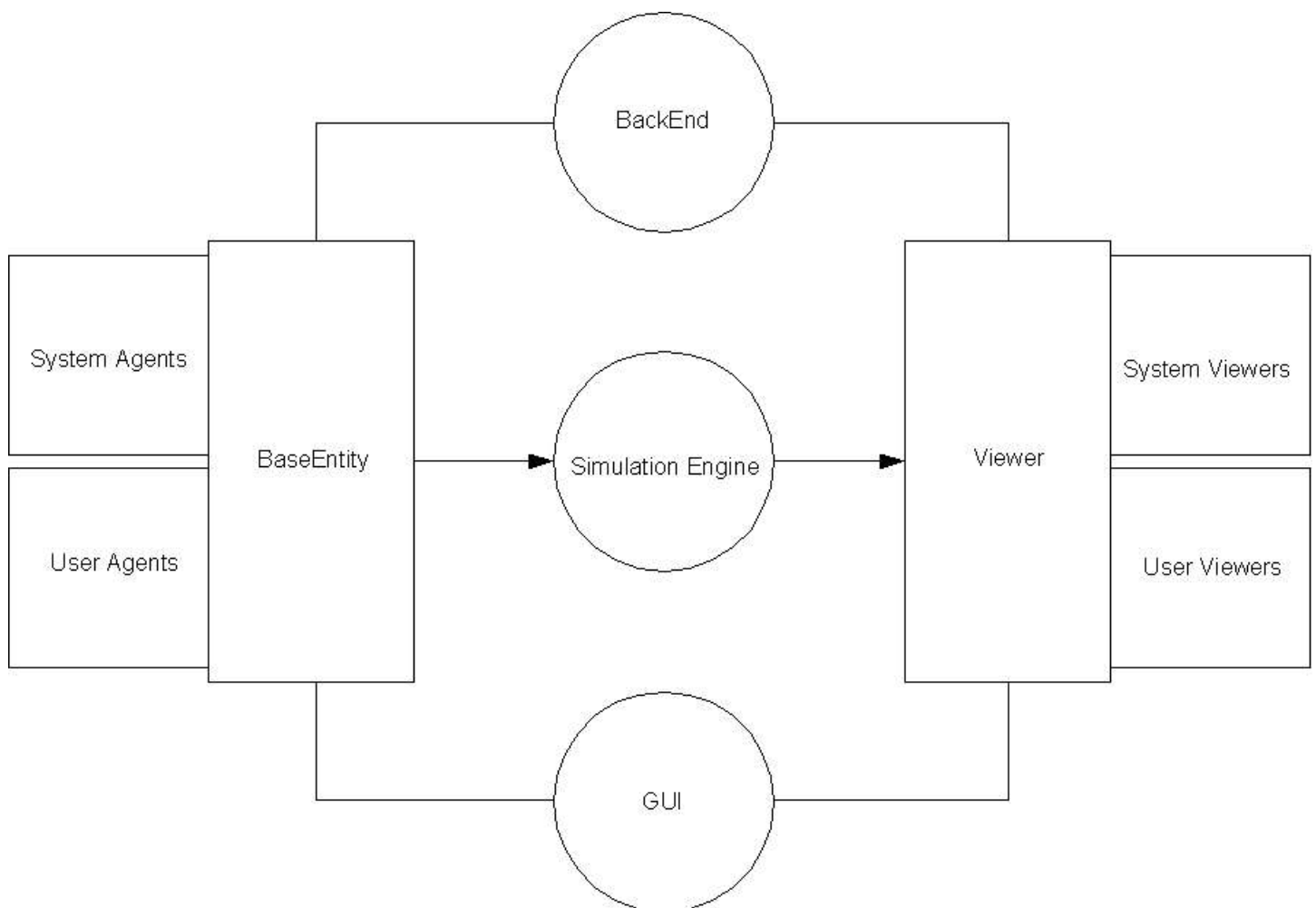


Figure 9 - Overview of MACS System

Over the course of the implementation process the design slowly evolved to resemble that given in the next figure. The figure illustrates only the core of the system and a number of utility classes are not shown to maintain clarity. The reader is directed to section 5.2 to obtain full details of all the classes used by the application. Also shown are 3 examples of how user defined classes, written using the application and extending the BaseEntity class, are situated within the overall structure of the system.

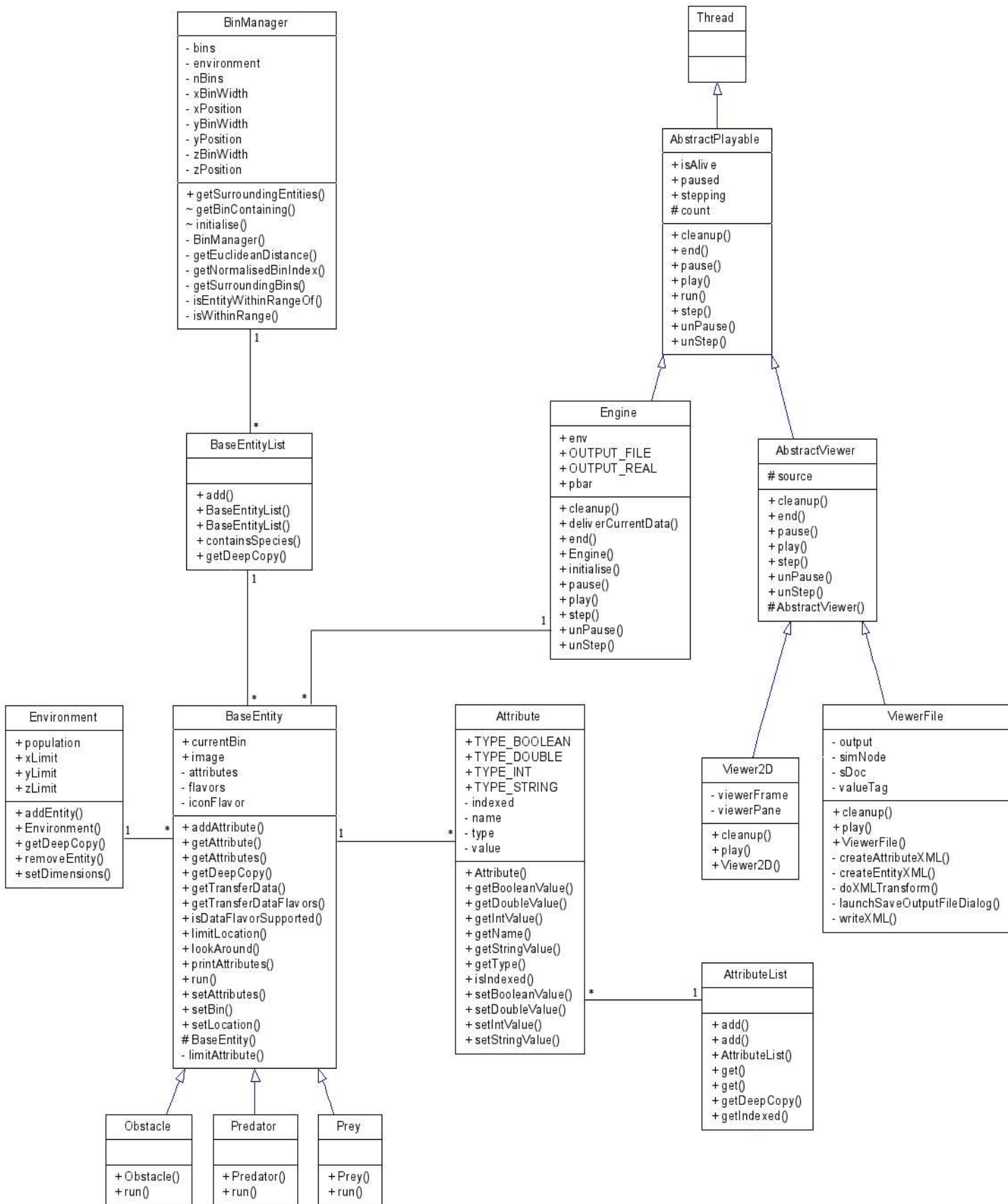


Figure 10 - Application UML Class Diagram

4.4 Data Requirements

The MACS system stores two types of data in persistent storage, the scenario and the species file types. The following is a brief outline of some of the predetermined data types that could be possible within the system:-

4.4.1 The Species File

Since each species of agent will be constructed using the Java language and will be imported directly into the application it must be (at least in its final form) a legitimate Java .class file. This being the case however it will also be necessary for it to be written to a Java source file (however temporarily) before it can be compiled.

Although the .class file is the minimum requirement of the system, and a Java source file is not required by it, it would, however, be sensible to store the intermediate source file for 2 reasons:

- The development of an agent may take a certain amount of refinement and since Java .class files are not human readable it is necessary for a developer to work with the intermediate file (this is standard usage).
- The development time of new species types would be greatly reduced if users had the ability to refer back to species written either at an earlier date or by others – either to use as a template for another species or to copy the functionality of the species.

To accompany this species file there is need of a visual representation of the species for use within the application. It was decided quite early in development that the GIF image format would easily meet all the requirements needed by the system. The final result being that all species classes are accompanied by a corresponding GIF image to represent that species.

4.4.2 The Scenario File

The scenario file type contains all parameters pertaining to a certain simulation. Since the underlying species being simulated are in a platform-independent format (.class) it is sensible to extend the same benefit to the format of this file. For that reason it was decided to store the scenario file in an XML format. An simple example of the structure of this file is shown below: -

```
<?xml version="1.0" encoding="UTF-8"?>
<environment>
  <xLimit value="500"/>
  <yLimit value="500"/>
  <zLimit value="500"/>
  <population>
    <entity type="species.obstacle.Obstacle">
      <attribute indexed="true" name="xPosition" type="0" value="135"/>
      <attribute indexed="true" name="yPosition" type="0" value="131"/>
      <attribute indexed="true" name="zPosition" type="0" value="0"/>
    </entity>
    <entity type="species.obstacle.Obstacle">
      <attribute indexed="true" name="xPosition" type="0" value="195"/>
      <attribute indexed="true" name="yPosition" type="0" value="251"/>
      <attribute indexed="true" name="zPosition" type="0" value="0"/>
    </entity>
  </population>
</environment>
```

This example contains four important elements, or *tags*: -

Environment Element – This is the “root” tag for the entire document – which makes sense if we consider that the document is attempting to record the information for a static or unchanging environment. This tag has no attributes itself but does contain tags for each attribute of the environment – currently these are simply the maximum boundaries in each of the 3 dimensions – xLimit, yLimit and zLimit respectively. This tag also acts as the parent tag for the population element of the scenario file.

Population Element – This tag again has no attributes itself but is used as a parent tag to indicate the area of the file devoted to what the environment contains within itself.

Entity Element – The entity element contains only one attribute, *type*, that indicates the *species* of entity this portion of the file is describing. This attribute is actually used within the application to dynamically find and load the Java .class file that represents the species into the system.

Attribute Element – An attribute tag represents a single attribute of the species. An entity element can contain an arbitrary number of attribute elements since these can be defined by the user for each species they create – but will contain at least three – inserted by the application – that represent the spatial location of the entity within the environment.

Each attribute element contains four XML attributes itself: -

The *indexed* attribute contains a boolean value that indicates whether or not this particular attribute will be included in any output to a simulation viewer (whether or not this is a real-time or file based viewer). This allows a user to streamline the data passed to a viewer and restrict it to only data that is important or relevant to the system they are simulating. In other words - *all* attributes are written to the scenario file but only essential attributes will be included in a simulation-run file.

The *name* attribute contains a string indicating the name of the attribute.

The *type* attribute contains an integer that indicates the type of the attribute. This value is used within the system to correctly interpret the data being read from a scenario file. An attribute can be any one of four types, these integer values are as follows: -

- 0 – The attribute is interpreted as an integer type.
- 1 – The attribute is interpreted as a string type.
- 2 – The attribute is interpreted as a double type.
- 3 – The attribute is interpreted as a boolean type.

Using this mechanism the system is able to distinguish and correctly interpret the users intended format for the data and is able to distinguish between the string “1.23” and the equivalent double value or the string “true” and the equivalent boolean value.

4.4.3 The Simulation-Run File

As the application allows a user to select a “deferred-time” simulation run of the scenario it becomes necessary to store the results of this simulation so it can be used in further processing by other applications. Again the XML format seems to suggest itself as a solution to this requirement as easy use of a file of this type can be done in a number of ways and from within a wide range of programming languages.

The file format contains time varying data regarding the position of each agent within the scenario at each time frame. This, of course, is as an absolute minimum-other parameters would be probably be highly desirable (such as an agent’s species type) so the minimum that a file would contain would be something approaching the following: -

```
<?xml version="1.0" encoding="UTF-8"?>
<simulation>
  <frame count="0">
    <entity type="species.obstacle.Obstacle">
      <attribute indexed="true" name="xPosition" type="0" value="179"/>
      <attribute indexed="true" name="yPosition" type="0" value="83"/>
      <attribute indexed="true" name="zPosition" type="0" value="0"/>
    </entity>
  </frame>
  . . . . .
  <frame count="100">
    <entity type="species.obstacle.Obstacle">
      <attribute indexed="true" name="xPosition" type="0" value="179"/>
      <attribute indexed="true" name="yPosition" type="0" value="83"/>
      <attribute indexed="true" name="zPosition" type="0" value="0"/>
    </entity>
  </frame>
</simulation>
```

The Simulation-Run file is very similar in design to the Scenario file format. It can be thought of as a dynamic scenario file containing time varying data whereas the Scenario file only defines the static data needed to construct the initial environment.

As with the Scenario-Run file the example given above contains four XML elements: -

Simulation Element – This is the “root” tag for the entire document. This tag has no attributes itself and acts only as the parent tag for the frame element of the simulation-run file.

Frame Element – This tag has only one attribute, *count*, that indicates the number of each frame in the simulation. This element provides the dynamic, or time varying, aspect of the file as it represents the state of each individual in the environment at a particular instant in time.

The remaining two elements, *entity* and *attribute*, are exactly as those of the same name encountered in the Scenario file and details of each can be found in the previous section regarding that file format.

4.4.4 Import Security

One possibility which needs careful attention concerns the Species Import functionality of the application. As a user is able to select the species to import from a file dialogue box the situation may arise where a user selects a valid .class file but one that does not conform to the requirements of the standard interface used by the system to control a species member – A user selecting a JButton.class file for example.

There are 2 ways in which this problem of security can be addressed. The first involves limiting the file extensions viewable within the relevant file dialogue box (which it may be possible for a user to bypass). The second involves performing a series of “compatibility tests” on the class file either before, or immediately after, importation. This would confirm that the contained class did indeed conform to a specific interface and possibly detect the presence of malicious code.

4.5 Performance Constraints

The amount of resources required at runtime for this type of system is more difficult to accurately quantify than the data storage requirements and is different for real-time and deferred time simulation runs. A simple “rule of thumb” is that, for an application such as, this the more system resources you can allocate (in terms of processing power, RAM etc.) the more complicated your scenarios can become and the more species instances you can simulate.

When using the proposed system in “real-time” mode the main aim is to be able to simulate as many species instances as possible without there being a noticeable impact on the graphical output of the simulation. As graphical output can consume a great deal of system resources it makes sense to streamline the performance of the system as much as possible to as to achieve a “glitch-free” simulation run.

In deferred time mode the graphical problems are not an issue although an analogous problem occurs and concerns the total length of time taken to run a simulation and write it to file.

To minimise the impact of a simulation’s complexity and reduce its simulation overhead (whether in real-time or deferred-time mod) there are a number of techniques that can be utilised, these include.

4.5.1 Regional Binning

The main overhead associated with simulating a large number of entities in a distributed environment such as that involved here is the method used to determine, for each entity, which other entities are present in its own particular field of view. A naive implementation would involve a 1 stage process whereby, for each entity’s lookup, examining the location of every other entity in the environment would be necessary.

To implement the lookup process as efficiently as possible a sensible data structure needs to be used to represent the current state of the entire environment. Regional Binning, amongst others, has been found to be an effective method of reducing the amount of “lookup” time necessary in a simulation[12].

Regional Binning involves creating a spatial data structure to represent the distribution of entities within the environment. The whole environment is subdivided into a set number of “bins”, each representing a certain area of the environment. At the start of the simulation each entity is placed in a bin based upon its current location in space. Each “bin” is actually a list of all the entities currently in that region of space and each entity possesses a link back to the bin that contains it.

Using this type of structure any “lookup” then becomes a 2 stage, rather than 1 stage, process. Stage 1 involves calculating only which bins are within an entity’s field of view. During stage 2 all other entities in only these bins are examined to see if they lie within the current entity’s field of view.

4.5.2 Graphical / Simulation Decoupling

This technique involves separating the simulation of the scenario with the graphical representation of the scenario. In other words – since graphical output has a high overhead associated with it – is it possible to reduce the number of frames displayed in relation to the number of simulation frames produced. A 1:1 ratio of simulation frames to output frames may not be necessary (especially during scenario development). The ratio can be adjusted so that more than one simulation frame is allowed to be performed for each graphical frame.

4.5.3 Behavioural Algorithm Use Reduction

This technique involves specifying that only a certain percentage of the population of a scenario (chosen at random) perform any computation related to behaviour at each simulation frame. Any members of the population not selected merely keep whatever internal state they had at the last simulation frame constant and do not “react” to any new stimuli – i.e. they “continue on their present course” – but are still drawn to the graphical display.

4.5.4 Obstacle Avoidance Algorithm Use Reduction

This technique is similar to the above but involves selecting which members of the population execute their obstacle avoidance routines at any simulation frame. Any members of the population not selected simply disregard any obstacles currently in their field of view.

4.6 Usability Requirements

Since the proposed system is to be designed with both novice and expert users in mind the issue of “usability” is of particular importance. In designing how the system is to be used one must address the needs of two (in many ways opposing) groups of users. The approach taken here is to have two distinct “modes” of the application: -

Viewer Mode – Only temporal/spatial controls and the environment view will be available allowing a user only to view the initial environment set-up and run scenarios but none of the more advanced scenario set-up or agent editor facilities.

Expert Mode – This mode will include all functionality available within the novice view but include extended functionality such as the agent editor facilities and advanced scenario set-up.

5. Implementation

5.1 Application Packages

The source code for the program is divided into five packages - not including the JEdit source code package which will not be covered within this report. Each package is responsible for one aspect of the overall program.

The following is a brief description of the purpose of each of the four packages and the role it plays within the whole application: -

Package Name	Role & Responsibilities
<i>backend</i>	This package contains data structures that cannot be modified by the user and the simulation engine that is responsible for running each of the simulation scenarios created by a user.
<i>gui</i>	This package provides the user interface used to reflect the state of the backend system. It also contains the “glue” classes used to pass data from the GUI to the back-end structures – in Java these are commonly called “listeners” since the “listen” for mouse and keyboard events generated as the user navigates the user interface.
<i>common</i>	This package contains classes that are utilised by classes in both the <i>backend</i> and the <i>gui</i> packages
<i>datatypes</i>	This package contains classes similar in role to those in the <i>common</i> package but also includes classes that are used by any species or viewer that can be designed by the user.
<i>viewers</i>	This packages contains two example viewers that can be used to monitor the results of a simulation.

5.2 Application Classes

The following table contains a brief description of each of the classes actually implemented in the application describing the role and responsibilities of each one in some detail: -

Class Name	Role & Responsibilities
<i>AbstractPlayable</i>	This class provides added functionality to the Thread class that allows the easy ability to pause, resume and "step" over the underlying Thread's run() method.
<i>AbstractViewer</i>	This class provides the basic framework of any viewer that can be used with the MACS system.
<i>Attribute</i>	This class represents a single attribute of an agent within the MACS system and is partly responsible for the flexibility of the system.
<i>AttributeList</i>	This class is an extension to the Vector class that overrides a number of methods so that it is more suited to being used within the MACS application to store collections of the Attribute class.
<i>BaseEntity</i>	This class represents the most basic entity in a Macs simulation and must be extended correctly by any user written class wishing to be used within the Macs system.

Class Name	Role & Responsibilities
<i>BaseEntityList</i>	This class is an extension to the Vector class that overrides a number of methods so that it is more suited to being used within the MACS application to store collections of the BaseEntity class.
<i>BinManager</i>	This class is spatial data structure used to store the dynamically changing positions of each agent within a simulation
<i>CanvasFrame</i>	This class represents the main 2D, top-down, view of the environment in its static or initial state
<i>ContextMenu</i>	This class represents a context menu for the CanvasFrame class and is used to initiate the distribution of a group of agents within that class and to delete agents that are no longer required in the simulation.
<i>DistributionFrame</i>	This class provides a simple GUI that allows a user to specify the details of a “many agent” drag & drop process.
<i>Engine</i>	This class is the core of the simulation engine and is responsible for synchronising the movement and actions of agents within the system.
<i>EntityEditorFrame</i>	This class provides a simple Java source code editor that allows a user to extend the basic agent code created with the SpeciesWizard class and adapt it to their own needs.
<i>Environment</i>	This class holds all information regarding the environment used with a Macs simulation such as the x,y and z dimension limits and the details of any agents that have been positioned within it.
<i>ErrorReporter</i>	This class is responsible for generating and displaying all necessary error messages that may be needed during application execution.
<i>FileIO</i>	This class implements all File input/output methods that the Macs application uses.
<i>FileIO.GIFFileFilter</i>	This class is a simple filter used when the user is searching for a GIF image to represent an agent they are creating
<i>FileIO.ScenarioFileFilter</i>	This class is a simple filter used when the user is searching for a MACS Scenario file they wish to load or save via the application.
<i>FileIO.SpeciesClassFileFilter</i>	This class is a simple filter used when the user is searching for a compiled Java agent they wish to import into the application.
<i>FileIO.SpeciesSourceFileFilter</i>	This class is a simple filter used when the user is searching for a Java source code file for an agent they wish to load or save via the application.
<i>FileIO.XMLFileFilter</i>	This class is a simple filter used when the user is about to run a file based simulation and wishes to specify a location to save the MACS Simulation file that will be generated.
<i>FrameMenuBar</i>	This class is responsible for building and displaying the main menu bar of the Macs application.
<i>MacSim</i>	This is the main class of the MACS system and is responsible for creating and initialising the whole application and its various component classes.
<i>MacsPreferences</i>	This class provides a graphical representation of all underlying current preferences set by the user and can be used to change those preferences at any time the program is executing.

Class Name	Role & Responsibilities
<i>PaletteFrame</i>	This class represents a display of all BaseEntity subtypes that are available for the user to use within the Macs application.
<i>ProgressDialog</i>	This class provides a simple progress bar for the application and is used to indicate the progress of a file based simulation.
RunMacs	This is a simple class that can be used to run the main application class.
ScenarioFileInfo	This class is used to transfer data from a ScenarioFileReader to the application.
ScenarioFileReader	This class is used to read a Scenario file and to get an object that contains all the data it contains in a form that is usable within the application.
ScenarioFileWriter	This class is used to write all relevant data to a Scenario file.
SpeciesWizard	This class provides a simple GUI that the user can use to specify the basic elements of a new agent species.
TimeControlFrame	This class constructs a simple “media player” style set of controls from which the user can control the “playback” of any real-time simulation run performed with the system.
Viewer2D	This is an example class demonstrating the ability of the MACS AbstractViewer class to produce real-time 2D visualisations
ViewerFile	This is an example class demonstrating the ability of the MACS AbstractViewer class to produce deferred-time file based simulations.

5.3 Functional Requirements Implementation

The following points discuss how the implemented application successfully achieves each of the functional requirements outlined in section 4.3 and the numbering used coincides with each requirement proposed in that section.

1. Through the use of the SpeciesWizard class the user is able to create a template for an agent class based upon the BaseEntity class (ensuring it conforms to a desired standard) and to then adapt and extend the template to form an agent that meets their own requirements using a built-in Java source code editor. After the design of an agent is complete the source code can then be compiled to form a valid Java class file, again from within the application.
2. A user is able to import any agent that properly extends the BaseEntity class (i.e. has been formed using the above process) into the PaletteFrame class of the application and from there to integrate the class within their own MACS scenarios via a drag & drop style user interface.
3. Using the PreferenceFrame class and underlying Preferences class a user is able to specify the configuration of the MACS application and the parameters of the simulation engine.
4. A User is able to open a valid XML format, scenario definition file containing the initial, static, information required to initialise a particular scenario. The process of opening a scenario file is divided into two parts. The first utilises the *parse()* method of the ScenarioFileReader class to construct a ScenarioFileInfo object that contains a set of data structures that duplicate the information found in the original XML file (the format of which is discussed in section 4.4.2) but that can be used directly by other classes within the application.

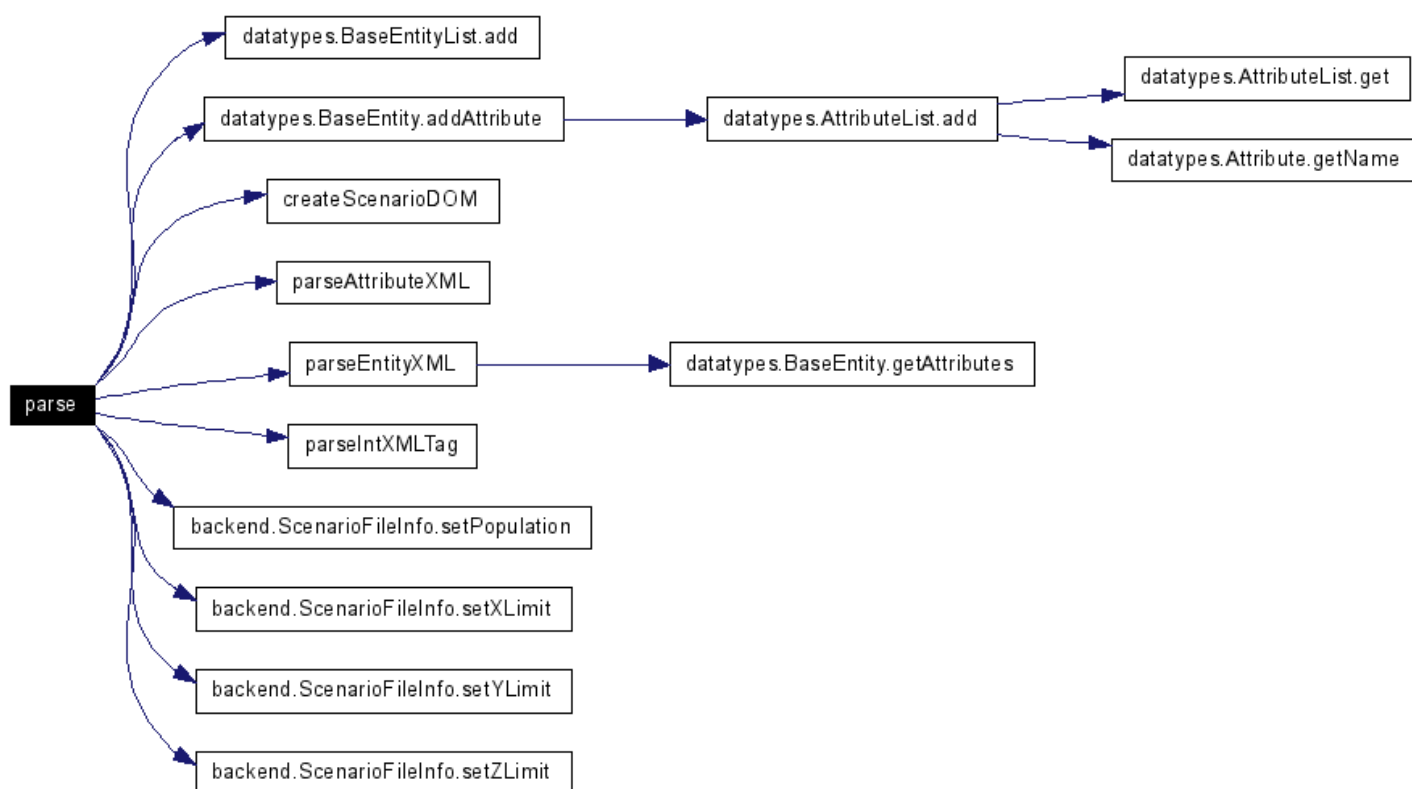


Figure 11 - The Scenario File Reading Process - Stage I

The second stage of the process is concerned with taking the ScenarioFileInfo object and transferring all the elements it contains to the various parts of the application that will store the data while the application is executing. This stage is summarised in the following figure.

