THE MYTHS (AND TRUTHS) OF JAVA GAMES PROGRAMMING

Andrew Davison
Department of Computer Engineering
Prince of Songkla University
Hat Yai, Songkhla 90112
Thailand
E-mail: ad@fivedots.coe.psu.ac.th

KEYWORDS

java, games programming, myths, criticisms

ABSTRACT

This paper examines the commonly-expressed criticisms of Java as a games programming language: that's it's too slow, too high-level, prone to memory problems, too hard to install, not available on games consoles, not used in 'real' games, and not even considered a gaming platform by Sun Microsystems. All of these views are incorrect, aside from the console issue.

INTRODUCTION

Java for games programming: are you joking? No, Java is a great games programming language. When you learnt Java, I'm sure it's many advantages were mentioned: an elegant object-oriented paradigm, cross-platform support, code reuse, ease of development, tool availability, reliability and stability, good documentation, support from Sun Microsystems, low development costs, the ability to use legacy code (e.g. C, C++), and increased programmer productivity (Eckel 2006). That list leaves out my personal reason for programming in Java – *it's fun*, especially when you're programming something inherently good-for-you, such as games.

Most Java-bashers tend to skip over advantages, preferring to concentrate on criticisms. Here's a typical list:

- Java is too slow for games programming;
- Java has memory leaks;
- Java is too high-level;
- Java application installation is a nightmare;
- Java isn't supported on games consoles;
- No one uses Java to write 'real' games;
- Sun Microsystems isn't interested in supporting Java gaming.

Almost all of these objections are *substantially wrong*. Java is roughly the same speed as C++. Memory leaks can be avoided with good programming, and techniques like profiling. Yes, Java is high-level, but it also offers more direct access to graphics hardware and external devices. Installation isn't a nightmare, if you use decent installation software. There's a growing number of excellent, fun Java games, and an enormous amount of support available from Sun and Sun-sponsored sites.

If you're keeping count, I haven't disagreed with the lack of a games consoles port, which is a tad embarrassing for a "write once, run anywhere" language. Things may be changing in this category, as I explain below.

A general point about these objections is that they had more validity in the late 1990s, when the language and its libraries were less sophisticated and slower. The 1990's were a long time ago – Java's user and developer communities are currently burgeoning, producing a plethora of useful tools, online help, and code examples.

Now, back to the criticisms...

JAVA IS TOO SLOW FOR GAMES PROGRAMMING

This is better rephrased as "Java is slow compared to C and C++, the dominant languages for games programming at the moment." This argument was valid when Java first appeared (around 1996), but has become increasingly ridiculous with each new release. Some figures put JDK 1.0, that first version of the language, at *20 to 40 times slower* than C++. However, J2SE 5—the current release—is typically only *1.1 times slower*. Many benchmarks indicate that Java SE 6 is about 20% faster than J2SE 5.

These numbers depend greatly on the coding style used. Java programmers must be good programmers in order to utilize Java efficiently, but that’s true of any language. Jack Shirazi's Java Performance Tuning site (http://www.javaperformancetuning.com/) is a good source for performance tips, and links to tools and other resources.

The speed-up in Java is mostly due to improvements in compiler design. The Hotspot technology introduced in J2SE 1.3 enables the run-time system to identify crucial areas of code that are utilized many times, and these are aggressively compiled. Hotspot technology is relatively new, and it's quite likely that future versions of Java will yield further speed-ups. For example, J2SE 5.0 is 1.2 to 1.5 times faster than its predecessor (version 1.4).

Hotspot technology has the unfortunate side-effect that program execution is often slow at the beginning until the code has been analyzed and compiled.

Swing is Slow

Swing often comes under attack for being slow. Swing GUI components are created and controlled from Java, with little OS support; this increases their portability and makes them more controllable from within a Java program. Speed is supposedly compromised because Java imposes an extra layer of processing above the OS. This is one reason why some games applications still utilize the original Abstract Windowing Toolkit (AWT)—it's mostly just simple wrapper methods around OS calls.

Even if Swing is slow (and I'm not convinced of that), most games don't require complex GUIs; full-screen game play with mouse and keyboard controls is the norm. GUI elements maintained by Swing, such as menu bars, button, and text fields, aren't needed, while mouse and keyboard processing is dealt with by the AWT. The latest versions of Java offer a very efficient full-screen mode by suspending the normal windowing environment.

My Program is Slow (Because of Java)

A crucial point about speed is knowing where to lay the blame when a program runs slowly. Typically, a large part of the graphics rendering of a game is handled by hardware or software outside of Java. For example, Java 3D passes its rendering tasks down to OpenGL or DirectX, which may emulate hardware capabilities such as bump mapping. Often the performance bottleneck in network games is the network, not the Java language.

