

Teaching Computer Programming and Applied Elementary Mathematics & Physics Using Video Games as Simulation

Jam Jenkins

cjj1@cs.duke.edu

<http://www.cs.duke.edu/~cjj1>

Department of Computer Science

Duke University

Keywords: simulation, discrete mathematics, physics, geometry, education, computer programming, video game

Abstract: Video games can be viewed as simulations with the purpose of entertainment. These simulations rely heavily upon linear algebra, discrete mathematics, geometry and elementary physics, and because these simulations (i.e. video games) are entertaining and fun, they provide an excellent teaching opportunity – namely, to teach introductory computer programming with applications in mathematics and physics. Student interest can be captured with the subject of video games, while their learning is focused on computer programming, mathematics, and physics concepts. The introduction to the wide range of physics and mathematical influences in these small simulations can spur interest in future explorations in physics, mathematics and simulations while also preparing students for future work and research by teaching them computer programming. This paper will detail the development and experience of using this approach to teach introduction to computer programming for non-majors at Duke University.

1. Introduction

Courses in software engineering need applications to study. What properties of applications make them particularly appealing for study in introductory software engineering courses? The application should capture the interest of the student and be simple enough to understand yet complex enough to provide substance. The application should also include general concepts that are relevant to a broad range of other applications. Ideally, the applications should also capitalize on and connect to student knowledge in related fields while motivating future explorations in these and other areas.

A second question that often arises is to what degree and in what manner should libraries and packages be used to teach software engineering courses. One approach is to teach at a high level in order to focus on high-level concepts and logic. Another more constructivist approach is to teach starting at the most basic level and build up to the high level. Both approaches have their strengths and weaknesses and, as with most educational approaches, the best approach is usually a combination of the two.

This paper discusses the rationale and implications of using video games as the application. The design necessary to balancing the use of packages versus the understanding of those packages is also explored. The primary questions motivating this discussions are:

- How can video games be used as an effective application to teach computer programming and related fields?
- How should packages be used to teach video games in an introductory programming course?

The next section describes the scope, contribution, and evaluation of this research. Section 3 describes and gives examples of the various related topics which can be taught as part of an introductory programming course using video games as simulations. Section 4 describes the design of the video game package used in the course to teach introductory programming to non-majors at Duke University. Section 5 describes experiences in using this package in Spring 2004. Section 6 concludes and describes future work related to the package and its use.

2. Contribution

Using video games to teach computer programming is not a new idea. Simulations, again, have been used before to teach computer programming, but less common is the combination of using both video games and simulations using the same structure. The strong similarity between these two topics enables selection of an application that is both motivating and academically rigorous enough to cover topics contained in and related to introductory programming.

The debate over what level programming language to use in introductory computer programming courses is not new. In fact, there was much debate about changing the curriculum of the Computer Science AP (Advanced Placement) exam from C++ to Java. In a large part, this debate centered around whether students benefit more from first learning the lower level constructs in C++ and only later progressing to higher level constructs such as are provided in Java or vice versa.

The realization that the two approaches are not necessarily mutually exclusive is complex. Is it possible to both use packages and attempt to understand the inner workings of the package within the same course? This approach is indeed very difficult to achieve when the applications are video games, because the package must be functional and fast, but also simple and designed so that it can be used as a learning tool.

The contribution of this research is in the combination of the two approaches. Video games as simulation provide the application to be studied. The study of this application is combined with using a video game package whose purpose is to provide functionality in the beginning of the course, and provide an example package design and implementation by the end of the course.

No attempt has been made to quantify the effectiveness of this approach because measures of meaning would be too difficult, too costly, and too error prone to have significant value. Instead, a better measure of the effectiveness is the individual evaluation of student work in the course. For this reason, students taking the course document their work in course web pages. This student work and other materials used in the course can be examined by contacting the author

or visiting the course web page <http://www.cs.duke.edu/education/courses/spring04/cps004>.

3. Applications

The key to using video games to study software engineering is to connect the lessons learned while developing video games to the many diverse fields which use similar structures as those used in video games. These fields include, but are not limited to, simulations, physics, mathematics, probability, statistics, and of course, software engineering. The following subsections will describe and give examples of how video games can be applied to these areas.

Simulations

Video games are interactive simulations for the primary purpose of entertainment, but video games can also have secondary purposes. For example, flight simulators are often meant to be entertaining, but they can also be used as a training tool for future pilots. Several games fit into this category of simulation; for example, all sports games are models of physical phenomenon.