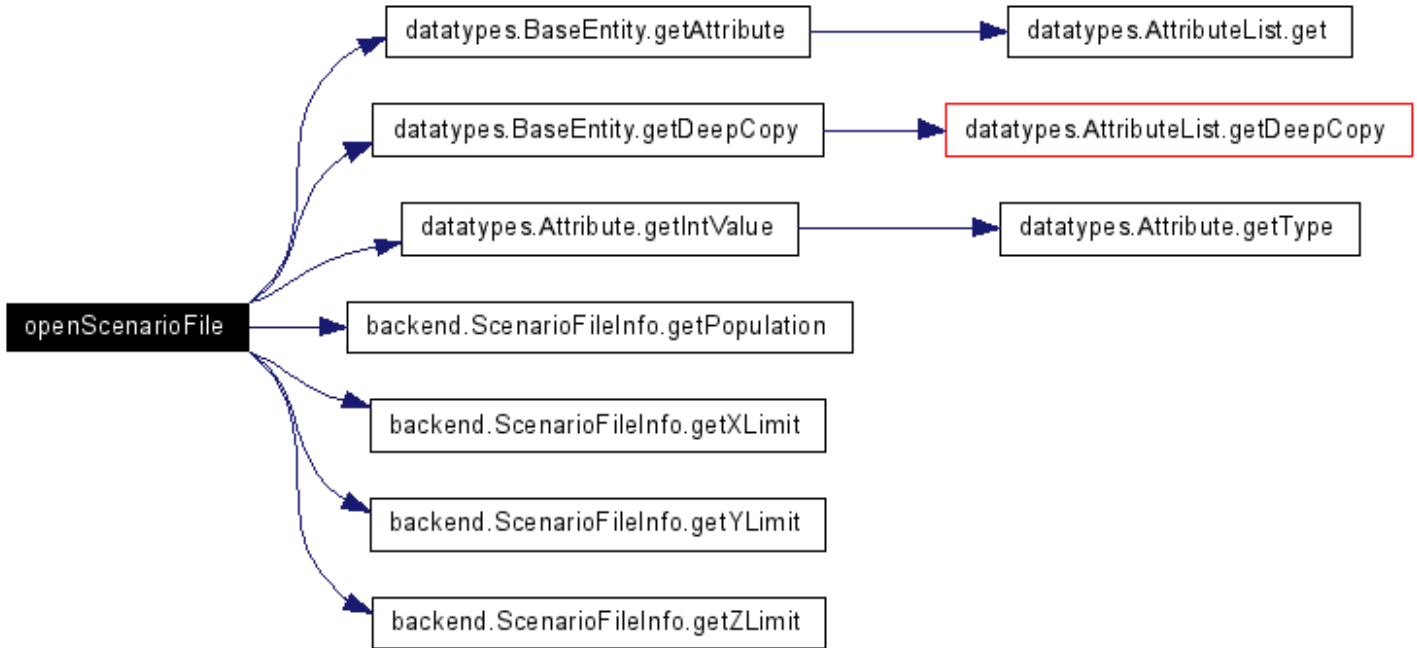


Figure 12 - The Scenario File Reading Process – Stage II

Once a new scenario has been created, or an old one modified, all aspects of it can be saved to an XML file in persistent storage for later use. In contrast to the file opening procedure discussed previously this is done in only one stage and utilises a ScenarioFileWriter object to write the data directly from the application to the XML file. The mechanism for accomplishing this is outlined in the next figure – although in reality the process is somewhat more complex.

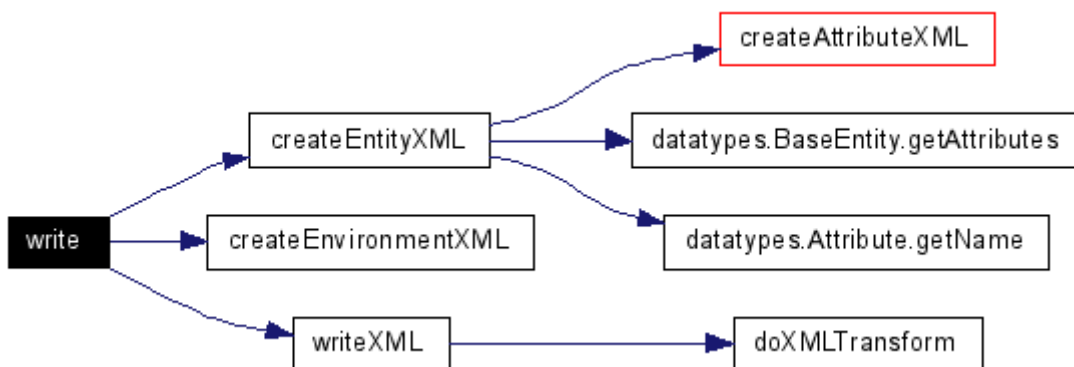


Figure 13 - The Scenario File Writing Process

5. The use of the PreferenceFrame class allows a user to specify all system settings and any available environmental parameters.

6. The user is able, via the drag & drop interface mentioned, to add single and groups of agents to the initial environment set-up and to reposition or remove them in order to create the form of agent scenario they require. The manner in which the user is able to add a single agent to the environment is shown in the next diagram. To achieve a drop of more than one agent the user is required to select an area of the CanvasFrame, specify, via a dialogue box, the number of agents required and the type of distribution that should be used when adding them to the selected area – the process illustrated below is then simply called multiple times. At the current time the user is able to choose between random and regular (organised in rank & file) distribution of agents over the selected area.

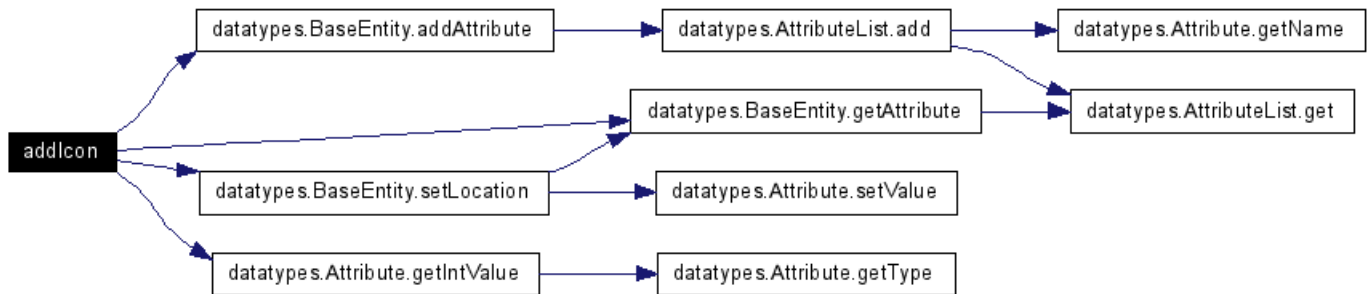


Figure 14 - Adding a New Agent

7. The simulation engine is able to perform both real-time and deferred time simulations and utilises the AbstractViewer class to allow users to create other forms of output that can be tailored to their own requirements. The Engine class runs as a thread, concurrently, alongside any any class that extends AbstractViewer, which acts as an “observer” of the engine, and passing all information about the current state of the simulation to the viewer via the *deliverCurrentData()* method.

Currently there are two classes that extend the AbstractViewer class. The ViewerFile class demonstrates the ability of the application to produce accurate deferred-time simulations. The Viewer2D class complements this by producing real-time animations based upon the the provided by the Engine class. The following figure illustrates the performance of the ViewerFile class and the methods involved in obtaining the data from the Engine class via the *deliverCurrentData()* method and writing it to an XML file.

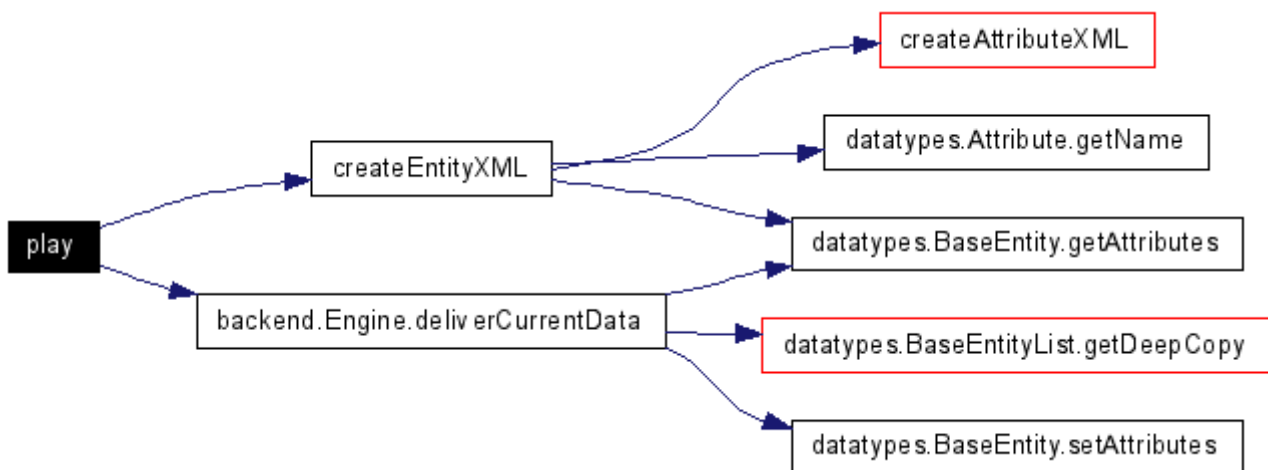


Illustration 1 - The Operation of The ViewerFile Class

8. The user is able to control both the temporal aspect (via the TimeControlFrame class) and the spatial aspect (via commonly encountered scroll bars) of a real-time simulation.

5.4 System Operation

This section of the report provides a graphical demonstration of how the final implemented version of the MACS application looks when being used. It is designed as a set of storyboards that illustrate certain aspects of the system.

5.4.1 Initial System Start-up

This is the first view the user gets of the MACS application. A user is initially provided with a “preferences” dialogue box through which various aspects of the system and simulation engine can be adjusted. The dialogue box is modal, meaning the user cannot access other parts of the application until they have either confirmed or cancelled the shown preferences, and this ensures that necessary system settings exist before other elements of the application begin to operate.

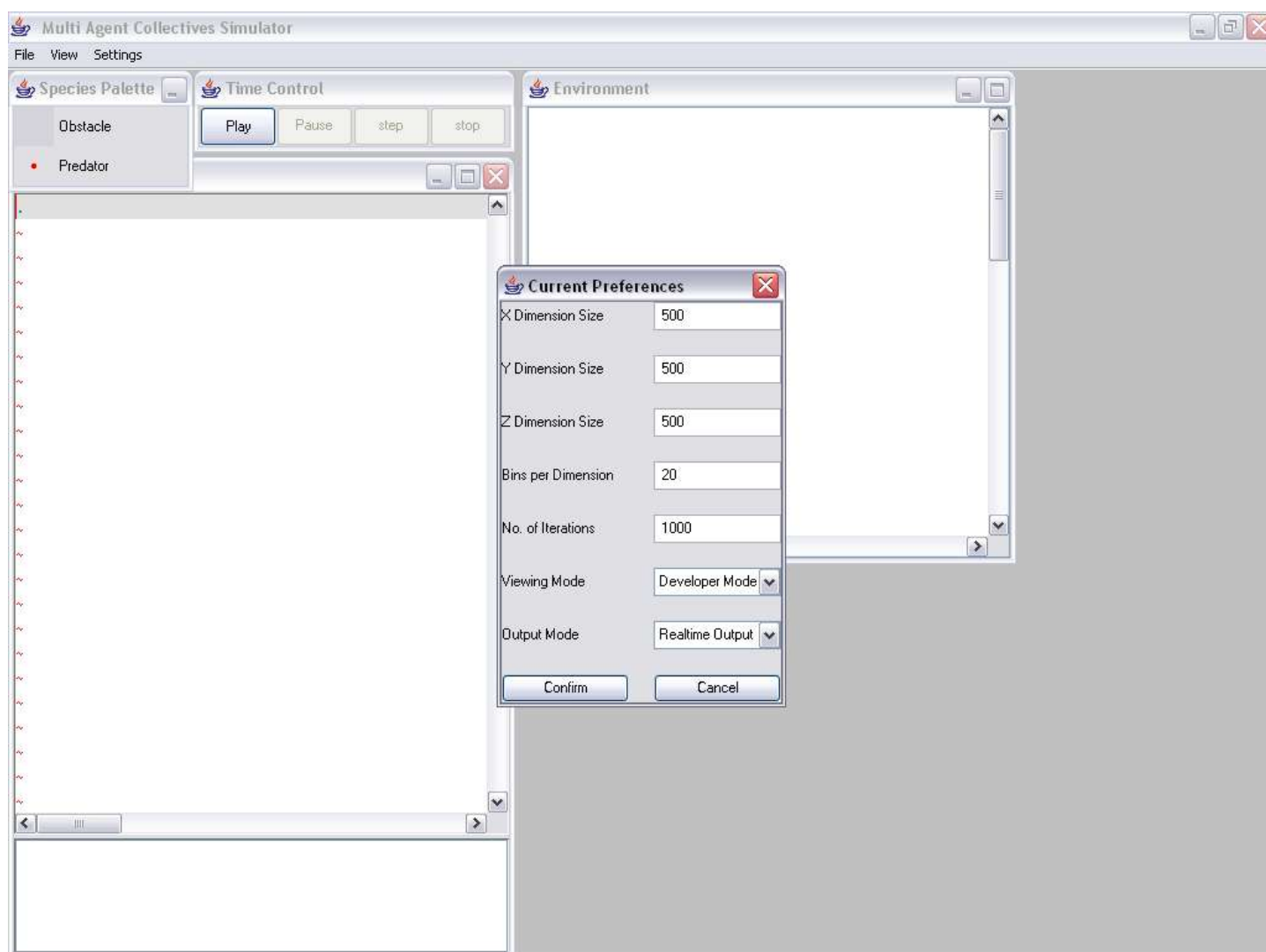


Illustration 2 - Initial Start-up View

5.4.2 Drag & Drop Interface

Here the main elements of the MACS environment can be seen clearly. The *species palette* in the top-left corner of the screen contains all the entities that are currently loaded into the application. From here the entities can be dragged over to the *environment window* and dropped into the current scenario – in this example the user has already dropped five predator entities and two obstacles. Also visible is the *time control* that will later be used to initiate the start of a simulation.

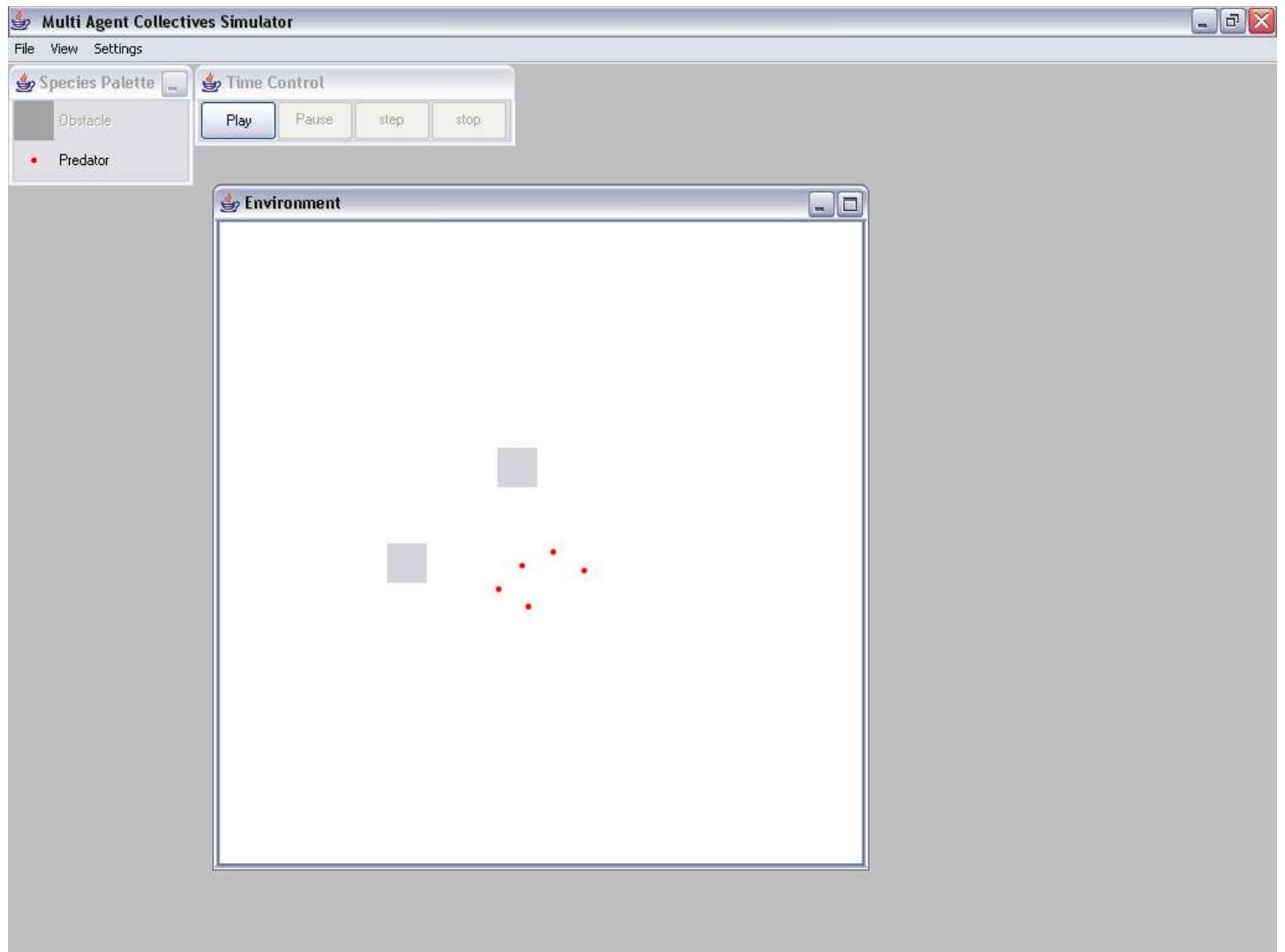


Illustration 3 - Dragging & Dropping Agents

5.4.3 Importing New Species

In this frame the user has selected to import a new species into the application. They have been presented with a file-open dialogue box through which they can select a Java class file to import. The dialogue box has been enabled with a filter so that it only displays files that have a matching .class extension. Here the user has decided to import a prey species.

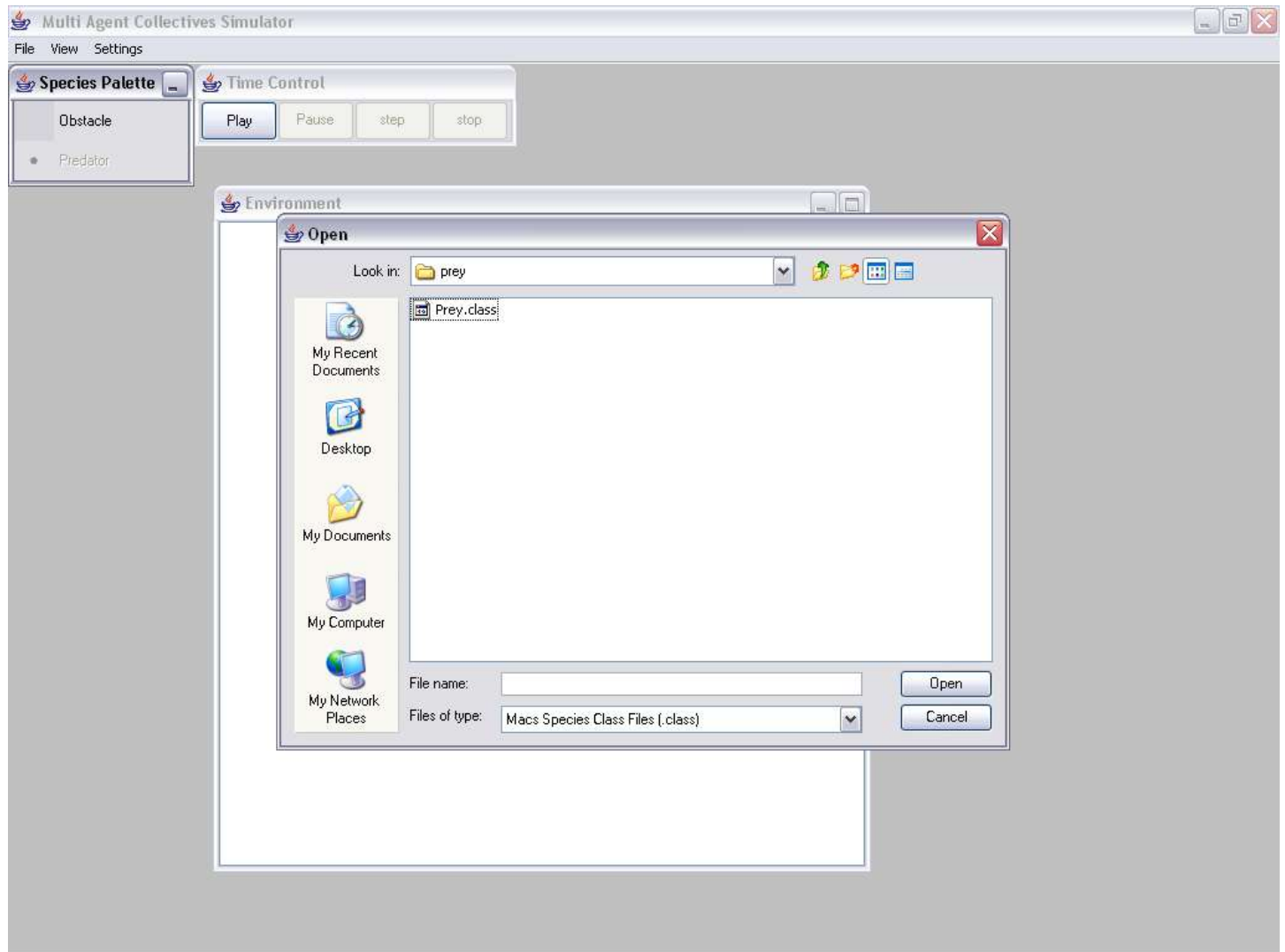


Illustration 4 - Importing a New Species

5.4.4 Successful Importation

After confirming they wish to import the new prey species the species is added to the *species palette* and is then available to be used in the same way as those present when the application first started. In this frame the new prey species can be clearly seen in the *species palette* and the user has already added five of these agents to the current scenario.

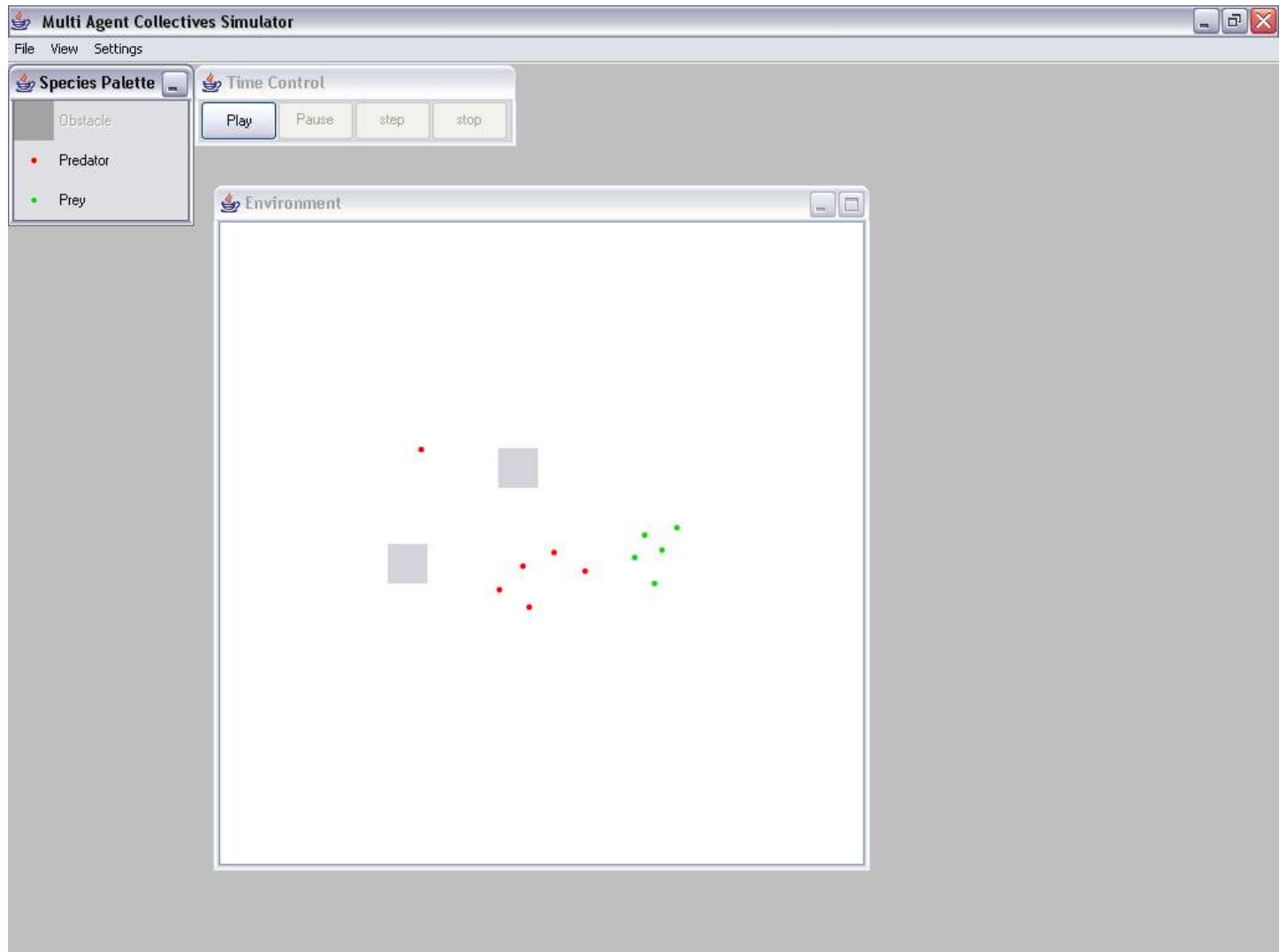


Illustration 5 - New Species Imported

5.4.4 Creating a New Species

Now the user has chosen to create a new species themselves and so has launched the *species wizard*. Using this interface they have entered the name of the new species, which by necessity follows standard Java class naming conventions, and also selected a GIF format image that will be used to represent members of the new species.

