Academic Community Engaged Teaching Case: Driving Innovation through Project-Based Learning in Game Programming

Shannon, Li-Jen
lys001@shsu.edu
Computer Science Department
Sam Houston State University,
Huntsville, TX 77340 USA

Dalat Ward, Yaprak
y_dalatward@fhsu.edu
Advanced Education Programs
Fort Hays State University,
Hays, KS 67601 USA

Abstract

The Master of Instructional Technology Program at Sam Houston State University, a public university in Texas has opened a door of opportunities for its students who are science/non-science educators, the K–16 administrators, and private firm employees so that they become more competent and competitive as future technology specialists / instructional designers. A successful completion of project-based learning in game programming, a fundamental component of the program, demonstrated how game programming, Scratch is fitted into K-16 curriculum and/or private firm training to help improve learner performances. In addition, game programming not only provided an effective approach to alter the students’ perspectives in promoting science, technology, engineering, and mathematics but it also entailed each student to establish a partnership with the local school district or private firm to practice their required skills in instructional technology.

Keywords: Academic Community Engagement, Project-Based Learning, Gaming, Scratch, Teaching Case

1. INTRODUCTION

In a recent Wall Street Journal piece when McDonald (2013) stated: "Sorry, College Grads, I probably Won’t Hire You – If you’re at all interested in media, technology or related fields, please learn a little computer programming," it led to two major questions about the fear of computer programming: a) Are our students terrified of learning computer programming; and b) should they be scared of programming disregarding how much they like or dislike science, technology, engineering, and mathematics (STEM)? This teaching case helps answer the aforementioned two questions and demonstrates how learners can overcome their fears of learning game programming to enjoy creating games for teaching, training, or just for fun by means of completing a program built on community-engaged projects.

In the last two decades, the STEM field records of the U.S. students have demonstrated poor performances when compared to the STEM records of other nation students (Education Week, 2013). These findings continue to be
alarming and could be interpreted as impacting the U.S. economic development in the long run and eventually, resulting in the U.S. losing her leading role among her global alliances. With this academic community engaged teaching case, the authors demonstrated that by using Web 2.0 free online game software, Scratch, the learners can be motivated to become more interested in computer programming resulting in affecting the STEM performances of students in whatever age group and at whatever grade level they may be. Furthermore, the project demonstrated that it was never too late for the learners to enjoy learning the concepts of logics, computing thinking, and trouble shooting skills which have a huge impact on getting potential educators involved in taking part in raising the bar in the STEM fields.

The project was introduced to the students of the Master of Instructional Technology (MIST) Program at Sam Houston State University (SHSU), a public university in Texas with the goal of opening doors of opportunities for them so that they would become competent technology specialists / instructional designers. Through this program, the students who were science educators, non-science educators, K-16 educational system administrators and/or private firm employees, not only became competent computer programmers but they also connected with their communities fulfilling the motto of the university which is: The measure of a life is its service; this program is built on academic community-engaged projects.

The project entailed each candidate to establish a partnership with the local school district or a private firm to showcase their skills in instructional technology. The aim of the program was to provide a more effective approach to overcome the candidates’ perspectives in computer programming by utilizing a game-programming, Scratch. In fact, many MIST graduates from SHSU had voluntarily started to successfully implement game programming into their K-16 curriculum and/or to their private firm training. A total of five cohorts including 63 students during the past three years demonstrated how learner performances could be improved in various subjects including language art, music, physical education, and particularly in STEM. One teaching case was made up of three-week course work which required a minimum of nine working hours per week. The most rewarding outcomes regarding the MIST student projects were the students’ comprehension of the computing-thinking process. In addition, the students felt comfortable reviewing the design process to troubleshoot encountered issues and were willing to share their learnings with their peers and other learners.

2. CASE DESCRIPTION

A teaching case, made up of three weeks, was achieved by the following segments:

I. Program Schedule

Week 1: Introducing the Scratch tools and functions (see appendix B); demonstrating the engineering design process and flowchart; and reviewing one online article: http://web.media.mit.edu/~kbrennan/files/Brennan_Resnick_AERA2012_CT.pdf

Week 2: Brainstorming the game design to fulfill the needs from the students’ partnership.

