
# Presenting Archaeological Information with Java Applets

R. M. Yorston

**Abstract**

In this paper I look at the use of Java applets in presenting archaeological information. I start by describing the Java language environment, and some of the reasons why it has been the subject of much excitement and hype over the past eighteen months. I then discuss some of the applets which were developed as part of a project, funded by the Nuffield Trust, to perform an integrated analysis of known monuments from the early Neolithic through to the Late Bronze Age in the area around Stonehenge. A discussion of some of the advantages of the technique is followed by mention of some of the problems encountered, and thoughts for the future.

## 1 Java

One of the more exciting developments in computing in recent years has been the invention of the Java programming language by Sun Microsystems. What has caught the attention of many people is Java's ability to generate applets, small programs, which can be downloaded over the Internet and run on Web browsers. This can be used to add dynamic features to Web pages which would be quite impossible to achieve with previous technologies.

Although caught up in the hype associated with the Internet, Java has many features which make it useful in other contexts. In "The Java Language Environment: A White Paper", Gosling and McGilton characterise Java in terms of a list of buzzwords: they claim it is a simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded and dynamic language. Any subset of these could, in the right circumstances, be a powerful argument for using Java. I shall expand on just a few of them.

Java is a language of the C/C++ family. The syntax for expressions and control structures will be familiar to C programmers. In adding object-oriented features, though, Java takes a more radical approach than C++: everything in Java (apart from the basic numeric and character types) is an object. There are no standalone functions, only object methods. It has also avoided some of the more obscure and little-used features of C++, like multiple inheritance.

Although one of the buzzwords was 'interpreted', Java is still a compiled language. Once a program has been written it has to be run through a compiler. Instead of assembler or machine code for a particular processor the compiler generates class files containing Java byte codes. These byte codes are the instruction set for an imaginary processor, the Java Virtual Machine, or JVM. To run Java applications you need an implementation of the JVM for your computer. This implementation interprets the byte codes.

Security is an important consideration where code of unknown provenance is to be downloaded across the Internet and executed on the local machine. The possibility for abuse is obvious. Before execution the code is subject to a verification stage to ensure that no invalid operations can be performed. There are also run-time checks on array bounds which prevent programmers from accessing memory which doesn't belong to them.

One departure from the C/C++ programming model is in the way memory is managed. Java has no concept of a pointer to memory and allocated memory is managed by an automatic garbage collector. This avoids a whole class of difficult-to-find bugs involving dynamic memory allocation, although it does tend to have a cost in reduced performance.

## 2 Applet Technology

One of the major uses of Java is in the development of applets to run in Web browsers. An applet is a small program which can be incorporated into a Web page in much the same way as graphics. A tag in the HTML of the Web page references the main class file of the applet. The executable code of the applet is downloaded along with the text of the page. Once the code arrives from the server it is executed by the browser on the local machine. This allows the programmer to do many things which would be impossible in a normal Web page. Among other things the applet can display dynamic graphics on the Web page, it can allow the user to interact with it and it can connect back to the server from which it came to download further information or query a database there. Some of these capabilities are illustrated in the present work.

The infrastructure required to support the applet is built into the latest versions of Web browsers like Microsoft Internet Explorer and Netscape Navigator. This has a number of consequences. Firstly, the applet will run on any platform on which a Java-enabled browser is available: the environment provided by the browser insulates the applet from the underlying operating system. This is obviously a key feature in allowing use over the Internet, where users may have machines with widely different processors and operating systems. Secondly, the applet code is compact. All of the standard libraries for such things as the graphical user interface and network access are provided on the client

platform. The only code which needs to be transferred across the Internet is that which is unique to the applet.

## 3 The Stonehenge Applets

There are undoubtedly many ways in which applets can be used to present archaeological information. The approach taken here is essentially an extension of the map-based illustration which is a common feature of the literature. This has the advantage of being familiar and easily interpreted. The dynamic nature of the applets provides the opportunity to add new features to a familiar medium. Very similar techniques could be used for graphs and scatter diagrams.

