# Minueto

# Student Software Engineering Project Courses Become Fun

Alexandre Denault and Jörg Kienzle
Software Engineering Laboratory, School of Computer Science
McGill University, Montreal, Quebec, Canada
Alexandre.Denault@mail.mcgill.ca, Joerg.Kienzle@mcgill.ca

## Abstract

*This paper presents Minueto, a cross-platform, Java-based game development framework specifically designed for undergraduate software engineering project courses. Minueto has been designed to hide complex graphics programming (full-screen mode, double-buffering, hardware acceleration), and keyboard/mouse input handling behind simple-to-use objects. Despite of being implemented in pure Java, Minueto achieves frame rates of over 60 frames-per-second on mid-range Windows, Linux and Macintosh systems. The large quantity of documentation, tutorials and sample code allows students familiar with basic Java programming to start developing games after a very short learning period. The effectiveness of Minueto is discussed based on 3 years of student evaluation.*

## 1. Introduction

Most university computer science curriculae includes at least one project course, where undergraduate students have to implement a considerable piece of software. The School of Computer Science at McGill University is no exception. In the course "COMP-361 Systems Development Project", students must "implement a large body of software" to gain substantial hands-on experience in object-oriented software development, multithreaded programming and distributed systems. The work is performed in groups of typically 4 or 5 students, which, as a side effect, improves the students' group communication and organizing skills.

Finding project topics which are interesting, challenging, motivating and feasible to complete in one semester is a difficult task. In order to prevent students to "reuse" source code of previous years, the application to develop has to be considerably different from year to year. On the other hand, the applications developed in the following year should be sufficiently similar to the previous one in order to keep the overhead for the professor and the teaching assistants under control.

It turns out that *computer games*, especially simple 2D action or turn-based games, are applications that are particularly well suited for a semester-long student project. Games are ideal for object-oriented analysis and design, since very often games contain abstractions of real-world objects (for instance, players, items, cities, space ships, planets, etc.) with well defined relationships and responsibilities (for instance transporter ships can carry land units, wormholes connect a set of star systems). Also, game rules are usually intuitive, not too complicated and therefore easy to understand. Local rules, i.e., game rules for specific game objects, can often be encapsulated within the object itself (for instance, a tank can make sure that it does not move over water, an airplane crashes when it runs out of fuel, etc.). Finally, computer games are very popular among students.

Using game programming to teach software engineering is not a new idea. Rudy Rucker, of San José State University, teaches software engineering using games as context for implementation [1]. Joe Warren, of Rice University, teaches a class where students are required to work as a team to complete a large-scale game project [2].

Creating a game, however, is very challenging. In addition to implementing the game rules correctly, the user interface of a game has to be intuitive, appealing and responsive. Performance is a key issue. Sluggish games are not "fun". In order to achieve smooth animations, screen updates of around 30 frames-per-second (fps) are required. Achieving such performance requires using advanced techniques such as double buffering, and in-depth technical knowledge of how to exploit hardware acceleration provided by modern graphic cards.

This article presents *Minueto*, a game development framework targeted at computer science / software engineering undergraduates. It allows students to rapidly de-

velop non-trivial games by simplifying game-programming concerns such as graphics and player input. As a result, students spend most of their time acquiring the essential skills of software development, namely object-oriented analysis and design, and stay motivated thanks to the "fun" factor of game development.

The rest of the paper is structured as follows. Section 2 lists the goals that were established for *Minueto*. Section 3 presents *Minueto*'s architecture. Section 4 shows a detailed performance evaluation of *Minueto* based on two benchmarks. Section 5 describes *Minueto*'s extensive documentation. Section 6 shows the beneficial effects *Minueto* had on the students of the McGill software engineering project course. Section 7 discusses related work, and the last section draws some conclusions.