JAVA HAS MEMORY LEAKS

When C/C++ programmers refer to memory leaks in Java, it often means that they don't understand how Java works. Java doesn't offer pointer arithmetic, and typical C-style memory leaks—such as out-of-bounds array accesses—are caught by the Java compiler.

However, these programmers may mean that objects which are no longer needed by the program are not being garbage collected. This becomes an issue if the program keeps creating new objects—requiring more memory—and eventually crashes when the maximum memory allocation is exceeded.

This kind of problem is a consequence of bad programming style, since the garbage collector can only do its job when an object is completely dereferenced, meaning the program no longer refers to the object. A good profiling tool, such as JProfiler (http://www.ej-technologies.com/products/jprofiler/overview.html), can be a great help in identifying code using excessive amounts of memory. JProfiler is a commercial product; many open source profilers are listed at http://java-source.net/; Java SE 6 comes with a great graphical profiler, jhat.

Another memory-related complaint is that the Java garbage collector is executing at poorly timed intervals, causing the application to halt for seconds while the collector sweeps and cleans. The JVM comes with several different garbage collectors, which collect in various ways, and can be selected and fine-tuned from the command line. Information on the performance of the chosen collector can be gathered and analyzed, and Java SE 6 offers many tools for these tasks, including jps, jstat, jhat, and jstack.

JAVA IS TOO HIGH-LEVEL

This complaint is the age old one of abstraction versus speed and control. The details of the argument often include the following statements:

1. Java's use of classes, objects, and inheritance add too much overhead without enough coding benefit;
2. Java's machine independence means that low-level, fast operations—such as direct Video RAM I/O—are impossible.

Statement 1 ignores the obvious benefits of reusing and extending Java's very large class library, which includes high-speed I/O, advanced 2D and 3D graphics, and an enormous range of networking techniques, from lowly sockets to distributed agents. Also forgotten are the advantages of object-oriented design, typified by UML, which makes complex, large real-world systems more manageable during development, implementation, and maintenance.

Statement 2 impacts gaming when we consider high-speed graphics, but it's been addressed in recent versions of Java. J2SE 1.4 introduced a full-screen exclusive mode (FSEM), which suspends the normal windowing environment, and allows an application to more directly access the underlying graphics hardware. It permits techniques such as page flipping, and provides control over the screen's resolution and image depth. The principal aim of FSEM is to speed up graphics-intensive applications, such as games. A lot

of the behind-the-scenes speed improvements in Java SE 6 are related to graphics rendering using OpenGL and DirectX.

Statement 2 also comes into play for game peripherals, such as joysticks and game pads; machine independence seems to suggest that non-standard I/O devices won't be useable. Java games requiring these types of devices can utilize JNI, the Java Native Interface, to link to C or C++, and therefore to the hardware. There's also JInput, a very versatile Java-based game controller API (https://jinput.dev.java.net/).

An interesting historical observation is that the gaming community used to think that C and C++ were too high-level for fast, efficient games programming, when compared to assembly language. Opinions started to change only after the obvious success of games written in C, such as Doom and Dungeon Master, in the mid 1980s. Also important was the appearance of cross-platform development tools that supported C, such as Renderware.

JAVA APPLICATION INSTALLATION IS A NIGHTMARE

The general point made here is that a user needs to be a Java expert in order to install and execute a Java application, whereas most game players just want to point and click on a few dialog boxes to get a game up and running. More specific comments include:

1. Java (specifically, the JRE) has to be on the machine before the application will run.
2. Code bloat—even small programs require a 16 MB JRE. Downloading this can be very slow.
3. Frequently changing JVMs make it hard to write code that will work for every possible version of Java.
4. Non-standard components are often required (e.g. Java 3D), causing even more installation problems.
5. It's not possible to compile the application for a specific platform.
6. The .jar extension is commonly hijacked by other software (e.g. by compression programs) at execution time, meaning that the user can't just double click on a JAR to get it to start.
7. The JRE is slower to start up compared to a native compiled application.

All these problems—aside from 2 and 7 perhaps—can be solved by using good installation software. Java applets can be delivered via the Web, and the Java SE 6 plug-in for Internet Explorer and Netscape starts very quickly. Java Web Start (JWS) can be utilized to download applications, and has been improved significantly since J2SE 1.4. There's numerous third-party installers, such as install4j (http://www.ej-technologies.com/products/install4j/overview.html).