As has been mentioned the applets described here were developed as part of a larger project to re-examine the relationship between the landscape and monuments of the Stonehenge region. They have been used in a number of capacities during this project. Initially they were developed as a means of exploring the data set. The greater part of the data processing was performed on a GIS at the Birmingham University Field Archaeology Unit. The author did not have ready access to these facilities and wrote the applets as a means of visualising data generated in Birmingham. Secondly, applets provided a means of communicating results between the geographically separated members of the multi-disciplinary team working on the project. As well as making the applets available on the Internet they have also been distributed on floppy disks to standalone machines. Finally, the applets provide a means of publishing some of the results of the project. They have already been used in two different ways: as part of a Web site describing the work (http://www.pobox.com/~rmy) and as an adjunct to more traditional visual aids during a presentation at the Liverpool TAG conference. In future it is also conceivable that they may be published on CD-ROM.

The first applet was developed to gain familiarity with the study area and to illustrate the intervisibility of the monuments (See Fig. 1). The display consists of a map of the region around Stonehenge with all of the monuments plotted. Positioning the mouse pointer close to a monument causes the monument name to be displayed in the browser status bar and all the other monuments visible from that point to be highlighted. A separate control panel allows the user to customise the appearance of the display. There are controls to change the backdrop, to turn the highlighting of visible monuments on or off, to enable different forms of highlighting based on barrow type and richness, and to turn the display of different types of monument on or off.

This way of presenting information has a number of advantages over the use of static diagrams.

1. To associate a descriptive legend with each of several hundred points would add considerable clutter to a map. This would typically be addressed by having a number beside each of the points and a separate table associating text with the numbers. Putting the identifying text in the browser status bar is an efficient use of space. One possible enhancement would be to add a text entry field to let the user type in the name of a monument and have that point highlighted.

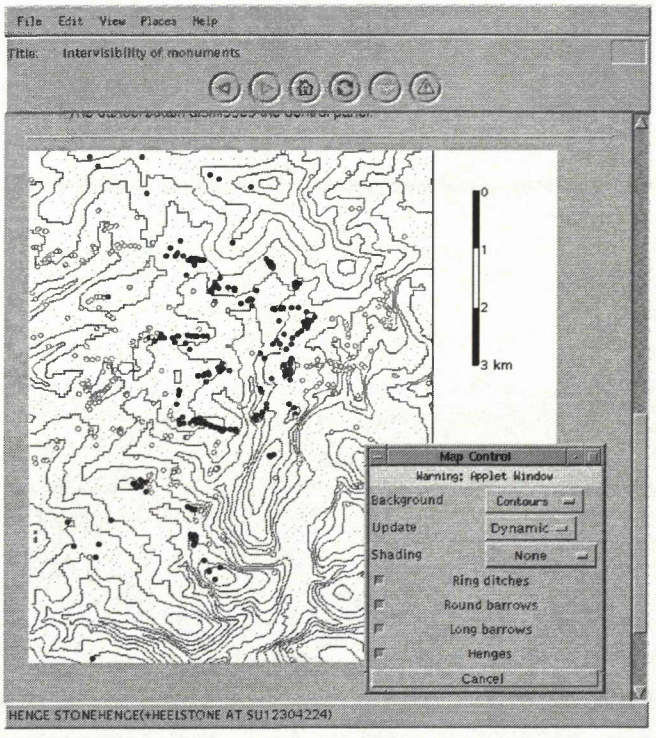


**Figure 1. The basic intervisibility applet, showing the control panel. The highlighted monuments are those visible from Stonehenge.**

2. To display intervisibility information for all of the monuments would require as many maps as monuments, which is clearly impractical. This would normally be dealt with by publishing intervisibility maps only for those monuments considered by the authors to be significant. Publishing intervisibility data for all the monuments in this way clearly avoids any preconceptions on the part of the authors and empowers the viewer.

3. Allowing the user to customise the appearance of the display lets them concentrate on aspects of the data which interest them.

The second applet (See Fig. 2) retains many of the features of the first (hardly surprising since it is based on the same code) but has backdrops showing the viewsheds of each of the henges in the Stonehenge region. This applet was written to investigate the relationship between the visible monuments and the viewshed edges. Monuments within the viewshed are highlighted differently depending on their distance from the viewshed edge. A text box in the control panel lets the user change the relevant distance.

This illustrates a further advantage of the dynamic nature of applets: the viewer can be provided with the means to interrogate the data set. In selecting values for the viewshed distance parameter they are not restricted to pre-computed values selected by the author: arbitrary values can be used. All the code necessary to calculate statistics based on the downloaded data is present in the applet.