## 2. Goals Of Minueto

To be suitable for game development in an academic undergraduate setting, the following goals were established for the Minueto framework:

1. The framework should be *easy to learn*, and *provide fast results*. Our experience has shown that students are more likely to use a framework if they get visually attractive initial results within a few hours of programming.
2. The framework should be *easy to install* and *platform independent*. Although the School of Computer Science at McGill provides all the resources students require to complete a game programming project, some students want to work at home on their own machines. Platform independence also makes it easier to offer downloadable versions of the games of previous years on the course web page.
3. The framework should provide *reasonable performance*, i.e. at least 30 fps screen updates on current mid-range computer systems.
4. The framework should *not* be *game specific*. Since the course project changes every year, it is important that the framework be flexible.

To answer requirements 1 and 4, it has been decided that *Minueto* only provides basic game programming features. Frameworks with an extensive feature set are difficult to learn, mostly because of the sheer number of options offered to the programmer. Most of the time, advanced features are incorporated into frameworks to provide game-specific functionality. We decided with *Minueto* to provide only the basic building blocks, but to make sure that the students can use them to build more advanced features, if necessary.

To address requirement 1, extensive and intuitive documentation for *Minueto* has been developed (see section 5).

Finally, to address requirement 1 and 2, *Minueto* has been developed in Java [3]. Java is currently the programming language used in most undergraduate classes offered at the School of Computer Science of McGill University. It is inherently cross-platform, and therefore runs among others on Windows, Linux and Macintosh systems. Despite the fact that Java uses garbage collection and runs on a virtual machine, we were able to achieve impressive performance (see section 4) and meet requirement 3.

## 3. Minueto Architecture

*Minueto* is designed to be a modular framework. Its core components include a 2D graphics engine and game input handling functionality. This functionality is complemented by additional services in form of expansion modules, such as sound and network support. The expansion modules are still under development and hence not described in this paper.

### 3.1. 2D Graphic Engine

*Minueto*'s core component includes a 2D graphic engine designed for raster/bitmap graphics. The core classes of this module are `MinuetoWindow` and `MinuetoImage`. `MinuetoWindow` is a drawing canvas that contains several graphic programming enhancements, such as the double buffering system. It also insures that images are drawn with the correct color depth and are hardware accelerated. Optimizations specific to windowed-mode display or full screen display are contained in its two subclasses, `MinuetoFrame` and `MinuetoFullscreen`, in order to provide the smooth and flicker-free performance required for a game programming framework. In addition, a `MinuetoPanel` subclass is currently being developed to improve on *Minueto*-Swing interoperability.

`MinuetoImages` are the basic blocks of a *Minueto* application. The different types of images available are sub-classes of the `MinuetoImage` class. The class hierarchy is shown in Figure 1. The `MinuetoImageFile` class is used to load images from files. Various popular image formats, such as JPEG, GIF and PNG are supported, as are transparency channels found in several image formats. The `MinuetoRectangle` and `MinuetoCircle` classes are used to create images of rectangles and circles respectively. *Minueto* includes functions to scale, rotate or crop images. These functions are built into the `MinuetoImage` class and return new transformed copies of the image. The original image, however, remains unchanged. Complex new images can be created dynamically by drawing several simple images on a common blank

