Proceedings of Student-Faculty Research Day, CSIS, Pace University, May 1st, 2015

Enhancing Student Learning of Programming via Gaming Technology

Ari Butowsky and Mary Courtney
Seidenberg School of CSIS, Pace University, White Plains, New York

Abstract

Computer programming is currently seen by many as a tedious and boring subject. Beginning students become frustrated with their assigned tasks and pursue other academic interests. The history of structured programming and object-oriented programming involves the development of programming languages, but learning them is another matter. Universities are trying to motivate students to take programming classes by using gaming technology. Software being explored to this end are MIT Scratch, AgentSheets, Alice, Greenfoot, and Objectdraw. A game using some aspects of today's programming techniques is Frogger as exemplified by its source code. An ideal module would include a curriculum that uses Alice/MIT Scratch/AgentSheets first, followed by Greenfoot and Objectdraw. That way, students become engaged in programming before learning about source code. Hopefully, the knowledge gained throughout the years regarding computer programming will allow educators to use this kind of module to incentivize kids to learn computer programming again.

Keywords: coding software, computer programming, gaming, object-oriented programming, structured programming

I. Introduction

Computer programming can be so complex that it can push beginning students away. In fact, there was a 50% decline in incoming freshman computer science majors nationwide from 2000 to 2005 and an 80% decline in female computer science majors from 1998 to 2004. This was due to students' initial difficulty in learning programming skills, as well as the notion that computer science programming is difficult and boring [5][28].

To better motivate students into pursuing computer programming, various universities have turned to using gaming technology to teach their students. Those efforts, along with some relevant history, are discussed below.

II. Background Information

There are two different types of computer programming: structured and object-oriented.

A. *Structured Programming*

Structured programming uses top-down analysis for problem solving, modularization for program structure and organization, and structured code for the individual modules [43]. Here is a list of terms that explain what is meant by that:

1. Top-Down Analysis: subdividing a large problem into several smaller tasks or parts. Doing this simplifies or reduces the complexity of the process of problem solving [43].

2. Modular Programming: organizing the instructions that programs require for a computer. Programs are divided into modules, subroutines, or subprograms, each having a job to do and is easy to write. This simplifies programming by making use of a highly structured organizational plan and by reducing the need for the GOTO statement. There is a correlation between the subdivisions of a problem obtained via top-down analysis and the modules obtained via modular programming [43].

3. Structured Coding: modules are subdivided into and organized within various control structures [43].

4. Control Structure: represents a pattern of execution for a specific set of instructions. It determines the order of instruction execution. Each pattern of execution represents one of three types: sequentially, conditionally, or repetitively. Component statements within each control structure are executed in either of these ways, so the order in which a module's instructions are executed is determined by the use of control structures. Therefore, the module in question is said to be "structured" and the code is structured [43].

5. GOTO Statement: provides a jump from the “goto” to a labeled statement in the same function. Figure 1 demonstrates this [31]. Structured code does not include a GOTO statement for a number of reasons: although it affects the order in which a module’s statements are executed, it does not contain any other statements; a GOTO statement represents no exact pattern of execution (i.e. It just jumps to a statement other than the next one in line.). Using structured code reduces program complexity because program instructions are organized into discernable patterns, thus eliminating the need for the GOTO statement [43].

6. Compound Statement: groups one or more statements into a single statement. It has its own rules regarding how it executes. Program execution is transferred to the first statement in the statement-sequence, followed by the rest of the statements in the order they occur in the program’s text. The compound statement’s execution ends when the execution of the last statement in the statement-sequence is terminated, or when one of the statements in the statement-sequence transfers program execution to a statement that lies outside the compound statement (i.e. See Figure 2 for the syntax of the compound statement.) [12].

In 1964, “structured programming” was first introduced with only three “control structures”:

1. **Sequence:** represents sequential execution and is implemented in Pascal via a compound statement with a straight line execution path (i.e. See Figure 3.) [43].

