BUILDING INTERACTIVE MATHEMATICS APPLICATIONS USING STENCYL

Dr. Robert L. Watson
Mathematics Department
University of Mount Olive

630 Henderson St.
Mount Olive, NC 28365

rwatson@umo.edu

Abstract

We will demonstrate how to use Stencyl, an emerging cross-platform application development environment, to quickly build and publish math applications to a variety of devices. Discussion will include a quick overview of the environment, and examples of implementation.

Examples will illustrate how to implement communication with a server (for use in a student response system), adaptive learning software, and assessment using the core Stencyl tools and mathematics extensions (Instructional Software Development Kit).

Once written, the application can then be published to Apple tablets and phones, Android-powered tablets and phones, PC, Mac, and Linux, with very little, if any, modifications needed between ports.

Suggestions for incorporation of the example applications in the classroom will be provided.

1 Introduction

Recent advances in classroom instruction have ushered in steeper software requirements. Such requirements may be problematic for smaller colleges, which cannot take advantage of the economy of scale that larger universities can. Simultaneously, institutions of any size can benefit from writing their own custom software, suited to their own particular needs. This is especially true in an environment, such as an experimental classroom, in which rapid updates to the software may be necessary. In this case, owning the code allows for easier experimentation and flexibility.

The usual difficulty in writing custom instructional software is the amount of time which must be invested. To an instructor with little to no programming back-

ground, finding the time to learn a language on top of developing the software would be problematic.

The goal of this paper is to discuss how Stencyl, a 2D game development environment, can be extended to allow for quick development of instructional mathematics software. Stencyl provides an environment that is similar to that of *Scratch*, an authoring environment developed by MIT Media Lab. Both Stencyl and Scratch use "code blocks", interlocking blocks that can be joined to form executable code.

Figure 1: Code blocks.

Stencyl also provides a library of pre-written modules, called "behaviors", that can be attached to a screen or element within the screen, and configured. In this manner, applications can be built using little or no code.

Stencyl seeks to simplify publishing across multiple platforms. Today, there are numerous platforms (PC, Mac (OSX), iOS, Android, Linux, etc.) Many institutions do not insist their students use a single, standard platform. So, any applications must be pushed out to all platforms. Haxe, the programming language Stencyl utilizes internally, addresses this issue. The developer needs to only write their application using the Haxe language. The code can then be ported to multiple platforms from the single Haxe source.

We will discuss the advantage and disadvantages of developing mathematics applications using Stencyl, and what may and may not be easily accomplished. We will introduce the *Instructional Software Development Kit*, which extends Stencyl's functionality. We will also discuss some example applications, and classroom environments they may be adapted to.

2 Overview of Stencyl

To better understand how Stencyl can be used for instructional software development, and what limitations may be encountered, we will provide a brief overview of how the platform is designed.

Stencyl is a 2D game development platform. It offers a point-and-click, drag-and-drop interface. While such an interface is targeted toward novice to intermediate-level programmers, advanced programmers will find it easy to extend its functionality with their own engine extensions. Due to this design, Stencyl is already being used at schools and universities in introductory-level programming courses.

In Stencyl, applications are developed using a variety of tools:

- A *scene editor*, which is used to create the display the user will interact with.
- The *inspector*. Interactive elements (including GUI elements) are referred to as *actors*. The inspector allows for the customization and configuration of actors that have been placed on the screen.

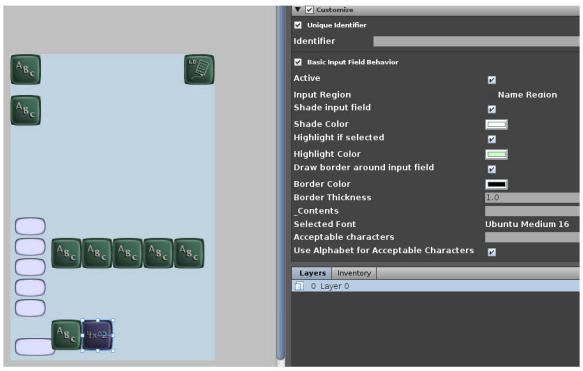

Figure 2: The "Submit Button" actor is drawn on the scene editor, along with other actors. To the right, the inspector allows the selected actor to be configured. These actors provide the elements the user interacts with, including text boxes, labels, captions, and buttons.