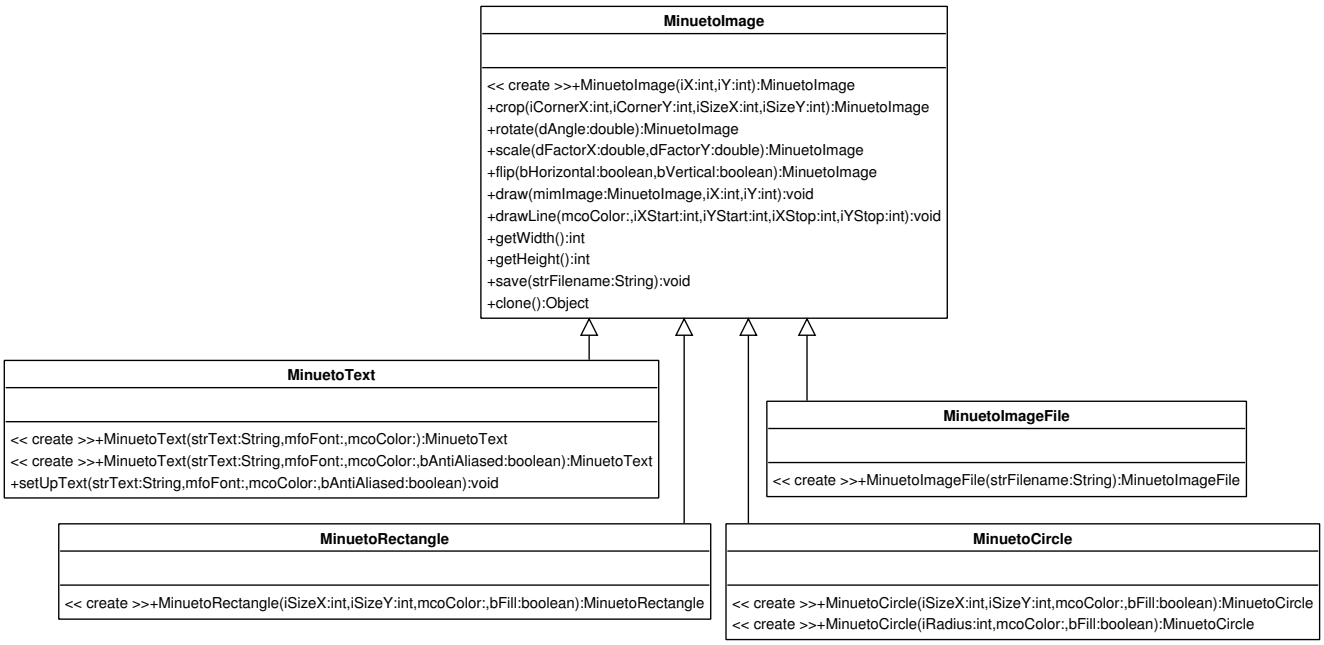


Figure 1. The Minueto Image Hierarchy

`MinuetoImage`.

### 3.2. Keyboard and Mouse Input

In a traditional GUI application, keyboard/mouse input is handled by an intuitive event-based system. A GUI framework, such as Swing [4], usually provides a multitude of common GUI components, such as panels and buttons. A programmer can instantiate the desired components and position them on the screen. To define the behavior that is executed when a component is activated, the programmer has to implement a pre-defined interface, often called a *listener*, within an object and register that object with the GUI component. At run-time, the GUI framework is in control. When a user presses the keyboard or moves the mouse, an appropriate event is created. Then, the main thread (or a thread created just for this event) finds the GUI component that the event was addressed to, which in turn dispatches the call to the *listener* code provided by the programmer.

The event-based way of handling user input is very intuitive. Students that learnt Java as their first programming language have been exposed to event-based systems since the beginning of their studies. In addition, *listeners* make it possible to implement behavior in a fine-grained way, and to distribute that behavior over several objects. Rigorous object-oriented analysis and design approaches assigns well-defined responsibilities to objects, and using listeners behavior that falls into the responsibilities of an object can be encapsulated within, which results in a design with high cohesion.

Unfortunately, the event-based way of handling user input can not be used in this form in a game environment. Games have to deliver steady performance. In a classical event-based system, the programmer has no control over when events are handled (and which thread handles them). A sudden burst of input events could delay screen updates temporarily, and hence result in sluggish game play.

To avoid this situation, games keep control of the processor at all times. A classical game implementation usually contains a main loop as shown in Figure 2 that executes over and over again.

Within the loop, the first step is to check for new user input. Based on the input, the game state is updated. In non-object-oriented systems, behavior is usually implemented in a massive switch statement containing hundreds of case blocks, which are very difficult to maintain. To prevent irregular performance, excess input events are either discarded or postponed to the next cycle. In the next step of the game loop, the game time is advanced and the game state updated accordingly. Finally, the next screen image is created based on the new game state and displayed on the screen.