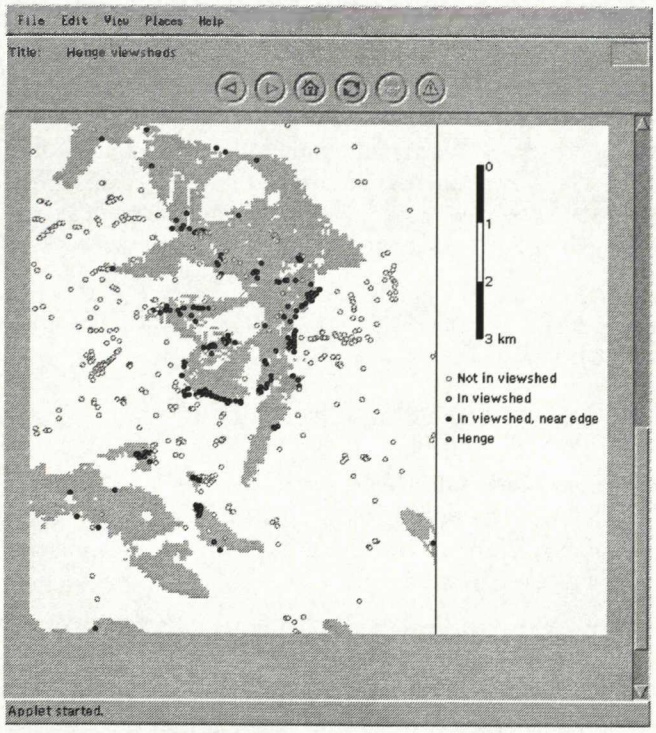


**Figure 2. The henge viewshed applet. The grey area represents the viewshed of Stonehenge. The highlighted monuments are those within 100 metres of the edge of the viewshed.**

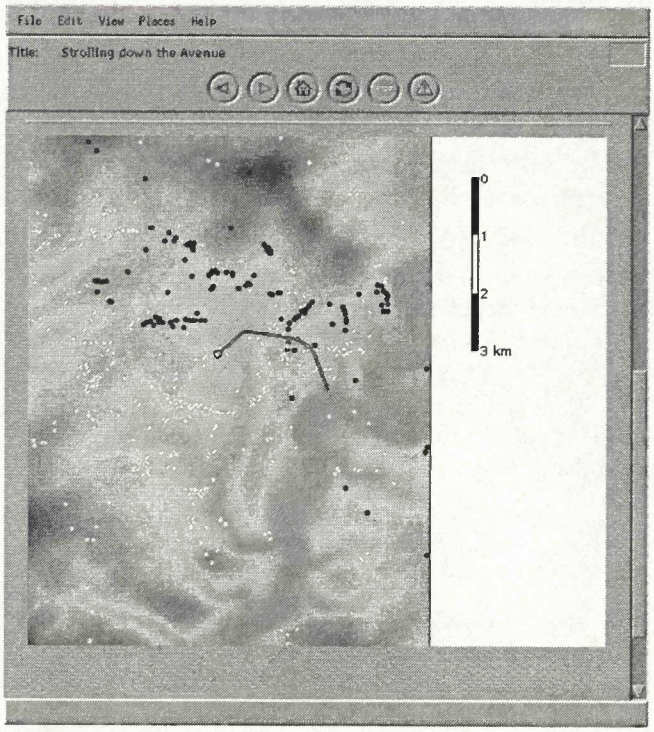


**Figure 3. A snapshot of the animation of a walk along the Avenue. The viewpoint is close to the crest of the King Barrow ridge. The backdrop represents the topography.**

The third applet (See Fig. 3) displays animations of the monuments visible from a number of points along extended features like the Stonehenge Avenue and the Greater Cursus. Again there is a control panel, this time to allow the user to select the backdrop, the path to be animated and the speed of motion.

The use of animation allows the viewer to see how vistas open and close as the viewpoint moves across the landscape. This style of display is well suited to extended monuments which can be considered to delineate processional routes. Similar animation techniques could be used to illustrate temporal development as well a spatial motion.

There are other advantages to the dynamic display of results. The type of display used here is a middle route between static maps and publication of the full data set. Most readers of a paper do not require access to the original data: they will have neither the time nor the inclination to perform a complete reinterpretation. The sort of applets described here allow authors to increase the amount of information they make available without the need for any additional interpretative effort on the part of their readers.

