Ucigame, A Java Library for Games

Daniel Frost
Donald Bren School of Information and Computer Sciences
University of California, Irvine
Irvine, CA 92697-3440 U. S. A.
frost@uci.edu

ABSTRACT
Ucigame (pronounced *OO-see-GAH-me*) is a Java package that supports the programming of 2D sprite-based computer games. Designed for novice programmers, it enables students in an introductory class to write computer games that have animated sprites, music and sound effects, and event-driven keyboard and mouse handling. Ucigame has also been used successfully in a senior-level class for experienced programmers.

Categories and Subject Descriptors
D.2.13 [Reusable Software]: Reusable Libraries.

General Terms
Design, Human Factors, Languages.

Keywords
Computer games, Java, Java library, Ucigame.

1. INTRODUCTION
> "Our practice of embedding a programming language in the first courses, started when languages were easy for beginners, has created a monster. Our students are being overwhelmed by the complexities of languages that many experts find challenging (typically Java and C++)… Many do not experience the joy of computing: the interplay between the great principles, the ways of algorithmic thinking, and the solutions of interesting problems." – Peter J. Denning [2].

The Ucigame (pronounced *OO-see-GAH-me*) framework is designed to enable novice programmers to create 2D computer games in Java. It has also been used by relatively experienced programmers in a senior-level game development course. Ucigame is a Java package, distributed as ucigame.jar, that handles many of the more complex game programming tasks, including loading and playing sound files, handling concurrency and multi-threading issues, and sprite-related tasks such as loading images from files, animating the images, and detecting collisions. A challenge in teaching an introductory Java programming course is to find short projects that excite the students, are not too difficult, and do not rely on Java libraries that have not yet been covered. Simple game projects with Ucigame may fulfill this need.

SIGCSE '08, March 12-15, 2008, Portland, Oregon, USA.

2. MOTIVATION
Ucigame was developed to make introductory Java programming more engaging. For many high school and college students, computer games are an important component of their culture, and also the most visible type of computer program. (The variety, frequent updates, and even the bugginess of computer games make them more visible as computer programs than the more widely used software in operating systems, office productivity applications, and electronic devices.) Familiarity, interactivity, and certainly the "fun factor" all make computer games effective programming projects for novice programmers.

Programming courses often teach, at least initially, a subset of the language and its libraries. A new programmer is learning new skills and a new way of thinking, and teachers often want to minimize complexity at the language and library levels. For example, in Java the *case* statement may be put off, in favor of solely using *if*. Sometimes teachers use a development environment that only accepts a subset of the language (e.g. DrJava [6]) or start with a different and simpler language altogether [12].

A related technique is to use simplified or modified versions of standard libraries, with the goal of substituting ease of learning and ease of use for flexibility and generality. A classic example in Java is to replace the console input facilities provided by *System.in* and the *InputStream* class. (The Java Standard Library's *Scanner* class, introduced in Java 1.5, alleviates this need somewhat.)

We have articulated two goals: to engage students with computer game projects, and to ease students' paths into Java programming with a digestible subset of the Java language and its libraries. These goals may be contradictory, since even a simple computer game can require a fairly complex implementation. We attempt to resolve the conflict by restricting the computer games to 2D, sprite-based games and by encapsulating in the Ucigame library many of the mundane and yet complex chores required for even a simple game. Ucigame contains a number of classes and objects which provide easily comprehended and used abstractions of common game components.

3. THE UCIGAME PACKAGE

Computer games written with Ucigame are based on sprites, small graphics that can move around in the game window. A sprite that doesn't move might be a tree, building, or some other part of the background. A moving sprite might be a character or a spaceship. We have found that students enjoy designing sprite-based games and that they provide plenty of room for students' creativity.

3.1 Pong.java

The best way to see how Ucigame works is to look at a relatively simple example program. The source code for Pong.java follows. A screen capture is in Figure 1. Although based on the traditional Pong video game, the Ucigame version is simpler: a single player controls a single paddle, and there are no goals and no scoring. Nevertheless, Pong.java illustrates the elements of all Ucigame-based games.

```
// Pong.java
import ucigame.*;

public class Pong extends Ucigame {
  Sprite ball;
  Sprite paddle;

  public void setup() {
    window.size(400, 250);
    window.title("Pong");
    framerate(30);

    Image bkg = getImage("images/pic.png");
    canvas.background(bkg);

    ball = makeSprite(getImage(
         "images/ball.gif", 255, 255, 255));
    paddle = makeSprite(getImage(
         "images/pad.png"));

    ball.position(
       canvas.width()/2 - ball.width()/2,
       canvas.height()/2 - ball.height()/2);
    ball.motion(6, 3);
    paddle.position(
      canvas.width() - paddle.width() - 10,
      canvas.height() - paddle.height())/2);
  }

  public void draw() {
    canvas.clear();  // draw background
    ball.move();

    ball.bounceIfCollidesWith(paddle);
    ball.bounceIfCollidesWith(TOPEDGE,
      BOTTOMEDGE, LEFTEDGE, RIGHTEDGE);
    paddle.stopIfCollidesWith(TOPEDGE,
      BOTTOMEDGE, LEFTEDGE, RIGHTEDGE);

    paddle.draw();
    ball.draw();
  }

  public void onKeyPress() {
    if (keyboard.isDown(
          keyboard.UP, keyboard.W))
      paddle.nextY(paddle.y() - 2);
    if (keyboard.isDown(
          keyboard.DOWN, keyboard.S))
      paddle.nextY(paddle.y() + 2);
    if (keyboard.isDown(
          keyboard.LEFT, keyboard.A))
      paddle.nextX(paddle.x() - 2);
    if (keyboard.isDown(
          keyboard.RIGHT, keyboard.D))
      paddle.nextX(paddle.x() + 2);
  }
}
```