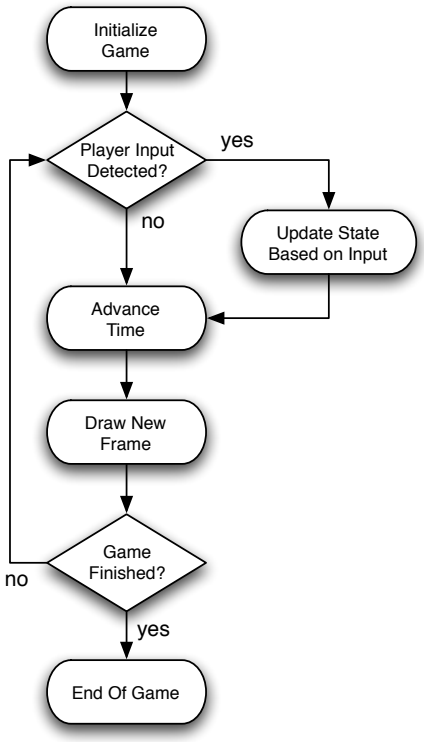


Figure 2. Simple Gameloop Control Flow

### 3.3. Minueto Input

Minueto input combines the flexibility of event-based systems while giving the programmer fine-grained control over input handling during the main game loop. In the background, invisible to the user, a thread monitors all inputs devices. Whenever a user presses a key or uses the mouse, the thread creates a corresponding event and places it into a first-in-first-out queue, an instance of `MinuetoEventQueue`.

To define the behavior for a certain type of input, a programmer can implement a predefined interface, for example `MinuetoMouseHandler`, and register it with the queue. This technique makes it possible, for instance, to separate mouse handling from keyboard handling. Handlers can also be switched at run-time, which makes it possible to achieve different behavior depending on the current view of the game. With a little effort, elegant design patterns such as model-view-controller [5] can be implemented on top of handlers.

However, handlers are not automatically executed when a corresponding input event is put into the queue. The main game thread performing the game loop (or game threads, if desired) must call the `handle()` method of the queue whenever it is ready for input processing. Each call to `handle()` processes one single event. When an event is processed, the appropriate method in the registered handler is called. For example, when a key press is processed, the `handle()` method executes the `keyPress()` method in the registered keyboard handler.

## 4. Performance Evaluation

Given the interpreted nature of Java and the slow performance of Swing when used for game development, we conducted a series of performance tests on the Java 2D engine before committing to a full Java implementation of *Minueto*. The results of the benchmark are also representative for the current version of *Minueto*.

### 4.1. Benchmarks

The first benchmark, *BlackWizardGrass*, features a small character sprite that can be moved over a field of grass using the keyboard. The field of grass is a tile map composed of grass tiles of 32x32 pixel size. The grass field is redrawn, tile-by-tile, at every frame. The sprite character is animated using eight 32x32 pixel tiles, each of which depicts a different orientation or step in the walk cycle of the character. One of these tiles is displayed at each frame, depending on the character's current orientation and step. The benchmark runs in a 800 by 600 pixel window.

The second benchmark, *TownMap*, is more elaborate. It features three small character sprites walking over a slightly more complex tile map. The tile map is composed of several different 32x32 pixels tiles, some of them depicting the edge of a small cliff. The user can control one of the characters. The two other characters move randomly. Like in the previous benchmark, each of the characters has eight animation tiles. A screenshot of the *TownMap* benchmark is shown in figure 3[1].

### 4.2. Benchmark Results

The benchmarks were run on 15 computers, all equipped with a variety of processor, operating system and video hardware. Both benchmarks were run in fullscreen and in windowed mode[2]. The results for windowed mode are shown in the top part, the results for fullscreen mode in the bottom part of figure 4. During startup, all benchmarks showed large instabilities in frame rates. However, the frame rate was only recorded after some seconds, once the game had stabilized.