Golf is a simple example. The player in the game is given information (visually and textually) about the conditions, which could include distance to the green, wind conditions, and potential hazards. Choices in the game could include choice of iron or wood, pitch of the club, speed of the swing and position of the club. When the player swings the club at the golf ball, the game must simulate the physical phenomenon, usually in real-time with a visual display.

In just this part of the game, there are ample areas of simulations which can be studied. Questions to be answered include:

- Should discrete time or continuous time be simulated?
- What level of detail should the physical phenomenon be modeled?
- What tradeoffs exist when making these decisions?

With a visual display and an interactive simulation, students will be able to explore these topics in great detail to gain a deeper understanding of the choices that must be made in developing a simulation.

Physics

Integral to most modeling of physical phenomenon is some application of basic physics concepts. In the above example of a golf game, the interactive simulation could apply to aerodynamics, angular momentum, projectile motion, and countless other topics typically taught in introductory physics. While golf may be a little more involved because of its three-dimensional nature, other games rely only on simpler two-dimensional physics, such as the very simple game Breakout shown below in Figure 1.

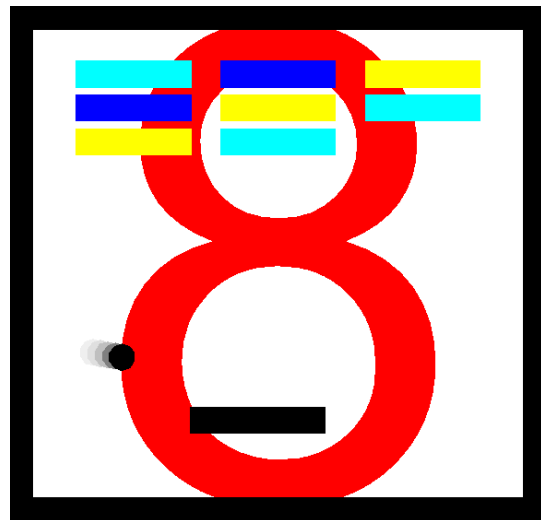


Figure 1

Breakout – a simple game in which the player tries to break the tiles on the top of the screen by bouncing the ball off the walls and the paddle at the bottom of the screen.

Even the simple game of Breakout can involve some non-trivial physics. The most interesting physics question in pong is what happens when the ball deflects off a surface?

There are many varying levels at which this deflection can be simplified. The easiest simplification is to restrict the ball to particle motion and require all surfaces to be vertical or horizontal only. This restricts the deflections to changes in the sign of the horizontal and vertical velocity vectors. Less of a simplification is to allow surfaces to be continuous and flat, but at different angles than horizontal or vertical. Removing other simplifying assumptions can make the physical modeling more realistic and more difficult. For example, the ball need not be a particle and the surfaces need not have continuous slopes. For even more complexity

allow for angular velocity of the ball and friction with the surfaces.

Pinball is another such game that has many opportunities for modeling Newtonian motion. One great benefit of using video games to explore these topics is that the correctness of the physical modeling will become immediately apparent due to the visual nature of the display.

Mathematics

One primary but often overlooked question is how does the video game know when two objects intersect? An example is determining when the ball and the paddle intersect in games such as Breakout. The opportunities for exploring geometry and calculus here are many. Again the depth of study is limited by the number of assumptions and the degree to which they simplify. Is an approximate time and location of intersection sufficient? Are the motions of the objects linear? Are the objects deformable? Are the objects convex? The answers to these questions determine the level and type of mathematics involved in developing the correct algorithms.

Particularly for video games and other optimized simulations, it is usually important to develop hybrid approaches to collision detection that reduce the possible objects to be tested for intersection. At a first level, objects are provided with rectangular boxes surrounding them entirely. It is necessary for two objects' bounding boxes to intersect for the objects to intersect. At this point, other methods may be used, such as using a set of bounding boxes to better model the shape of the original object. These and other algorithms and data structures may be implemented and tested through very simple games such as Breakout.

One very important aspect of many simulations and video games is a degree of uncertainty. For games, uncertainty greatly increases the complexity of implementing various strategies. (Simulations of physical phenomenon are also written for a similar purpose – because the random behavior of systems can make them very difficult to analyze theoretically, but studying them empirically through simulation is much easier). Card and dice games are among the most straightforward uses of random distributions, but probability can also be used to design artificially intelligent agents in video games.