- Actors can be configured with any number of *behaviors*–pre-programmed code libraries that add functionality to the actor. Behaviors can be downloaded from the web, and imported into any number of projects.

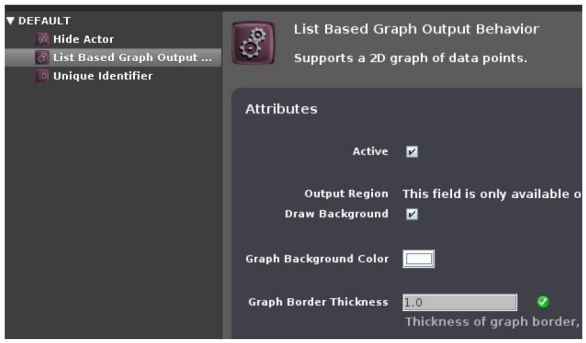

Figure 3: A behavior and configurable attributes

- A *behavior editor*, which is used to author behavior code. Code can be authored using raw Haxe code, or drag-and-drop *code blocks*.

A developer can add their own code blocks, through the use of an *engine extension*. The engine extension is raw Haxe code.

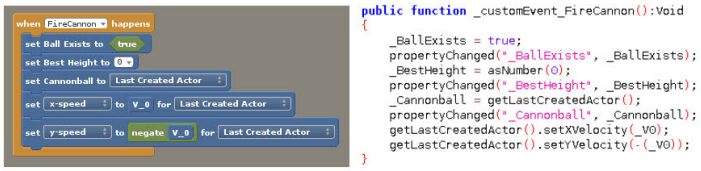

Figure 4: Scratch-like code blocks and equivalent Haxe code

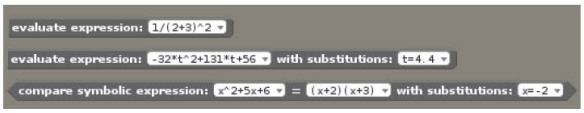

Figure 5: Custom code blocks which use the engine extension capabilities

3 Instructional Software Development Kit

Some tools required for instructional software are not provided out-of-the-box. These include tools ranging from basic GUI elements, such as buttons and check-boxes, to mathematical interpreters. Some of these features can be downloaded from the user community. However, a single library in which every object is designed to work together would be more efficient. The *Instructional Software Development Kit* was designed to address this concern.

The purpose of the Instructional Software Development Kit is to extend Stencyl's capabilities so that effective (mathematics) instructional software can be developed. The Instructional Software Development Kit is divided into two packages. It includes engine extensions for mathematical expression interpretation, and a library of behaviors designed for use in instructional software.

The Instructional Software Development Kit provides a framework in which entire applications can be developed with no coding. This is accomplished through the extensive configurablility of the behavior library.

However, at some point it would be easier to start from scratch than modify the existing behaviors in the Instructional Software Development Kit. The kit was designed with certain applications in mind. Outside of these applications, the kit should be considered a starting point, rather than an all-inclusive tool set.

The applications the kit was designed to build, with little to no coding, are:

- "Clicker" type applications, which present quiz questions and record student feedback
- Interactive demonstrations, such as those which would accompany online course reading material
- Special-purpose calculators

4 Mathematics Evaluation Engine Extension

The mathematics evaluation engine extension allows the developer to evaluate mathematical expressions in infix notation. It provides code blocks that the developer can use in their own custom behaviors.

The extension accepts inputs as strings, and so is suitable for evaluating algebraic expressions as input. The extension does not allow for true symbolic evaluation, but does allow for substitutions. Thus, comparing two symbolic expressions for equality can be accomplished by "sampling", using either random, or pre-decided inputs.