Every Ucigame-based game imports ucigame.* and has a class that extends Ucigame. The Ucigame superclass contains a public static void main() method which is invoked when Pong is run from the command line, and init(), start(), stop(), and destroy() methods which override those in Applet, which is Ucigame's superclass. The Ucigame superclass also contains the methods setup(), draw(), and onKeyPress() which are meant to be overridden by game programs. The setup() method is called once at the start of the game, and is a good place to load images and sounds and to initialize sprites. The draw() method is called every time the game's window needs to be refreshed—either due to normal window events such as a covered area being exposed, or due to the game's animation. The onKeyPress() method is called when the player presses a key on the keyboard.

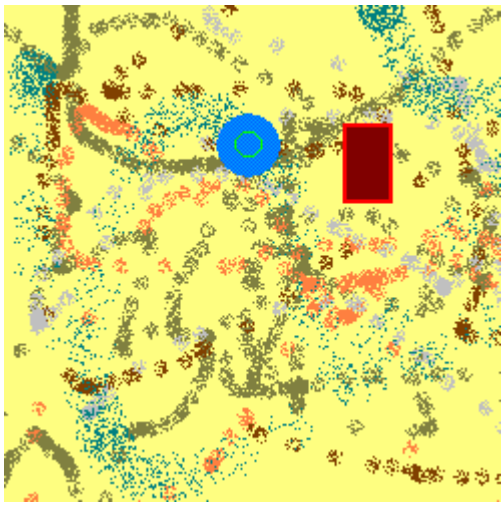

Figure 1. The Pong game, with a bouncing ball, a player controlled paddle, and programmer art background.

Ucigame provides several useful built-in objects; three used in Pong.java are window, canvas, and keyboard. window represents the operating system window on the screen in which the game appears, canvas refers to the drawable area within that window, and keyboard, of course, represents the physical keyboard. A complete list of objects appears in Table 1.

Now let's look inside each of Pong.java's methods. The first line in the setup() method sets the game window's width and height (the inner dimensions; when the borders and title area are included the window will be slightly larger). This method has no effect when the program is running as an applet. In the next line window.title() sets the text in the caption area. (If the program is running as an applet the text displays in the browser's status area.)

The parameter passed to the framerate() method specifies the number of times per second that draw() will be called. The next two lines in setup() create a ucigame.Image object based on a file's data, and specify that the Image object should be the canvas's background—that is, the image will be painted on the canvas when canvas.clear() is called. The remaining lines in setup() create and position the ball and paddle sprites. Note that ball is initialized using a four-parameter version of getImage(). The last three parameters specify that any pixels in the images/ball.gif file with RGB color (255, 255, 255), that is, white, will be transparent. (Image files always contain rectangular images, but when some pixels are transparent the image can appear to have a non-rectangular shape.) The motion, if any, of a sprite can be completely controlled by the programmer, or can be handled automatically by Ucigame. Specifying ball.motion(6,3) makes the ball sprite move 6 pixels to the right and 3 pixels down every time ball.move() is called (in draw()).

Table 1. Objects and classes provided by Ucigame.

window object	Represents the entire game window.
canvas object	Represents the rectangular interior of the window..
keyboard object	Represents the computer keyboard.
mouse object	Represents the computer mouse.
Sprite class	A Sprite object is a movable, possibly animated image.
Image class	An Image object is created by copying the pixels from an image file.
Sound class	A Sound object is a sound or a piece of music.
Ucigame class	The superclass that initiates and controls the game.

