Minueto, a Game Development Framework for Teaching Object-Oriented Software Design Techniques

Alexandre Denault Jörg Kienzle
School of Computer Science, McGill University

This paper presents Minueto, a Java game development framework specifically designed for undergraduate students. It is a multi-platform framework who's goal is to simplify the game development process by encapsulating complex programming tasks such as graphics, audio and keyboard/mouse programming into simple to use objects. This simple design and the large quantity of documentation allow students to start developing games after a very short learning period.

At McGill University, the system development project course requires students to implement a large non-trivial turn-based game. Minueto was developed to help students focus on the design their project and avoid common game programming pitfalls such as graphics and sound. Since its introduction in winter 2005, there has been an impressive increase in the quality of student's projects.

1. Introduction

Finding project topics which are both challenging and interesting for students is a difficult task for any university professor. Computer games have emerged as a popular topic among students. However, most game development tool are design for the professional industry. Their difficult learning curves make them ill-suited for the academic world. Thus, there is need for specialized academic game development tools.

We present Minueto, a game development framework targeted for the computer science undergraduate population. It allows students to rapidly develop non-trivial games by simplifying several game-programming concerns such as graphics, sound, networking and player input. Thus, professors can ask the students to implement a game-related project knowing that students can focus on the behavior of the game.

2. Framework Goals

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 [Ruc03]. Joe Warren, of Rice University, teaches a class where students are required to work as a team to complete a large-scale game project [SW04].

At McGill University, the system development project requires students to implement a non-trivial turned-based computer game. Students are free to choose the object-oriented technology they will use to complete their project. The course itself has three main goals: teach students how to work in teams or three to four students, force students to make design decisions by providing incomplete specification and give students the opportunity to work a on large object-oriented project.

The topic of game development was chosen because of its popularity among the students. However, this is not a course about game developments. Technical game development issues such as frame-rate and double-buffering are outside the scope of the course and are not discussed. As such, the course requires a tool that will abstract these technical details, allowing students to focus on the design aspects, as outline in the three course goals.

To answer the course needs, the following goals were established for the Minueto framework:

1) The framework should not be game specific. Since the course project changes every year, it is important that the framework be flexible.

2) The framework should be easy to learn. Frameworks with an extensive feature set are difficult to learn, mostly because of the sheer number of options offered to the programmer. A simple API is easier to learn.

3) The framework should provide fast results. Students are motivated by visual results. Our work has shown that students are more likely to accept the framework if they can easily produce results.

4) The framework should be properly documented. This was the major drawback of previous course specific tools used by the course. An undocumented API is difficult to learn.

5) The framework should be reasonability fast. Even though the game is turn-based, students might want to include real time animations into their game.

6) The framework should be easy to install. Although the School of Computer Science provides all the resources students require to complete the project, some students prefer to work from their home.

3. Architecture

Minueto is designed to be a modular framework. Its core components include a 2D graphics engine and a keyboard/mouse listener. This modular infrastructure allows a programmer to provide additional services by creating expansion modules, such has sound support or networking.

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.

The MinuetoWindow object is the canvas for the game. Images representing the game are drawn on the MinuetoWindow, which is displayed to the user. At the operating system layer, a MinuetoWindow is a hardware accelerated drawing surface.

MinuetoImages are the basic blocks of a Minueto application. The different types of images available are sub-classes of the MinuetoImage class. 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 layers 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 MinuetoImage.

3.2 Keyboard / Mouse Listener

In a traditional game, keyboard/mouse input is handled with a massive switch statement. Keyboard input is typically stored in a fixed length buffer. A switch statement must test the first character in the buffer to determine which button was pressed. Given that one case statement is required for each key, these switch statements typically contain hundreds of case blocks that, although can be implemented very effciently, are often very diffcult to maintain.

In a traditional GUI application, keyboard/mouse input is handled by an event-based system. When a keyboard/mouse event is detected, a new thread is created to handle that event and execute the associated code. A programmer defines the behavior of a GUI component by implementing a particular interface and registering it with the component. Applications with a high rate of input, such as games, often suffer performance degradation, because the high number of short lived threads created to handle the input.

Minueto uses the MinuetoEventQueue object to record input from the keyboard and the mouse. The events stored in the queue can then be processed using the handle method found in the queue. In the background, invisible to the user, events are added in real time to the queue by a thread monitoring the inputs devices. On the user side, events can be processed by the game thread one at a time by calling the handle method.