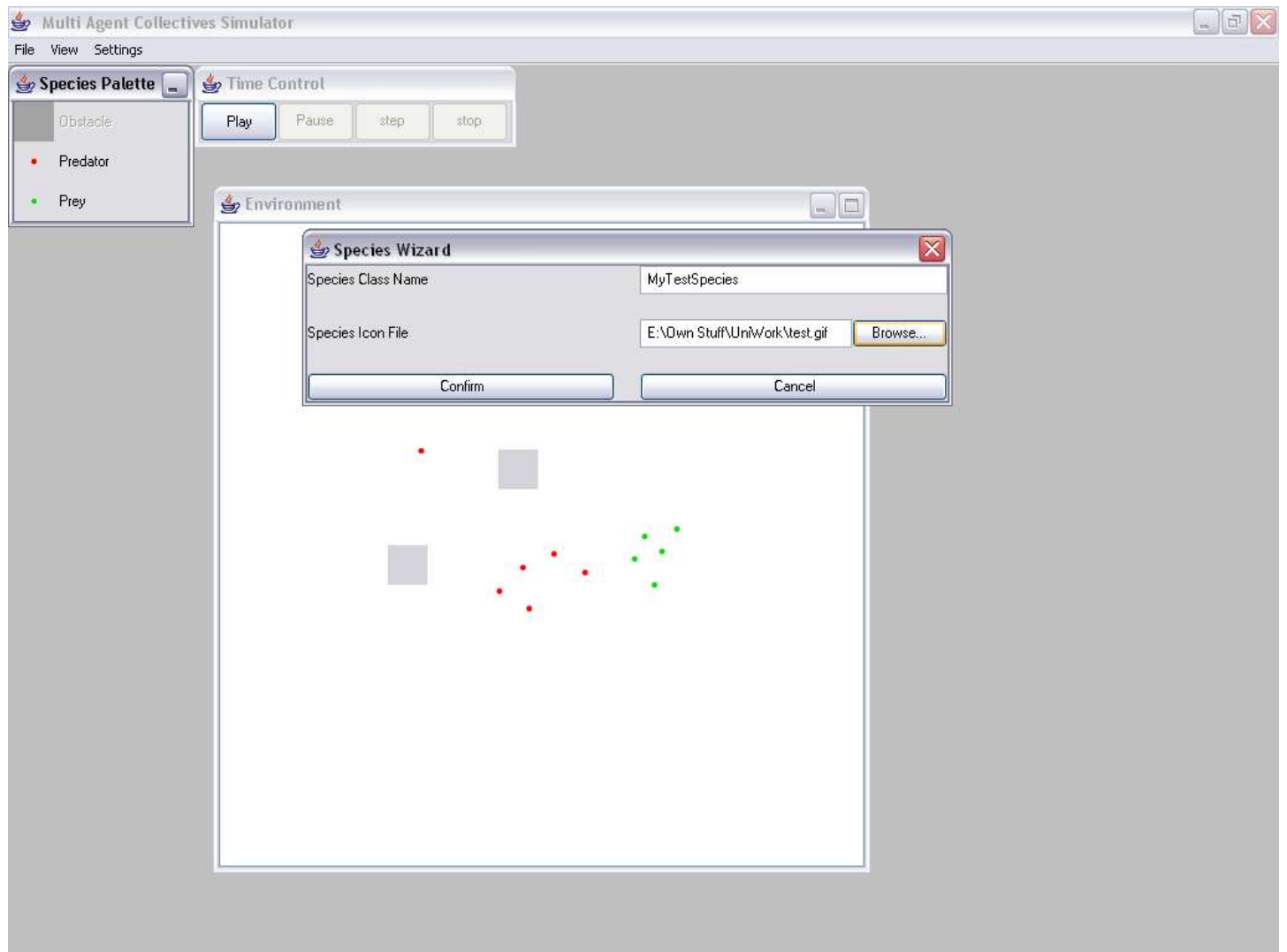
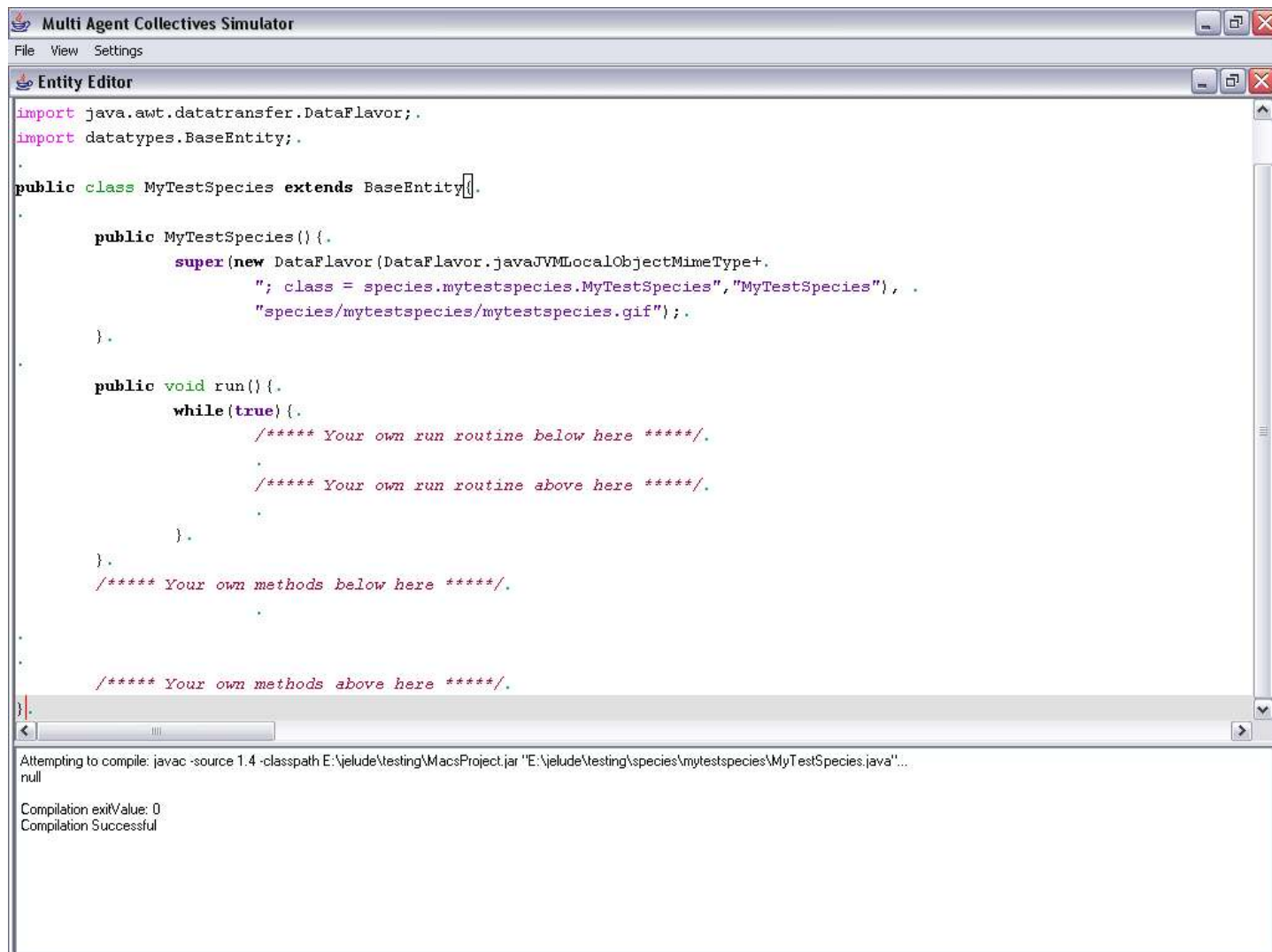


Illustration 6 - Using the Species Wizard

5.4.6 Species Compilation

This frame shows that the user has chosen to compile the species they have just created. This results of the compilation are visible in the lower portion of the *entity editor window* and, in this instance, indicate that the class was successfully compiled. From this point on the species can be imported into the *species palette* at any time but will not do anything when added to a scenario as it does not yet possess any real functionality.



```
Multi Agent Collectives Simulator
File View Settings

Entity Editor

import java.awt.datatransfer.DataFlavor;.
import datatypes.BaseEntity;.
.
public class MyTestSpecies extends BaseEntity{
.
    public MyTestSpecies() {
        super(new DataFlavor(DataFlavor.javaJVMLocalObjectMimeType+.
            "; class = species.mytestspecies.MyTestSpecies", "MyTestSpecies"), .
            "species/mytestspecies/mytestspecies.gif"); .
    }
.
    public void run() {
        while(true) {
            /***** Your own run routine below here *****/.
            .
            /***** Your own run routine above here *****/.
            .
        }
    }
.
    /***** Your own methods below here *****/.
    .
.
    /***** Your own methods above here *****/.
}
.

Attempting to compile: javac -source 1.4 -classpath E:\jelude\testing\MacsProject.jar "E:\jelude\testing\species\mytestspecies\MyTestSpecies.java"...
null
Compilation exitValue: 0
Compilation Successful
```

Illustration 8 - Successful Compilation

5.5 Implementation Issues & Solutions

As the design of the application progressed through the initial design and implementation phases a number of issues became apparent that needed to be considered and resolved. These issues are now discussed to provide some insight into how the design of the system evolved over time and resulted in the final version.

5.5.1 The Use of Multi-Threading

Initially it was intended that each simulation would run on a “simulation engine” (responsible for coordinating the many agents within the environment) and that this would then pass on data to a “viewer” (responsible for rendering the data to a file or the display). Both of these, *and each agent* (responsible for its own update routine) would exist as an individual Java Thread.

During the implementation of the system however it was decided that, as the simulation was required to possess the ability to be “stepped” through (in real-time mode), it would be pointless to provide independently running threads for each agent as these would need to be “synchronised” (in time) after each iteration of a simulation – so as to provide an accurate, single iteration, stepping mechanism. In doing this the advantage of using threads would quickly be lost.

As a result of this decision the model finally implemented retained individual threads for the simulation engine and the viewer but each agent simply possesses a run() method, that, for each agent in a simulation, is triggered once per simulation iteration by the simulation engine.

5.5.2 Syntax Highlighting

At the outset of the design process it was intended that part of the application would be devoted to the design of agent species. The definition for each agent was to be written in the Java language as this would allow any user that had some familiarity with the language to begin to define their own agents. To enable the user to accomplish this from within the application a Java code editing window was created that initially resembled a simple text editor. Although this would have been perfectly acceptable in terms of the functionality it possessed it was decided that editing Java code in a plain text interface could be confusing to the user.

To avoid this from happening it was decided that this part of the program would be replaced by a customised editor more suited to editing Java source code. After researching a number of available packages it was decided that the JEdit Syntax Highlighter package[4] seemed to possess the required functionality, with scope for further additional functionality if necessary, and was provided under a licensing agreement that allowed the source code to be used, modified and redistributed for any purpose – thus making it ideally suited for inclusion in the application.

The addition of the JEdit package provided the application with a Java editor that had built-in support for syntax highlighting of Java source code and this addition would greatly improve the readability of any code written within the application.

5.5.3 Automated Windows Installation

From the outset the application was intended to be used by experienced and inexperienced users alike. One aspect that was of concern was the deployment phase of the software engineering process and in particular how the system would be provided to, and installed by, the user.

In the experience of the author many Java source code packages / programs can be difficult and/or confusing to initially install on a PC and it was considered that this could result in many inexperienced users avoiding using the system. As a result of this much time was devoted to researching how this process could be made as simple and pain-free as possible.

The Jelude script[5] for the Nullsoft Scriptable Install System[8] provided the basis for developing a customised installation system that would allow users of the Microsoft Windows operating system an easy method of installing the program on a computer. Although the installer is not available to users of *NIX based operating systems these same users are generally considered more experienced than those using the Windows operating system and, due to the nature of those operating systems, more familiar with the manual installation of source code packages.

The NSIS installer for the MACS system provides a “double-click” executable file that performs the following installation routine: -

1. Conducts a check for the presence of a JAVA SDK – needed for the construction of the users own agents.
2. If a Java SDK is not found in step 1 the user is asked if they wish to download the necessary SDK from the Sun website.
3. If the user wishes to download the SDK a web browser is launched that automatically goes to the Sun download site. If not then the installer exits without installing anything on the target system.
4. Once a Java SDK is present on the target system the installer unpacks a JAR file containing all MACS application .class files to the target system.
5. The installer unpacks 3 working directories to the host system that are required for the MACS application to function correctly, these are: -
 - The *species* directory – This directory contains a set of subdirectories each one of which corresponds to a particular agent type and contains the Java source code, compiled JVM code and a GIF file representing each agent type. Any agent designed within the MACS application will be stored within this directory.
 - The *scenarios* directory – This directory is used by the MACS application to save any Scenario created by the user.
 - The *sims* directory – This directory is used by the MACS application to save any simulation-run that is written to file rather than performed in real-time.

From this point on the MACS application is now fully installed on the target system and can be activated simply by double-clicking on the executable installer just as though it was a standard executable program. The process could be made even easier if a “shortcut” is made on the desktop of the target system that was directed to the installer program.

5.6 Testing

Initially it was planned that much of the testing regime would be done using automated Junit tests (Java coded testing schemes designed specifically for automated testing). In actual fact as the design of the program evolved it became increasingly obvious that, due to the nature of the system, that degree of test automation would not be possible. This was mainly due to the large amount of coupling between a number of the main classes (a high coupling between two classes indicates that one class is dependent on data from the other class). To maintain the ability to locate and eradicate bugs in the system a great deal of emphasis was devoted to testing each piece of code as it was being written. In doing this many bugs were eradicated quite early in the development process and those that remained for extended periods of time were usually due to unexpected effects when dealing with little known Java SDK classes rather than within procedural code written by the author. Some, small, bugs do remain in the code as it is submitted but these are related to the Swing user interface and not with the simulation engine/process itself.

Consideration was also given to the fact that a simulation system such as this (i.e. one concerned with real-time simulation) should be as efficient, within reason, as possible. To keep track of the performance of different elements of the application regular “profiling” of the system was performed during which each section of code was analysed to determine if changes could be made that would optimise the applications overall efficiency.

The following table provides a section of profile results – the whole file is extensive and covers every aspect of the application. From this it is quite easy to see that the class responsible for most of the time taken by the program is the BaseEntity class, and particularly its *lookAround()* method – which makes sense as this method is responsible for locating all agents within another agents immediate environment. Using this form of profile it is possible to identify sections of code that could be adapted to yield better performance from the system.

Name	Calls	%	Time	%	Time/Call	Total time	Inst. time
BaseEntity	5763415	6.21	20238	5.60	0.003511	315371	78
lookAround	110000	0.12	424	0.12	0.003855	120300	78
getAttribute	4540902	4.89	6256	1.73	0.001378	119304	78
getDeepCopy	110200	0.12	7838	2.17	0.071125	29586	78
limitLocation	110200	0.12	204	0.06	0.001851	19515	78
limitAttribute	330600	0.36	715	0.20	0.002163	19311	78
<init>	110302	0.12	4381	1.21	0.039718	4413	78
addAttribute	221011	0.24	343	0.09	0.001552	2865	78
getAttributes	120000	0.13	47	0.01	0.000392	47	78
setAttributes	110000	0.12	30	0.01	0.000273	30	78
run	0	0.00	0	0.00	0.000000	0	78
setLocation	0	0.00	0	0.00	0.000000	0	78
getTransferData	0	0.00	0	0.00	0.000000	0	78
setBin	200	0.00	0	0.00	0.000000	0	78
isDataFlavorSupported	0	0.00	0	0.00	0.000000	0	78
printAttributes	0	0.00	0	0.00	0.000000	0	78
getTransferDataFlavors	0	0.00	0	0.00	0.000000	0	78
AbstractPlayable	9	0.00	35573	9.84	3952.555556	296030	15

6. Conclusions & Further Work

In terms of the requirements proposed initially the MACS system accomplishes many of them completely as can be seen from the discussion in section 5.3, however, in performing usability testing on the application a number of areas came to light that indicate that further work is possible, and indeed, desirable.

6.1 User Feedback Mechanisms

Currently the amount of feedback and 'tips' given by the system to the user (as to current state, processing and instruction on usage) is quite minimal. As the system was designed with both expert and novice user levels in mind a good addition would be to increase the amount of user feedback that the system gives. A good example of this is the need for an agent “debugger” that would allow the designer of an agent to print out debugging messages from within their agent so that they could more easily diagnose errors or unexpected results.

6.2 Viewer Wizard

Initially it was intended that the user would only have control over the importation of agents into the MACS architecture but as the design progressed it became clear that a similar mechanism for the design of custom viewer types would also be a great addition to the application. In this way a user could specify the precise nature of the viewer and could adapt it to their own specifications.

6.3 Code Efficiency

As the system currently stands the code has not in any way been optimised. Since optimisation, at the design level) can drastically reduce the maintainability and readability of source code it was not felt appropriate and was avoided primarily for reasons of comprehension. This type of real-time system must be made to be as efficient as possible so as to reduce the total amount of system resources used. Performing a number of optimisations in certain areas of the code would, I feel, make a significant improvement to the performance of the application.

6.4 Redundant Code Elimination

A number of the classes in the application as it stands could, at least in its current form, be safely removed from the system without much impact. The *Environment* class is a good example of this. Currently it merely embodies an abstract space that's only attributes are its maximum spatial extents and a list of all the entities currently placed within it. These alone can be located in other classes in the application and do not warrant a separate class. If however the class was enhanced to have more attributes it may well be worthy of its own class, factors such as this only develop over time as the application is used and the needs of the users are adapted to.

Overall I feel the application is a good approximation to what I set out to achieve although with some slight modifications. I do not feel that it is yet suitable for public use but with a little more refinement of the current design this may be possible. The standard of design is reasonable and coding style good with emphasis made to maintain a certain standard of clarity – even if this is initially at the cost of performance.

7. References & Bibliography

- [1] British Computing Society, *BCS Code of Conduct*, September 2001, Available at <http://www.bcs.org/NR/ronlyres/427E7CA1-87D9-4C34-BF8A-16F70A48CB4D/0/conduct.pdf>
- [2] British Computing Society, *BCS Code of Practice*, September 2004, Available at <http://www.bcs.org/NR/ronlyres/783C9D32-925A-4B2E-A05D-695157DD7244/0/cop.pdf>
- [3] Fagerlund, M., MatFa's Boids., 1997., Available at <http://delphi.icm.edu.pl/ftp/d20free/mfboid3s.zip>
- [4] The JEdit Syntax Highlighter. Available at <http://syntax.jedit.org/>
- [5] The Jelude NSIS Installer Script. Available at <http://www.sfu.ca/~tyuen/jelude/>
- [6] Klein, J., *Breve : A 3D Environment for the Simulation of Decentralized Systems and Artificial Life*, Artificial Life VIII – Proceedings of the 8th International Conference on Artificial Life., 2003. Application available at http://www.spiderland.org/breve/download/breve_windows_2.0.zip
- [7] von Neumann, J., *The General and Logical Theory of Automata*, J. von Neumann. "Collected Works" (ed. A.H. Taub), Vol. 5, p.288
- [8] The Nullsoft Scriptable Installer System. Available at <http://nsis.sourceforge.net/home/>
- [9] Reeves, W., *Particle Systems-A Technique for Modeling a Class of Fuzzy Objects*, ACM Transactions on Graphics, V2 #2, April 1983. and reprinted in Computer Graphics. V17 #3, July 1983, (acm SIGGRAPH '83 Proceedings), pp. 359-376.
- [10] Reynolds, C.W., *Flocks, Herds, and Schools: A Distributed Behavioral Model.*, Computer Graphics, 21(4), July 1987, pp. 25-34.
- [11] Reynolds, C.W., *Steering Behaviors For Autonomous Characters*, Proceedings of Game Developers Conference 1999, pp. 763-782. Available at <http://www.red3d.com/cwr/steergdc99/>
- [12] Samet H., *Applications of Spatial Data Structures*. 1990., Addison-Wesley Publishing.
- [13] Sun Microsystems, *Code Conventions for the Java Programming Language*, April 1999, Available at <http://java.sun.com/docs/codeconv/CodeConventions.pdf>
- [14] The Swarm System. Available at <ftp://ftp.swarm.org/pub/swarm/binaries/w32/swarm-2.1.1.exe>
- [15] Wolfram, S., *A New Kind of Science.*, 2002., Wolfram Media, Inc. Available at <http://www.wolframscience.com/nksonline/toc.html>
- [16] Wooldridge, M. and Jennings, N. R., *Intelligent Agents: Theory and Practice.*, Knowledge Engineering Review 10(2), 1995. Available at <http://www.csc.liv.ac.uk/~mjw/pubs/ker95.pdf>

Appendix I – Project Log

The following is a copy of the log kept to document my work on the project.

Term 1

Pre-Proposal Meetings

6-10-2004 Discussed my initial ideas for the project and the areas in which the application may have relevance. I also agreed to consider the use of 3D representation using a pre-built graphics engine such as the Quake Engine.

15-10-2004 Submitted a draft of my initial proposal, discussed the related reading material and my reasons for including it. Agreed to start construction of use case scenarios and begin basic system design.

Week 3

Wrote up and submitted my final proposal document. Continued with the low-level design and the requirements analysis. Created use-case and class diagrams for a number of the major classes although these do not accurately show the functionality of the classes they depict they are a initial indication as to my thinking. The GUI has not been included in the class diagram since it is comprised of many different classes itself and this would only bloat the design at this stage. Began researching the various standards that exist for agent-agent communication (KIF, OMG etc.) and also started to consider how sensory perception is to be handled by the application.

21-10-04 Meeting with Sharon Wood – Submitted initial proposal.

Week 4

Started the coding of the GUI itself. Doing this gives me an indication of how the user will use the system and how its different elements will interact. Although not usually advisable I have only started to implement parts of the GUI that will remain largely unchanged as the system progresses in construction.

29-10-04 Meeting with Sharon Wood – Discussed my system and requirements analysis so far.

Week 5

Very little progress was made this week due to assignments being due for submission. Did some further reading on Threads, Java swing components and more GUI coding

No meeting with Sharon Wood due to work commitments

Week 6

Continued work on the coding and requirements analysis. Again work hampered by other work commitments.

12-11-04 Meeting with Sharon Wood – Discussed possible avenues for research into similar agent frameworks and academic studies into agent architectures

Week 7

Located a number of academic papers in Adobe PDF format that may be of use – but have yet to read them all. Two will definitely be usable since they discuss the various types of agent architecture and an, as yet unheard of, system known as “breve” mentioned in AI-VIII that does many of the tasks I wish my own application to achieve, only in 3D – the difference being that “breve” uses its own language (called “Steve”) that is similar to objective-C whereas my application will use Java exclusively.

Spent most of the week trying to solve a major bug in the drag and drop interface that was preventing it from working as intended. The switch in the code from using pre-set classes to using user-definable classes seemed to be the cause. The symptoms being that when an instance of the desired class was dropped onto the Canvas it would “forget” its own type and resort to using the attributes of the BaseEntity class. The problem has now been solved – although not elegantly – by “force-feeding” the parameters to the BaseEntity class via its constructor. Although the end result is the same it means that the constructor call in a derived class is now quite long – but – on the upside means that I only have to disable one line in the derived class’s constructor.

Week 8

Having looked into the KIF (knowledge Interchange Format) I’m considering whether it would be wise to use it. On the surface it seems like a good idea but it may require the inclusion of a Java KIF parser (available on-line) to avoid the user having to directly code a parser in any agent that uses KIF. It maybe slight overkill for the needs of this project. I am also considering how to achieve the functionality I require from the agent code editing window in the GUI. I need it to be able to disable individual lines of application generated code so that the user is unable to edit them (and thus create an agent that functions wrongly). Have conducted refactoring on the code written in iteration 1 (shallow model) and produced full “Java-style” documentation for it. Although not accomplishing all the coding I wished to I have managed to solve many of the issues that may have made a significant impact on the development of the system if not addressed at this stage.

26-11-04 Meeting with Sharon Wood – Discussed my plans for writing and submitting the interim report in 2 weeks time.

Week 9

Concentrated on making progress on the interim report on the project. I feel that creating decent UML diagrams, especially of the operation of the program may prove difficult. Creating such diagrams by hand seems not to be much of an option due tot he complexity involved and would take up a lot of time that would be better spent elsewhere. Am continuing to look into software that will allow me to automatically generate diagrams but so far i have not been able to find the “right” thing. Many of the available programs to do this are either only commercially available – and have locked features when used in demonstration mode – or do not produce the correct type of diagrams with the right level of complexity.

Week 10

Completed and handed in the interim report on time.

Term 2

Week 1

Have spent a lot of time ironing out a number of bugs in the user interface that may have caused problems if not addressed at this time. Due to a computer failure over the Christmas break i have had to begin to reassemble my suite of application tools and have opted to change the IDE i use to create the application. Although the change may effect my productivity initially the problems i am encountering with the Netbeans IDE are making code development intolerable so i am switching to Eclipse as it seems to perform better in many respects.

Week 2

I have spent most of the time this week making changes to the “glue” classes. The initial code i had written, although perfectly functioning, was untidy and used a number of mechanisms i was not entirely satisfied with. The new version is now a lot tidier and uniform in nature. In doing this i have been able to remove a number of classes that were not really necessary and this has made the code somewhat easier to deal with.

Week 3

As my PC has been behaving rather erratically over quite a long period of time i decided to change the motherboard. Unfortunately this has “upset” Windows to the extent that i have been reduced to the same situation i mentioned in week 1. Have spent a large portion of my time reinstalling the applications necessary to continue application development.

Week 4

Implemented the EntityEditorFrame class. This is to be the main window that a developer can use to write the Java code that will make up an agent. The framework of the code will be generated by a “wizard” and inserted into this frame the code where it will then be adaptable by the user to meet their own requirements.

Implemented the TimeControlFrame class that will allow the user to control the playback of the simulation run when in real-time mode with well known “media player” style controls.

Week 5

Started working on the Attribute class. It has occurred to me that this class may cause some problems as it is required to hold a number of different basic data types and must also handle converting between these data types in a sensible manner. I investigated using Generic types that have become available in Java 1.5 but these don't seem to quite fit the purpose.

Came across the Preferences class within the Java API and have decided to use it within my application. In this way i will be able to store and recall user preferences to and from persistent memory in a very easy way between program runs. The PreferenceFrame class provides both an easy mechanism for accessing the underlying preferences class and a GUI component that will eventually become the “preferences” window of the application.

Week 6

Implemented the DistributionFrame class that provides a graphical user interface to allow the user to indicate their preferences for distributing a number of agents over a predefined area of the environment.

Week 7

Implemented the SpeciesWizard class. This class provides the user interface for specifying the framework of a artificial agent that will then be enhanced within the EditorFrame class.

Found a small application that will allow me to create an installer program for the application. It is made up of a small script, named "Jelude", that is used with the NSIS (Nullsoft Scriptable Install System) program. This script could be used as the basis for my own script that would allow a Windows user to have a very simple "1 click install and run" mechanism and therefore hide much of the complexity of of setting up the program within an "exe wrapper".

Implemented the ErrorReporter class. This class attempts to avoid the code being scattered with almost identical method calls to show error dialogue boxes to the user. Although a similar problem still occurs each code fragment now references a single class that contains both the dialogue box creation method and a set of strings that can be used as error messages. This means that when trying to debug where an error has occurred in the file it is only necessary to go to one file rather than look through the entire code base.

Week 8

Contacted Guy McCusker regarding my problems with using a JEditorPane class for displaying agent Java code clearly. Received reply stating that he was unable to help so continued to search for a way of displaying the code in a clearer way than standard "black and white" text. I eventually discovered the JEdit Syntax Highlighter which, although it does not – in its current form – allow me to restrict the user from editing application generated code, does do a very good job of highlighting the Java code. I have started work on integrating it with my own code which does not appear that difficult.

Started work on the installer for the Windows version which will ensure the simplest possible installation process for the user and offers the possibility of other useful options such as the automatic detection and/or download of the required Java SDK and even to use the "File Association" mechanism present in the Windows operating system to register certain file types with the MACS system allowing those file types to be automatically opened by the application.

Week 9

Have made good progress on the simulation engine for the application and have finished work on the installer for Windows and the JEdit integration. The installer now extracts all necessary directories and searches for the Java SDK before starting the application.

Implemented a testing regime for the Attribute class. The class has passed perfectly but the testing has indicated one area that could be improved and that is how to interpret a string containing a boolean value. There appears to be more than one way it can be done so i will consider revising it if i feel it is really necessary.

Week 10

Have devoted all of my time on the project to creating the draft report to hand in at the end of the week. Integrating the diagrams is proving a little difficult as OpenOffice.org does not render them as clearly as i would have liked. Have opted for producing them in another application and then merging the two documents before handing in.

Appendix II – Source Code

```

package backend;

import java.util.Vector;

import common.MacsPreferences;

import datatypes.BaseEntity;
import datatypes.BaseEntityList;
import datatypes.Environment;

/**
 * This class is spatial data structure used to store the dynamically changing
 * positions of each agent within a simulation. It provides methods for the
 * efficient lookup of agents within the environment while a simulation is
 * being run on the simulation engine.
 *
 * @author Iain Robinson
 */
public class BinManager{
    /**
     * A 3 dimensional array of bins used to subdivide an environment into a
     * spatial data structure for efficient procesing while the simulation
     * engine is running.
     */
    private static BaseEntityList[][][] bins;

    /** The number of bins to divide each dimension into. */
    private static int nBins;

    /** The width of each bin in the X dimension. */
    private static double xBinWidth;

    /** The width of each bin in the Y dimension. */
    private static double yBinWidth;

    /** The width of each bin in the Z dimension. */
    private static double zBinWidth;

    /** The environment to be represented by the spatial data structure. */
    private static Environment environment;

    /**
     * A string used by a number of methods. Simply used to avoid repetition of
     * the same String literal.
     */
    private static String xPosition = "xPosition";

    /**
     * A string used by a number of methods. Simply used to avoid repetition of
     * the same String literal.
     */
    private static String yPosition = "yPosition";

    /**
     * A string used by a number of methods. Simply used to avoid repetition of
     * the same String literal.
     */
    private static String zPosition = "zPosition";

    /**
     * Creates a new instance of BinManager
     */
    private BinManager(){
    }

```

```

/**
 * This method is used to obtain a bin containing a given entity BASED ON
 * ITS CURRENT LOCATION. This does not guarantee that the entity will be
 * found within the list of contained entities that bin currently
 * possesses - but is used to check if the two match or if a reassignment
 * is needed.
 *
 * @param current The entity who's bin need to be found.
 *
 * @return The bin containing the given entity.
 */
static BaseEntityList getBinContaining(BaseEntity current){
    int xpos = current.getAttribute(xPosition).getIntValue();
    int ypos = current.getAttribute(yPosition).getIntValue();
    int zpos = current.getAttribute(zPosition).getIntValue();

    int xind = (int) (xpos / xBinWidth);
    int yind = (int) (ypos / yBinWidth);
    int zind = (int) (zpos / zBinWidth);

    //make sure the indices are within bounds
    int[] newcoord = getNormalisedBinIndex(xind, yind, zind);

    xind = newcoord[0];
    yind = newcoord[1];
    zind = newcoord[2];

    //adjust entity coords to stop this happening again
    current.limitLocation(environment.xLimit, environment.yLimit,
        environment.zLimit);

    //now return the correct bin needed after the re-adjustment
    return bins[xind][yind][zind];
}

/**
 * This method returns a list of all entities that surround a given entity
 * within a given range.
 *
 * @param entity The entity that is at the centre of the range circle.
 * @param range The range within which entities are to be considered
 *             "seeable".
 *
 * @return A list of all entities within the specified range of the entity.
 */
public static BaseEntityList getSurroundingEntities(BaseEntity entity,
    int range){
    BaseEntityList entities = new BaseEntityList(0, 1);

    int xLocation = entity.getAttribute(xPosition).getIntValue();
    int yLocation = entity.getAttribute(yPosition).getIntValue();
    int zLocation = entity.getAttribute(zPosition).getIntValue();

    // first get all bins that fall within the range to cut down
    // on the number of entities we need to query
    Vector binsInRange = getSurroundingBins(range, xLocation, yLocation,
        zLocation);

    //now ask each entity in each collected bin if it is within
    //range and add it to list if so
    for(int i = 0; i < binsInRange.size(); i++){
        BaseEntityList current = (BaseEntityList) binsInRange.get(i);

        for(int j = 0; j < current.size(); j++){
            BaseEntity currEntity = (BaseEntity) current.get(j);

```

```

        if(!currEntity.equals(entity)){
            int xCoord = currEntity.getAttribute(xPosition).getIntValue();
            int yCoord = currEntity.getAttribute(yPosition).getIntValue();
            int zCoord = currEntity.getAttribute(zPosition).getIntValue();

            boolean inRange = isEntityWithinRangeOf(xLocation,
                yLocation, zLocation, range, xCoord, yCoord, zCoord);

            //then the plane below
            inRange = inRange ||
                isEntityWithinRangeOf(xLocation, yLocation,
                    zLocation - environment.zLimit, range, xCoord,
                    yCoord, zCoord);

            //then the plane above
            inRange = inRange ||
                isEntityWithinRangeOf(xLocation, yLocation,
                    zLocation + environment.zLimit, range, xCoord,
                    yCoord, zCoord);

            if(inRange){
                entities.add(currEntity);
            }
        }
    }
}

return entities;
}

/**
 * This method is used to obtain all bins that lie wholly or partly within
 * a given range of a point.
 *
 * @param range The viewable range.
 * @param xCoord The location of the point in the X dimension.
 * @param yCoord The location of the point in the Y dimension.
 * @param zCoord The location of the point in the Z dimension.
 *
 * @return A List of all the bins that are within range of the given point.
 */
private static Vector getSurroundingBins(int range, int xCoord, int yCoord,
    int zCoord){
    Vector binsInRange = new Vector(0, 1);

    //we only need to examine those bins that fall within the
    //bounding box of the range circle - these get the max
    //and min extent of those bins in each dimension
    int minXBin = (int) ((xCoord - range) / xBinWidth);
    int maxXBin = (int) ((xCoord + range) / xBinWidth);

    int minYBin = (int) ((yCoord - range) / yBinWidth);
    int maxYBin = (int) ((yCoord + range) / yBinWidth);

    int minZBin = (int) ((zCoord - range) / xBinWidth);
    int maxZBin = (int) ((zCoord + range) / xBinWidth);

    int xind;
    int yind;
    int zind;

    for(int xIndex = minXBin;xIndex <= maxXBin;xIndex++){
        for(int yIndex = minYBin;yIndex <= maxYBin;yIndex++){
            for(int zIndex = minZBin;zIndex <= maxZBin;zIndex++){
                // first we check to see if the index we have falls
                // outside the area that has bins associated with it