The code bloat comment is increasingly irrelevant, with many games weighing in at over 100 MB, and even many graphics and sound card drivers are larger than 15 MB. Adobe Acrobat requires around 25 MB, Real Player 13 MB, and .NET 23 MB. Network speeds are a problem, especially overseas, but broadband usage is growing rapidly.

Sun Microsystems estimates that around 70% of all new PC's come with a JRE pre-installed, although a game installer must still cater for the others.

There's some truth to point 7, but the slow start-up time is fairly negligible compared to the total running time of an average game. Also, Java SE 6's splash screen feature can be employed to 'entertain' the user during start-up.

JAVA ISN'T SUPPORTED ON GAMES CONSOLES

Unfortunately, this criticism has some justification. Video gaming is a multi-billion dollar industry, with estimates placing revenues at $29 billion by 2007—the market will cater to over 235 million gamers. PCs and game consoles account for almost all the income, but only about 10-20% of it is from PCs, the majority coming from three consoles: Sony’s PlayStation 2 (PS2), Microsoft’s XBox, and Nintendo’s GameCube. Sony is the dominant console maker, having nearly twice as many units in homes compared to Microsoft and Nintendo combined. Microsoft accounts for about 95% of the desktop PC market. Arguably, there are only two important games platforms: the PS2 and Windows—and Java isn't available on the PlayStation.

This problem has long been recognized by Sun: back at the JavaOne conference in 2001, Sony and Sun announced their intention to port the JVM to the PS2. Nothing was ever officially released, although it is possible to run Java on Sony's version of Linux, but the OS requires the PS2 to have a hard disk, and only has limited access to the PS2's other hardware.

The difficulties of this approach should be contrasted to the availability of feature rich C/C++ tools and engines for consoles, such as RenderWare (http://www.renderware.com/) and Gamebryo (http://www.ndl.com/). They have a track record of best-selling games, and can port games across the PS2, Xbox, GameCube, and PCs.

In the future, Java may have a better chance of acceptance into the closed-world of console makers because of two trends: consoles are mutating into home media devices, and the meteoric rise of online gaming. Both require consoles to offer complex networking and server support, strong areas for Java and Sun.

The prospects for Java on the PlayStation 3 (PS3) look fairly bright. Both the basic and premium PS3 versions will have 512 MB of RAM, a large hard drive, will support Linux, and use an extended version of OpenGL. Sony's software development chief, Izumi Kawanishi, has spoken of making it easier for individuals to create games on the PS3. Development kits are expected to appear in Spring 2007.

Applications will be written in a high-level, object-oriented language, but currently there's no word on what it'll be. It's likely that a virtual machine will execute the code, utilizing JIT technology.

The PS3 will include a Blu-ray disc for storing high-definition video and data. All Blu-ray drives support a version of Java called BD-J for implementing interactive menus and other GUIs. Also, Blu-ray's network connectivity means that BD-J can be utilized for networking applications such as downloading subtitles, short movies, and adverts.

The lack of Java on consoles **is** a serious issue, but the remaining PC market is far from miniscule. The Gartner Group believes there are 661 million PC users in 2006. The number will hit 953 million by the end of 2008, and cross over the billion mark in 2009.

Games on PCs benefit from superior hardware—such as video cards, RAM, and internet connections—so can offer more exciting game play. There are many more PC games, particularly in the area of multiplayer online games.

Another rapidly expanding market is the one for mobile games, with sales of $530 million in 2003, potentially rising to $2.5 billion in 2007. There are thought to be around 250 million Java-enabled phones at the moment.

NO ONE USES JAVA TO WRITE REAL GAMES

The word "real" here probably means commercial games. The number of commercial Java games is small compared to ones coded in C++ or C, but the number is growing, and many have garnered awards and become bestsellers: Tribal Trouble, Puzzle Pirates, Call of Juarez, Chrome, Titan Attacks, Star Wars Galaxies, Runescape, Alien Flux, Kingdom of Wars, Law and Order II, Ultratron, Roboforge, IL-2 Sturmovik, Galactic Village, and Wurm Online. Many are written entirely in Java, others employ Java in sub-components such as game logic.

Java is used widely in the casual gaming market, where game-play is generally less complex and time-consuming. Implementation timelines are shorter, budgets smaller, and the required man-power is within the reach of small teams. By 2008, industry analysts believe the casual games market will surpass $2 billion in the US alone.

There are numerous Java gaming sites, including a showcase at Sun Microsystems (http://www.java.com/en/games/), community pages at http://community.java.net/games/, a collection of open-source gaming tools at https://games.dev.java.net/, the Java Games factory (http://javagamesfactory.org/), works-in-progress at https://games-forge.dev.java.net/, and many, very helpful forums at http://www.javagaming.org/.