The code blocks provided are:

- evaluate expression •: takes a numerical, infix input and produces a real number (float) as an output. e.g. Input: $1/(2+3)$ ^ 2 produces output: 0.04.
- evaluate expression • with substitutions •: similar to the block above, but will substitute a specified value for each variable before running the calculations. e.g. Input: $-10 * t \text{ ^ } 2, t = 4$ produces output: -160
- compare symbolic expression • = • with substitutions •: compares two symbolic expressions for equality at the specified values for the variables. e.g. Input: $x \text{ ^ } 2 - 1, (x-1) * (x+1), t = 6$ will produce output: **true**

Example 1. The following psuedo-code demonstrates how to use the extension to determine if a student correctly computed the derivative of the polynomial $x^2 - x$.

```
studentRespose := x-1
correctAnswer := 2*x-1

correct := true

for i from 0 to 99
   t := randomFloat * 20 - 10     // Pick a random
                                     real number from -10 to 10

   if not [compare symbolic expression, studentResponse,
           correctResponse, t]
      correct := false
      break
   end if
end for

return correct
```

The example program tests the student's response with the correct response for

100 different, randomly selected inputs. In this example, it is highly improbable (though not impossible) that the student's incorrect response will be marked as "correct". The developer can decrease the chances of a false positive by repeating the loop more than 100 times, or hand-picking known inputs that will yield a solid test.

It would not be difficult to extend the above program so that the student was given a random polynomial to differentiate. The code blocks were designed with adaptive learning software in mind.

Adaptive learning software learns the student's strengths and weaknesses, and tailors its presentation to them. Such software can be especially beneficial in a developmental course. However, the software needs the capacity to analyze student responses on the fly.

Example 2. The sample code below could be used in a lesson on the power rule for derivatives:

```
if studentScore >= 70 then p := randomInteger( -9, 9 )
else p := randomInteger( 1, 9 )

if studentScore >= 70 then q := randomInteger( -9, 9 )
else q := randomInteger( 1, 9 )

a := randomInteger( -10, 10 )
b := randomInteger( -10, 10 )

problemDisplayed := [string a*x^p + b*x^q]
correctAnswer := [string p*a*x^(p-1) + q*b*x^(q-1)]
```

If the student's score is less than 70, then she'll be given a simple polynomial, such as $6x^3 - 4x^7$. However, if her score is at least 70, she could be assigned the relatively harder polynomial-like function with negative powers.

By taking advantage of the mathematics evaluation code blocks, the developer could write a program that adapts its difficulty to the student's needs. The students responses could also be analyzed to look for certain common mistakes.

5 Behavior Library

The behavior library provides tools to implement the user-level functionality, ranging from buttons and text input boxes to client-server communication. To better understand what the purpose of the behavior library is, we'll outline a typical development work-flow:

- First, the developer places *actors* on the screen. For example, two such actors may be a "math input" box, and accompanying "submit" button

- Second, each type of actor comes pre-configured with select behaviors. These behaviors can be configured using the *inspector*. (Refer back to Figure 2 on page 3). More behaviors can be loaded, or existing ones removed.
- Third, for each behavior, *attributes*, or variables, are configured. Some of these attributes configure the Instructional Software Development Kit's *event-triggering system*, which is discussed below.

One of the more tedious issues is allowing the actors to communicate with each other. Suppose the developer wishes their application to show a student a question, then upload the student's response to a server. The Instructional Software Development Kit provides the required functionality.

The developer will need to place a button, a text input box, and the "Simple Data Submission" actor that handles communication with a specified server. The button is configured with the "Push Button Behavior", the text input box with "Math Input Behavior" and the data submission module with "Simple Data Submission Behavior".

When the user clicks the "submit" button, this action must tell the "Simple Data Submission" module actor to fetch the user's input, store in the text input box, and upload it to the server.

In our example, one event is needed. The event is triggered when the user clicks the "submit" button. The code associated with the event is part of the "Simple Data Submission Behavior". However, this behavior is not attached to the "submit" button actor. Rather, the data submission is its own stand-alone actor. Therefore, the following must occur:

- User (the student) enters their response into the text input box, which is stored in a variable within the "Math Input Behavior", then clicks the "submit" button.
- The submit button triggers an event, "SubmitData", that is code housed in the "Simple Data Submission Behavior".
- The "Simple Data Submission Behavior" reads the student's input from the "Math Input Behavior"
- The "Simple Data Submission Behavior" connects to a specified server, and uploads the data.

The Instructional Software Development Kit provides an approach that allows such configuration using only the inspector–hence, with no coding. For any behavior that specifies the name of an event, such as the event to be triggered when a button is pushed, the development kit's referencing system can be used.

As part of the kit, a "Unique Identifier" behavior is provided. This behavior, when attached to an actor, will allow the developer to give that actor a unique name.