*Minueto*'s performance on mid and high-end systems was more than satisfying. The only disappointing perfor-

[1] The sprite graphics used in the *TownMap* benchmark are copyright of their respective owners and are only used for demonstration purposes.

[2] Fullscreen mode was at that time not supported under Linux, hence fullscreen performance data is unavailable for that operating system.

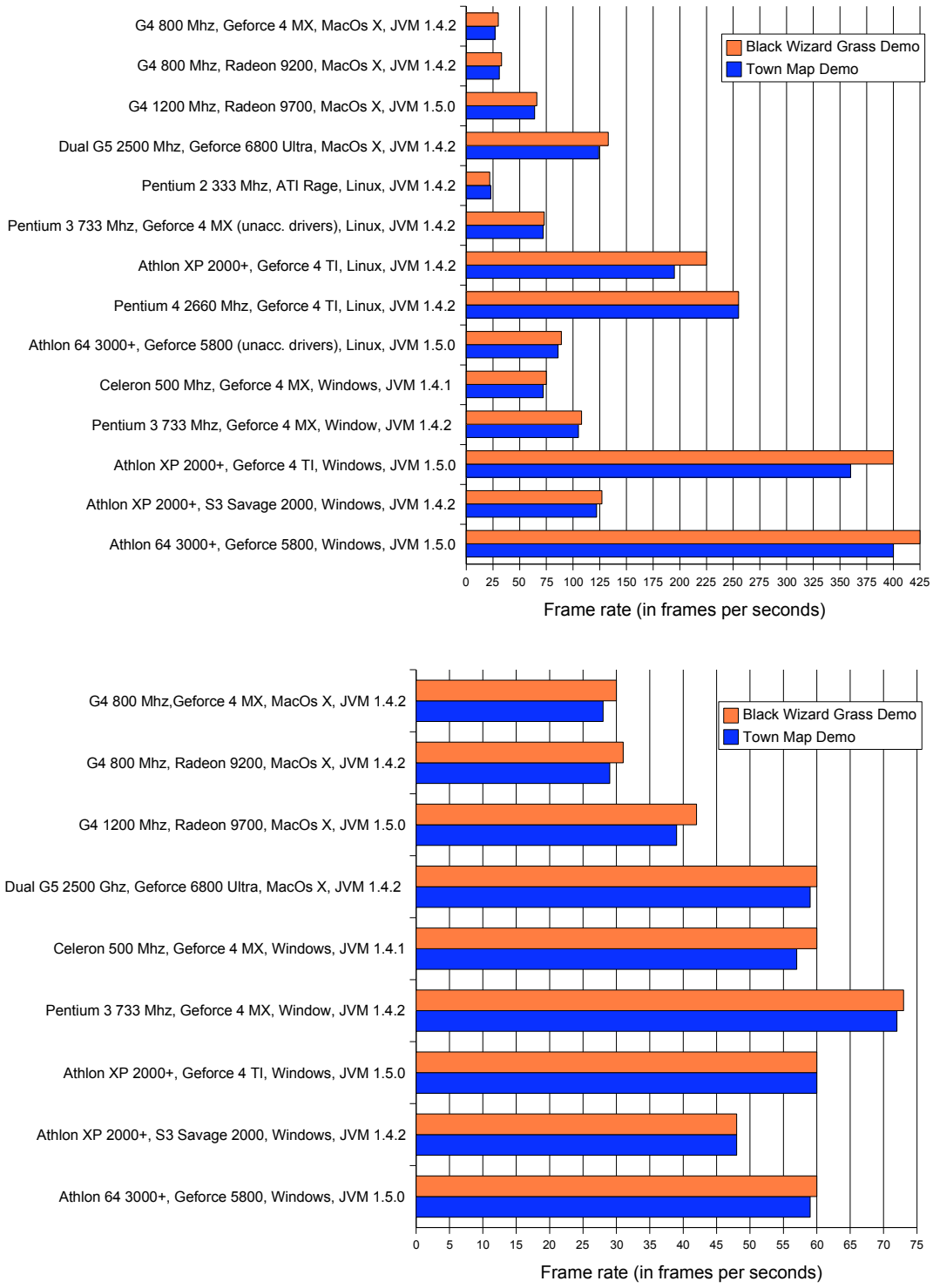


Figure 4. Frame Rates for Black Wizard Grass and Town Map in Windowed (top) and Fullscreen (bottom) Mode


Figure 3. Screenshot of the *TownMap* Benchmark

mance, namely 22 fps, was achieved on a rather low-end system, a Pentium 2 333 MHz with an ATI Rage graphics card running JVM 1.4.2 under Linux. All other systems reached or exceeded the desired 30 fps that are required for visually smooth animations. The top performance was achieved on a Athlon 64 3 GHz with a GeForce 5800 graphics card running Windows and JVM 1.5.0. On this machine, the first benchmark achieved 425 fps, whereas the second one reached an impressive 400 fps.

Several other interesting conclusions can be drawn from the collected results:

- Frame rates for fullscreen are capped at 60 or 75 fps. This suggests that the frame rate for an application in fullscreen mode is limited by the refresh rate of the screen. This is not a problem since a frame rate higher than the refresh rate of the screen is useless.
- The Athlon XP 2000+ equipped with the S3 Savage 2000 video card suffers from poor performance, especially when compared to its GeForce 4 TI counterpart. This would suggest that poor video hardware can have a significant impact on *Minueto*.
- The Athlon64 3000+ using unaccelerated drivers provided a very bad performance under Linux. However, the Pentium 3 733 MHz, using the same drivers, offered a similar performance. This suggests that the performance of the Linux unaccelerated drivers is capped.
- Relatively poor performance was obtained by the 800 MHz MacOs X computer. Although it is already known that the Apple JVM is slower than the Sun JVM, faster MacOS X computers do offer reasonable frame rates when using Minueto. The 1.2 GHz Powerbook presented a surprisingly high (for MacOS X) frame rate. This suggests that the Apple's JVM 1.5 does offer an important increase in Java 2D's speed.