Pong.java's draw() method is invoked 30 times per second, providing smooth animation. As mentioned above, canvas.clear() overwrites the canvas area with the specified background, and ball.move() repositions the ball sprite by its movement amount. The next three lines use the Sprite class's bounceIfCollidesWith() and stopIfCollidesWith() methods. These methods can take any number of Sprite objects as parameters. If there is a collision or overlap between the "this" Sprite and any listed as parameters, then the "this" Sprite object's movement amounts are modified appropriately—either to cause a bounce or to stop the sprite completely. TOPEDGE, BOTTOMEDGE, LEFTEDGE, and RIGHTEDGE are final ucigame.Sprite objects defined in the superclass and representing the edges of the canvas area. Finally, the Sprite objects' draw() methods are called, which display the images on the canvas at the correct locations.

The onKeyPress() method is optional, but if it is coded it is called whenever the player presses down or holds down one or more keyboard keys. The programmer can turn off the autorepeat feature, so that holding down a key only results in a single call to onKeyPress(). onKeyPress() is executed immediately before draw(), which means that the autorepeat rate is determined by the window's frame rate. The keyboard.isDown() method is passed any number of (integer) constants representing specific keys, and returns true if any of those keys is down. nextX() and nextY() from the Sprite class specify where the sprite will be positioned on the next call to draw(), subject to modification by the stopIfCollidesWith() method. In Pong.java, the arrow and WASD keys move the paddle by two pixels per frame.

For a new programmer, Pong.java is a fairly long program and certainly cannot be comprehended without time and effort. However, two factors mitigate the difficulty and make this program pedagogically effective. First, the only control structure used in Pong.java is a simple if; otherwise there are no conditionals, loops, or blocks. Of course, a lot of implicit control is occurring: the setup(), draw(), and onKeyPress() methods are invoked at the right times, and the bounce/stopIfCollidesWith() methods have implicit conditionals. Secondly, most of the code consists of calls to methods in objects. Since the objects in the program all correspond to visible aspects of the computer game, and their methods are all easily understood functions of those objects, students have little trouble tinkering with the code to extend or modify it. In our freshman class students are introduced to Pong.java in the second lab, and they add a second paddle to the game with its own keys to control it. By the end of the ten week quarter, many students are writing moderately complex games with multiple moving objects, interesting gameplay, original art and music, and often accommodating two players. The students' games are usually modeled closely on classic arcade games of the 1980s, but for the students the implementations are satisfying accomplishments.

3.2 Other Ucigame Features

Several capabilities of Ucigame not illustrated by Pong.java will be briefly described.

An **animated sprite** has more than one "frame," each frame being a separate image. Cycling through the frames creates the illusion of motion within the sprite. (Whether the sprite's position on the canvas is changing is an orthogonal issue.) The frame images may be stored in multiple files, but often they are stored within a single large file. In the following snippet, a Sprite object named dancer has already been declared, and its frames come from three segments of a PNG image file, with upper left hand pixels at (0, 0), (50,0), and (100, 0).

```
Image frames =
   getImage("images/frames.png", 0, 0, 0);
dancer = makeSprite(50, 75); // width, ht
dancer.addFrames(frames,0,0, 50,0, 100,0);
```

Each time dancer.draw() is called, Ucigame displays the next frame, cycling from the last back to the first.

Ucigame programs can be run as **applets or applications** without recompilation. The Ucigame superclass includes both a public static void main() method (for applications) and is itself a subclass of Applet, overriding the methods init(), start(), stop(), and destroy().

Computer games written in Java utilize **multithreading** to handle keyboard, mouse, and window events, maintain timers for steady animation and gameplay related occurrences, and perform calculations. The concept of multithreading is within reach of most college level beginning programmers, but coordinating threads with Java's Swing library—for example, with

SwingUtilities.invokeLater()—is not. Ucigame makes most aspects of multithreading and event driven programming trivial, since the superclass manages the threads and calls programmer supplied methods at the correct times. The Ucigame programmer does not need to know about any threading issues.

Sound effects and background music are important components of games. Java's AudioClip class makes playing, stopping, and looping one or more sound files fairly easy, but it only handles uncompressed audio formats such as .au and .wav files. Ucigame includes the open source JLayer library to decode and play MP3 format audio files [10]. To the game programmer all audio files are handled with the same methods. For example:

```
Sound theme =
   getSound("sounds/MyTheme.mp3");
theme.loop();
```

Game programmers can create sprites which act as **buttons,** by supplying three images for the button's resting, mouse-over, and clicked states. When the button is clicked, Ucigame calls a method specified by the programmer.

Text of any size, color, and font can be drawn by Ucigame on sprites or on the background of the window.

The Ucigame library is **compatible with Java 1.5**. As Java has evolved, class files created with one version of Java can usually not be run by earlier versions of Java. This is primarily a concern with applets, where the programmer and web page writer cannot control what version of Java exists on the user's computer. Ucigame makes use of several Java features introduced in version 1.5, most visibly varargs. For instance, the keyboard.isDown() method can be passed any number of (int) parameters.