```

```

        // if it does then we have to adjust the index and enforce
        //wraparound
        int[] point = getNormalisedBinIndex(xIndex, yIndex, zIndex);
        xind = point[0];
        yind = point[1];
        zind = point[2];

        // so this is now a valid bin in range 0 -> nbins-1 for each
dimension
        binsInRange.add(bins[xind][yind][zind]);

        // now we could cut down the no. of bins looked into
        // even further by finding only those that are within
        //the circle itself.
    }
}

return binsInRange;
}

/**
 * This method tests if the location of an entity is within range of
 * another given entity.
 *
 * @param point1x The first entity's position in the X dimension.
 * @param point1y The first entity's position in the Y dimension.
 * @param point1z The first entity's position in the Z dimension.
 * @param range The range that is visible to the first entity.
 * @param point2x The second entity's position in the X dimension.
 * @param point2y The second entity's position in the Y dimension.
 * @param point2z The second entity's position in the Z dimension.
 *
 * @return true if point 2 lies within range of point 1 - false otherwise.
 */
private static boolean isEntityWithinRangeOf(int point1x, int point1y,
int point1z, int range, int point2x, int point2y, int point2z){
    int xoffset = environment.xLimit;
    int yoffset = environment.yLimit;

    //first test against the observing entity in its original position
    boolean inRange = isWithinRange(point1x, point1y, point1z, range,
point2x, point2y, point2z);

    //now test against the observing entity in its 8 wrapped positions for this
plane
    inRange = inRange ||
isWithinRange(point1x - xoffset, point1y, point1z, range, point2x,
point2y, point2z);

    inRange = inRange ||
isWithinRange(point1x + xoffset, point1y, point1z, range, point2x,
point2y, point2z);

    inRange = inRange ||
isWithinRange(point1x - xoffset, point1y - yoffset, point1z, range,
point2x, point2y, point2z);

    inRange = inRange ||
isWithinRange(point1x, point1y - yoffset, point1z, range, point2x,
point2y, point2z);

    inRange = inRange ||
isWithinRange(point1x + xoffset, point1y - yoffset, point1z, range,
point2x, point2y, point2z);

```

```

    inRange = inRange ||
        isWithinRange(point1x - xoffset, point1y + yoffset, point1z, range,
            point2x, point2y, point2z);

    inRange = inRange ||
        isWithinRange(point1x, point1y + yoffset, point1z, range, point2x,
            point2y, point2z);

    inRange = inRange ||
        isWithinRange(point1x + xoffset, point1y + yoffset, point1z, range,
            point2x, point2y, point2z);

    return inRange;
}

/**
 * This method tests if the Euclidean distance between two points is less
 * than a specified range.
 *
 * @param point1x The location of point 1 in the X dimension.
 * @param point1y The location of point 1 in the Y dimension.
 * @param point1z The location of point 1 in the Z dimension.
 * @param range The range visible from point 1.
 * @param point2x The location of point 2 in the X dimension.
 * @param point2y The location of point 2 in the Y dimension.
 * @param point2z The location of point 2 in the Z dimension.
 *
 * @return true if point 2 is within range of point 1 - false otherwise.
 */
private static boolean isWithinRange(int point1x, int point1y, int point1z,
    int range, int point2x, int point2y, int point2z){
    double result = getEuclideanDistance(point1x, point1y, point1z,
        point2x, point2y, point2z);

    if(result < range){
        return true;
    }

    return false;
}

/**
 * This method is used to obtain a valid index into the 3D array of Bins.
 * It enforces "wraparound" to the 3 given indices so that a
 * NullPointerException is impossible no matter what the indices are.
 *
 * @param xind The desired X index.
 * @param yind The desired Y index.
 * @param zind The desired Z index.
 *
 * @return a 3 element array containing the normalised X, Y and Z indices
 * respectively.
 */
private static int[] getNormalisedBinIndex(int xind, int yind, int zind){
    //TODO consider changing this so that Z (i.e. height) dimension
    // is not wrapped around -- OK for 2D stuff though
    int xIndex = xind;
    int yIndex = yind;
    int zIndex = zind;

    while((xIndex < 0)){
        xIndex += nBins;
    }

    while((xIndex >= nBins)){
        xIndex -= nBins;
    }
}

```

```

    }

    while((yIndex < 0)){
        yIndex += nBins;
    }

    while((yIndex >= nBins)){
        yIndex -= nBins;
    }

    while((zIndex < 0)){
        zIndex += nBins;
    }

    while((zIndex >= nBins)){
        zIndex -= nBins;
    }

    int[] out = {xIndex, yIndex, zIndex};

    return out;
}

/**
 * This method initialises all aspects of the BinManager so that it is
 * ready for use.
 *
 * @param env The environment the BinManger is to divide up.
 */
static void initialise(Environment env){
    nBins = MacsPreferences.getPrefNoBins();
    bins = new BaseEntityList[nBins][nBins][nBins];

    environment = env;

    xBinWidth = ((float) env.xLimit) / nBins;
    yBinWidth = ((float) env.yLimit) / nBins;
    zBinWidth = ((float) env.zLimit) / nBins;

    // divide the environment into equal nBins size chunks along each
    // dimension
    for(int xIndex = 0;xIndex < nBins;xIndex++){
        for(int yIndex = 0;yIndex < nBins;yIndex++){
            for(int zIndex = 0;zIndex < nBins;zIndex++){
                BaseEntityList current = new BaseEntityList();
                bins[xIndex][yIndex][zIndex] = current;
            }
        }
    }

    // assign each member of the population to the correct bin for its
    // position
    for(int index = 0;index < env.population.size();index++){
        BaseEntity current = (BaseEntity) env.population.get(index);

        BaseEntityList holder = getBinContaining(current);

        holder.add(current);
        current.setBin(holder);
    }
}

/**
 * This method calculates the Euclidean distance between 2 given points in
 * 3D space.
 */

```

```
* @param point1x The location of point 1 in the X dimension.
* @param point1y The location of point 1 in the Y dimension.
* @param point1z The location of point 1 in the Z dimension.
* @param point2x The location of point 2 in the X dimension.
* @param point2y The location of point 2 in the Y dimension.
* @param point2z The location of point 2 in the Z dimension.
*
* @return The Euclidean distance between the two given points.
*/
private static double getEuclideanDistance(double point1x, double point1y,
double point1z, double point2x, double point2y, double point2z){
    double result = Math.sqrt(Math.pow((point1x - point2x), 2) +
        Math.pow((point1y - point2y), 2) +
        Math.pow((point1z - point2z), 2));

    return result;
}
}
```

```

package backend;

import common.MacsPreferences;
import common.ProgressDialog;

import datatypes.AbstractPlayable;
import datatypes.BaseEntity;
import datatypes.BaseEntityList;
import datatypes.Environment;

import gui.TimeControlFrame;

/**
 * This class is the core of the simulation engine and is responsible for
 * synchronising the movement and actions of agents within the system.
 *
 * @author Iain Robinson
 */
public class Engine extends AbstractPlayable{
    /**
     * A progress bar used to indicate the progress of a "write to file"
     * simulation.
     */
    public static ProgressDialog pbar;

    /** Indicates simulation output should be written to file. */
    public static final int OUTPUT_FILE = 0;

    /** Indicates simulation output should be displayed on screen. */
    public static final int OUTPUT_REAL = 1;

    /** The environment that will be simulated by the engine. */
    public Environment env;

    /**
     * Creates a new instance of Engine
     */
    public Engine(){
    }

    /**
     * This method initialises all aspects of the simulation engine in
     * readiness for a simulation run.
     *
     * @param env The environment that is to be simulated.
     */
    public void initialise(Environment env){
        //env is a clone of the original environment
        //this allows a simulation to be run...but no to affect the data that
        //it was initialised from...meaning that it can be re-run...given the same
data.
        this.env = env;
        BinManager.initialise(env);

        if(MacsPreferences.getPrefOutput() == Engine.OUTPUT_FILE){
            pbar = new ProgressDialog("Progress",
                MacsPreferences.getPrefNoIter());
        }
    }

    /**
     * This method causes a single iteration of the simulation to be executed -
     * i.e. each member of the population is allowed to execute its run method
     * once.
     */
}

```

```

public void play(){
    for(int i = 0;i < env.population.size();i++){
        BaseEntity current = (BaseEntity) env.population.get(i);
        current.run();

        BaseEntityList newLocation = BinManager.getBinContaining(current);
        BaseEntityList oldLocation = current.currentBin;

        if(!newLocation.equals(oldLocation)){
            oldLocation.remove(current);
            newLocation.add(current);
            current.currentBin = newLocation;
        }
    }

    System.out.println("count is " + count);

    if(MacsPreferences.getPrefOutput() == OUTPUT_FILE){
        pbar.update(count);
    }
}

/**
 * This method is used to specify what happens after the engine has
 * finished execution but not yet been disposed of.
 */
public void cleanup(){
    TimeControlFrame.setState(TimeControlFrame.STATE_PLAYABLE);
}

/**
 * Creates a 'snapshot' of the System in its current state.
 *
 * @return A clone of the system rather than just a pointer to the
 *         original.
 */
public BaseEntityList deliverCurrentData() {
    BaseEntityList copy = env.population.getDeepCopy();

    for(int i = 0;i < copy.size();i++){
        BaseEntity current = (BaseEntity) copy.get(i);
        current.setAttributes(current.getAttributes().getIndexed());
    }

    return copy;
}

/**
 * Suspend execution of the simulation engine until told to unPause.
 */
public void pause(){
    paused = true;
}

/**
 * Resume execution of the simulation engine.
 */
public void unPause(){
    paused = false;

    synchronized(this){
        this.notify();
    }
}

/**

```

```
    * This method moves the simulation on by one iteration only.
    */
    public void step(){
        stepping = true;
        unPause();
    }

    /**
     * This method suspends the stepping mode initiated by the step method.
     */
    public void unStep(){
        stepping = false;
    }

    /**
     * This method ends execution of a simulation.
     */
    public void end(){
        isAlive = false;
        cleanup();
    }
}
```

```

package backend;

import datatypes.BaseEntityList;

/**
 * This class is used to transfer data froma ScenarioFileReader to the
 * application. It parses all data found in a Scenario file and creates a data
 * structure for each that can be used directly by the application.
 *
 * @author Iain Robinson.
 */
public class ScenarioFileInfo{
    /** The scenario environment's maximum extent in the X dimension. */
    private int xLimit = 0;

    /** The scenario environment's maximum extent in the Y dimension. */
    private int yLimit = 0;

    /** The scenario environment's maximum extent in the Z dimension. */
    private int zLimit = 0;

    /** A list of all BaseEntitys that are used within the scenario. */
    private BaseEntityList population = new BaseEntityList(0, 1);

    /**
     * Creates a new ScenarioFileInfo object.
     */
    ScenarioFileInfo(){
    }

    /**
     * Get method for the maximum extent of the environment in the X dimension.
     *
     * @return Returns the xLimit.
     */
    public int getXLimit(){
        return xLimit;
    }

    /**
     * Set method for the maximum extent of the environment in the X dimension.
     *
     * @param limit The xLimit to set.
     */
    void setXLimit(int limit){
        xLimit = limit;
    }

    /**
     * Get method for the maximum extent of the environment in the Y dimension.
     *
     * @return Returns the yLimit.
     */
    public int getYLimit(){
        return yLimit;
    }

    /**
     * Set method for the maximum extent of the environment in the Y dimension.
     *
     * @param limit The yLimit to set.
     */
    void setYLimit(int limit){
        yLimit = limit;
    }
}

```

```
/**
 * Get method for the maximum extent of the environment in the Z dimension.
 *
 * @return Returns the zLimit.
 */
public int getZLimit(){
    return zLimit;
}

/**
 * Set method for the maximum extent of the environment in the Z dimension.
 *
 * @param limit The zLimit to set.
 */
void setZLimit(int limit){
    zLimit = limit;
}

/**
 * Get method for the population list of the parsed scenario file.
 *
 * @return Returns the population.
 */
public BaseEntityList getPopulation(){
    return population;
}

/**
 * Set method for the population list of the parsed scenario file.
 *
 * @param population The population to set.
 */
void setPopulation(BaseEntityList population){
    this.population = population;
}
}
```

```

package backend;

import common.ErrorReporter;
import common.FileIO;

import datatypes.Attribute;
import datatypes.BaseEntity;
import datatypes.BaseEntityList;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

import org.xml.sax.SAXException;

import java.io.IOException;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;

/**
 * This class is used to read a Scenario file and to get an object that
 * contains all the data it contains in a form that is usable within the
 * application.
 *
 * @author Iain Robinson.
 */
public class ScenarioFileReader{
    /** The current instance of the ScenarioFileReaderObject. */
    private static ScenarioFileReader inst;

    /**
     * Creates a new ScenarioFileReader object.
     */
    private ScenarioFileReader(){
    }

    /**
     * This method provides a convenient way of retrieving the current
     * singleton instance.
     *
     * @return The current instance of the class.
     */
    public static ScenarioFileReader getInstance(){
        if(inst == null){
            inst = new ScenarioFileReader();
        }

        return inst;
    }

    /**
     * This method is used to parse a given Scenario file into objects useable
     * to the rest of the application.
     *
     * @param fname The name of the file to parse.
     *
     * @return A ScenarioFileInfo object that contains all the data in a form
     *         that can be used within the rest of the application.
     */
    public static ScenarioFileInfo parse(String fname){
        ScenarioFileInfo info = new ScenarioFileInfo();

```

```

//create the initial DOM for the scenario
Document sDoc = createScenarioDOM(fname);

//parse xLimit tag
info.setXLimit(parseIntXMLTag(sDoc, "xLimit"));

//parse yLimit tag
info.setYLimit(parseIntXMLTag(sDoc, "yLimit"));

//parse zLimit tag
info.setZLimit(parseIntXMLTag(sDoc, "zLimit"));

//parse the population element of the data
Node popTag = (sDoc.getElementsByTagName("population")).item(0);
NodeList individuals = ((Element) popTag).getElementsByTagName("entity");

BaseEntityList tempPop = new BaseEntityList(0, 1);

for(int i = 0;i < individuals.getLength();i++){
    //get each member of the population from the file
    Node currentEnt = individuals.item(i);

    // parse it, create new entity
    BaseEntity entity = parseEntityXML(currentEnt);

    NodeList attributeList = ((Element) currentEnt).getElementsByTagName(
        "attribute");

    for(int i2 = 0;i2 < attributeList.getLength();i2++){
        //parse each attribute
        Node currentAtt = attributeList.item(i2);
        Attribute newAtt = parseAttributeXML(currentAtt);

        //and add it to the entity
        entity.addAttribute(newAtt);
    }

    tempPop.add(entity);
}

info.setPopulation(tempPop);

return info;
}

/**
 * This method creates a full Scenario XML DOM for the Macs scenario file
 * specified by the given filename. Allows further processing to be done
 * on the DOM.
 *
 * @param fname The name of the scenario file.
 *
 * @return A blank template Scenario file XML document.
 */
private static Document createScenarioDOM(String fname){
    DocumentBuilder docBuilder = null;

    try{
        //parse the full scenario data into document structure
        docBuilder = DocumentBuilderFactory.newInstance()
            .newDocumentBuilder();
    }catch(ParserConfigurationException e){
        ErrorReporter.report(ErrorReporter.FAILED_PARSER);
    }
}

```

```

Document sDoc = null;

try{
    sDoc = docBuilder.parse(fname);
} catch(IOException error){
    ErrorReporter.report(ErrorReporter.FAILED_SCENARIO_LOAD);
} catch(SAXException error){
    ErrorReporter.report(ErrorReporter.FAILED_SCENARIO_LOAD);
}

return sDoc;
}

/**
 * This method parses a given XML entity node from a Scenerio file and
 * returns the class type of the entity in question.
 *
 * @param currentEnt The XML Entity node to be parsed.
 *
 * @return A String containing the species class type of the given entity
 *         XML code
 */
private static BaseEntity parseEntityXML(Node currentEnt){
    //get its attributes (i.e. its type in other words its class name)
    NamedNodeMap attributes = currentEnt.getAttributes();
    Node memTypeValue = attributes.getNamedItem("type");
    String typeValue = memTypeValue.getNodeValue();

    // create new entity
    BaseEntity entity = FileIO.importSpeciesClassFile(typeValue);

    return entity;
}

/**
 * This method parses a given XML attribute node from a Scenerio file and
 * returns a valid Attribute containing the read information.
 *
 * @param currentAtt The XML Attribute Node to be parsed.
 *
 * @return A valid Attribute instance representing the XML data
 */
private static Attribute parseAttributeXML(Node currentAtt){
    NamedNodeMap atts = currentAtt.getAttributes();

    String attName = atts.getNamedItem("name").getNodeValue();

    Node nValue = atts.getNamedItem("value");

    Node nType = atts.getNamedItem("type");
    int typeCode = Integer.parseInt(nType.getNodeValue());

    boolean isIndexed = Boolean.parseBoolean(atts.getNamedItem("indexed")
                                           .getNodeValue());

    Object attValue;

    switch(typeCode){
    case Attribute.TYPE_INT:
        attValue = new Integer(nValue.getNodeValue());

        break;

    case Attribute.TYPE_STRING:
        attValue = nValue.getNodeValue();

```

```

        break;

    case Attribute.TYPE_DOUBLE:
        attValue = new Double(nValue.getNodeValue());

        break;

    case Attribute.TYPE_BOOLEAN:
        attValue = Boolean.valueOf(nValue.getNodeValue());

        break;

    default:

        // next line just to initialise the value to something
        attValue = new Integer(0);

        //set the type here as if this point is reached it wouldn't be valid.
        typeCode = Attribute.TYPE_INT;

        break;
    }

    return new Attribute(attName, attValue, typeCode, isIndexed);
}

/**
 * This method parses a given XML node from a Scenerio file and returns an
 * integer value.
 *
 * @param doc The document that contains the XML tag to be parsed.
 * @param tagName The name of the XML tag.
 *
 * @return A integer representing the XML data
 */
private static int parseIntXMLTag(Document doc, String tagName){
    Node tag = (doc.getElementsByTagName(tagName)).item(0);
    NamedNodeMap attributes = tag.getAttributes();
    Node nValue = attributes.getNamedItem("value");
    int value = Integer.parseInt(nValue.getNodeValue());

    return value;
}
}

```

```

package backend;

import common.ErrorReporter;

import datatypes.Attribute;
import datatypes.BaseEntity;
import datatypes.Environment;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;

import java.io.File;

import java.util.Enumeration;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.OutputKeys;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerConfigurationException;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;

/**
 * This class is used to write all relevant data to a Scenario File.
 *
 * @author Iain Robinson
 */
public class ScenarioFileWriter{
    /**
     * A String containing the tag name "value". Simply used to avoid
     * repetition of the string literal in the code.
     */
    private static String valueTag = "value";

    /** The current instance of the ScenarioFileWriterObject. */
    private static ScenarioFileWriter inst;

    /** The XML DOM that will form the basis of the written XML file */
    private static Document sDoc;

    /**
     * Creates a new ScenarioFileWriter object.
     */
    private ScenarioFileWriter(){
    }

    /**
     * This method provides a convenient way of retrieving the current
     * singleton instance.
     *
     * @return The current instance of the class.
     */
    private static ScenarioFileWriter getInstance(){
        if(inst == null){
            inst = new ScenarioFileWriter();
        }

        return inst;
    }
}

```

```

/**
 * This method saves the Macs Scenario in a file specified by the given
 * filename.
 *
 * @param env The environment to write to the file.
 * @param fname The name of the project file.
 */
public static void write(Environment env, String fname){
    String correctfname = fname;

    // check the file has the correct extension
    if(!fname.endsWith(".msf")){
        // and give it 1 if not
        correctfname = fname + ".msf";
    }

    DocumentBuilder docBuilder = null;

    try{
        //then write scenario info to the file
        docBuilder = DocumentBuilderFactory.newInstance()
            .newDocumentBuilder();
    }catch(ParserConfigurationException e){
        ErrorReporter.report(ErrorReporter.FAILED_PARSER);
    }

    SDoc = docBuilder.newDocument();

    Node sNode = sDoc;

    //create the environment section
    Element envTag = createEnvironmentXML(env);

    // create the population section
    Element popTag = sDoc.createElement("population");

    for(int i = 0;i < env.population.size();i++){
        // get each member
        BaseEntity currentEntity = (BaseEntity) env.population.get(i);

        //create the members XML tag
        Element memberTag = createEntityXML(currentEntity);
        popTag.appendChild(memberTag);
    }

    envTag.appendChild(popTag);
    sNode.appendChild(envTag);

    //write to file via transform...which doesnt seem 2 pretty print ???
    File file = new File(correctfname);
    writeXML(sDoc, file);
}

/**
 * This method creates the environment portion of a Scenario file and
 * inserts all necessary data values.
 *
 * @param env The environment to write to the file.
 *
 * @return An XML element containing all required information about the
 *         Environment.
 */
private static Element createEnvironmentXML(Environment env){
    Element envTag = sDoc.createElement("environment");

    // create nodes for all the environment settings

```

```

Element xLimitTag = sDoc.createElement("xLimit");
xLimitTag.setAttribute(valueTag, Integer.toString(env.xLimit));

Element yLimitTag = sDoc.createElement("yLimit");
yLimitTag.setAttribute(valueTag, Integer.toString(env.yLimit));

Element zLimitTag = sDoc.createElement("zLimit");
zLimitTag.setAttribute(valueTag, Integer.toString(env.zLimit));

//add environment settings nodes to the environment section
envTag.appendChild(xLimitTag);
envTag.appendChild(yLimitTag);
envTag.appendChild(zLimitTag);

return envTag;
}

/**
 * This method creates the entity portion of a Scenario file and inserts
 * all necessary data values.
 *
 * @param currentEntity The entity to add to the document.
 *
 * @return An XML element containing all required information about the
 *         entity.
 */
private static Element createEntityXML(BaseEntity currentEntity){
    // create a member section
    Element memberTag = sDoc.createElement("entity");

    //set its class type
    memberTag.setAttribute("type",
        currentEntity.getClass().getCanonicalName());

    // process all its attributes
    Enumeration attributeList = currentEntity.getAttributes().elements();

    while(attributeList.hasMoreElements()){
        Attribute currentAttr = (Attribute) attributeList.nextElement();
        String attributeName = currentAttr.getName();

        //create the attribute XML tag
        Element attributeTag = createAttributeXML(currentEntity,
            attributeName);
        memberTag.appendChild(attributeTag);
    }

    return memberTag;
}

/**
 * This method creates the XML of the attribute of an entity when saving
 * the details of a Scenario to file.
 *
 * @param currentEntity The entity to get the attribute from.
 * @param attributeName The name of the attribute to retrieve.
 *
 * @return An XML element containing all required information about the
 *         given attribute contained in the entity.
 */
private static Element createAttributeXML(BaseEntity currentEntity,
    String attributeName){
    Attribute currentAttribute = currentEntity.getAttribute(attributeName);

    // create attribute section
    Element attributeTag = sDoc.createElement("attribute");

```

```

attributeTag.setAttribute("name", attributeName);

int type = currentAttribute.getType();
attributeTag.setAttribute("type", "" + type);

switch(type){
case Attribute.TYPE_INT:
    attributeTag.setAttribute(valueTag,
        Integer.toString(currentAttribute.getIntValue()));

    break;

case Attribute.TYPE_STRING:
    attributeTag.setAttribute(valueTag,
        currentAttribute.getStringValue());

    break;

case Attribute.TYPE_DOUBLE:
    attributeTag.setAttribute(valueTag,
        Double.toString(currentAttribute.getDoubleValue()));

    break;

case Attribute.TYPE_BOOLEAN:
    attributeTag.setAttribute(valueTag,
        Boolean.toString(currentAttribute.getBooleanValue()));

    break;

default:
    attributeTag.setAttribute(valueTag, Integer.toString(0));

    break;
}

attributeTag.setAttribute("indexed",
    Boolean.toString(currentAttribute.isIndexed()));

return attributeTag;
}

/**
 * This method writes a fully created XML document to a given file.
 *
 * @param doc The full XML document.
 * @param out The desired file to save to.
 */
private static void writeXML(Document doc, File out){
    DOMSource source = new DOMSource(doc);
    StreamResult result = new StreamResult(out);
    doXMLTransform(source, result);
}

/**
 * This method uses an identity transform to "pretty print" the XML saved
 * to file. Due to the capabilities of the in-built java XML engine the
 * output is not tabbed.
 *
 * @param source The XML to transform.
 * @param result The stream to write the result to.
 */
private static void doXMLTransform(DOMSource source, StreamResult result){
    TransformerFactory tfactory = TransformerFactory.newInstance();
    Transformer trans = null;

```

```
try{
    trans = tfactory.newTransformer();
}catch(TransformerConfigurationException e){
    ErrorReporter.report(ErrorReporter.FAILED_TRANSFORMER_CONFIG);
}

trans.setOutputProperty(OutputKeys.METHOD, "xml");
trans.setOutputProperty(OutputKeys.ENCODING, "UTF-8");
trans.setOutputProperty(OutputKeys.INDENT, "yes");
trans.setOutputProperty("{http://xml.apache.org/xslt}indent-amount", "4");

try{
    trans.transform(source, result);
}catch(TransformerException e){
    ErrorReporter.report(ErrorReporter.FAILED_TRANSFORMER_RUN);
}
}
```

```

package common;

import javax.swing.JOptionPane;

/**
 * This class is responsible for generating and displaying all necessary error
 * messages that may be needed during application execution. It provides a
 * "one-stop" solution to locating errors within the system rather than having
 * many separate calls scattered throughout the code base.
 *
 * @author Iain Robinson
 */
public final class ErrorReporter{
    /** A standard error message. */
    private static final String STD_MESSAGE = "An Error Has Occured";

    /**
     * This message indicates that a problem has occurred when loading the Look
     * & Feel of the application.
     */
    public static final String LOOK_AND_FEEL = "The Correct Look & Feel Could Not Be
Loaded";

    /**
     * This message indicates that a problem has occurred when importing a
     * Species class file.
     */
    static final String FAILED_CLASS_IMPORT = "The Specified Class Could Not Be
Imported";

    /**
     * This message indicates that a problem has occurred when trying to
     * instantiate an imported Species class file.
     */
    public static final String FAILED_INSTANTIATION = "The Specified Class Could Not Be
Instantiated";

    /**
     * This message indicates that a problem has occurred when loading a
     * Scenario file.
     */
    public static final String FAILED_SCENARIO_LOAD = "The Specified Scenario File
Could Not Be Loaded";

    /**
     * This message indicates that a problem has occurred when attempting to
     * setup the Scenario file XML parser during a file load.
     */
    public static final String FAILED_PARSER = "Unable To Instantiate XML Parser";

    /**
     * This message indicates that a problem has occurred when attempting to
     * setup the Scenario file XML transformer during a file save.
     */
    public static final String FAILED_TRANSFORMER_CONFIG = "Unable To Configure XML
Transformer";

    /**
     * This message indicates that a problem has occurred when attempting to
     * execute the Scenario file XML tranformer during a file save.
     */
    public static final String FAILED_TRANSFORMER_RUN = "Unable To Perform XML
Transform";

    /**

```

```

    * This message indicates that a problem has occurred during a simulation
    * run that has resulted in an entity being interrupted whilst executing.
    */
    public static final String THREAD_INTERRUPTED = "A Thread Has Been Interrupted
    Whilst Executing";

    /**
     * This message indicates that a problem has occurred when attempting to
     * transfer an entity from the PaletteFrame to the CanvasFrame.
     */
    public static final String FAILED_ENTITY_DROP = "The Entity Drop Was Unable To Be
    Completed";

    /**
     * This message indicates that a problem has occurred when attempting to
     * load a Species source file.
     */
    static final String FAILED_SPECIES_SOURCE_LOAD = "The Specified Species Source File
    Could Not Be Loaded";

    /**
     * This message indicates that a problem has occurred when attempting to
     * save a Species source file.
     */
    static final String FAILED_SPECIES_SOURCE_SAVE = "The Specified Species Source File
    Could Not Be Saved";

    /** This message indicates that the specified GIF file was not found. */
    public static final String GIF_FILE_NOT_FOUND = "The specified GIF file was not
    found";

    /**
     * This message indicates that a problem has occurred when attempting to
     * copy the specified GIF file to a species subdirectory.
     */
    public static final String GIF_FILE_TRANSFER_FAILURE = "The specified GIF file was
    unable to be copied";

    /**
     * This message indicates that a problem has occurred when attempting to
     * compile the specified species source code.
     */
    public static final String SOURCE_COMPILATION_FAILURE = "The specified source file
    was unable to be compiled properly";

    /**
     * This method handles all exceptions that occur during program execution
     * and provide an easy way of providing error messages to a user without a
     * lot of code being placed throughout the main classes.
     *
     * @param error The error that has caused the exception to be thrown.
     */
    public static void report(String error){
        JOptionPane.showMessageDialog(null, error, ErrorReporter.STD_MESSAGE,
        JOptionPane.ERROR_MESSAGE);
    }
}