2. **Selection:** represents conditional execution and is implemented via statements that support decision making, where a particular computer program selects a result from among a list of alternatives. This is an “If…then…else” statement, whose template is [38][43]:

```
If logical-predicate Then
-- statements to execute if the predicate evaluates to true –
Else
-- statements to execute if the predicate evaluates to false –
EndIf
```

The logical predicate comprises one or more logical or relational operations and produces a 0 (false) or 1 (true) as a result (i.e. See Figures 4 and 5.) [38]. If the predicate evaluates to true, then certain statements will execute. If the predicate evaluates to false, then other statements will execute [38].

3. **DO-WHILE:** represents repetitive execution and is implemented via statements where a certain condition is tested after execution of the loop body. If the condition is true, the loop is executed again, but if the condition is false, the loop is terminated [35][36]. Below is an example program that demonstrates this in Pascal, except that instead of using a do-while loop, it uses the equivalent repeat-until loop:

```
program prog_19(input, output);
var score, sum: integer;
begin sum := 0;
repeat write('Enter score (-1 to quit): ');
readln(score);
if score <> -1 then sum := sum + score until score = -1;
writeln('Sum = ', sum);
end.
```

The above program requires that the user enter a number and the screen will display it as, for example, “Enter score (-1 to quit): 74”. If the user enters -1, however, the program will terminate and display the total sum of all the numbers the user entered before typing -1 [35].

While the three control structures made the GOTO statement unnecessary, they and their implications were almost entirely ignored in the United States due to the GOTO statement being heavily ingrained in programmers’ minds during the mid-1960s [43]. Then, in 1968, computer scientist Edsger Dijkstra furthered his then 20 year crusade for structured programming by publishing a letter to the editor in the Communications of the ACM arguing that the quality of a programmer’s code increases as the number of GOTO statements used decreases [16][43][46].

Structured programming’s wide acceptance, came after the success of the 1972 New York Times project, which was developed by Harlan Mills’ programming team at IBM. It was a system utilized in automating the New York Times’ clipping file [43].

Structured programming was a programming revolution and an important advancement in computer software. People began turning more to its philosophy and techniques [43].

B. Object-Oriented Programming

Object-Oriented Programming (OOP) is based on a hierarchy of classes, as well as well-defined and cooperating objects [27]. These terms explain further:

1. **Class:** defines the data and the methods needed to work on that data. All program data is wrapped in a class. For example, the following program uses the “java.lang.System” class to print a character string to the command line [27]:

```
class Example {
public static void main(String[] args) {
System.out.println(“I’m a simple program.”); }}
```

Classes in the Java platform API libraries define objects that share a common structure and behavior. The “java.lang.System” class as seen here defined standard input, output, and error streams, along with access to system properties [27].

2. **Object:** an executable copy of a class. There can be any number of objects of a certain class in memory at one time. For example, the following program is a rewritten version of the example program in the definition of the term class. It illustrates how to access String methods to create String objects [27]:

```
class Example {
public static void main(String[] args) {
String text = new String("I'm a simple program. ");
System.out.println(text);
String text2 = text.concat("that uses classes and objects.");
System.out.println(text2); }}
```

Output - I'm a simple program.
I'm a simple program that uses classes and objects.

This program has four String objects, which were created for the concatenation operation, text object, text2 object, and a String object created behind the scenes from the "that uses classes and objects" string. The "java.lang.String.concat" method converts these String objects into editable string objects to perform the concatenation. In addition, an object of the Example class has been stored in memory [27].

3. Inheritance: defines relationships among classes. For example, in Java, the methods from "java.lang.Object" are inherited and implemented by its subclasses, namely every class in the Java API libraries. In addition, each class adds its own set of fields and methods to what it inherits from its superclass(s), kind of like what a child inherits from his parent(s) [27].

OOP began in May 1967 when the first Object-Oriented Language (OOL), Simula 67, was introduced [41]. It was designed for creating simulations, the work on which at that time dealt with exploding ships. It was discovered that these ships could be grouped into different categories, with each ship type having its own class, which would generate its own unique behavior and data. Basically, Simula 67 introduced the concept of a class and the objects of a class [47].
In the early 1970s, Alan Kay's team at Xerox Parc created Dynabook, the first ever personal computer, which provided graphics-oriented applications based on Simula 67 [2][4][46]. It used Smalltalk, an object-oriented, simulation, and graphics-oriented language that allowed objects to be changed, created, or deleted. Smalltalk introduced the concept of inheritance, letting it surpass Simula 67 in functionality. However, OOP gained momentum during the 1970s thanks to Simula 67, leading to the creation of programming languages like Pascal and Lisp [1][47].