Minueto uses handler interfaces to process events stored in the queue. To deal with a specificc type of input, a programmer must implement a predefined interface and register it with the MinuetoEventQueue. When an event is processed, the appropriate method in the registered handler is called. For example, if a keyboard event, such as a key press is processed, the handle method will execute the keyPress method in the keyboard handler.

4. Implementation Concerns

The first year of undergraduate studies at McGill in Computer Science heavily focuses on object-oriented programming in Java. However, we had reservations about game development in Java, given its interpreted nature and the slow performance of Swing. Before committing Minueto to Java, we needed to test the performance of the Java 2D engine to determine if it was suitable for our needs.

4.1 Benchmarks Descriptions

The first goal of the benchmarks were to determine if the Java 2D engine had all the necessary features to implement Minueto. The second goal was to evaluate its performance and ascertain if a sufficient frame rate could be achieved.

The first round of tests used two small benchmark applications. The first benchmark, BlackWizardGrass, features a small character sprite that can be moved over a field of grass (see figure 6.1). The field of grass is a tile map

composed of 32x32 pixels grass tiles. The field is redrawn, tile-by-tile, at every frame. The sprite character is composed of eight 32x32 pixels tiles, each of them representing a different orientation or step in the walk cycle of the character. Only 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 very similar to the first one. 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 demo, each of the characters has eight animation tiles, each one depicting a different step or orientation in the walk cycle of the character.

4.2 Benchmark Results

To gather the required data, 15 computers were chosen, all equipped with a variety of processor, operating system and video hardware. Both demos were run on each machine, both in fullscreen and in windowed mode. Thus, please note that fullscreen mode is not available under Linux, so fullscreen performance data is unavailable for that operating system.

When first started, all the demos showed large instabilities in frame rate. However, the frame rate was only recorded after a few minutes, once it had stabilized.

Several conclusions can be drawn from the collected results :

1) Frame rate for fullscreen results 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.

2) 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. A Geforce 4 MX video card, which is a fairly inexpensive, is required to effectively use Minueto.

3) The Atlhon64 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.

4) There seems to be a linear relation between frame rate and processor speed for the MacOs X computers. This would suggest that the poor performance obtained by the 800 Mhz MacOs X computer is caused, in part, by the processor.

5) 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.

6) 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 Java2D's speed.

5. Documentation Strategy

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 [HK73]. 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 [W.94]. This theory can easily be extended to programmers, where the difference between a novice and an expert is years of problem solving experience [GS02]. An experienced programmer can draw upon years of previous programming challenges to find similarities between previous and current problems.

The main purpose of Minueto is to provide a tool that is easy to learn and to use. Students with little or no game programming background do not have the required experiences to deal easily with problems commonly associated with game programming. Minueto's documentation must address that fact and allow the programmer to learn about various problematic situations and their solutions.

Minueto's documentation is available in three different formats: tutorials, examples and specifications. These different documentation formats are design to match the different learning habits of students.

On the Minueto website, students can find a collection of Howto documents. These short documents are tutorials that cover different specific aspects of Minueto. By reading these tutorials, students can gain insight on how to achieve basic tasks with Minueto and how to solve common problems. This type of documentation targets students who prefer learning by reading books and manuals.

Some students prefer to learn by example. Minueto is distributed with over 20 examples that cover a wide range of topics. Each example is designed to solve exactly one problem. This allows the code for the examples to be as short as possible, thus making it easier for students to understand. By playing around with the examples, the student can gain hands-on experience with the Minueto framework and improve on future problem solving with Minueto.

The third and last type of documentation is the specification. Early in their Bachelor program, McGill students are encouraged to learn how to use the Java API documentation. This documentation is an essential tool for both novice and experienced programmers. Given that Minueto is also a framework, an API documentation is an essential tool. Minueto's API documentation is built using the standard JavaDoc tool and follows the documentation guidelines for Java applications as outlined by Sun Microsystem [Mic00].

6. Case Study

The Java implementation of Minueto was completed at the end of December 2004, just in time for the Winter 2005

term. Minueto was presented to the students during the third lecture, the two first lectures of the term being reserved for the course introduction and the game description.

Although the students were free to use the technology of their choice, they were warned that technical support would only be provided for Java. This was a practical decision, since actively supporting multiple game development platforms would have been a heavy burden on both the professor and his assistant.

6.1 Student Progression

Students were required to attend weekly meetings with the class professor so their progress could be monitored. Three months into the development of their projects, students were called to a special meeting to present a "demo" version of their project. During this "demo" meeting, students presented a working version of their game with a minimal set of features. Students then meet the professor a month later to present the final version of their game.