There are several excellent books on Java games programming (Brackeen *et al.* 2003, Clingman *et al.* 2004, Croft 2004, Davison 2005).

SUN MICROSYSTEMS ISN'T INTERESTED IN SUPPORTING JAVA GAMING

The games market isn’t a traditional one for Sun, and it'll probably never have the depth of knowledge of a Sony or Nintendo. However, the last few years have demonstrated Sun's increasing commitment to gaming.

J2SE has strengthened its games support through successive versions: version 1.3 improved its graphics and audio capabilities, and version 1.4 introduced full screen mode and page flipping in hardware. Faster I/O, memory mapping, and support for non-block sockets, which is especially useful in client/server multiplayer games, also appeared first in 1.4. Version 5.0 has a decent nanosecond timer at last. Java extension libraries, such as Java 3D, the Java Media Framework (JMF), the Java Communications API, Jini, and JAXP (Java’s peer-to-peer API) all offer something to games programmers. Java SE 6 has improved graphics rendering speeds, and offers new features useful for gaming, such as splash screens, scripting, and a desktop API.

Sun started showing an interest in gaming back in 2001, with its announcement of the Java Game Profile, a collaboration with several other companies, including Sega and Sony, to develop a Java gaming API. The profile was perhaps too ambitious, and was abandoned at the end of 2003. However, it did produce three game-focused technologies: a Java binding for OpenGL called JOGL, a binding for OpenAL (a 3D audio library) called JOAL, and JInput.

Part of the 2001 initiative was the creation of the JavaGaming.org website (http://www.javagaming.org), initially manned by volunteers. In 2003, the Game Technology Group was formed, and JavaGaming.org received a substantial makeover as part of the creation of the new java.net portal (http://www.java.net) aimed at the technical promotion of Java. java.net hosts many discussion forums, user groups, projects, communities, and news. The communities include: Java Desktop, Java Education and Learning, Java Enterprise, and Java Games.

The Java Games community pages can be accessed through http://www.javagaming.org or

http://community.java.net/games/. The site includes Java games forums, projects, news, weblogs, a wiki (http://wiki.java.net/bin/view/Games/WebHome), and links to games affiliates.

Numerous Java game forums can be accessed from http://www.javagaming.org/forums/index.php. These are probably the best sources of technical advice on Java gaming on the Web, with over 8500 highly opinionated registered users. Discussion topics include Java 3D, Java 2D, Java Sound, J2ME, networking, online games development, performance tuning, JOGL, JOAL, and JInput. There are also sections on projects and code examples.

The project sections (https://games.dev.java.net/) mostly concentrate on JOGL, JOAL, and JInput, but the games-middleware and games-forge sections are wider ranging. The games-forge projects include Chinese chess, jbantumi (a strategic game from Africa), and an online fantasy football management system.

Sun's substantial presence at http://community.java.net/games/ is mostly as a host for community forums and open source projects (or projects with licenses very close to open source). The projects include JOGL, JOAL, JInput, and Java 3D. Sun is relying on community involvement to move these projects forward, since the Game Technology Group is quite small (Twilleager *et al.* 2004).

One in-house product is Sun's Project DarkStar (http://games-darkstar.dev.java.net), aimed at developing tools for supporting massive multi-player online games. The Sun Game Server (SGS) is it's server-side platform, and there are client APIs for C++, Java SE, and Java ME

REFERENCES

Brackeen , D., Barker, B., Vanhelswue. L. 2003. *Developing Games in Java*, New Riders Games; August.

Clingman, C., Kendall. S., and Mesdaghi . S. 2004. *Practical Java Game Programming*, Charles River Media, June

Croft, D.W. 2004. *Advanced Java Game Programming*, Apress, April.

Davison, A. 2005. *Killer Game Programming in Java*, O'Reilly Media, May.

Eckel, B. 2006. *Thinking in Java*, Prentice Hall, 4th ed., February

Twilleager, D., Kesselman, J., Goldberg, A., Petersen, D., Soto, J.C., and Melissinos, C. 2004. "Java Technologies For Games", *ACM Computers in Entertainment*, Vol. 2, No. 2, April.

BIBLIOGRAPHY

ANDREW DAVISON received his Ph.D. from Imperial College in London in 1989. He was a lecturer at the University of Melbourne for six years before moving to Prince of Songkla University in Thailand in 1996.

His research interests include scripting languages, logic programming, visualization, and teaching methodologies. This latter topic led to an interest in teaching games programming in 1999.

His O'Reilly book, *Killer Game Programming in Java*, was published in 2005, accompanied by a website at http://fivedots.coe.psu.ac.th/~ad/jg/.