Computer Programming

The primary goal of using video games as simulations is to teach computer programming. Video games are of an adequate complexity to explore many topics, but not so overly complex that they can only be studied with the assistance of high-level packages. The next section describes in detail how the design addresses the complexity involved in creating a video game package suitable for teaching introductory programming.

4. Design

Motivation

One of two often-divergent approaches to teaching programming is to teach at a high level using many packages and libraries. The other is to teach starting from the basics so that students will know how the libraries are made and therefore be better equipped to use them properly. Like most educational approaches, the optimal choice is to use a variety and balance of approaches. For this reason, it is not sufficient to teach students about video games as simulations by only using simulations software. This approach would limit student understanding of limitations of simulations software. Also, students would be overburdened by requiring them to write video game simulators entirely without first having some experience using them.

For exactly these reasons, the video gaming package was developed – to teach students about simulations and programming packages by first using the package, and slowly coming to understand exactly how the package is designed and implemented, all within one course. The next section describes how this is achieved.

Goals

The video game package is designed with the following goals in mind:

- The package must be simple to use.
- The package must be reasonably fast.
- The design must be clean enough to study and modify in introductory computer science courses.
- The base package must contain a small amount of code.

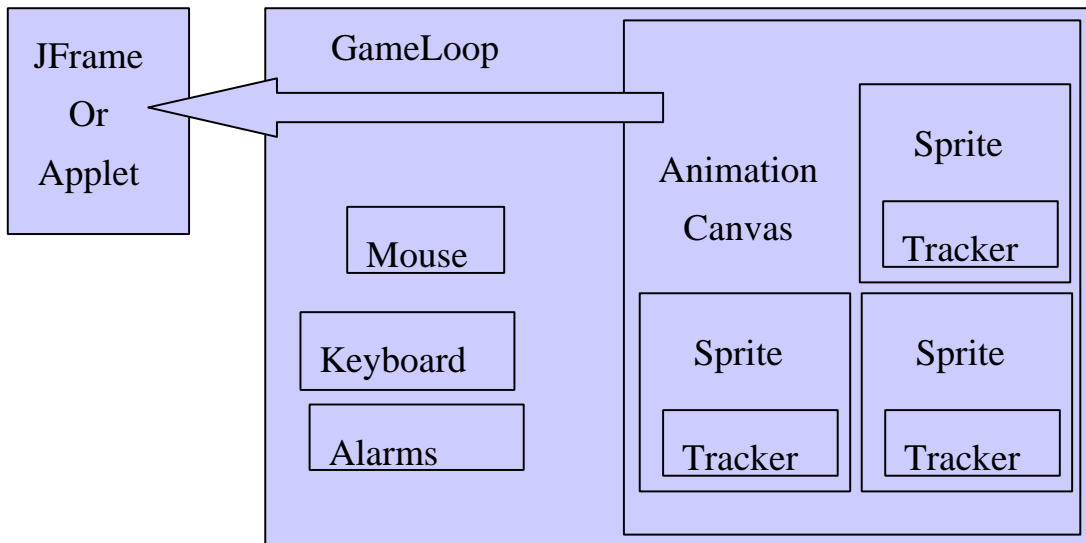


Figure 2 Design of the video game package.

For these reasons, simplicity takes precedence over creating a high performance gaming engine. This is because large pieces of complex code prevent beginning programmers from understanding how to use and modify them. The package is not made for scripting games because the purpose is for users of the package to learn programming, not just game design. Most games made using the package are relatively simple due to the programming ability of beginning programmers; therefore complex options for optimization are not included.

Details

Figure 2 above graphically depicts the basic design of the video game package. The most basic component of the package is a `Sprite`, which is a graphical component with size, shape, orientation and location. Optionally attached to `Sprites` are `Trackers` which can alter the size, orientation, and location with respect to time.

The `Sprites` are added to an `AnimationCanvas`, which is responsible for iterating over the `Sprites` in order to display them. The `GameLoop` contains an `AnimationCanvas`, `Keyboard` and `Mouse` for user input, and `Alarms` for scheduling time events. The game is eventually displayed by adding the `AnimationCanvas` from the `GameLoop` to a container, typically a `JApplet` or `JFrame`.

There is also one other class which facilitate the animation and manipulation of `Sprites` – the

`FrameAdvancer`. This class updates the `AnimationCanvas` at regular time intervals. This can be done in two ways – in a single thread (the AWT Event Loop Thread), or in a separate thread. The default operation is in a separate thread. (The tradeoff between one and two threads is between maximum performance and maximum responsiveness.) The `FrameAdvancer` is also responsible for keeping time. When the computer is able to process all events in the given time, game time will correspond to actual time; otherwise, game time will progress at a slower rate and the `Trackers` and `Alarms` will respond accordingly.