In the 1980s, computer scientist Bjorn Stroustrup created C++, an OOL that played an important role in the creation of graphical user interfaces (GUI) that are compatible with OOP languages. This paradigm of programming aided the development of event-driven programming, where a program execution's flow is determined by events like a mouse click or key press [1][15][42][47].

In 1994, James Gosling's group at the Sun Company created Java, a programming language designed to be a simpler version of C++. It was eventually marketed to be used for programming Internet applications, which granted it widespread popularity for programming applications for the then booming Internet [1][18].

Advancements in many areas of computing were made thanks to using an object-oriented approach. OOP will continue to evolve in the future since it is a powerful language that has been improving overtime [47].

III. How Universities Enhance Student Learning of Programming

Universities are employing unique methods to make software programming appealing to their student populations.

A. Using Scratch!

During the summer, the University of Texas (UT) at Dallas holds coding camps where students are exposed to computer programming using MIT Scratch, a programming environment with an interface that enables users to learn how to create programs easily. It has a level of functionality to keep users interested while they learn programming concepts. MIT Scratch also features drag-and-drop programming, which leads to no syntax errors and allows users to focus on the programming's logic rather than code. Users can create 2D animations of objects, make characters sing, create drawings, develop interactive games, and even narrate stories using images (i.e. See Figure 6.) [14][40][45].

Students can also develop games and make music. Sprites/objects, each having their own unique code segments, are used for the games and various programming concepts are used to make music. Sophisticated concepts like event driven programming and cloning objects are used to complete certain projects [14].

Students can also use MIT Scratch to control finch robots, which are USB-powered robots developed to learn programming. This entails writing programs that cause the robots to perform. The very nature of MIT Scratch itself makes writing these programs easy and fun for users [14].

B. Using Video Games to Teach Computer Coding

Professor Alexander Repenning of the University of Colorado developed a curriculum called Scalable Game Design, which teaches students concepts of math and science, as well as how to create games. The goal of this curriculum is twofold: to bring computer science to middle schools with the aim of creating a larger IT workforce and combat the notion that computer programming is tedious and boring [28][33].

One tool used is AgentSheets, which utilizes Visual AgenTalk, a programming language that employs Tactile Programming. This allows programs to be created with enhanced visual representations and interactive interfaces [34]. Using AgentSheets, a Frogger-like game can be created in less than 3 hours. In addition, AgentSheets is powerful enough to enable middle school students to implement sophisticated AI algorithms for, as an example, finding the shortest path in a maze. Finally, AgentSheets works for game and computational science applications, as well as supports the transition to traditional programming such as Java [35].

Scalable Game Design makes students experience Flow so their design skills match certain design challenges. According to Mihalyi Csikszentmihalyi, the former chair of the University of Chicago's Department of Psychology, Flow means engagement with what you are doing, wanting to continue the activity for its own sake, and "the way people describe their state of mind when consciousness is harmoniously ordered." Examples of activities that lead to this "state of mind" include sports, art, work, hobbies, and games. However, Flow only happens when a person is interested in what they are doing, but it is only after doing the activity that he feels it was enjoyable. Flow can also be experienced while doing something active that entails doing difficult, risky, or painful things. Such activities usually stretch people's capacity, provide a challenge to their capabilities, and/or involve some discovery or novelty. This is because they are motivated by the quality of the experience they are having while engaged in the activity [48].

Students experience Flow by scaffolding through various game design patterns to progress from simple arcade games to games requiring sophisticated Artificial Intelligence. Throughout, students develop IT fluency based on intellectual capabilities and learn fundamental IT concepts and contemporary IT skills [33].

IV. Coding Software Programs

Despite the best efforts of universities, however, new software needed to be developed to bring back an interest in computer programming amongst students due to the complex and frustrating nature of computer programming itself. This software included Alice, Greenfoot, and Objectdraw.