Several features that could be put into Ucigame have been deliberately left out. One example is a possible camera object. Many games have a world, level, or map that is larger than the game's window. Often the player uses the arrow keys to move the window over the map. For the programmer, it can be challenging to correctly set the coordinates of the background image and the sprites so that they appear in the correct positions relative to each other and to the visible window. A camera object could simplify this task by encapsulating the appropriate offsets that must be added to the sprites' positions. However, the goal of Ucigame is to hide infrastructure and particularly tricky game code (such as collision detection), not to take all game logic programming out of the student's hands.

4. UCIGAME FOR SENIORS

Although the original intention was to use Ucigame in an introductory course, we also used it in a capstone type course for seniors on computer game development. In this course students work in teams of three or four, and teams spend the first half of the term writing a design document. During this period we assigned each student to work independently on a Ucigame-based game. Since the students were already competent Java programmers, a 45 minute introduction to the library was sufficient. The seniors generally liked this assignment, and many of the resulting games were ambitious and extensive.

5. RELATED WORK

The development of Ucigame—most importantly the setup() and draw() method structure—was influenced by the marvelous Processing language [13].

Several fine libraries have been designed to shield new Java programmers from some of the language's murkier features. The Java Task Force's jtf.jar package provides simplified and improved (at least for beginners) input mechanisms, GUI components, and graphical capabilities [9]. The objectdraw library, developed at Williams College, supports events, multiple threads, and a more object-oriented graphics system [7].

A wide variety of tools and kits exists to help with the writing of computer games. All give the game designer at least a taste of the programmer's task of iteratively refining a design into a satisfactory executable program. At the "drag and drop" end of the spectrum are tools such as GameMaker [15], Scratch [12,14], and Alice [1], all of which have the wonderful characteristic of making syntax errors impossible. LWJGL [5], the Java Game Toolkit [8], and the Golden T Game Engine [3] are Java libraries which overlap Ucigame in functionality, but none is as strongly oriented towards beginning programmers.

6. CONCLUSIONS

> [T]he most significant bit in helping people learn programming is the motivation. What are you *doing* with programming? The language can get in the way, but people will go through a huge amount of effort to do something that they *want* to do. – Mark Guzdial [4]

Learning to program is difficult, and many approaches to teaching programming have been proposed. The focus of Ucigame is on motivating students by giving them a tool to write simple computer games as they acquire the fundamentals of Java programming. The Ucigame website, at www.ucigame.org, provides documentation, a link to the .jar file, access to the source code, and a number of sample programs, all of which are freely available.

7. REFERENCES

[1] Alice: Free, Easy, Interactive 3D Graphics for the WWW. http://www.alice.org/. Accessed September, 2007.

[2] Denning, P. J. 2003. Great Principles of Computing. In *Comm. of the ACM*, Nov. 2003, Vol. 46 No. 11.

[3] Golden T Studios – Golden T Game Engine (GTGE). http://www.goldenstudios.or.id/products/GTGE/index.php. Accessed September, 2007.

[4] Guzdial, M. 2007. What is the role of language in learning programming? Blog entry dated Oct. 25, 2007, at http://www.amazon.com/gp/blog/post/PLNK161I4L1UV4A43.

[5] Home of the Lightweight Java Game Library. http://lwjgl.org/. Accessed September, 2007.

[6] Hsia, J. I., Simpson, E., Smith, D., Cartwright, R. Taming Java for the Classroom. *SIGCSE 2005: Proceedings of the Thirty-Sixth SIGCSE Technical Symposium on Computer Science Education*, St. Louis, Missouri, 2005.

[7] Java: An Eventful Approach. http://eventfuljava.cs.williams.edu/. Accessed September, 2007.

[8] Java Game Toolkit. http://www.jpemartin.com/jgt/. Accessed September, 2007.

[9] Java Task Force Home Page. http://jtf.acm.org. Accessed September, 2007.

[10] JLayer MP3 library for the Java Platform. http://www.javazoom.net/javalayer/javalayer.html. Accessed September, 2007.

[11] Leutenegger, S. and Edgington, J. 2007. A Games First Approach to Teaching Introductory Programming. In *SIGCSE 2007: Proceedings of the Thirty-Eighth SIGCSE Technical Symposium on Computer Science Education*, Covington, Kentucky, 2007.

[12] Malan, D. J. and Leitner, H. H. Scratch for Budding Computer Scientists. In *SIGCSE 2007: Proceedings of the Thirty-Eighth SIGCSE Technical Symposium on Computer Science Education*, pages 223-227, Covington, Kentucky, 2007.

[13] Processing 1.0 website. http://www.processing.org/. Accessed September, 2007.

[14] Scratch | Home | imagine, program, share. http://scratch.mit.edu/. Accessed September, 2007.

[15] YoYo Games | Gamemaker. http://www.yoyogames.com/gamemaker/. Accessed September, 2007.