Although the Minueto framework was introduced during the third lecture, students started using the tool after the first lecture. Only a few days later, a small Minueto demo application had already been posted on the class bulletin board by a student. After the class presentation of Minueto, the number of groups using Minueto doubled. Students also started sending comments and questions. This was concrete proof that students were using the framework.

An important change in the weekly meetings was also noticed. In the previous year, students often complained about the difficulty of developing their custom GUI. Problems using Java/Swing were a big concern for the students. This is understandable, since Swing was never meant to be used for game development. The number of technical complaints and problems dramatically decreased for the class of Winter 2005. Although some Minueto concerns did come up during the meetings, most were diagnosed as incorrect or inefficient uses of Minueto and quickly solved.

An increase in the quality of the projects was also noticed during the "demo" presentations; the games looked more professional and polished. More importantly, all the student's projects met the minimum requirements for the demonstration. Last's year demonstration had no minimum requirements and some teams did not even manage to present a working project. Hence, the 2005 students were able to develop a better working product given the same amount of time. Given that the 2004 students devoted much time to developing and debugging their GUI, we can deduce that students using Minueto were able to concentrate their efforts on other tasks.

6.2 Evolution

Although Minueto does offer all the tools required to complete the project, several of the student groups chose to improve Minueto. Since Minueto's source is commented and distributed with the SDK, students can easily modify Minueto's core components to better answer their needs.

Some of the improvements include:

1) Alpha Channel in MinuetoImage, allowing images to have varying degrees of transparencies.
2) Swing UI component integration, to avoid having to code textboxes and buttons.
3) Active message handling, instead of the passive model normally offered by Minueto.
4) Improved keyboard handler, so implementing textboxes would be easier.

6.3 Final Project

The final project presentations were a learning experience for the Minueto development team. 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. Furthermore, although some games did not meet the final minimum requirements for this project, all the game projects did have a functional user interface.

Although the student of Winter 2005 were free to use the tool of their choice to make their game, a high percentage of the groups chose to use Minueto. Very early into the projects, students sent questions and comments by email to the Minueto development team. Although the development team could not accommodate every feature requested by the students, the team was very helpful in explaining how students could modify Minueto so it would better fit their needs. In addition, several students sent emails to Minueto's developers, thanking them for the large amounts and the quality of the documentation that was available.

7. Discussion

Minueto was successfully tested by the students of the Winter 2005 Systems Development Project class. In addition, it was used by Marc Lanctot to develop a data gathering tool/game for his research [Lan05]. Furthermore, at least two other games have so far been implemented using Minueto.

7.1 Improvements to the Course

Although the students of the 2005 class were all successful in implementing Strategic Conquest, the final testing of the game revealed that focus should be added on two software engineering topics: user interface design and usability testing.

After playing each team's implementation of the game for twenty minutes, it was obvious that several teams had not properly tested or played their games. Although the game themselves were not flawed, their user interface made the game-play slow and unintuitive. Some teams had complex menu systems where several clicks were required to accomplish basic tasks. Other teams required players to know a large amount of keyboard shortcuts, without any built-in reminder/help system in the game.

Many of these problems could have been avoided if user interface design and usability testing had been taught in class.

7.2 Future Additions to Minueto

The student improvements described in section 6.2 are good indications of the need additions to Minueto.

The most important new feature would be the addition of effcient and reusable GUI components (buttons, text boxes, etc.). This could be achieved by either integrating Swing components or building the components with Minueto objects. The latter solution would be preferable because it would be implementation independent. This feature would greatly improve student productivity since students spend a lot of time coding these components themselves.

A non-essential, but useful improvement, would be to automate the regression testing infrastructure. Manually running all the regression tests requires several hours, especially when running them on multiple virtual machines and on multiple platforms.

7.3 More information

More information on Minueto can be found at http://minueto.cs.mcgill.ca/

8. Reference

[GS02] Garner and Stuart. Reducing the cognitive load on novice programmers. Association for the Advancement of Computing in Education (AACE), page 7, June 2002.

[HK73] Simon H. and Gilmartin K. A simulation of memory for chess positions. Cognitive Psychology, 1973.

[Lan05] Marc Lanctot. Adaptive virtual environments in modern multi-player computer games. Master's thesis, McGill University, 2005.

[Ruc03] Rudy Rucker. Software Engineering and Computer Games. Addison Wesley, 2003.

[SW04] Scott Schaefer and Joe Warren. Teaching computer game design and construction. Computer-Aided Design, 36(14), December 2004.

[W.94] Caroll W. Using worked examples as an instructional support in the algebra classroom. Journal of Education, 1994.