```

```

package common;

import backend.ScenarioFileInfo;
import backend.ScenarioFileReader;
import backend.ScenarioFileWriter;

import datatypes.Attribute;
import datatypes.BaseEntity;

import gui.CanvasFrame;
import gui.EntityEditorFrame;
import gui.MacSim;
import gui.PaletteFrame;
import gui.SpeciesWizard;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;

import javax.swing.JFileChooser;
import javax.swing.filechooser.FileFilter;

/**
 * This class implements all File input/output methods that the Macs
 * application uses.
 *
 * @author Iain Robinson
 */
public class FileIO{
    /**
     * Used as an argument for the getFileFilter method to indicate a Species
     * Class File filter.
     */
    private static final int SPECIES_CLASS_FILE = 0;

    /**
     * Used as an argument for the getFileFilter method to indicate a Species
     * Source File filter.
     */
    private static final int SPECIES_SOURCE_FILE = 1;

    /**
     * Used as an argument for the getFileFilter method to indicate a Scenario
     * File filter.
     */
    private static final int SCENARIO_FILE = 2;

    /**
     * Used as an argument for the getFileFilter method to indicate an XML File
     * filter.
     */
    public static final int XML_FILE = 3;

    /**
     * Used as an argument for the getFileFilter method to indicate a GIF File
     * filter.
     */
    private static final int GIF_FILE = 4;

    /** The "root" directory that the application is installed in. */

```

```

public static final String PATH = System.getProperty("exedir");

/** The application's scenario subdirectory. */
private static final String SCENARIO_DIR = PATH + "\\scenarios";

/** The application's species subdirectory. */
public static final String SPECIES_DIR = PATH + "\\species";

/**
 * Imports a Java class file. The class file must properly extend the
 * BaseEntity class to function correctly although this method will report
 * any error if this is not the case it does not as yet enforce such a
 * requirement. Even though this method accepts any class file no further
 * processing of it will be possible by the Macs system.
 *
 * @param className The file name of the class to be loaded.
 *
 * @return A valid BaseEntity instance
 */
public static BaseEntity importSpeciesClassFile(final String className){
    Class classToLoad = null;

    try{
        classToLoad = Class.forName(className);

        try{
            Object obj = classToLoad.newInstance();

            return (BaseEntity) obj;
        }catch(InstantiationException err3){
            ErrorReporter.report(ErrorReporter.FAILED_INSTANTIATION);
        }catch(IllegalAccessException err4){
            ErrorReporter.report(ErrorReporter.FAILED_INSTANTIATION);
        }
    }catch(ClassNotFoundException err2){
        ErrorReporter.report(ErrorReporter.FAILED_CLASS_IMPORT);
    }

    return null;
}

/**
 * This method will compile the source code in the currently opened entity
 * source code file - visible in the EntityEditor frame.
 *
 * @param fname The name of the source file to be compiled.
 */
public static void compileSourceFile(String fname){
    String command = "javac -source 1.4 -classpath " + FileIO.PATH +
        "\\MacsProject.jar " + "\"" + fname + "\"";

    String current = "Attempting to compile: " + command + "...\\n";
    EntityEditorFrame.debugPane.setText(current);

    try{
        Runtime runtime = Runtime.getRuntime();
        Process proc = runtime.exec(command);
        InputStream stderr = proc.getErrorStream();
        InputStreamReader instream = new InputStreamReader(stderr);
        BufferedReader reader = new BufferedReader(instream);

        String line = null;

        do{
            line = reader.readLine();
            current = current + line + "\\n";
        }
    }
}

```

```

    }while(line != null);

    int exitVal = proc.waitFor();

    if(exitVal == 0){
        current = current + "\nCompilation exitValue: " + exitVal +
            "\nCompilation Successful";
    }else{
        current = current + "\nCompilation exitValue: " + exitVal +
            "\nCompilation NOT Successful";
    }

    EntityEditorFrame.debugPane.setText(current);
}catch(IOException t){
    ErrorReporter.report(ErrorReporter.SOURCE_COMPILATION_FAILURE);
}catch(InterruptedException t){
    ErrorReporter.report(ErrorReporter.SOURCE_COMPILATION_FAILURE);
}
}

/**
 * Opens the Macs scenario file specified by the given filename.
 *
 * @param fname The name of the scenario file.
 */
private static void openScenarioFile(String fname){
    //parse the file
    ScenarioFileInfo currentFile = ScenarioFileReader.parse(fname);

    //set up the required environment space
    MacsPreferences.setPrefEnvDimensions(currentFile.getXLimit(),
        currentFile.getYLimit(), currentFile.getZLimit());

    for(int i = 0;i < currentFile.getPopulation().size();i++){
        //get each member of the population
        BaseEntity entity = (BaseEntity) currentFile.getPopulation().get(i);
//ScenarioFileReader.parseEntityXML(currentEnt);

        // create new entity for the pallette
        BaseEntity entityCopy = entity.getDeepCopy();

        //and add to palette
        if(!PaletteFrame.getSpeciesList().contains(entity)){
            MacSim.iconList.addSpecies(entityCopy);
        }

        Attribute xLoc = entity.getAttribute("xPosition");
        Attribute yLoc = entity.getAttribute("yPosition");

        // and add original to the canvas
        CanvasFrame.addIcon(entity, xLoc.getIntValue(), yLoc.getIntValue());
    }
}

/**
 * This method is used to open any species Java source (.java) file.
 *
 * @param fname The name of the Java source file.
 */
public static void openSpeciesSourceFile(String fname){
    try{
        File input = new File(fname);
        FileInputStream text = new FileInputStream(input);

        int size = (int) input.length();
        byte[] buff = new byte[size];

```

```

        text.read(buff, 0, size);

        String out = new String(buff);
        EntityEditorFrame.editorPane.setText(out);
        EntityEditorFrame.jDoc.tokenizeLines();
        MacSim.editor.setVisible(true);

        EntityEditorFrame.currentSpeciesSourceFileName = fname;
    } catch (FileNotFoundException error) {
        ErrorReporter.report(ErrorReporter.FAILED_SPECIES_SOURCE_LOAD);
    } catch (IOException error) {
        ErrorReporter.report(ErrorReporter.FAILED_SPECIES_SOURCE_LOAD);
    }
}

/**
 * This method is used to save any species Java source file.
 *
 * @param fname The name of the Java source file to save to.
 * @param output The source code as a string.
 */
public static void saveSpeciesSourceFile(String fname, String output){
    String correctfname = fname;

    // check the file ahs the correct extension
    if(!fname.endsWith(".java")){
        // and give it 1 if not
        correctfname = fname + ".java";
    }

    try{
        FileOutputStream out = new FileOutputStream(new File(correctfname));
        out.write(output.getBytes());
        out.close();
    } catch (FileNotFoundException error1){
        ErrorReporter.report(ErrorReporter.FAILED_SPECIES_SOURCE_SAVE);
    } catch (IOException error1){
        ErrorReporter.report(ErrorReporter.FAILED_SPECIES_SOURCE_SAVE);
    }
}

/**
 * This method provides an instance of a Species class file filter for use
 * in file dialogs.
 *
 * @param type An integer representing the type of file filter required.
 *
 * @return A file filter of the desired type - or null if type is
 *         unrecognised.
 */
public static FileFilter getFileFilter(int type){
    switch(type){
        case SPECIES_CLASS_FILE:
            return new SpeciesClassFileFilter();

        case SPECIES_SOURCE_FILE:
            return new SpeciesSourceFileFilter();

        case SCENARIO_FILE:
            return new ScenarioFileFilter();

        case XML_FILE:
            return new XMLFileFilter();

        case GIF_FILE:
            return new GIFFileFilter();
    }
}

```

```

        default:
            return null;
    }
}

/**
 * This method executes a certain "file action" after receiving a request
 * from the user. Opening a species or scenario file are examples of this.
 *
 * @param type The type of "file action" to perform.
 * @param fname The filename of the file to perform the action upon.
 */
private static void doFileAction(String type, String fname){
    if(type.equals("OpenScenario")){
        openScenarioFile(fname);
    }

    if(type.equals("SaveScenario")){
        ScenarioFileWriter.write(MacSim.environment, fname);
    }

    if(type.equals("OpenSpecies")){
        openSpeciesSourceFile(fname);
    }

    if(type.equals("SaveSpecies")){
        saveSpeciesSourceFile(fname, EntityEditorFrame.editorPane.getText());
    }

    if(type.equals("Import")){
        //strip all path details
        int lastSlashPos = fname.lastIndexOf("\\");
        String className = fname.substring(lastSlashPos + 1);

        //strip .class extension
        int classExtPos = className.indexOf(".");
        String pureClassName = className.substring(0, classExtPos);

        String fullclassName = "species." + pureClassName.toLowerCase() +
            "." + pureClassName;

        BaseEntity typeInstance = importSpeciesClassFile(fullclassName);

        //if the instance exists then now import it
        if(typeInstance != null){
            MacSim.iconList.addSpecies(typeInstance);
        }
    }

    if(type.equals("OpenGIF")){
        SpeciesWizard.iconField.setText(fname);
    }
}

/**
 * This method implements the launch of any relevant file dialogs for
 * commands such as file open/save and species import.
 *
 * @param type The type of file operation to be performed such as
 * open/save.
 */
public static void launchFileDialog(String type){
    JFileChooser fManager = new JFileChooser();

    //TODO needs to check if user really want to continue...then clear the

```

```

// canvas (and the environment)
if(type.equals("OpenScenario") || type.equals("SaveScenario")){
    openScenarioDirectory(type, fManager);
}

if(type.equals("OpenSpecies") || type.equals("SaveSpecies") ||
    type.equals("Import")){
    openSpeciesDirectory(type, fManager);
}

if(type.equals("OpenGIF")){
    fManager.setCurrentDirectory(new File(PATH));
    fManager.setFileFilter(FileIO.getFileFilter(FileIO.GIF_FILE));
}

int returnVal = fManager.showOpenDialog(null);

if(returnVal == JFileChooser.APPROVE_OPTION){
    String fname = fManager.getSelectedFile().getAbsolutePath();
    doFileAction(type, fname);
}
}

/**
 * This method opens the species subdirectory after receiving a request to
 * do so from the launchFileDialog method.
 *
 * @param type The type of "file action" to perform.
 * @param fManager The file dialog to use.
 */
private static void openSpeciesDirectory(String type, JFileChooser fManager){
    if(type.equals("OpenSpecies")){
        fManager.setCurrentDirectory(new File(SPECIES_DIR));
        fManager.setFileFilter(getFileFilter(SPECIES_SOURCE_FILE));
    }

    if(type.equals("SaveSpecies")){
        fManager.setCurrentDirectory(new File(SPECIES_DIR));
        fManager.setFileFilter(getFileFilter(SPECIES_SOURCE_FILE));
    }

    if(type.equals("Import")){
        fManager.setCurrentDirectory(new File(SPECIES_DIR));
        fManager.setFileFilter(getFileFilter(SPECIES_CLASS_FILE));
    }
}

/**
 * This method opens the scenario subdirectory after receiving a request to
 * do so from the launchFileDialog method.
 *
 * @param type The type of "file action" to perform.
 * @param fManager The file dialog to use.
 */
private static void openScenarioDirectory(String type, JFileChooser fManager){
    if(type.equals("OpenScenario")){
        fManager.setCurrentDirectory(new File(SCENARIO_DIR));
        fManager.setFileFilter(getFileFilter(SCENARIO_FILE));
    }

    if(type.equals("SaveScenario")){
        fManager.setCurrentDirectory(new File(SCENARIO_DIR));
        fManager.setFileFilter(getFileFilter(SCENARIO_FILE));
    }
}

```

```

/**
 * This class is a simple filter used when the user is searching for a
 * compiled Java agent they wish to import into the application.
 *
 * @author Iain Robinson
 * @version $Revision$
 */
private static class SpeciesClassFileFilter extends FileFilter{
    /**
     * Confirms if a given file is "passed through" the filter.
     *
     * @see javax.swing.filechooser.FileFilter#accept(java.io.File)
     */
    public boolean accept(File file){
        if(file.isDirectory()){
            return true;
        }

        //String extension = file.getName().split("\\.")[1];
        if(file.getName().endsWith(".class")){
            return true;
        }

        return false;
    }

    /**
     * Provides a textual description of the filter type.
     *
     * @see javax.swing.filechooser.FileFilter#getDescription()
     */
    public String getDescription(){
        return "Macs Species Class Files (.class)";
    }
}

/**
 * This class is a simple filter used when the user is searching for a Java
 * source code file for an agent they wish to load or save via the
 * application.
 *
 * @author Iain Robinson
 * @version $Revision$
 */
private static class SpeciesSourceFileFilter extends FileFilter{
    /**
     * Confirms if a given file is "passed through" the filter.
     *
     * @see javax.swing.filechooser.FileFilter#accept(java.io.File)
     */
    public boolean accept(File file){
        if(file.isDirectory()){
            return true;
        }

        if(file.getName().endsWith(".java")){
            return true;
        }

        return false;
    }

    /**
     * Provides a textual description of the filter type.
     *
     * @see javax.swing.filechooser.FileFilter#getDescription()

```

```

        */
        public String getDescription(){
            return "Macs Species Source Files (.java)";
        }
    }

/**
 * This class is a simple filter used when the user is searching for a MACS
 * Scenario file they wish to load or save via the application.
 *
 * @author Iain Robinson
 * @version $Revision$
 */
private static class ScenarioFileFilter extends FileFilter{
    /**
     * Confirms if a given file is "passed through" the filter.
     *
     * @see javax.swing.filechooser.FileFilter#accept(java.io.File)
     */
    public boolean accept(File file){
        if(file.isDirectory()){
            return true;
        }

        if(file.getName().endsWith(".msf")){
            return true;
        }

        return false;
    }

    /**
     * Provides a textual description of the filter type.
     *
     * @see javax.swing.filechooser.FileFilter#getDescription()
     */
    public String getDescription(){
        return "Macs Scenario Files (.msf)";
    }
}

/**
 * This class is a simple filter used when the user is about to run a file
 * based simulation and wishes to specify a location to save the MACS
 * Simulation file that will be generated.
 *
 * @author Iain Robinson
 * @version $Revision$
 */
private static class XMLFileFilter extends FileFilter{
    /**
     * Confirms if a given file is "passed through" the filter.
     *
     * @see javax.swing.filechooser.FileFilter#accept(java.io.File)
     */
    public boolean accept(File file){
        if(file.isDirectory()){
            return true;
        }

        if(file.getName().endsWith(".xml")){
            return true;
        }

        return false;
    }
}

```

```

    /**
     * Provides a textual description of the filter type.
     *
     * @see javax.swing.filechooser.FileFilter#getDescription()
     */
    public String getDescription(){
        return "XML Files (.xml)";
    }
}

/**
 * This class is a simple filter used when the user is searching for a GIF
 * image to represent an agent they are creating.
 *
 * @author Iain Robinson
 * @version $Revision$
 */
private static class GIFFileFilter extends FileFilter{
    /**
     * Confirms if a given file is "passed through" the filter.
     *
     * @see javax.swing.filechooser.FileFilter#accept(java.io.File)
     */
    public boolean accept(File file){
        if(file.isDirectory()){
            return true;
        }

        if(file.getName().endsWith(".gif")){
            return true;
        }

        return false;
    }

    /**
     * Provides a textual description of the filter type.
     *
     * @see javax.swing.filechooser.FileFilter#getDescription()
     */
    public String getDescription(){
        return "GIF Image Files (.gif)";
    }
}
}

```

```

package common;

import backend.Engine;

import gui.CanvasFrame;
import gui.FrameMenuBar;
import gui.MacSim;
import gui.RunMacs;

import java.awt.GridLayout;
import java.awt.Point;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import java.util.prefs.Preferences;

import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JDialog;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;

/**
 * This class provides a graphical representation of all underlying current
 * preferences set by the user and can be used to change those preferences at
 * any time the program is executing.
 *
 * @author Iain Robinson
 */
public class MacsPreferences extends JDialog{
    /** The underlying Preferences class that this class provides a GUI for. */
    private static Preferences prefs;

    /** The field a user can use to specify the default viewing mode. */
    private JComboBox defModeField;

    /**
     * The field a user can use to specify the default number of bins used
     * within the simulation engine.
     */
    private JTextField defNoBinsField = new JTextField("10", 4);

    /** The field a user can use to specify the default output mode. */
    private JComboBox defOutField;

    /**
     * A string indicating the default value of "500". Simply used to avoid
     * multiple uses of the same String literal in the code.
     */
    private String defaultValue = "500";

    /**
     * The field a user can use to specify the default maximum X limit of the
     * environment.
     */
    private JTextField defXDimField = new JTextField(defaultValue, 4);

    /**
     * The field a user can use to specify the default maximum Y limit of the
     * environment.
     */
    private JTextField defYDimField = new JTextField(defaultValue, 4);
}

```

```

* The field a user can use to specify the default maximum Z limit of the
* environment.
*/
private JTextField defZDimField = new JTextField(defaultValue, 4);

/**
* The field a user can use to specify the default number of iterations
* used when performing a simulation.
*/
private JTextField defNoIterField = new JTextField(defaultValue, 4);

/** A list of the different output modes available. */
private Object[] outChoices = {"File Output", "Realtime Output"};

/** a list of the different viewing modes available. */
private Object[] viewChoices = {"Viewer Mode", "Developer Mode"};

/**
* This constructor creates the graphical interface to the underlying
* preferences
*/
public MacsPreferences(){
    defModeField = new JComboBox(viewChoices);
    defOutField = new JComboBox(outChoices);

    this.setTitle("Current Preferences");
    this.setModal(true);

    prefs = Preferences.userNodeForPackage(this.getClass());
    setViewFromPrefs();

    JButton okButton = new JButton("Confirm");
    JButton cancelButton = new JButton("Cancel");

    JLabel defXDimLbl = new JLabel("X Dimension Size");
    JLabel defYDimLbl = new JLabel("Y Dimension Size");
    JLabel defZDimLbl = new JLabel("Z Dimension Size");
    JLabel defNoBinsLbl = new JLabel("Bins per Dimension");
    JLabel defNoIterLbl = new JLabel("No. of Iterations");
    JLabel defModeLbl = new JLabel("Viewing Mode");
    JLabel defOutLbl = new JLabel("Output Mode");

    //-- Set action listeners.
    okButton.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent event){
            int xLimit = Integer.parseInt(defXDimField.getText());
            int yLimit = Integer.parseInt(defYDimField.getText());
            int zLimit = Integer.parseInt(defZDimField.getText());
            int bins = Integer.parseInt(defNoBinsField.getText());
            int iter = Integer.parseInt(defNoIterField.getText());

            setPrefEnvDimensions(xLimit, yLimit, zLimit);
            setPrefNoBins(bins);
            setPrefNoIter(iter);

            String modeChoice = (String) defModeField.getSelectedItem();

            if(modeChoice.equals("Viewer Mode")){
                setPrefView(MacSim.VIEW_VIEWER);
            }else{
                setPrefView(MacSim.VIEW_DEV);
            }

            String outChoice = (String) defOutField.getSelectedItem();

            if(outChoice.equals("File Output")){

```

```

        setPrefOutput(Engine.OUTPUT_FILE);
    }else{
        setPrefOutput(Engine.OUTPUT_REAL);
    }

    MacSim.prefs.setVisible(false);
    });

// just close the frame if the user cancels
cancelButton.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent event){
        MacSim.update(getPrefView());
        FrameMenuBar.update(getPrefView());
        MacSim.prefs.setVisible(false);
    }
});

GridLayout grid = new GridLayout(8, 2);
grid.setVgap(20);
grid.setHgap(20);

// Layout components.
JPanel content = new JPanel(grid);

content.add(defXDimLbl);
content.add(defXDimField);
content.add(defYDimLbl);
content.add(defYDimField);
content.add(defZDimLbl);
content.add(defZDimField);
content.add(defNoBinsLbl);
content.add(defNoBinsField);
content.add(defNoIterLbl);
content.add(defNoIterField);
content.add(defModeLbl);
content.add(defModeField);
content.add(defOutLbl);
content.add(defOutField);
content.add(okButton);
content.add(cancelButton);

this.setContentPane(content);
this.pack();
}

/**
 * This method is used to display the preferences frame in the centre of
 * the screen.
 */
public void display(){
    Point appLocation = RunMacs.inst.getLocation();

    this.setLocation(appLocation.x +
        ((RunMacs.inst.getWidth() - this.getWidth()) / 2),
        appLocation.y +
        ((RunMacs.inst.getHeight() - this.getHeight()) / 2));
    MacSim.prefs.setVisible(true);
}

/**
 * A get method that indicates if ththe application is been started on a
 * previous occansion.
 *
 * @return true if the application has been launched before, false if not.
 */

```

```

public static boolean getFirstView(){
    return prefs.getBoolean("firstView", true);
}

/**
 * A get method for the preferred number of bins used by the simulation
 * engine.
 *
 * @return The preferred number of bins used by the simulation engine.
 */
public static int getPrefNoBins(){
    return prefs.getInt("noBins", 10);
}

/**
 * A get method for the preferred number of iterations used by the
 * simulation engine.
 *
 * @return The preferred number of iterations used by the simulation
 * engine.
 */
public static int getPrefNoIter(){
    return prefs.getInt("noIterations", 500);
}

/**
 * This is an accessor method for the current output type.
 *
 * @return The current preferred output type - either MacSim.OUTPUT_FILE or
 * MacSim.OUTPUT_REAL
 */
public static int getPrefOutput(){
    return prefs.getInt("defaultOut", Engine.OUTPUT_FILE);
}

/**
 * This is an accessor method for the current view type.
 *
 * @return The current preferred output type - either MacSim.VIEW_DEV or
 * MacSim.VIEW_VIEWER
 */
public static int getPrefView(){
    return prefs.getInt("defaultView", MacSim.VIEW_VIEWER);
}

/**
 * This is a set method for the current environment dimensions. It is also
 * responsible for initiating the updating of the main GUI.
 *
 * @param xDimension The maximum extent of the X dimension.
 * @param yDimension The maximum extent of the Y dimension.
 * @param zDimension The maximum extent of the Z dimension.
 */
static void setPrefEnvDimensions(int xDimension, int yDimension,
    int zDimension){
    prefs.putInt("defXDim", xDimension);
    prefs.putInt("defYDim", yDimension);
    prefs.putInt("defZDim", zDimension);

    MacSim.environment.setDimensions(xDimension, yDimension, zDimension);

    //remove old canvas as needs replacing
    MacSim.desktop.remove(MacSim.canvas);

    Point prevLoc = MacSim.canvas.getLocation();
    MacSim.canvas = new CanvasFrame();

```

```

    MacSim.desktop.add(MacSim.canvas);

    //set up the new instance
    MacSim.canvas.setLocation(prevLoc);
    MacSim.desktop.repaint();
}

/**
 * This is an accessor method for the current amount of "bins" used when
 * simulating a scenario.
 *
 * @param bins The preferred number of bins to be used by the simulation
 * engine.
 */
private static void setPrefNoBins(int bins){
    prefs.putInt("noBins", bins);
}

/**
 * This is an accessor method for the current number of iterations used
 * when simulating a scenario.
 *
 * @param iterations he preferred number of iterations to be used by the
 * simulation engine.
 */
private static void setPrefNoIter(int iterations){
    prefs.putInt("noIterations", iterations);
}

/**
 * This is an accessor method for the current output type.
 *
 * @param output The preferred output mode.
 */
private static void setPrefOutput(int output){
    prefs.putInt("defaultOut", output);
}

/**
 * This is an accessor method for the current viewing mode.
 *
 * @param view The preferred viewing mode.
 */
public static void setPrefView(int view){
    prefs.putInt("defaultView", view);
    MacSim.update(view);
    FrameMenuBar.update(view);
}

/**
 * Sets all the fields of the preferences frame from the correct underlying
 * values.
 */
private void setViewFromPrefs(){
    defXDimField.setText(Integer.toString(prefs.getInt("defXDim", 500)));
    defYDimField.setText(Integer.toString(prefs.getInt("defYDim", 500)));
    defZDimField.setText(Integer.toString(prefs.getInt("defZDim", 500)));
    defNoBinsField.setText(Integer.toString(prefs.getInt("noBins", 10)));
    defNoIterField.setText(Integer.toString(prefs.getInt("noIterations", 500)));
    defModeField.setSelectedIndex(prefs.getInt("defaultView",
        MacSim.VIEW_VIEWER));
    defOutField.setSelectedIndex(prefs.getInt("defaultOut",
        Engine.OUTPUT_FILE));
}
}

```

```

package common;

import gui.MacSim;
import gui.RunMacs;

import java.awt.Dimension;
import java.awt.Point;

import javax.swing.JDialog;
import javax.swing.JProgressBar;

/**
 * This class provides a simple progress bar for the application.
 *
 * @author Iain Robinson
 * @version 1.0
 */
public class ProgressDialog extends JDialog{
    /** The underlying progress bar. */
    private static JProgressBar progressBar;

    /**
     * Creates a new ProgressDialog object.
     *
     * @param name The title that will appear on the progress dialog.
     * @param upperLimit The maximum value that the progress can measure.
     */
    public ProgressDialog(String name, int upperLimit){
        progressBar = new JProgressBar(0, upperLimit);
        progressBar.setValue(0);
        progressBar.setStringPainted(true);

        setSize(new Dimension(200, 60));
        setResizable(false);

        Point appLocation = RunMacs.inst.getLocation();
        this.setLocation(appLocation.x +
            ((RunMacs.inst.getWidth() - this.getWidth()) / 2),
            appLocation.y +
            ((RunMacs.inst.getHeight() - this.getHeight()) / 2));
        setTitle(name);
        getContentPane().add(progressBar);
        setVisible(true);
    }

    /**
     * This method is used to update the progress bar display.
     *
     * @param newVal The current amount of progress.
     */
    public void update(int newVal){
        progressBar.setValue(newVal);

        if(newVal == progressBar.getMaximum()){
            //clean up and restore media player control.. FIXME Why here...Engine
            doesnt seem to work
            MacSim.output.end();
            MacSim.engine.end();
            setVisible(false);
        }
    }
}

```

```

package datatypes;

import common.ErrorReporter;
import common.MacsPreferences;

/**
 * This class provides added functionality to the Thread class that allows the
 * easy ability to pause, resume and "step" over the underlying Thread's run()
 * method.
 *
 *
 * @author Iain Robinson
 */
public abstract class AbstractPlayable extends Thread{
    /** Indicates if the thread is still running. */
    public boolean isAlive = true;

    /** Indicates if the thread has been paused. */
    public boolean paused = false;

    /** Indicates if the thread is being stepped rather than run. */
    public boolean stepping = false;

    /**
     * A count of the number of iterations already processed by the player
     * object.
     */
    protected int count = 0;

    /**
     * This method performs one iteration of a run.
     */
    public abstract void play();

    /**
     * This method start execution of the object.
     */
    public void run(){
        // Keep looping until told to stop
        // TODO try to remove the getPrefNoIter call
        while(isAlive && (count < MacsPreferences.getPrefNoIter())){
            if(!paused){
                count++;
                play();

                if(stepping){
                    pause();
                }
            }else{
                try{
                    synchronized(this){
                        this.wait();
                    }
                }catch(InterruptedException e){
                }
            }

            try{
                Thread.sleep(10); // Pause between each iteration
            }catch(InterruptedException error){
                ErrorReporter.report(ErrorReporter.THREAD_INTERRUPTED);
            }
        }

        end();
    }
}

```

```
/**
 * This method determines what actions occur after a run has ended but
 * before the object is disposed of.
 */
public abstract void cleanup();

/**
 * Ends execution of the object.
 */
public abstract void end();

/**
 * This method performs one iteration of the play method.
 */
public abstract void step();

/**
 * This method stops the object from being in the stepping state.
 */
public abstract void unStep();

/**
 * This method pauses the run method until the unPause method is called.
 */
public abstract void pause();

/**
 * This method stops the paused state initiated by the pause() method.
 */
public abstract void unPause();
}
```

```

package datatypes;

import backend.Engine;

/**
 * This class provides the basic framework of any viewer that can be used with
 * the MACS system.
 *
 * @author Iain Robinson
 * @version 1.0
 */
public abstract class AbstractViewer extends AbstractPlayable{
    /**
     * This is the instance of the Engine class that the AbstractViewer uses to
     * get its data from.
     */
    protected Engine source;

    /**
     * Creates a new AbstractViewer object.
     *
     * @param engine The source of the simulation data to be viewed.
     */
    protected AbstractViewer(Engine engine){
        source = engine;
    }

    /**
     * This method performs one iteration of a run.
     */
    public abstract void play();

    /**
     * This method determines what actions occur after a run has ended but
     * before the object is disposed of.
     */
    public abstract void cleanup();

    /**
     * Ends execution of the object.
     */
    public void end(){
        isAlive = false;
        cleanup();
    }

    /**
     * This method performs one iteration of the play method.
     */
    public void step(){
        stepping = true;
        unPause();
    }

    /**
     * This method stops the object from being in the stepping state.
     */
    public void unStep(){
        stepping = false;
    }

    /**
     * This method pauses the run method until the unpause method is called.
     */
    public void pause(){