Week 3: Designing, implementing, testing, and debugging the game.

II. Course Goals

The goal of this assignment was to provide basic programming software knowledge to develop useful educational materials based on IDEA (Individual Development and Educational Assessment) goals:

1. The students developed specific skills, competencies, and points of view needed by professionals in instructional technology.
2. The students learned how to find and use resources for answering questions or solving problems.
3. The students learned to analyze and critically evaluate ideas, arguments, and points of views.

III. Course Objectives

1. Each student created one teaching topic with an interactive format which would be created by Scratch.
2. Each student had to use his/her website to provide a diagram (game design) and input for Scratch Review.
3. Each student uploaded their game design into http://scratch.mit.edu/.
4. Each student uploaded their project link to his/her website.
5. Each student provided the text content in the website to explain what tools and effects had been applied to this educational game.
3. COURSE INSTRUCTION

The students were asked to create an interactive game project (Event-Driven Program) which was either a math problem, science experiment, color wheel, music lesson, or any task which required the user/s to respond by pressing the keystroke or clicking on the mouse. Students had to locate an institution or organization to have a targeted population to use their final products. This project had to be evaluated by at least three targeted populations (audiences). The evaluation had to be posted on student website. The rubric used for the project was to motivate the students that “Try to keep this project simple, short, and fun!” (See Appendix A).

Flowchart Instruction: Developing Software Methods

As part of the teaching case, the students were introduced to algorithm as a step-by-step process that could clearly be defined to assist the project design flow. When a computer program is created, an algorithm is typically implemented. Long and complex algorithms can be broken down into a collection of smaller algorithms that are easier to understand and manage (Kelleher & Pausch, 2011). In fact, by creating an animation by means of using Scratch, algorithm becomes easy to comprehend (Lamb and Johnson, 2011). As computer users create a new project, they are required to write a script. A screenplay is usually organized as a series of scenes which are made up of multiple shots (frames). A shot is a piece of the story that the objects (sprites) are displayed at the designated positions. Once the screenplay is accomplished, the designer develops actors and actresses (sprites) in the storyboards (MIT, 2011).

Each character in motion and action within a shot is defined with a sequence of codes (blocks) under each sprite. These blocks are to define where the actors should stand or move, and when and what to react if the condition was met. Each change of the scene requires a movement which is a sequence of additional codes (function blocks) to accomplish the desired movements of showing the shots. Creating a Scratch program is much like creating a 2D animation film, and modern computer software projects are often managed in a way that is quite similar to film projects.

The writing can easily be followed by a flow chart or a diagram as indicated with the following examples: http://www.flowhelp.com/flowchart/index.html#Functional%20Charts http://www.rff.com/flowchart_samples.htm

The Software Development Cycle

![Figure 1. The software development cycle](http://www.flowhelp.com/flowchart/index.html#Functional%20Charts)

**The Design Phase.** The software design is based on specifications from the client, which describe what the program should do. Modern object-oriented software specifications include descriptions of the objects needed for the software and the methods associated with those objects. The design for a virtual world should include descriptions of:
- The necessary sprites
- The function blocks associated with each sprite
- The initial stage

**The Implementation Phase.** For virtual world software this includes adding the necessary sprites to each stage; setting the initial scene; and coding the function blocks for the stages. The process of coding includes translation of each block’s algorithm into a particular programming language as well as actually entering the instructions into a computer system.

**The Testing Phase.** Several different types of software tests exist including unit tests and integration tests. A unit test checks an individual software method to see if it works all by itself. An integration test checks to see if a method works when it is mixed in with other methods to form an overall software package. Typically, unit
tests focus on the algorithmic correctness and efficiency of an individual method, while integration tests focus on side effects.

**The Debugging Phase.** Debugging is what leads the development process to become a cycle, because it’s often necessary to go back to the earlier phases when debugging software methods. Software developers may need to correct errors in the design of methods, in their coding, or in the initial setup of the software, or the initial placement of the sprites on the stage.

**Programming With Logical Structures**

In modern object-oriented programming, an algorithm is a method of step-by-step procedure. A collection of methods together structure the components of a more complex algorithm in a sequential manner. However, several constructs can be executed at the same time, such as in a Do together function. There are various types of logic sequences in algorithm. The common used ones are such as Linear, Selection Sequences – Branching, and Boolean Logic.