A. Alice

Alice allows students to create 3D animated movies and games where they control the behavior of 3D objects and characters in a virtual world [5][19].

Using Alice requires dragging and dropping graphical tiles to create instructions in a program that correspond to statements in programming languages like Java, C++, and C#. This kind of interface introduces students to programming in a supportive and engaging environment, from which they gradually transition to programming. Alice serves as a gateway for students to all the programming concepts taught in introductory computing courses [5].

Alice's drag-and-drop interface prevents users from making syntax errors and program sequences are displayed as animations so users can see their mistakes and more readily fix them. This way, users develop problem-solving skills [5].

B. Greenfoot

Greenfoot enables users to create classes, instances, and members using a graphical user interface (GUI) [4]. Each Greenfoot program divides user-created classes into three categories: World, Actor, and Utility. Subclasses of the World class are created to customize the program, subclasses of the Actor class are created for each type of game entity, and Utility classes are used to accomplish the functionality needed by several classes. Upon the program's execution, Greenfoot starts a main loop where each iteration invokes the act() method of every object instantiated from subclasses of the Actor class [3].

A Greenfoot World is an invisible grid of cells, each containing one or more Actor objects. This grid corresponds to a coordinate system, except that the Y-axis points down instead of up (i.e. See Figure 7.) [3][21].

Users can invoke methods and create classes and objects by clicking the mouse. This helps in teaching the differences between classes and objects while the concepts of class, object instance, and methods are taught using the Actor class [3].

The Greenfoot environment provides much of the foundation needed to implement many games and simulations. For example, say a user wants to make a character move to the right and left and bounce off world boundaries using the keyboard. No problem, for Greenfoot keyboard handling methods make keyboard control easy. As for boundary reflection and direction changing, the user just needs to add if/else statements to the code [3].

Indeed, Greenfoot makes it easy for beginners to create their own games thanks to its easy-to-understand interface and working environment. Best of all, Java makes the Greenfoot environment powerful enough for users to write impressive, flexible, and sophisticated applications [3][21].

C. Objectdraw

Objectdraw takes the following approach: "objects first", "events first", and "concurrency early". Graphics are used as a pedagogical aid, meaning that graphics are used to help teach various computing concepts to students. Graphical objects are used and event-driven programming is done by students from the start and concurrency is introduced about one-third of the way through the textbook [6][7].

Objectdraw supports the teaching of Java by simplifying the construction of programs that draw graphics and interact by reacting to mouse events (ex. Clicking and dragging the mouse.). It provides simplified support for handling mouse events, facilities that help simplify the management of multiple Threads, and an assortment of classes for displaying drawings that provide a more object-oriented "look and feel" than the standard Java Graphics class [8].

D. Which software is the best?

Alice, Greenfoot, and Objectdraw are all very powerful. All three present similarities and differences, which cause them to be most suitable for different types of users (i.e. See Table 1.).

Alice is the easiest to install because it can be downloaded and installed from its online source without the need for a Java Development Kit (JDK), but the JDK will be required for versions 3.0 or higher. Alice's tutorial is fun, interactive, easy to follow, and is found on the software itself. Also, utilizing Alice requires no knowledge of Java because methods can be created and placed in the editor screen while the code writes itself. This user input helps create animations, backgrounds, and shapes that are displayed on a separate screen. Alice creates a real introduction to computer programming for beginning students without going into any tedious coding from the start. As a result, Alice can be recommended for middle school, high school, and first-year college students.

As for Greenfoot and Objectdraw, both have to be installed with the proper JDK due to their utilization of Java programming. However, while Greenfoot can be downloaded and installed from its online source (like Alice), installing Objectdraw requires an understanding of how to install it as an extension to another piece of software like BlueJ [20]. In addition, tutorials on how to use them are only found online [22][23][24][25][26][44]. Furthermore, using Greenfoot and Objectdraw requires lots of knowledge in Java due to their coding aspects, but they are similar to Alice in that user input translates into animations, shapes, and backgrounds shown on a separate screen. Therefore, both Greenfoot and Objectdraw are recommended for high school and first-year college students who are interested in programming and coding while Alice is recommended for people who want to get an introduction to just programming.