## 5. Minueto Documentation

Students have different learning habits, and a good documentation should account for that. This is why *Minueto*'s documentation is available in three different formats: tutorials, examples and the API specification.

### 5.1. Tutorials

On the *Minueto* website (`http://minueto.cs.mcgill.ca`) students can find a collection of tutorials. By reading these *HowTo* documents, students can gain insight on how to achieve basic tasks with *Minueto* and how to solve common problems. The tutorials explain, using illustrations, every specific game programming aspect, such as double buffering, and the way that functionality is provided in *Minueto*. This type of documentation targets students who prefer learning by reading books and manuals.

### 5.2. Sample Code

It has been shown that the difference between a novice and an expert chess player is the fact that the latter has thousands of board configuration stored in his long time memory [6]. Expert players can use these memorized board configurations to derive their next move without having to rely too much on their limited working memory. Further research has shown that problem solving relies more on stored memories than complex reasoning [7]. This theory can easily be extended to programmers, where the difference between a novice and an expert is years of problem solving experience [8]. An experienced programmer can draw upon years of previous programming challenges to find similarities between previous and current problems.

The *Minueto* documentation provides this "experience" to novice game programmers by means of a multitude of code examples that cover a wide range of topics. Each example is designed to solve exactly one problem, for instance, how to draw a rectangle on the screen. This allows the code for the examples to be as short as possible, typically less than 10 lines of code, thus making it very easy for students to understand. Each sample code is a complete stand-alone application that can be compiled and run as is. As a result, the student can instantly gain hands-on experience with the *Minueto* framework, and start experimenting with *Minueto* by making small modifications to the provided code.