**Linear Sequences.**
The instructions in this logical structure is to activate the "methods"(commands) in a straight line. The sequent functions must be clearly stated with correct entry and exit conditions. A successful procedure depend on a completed and correct actions in the appropriate order.

![Figure 2. Linear Sequences](image)

**Selection Sequences – Branching.** A selection sequence, or branching routine, splits the direction into separate paths according to the computing reaction to the flow of instructions. Based on a simple true-or-false condition, binary branching occurs when the flow of logic splits into two possible predesigned paths. When the design is based on a single condition with many possible values, multiple branching occurs when the path of logic in an algorithm divides into more than two paths.

![Figure 3. Branching](image)

**Boolean Logic.** The true-or-false conditions that exist in selection sequences and repetition sequences are most often a comparison of two values. There are six logical comparison operators used in mathematics and computer programming: equals, does not equal to, is less than, is greater than, is less than or equal to, and is greater than or equal to as indicated in the table (See Appendix C). The form of Boolean logic exists in branching and looping routines are computed with three common operations: AND, OR, and NOT (See Appendix D).

The logic theory is the foundation of computing thinking when designing a good program. With the basic concept of algorithm as the cornerstone, the students should be able to create a simple but fun game without advanced computer programming background.

**5. CONCLUSIONS**

As the students of the Master of Instructional Technology Program at Sam Houston State University, a public university in Texas completed their three-week academic community-engaged projects which involved game programming, the students were able to demonstrate their knowledge in the engineering design process, fulfill the needs from the students’ partnership, and reinforce their learning by playing games. One remarkable fact about these students was that they were not computer scientists, but they all had the desire to be the best instructional designers in the K-16 education systems and/or in private firms. This teaching case not only helped the students overcome their fear of learning computing science but also provided the opportunity for the students to outreach to the their communities. Many students shared their experiences regarding how much they enjoyed creating
games for teaching and training even after having earned their degrees.

As a conclusion of this project, the implications were that if more educators implemented this teaching case in their environments, there would be resulting in more confident hiring practices of college graduates by leading companies. Regarding various subjects, a great learning curve may occur as students are asked to create an animation or game from Scratch as an assignment rather than a typical digital presentation using PowerPoint. In fact, this case demonstrated that, students should be allowed to use their imagination to define and present what they learn from the course to become more creative. The authors have witnessed that this teaching case motivated many students resulting in an interest in computing education by using Web 2.0 free online game software. The authors will continue conducting research in applying newer game software and technology to further motivate the educators to implement game programming into any subject.

6. REFERENCES


APPENDIX A

Teaching Resources for Scratch Lessons

I. MIT
GETTING STARTED WITH SCRATCH:

SCRATCH CARDS:
http://info.scratch.mit.edu/Support/Scratch_Cards

FEATURED PROJECTS:
http://scratch.mit.edu/channel/featured

VIDEO TUTORIALS:
http://info.scratch.mit.edu/Video_Tutorials

II. Scratch.Redware.Com

INTRODUCTION:
http://scratch.redware.com/lessonplan/introduction

DRAW A SPRITE AND BACKGROUND:
http://scratch.redware.com/lessonplan/drawspriteandbackground

TURTLE GRAPHICS:
http://scratch.redware.com/lessonplan/turtlegraphics

MOVING AND SENSING:
http://scratch.redware.com/lessonplan/movingandsensing

SOUNDS AND GRAPHICS:
http://scratch.redware.com/lessonplan/soundsandgraphics

VARIABLES:
http://scratch.redware.com/lessonplan/variables

BROADCAST AND RECEIVE:
http://scratch.redware.com/lessonplan/broadcastandreceive

MAKE A GAME:
http://scratch.redware.com/lessonplan/makeagame

BUILD A MULTIMEDIA PRESENTATION:
http://scratch.redware.com/lessonplan/buildamultimediapresentation