V. Example Game

Figure 8 shows the link to a version of Frogger created using Objectdraw [10]. It is composed of various loops, classes, objects, and other programming aspects that come together to bring an entire game to life. Here are some hints that should simplify everything somewhat.

Importing Objectdraw lets the user utilize all of the methods associated with it while importing "java.awt*" allows the user to use the classes necessary for creating user interfaces and for painting graphics and images [30].

Furthermore, the class Frogger extends WindowController, a class that produces programs that handle mouse events and draw graphics in one window on the screen. Frogger inherits the properties of WindowController [6].

Next, there are these statements:

```
private static final int HIGHWAY_LENGTH = 700;
private static final int LANE_WIDTH = 100;
private static final int NUM_LANES = 4;
private static final int HIGHWAY_WIDTH = LANE_WIDTH *
NUM_LANES;
private static final int LINE_WIDTH = LANE_WIDTH / 10;
private static final int HIGHWAY_LEFT = 50;
private static final int HIGHWAY_RIGHT = HIGHWAY_LEFT +
HIGHWAY_LENGTH;
private static final int HIGHWAY_TOP = 100;
private static final int HIGHWAY_BOTTOM = HIGHWAY_TOP +
HIGHWAY_WIDTH;
private static final int LINE_SPACING = LINE_WIDTH / 2;
private static final int DASH_LENGTH = LANE_WIDTH / 3;
private static final int DASH_SPACING = DASH_LENGTH / 2;
```

These are declarations of constants, variables/objects that have a fixed value [13]. Take the following for example:

```
public void drawPassingLine (int y) {
FilledRect dash;
for (int x = HIGHWAY_LEFT;
 x<HIGHWAY_RIGHT;
x+=DASH_LENGTH + DASH_SPACING) {
```

In the drawPassingLine (int y) method, a for-loop is declared that starts at 50 and as long as HIGHWAY_LEFT is less than HIGHWAY_RIGHT, x = x + 16.6 [41]. Altering the value of the constants in this method changes how it operates.

There are also while loops and if-else statements in the source code. For example,

```
// Create the lanes.
while (lane <= NUM_LANES) {
// Create the lane telling it where it is, which direction its
// cars should go, what its cars should look like, and what
// the frog is.
// The bottom half lanes go to the right.
if (lane > NUM_LANES / 2) {
new Lane (HIGHWAY_TOP + (lane - 1) * LANE_WIDTH,
HIGHWAY_LEFT, HIGHWAY_RIGHT, LANE_WIDTH,
true, rightCars, theFrog, highway, canvas); }
 else {
new Lane (HIGHWAY_TOP + (lane - 1) * LANE_WIDTH,
HIGHWAY_LEFT, HIGHWAY_RIGHT, LANE_WIDTH, false,
leftCars, theFrog, highway, canvas); } lane = lane + 1; }}
```

This is basically saying that while the variable "lane" is less than or equal to 4, then if "lane" is greater than 2, a new method called Lane will be created with the values (HIGHWAY_TOP + (lane - 1) * LANE_WIDTH), HIGHWAY_LEFT, HIGHWAY_RIGHT, LANE_WIDTH, true, rightCars, theFrog, highway, and canvas. However, if "lane" is less than or equal to 2, then the method Lane will include the values (HIGHWAY_TOP + (lane - 1) * LANE_WIDTH), HIGHWAY_LEFT, HIGHWAY_RIGHT, LANE_WIDTH, false, leftCars, theFrog, highway, and canvas.

When all of these classes, objects, loops, and imports come together, they make a Frogger game. That is the true beauty of programming: being able to type in code and create whatever one wants out of it like a wizard!

VI. Creating a Module to Peak Student Interest in Programming

The ideal module would include the following curriculum:

1. Students would use either Alice, MIT Scratch, or AgentSheets first so they are introduced to programming without having to learn any coding. That way, they can decide whether or not they like the concept of programming and want to continue taking the class.

2. After some lessons in using Alice, students should start using Greenfoot because installing it is easier than installing Objectdraw and it has a fun interface that is easy to understand. The students are introduced to Java for the first time and methods like World() and Actor() are already set up for them. If the students learn from online tutorials how these preset Greenfoot methods work, it should be easy for them to have fun with this software.