### 5.3. API Specification

The last type of documentation is the API specification. Students that have already taken a Java class are already familiar with the Java way of describing APIs. This is why *Minueto*'s API documentation is built using the standard JavaDoc tool and follows the documentation guidelines for Java applications as outlined by Sun Microsystems [9]. A detailed API specification is essential, especially in later phases of the project.

## 6. Student Evaluation

### 6.1. Background

*Minueto* has been developed as an infrastructure for the COMP-361 Systems Development Project class taught at the School of Computer Science at McGill University. The class was first taught in Winter 2004 as part of the new Software Engineering program. The 38 students had to implement a computerized version of Battleships.

Since *Minueto* was not yet available at that time, the students were not given any particular game programming framework. They were allowed to use the programming language and platform of their choice. Most of the students decided on using the Java programming language and implemented their user interface based on Swing, some used C++ and DirectX, or Python and PyGame (a game development framework for Python).

Students were required to attend weekly meetings with the professor to monitor their progress. Three months into the development of their projects, students were called to a special meeting to present a first "demo" – a working version of their game – with a minimal set of features.

The weekly meetings, the final game implementations and the feedback from the students showed that working on a game is indeed very motivating for most students. Unfortunately, though, most teams struggled with graphic programming and graphic performance issues. The technical difficulties experienced by some groups did not give them enough time to deliver a fully functional product. Based on these observations, the development of *Minueto* was initiated.

### 6.2. Effects of Using Minueto

In the next session of the course in Winter 2005, we offered the 36 students the possibility to use *Minueto* to implement their project – a turn-based strategy game played on land and sea. Although the students were free to use the technology and programming language of their choice, they were warned that technical support would only be provided for Java and *Minueto*. This was a practical decision, since actively supporting multiple game development platforms is a heavy burden on both the professor and his teaching assistants.

In the first week of classes already, a small *Minueto* demo application was posted on the class bulletin board by a student. After the class presentation of *Minueto* in week 3, about 60% of the class decided on using *Minueto*.

This had a very positive effect on the weekly group meetings. The number of technical complaints and problems dramatically decreased, and more time was spent on discussing architectural and design decisions. An increase in the quality of the projects was also noticed during the "demo" and final presentations. The creativity of some of the student's projects was amazing. Several of *Minueto*'s features had been used in ingenious ways, expanding our vision of *Minueto*'s capabilities. Although the top projects from 2005 were not necessarily superior to the top projects from the previous year, the quality of the average projects had greatly increased. The games looked more professional and polished, and the *Minueto*-using groups were able to present a product that provided all of the required functionality.

Over the last year, based on the feedback from the students of Winter 2005, *Minueto* matured further. Additional features such as support for transparency were added. The current version of *Minueto* is so convincing that 100% of the 33 students of Winter 2006 used it to implement their project – a turn-based space game. The students were extremely happy with Minueto's features and performance.

### 6.3. Student Suggestions

Some students would have liked to see elaborate Swing-like GUI components offered in *Minueto*. Although such components are often used in computer games, it is a non-trivial task to integrate them into a game programming framework. Care must be taken to not degrade performance, and to continue to provide a clean, simple and easy to learn interface. We are currently evaluating a possible *Minueto*-Swing integration, or else plan to provide simple GUI components, such as buttons and textboxes, directly in *Minueto*.

## 7. Related Work

Although a multitude of game development frameworks and tools have emerged in the past years, most of them have been designed for the professional industry. These game development kits or game engines, as they are often called, are extremely powerful, providing sophisticated 3D graphics, real-time physics and AI scripting languages. Their difficult learning curves and their price tag, however, make them ill-suited for the academic world. A review of

open source game development frameworks such as SDL [10] has shown that they too are complicated to learn, and not suited for a typical undergraduate course. Some game development kits also often force the developer to use a particular programming language or a specific development platform.