In addition to these classes in the basic package, there are also classes provided which are commonly used but not part of the the framework. One of these is the `NormalGenerator` which can generate normals for polygons and portions of polygons. This is necessary for many applications in responding to collision detections (for example, determining the deflection angle, velocity and rotation upon collision).

The `TimedSpriteEnabler` and `TimedSpriteKiller` classes use the `Alarm` class to enable `Sprites` after a given period of time and to remove `Sprites` from the `AnimationCanvas` after a given period of time respectively. `PathBlur` is used in conjunction with the `BlurSprite` to provide motion blur for fast moving objects.

Since these classes are derived from the basic

package, they both provide an example of how to use the package and also enable more rapid game development.

All classes are developed and coded in such a manner that beginning programmers should be able to understand each class in its entirety. Every class is provided with commented source code, documentation, and only uses packages in the standard Java 1.4.2 API. The purpose of this approach is that students using of the package will learn how to program using the standard Java API and not be restricted to programming exclusively using the video game package. Through the simplicity of the design and the code, students will learn how packages are designed and how to program in Java using the standard API. This approach is intended to satisfy both those who vie for teaching computer programming using custom high-level libraries and those who prefer to teach using only standard libraries.

5. Experiences Using the Video Game Package

This video game package was used to teach the CPS004 Java for Video Games: Introduction to Programming course at Duke University in the Spring of 2004. The intended goal of the course is for students to leave the course with a basic understanding of programming. Java and video games are just the mechanism used for teaching while the concepts are to be focused squarely upon computer programming.

Early in the semester students were given the assignment to alter the game Pong to include gravity and also to provide cosmetic changes to the paddles and ball. This assignment was essentially geared toward gaining a working knowledge of the gaming package. In the next assignment students were required to use the Mouse class to direct the motion of a paddle in Breakout, a game similar to Pong. They were also instructed to keep score and provide additional levels. For these assignments, source code was provided for students to modify.

The next assignment was geared toward understanding the mechanisms behind the video game package. The package uses two-dimensional graphics because they can be modeled more simply than three-dimensional graphics. In order for the students to understand some of the concepts behind two-dimensional graphics and simulations, they were to construct a simplified one-dimensional simulation of point and segment motion. The purpose of this

simulator was to detect time and location of collisions along a line. The concept of the simulator is very simple, but the explorations involved very much resembled the types of explorations that are necessary in developing a two-dimensional simulator. Using this analogy, students were better able to appreciate and understand the complexities in the gaming package itself.

Other assignments in the semester included developing the game tic-tac-toe using only the standard Java API, and making splash screens using the gaming package to explore the use of different Trackers and Sprites. Not all assignments in the course used the video game package – in fact some of the early assignments involved students writing programs entirely from the start. The video game package is useful for students in the class to write programs, but being able to write programs without the package was also deemed an important goal of the course. The final project involved students working in groups to apply the computer programming skills learned during the course of the semester. These final projects, course assignments, and student work are linked from the course web page <http://www.cs.duke.edu/education/courses/spring04/cps004/>

6. Conclusion & Future Work

After years of work invested in developing the video game package and an approach to using it to teach introductory computer programming, a simple yet functional package has been developed. This package has been successfully used over the summer of 2003 at the Duke Talent Identification Program and in the Spring 2004 Duke University course Java for Video Games. The most recent version of the package will be used in Summer 2004 to teach the Java for Video games for the Duke Talent Identification Program. The package and approach are also being examined to decide if they will be integrated into the curriculum intended for computer science majors at Duke University.

While a select group of college students during the academic year and high school students during the summer have benefited from this research, more opportunities to use the package and the approach are being explored. Of particular interest is the use of this method in high schools for covering Computer Science Advanced Placement Exam topics. Also because the package is small and relatively simple, parts of the package can be further developed in graphics and gaming courses having more advanced programmers.

7. Acknowledgements

I would like to thank Dr. Dietolf Ramm for his assistance and guidance in teaching Java for Video Games at Duke University. I would also like to thank my teaching assistants: Daag Alemayehu for his assistance in the Duke Talent Identification Program Computer Science courses and Peter Guarnieri and Garver Moore for their assistance in Computer Science 004 at Duke University.