3. Next, students should learn how to use Objectdraw to understand how Java works in terms of methods requiring specific parameters in order to function correctly.

4. Finally, students should have to create their own projects by utilizing all of the skills they learned using Alice/MIT Scratch/AgentSheets, Greenfoot, and Objectdraw.

A module that proceeds like this one should help students learn about programming in a fun and interesting way.

VII. Conclusion

In conclusion, computer programming has come a long way since the 1960s. Hopefully the knowledge gained throughout the decades can help educators find ways to get students interested in this topic area by utilizing the one thing many kids of the 21st century like above all else: video games and/or gaming technology.

Appendix

Table 1: Software Comparison

	Alice	**Greenfoot**	**Objectdraw**
Installation	JDK required for versions below 3.0.	Available on website, but requires JDK.	Must be added as an extension of current Java coding software [20].
Tutorial	Found on the software. Interactive and easy to follow.	Found online. Comprises several articles [22][23][24][25][26].	Found online. Comprises several articles [44].
Usage	Requires no knowledge of Java. Methods are created and put in editor screen without coding.	Requires much knowledge of Java.	Requires much knowledge of Java.
Display	Operates based on user input.	Operates based on user input.	Operates based on user input [11].
Suggested Education Level	Middle school, high school, and college students.	High school and college students.	High school and college students.

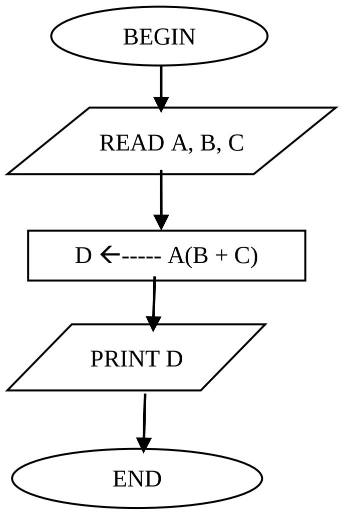
```
program exGoto;
label 1;
var a : integer;
begin a := 10;
(* repeat until loop execution *) 1: repeat if ( a = 15) then begin (* skip the iteration *) a := a + 1;
goto 1;
end;
writeln('value of a: ', a);
 a:= a +1;
until a = 20;
end.
```

Output -
```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

Fig. 1. This demonstrates the GOTO statement. It increases the value by 1 and displays the values reached until a = 20. This loop iteration functions in this manner because it is equal to 10 and not 15 at the start and therefore, the iteration is not skipped and the GOTO statement does not take effect. If it had, it would have sent the program execution back to label #1 [31].

```
compound-statement = 'begin' statement-sequence 'end'
   statement-sequence = statement { ';' statement }
```

Fig. 2. The syntax of a compound statement [12].



```
PROGRAM EXAMPLE_SEQUENCE (INPUT, OUTPUT);
VAR A, B, C, D: REAL;
BEGIN
READLN (A, B, C);
D = A * (B + C);
WRITELN (D)
END.
```

Fig. 3. Flow diagram and code of the Pascal programming language's version of the sequence control structure [43].

```
NOT(x)    x=0  x=1
     --------+----------+
             | 1     0  |
             +----------+
 AND(x,y)  x=0  x=1
     --------+----------  +
        y=0  |    0     0 |
             |            |
        y=1  |    0     1 |
             + ----------  +
 OR(x,y)   x=0  x=1
```