Moreover, there may be sound reasons, such as copyright or commercial confidentiality, why it is not possible for the original data to be published. Often it would be permissible to print a static map of such data because the resolution of the diagram would limit the precision of the information which could be gleaned from it. Where such sensitive data is to be downloaded into an applet for interactive manipulation a similar technique can be applied: the data points can be provided at a precision sufficient for the final display but without revealing the full detail of the original data set.

## 4 Problems

The technology used here is still relatively immature. There are bugs and inconsistencies in the implementation of the Java Virtual Machine and Abstract Windowing Toolkit on different platforms. These manifest themselves in inexplicable failures: sometimes the animation fails to start in Microsoft's Internet Explorer; some of the components of the control panel fail to appear in Sun's HotJava; and attempting to run the applets in early versions of Netscape Navigator cause a fatal crash.

As well as being immature the technology is in a state of flux. At the time of writing version 1.1 of Java had recently been introduced, with version 1.2 on the horizon. When new systems are introduced it takes some time for ports to become available on all platforms, and for all browsers to support them. In addition, there is intense competition between suppliers such as Netscape and Microsoft to develop foundation classes which can be used to build applications. Anyone choosing to develop in Java is currently faced with difficult decisions as to which technologies to learn and use, with the danger that an investment in an unsuccessful technology will be wasted.

Another problem is the speed of the Internet. The applets consist of a number of class files, graphical images and data files. It can take some time for all of these to be downloaded. There are ways of mitigating this limitation of current technology.

Connection setup is often the slowest part of downloading a file from a Web server. The class files can all be combined into one archive file, so avoiding the need to make a number of separate connections to the server. A drawback of archive files is that there are currently three different technologies in

use: uncompressed zip files, Microsoft's cabinet files and Sun's JAR files.

The backdrop images of the henge viewsheds have been combined into one large graphics file, with the appropriate section being displayed for the henge selected by the user. This again avoids the overhead of making connections to transmit multiple graphics files.

Another work around for the slow speed of the Internet is to use the multi-threading capabilities of Java to permit some limited operation of the applet while the required data is being downloaded. Thus data which is vital to the operation of the applet can be fetched first with less significant files, such as the different backdrops, being fetched later. As data arrives the different features of the applet are progressively enabled.

A third technique is to minimise the amount of information which needs to be transmitted.

For example, the positions of the monuments are encoded, not as full grid references, but in terms of their pixel positions within the final display.

In the animation, for each point along the path there is a single bit of data for each monument, to indicate whether it is visible from the point. These arrays of bits have been encoded into a GIF file. Such files are normally used for graphical images, but are used here to transmit raw data because the GIF standard includes compression which considerably reduces the file size. (Version 1.0 of Java includes support for GIFs. Version 1.1 has support for compressed data streams, which provide a more natural and more general solution.)

Downloading the code is not the only limiting factor. Because Java is interpreted it suffers a performance penalty relative to a language which is compiled into native code. This was found to result in the applets having a somewhat sluggish response. Performance was improved by using some of the usual optimisation techniques, such as pre-computing values in advance rather than working them out on the fly. The main improvement in the response of the first applet was obtained by performing the minimum amount of redraw. When highlighting the monuments as the cursor was moved around the display only the monuments whose state changed are now redrawn. This is much faster than trying to redraw everything.

Even with these improvements the response is still inadequate on slow hardware. The situation will improve in future:

1. Java compiler technology will increasingly make use of optimisation techniques.
2. New versions of the Java Virtual Machine are becoming available which employ 'just-in-time' compiler technology to convert Java byte codes into native machine instructions.
3. Hardware is constantly becoming faster.

## 5 Conclusion

The above considerations suggest that the generation of efficient and effective applets currently requires careful design and cunning programming. In future, as these techniques are developed, it may become possible to package the technology so that non-technical users can publish information with applets as easily as they can now produce tables and charts from a spreadsheet.

## Acknowledgements

The work described here forms part of a project sponsored by The Nuffield Trust. The other members of the project team are Sally Exon, Vince Gaffney and Ann Woodward, all of Birmingham University Field Archaeology Unit.

## Bibliography

Flanagan, D, 1996 *Java in a Nutshell* Sebastopol, O'Reilly & Associates
Gosling, J and McGilton, H, 1995 *The Java Language Environment: A White Paper*, Mountain View, Sun Microsystems
Arnold, K and Gosling, J, 1996 *The Java Programming Language*, Reading,Addison-Wesley

**Contact details**
R. M. Yorston
1 Church Terrace
Lower Field Road
Reading RG1 6AS
UK
Email: rmy@pobox.com