The name can be specified with the event in any attribute which requires an event name. The format is **Event Name | Identifier**.

For example, in the case of our data submission, the "Simple Data Submission Behavior" houses an event called "SubmitData", which, when called, executes the code to communicate with the server. The "Push Button Behavior" requires the developer to specify an event to trigger when the button is pushed. To make our example work, we need the button to trigger the "SubmitData" event.

The "Simple Data Submission Behavior" is attached to an actor. Suppose we name this actor "DataUpload" using the Unique Identifier.

The "Push Button Behavior" event should be set to **SubmitData | DataUpload**. This tells Stencyl to loop through all the actors with the identifier "DataUpload", and trigger the event "SubmitData" in all behaviors attached to that actor. Since one of these behaviors is the "Simple Data Submission Behavior", the code which uploads the data will execute.

Figure 6: Configuring a button to trigger the event "SubmitPrompt" in an actor named "Clicker"

Actors must also be able to read and write attributes housed in behaviors in other actors. The Instructional Software Development Kit implements the following scheme for behaviors that specify a "target"–an attribute where data should be read or stored.

[Unique ID].[Behavior].[Attribute Name]

It should be noted that the internal names of all attributes are prefixed with an underscore. For example, of a behavior has an attribute "width", then the internal name is "_width", and so should be referenced in this way in the "Attribute Name" above.

Figure 7: Configuring a variable to be written into ("Targets"), and read from ("Value")

6 Intended Uses

The purpose of the Instructional Software Development Kit is to simplify the process of authoring small, light-use applications. It's not designed to efficiently build large-scale applications to compete with the major course content management systems.

Since small-scale apps can be quickly developed, the Instructional Software Development Kit should find use in experimental courses and higher-level courses which are beyond the scope of the tools provided by the major publishers.

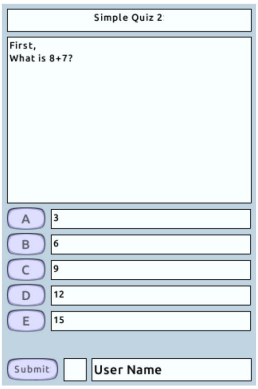

Figure 8: This simple clicker demonstration was written using the Instructional Software Development Kit. The clicker is designed as a mobile app. It downloads an instructor's quiz, prompts the student for responses, then uploads them to a server.

Stencyl also implements Box-2D as its physics engine, which allows for quick development of interactive demonstrations.

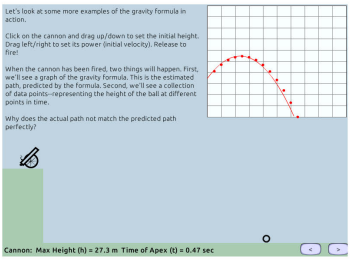

Figure 9: Box-2D is used to power the cannonball in this interactive demonstration. Such demonstrations could prove particularly useful in an online classroom setting, in which live demonstrations would not be feasible.

7 Limitations

There are a number of limitations to the Stencyl platform that are worth noting. First and foremost, while the platform is finding increasing use in the classroom, it was designed as a game development tool. Hence, much of the documentation is written with this use in mind. Furthermore, most articles and user communities are targeted toward game developers.

Stencyl is also proprietary. While the cost is modest (199 USD yearly subscription fee to target mobile platforms), it is a consideration.

There are also a few issues regarding iOS devices. As of time of writing, iOS apps must either be downloaded from Apple's store, or the developer must pay a 300 USD fee to distribute apps outside of the Apple network. This matter can complicate distributing apps to students with iOS devices.

While the Instructional Software Development Kit may ease application development on the user-end, server-side code must be authored using traditional tools (e.g. PHP, Perl). As an example, the server-side code in our data submission example would receive the data, store it in a file, and produce the output for instructor use. None of this code could be developed with Stencyl.

8 User Manual and Source Download

The Instructional Software Development Kit can be downloaded at:

`http://anorthogonaluniverse.com/math/isdk.html`

and user manual at:

`http://anorthogonaluniverse.com/math/isdk-manual.html`

Other resources:

- Stencyl Main Site: `www.stencyl.com`
- Stencyl Education Site: `www.stencyl.com/education/faq`