```
--------+----------+
  y=0   |  0     1 |
        |          |
  y=1   |  1     1 |
        +----------+
```

Fig. 4. These provide definitions of the logical operators NOT(x), AND(x,y), and OR(x,y). 0=false and 1=true [38].

NAME	SYMBOL	MEANING
greater-than	*x* > *y*	"x" is greater than "y"
greater-equal	*x* >= *y*	"x" is greater than or equal to "y"
equals	*x* == *y*	"x" equals "y"
not-equals	*x* <> *y*	"x" does not equal "y"
less-equal	*x* <= *y*	"x" is less than or equal to "y"
less-than	*x* < *y*	"x" is less than "y"

Fig. 5. Definitions of the relational operators [38].

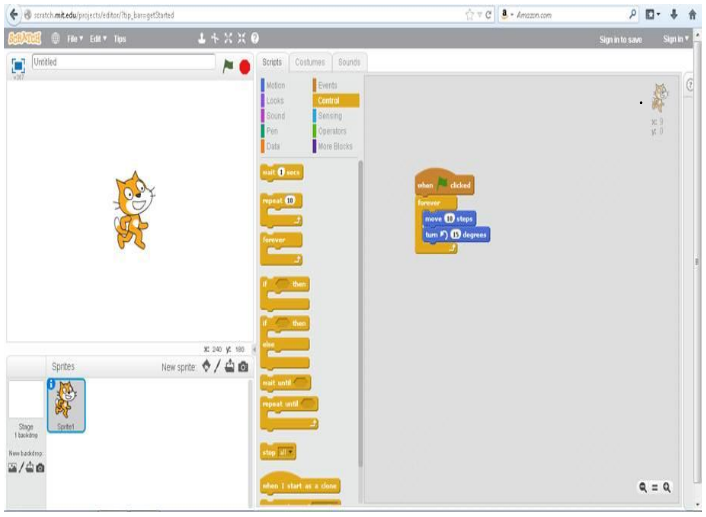


Fig. 6. Interface provided by MIT Scratch. Scripts are dragged and placed onto the grey workspace. Double-clicking the first block of script runs each group placed in the workspace. This makes the sprite/object, on the left, respond [40].

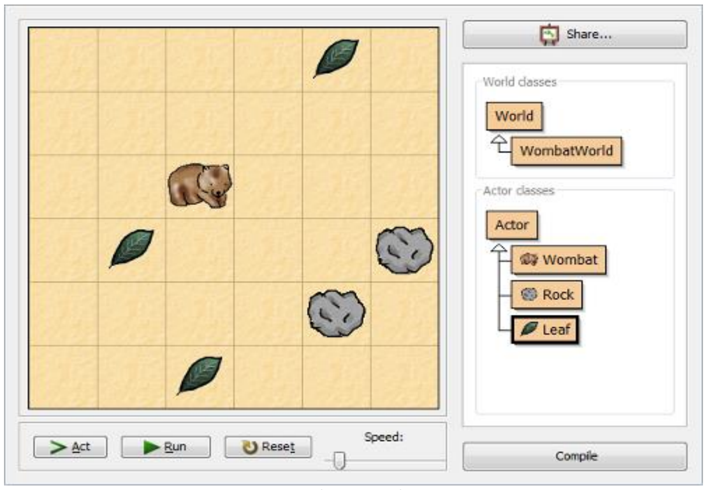


Fig. 7. Greenfoot interface [21].

http://eventfuljava.cs.williams.edu/s04/lectures/Lecture19/CarShowFrogger/Frogger.java

Fig. 8. Link to source code for creating an example version of Frogger [10].

References

[1] "A Brief History of Object-Oriented Programming." *University of Tennessee Department of Electrical Engineering and Computer Science*. University of Tennessee Department of Electrical Engineering and Computer Science, 19 Feb. 2015. http://web.eecs.utk.edu/~huangj/CS302S04/notes/oo-intro.html

[2]"About PARC - PARC, a Xerox Company." *About PARC - PARC, a Xerox Company*. Palo Alto Research Center Inc., 24 Feb. 2015. http://www.parc.com/about/

[3] Al-Bow, Mohammed, Debra Austin, Jeffrey Edgington, Rafael Fajardo, Joshua Fishburn, Carlos Lara, Scott Leutenegger, and Susan Meyer. "Using Greenfoot and Games to Teach Rising 9th and 10th Grade Novice Programmers." *CiteSeerX*. 5 Mar. 2015. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.158.8303&rep=rep1&type=pdf

[4] "Alan Kay." *TED*. TED Conferences, LLC, 24 Feb. 2015. http://www.ted.com/speakers/alan_kay

[5] "Alice Educational Software." - *Carnegie Mellon University*. Carnegie Mellon University, 21 Feb. 2015. http://www.cmu.edu/corporate/news/2007/features/alice.shtml

[6] Bruce, Kim B., Andrea P. Danyluk, and Thomas P. Murtagh. "Objectdraw: Class WindowController." *Java: An Eventful Approach*. National Science Foundation. 22 Mar. 2015. http://eventfuljava.cs.williams.edu/library/objectdrawJavadocV1.1.1/objectdraw/WindowController.html

[7] Bruce, Kim B., Andrea P. Danyluk, and Thomas P. Murtagh. " JAVA: An Eventful Approach ." *Java: An Eventful Approach*. National Science Foundation. 06 Mar. 2015. http://eventfuljava.cs.williams.edu/index.html

[8] Bruce, Kim B., Andrea P. Danyluk, and Thomas P. Murtagh. "JAVA: An Eventful Approach ." *Java: An Eventful Approach*. National Science Foundation. 21 Feb. 2015. http://eventfuljava.cs.williams.edu/library.html