EXAMPLE PROJECTS:
http://scratch.redware.com/projects

III. VIDEO TUTORIALS

INTRODUCTION:
http://www.youtube.com/watch?v=EfGE-PfMPIA

CREATING YOUR OWN SPRITE(S):
http://www.youtube.com/watch?v=Qi9ooZcBBWg

ANIMATING YOUR SPRITES:
http://www.youtube.com/watch?v=7A_fyik4Qw

ADDING SOUND TO YOUR SPRITE:
http://www.youtube.com/watch?v=AAGHGERpVCo

SPRITE INTERACTIONS WITH VARIABLES:
http://www.youtube.com/watch?v=KFKvgoUSFY8

CREATING A STAGE BACKGROUND:
http://www.youtube.com/watch?v=KFKvgoUSFY8

IV. USEFUL LINKS:
STEP BY STEP: (Written Document)
ANGRY BIRDS: (Robotics & Scratch)

http://www.simonhaughton.co.uk/scratch-programming/


V. OTHER:

http://pwoessner.com/scratch-programming/

http://learnscratch.org/

http://math2033.uark.edu/wiki/index.php/Scratch_Programming

http://www.instructables.com/id/Programming-in-Scratch/

Appendix B. Project Rubrics

<table>
<thead>
<tr>
<th>Rubrics</th>
<th>Indicator Not Met</th>
<th>Indicator Partially Met</th>
<th>Indicator Met</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diagram (Flow Chart)</td>
<td>0 – 3 points Provided a brief storyboard without detailed movements and chart</td>
<td>4 – 7 points Provided the main theme with partially detailed movements or flow chart</td>
<td>8 points Provided detailed movements and flow chart</td>
</tr>
<tr>
<td>Grade:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sprites</td>
<td>0 – 3 points Provided one Sprite</td>
<td>4 – 6 points Provided two Sprites</td>
<td>7 points Provided 3 unique Sprites</td>
</tr>
<tr>
<td>Grade:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Blocks (Color-Coded Blocks)</td>
<td>0 – 3 points Provided less than 3 blocks which did not have proper instructions</td>
<td>4 – 6 points Provided 4-5 blocks with brief instructions</td>
<td>7 points Provided at least 6 out of 8 types of blocks with details and correct instructions</td>
</tr>
<tr>
<td>Grade:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control &amp; Logic Tools</td>
<td>0 – 3 points Provided only one Control block which the instructions might need to be modified and corrected</td>
<td>4 – 6 points Provided only two Control blocks. However, the instructions might need to be modified</td>
<td>7 points Provided at least three Control blocks which presented the correct procedures and functions.</td>
</tr>
<tr>
<td>Grade:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User</td>
<td>0 – 3 points</td>
<td>4 – 6 points</td>
<td>7 points</td>
</tr>
</tbody>
</table>

©2013 EDSIG (Education Special Interest Group of the AITP)
Appendix C. Boolean Logic

<table>
<thead>
<tr>
<th>Condition</th>
<th>In Mathematics</th>
<th>In Computer Programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>A equals B</td>
<td>( A = B )</td>
<td>( A = B ) or ( A == B )</td>
</tr>
<tr>
<td>A does not equal B</td>
<td>( A \neq B )</td>
<td>( A &lt;&gt; B ) or ( A != B )</td>
</tr>
<tr>
<td>A is less than B</td>
<td>( A &lt; B )</td>
<td>( A &lt; B )</td>
</tr>
<tr>
<td>A is greater than B</td>
<td>( A &gt; B )</td>
<td>( A &gt; B )</td>
</tr>
<tr>
<td>A is less than or equal to B</td>
<td>( A \leq B )</td>
<td>( A &lt;= B )</td>
</tr>
<tr>
<td>A is greater than or equal to B</td>
<td>( A \geq B )</td>
<td>( A &gt;= B )</td>
</tr>
</tbody>
</table>

Appendix D. Boolean Value

<table>
<thead>
<tr>
<th>AND</th>
<th>OR</th>
<th>NOT</th>
</tr>
</thead>
<tbody>
<tr>
<td>true and true = true</td>
<td>true or true = true</td>
<td>not true = false</td>
</tr>
<tr>
<td>true and false = false</td>
<td>true or false = true</td>
<td>not false = true</td>
</tr>
<tr>
<td>false and true = false</td>
<td>false or true = true</td>
<td></td>
</tr>
<tr>
<td>false and false = false</td>
<td>false or false = false</td>
<td></td>
</tr>
</tbody>
</table>