```

```
        paused = true;
    }

    /**
     * This method stops the paused state initiated by the pause() method.
     */
    public void unPause(){
        paused = false;

        synchronized(this){
            this.notify();
        }
    }
}
```

```

package datatypes;

import backend.Engine;

/**
 * This class provides the basic framework of any viewer that can be used with
 * the MACS system.
 *
 * @author Iain Robinson
 * @version 1.0
 */
public abstract class AbstractViewer extends AbstractPlayable{
    /**
     * This is the instance of the Engine class that the AbstractViewer uses to
     * get its data from.
     */
    protected Engine source;

    /**
     * Creates a new AbstractViewer object.
     *
     * @param engine The source of the simulation data to be viewed.
     */
    protected AbstractViewer(Engine engine){
        source = engine;
    }

    /**
     * This method performs one iteration of a run.
     */
    public abstract void play();

    /**
     * This method determines what actions occur after a run has ended but
     * before the object is disposed of.
     */
    public abstract void cleanup();

    /**
     * Ends execution of the object.
     */
    public void end(){
        isAlive = false;
        cleanup();
    }

    /**
     * This method performs one iteration of the play method.
     */
    public void step(){
        stepping = true;
        unPause();
    }

    /**
     * This method stops the object from being in the stepping state.
     */
    public void unStep(){
        stepping = false;
    }

    /**
     * This method pauses the run method until the unpause method is called.
     */
    public void pause(){

```

```
        paused = true;
    }

    /**
     * This method stops the paused state initiated by the pause() method.
     */
    public void unPause(){
        paused = false;

        synchronized(this){
            this.notify();
        }
    }
}
```

```

package datatypes;

/**
 * This class represents a single attribute of an agent within the MACS system
 * and is partly responsible for the flexibility of the system. An attribute
 * is made up of an attribute name/value pair and an indication of the type of
 * the value. The type can be either an integer, double, string or boolean
 * value (covering the basic data types of most languages) and the Attribute
 * class provides methods that attempt to give sensible results when
 * interpreting the stored value regardless of its underlying type. Although
 * not initially planned it would be quite simple for a user to extend the
 * functionality of this class since the underlying type used to store the
 * data value is a Java Object type - therefore allowing user define data
 * types to be easily incorporated into the class design.
 *
 * @author Iain Robinson
 */
public class Attribute{
    /** Indicates the type of the attribute value is an integer. */
    public static final int TYPE_INT = 0;

    /** Indicates the type of the attribute value is an String. */
    public static final int TYPE_STRING = 1;

    /** Indicates the type of the attribute value is a double. */
    public static final int TYPE_DOUBLE = 2;

    /** Indicates the type of the attribute value is a boolean. */
    public static final int TYPE_BOOLEAN = 3;

    /** The value of the attribute. */
    private Object value;

    /** The name of the attribute. */
    private String name;

    /**
     * Indicates if this attribute should be included in any data passed to a
     * viewer.
     */
    private boolean indexed;

    /** The data type of the attribute */
    private int type;

    /**
     * Creates a new instance of Attribute that contains an Object type. The
     * object could, and should, be a decendant such as Integer, Double,
     * String etc.
     *
     * @param name The name of the attribute.
     * @param value The value of the attribute.
     * @param type The type of the attribute.
     * @param indexed True if the attribute should be passed to a viewer, false
     * if not.
     */
    public Attribute(String name, Object value, int type, boolean indexed){
        this.name = name;
        this.value = value;
        this.type = type;
        this.indexed = indexed;
    }

    /**
     * This method gets the name of the attribute as a string.
     */
}

```

```

    * @return The name of the attribute.
    */
    public String getName(){
        return name;
    }

    /**
     * This method determines the type of the underlying attribute value.
     *
     * @return An integer representing the type of the underlying value.
     */
    public int getType(){
        return type;
    }

    /**
     * This method attempts to return a sensible integer value - based on the
     * attribute type.
     *
     * @return An integer value of the the underlying attribute value.
     */
    public int getIntValue(){
        int ret = 0;

        switch(getType()){
            case TYPE_INT:
                ret = ((Integer) value).intValue();

                break;

            case TYPE_STRING:

                try{
                    ret = Integer.parseInt((String) value);
                }catch(NumberFormatException e){
                }

                break;

            case TYPE_DOUBLE:
                ret = ((Double) value).intValue();

                break;

            case TYPE_BOOLEAN:

                boolean bool = ((Boolean) value).booleanValue();

                //if true
                if(bool){
                    ret = 1;
                }else{
                    ret = 0;
                }

                break;

            default:
                return ret;
        }

        return ret;
    }

    /**
     * This method attempts to return a sensible String value - based on the

```

```

* attribute type.
*
* @return A String value of the the underlying attribute value.
*/
public String getStringValue(){
    String ret = "";

    switch(getType()){
    case TYPE_INT:
        ret = ((Integer) value).toString();

        break;

    case TYPE_STRING:
        ret = (String) value;

        break;

    case TYPE_DOUBLE:
        ret = ((Double) value).toString();

        break;

    case TYPE_BOOLEAN:
        ret = value.toString();

        break;

    default:
        return ret;
    }

    return ret;
}

/**
* This method attempts to return a sensible double value - based on the
* attribute type.
*
* @return A double value of the the underlying attribute value.
*/
public double getDoubleValue(){
    double ret = 0.0;

    switch(getType()){
    case TYPE_INT:
        ret = ((Integer) value).doubleValue();

        break;

    case TYPE_STRING:

        try{
            ret = Double.parseDouble((String) value);
        }catch(NumberFormatException err){
        }

        break;

    case TYPE_DOUBLE:
        ret = ((Double) value).doubleValue();

        break;

    case TYPE_BOOLEAN:

```

```

        boolean bool = ((Boolean) value).booleanValue();

        //if true
        if(bool){
            ret = 1.0;
        }else{
            ret = 0.0;
        }

        break;

    default:
        return ret;
    }

    return ret;
}

/**
 * This method attempts to return a sensible boolean value - based on the
 * attribute type.
 *
 * @return A boolean value of the the underlying attribute value.
 */
public boolean getBooleanValue(){
    boolean ret = false;

    switch(getType()){
        case TYPE_INT:

            int tempint = ((Integer) value).intValue();

            if(tempint >= 1){
                ret = true;
            }else{
                ret = false;
            }

            break;

        case TYPE_STRING:

            //TODO check test results for this method
            //consider revising the use of parse boolean
            ret = Boolean.parseBoolean((String) value);

            break;

        case TYPE_DOUBLE:

            int temp2 = ((Double) value).intValue();

            if(temp2 >= 1){
                ret = true;
            }else{
                ret = false;
            }

            break;

        case TYPE_BOOLEAN:
            ret = ((Boolean) value).booleanValue();

            break;

        default:

```

```

        return ret;
    }

    return ret;
}

/**
 * Replaces the current value of the attribute (if one exists).
 *
 * @param value The new value of the attribute.
 * @param type The type of the new value.
 */
public void setValue(Object value, int type){
    this.value = value;
    this.type = type;
}

/**
 * Determines whether or not the attributed should be included in any
 * output to a AbstractViewer.
 *
 * @return True if the attribute should be included in output - false
 *         otherwise.
 */
public boolean isIndexed(){
    return indexed;
}

/**
 * This method creates an identical, but totally distinct copy of the
 * instance it is called upon.
 *
 * @return A true copy of the original instance.
 */
public Attribute getDeepCopy(){
    Attribute copy = new Attribute(name, value, type, indexed);

    return copy;
}
}

```

```

package datatypes;

import java.util.Vector;

/**
 * This class is an extension to the Vector class that overrides a number of
 * methods so that it is more suited to being used within the MACS
 * application. It enforces uniqueness amongst the attributes it holds.
 *
 * @author Iain Robinson
 */
public class AttributeList extends Vector{
    /**
     * Creates a new AttributeList object.
     *
     * @param cap The initial capacity of the list.
     * @param inc The number of spaces to increase the size of the list by when
     *            needed.
     */
    public AttributeList(int cap, int inc){
        super(cap, inc);
    }

    /**
     * This method add a specified attribute. If an attribute with the same
     * name is already present it simply replaces to old attribute with the
     * new one.
     *
     * @param attrib The attribute to be added.
     */
    public synchronized void add(Attribute attrib){
        String aName = attrib.getName();
        Attribute current = get(aName);

        //if already present then replace
        if(current != null){
            int index = indexOf(current);
            set(index, attrib);
        }else{
            //otherwise add new attribute;
            this.addElement(attrib);
        }
    }

    /**
     * This method overrides the same method in the extended Vector class to
     * provide thread safe access to data elements held by the class.
     *
     * @param index The index of the object to be obtained.
     *
     * @return The object at the specified index.
     */
    public synchronized Object get(int index){
        return super.get(index);
    }

    /**
     * This method overrides the same method in the extended Vector class to
     * provide thread safe access to data elements held by the class.
     *
     * @param obj The object to be added to the class
     *
     * @return true if the object was added and false otherwise.
     */
    public synchronized boolean add(Object obj){

```

```

        return super.add(obj);
    }

    /**
     * This method returns an attribute matching the given attribute name.
     *
     * @param name The name of the attribute.
     *
     * @return The attribute if it exists - null otherwise.
     */
    public synchronized Attribute get(String name){
        for(int i = 0;i < size();i++){
            Attribute attrib = (Attribute) get(i);

            if(attrib.getName().matches(name)){
                return attrib;
            }
        }

        return null;
    }

    /**
     * This method creates an identical, but totally distinct copy of the
     * instance it is called upon.
     *
     * @return A true copy of the original instance.
     */
    public AttributeList getDeepCopy(){
        AttributeList copy = new AttributeList(0, 1);

        for(int i = 0;i < this.size();i++){
            Attribute elementCopy = ((Attribute) this.get(i)).getDeepCopy();
            copy.add(elementCopy);
        }

        return copy;
    }

    /**
     * This method provides a list of all Attributes within the object whos
     * isIndexed() method retruns true.
     *
     * @return A list of all indexed attributes contained in the object.
     */
    public AttributeList getIndexed(){
        AttributeList indexed = new AttributeList(0, 1);

        for(int attIndex = 0;attIndex < size();attIndex++){
            Attribute currentAtt = (Attribute) get(attIndex);

            if(currentAtt.isIndexed()){
                indexed.add(currentAtt);
            }
        }

        return indexed;
    }
}

```

```

package datatypes;

import backend.BinManager;

import common.ErrorReporter;

import java.awt.Dimension;
import java.awt.Image;
import java.awt.MediaTracker;
import java.awt.Point;
import java.awt.Toolkit;
import java.awt.datatransfer.DataFlavor;
import java.awt.datatransfer.Transferable;
import java.awt.datatransfer.UnsupportedFlavorException;

import javax.swing.ImageIcon;
import javax.swing.JLabel;

/**
 * This class represents the most basic entity in a Macs simulation and must be
 * extended correctly by any user written class wishing to be used operate
 * within the Macs system. It provides the essential elements for building a
 * custom agent.
 *
 * @author Iain Robinson
 */
public class BaseEntity extends JLabel implements Transferable{
    /** A list of all the attributes that the Entity possesses. */
    private AttributeList attributes = new AttributeList(0, 1);

    /**
     * The bin used by the simulation engine that currently contains this
     * entity.
     */
    public BaseEntityList currentBin;

    /** An array containing all the data flavors supported by this class. */
    private DataFlavor[] flavors;

    /**
     * The dataflavor of the entity. This is used by the drag & dropinterface
     * to correctly establish the required type of data to be used by the
     * GUI's drag & drop methods.
     */
    private DataFlavor iconFlavor;

    /**
     * The image used to represent the entity (usually a 32x32 GIF image). To
     * correctly locate the file the image must have the same name as the
     * entity it represents and must be located within the same directory as
     * the corresponding class file.
     */
    public Image image;

    /**
     * A generic constructor to be used by all extending classes in their own
     * constructors via a call to super().
     *
     * @param iconFlavor A flavor specified by the extending class to uniquely
     * identify itself.
     * @param fileName A GIF image specified by the extending class to
     * represent itself within the Macs system.
     */
    protected BaseEntity(DataFlavor iconFlavor, String fileName){
        flavors = new DataFlavor[1];

```

```

    flavors[0] = iconFlavor;
    this.iconFlavor = iconFlavor;

    Toolkit toolkit = Toolkit.getDefaultToolkit();
    Image image = toolkit.getImage(fileName);

    setPreferredSize(new Dimension(100, 32));
    setSize(getPreferredSize());

    String className = getClass().getName();
    String[] tokens = className.split("\\.");
    String title = tokens[2];

    /*
     * sets the name of the class BUT only when it is on the palette - it is
     * cleared when an instance is dropped on the Canvas frame inside that
     * objects drop method
     */
    this.setText(title);

    ImageIcon icon = new ImageIcon(fileName);

    if(icon.getImageLoadStatus() == MediaTracker.COMPLETE){
        this.setIcon(icon);
    }
}

/**
 * This method can be used to add an attribute to a specific entity. If an
 * attribute with the same name is already present it is replaced.
 *
 * @param attrib The attribute to add.
 */
public void addAttribute(Attribute attrib){
    attributes.add(attrib);
}

/**
 * This method sets the location of an entity - Both in terms of its
 * graphical representation and its underlying data model.
 *
 * @see java.awt.Component#setLocation(java.awt.Point)
 */
public void setLocation(Point location){
    //      minus 16 dues to image size being 32x32
    // we wish it to center on the entity
    super.setLocation(location.x, location.y);

    //TODO these need to be rethought
    Attribute xpos = getAttribute("xPosition");
    xpos.setValue(new Integer(location.x + 16), Attribute.TYPE_INT);

    Attribute ypos = getAttribute("yPosition");
    ypos.setValue(new Integer(location.y + 16), Attribute.TYPE_INT);
}

/**
 * This method returns a list of all the attributes currently possessed by
 * the entity.
 *
 * @return A list of the entity's current attributes.
 */
public synchronized AttributeList getAttributes(){
    return attributes;
}

```

```

/**
 * This method returns a single entity attribute based on the attribute
 * name.
 *
 * @param attributeName The name of the attribute.
 *
 * @return A valid attribute if one is found in the entity that has the
 *         given name - null otherwise.
 */
public synchronized Attribute getAttribute(String attributeName){
    return attributes.get(attributeName);
}

/**
 * Used during drag & drop operations this returns an object instance
 * corresponding to the type of data specified by the given flavor.
 *
 * @param flavor The required type of the object.
 *
 * @return An instance of the required data flavor cast as an Object.
 *
 * @throws UnsupportedOperationException If the flavor of the data cannot be
 *         supported by this class or the extending class.
 */
public synchronized Object getTransferData(DataFlavor flavor)
    throws UnsupportedOperationException{
    if(flavor.equals(iconFlavor)){
        return this;
    }

    throw new UnsupportedOperationException(flavor);
}

/**
 * Used during drag & drop operations this method returns an array of all
 * data flavors supported by this class.
 *
 * @return An array of all data flavors supported by this class.
 */
public synchronized DataFlavor[] getTransferDataFlavors(){
    return flavors;
}

/**
 * Check to see if this class supports a specified data flavor.
 *
 * @param flavor The flavor of data to check for support
 *
 * @return true if the specified data flavor is supported by this class -
 *         false otherwise.
 */
public boolean isDataFlavorSupported(DataFlavor flavor){
    return flavor.equals(iconFlavor);
}

/**
 * Allows the Entity to quiz the engine when a simulation is running and to
 * get all other entities within an area defined by its "sightRange"
 * attribute.
 *
 * @return A list of all other entities that fall within the "sightRange"
 *         of the entity.
 */
public BaseEntityList lookAround(){
    BaseEntityList surrounding = new BaseEntityList(0, 1);

```

```

    int range = this.getAttribute("sightRange").getIntValue();

    surrounding = BinManager.getSurroundingEntities(this, range);

    return surrounding;
}

/**
 * This method should be overridden by any extending class and should
 * contain all code that must be executed once every cycle of the
 * simulation engine when a simulation is running. Although the method
 * here could be declared abstract this would cause the whole class to be
 * in a "abstract class extends non-abstract class" relationship and this
 * was avoided for this reason.
 */
public void run(){
}

/**
 * This method sets the new bin that the entity can be found in.
 *
 * @param newBinValue The new bin the entity is to be found in.
 */
public void setBin(BaseEntityList newBinValue){
    currentBin = newBinValue;
}

/**
 * This method will cause a "wraparound" to be enforced on an entity's
 * current location in space ensuring that it does not go outside the
 * bounds set by the three specified limits. Minimum limits are considered
 * to be zero.
 *
 * @param xLimit The maximum limit of the X dimension.
 * @param yLimit The maximum limit of the Y dimension.
 * @param zLimit The maximum limit of the Z dimension.
 */
public void limitLocation(int xLimit, int yLimit, int zLimit){
    String xPosition = "xPosition";
    String yPosition = "yPosition";
    String zPosition = "zPosition";

    limitAttribute(0, xLimit, xPosition, Attribute.TYPE_INT);
    limitAttribute(0, yLimit, yPosition, Attribute.TYPE_INT);
    limitAttribute(0, zLimit, zPosition, Attribute.TYPE_INT);
}

/**
 * This method will enforce a "wraparound" on any specified attribute to
 * ensure that it falls within the range lowLimit -> highLimit.
 *
 * @param lowLimit The minimum possible value of the attribute.
 * @param highLimit The maximum possible value of the attribute.
 * @param attribute The name of the attribute to be limited.
 * @param type The type of the attribute to be limited.
 */
private void limitAttribute(int lowLimit, int highLimit, String attribute,
int type){
    while(getAttribute(attribute).getIntValue() < lowLimit){
        Attribute newValue = new Attribute(attribute,
            new Integer(getAttribute(attribute).getIntValue() +
                highLimit), type, true);
        addAttribute(newValue);
    }

    while(getAttribute(attribute).getIntValue() > highLimit){

```

```

        Attribute newValue = new Attribute(attribute,
            new Integer(getAttribute(attribute).getIntValue() -
                highLimit), type, true);
        addAttribute(newValue);
    }
}

/**
 * This method provides access to a list of all the entity's current
 * attributes with their respective values.
 *
 * @return A String representation of all the entity's current attributes.
 */
public String printAttributes(){
    String out = "";

    for(int i = 0;i < attributes.size();i++){
        Attribute current = (Attribute) attributes.get(i);

        out = out + "Attribute Name: " + current.getName() + "\tValue: " +
            current.getStringValue() + "\n";
    }

    return out;
}

/**
 * Sets the list of attributes of the entity to the specified
 * AttributeList.
 *
 * @param atts The desired list of attributes.
 */
public void setAttributes(AttributeList atts){
    attributes = atts;
}

/**
 * This method provides an identical, but totally distinct, copy of the
 * instance on which it is called.
 *
 * @return A unique copy of the object.
 */
public BaseEntity getDeepCopy(){
    BaseEntity copy = null;

    try{
        copy = (BaseEntity) this.getClass().newInstance();
    }catch(IllegalAccessException error){
        ErrorReporter.report(ErrorReporter.FAILED_INSTANTIATION);
    }catch(InstantiationException error){
        ErrorReporter.report(ErrorReporter.FAILED_INSTANTIATION);
    }
}

    copy.image = this.image;
    copy.currentBin = this.currentBin;
    copy.flavors = this.flavors;
    copy.iconFlavor = this.iconFlavor;

    AttributeList attributesCopy = attributes.getDeepCopy();
    copy.attributes = attributesCopy;

    return copy;
}
}

```

```

package datatypes;

import java.util.Vector;

/**
 * This class is an extension to the Vector class that overrides a number of
 * methods so that it is more suited to being used within the MACS
 * application. It enforces uniqueness amongst the entities it holds and has a
 * containsSpecies method that allows a user to check for the presence of a
 * certain species type.
 *
 * @author Iain Robinson
 */
public class BaseEntityList extends Vector{
    /**
     * Creates a new BaseEntityList object.
     */
    public BaseEntityList(){
        super();
    }

    /**
     * Creates a new BaseEntityList object.
     *
     * @param cap The initial capacity of the list.
     * @param inc The number of spaces to increase the size of the list by when
     *            needed.
     */
    public BaseEntityList(int cap, int inc){
        super(cap, inc);
    }

    /**
     * This method allows an entity to be added to the list only if it is not
     * already present.
     *
     * @param entity The entity to be added.
     */
    public void add(BaseEntity entity){
        //check for uniqueness
        if(!this.contains(entity)){
            addElement(entity);
        }
    }

    /**
     * This method indicates if a particular species is already present within
     * the list.
     *
     * @param species The species to be looked for.
     *
     * @return true if the species is present - false if not.
     */
    public boolean containsSpecies(BaseEntity species){
        String ident = species.getClass().getName();

        for(int i = 0; i < size(); i++){
            BaseEntity candidate = (BaseEntity) get(i);

            if(candidate.getClass().getName().equals(ident)){
                return true;
            }
        }

        return false;
    }
}

```

```
}

/**
 * This method creates an identical, but totally distinct copy of the
 * instance it is called upon.
 *
 * @return A true copy of the original instance.
 */
public BaseEntityList getDeepCopy(){
    BaseEntityList copy = new BaseEntityList(0, 1);

    for(int i = 0;i < this.size();i++){
        BaseEntity elementCopy = ((BaseEntity) this.get(i)).getDeepCopy();
        copy.add(elementCopy);
    }

    return copy;
}
}
```

```

package datatypes;

import java.io.Serializable;

/**
 * This class holds all information regarding the environment used with a Macs
 * simulation such as the x,y and z dimension limits and the details of any
 * agents that have been positioned within it.
 *
 * @author Iain Robinson
 */
public class Environment implements Serializable{
    /**
     * A List of BaseEntity instances that represent the entire population of
     * the environment.
     */
    public BaseEntityList population = new BaseEntityList();

    /**
     * The current X dimension "limit" of the environment - this is not fixed
     * but simply provides sensible numbers to the gui at the start of the
     * simulation.
     */
    public int xLimit = 1000;

    /**
     * The current Y dimension "limit" of the environment - this is not fixed
     * but simply provides sensible numbers to the gui at the start of the
     * simulation.
     */
    public int yLimit = 1000;

    /**
     * The current Z dimension "limit" of the environment - this is not fixed
     * but simply provides sensible numbers to the gui at the start of the
     * simulation.
     */
    public int zLimit = 1000;

    /**
     * Creates a new Environment object.
     */
    public Environment(){
    }

    /**
     * Adds an entity to the population of the environment.
     *
     * @param entity The entity to add to the population
     */
    public void addEntity(BaseEntity entity){
        population.add(entity);
    }

    /**
     * This method removes a specific entity from the current environment.
     *
     * @param entity The entity to be removed from the environment.
     */
    public void removeEntity(BaseEntity entity){
        population.removeElement(entity);
    }

    /**
     * This method sets the X, Y and Z dimensions of the environment.
     *
     * @param xCoord The maximum value of the X Dimension.
     * @param yCoord The maximum value of the Y Dimension.
     * @param zCoord The maximum value of the Z Dimension.
     */
    public void setDimensions(int xCoord, int yCoord, int zCoord){
        xLimit = xCoord;
        yLimit = yCoord;
        zLimit = zCoord;
    }

    /**
     * This method provides an exact, but totally distincy copy of the
     * environment object.
     *
     * @return A copy of the environment object.
     */
    public Environment getDeepCopy(){
        Environment copy = new Environment();
        copy.setDimensions(this.xLimit, this.yLimit, this.zLimit);
        copy.population = this.population.getDeepCopy();

        return copy;
    }
}

```

```

package gui;

import common.ErrorReporter;

import datatypes.Attribute;
import datatypes.BaseEntity;
import datatypes.BaseEntityList;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics2D;
import java.awt.Insets;
import java.awt.Point;
import java.awt.Rectangle;
import java.awt.datatransfer.DataFlavor;
import java.awt.datatransfer.Transferable;
import java.awt.datatransfer.UnsupportedFlavorException;
import java.awt.dnd.DnDConstants;
import java.awt.dnd.DropTarget;
import java.awt.dnd.DropTargetDragEvent;
import java.awt.dnd.DropTargetDropEvent;
import java.awt.dnd.DropTargetEvent;
import java.awt.dnd.DropTargetListener;
import java.awt.event.ComponentAdapter;
import java.awt.event.ComponentEvent;
import java.awt.event.InputEvent;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.event.MouseMotionListener;

import java.beans.PropertyVetoException;

import java.io.IOException;

import java.util.Hashtable;

import javax.swing.JInternalFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.event.MouseInputAdapter;

/**
 * This class represents the main 2D, top-down, view of the environment in its
 * static or initial state.
 *
 * @author Iain Robinson
 */
public class CanvasFrame extends JInternalFrame implements DropTargetListener{
    /** The actual view of the environment. */
    private static JPanel canvas = new JPanel(null);

    /**
     * Indicates a random distribution of entities. Used when adding a group of
     * entities to the canvas.
     */
    public static final int DIST_RANDOM = 0;

    /**
     * Indicates a regular distribution of entities. Used when adding a group
     * of entities to the canvas.
     */
    public static final int DIST_REGULAR = 1;

    /** The currently selected entities */
    private static BaseEntityList selection = new BaseEntityList();

```

```

/**
 * The positions of all entities on the canvas. Used to ensure smoothness
 * when moving.
 */
private static Hashtable compPos = new Hashtable();

/**
 * The listener for this class and all classes contained by it (such as
 * BaseEntities). Determines what action has been selected by the user.
 */
private static MouseInputAdapter mouseSensor = new MouseInputAdapter(){
    /**
     * Indicates that user events should be considered as being done to
     * a group rather than to a single entity.
     */
    private static final int MODE_GROUP = 2;

    /**
     * Indicates that user has selected to distribute entities in the
     * environment.
     */
    private static final int MODE_DISTRIB = 1;

    /**
     * Indicates that user events should be considered as being done to
     * a single entity rather than to a group.
     */
    private static final int MODE_SINGLE = 3;

    /**
     * Indicates that user has selected to select entities in the
     * environment.
     */
    private static final int MODE_SELECT = 0;

    /** The current mode that the CanvasFrame is being used in. */
    private int currentMode = MODE_SELECT;

    /**
     * The top-left corner of the selection area specified by
     * selectedArea.
     */
    private Point selectedTL = null;

    /**
     * The bottom-right corner of the selection area specified by
     * selectedArea.
     */
    private Point selectedBR = null;

    /**
     * This method determines what action to take if the mouse has been
     * dragged within the CanvasFrame.
     *
     * @param event The event that triggered the action.
     */
    public void mouseDragged(MouseEvent event){
        // if this is a panel that has been dragged over...
        if(JPanel.class.isInstance(event.getComponent())){
            // do stuff common to both modes
            selectedBR = event.getPoint();

            CanvasFrame.refreshSelectionBox(selectedTL, selectedBR);

            // if this is a selection...