One notable exception is the Panda3D game engine, which is specifically designed to have a short learning curve and promote rapid development [11]. However, Panda3D was primarily designed to create 3D virtual worlds. The added complexity of a 3D environment and animation would adversely shift the focus away from the software engineering tasks. In addition, Panda3D is only available on specific operating systems.

Several teaching tools for game programming are also available. With their visual interface and custom scripting languages, Alice [12] and GameMaker [13] are designed to provide the best introduction to programming for high school and college students. However, our target population, the students of COMP-361, have alread completed at least three programming courses at the undergraduate level: introduction to computer science, introduction to software systems, and algorithms and data structures. Rather than teaching them new programming techniques, we would like students to gain experience in developing an object-oriented application of considerable size using the techniques they already learnt.

## 8. Conclusions

Developing a game is very motivating for students, but without a development framework, students have to master advanced programming techniques to deliver smooth graphics. Fighting with many low-level technical problems or sluggish gameplay removes the fun and prevents the students from focussing on software architecture and design.

In this paper we presented *Minueto*, a cross-platform, Java-based game development framework specifically designed for undergraduate software engineering project courses. *Minueto* simplifies game development for students by hiding complex game programming details (full-screen mode, double-buffering, hardware acceleration) behind a simple-to-use interface. The large quantity of documentation, tutorials and sample code allows students familiar with basic Java programming to start developing games after a very short learning period. Students can get visually appealing results with very little effort. Despite of being implemented in pure Java, *Minueto* achieves frame rates of over 60 frames-per-second on mid-range Windows, Linux and Macintosh systems. The purpose of *Minueto* is not to teach programming, but to reduce the burden of game development, and allow students to focus on software development tasks such as object-oriented design.

The experience gained by introducing *Minueto* into the COMP-361 Systems Development Project course at McGill University has demonstrated that *Minueto* lives up to our expectations. It reduced the amount of time that was spent during weekly meetings discussing graphic-related technical issues, and improved the overall quality, look and performance of the final deliverables.

## 9. Acknowledgments

We would like to thank the students of the course COMP-361 Systems Development Project taught at the School of Computer Science of McGill University in Winter 2004, Winter 2005 and Winter 2006 for their enthousiasm and constructive feedback. Without their help the development of *Minueto* would not have been possible.

## References

[1] R. Rucker, *Software Engineering and Computer Games*. Addison Wesley, 2003.

[2] S. Schaefer and J. Warren, "Teaching computer game design and construction," *Computer-Aided Design*, vol. 36, December 2004.

[3] J. Gosling, B. Joy, and G. L. Steele, *The Java Language Specification*. The Java Series, Reading, MA, USA: Addison Wesley, 1996.

[4] Sun Microsystems, "Java Foundation Classes (JFC/Swing)." http://java.sun.com/products/jfc/, 2006.

[5] G. E. Krasner and S. T. Pope, "A cookbook for using the model-view-controller user interface paradigm in smalltalk-80," *Journal of Object-Oriented Programming*, vol. 1, pp. 26 – 49, August 1988.

[6] H. Simon and K. Gilmartin, "A simulation of memory for chess positions," *Cognitive Psychology*, 1973.

[7] C. W., "Using worked examples as an instructional support in the algebra classroom," *Journal of Education*, 1994.

[8] Garner and Stuart, "Reducing the cognitive load on novice programmers," *Association for the Advancement of Computing in Education (AACE)*, p. 7, June 2002.

[9] Sun Microsystems, "How to write doc comments for the javadoc tool." http://java.sun.com/j2se/javadoc/writingdoccomments/, 2000.

[10] "Simple Directmedia Layer." http://www.libsdl.org/, 2006.

[11] Carnegie Mellon University, "Panda3D." http://www.panda3d.org/, 2006.

[12] Carnegie Mellon University, "Alice." http://www.alice.org/, 2006.

[13] M. Overmars, "Gamemaker." http://www.gamemaker.nl/, 2006.