```

```

        if(currentMode == MODE_SELECT){
            // first clear the current selection
            CanvasFrame.clearSelection();

            //then get all entities within selection area and add to the
            // selection
            //TODO this code is inefficient as it clears stuff already
            // added that will
            //just need to be added once again...consider revising
            for(int i = 0;i < MacSim.environment.population.size();
                i++){
                BaseEntity target = (BaseEntity)
MacSim.environment.population.get(i);

                if(CanvasFrame.selectedArea.contains(
                    target.getLocation())){
                    CanvasFrame.addToSelection(target);
                }
            }
        }
        //otherwise we must be attempting to move 1 or more entity icons
        else{
            //the component used here just is the underlying graphical
            //representation of the BaseEntity class - which could be
            //used here instead just as easily
            BaseEntity comp = (BaseEntity) event.getComponent();

            //get the entity clicked on and use this as a base to alter the
            //deltas of ALL selected entities
            Point oldPoint = (Point) CanvasFrame.compPos.get(comp);
            Point newPoint = event.getPoint();

            int deltax = newPoint.x - oldPoint.x;
            int deltax = newPoint.y - oldPoint.y;

            if(currentMode == MODE_SINGLE){
                moveOne(comp, deltax, deltax);
            }

            if(currentMode == MODE_GROUP){
                moveSelection(deltax, deltax);
            }
        }
    } //end MouseDragged method

/**
 * This method moves all selected entities by the specified amounts
 * in the X and Y dimensions.
 *
 * @param deltax The amount to move each entity by in the X
 * dimension.
 * @param deltax The amount to move each entity by in the Y
 * dimension.
 */
private void moveSelection(int deltax, int deltax){
    //alter all selected entities positions
    BaseEntityList selection = CanvasFrame.getSelection();

    for(int i = 0;i < selection.size();i++){
        BaseEntity base = (BaseEntity) selection.get(i);
        Point compLoc = base.getLocation();

        compLoc.x += deltax;
        compLoc.y += deltax;
        base.setLocation(compLoc);
    }
}

```

```

    }
}

/**
 * This method moves the specified entity by the specified amounts
 * in the X and Y dimensions.
 *
 * @param comp The entity to be moved.
 * @param deltax The amount to move each entity by in the X
 * dimension.
 * @param deltax The amount to move each entity by in the Y
 * dimension.
 */
private void moveOne(BaseEntity comp, int deltax, int deltax){
    Point compLoc = comp.getLocation();

    compLoc.x += deltax;
    compLoc.y += deltax;
    comp.setLocation(compLoc);
}

/**
 * This method determines what action to take if the mouse has been
 * pressed within the CanvasFrame.
 *
 * @param event The event that triggered the action.
 */
public void mousePressed(MouseEvent event){
    CanvasFrame.compPos.put(event.getComponent(), event.getPoint());

    // if the background has been clicked on...
    if(JPanel.class.isInstance(event.getComponent())){
        // work out what mode we're in
        if((event.getModifiersEx() & InputEvent.BUTTON3_DOWN_MASK) ==
InputEvent.BUTTON3_DOWN_MASK){
            currentMode = MODE_DISTRIB;
        }

        if((event.getModifiersEx() & InputEvent.BUTTON1_DOWN_MASK) ==
InputEvent.BUTTON1_DOWN_MASK){
            currentMode = MODE_SELECT;
        }

        // then do stuff common to both modes..
        // clear all previously selected components
        CanvasFrame.clearSelection();

        //and get the point that was clicked on
        selectedTL = event.getPoint();
    }else{ //.... otherwise it must have been a entity icon

        // work out what mode we're in
        // TODO improve ...make if(shift){if(button1)}
        if((event.getModifiersEx() & InputEvent.BUTTON1_DOWN_MASK) ==
InputEvent.BUTTON1_DOWN_MASK){
            currentMode = MODE_SINGLE;
        }

        if((event.getModifiersEx() & InputEvent.SHIFT_DOWN_MASK) ==
InputEvent.SHIFT_DOWN_MASK){
            currentMode = MODE_GROUP;
        }

        if((event.getModifiersEx() & InputEvent.BUTTON3_DOWN_MASK) ==
InputEvent.BUTTON3_DOWN_MASK){
            Point loc = event.getPoint();

```

```

        new ContextMenu(event.getComponent(), loc.x, loc.y);
    }

    // so get the entity
    BaseEntity member = (BaseEntity) event.getComponent();

    // if the shift key WAS pressed then just add this entity to the
    // list of selected entities
    if(currentMode == MODE_GROUP){
        // check entity is not already selected
        if(member.isEnabled()){
            CanvasFrame.addToSelection(member);
        }
    }

    // otherwise make this entity the only selected one
    if(currentMode == MODE_SINGLE){
        CanvasFrame.clearSelection();
        CanvasFrame.addToSelection(member);
    }
} // end else not CanvasFrame
}

/**
 * This method dictates what will happen when the user releases the
 * mouse button when the CanvasFrame has focus.
 *
 * @param event The event that triggered the method.
 */
public void mouseReleased(MouseEvent event){
    //if the background has been the target
    if(JPanel.class.isInstance(event.getComponent())){
        // do stuff common to both modes...
        // ...draw the selected area
        JPanel canvas = (JPanel) event.getComponent();
        Graphics2D graphics = (Graphics2D) canvas.getGraphics();
        graphics.setColor(Color.WHITE);
        graphics.draw(CanvasFrame.selectedArea);

        //TODO the following lookup is very inefficient
        //redraw all other icons
        int popCount = MacSim.environment.population.size();

        for(int i = 0; i < popCount; i++){
            ((BaseEntity) MacSim.environment.population.get(i)).repaint();
        }

        //work out what mode we're in
        if(currentMode == MODE_DISTRIB){
            //TODO currently the user can just right click and use a
            // previous bounding box....how can this be avoided
            Point loc = event.getPoint();
            new ContextMenu(event.getComponent(), loc.x, loc.y);
            graphics.setColor(Color.BLACK);
            graphics.draw(CanvasFrame.selectedArea);
        }
    }
}

};

/**
 * The previously selected area - used by the drawing routine to avoid
 * glitches in drawing the selection area.
 */
private static Rectangle oldSelectBox = new Rectangle();

```

```

/**
 * The area within which all BaseEntity objects will be deemed as selected.
 */
static Rectangle selectedArea = new Rectangle(0, 0,
        MacSim.environment.xLimit, MacSim.environment.yLimit);

/**
 * A wrapper for the canvas object that allows it to fit neatly within the
 * CanvasFrame while still retaining it overall dimensions.
 */
private static JScrollPane view = new JScrollPane();

/**
 * Creates an instance of the class.
 */
public CanvasFrame(){
    super("Environment", true, false, true, true);

    //TODO get the env vars from preferences??
    canvas.setPreferredSize(new Dimension(MacSim.environment.xLimit,
        MacSim.environment.yLimit));

    this.setMaximumSize(new Dimension(MacSim.environment.xLimit,
        MacSim.environment.yLimit));

    addComponentListener(new ComponentAdapter(){
        public void componentResized(ComponentEvent event){
            CanvasFrame tmp = (CanvasFrame) event.getSource();

            //TODO still needs some work to get rid of scrollbars
            //properly
            int sbwidth = CanvasFrame.view.getVerticalScrollBar()
                .getWidth();
            int sbheight = CanvasFrame.view.getHorizontalScrollBar()
                .getHeight();

            Insets in = tmp.getInsets();

            int xoff = sbwidth + in.left + in.right + 3;
            int yoff = sbheight + in.top + in.bottom + 28;

            if(tmp.getWidth() > MacSim.environment.xLimit){
                tmp.setSize(MacSim.environment.xLimit + xoff,
                    tmp.getHeight());
            }

            if(tmp.getHeight() > MacSim.environment.yLimit){
                tmp.setSize(tmp.getWidth(),
                    MacSim.environment.yLimit + yoff);
            }

            //if the user has attempted to maximise the
            //canvasframe then just allow them to make it
            //the same size as the environment.
            if(tmp.isMaximum){
                try{
                    tmp.setMaximum(false);
                    tmp.setSize(MacSim.environment.xLimit + xoff,
                        MacSim.environment.yLimit + yoff);
                }catch(PropertyVetoException err){
                }
            }
        }
    });

    view = new JScrollPane(canvas);

```

```

view.setViewportView(canvas);
getContentPane().add(view);

// set the size of the viewing area...but not the underlying
// environment
setSize(400, 400);
setLocation(415, 0);
setVisible(true);
canvas.setBackground(Color.WHITE);

canvas.addMouseListener(getMouseListener());
canvas.addMouseMotionListener(getMouseMotionListener());

// specifies the canvas as target for D & D and this frame as listener
// for those events
new DropTarget(canvas, DnDConstants.ACTION_COPY_OR_MOVE, this);
}

/**
 * Adds a BaseEntity to the canvas so its position can be displayed
 * visually.
 *
 * @param icon The BaseEntity to add.
 * @param xLocation The X coordinate at which to place it.
 * @param yLocation The Y coordinate at which to place it.
 */
public static void addIcon(BaseEntity icon, int xLocation, int yLocation){
    icon.setText(null);
    icon.setPreferredSize(new Dimension(32, 32));
    icon.setSize(icon.getPreferredSize());

    //set the position parameters of the underlying entity
    icon.addAttribute(new Attribute("xPosition", new Integer(xLocation),
        Attribute.TYPE_INT, true));
    icon.addAttribute(new Attribute("yPosition", new Integer(yLocation),
        Attribute.TYPE_INT, true));
    icon.addAttribute(new Attribute("zPosition", new Integer(0),
        Attribute.TYPE_INT, true));

    //add it to the environment
    MacSim.environment.addEntity(icon);

    // and put it on the screen in the correct position -- only with 2
    // dimensions
    icon.setLocation(icon.getAttribute("xPosition").getIntValue() - 16,
        icon.getAttribute("yPosition").getIntValue() - 16);
    canvas.add(icon);
    icon.repaint();
    icon.addMouseListener(getMouseListener());
    icon.addMouseMotionListener(getMouseMotionListener());
    CanvasFrame.canvas.repaint();
}

/**
 * This method adds a number of entities to the canvas.
 *
 * @param noEntities The number of entites to add.
 * @param species The type of the entities.
 * @param area The area within which to distribute the entities.
 * @param type The type of distribution. Either DIST_RANDOM or
 *             DIST_REGULAR.
 */
static void addManyIcons(int noEntities, BaseEntity species,
    Rectangle area, int type){
    for(int index = 0;index < noEntities;index++){
        Point location = new Point();

```

```

        if(type == DIST_RANDOM){
            location = generateRandomPoint(area);
        }

        if(type == DIST_REGULAR){
            location = generateRegularPoint(noEntities, area, index);
        }

        try{
            BaseEntity entity = (BaseEntity) species.getClass().newInstance();
            addIcon(entity, location.x, location.y);
        }catch(InstantiationException err3){
            ErrorReporter.report(ErrorReporter.FAILED_INSTANTIATION);
        }catch(IllegalAccessException err4){
            ErrorReporter.report(ErrorReporter.FAILED_INSTANTIATION);
        }
    }
}

/**
 * This method generates a random point within the given area.
 *
 * @param area The boundaries of the area within which the point can lie.
 *
 * @return A randomly generated point that lies within the bounds of the
 *         given input rectangle.
 */
private static Point generateRandomPoint(Rectangle area){
    Point location = new Point();
    location.x = (int) (area.x + (Math.random() * area.width));
    location.y = (int) (area.y + (Math.random() * area.height));

    return location;
}

/**
 * This method generates a regular point in a given area. Based upon the
 * number of entities being distributed in a grid form across the area.
 * Hence different values of index - from 0 to noEntities will return
 * different points on the grid.
 *
 * @param noEntities The number of entities to be distributed.
 * @param area The area within which the points can lie.
 * @param index The particular point of the grid to find.
 *
 * @return A point lying within the given area rectangle based upon an even
 *         distribution of noEntities within that space.
 */
private static Point generateRegularPoint(int noEntities, Rectangle area,
    int index){
    Point location = new Point();

    //TODO not perfect distribution should be left2right
    //not top2bottom order first
    int noFiles = (int) Math.ceil(Math.sqrt(noEntities));
    int currentRank = (index / noFiles);
    int currentFile = index;
    int spacingX = (area.width / noFiles);
    spacingX = spacingX + (spacingX / (noFiles-1));

    int spacingY = (area.height / noFiles);
    spacingY = spacingY + (spacingY / (noFiles-1));

    if((index % noFiles) == 0){

```

```

        if(index != 0){
            currentRank = (index / noFiles);
        }

        currentFile = 0;
    }else{
        while(currentFile >= noFiles){
            currentFile = currentFile - noFiles;
        }
    }

    location.x = (area.x + (spacingX * currentFile));
    location.y = (area.y + (spacingY * currentRank));

    return location;
}

/**
 * This method adds the given entity to the group of currently selected
 * entities.
 *
 * @param entity The entity to add to the selection.
 */
private static void addToSelection(BaseEntity entity){
    selection.add(entity);
    entity.setEnabled(false);
}

/**
 * This method clears the group of all currently selected entities but does
 * not remove or delete the contents of the selection.
 */
private static void clearSelection(){
    // deselects any previously selected components
    for(int i = 0;i < selection.size();i++){
        ((BaseEntity) selection.get(i)).setEnabled(true);
    }

    selection.clear();
}

/**
 * This method deletes all currently selected entities on the CanvasFrame
 * from both the gui representation and the underlying list of entities.
 */
static void removeSelection(){
    for(int i = 0;i < selection.size();i++){
        BaseEntity current = (BaseEntity) selection.get(i);
        MacSim.environment.removeEntity(current);
        canvas.remove(current);
    }

    selection.clear();
    canvas.repaint();
}

/**
 * This method returns the current MouseEventListener for the canvas.
 *
 * @return The current MouseEventListener
 */
private static MouseEventListener getMouseListener(){
    return mouseSensor;
}

/**

```

```

* This method gets the listener responsible for responding to all events
* generated due to mouse motion within the object.
*
* @return The objects mouse motion listener.
*/
private static MouseMotionListener getMouseMotionListener(){
    return mouseSensor;
}

/**
* This method is used to get all currently selected entities in the
* CanvasFrame.
*
* @return A vector containing all currently selected entities.
*/
private static BaseEntityList getSelection(){
    return selection;
}

/**
* This method is used to refresh the area in and about the selection box
* so that graphical artifacts are not generated.
*
* @param selectedTL The top left point of the selection area.
* @param selectedBR The bottom right point of the selection area.
*/
private static void refreshSelectionBox(Point selectedTL, Point selectedBR){
    Graphics2D graphics = (Graphics2D) canvas.getGraphics();

    //get rid of old selection area
    graphics.setColor(canvas.getBackground());
    graphics.draw(oldSelectBox);

    //TODO the following lookup is very inefficient
    //redraw all other icons
    int popCount = MacSim.environment.population.size();

    for(int i = 0;i < popCount;i++){
        ((BaseEntity) MacSim.environment.population.get(i)).repaint();
    }

    // calculate bounds of selection area
    int width = selectedBR.x - selectedTL.x;
    int height = selectedBR.y - selectedTL.y;

    //draw it on canvas
    Dimension bounds = new Dimension(width, height);
    selectedArea = new Rectangle(selectedTL, bounds);
    graphics.setColor(Color.BLACK);
    graphics.draw(selectedArea);
    oldSelectBox = selectedArea;
}

// used in d&d interface

/**
* Currently unused.
*
* @param event The event that triggered this method.
*/
public void dragEnter(DropTargetDragEvent event){
}

/**
* Currently unused.
*

```

```

    * @param event The event that triggered this method.
    */
    public void dragExit(DropTargetEvent event){
    }

    /**
     * Currently unused.
     *
     * @param event The event that triggered this method.
     */
    public void dragOver(DropTargetDragEvent event){
    }

    /**
     * This method determines the actions that occur when a BaseEntity is
     * dropped onto the canvas from the species palette i.e. it is added to
     * both the canvas and the environments population.
     *
     * @param event The event that triggered this method.
     */
    public void drop(DropTargetDropEvent event){
        try{
            DataFlavor icon = event.getCurrentDataFlavors()[0];

            Transferable transfer = event.getTransferable();

            event.acceptDrop(DnDConstants.ACTION_COPY_OR_MOVE);

            BaseEntity thisIcon = (BaseEntity) transfer.getTransferData(icon);

            addIcon(thisIcon, event.getLocation().x, event.getLocation().y);

            event.dropComplete(true);
        } catch (IOException err1){
            ErrorReporter.report(ErrorReporter.FAILED_ENTITY_DROP);
        } catch (UnsupportedFlavorException err2){
            ErrorReporter.report(ErrorReporter.FAILED_ENTITY_DROP);
        }
    }

    /**
     * Currently unused.
     *
     * @param event The event that triggered this method.
     */
    public void dropActionChanged(DropTargetDragEvent event){
    }
}

```

```

package gui;

import datatypes.BaseEntity;

import java.awt.Component;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JMenuItem;
import javax.swing.JPanel;
import javax.swing.JPopupMenu;

/**
 * This class represents a context menu for the CanvasFrame class and is used
 * to initiate the distribution of a group of agents within a defined area of
 * the CanvasFrame class at the same time.
 *
 * @author Iain Robinson
 */
public class ContextMenu extends JPopupMenu{
    /**
     * The GUI frame that allows the user to specify the details of the entity
     * distribution.
     */
    static DistributionFrame dist = new DistributionFrame();

    /**
     * Creates and instance of the context menu.
     *
     * @param invoker The BaseEntity instance that triggered the context menu.
     * @param xCoord The current mouse X coordinate.
     * @param yCoord The current mouse X coordinate.
     */
    ContextMenu(Component invoker, int xCoord, int yCoord){
        super();

        if(JPanel.class.isInstance(invoker)){
            JMenuItem distribute = new JMenuItem("Distribute Entities");
            distribute.addActionListener(new ActionListener(){
                /**
                 * This method determines which action on the context menu
                 * has been selected by the user.
                 *
                 * @param event The event from the context menu that
                 * triggered the action.
                 */
                public void actionPerformed(ActionEvent event){
                    JMenuItem source = (JMenuItem) event.getSource();

                    if(source.getText().equals("Distribute Entities")){
                        dist.setVisible(true);
                    }
                }
            });
            add(distribute);
        }

        if(BaseEntity.class.isInstance(invoker)){
            JMenuItem delete = new JMenuItem("Delete");
            delete.addActionListener(new ActionListener(){
                /**
                 * This method determines which action on the context menu
                 * has been selected by the user.
                 *
                 * @param event The event from the context menu that

```

```
        *           triggered the action.
        */
    public void actionPerformed(ActionEvent event){
        JMenuItem source = (JMenuItem) event.getSource();

        if(source.getText().equals("Delete")){
            CanvasFrame.removeSelection();
        }
    }
});
add(delete);
}

show(invoker, xCoord, yCoord);
}
}
```

```

package gui;

import java.awt.GridLayout;
import java.awt.Point;
import java.awt.Rectangle;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.ButtonGroup;
import javax.swing.JButton;
import javax.swing.JDialog;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JRadioButton;
import javax.swing.JTextField;

/**
 * This class provides a simple GUI that allows a user to specify the details
 * of a "many agent" drag & drop agent. It allows a user to distribute any
 * number of agents over a predefined area of the environment in either a
 * random distribution pattern or a regular "rank and file" distribution.
 *
 * @author Iain Robinson
 */
public class DistributionFrame extends JDialog{
    /**
     * This field is used by the user to enter the number of entities to be
     * distributed.
     */
    private JTextField noEntField = new JTextField("9", 4);

    /** Used by the user to indicate a random distribution. */
    private JRadioButton randomBtn = new JRadioButton();

    /** Used by the user to indicate a regular distribution. */
    private JRadioButton regularBtn = new JRadioButton();

    /**
     * Buttongroup that stops the user from selecting random and regular
     * distributions at the same time.
     */
    private ButtonGroup type = new ButtonGroup();

    /**
     * Creates a new instance of DistributionFrame
     */
    DistributionFrame(){
        this.setTitle("Entity Distribution");
        this.setModal(true);
        setSize(400, 400);

        Point appLocation = RunMacs.inst.getLocation();
        this.setLocation(appLocation.x +
            ((RunMacs.inst.getWidth() - this.getWidth()) / 2),
            appLocation.y +
            ((RunMacs.inst.getHeight() - this.getHeight()) / 2));

        JButton okButton = new JButton("Confirm");
        JButton cancelButton = new JButton("Cancel");

        okButton.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent event){
                //gather parameters
                int noEntities = Integer.parseInt(noEntField.getText());
            }
        });
    }
}

```

```

        Rectangle area = CanvasFrame.selectedArea;
        int type = CanvasFrame.DIST_RANDOM;

        if(randomBtn.isSelected()){
            type = CanvasFrame.DIST_RANDOM;
        }

        if(regularBtn.isSelected()){
            type = CanvasFrame.DIST_REGULAR;
        }

        //add the new entities to the canvas
        CanvasFrame.addManyIcons(noEntities,
            PaletteFrame.getSelectedSpecies(), area, type);
        ContextMenu.dist.setVisible(false);
    }
});

// just close the frame if the user cancels
cancelButton.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent event){
        ContextMenu.dist.setVisible(false);
    }
});

// Layout components.
JPanel content = new JPanel(new GridLayout(4, 2));

JLabel noEntitiesLbl = new JLabel("No. Entities");
JLabel randomLbl = new JLabel("Random");
JLabel regularLbl = new JLabel("Regular");

content.add(noEntField);
type.add(randomBtn);
type.add(regularBtn);
regularBtn.setSelected(true);

content.add(noEntitiesLbl);
content.add(noEntField);
content.add(randomLbl);
content.add(randomBtn);
content.add(regularLbl);
content.add(regularBtn);
content.add(okButton);
content.add(cancelButton);

this.setContentPane(content);
this.pack();
}
}

```

```

package gui;

import common.MacsPreferences;

import jedit.JEditTextArea;
import jedit.JavaTokenMarker;
import jedit.SyntaxDocument;

import java.awt.Font;

import javax.swing.JInternalFrame;
import javax.swing.JScrollPane;
import javax.swing.JSplitPane;
import javax.swing.JTextPane;
import javax.swing.WindowConstants;
import javax.swing.event.InternalFrameAdapter;
import javax.swing.event.InternalFrameEvent;

/**
 * This class provides a simple Java source code editor that allows a user to
 * extend the basic agent code created with the SpeciesWizard class and adapt
 * it to their own needs.
 *
 * @author Iain Robinson
 */
public class EntityEditorFrame extends JInternalFrame{
    /** The code editor window of the frame. */
    public static JEditTextArea editorPane = new JEditTextArea();

    /**
     * The debug pane of the frame. Provides a way to display compiler
     * messages.
     */
    public static JTextPane debugPane = new JTextPane();

    /** The underlying representation of the code document. */
    public static SyntaxDocument jDoc = new SyntaxDocument();

    /** The name of the species source code file being edited (if any exists). */
    public static String currentSpeciesSourceFileName = null;

    /**
     * Creates a new instance of EntityEditorFrame
     */
    EntityEditorFrame(){
        super("Entity Editor", true, true, true, true);

        this.addInternalFrameListener(new InternalFrameAdapter(){
            /**
             * This method dictates what will happen when the user chooses
             * to close the EntityEditor frame. Simply overrides the
             * default behaviour and instead hides the frame rather than
             * destroying it.
             *
             * @param event The event that triggered the close action.
             */
            public void internalFrameClosing(InternalFrameEvent event){
                JInternalFrame source = (JInternalFrame) event.getSource();
                source.setVisible(false);

                source.setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE);
            }
        });

        jDoc.setTokenMarker(new JavaTokenMarker());

```

```

    editorPane.setDocument(jDoc);

    editorPane.setFont(new Font("Courier New", Font.PLAIN, 12));

    //editorPane.setSize(150, 150);
    JScrollPane debugScrollPane = new JScrollPane(debugPane);
    debugPane.setEditable(false);

    //      Create a split pane with the two scroll panes in it.
    JSplitPane splitPane = new JSplitPane(JSplitPane.VERTICAL_SPLIT,
        editorPane, debugScrollPane);

    getContentPane().add(splitPane);
    this.setSize(410, 650);
    this.setLocation(0, 70);
    this.setVisible(true);
}

/**
 * This method overrides the usual setVisible method to allow the editor to
 * notify the main application menu of its current visibility status.
 *
 * @param visible The desired visibility state of the editor frame.
 */
public void setVisible(boolean visible){
    super.setVisible(visible);
    FrameMenuBar.update(MacsPreferences.getPrefView());
}
}

```

```

package gui;

import common.FileIO;
import common.MacsPreferences;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.ButtonGroup;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JRadioButtonMenuItem;

/**
 * This class is responsible for building and displaying the main menu bar of
 * the Macs application.
 *
 * @author Iain Robinson
 */
public class FrameMenuBar extends JMenuBar{
    /**
     * A radio button that allows the user to specify that they require the
     * Developer viewing mode to be used.
     */
    private static JRadioButtonMenuItem modeDeveloper = new JRadioButtonMenuItem(
        "Developer Mode");

    /**
     * A radio button that allows the user to specify that they require the
     * AbstractViewer viewing mode to be used.
     */
    private static JRadioButtonMenuItem modeViewer = new JRadioButtonMenuItem(
        "Viewer Mode");

    /** The menu element specifying all the alternative viewing operations. */
    private static JMenu viewMenu = new JMenu("View");

    /** The menu element specifying all file operations. */
    private static JMenu fileMenu = new JMenu("File");

    /** The Species sub-menu. */
    private static JMenu speciesMenu = new JMenu("Species");

    /** The "Open Scenario" menu item. */
    private static JMenuItem scenarioOpenMenu = new JMenuItem(
        "Open Scenario...");

    /** The "Save Scenario" menu item. */
    private static JMenuItem scenarioSaveMenu = new JMenuItem(
        "Save Scenario...");

    /** The "Open Species" menu item. */
    private static JMenuItem speciesOpenMenu = new JMenuItem("Open Species...");

    /** The "Save Species" menu item. */
    private static JMenuItem speciesSaveMenu = new JMenuItem("Save Species...");

    /** The "Add To Palette" menu item. */
    private static JMenuItem importMenu = new JMenuItem("Add To Palette...");

    /** The "Compile" menu item. */
    private static JMenuItem compileMenu = new JMenuItem("Compile");

    /** The "Species Wizard" menu item. */

```

```

private static JMenuItem designMenu = new JMenuItem("Wizard...");

/** The menu element specifying all settings operations. */
private static JMenuItem settingsMenu = new JMenuItem("Settings");

/**
 * The menu element specifying all species specific operations operations.
 */
private static ButtonGroup modeGroup = new ButtonGroup();

/**
 * This is the listener responsible for responding to user commands given
 * via any of the controls on the main application menu.
 */
private static ActionListener listener = new ActionListener(){
    /**
     * This method actually determines which action on the main menu
     * has been selected by the user.
     *
     * @param event The event from the main menu that triggered the
     *             action.
     */
    public void actionPerformed(ActionEvent event){
        JMenuItem comp = (JMenuItem) event.getSource();

        // open the species design dialog
        if(comp.getActionCommand().equals("Wizard...")){
            MacSim.wizard.display();
        }

        //open the preferences dialog
        if(comp.getActionCommand().equals("Preferences...")){
            MacSim.prefs.setVisible(true);
        }

        //open the species import dialog
        if(comp.getActionCommand().equals("Add To Palette...")){
            FileIO.launchFileDialog("Import");
        }

        //open the species file dialog (open)
        if(comp.getActionCommand().equals("Open Species...")){
            FileIO.launchFileDialog("OpenSpecies");
        }

        //          open the species file dialog (save)
        if(comp.getActionCommand().equals("Save Species...")){
            FileIO.launchFileDialog("SaveSpecies");
        }

        // compile the current source file ;
        if(comp.getActionCommand().equals("Compile")){
FileIO.compileSourceFile(EntityEditorFrame.currentSpeciesSourceFileName);
        }

        //open the scenerio file dialog (open)
        if(comp.getActionCommand().equals("Open Scenario...")){
            FileIO.launchFileDialog("OpenScenario");
        }

        // open the scenerio file dialog (save)
        if(comp.getActionCommand().equals("Save Scenario...")){
            FileIO.launchFileDialog("SaveScenario");
        }
    }
}

```

```

        if(comp.getActionCommand().equals("Developer Mode")){
            MacsPreferences.setPrefView(MacSim.VIEW_DEV);
        }

        if(comp.getActionCommand().equals("AbstractViewer Mode")){
            MacsPreferences.setPrefView(MacSim.VIEW_VIEWER);
        }
    }
};

/**
 * Creates a new instance of Macs main menu bar.
 */
FrameMenuBar(){
    super();

    // set up menus
    scenarioOpenMenu.addActionListener(getListener());

    scenarioSaveMenu.addActionListener(getListener());

    speciesOpenMenu.addActionListener(getListener());
    speciesSaveMenu.addActionListener(getListener());
    importMenu.addActionListener(getListener());
    designMenu.addActionListener(getListener());
    compileMenu.addActionListener(getListener());

    speciesMenu.add(speciesOpenMenu);
    speciesMenu.add(speciesSaveMenu);
    speciesMenu.add(importMenu);
    speciesMenu.add(designMenu);
    speciesMenu.add(compileMenu);
    fileMenu.add(speciesMenu);

    JMenu scenarioMenu = new JMenu("Scenario");
    scenarioMenu.add(scenarioOpenMenu);
    scenarioMenu.add(scenarioSaveMenu);
    fileMenu.add(scenarioMenu);

    JMenuItem prefMenu = new JMenuItem("Preferences...");
    prefMenu.addActionListener(getListener());
    viewMenu.add(prefMenu);

    JMenu modeMenu = new JMenu("Mode");

    modeDeveloper.addActionListener(getListener());
    modeViewer.addActionListener(getListener());

    modeGroup.add(modeDeveloper);
    modeGroup.add(modeViewer);

    modeMenu.add(modeViewer);
    modeMenu.add(modeDeveloper);

    settingsMenu.add(modeMenu);

    add(fileMenu);
    add(viewMenu);
    add(settingsMenu);
}

/**
 * This method is used to retrieve the listener responsible for responding
 * to all events generated within the main menu of the application.
 *
 * @return The main menu's event listner.
 */

```



```

package gui;

import backend.Engine;

import common.MacsPreferences;

import datatypes.AbstractViewer;
import datatypes.Environment;

import java.awt.Color;
import java.awt.Frame;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

import javax.swing.JDesktopPane;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.WindowConstants;

/**
 * This is the main class of the MACS system and is responsible for creating
 * and initialising the whole application and its various component classes.
 *
 * @author Iain Robinson
 */
public class MacSim extends JFrame{
    /** The environment that is represented by the CanvasFrame class. */
    public static Environment environment = new Environment();

    /** The simulation engine that will be used to run the simulation on. */
    public static Engine engine = new Engine();

    /** The 2D static representation of the Environment in the application. */
    public static CanvasFrame canvas = new CanvasFrame();

    /** The container for all other "internal" frames. */
    public static JDesktopPane desktop = new JDesktopPane();

    /**
     * The visual representation for the underlying user preferences for the
     * application.
     */
    public static MacsPreferences prefs = new MacsPreferences();

    /** The entity source code editor frame. */
    public static EntityEditorFrame editor = new EntityEditorFrame();

    /** The species "wizard" frame. */
    static SpeciesWizard wizard = new SpeciesWizard();

    /** The Frame containing all available species types. */
    public static PaletteFrame iconList = new PaletteFrame();

    /** The main menu structure for the application. */
    private static FrameMenuBar mainMenu = new FrameMenuBar();

    /** The set of "media player" controls for simulation control. */
    private static TimeControlFrame timeCtrl = new TimeControlFrame();

    /** Indicates the application should be run in Developer mode. */
    public static final int VIEW_DEV = 1;

    /** Indicates the application should be run in AbstractViewer mode. */
    public static final int VIEW_VIEWER = 0;

```

```

/** The viewer that will receive the output from the simulation engine. */
public static AbstractViewer output;

/**
 * Creates a new MacSim object.
 */
MacSim(){
    setTitle("Multi Agent Collectives Simulator");
    setJMenuBar(mainMenu);

    // set up main frame components
    getContentPane().add(desktop);
    desktop.setBackground(Color.LIGHT_GRAY);

    // add internal components to desktop
    desktop.add(iconList);
    desktop.add(canvas);
    desktop.add(timeCtrl);
    desktop.add(editor);

    // change the frame properties before showing
    setSize(1024, 768);
    setExtendedState(Frame.MAXIMIZED_BOTH);
    addWindowListener(new WindowAdapter(){
        public void windowClosing(WindowEvent event){
            int result = JOptionPane.showConfirmDialog(null,
                "Are you sure you wish to exit? Any unsaved work will be
lost.", "Warning", JOptionPane.WARNING_MESSAGE, JOptionPane.YES_NO_OPTION);

            if(result == JOptionPane.YES_OPTION){
                System.exit(0);
            }else{
                JFrame tmp = (JFrame) event.getSource();

tmp.setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE);
                }
            }
        });
    setVisible(true);
}

/**
 * Updates the application GUI to reflect changes in the user preferences.
 *
 * @param view The preferred user mode. Either VIEW_DEV or VIEW_VIEWER.
 */
public static void update(int view){
    if(view == VIEW_DEV){
        iconList.setVisible(true);
        canvas.setVisible(true);
        timeCtrl.setVisible(true);
        editor.setVisible(false);
    }

    if(view == VIEW_VIEWER){
        iconList.setVisible(false);
        canvas.setVisible(true);
        timeCtrl.setVisible(true);
        editor.setVisible(false);
    }
}
}

```

```

package gui;

import common.ErrorReporter;
import common.FileIO;

import datatypes.BaseEntity;
import datatypes.BaseEntityList;

import java.awt.Component;
import java.awt.GridLayout;
import java.awt.dnd.DnDConstants;
import java.awt.dnd.DragGestureEvent;
import java.awt.dnd.DragGestureListener;
import java.awt.dnd.DragSource;
import java.awt.dnd.DragSourceDragEvent;
import java.awt.dnd.DragSourceDropEvent;
import java.awt.dnd.DragSourceEvent;
import java.awt.dnd.DragSourceListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;

import javax.swing.JInternalFrame;
import javax.swing.JPanel;

/**
 * This class represents a display of all BaseEntity subtypes that are
 * available for the user to use within the Macs application.
 *
 * @author Iain Robinson
 */
public class PaletteFrame extends JInternalFrame implements DragGestureListener,
    DragSourceListener {
    /** This class determines what action to perform after input by the user. */
    private static MouseAdapter mouseSensor = new MouseAdapter(){
        //TODO request for focus does not work
        public void mouseEntered(MouseEvent event){
            Component comp = (Component) event.getSource();
            comp.requestFocus();
        }

        /**
         * The action taken when a BaseEntity is clicked on in the species
         * palette.
         *
         * @param event The event that triggered the action.
         */
        public void mousePressed(MouseEvent event){
            PaletteFrame.setSelectedSpecies((BaseEntity) event.getComponent());
        }
    };

    /** The currently selected species. */
    private static BaseEntity selectedEnt = null;

    /** A vector of all the species currently on the species palette. */
    private static BaseEntityList speciesList = new BaseEntityList(0, 1);

    /** The layout manager for the iconPanel variable */
    private static GridLayout grid = new GridLayout(1, 1);

    /** The panel on which all added species are displayed as labels. */
    private static JPanel iconPanel = new JPanel(grid);

    /**
     * Creates an instance of the class.

```

```

*/
PaletteFrame(){
    super("Species Palette", false, false, false, true);

    BaseEntity obstacle1 = FileIO.importSpeciesClassFile(
        "species.obstacle.Obstacle");
    addSpecies(obstacle1);
    selectedEnt = obstacle1;

    //TODO remove in final version
    BaseEntity pred1 = FileIO.importSpeciesClassFile(
        "species.predator.Predator");
    addSpecies(pred1);

    getContentPane().add(iconPanel);
    setVisible(true);
}

/**
 * Returns the current MouseEventListener for the object.
 *
 * @return The current MouseEventListener for the object.
 */
private static MouseAdapter getMouseListener() {
    return mouseSensor;
}

/**
 * This method is used to find out which Species is currently selected
 * within the palette frame.
 *
 * @return The currently selected Species.
 */
static BaseEntity getSelectedSpecies() {
    return selectedEnt;
}

/**
 * This method sets which species in the palette is to be currently
 * selected.
 *
 * @param species An instance of the desired selection species.
 */
private static void setSelectedSpecies(BaseEntity species) {
    selectedEnt = species;

    for(int i = 0; i < speciesList.size(); i++) {
        if(speciesList.get(i).equals(selectedEnt)) {
            PaletteFrame.selectedEnt.setEnabled(false);
        } else {
            ((BaseEntity) speciesList.get(i)).setEnabled(true);
        }
    }
}

/**
 * This method adds a species to the species palette. Usually done via the
 * species..import option on the main application menu.
 *
 * @param species The BaseEntity to add to the species palette.
 */
public final void addSpecies(BaseEntity species) {
    if(!speciesList.containsSpecies(species)) {
        speciesList.add(species);
        grid = new GridLayout(speciesList.size(), 1);
        iconPanel.setLayout(grid);
    }
}

```

```

        iconPanel.add(species);
        species.addMouseListener(getMouseListener());

        /*
         * TODO This next line needs fixed offsets of 35 and ?? to account
         * for the size of the top border and the width of the label text --
         * is there a more dynamic way of achieving the same that will work
         * in every instance?
         */
        setSize(150,
            (speciesList.size() * species.getPreferredSize().height) + 35);

        DragSource dragSource = DragSource.getDefaultDragSource();
        dragSource.createDefaultDragGestureRecognizer(species,
            DnDConstants.ACTION_COPY_OR_MOVE, this);
    }
}

/**
 * This method is used to get the list of species currently available
 * within the palette frame.
 *
 * @return A list of the currently available species within the palette
 *         frame.
 */
public static BaseEntityList getSpeciesList() {
    return speciesList;
}

// used in d&d interface

/**
 * Currently unused.
 *
 * @param event The event that triggered this action.
 */
public void dragDropEnd(DragSourceDropEvent event) {
}

/**
 * Currently unused.
 *
 * @param event The event that triggered this action.
 */
public void dragEnter(DragSourceDragEvent event) {
}

/**
 * Currently unused.
 *
 * @param event The event that triggered this action.
 */
public void dragExit(DragSourceEvent event) {
}

/**
 * This method is triggered when a user starts to drag an entity from the
 * Palette frame to the Canvas frame.
 *
 * @param event The event that triggered this action.
 */
public void dragGestureRecognized(DragGestureEvent event) {
    Class type = event.getComponent().getClass();

    BaseEntity transferred = null;

```

```
    try{
        transfered = (BaseEntity) type.newInstance();
        event.startDrag(DragSource.DefaultCopyDrop, transfered, this);
    }catch(InstantiationException err){
        ErrorReporter.report(ErrorReporter.FAILED_INSTANTIATION);
    }catch(IllegalAccessException err2){
        ErrorReporter.report(ErrorReporter.FAILED_INSTANTIATION);
    }
}

/**
 * Currently unused.
 *
 * @param event The event that triggered this action.
 */
public void dragOver(DragSourceDragEvent event){
}

/**
 * Currently unused.
 *
 * @param event The event that triggered this action.
 */
public void dropActionChanged(DragSourceDragEvent event){
}
}
```

```

package gui;

import common.ErrorReporter;
import common.MacsPreferences;

import javax.swing.UIManager;

/**
 * This is a simple class that can be used to run the main application class.
 *
 * @author Iain Robinson
 */
public class RunMacs{
    /** The instance of the full MACS application that is run. */
    public static MacSim inst = null;

    /**
     * This is the main entry point for the Macs Simulator. Currently no
     * arguments are passed to the application.
     *
     * @param args Not currently used.
     */
    public static void main(String[] args){
        // set system specific UI look - gets the XP look and feel when run on
        // windows XP
        try{
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        }catch(Exception e){
            ErrorReporter.report(ErrorReporter.LOOK_AND_FEEL);
        }

        inst = new MacSim();

        /**
         * if the first time application has been launched show the preferences
         * frame so the user can set up
         */
        boolean firstView = MacsPreferences.getFirstView();

        if(firstView){
            MacSim.prefs.display();

            //TODO - uncomment next line for final version
            //prefs.getPreferences().putBoolean("firstView", false) ;
        }
    }
}

```

```

package gui;

import common.ErrorReporter;
import common.FileIO;

import java.awt.BorderLayout;
import java.awt.GridLayout;
import java.awt.Point;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

import javax.swing.JButton;
import javax.swing.JDialog;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;

/**
 * This class provides a simple GUI that the user can use to specify the basic
 * elements of a new agent species.
 *
 * @author Iain Robinson
 */
public class SpeciesWizard extends JDialog{
    /** The label for the species class name field. */
    private static JLabel nameLbl = new JLabel("Species Class Name");

    /**
     * A field allowing the user to provide a Name for the new species class.
     */
    private static JTextField nameField = new JTextField();

    /** The label for the species icon field. */
    private static JLabel iconLbl = new JLabel("Species Icon File");

    /**
     * A field allowing the user to provide a filename of a GIF file to be
     * used as an icon for the new species class.
     */
    private static JTextField iconField = new JTextField();

    /**
     * A button that triggers a file chooser dialog to open that allows the
     * user to choose a GIF file for use as a species icon.
     */
    private static JButton browseBtn = new JButton("Browse...");

    /**
     * Creates a new SpeciesWizard object.
     */
    SpeciesWizard(){
        this.setTitle("Species Wizard");
        this.setModal(true);

        JButton okButton = new JButton("Confirm");
        JButton cancelButton = new JButton("Cancel");

        okButton.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent event){

```

```

        //create a new directory under \species
        String path = FileIO.SPECIES_DIR + "\\ " +
            nameField.getText().toLowerCase();

        File pathTo = new File(path);
        pathTo.mkdir();

        try{
            FileInputStream graphic = new
FileInputStream(iconField.getText());

            try{
                FileOutputStream fileout = new FileOutputStream(path +
                    "\\ " + nameField.getText().toLowerCase() +
                    ".gif");
                fileout.write(graphic.read());
            }catch(FileNotFoundException error){
                ErrorReporter.report(ErrorReporter.GIF_FILE_NOT_FOUND);
            }
        }catch(IOException error2){
            ErrorReporter.report(ErrorReporter.GIF_FILE_TRANSFER_FAILURE);
        }

        //create the source file
        String sourceName = nameField.getText();
        String sourceFile = pathTo + "\\ " + sourceName + ".java";

        //create the source code
        String source = getSource();

        //write the source to the new file
        FileIO.saveSpeciesSourceFile(sourceFile, source);

        //and load it into the editor
        FileIO.openSpeciesSourceFile(sourceFile);

        MacSim.editor.setVisible(true);
        MacSim.wizard.setVisible(false);
    }
});

// just close the frame if the user cancels
cancelButton.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent event){
        //this disappears when cancel is pressed
        MacSim.wizard.setVisible(false);
    }
});

browseBtn.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent event){
        FileIO.launchFileDialog("OpenGIF");
    }
});

GridLayout grid = new GridLayout(3, 2);
grid.setVgap(20);
grid.setHgap(20);

// Layout components.
JPanel content = new JPanel(grid);
content.add(nameLbl);
content.add(nameField);
content.add(iconLbl);

iconField.setColumns(20);

```

```

JPanel panel = new JPanel(new BorderLayout());
panel.add(iconField, BorderLayout.CENTER);
panel.add(browseBtn, BorderLayout.EAST);

content.add(panel);
content.add(okButton);
content.add(cancelButton);

this.setContentPane(content);
this.pack();
}

/**
 * This method provides access to template source constructed from the
 * details given by the user via the wizard GUI.
 *
 * @return A string containing the template source.
 */
private static String getSource(){
    //TODO remember to keep the source up to date with the actual needed
    // source
    String source = "package species." + nameField.getText().toLowerCase() +
        "; " + "\n\n" + "import java.awt.datatransfer.DataFlavor;" + "\n" +
        "import datatypes.BaseEntity;" + "\n\n" + "public class " +
        nameField.getText() + " extends BaseEntity{" + "\n\n\t" +
        "public " + nameField.getText() + "(){ " + "\n\t\t" +
        "super(new DataFlavor(DataFlavor.javaJVMLocalObjectType" + "+" +
        "\n\t\t\t" + "\""; class = species." +
        nameField.getText().toLowerCase() + ". " + nameField.getText() +
        "\",\"" + nameField.getText() + "\"), " + "\n\t\t\t" +
        "\"species/" + nameField.getText().toLowerCase() + "/" +
        nameField.getText().toLowerCase() + ".gif\"" + ");" + "\n\t" + "}" +
        "\n\n\t" + "public void run(){ " + "\n\t\t" +
        "/***** Your own run routine below here *****/" + "\n\t\t" +
        "\n\t\t" + "/***** Your own run routine above here *****/" +
        "\n\t" + "}" + "\n\t" +
        "/***** Your own methods below here *****/" + "\n\t\t\t\t\n\n\n\t" +
        "/***** Your own methods above here *****/" + "\n" + "};

    return source;
}

/**
 * This methods displays the Species Wizard at the centre of the screen.
 */
void display(){
    Point appLocation = RunMacs.inst.getLocation();

    this.setLocation(appLocation.x +
        ((RunMacs.inst.getWidth() - this.getWidth()) / 2),
        appLocation.y +
        ((RunMacs.inst.getHeight() - this.getHeight()) / 2));
    MacSim.wizard.setVisible(true);
}
}

```

```

package gui;

import backend.Engine;

import common.MacsPreferences;

import viewers.Viewer2D;
import viewers.ViewerFile;

import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JInternalFrame;

/**
 * This class constructs a simple "media player" style set of controls from
 * which the user can control the "playback" of any real-time simulation run
 * performed with the system. This is a "state-based" control that implements
 * the state transition model depicted in figure 6 of Appendix III.
 *
 * @author Iain Robinson
 */
public class TimeControlFrame extends JInternalFrame{
    /** Indicates that the tim control frame is in the playable state. */
    public static final int STATE_PLAYABLE = 0;

    /** Indicates that the tim control frame is in the playing state. */
    private static final int STATE_PLAYING = 1;

    /** Indicates that the tim control frame is in the paused state. */
    private static final int STATE_PAUSED = 2;

    /** Indicates that the tim control frame is in the playing file state. */
    private static final int STATE_STEPPING = 3;

    /** The play button of the time control frame. */
    private static JButton playBtn = new JButton("Play");

    /** The pause button of the time control frame. */
    private static JButton pauseBtn = new JButton("Pause");

    /** The step button of the time control frame. */
    private static JButton stepBtn = new JButton("step");

    /** The stop button of the time control frame. */
    private static JButton stopBtn = new JButton("stop");

    /** The current state of the time control frame. */
    private static int currentState;

    /**
     * Creates a new instance of TimeControlFrame
     */
    TimeControlFrame(){
        super("Time Control", false, false, false, false);
        this.setSize(260, 67);
        this.setLocation(150, 0);
        setState(STATE_PLAYABLE);
        this.setVisible(true);
        this.getContentPane().setLayout(new GridLayout(1, 4));

        //add each button
        this.getContentPane().add(playBtn);

```

```

playBtn.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent event){
        if(currentState == STATE_PLAYABLE){
            //create a new engine if we are about to start playing
            //rather than resume playing
            MacSim.engine = new Engine();

            //initialise with a copy of the original environment
            MacSim.engine.initialise(MacSim.environment.getDeepCopy());

            if(MacPreferences.getPrefOutput() == Engine.OUTPUT_FILE){
                //create a new file output and progress bar
                MacSim.output = new ViewerFile(MacSim.engine);
            }else{
                //or create a new 2D viewer
                MacSim.output = new Viewer2D(MacSim.engine);
            }

            //run the simulation
            play();
        }

        //otherwise simply resume where we left off
        if(currentState == STATE_PAUSED){
            resume();
        }

        if(currentState == STATE_STEPPING){
            unStep();
            resume();
        }
    }
});

this.getContentPane().add(pauseBtn);
pauseBtn.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent event){
        if(currentState == STATE_PLAYING){
            pause();
        }
    }
});

this.getContentPane().add(stepBtn);
stepBtn.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent event){
        setState(STATE_STEPPING);
        MacSim.engine.step();
        MacSim.output.step();
    }
});

this.getContentPane().add(stopBtn);
stopBtn.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent event){
        MacSim.engine.end();
        MacSim.output.end();
        setState(STATE_PLAYABLE);
    }
});
}

/**
 * This method pauses the simulation engine during execution.
 */
private static void pause(){

```

```

        setState(STATE_PAUSED);
        MacSim.engine.pause();
        MacSim.output.pause();
    }

    /**
     * This method resumes execution of the simulation after the pause state.
     */
    private static void resume() {
        setState(STATE_PLAYING);
        MacSim.engine.unPause();
        MacSim.output.unPause();
    }

    /**
     * This method takes the user interface out of stepping mode initiated by
     * the step() method.
     */
    private static void unStep() {
        MacSim.engine.unStep();
        MacSim.output.unStep();
    }

    /**
     * This method starts execution of the simulation engine.
     */
    private static void play() {
        setState(STATE_PLAYING);
        MacSim.output.start();
        MacSim.engine.start();
    }

    /**
     * This method sets the state of the time control frame and in doing so the
     * simulation engine as well.
     *
     * @param state The desired state of the time control frame and simulaiton
     *             engine.
     */
    public static void setState(int state) {
        switch(state) {
            case STATE_PLAYABLE:
                setPlayableState();

                break;

            case STATE_PLAYING:
                setPlayingState();

                break;

            case STATE_PAUSED:
                setpausedState();

                break;

            case STATE_STEPPING:
                setSteppingState();

                break;

            default:
                /* do nothing */
        }

        currentState = state;

```

```

}

/**
 * This method updates the display of the tim control frame to correspond
 * to the paused state.
 */
private static void setPausedState(){
    playBtn.setEnabled(true);
    pauseBtn.setEnabled(false);
    stepBtn.setEnabled(true);
    stopBtn.setEnabled(true);
}

/**
 * This method is responsible for setting the state of the class to
 * correspond to the simulation engine being in stepping mode.
 */
private static void setSteppingState(){
    playBtn.setEnabled(true);
    pauseBtn.setEnabled(false);
    stepBtn.setEnabled(true);
    stopBtn.setEnabled(true);
}

/**
 * This method updates the display of the tim control frame to correspond
 * to the playing state.
 */
private static void setPlayingState(){
    if(MacsPreferences.getPrefOutput() == Engine.OUTPUT_FILE){
        playBtn.setEnabled(false);
        pauseBtn.setEnabled(false);
        stepBtn.setEnabled(false);
        stopBtn.setEnabled(false);
    }else{
        playBtn.setEnabled(false);
        pauseBtn.setEnabled(true);
        stepBtn.setEnabled(false);
        stopBtn.setEnabled(true);
    }
}

/**
 * This method updates the display of the tim control frame to correspond
 * to the playable state.
 */
private static void setPlayableState(){
    playBtn.setEnabled(true);
    pauseBtn.setEnabled(false);
    stepBtn.setEnabled(false);
    stopBtn.setEnabled(false);
}
}

```

```

package viewers;

import backend.Engine;

import datatypes.AbstractViewer;
import datatypes.BaseEntity;
import datatypes.BaseEntityList;

import gui.MacSim;

import java.awt.Dimension;
import java.awt.Frame;
import java.awt.Insets;
import java.awt.event.ComponentAdapter;
import java.awt.event.ComponentEvent;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.WindowConstants;

/**
 * This is an example class demonstrating the ability of the MACS
 * AbstractViewer class. It renders a simulation on a 2D canvas.
 *
 * @author Iain Robinson.
 */
public class Viewer2D extends AbstractViewer{
    /** The main animation frame */
    private static JFrame viewerFrame = new JFrame("2D AbstractViewer");

    /** The panel on which animation is actually performed. */
    private static JPanel viewerPane = new JPanel(null);

    /**
     * The scroll pane that allows only part of the environment to be visible.
     */
    private static JScrollPane view = new JScrollPane(viewerPane);

    /**
     * Creates a new Viewer2D object.
     *
     * @param engine The source of the Simulation data - i.e. the running
     * simulation engine.
     */
    public Viewer2D(Engine engine){
        super(engine);

        //viewerPane.setSize(engine.env.xLimit, engine.env.yLimit);
        viewerPane.setPreferredSize(new Dimension(engine.env.xLimit,
            engine.env.yLimit));

        view.setSize(100, 100);
        view.setViewportView(viewerPane);
        view.setDoubleBuffered(true);
        viewerFrame.getContentPane().add(view);
        viewerFrame.setSize(400, 400);
        viewerFrame.setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE);
        viewerFrame.setVisible(true);

        viewerFrame.addComponentListener(new ComponentAdapter(){
            public void componentResized(ComponentEvent event){
                JFrame tmp = (JFrame) event.getSource();

```

```

//TODO still needs some work to get rid of scrollbars
//properly
int sbwidth = Viewer2D.view.getVerticalScrollBar().getWidth();
int sbheight = Viewer2D.view.getHorizontalScrollBar()
    .getHeight();

Insets in = tmp.getInsets();

int xoff = sbwidth + in.left + in.right + 3;
int yoff = sbheight + in.top + in.bottom + 28;

if(tmp.getWidth() > MacSim.environment.xLimit){
    tmp.setSize(MacSim.environment.xLimit + xoff,
        tmp.getHeight());
}

if(tmp.getHeight() > MacSim.environment.yLimit){
    tmp.setSize(tmp.getWidth(),
        MacSim.environment.yLimit + yoff);
}

//if the user has attempted to miximise the
//canvasframe then just allow them to make it
//the same size as the environment.
if((tmp.getExtendedState() == Frame.MAXIMIZED_HORIZ) ||
    (tmp.getExtendedState() == Frame.MAXIMIZED_VERT)){
    tmp.setExtendedState(Frame.NORMAL);
    tmp.setSize(MacSim.environment.xLimit + xoff,
        MacSim.environment.yLimit + yoff);
}
}
});
}

/**
 * This method indicates what action will be taken by the class on each
 * iteration of a simulation.
 */
public void play(){
    viewerPane.getGraphics().clearRect(0, 0, viewerPane.getWidth(),
        viewerPane.getHeight());

    // Get a snapshot of the Backends current state.
    BaseEntityList frame = source.deliverCurrentData();

    for(int i = 0;i < frame.size();i++){
        BaseEntity current = (BaseEntity) frame.get(i);
        int xpos = current.getAttribute("xPosition").getIntValue();
        int ypos = current.getAttribute("yPosition").getIntValue();
        viewerPane.getGraphics().drawImage(current.image, xpos, ypos, null);
    }
}

/**
 * This method indicates what action will be taken by the class after a
 * simulation has ended but before the object is disposed of.
 */
public void cleanup(){
    viewerFrame.dispose();
}
}

```

```

package viewers;

import java.io.File;
import java.util.Enumeration;

import javax.swing.JFileChooser;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.OutputKeys;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerConfigurationException;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;

import backend.Engine;

import common.ErrorReporter;
import common.FileIO;

import datatypes.AbstractViewer;
import datatypes.Attribute;
import datatypes.BaseEntity;
import datatypes.BaseEntityList;

/**
 * This is an example class demonstrating the ability of the MACS
 * AbstractViewer class. It renders a simulation to a file.
 *
 * @author Iain Robinson
 */
public class ViewerFile extends AbstractViewer{
    /**
     * A String containing the tag name "value". Simply used to avoid
     * repetition of the string literal in the code.
     */
    private static String valueTag = "value";

    /** The basic XML document used to store simulation results. */
    private static Document sDoc;

    /** The file used to write the final simulation result document to. */
    private File output;

    /** The root element of the resulting XML document. */
    private Element simNode;

    /**
     * Creates a new ViewerFile object.
     *
     * @param source The source engine of the simulation data.
     */
    public ViewerFile(Engine source){
        super(source);

        JFileChooser fManager = new JFileChooser();
        output = launchSaveOutputFileDialog(fManager);

        if(output != null){

```

```

        DocumentBuilder docBuilder = null;

        try{
            docBuilder = DocumentBuilderFactory.newInstance()
                .newDocumentBuilder();
        }catch(ParserConfigurationException e){
            ErrorReporter.report(ErrorReporter.FAILED_PARSER);
        }

        SDoc = docBuilder.newDocument();

        Node sNode = sDoc;

        simNode = sDoc.createElement("simulation");

        sNode.appendChild(simNode);
    }
}

/**
 * This method indicates what actions will be executed by the object during
 * each iteration of a simulation.
 */
public void play(){
    Element xmlFrame = sDoc.createElement("frame");
    xmlFrame.setAttribute("count", Integer.toString(count));
    simNode.appendChild(xmlFrame);

    // Get a snapshot of the Backends current state.
    BaseEntityList frame = source.deliverCurrentData();

    for(int i = 0;i < frame.size();i++){
        BaseEntity current = (BaseEntity) frame.get(i);

        Element xmlEntity = createEntityXML(current);
        xmlFrame.appendChild(xmlEntity);
    }
}

/**
 * This method indicates what actions will be taken by the object once a
 * simulation has finished but before the object has been disposed of.
 */
public void cleanup(){
    writeXML(sDoc, output);
}

/**
 * This method creates the entity portion of a Scenario file and inserts
 * all necessary data values.
 *
 * @param currentEntity The entity to add to the document.
 *
 * @return An XML element containing all required information about the
 *         entity.
 */
private static Element createEntityXML(BaseEntity currentEntity){
    // create a member section
    Element memberTag = sDoc.createElement("entity");

    //set its class type
    memberTag.setAttribute("type",
        currentEntity.getClass().getCanonicalName());

    // process all its attributes
    Enumeration attributeList = currentEntity.getAttributes().elements();

```

```

        while(attributeList.hasMoreElements()){
            Attribute currentAttr = (Attribute) attributeList.nextElement();
            String attributeName = currentAttr.getName();

            //create the attribute XML tag
            Element attributeTag = createAttributeXML(currentEntity,
                attributeName);
            memberTag.appendChild(attributeTag);
        }

        return memberTag;
    }

/**
 * This method creates the XML of the attribute of an entity when saving
 * the details of a Scenario to file.
 *
 * @param currentEntity The entity to get the attribute from.
 * @param attributeName The name of the attribute to retrieve.
 *
 * @return An XML element containing all required information about the
 *         given attribute contained in the entity.
 */
private static Element createAttributeXML(BaseEntity currentEntity,
    String attributeName){
    Attribute currentAttribute = currentEntity.getAttribute(attributeName);

    // create attribute section
    Element attributeTag = sDoc.createElement("attribute");
    attributeTag.setAttribute("name", attributeName);

    int type = currentAttribute.getType();
    attributeTag.setAttribute("type", "" + type);

    switch(type){
    case Attribute.TYPE_INT:
        attributeTag.setAttribute(valueTag,
            Integer.toString(currentAttribute.getIntValue()));

        break;

    case Attribute.TYPE_STRING:
        attributeTag.setAttribute(valueTag,
            currentAttribute.getStringValue());

        break;

    case Attribute.TYPE_DOUBLE:
        attributeTag.setAttribute(valueTag,
            Double.toString(currentAttribute.getDoubleValue()));

        break;

    case Attribute.TYPE_BOOLEAN:
        attributeTag.setAttribute(valueTag,
            Boolean.toString(currentAttribute.getBooleanValue()));

        break;

    default: //set to zero
        attributeTag.setAttribute(valueTag, "0");

        break;
    }
}

```

```

        attributeTag.setAttribute("indexed",
            Boolean.toString(currentAttribute.isIndexed()));

        return attributeTag;
    }

    /**
     * This method writes a fully created XML document to a given file.
     *
     * @param doc The full XML document.
     * @param out The desired file to save to.
     */
    private static void writeXML(Document doc, File out){
        DOMSource source = new DOMSource(doc);
        StreamResult result = new StreamResult(out);
        doXMLTransform(source, result);
    }

    /**
     * This method uses an identity transform to "pretty print" the XML saved
     * to file. Due to the capabilities of the in-built java XML engine the
     * output is not tabbed.
     *
     * @param source The source XML document.
     * @param result The stream to write the result to.
     */
    private static void doXMLTransform(DOMSource source, StreamResult result){
        TransformerFactory tfactory = TransformerFactory.newInstance();
        Transformer trans = null;

        try{
            trans = tfactory.newTransformer();
        }catch(TransformerConfigurationException e){
            ErrorReporter.report(ErrorReporter.FAILED_TRANSFORMER_CONFIG);
        }

        trans.setOutputProperty(OutputKeys.METHOD, "xml");
        trans.setOutputProperty(OutputKeys.ENCODING, "UTF-8");
        trans.setOutputProperty(OutputKeys.INDENT, "yes");
        trans.setOutputProperty("{http://xml.apache.org/xslt}indent-amount", "4");

        try{
            trans.transform(source, result);
        }catch(TransformerException e){
            ErrorReporter.report(ErrorReporter.FAILED_TRANSFORMER_RUN);
        }
    }

    /**
     * This method launches the file dialog from where the user can select a
     * file to save the result of the simulation to.
     *
     * @param fManager The file chooser dialog to launch.
     *
     * @return The file that will be used to save the result to.
     */
    private static File launchSaveOutputFileDialog(final JFileChooser fManager){
        String path = System.getProperty("exedir");
        path = path + "\\sims";
        fManager.setCurrentDirectory(new File(path));

        fManager.setFileFilter(FileIO.getFileFilter(FileIO.XML_FILE));

        int returnVal = fManager.showSaveDialog(null);

        if(returnVal == JFileChooser.APPROVE_OPTION){

```

```
String fname = fManager.getSelectedFile().getAbsolutePath();

if(!fname.endsWith(".xml")){
    fname = fname + ".xml";
}

return new File(fname);
}

return null;
}
}
```