[9] Bruce, Kim B., Andrea P. Danyluk, and Thomas P. Murtagh."Package Objectdraw." *Java: An Eventful Approach*. National Science Foundation. 06 Mar. 2015. http://eventfuljava.cs.williams.edu/library/objectdrawJavadocV1.1.1/

[10] Bruce, Kim B., Andrea P. Danyluk, and Thomas P. Murtagh. "Programming Examples." *Java: An Eventful Approach*. National Science Foundation. 22 Mar. 2015. http://eventfuljava.cs.williams.edu/s04/lectures/Lecture19/CarShowFrogger/Frogger.java

[11] Bruce, Kim B., Andrea P. Danyluk, and Thomas P. Murtagh. "Using the Objectdraw Library with BlueJ." *Java: An Eventful Approach*. National Science Foundation. Web. 16 Mar. 2015. http://eventfuljava.cs.williams.edu/library/readmes/objectdraw4BlueJ.pdf

[12] "Compound Statement." *Irie Tools: The Pascal Compiler Company*. Irie Tools Limited. 24 Feb. 2015. http://www.irietools.com/iriepascal/progref355.html

[13] "Constants." *Cplusplus.com.* C++ Resources Network, 2013. 22 Mar. 2015. http://www.cplusplus.com/doc/tutorial/constants/

[14] "Descriptions - K12 - The University of Texas at Dallas." *Descriptions - K12 - The University of Texas at Dallas*. StateFarm, University of Texas at Dallas. 21 Feb. 2015. http://www.utdallas.edu/k12/desc/

[15] "Event-driven Programming." *TechnologyUK*. TechnologyUK, W3C, 02 Mar. 2015. http://www.technologyuk.net/computing/software_development/event_driven_programming.shtml

[16] "Go To Statement Considered Harmful." *The University of Arizona – University Information Technological Services (UITS).* Arizona Board of Regents, 04 Mar. 2015. http://www.u.arizona.edu/~rubinson/copyright_violations/Go_To_Considered_Harmful.html

[17] Henriksen, Poul, and Michael Kolling. *Greenfoot: Combining Object Visualisation with Interaction*.

[18] "James Gosling and the History of Java." *About.com Inventors*. About.com, 24 Feb. 2015. http://inventors.about.com/od/gstartinventors/a/James_Gosling.htm

[19] Kelleher, Caitlin. "Storytelling Alice." *Washington University in St. Louis: Department of Computer Science & Engineering.* Washington University in St. Louis School of Engineering & Applied Science, Washington University in St. Louis Department of Computer Science & Engineering, 21 Feb. 2015. http://www.alice.org/kelleher/storytelling/index.html

[20] Kolling, Michael, and John Rosenberg. "BlueJ Extensions." *BlueJ*. Oracle, University of Kent, 16 Mar. 2015. http://www.bluej.org/extensions/extensions.html

[21] Lenton, Joseph. "About Greenfoot:." *Greenfoot*. Oracle, University of Kent, 21 Feb. 2015. http://www.greenfoot.org/overview

[22] Lenton, Joseph. "Tutorial 1: Interacting with Greenfoot." *Greenfoot*. Oracle, University of Kent, 16 Mar. 2015. http://www.greenfoot.org/doc/tut-1

[23] Lenton, Joseph. "Tutorial 2: Movement and Key Control." *Greenfoot*. Oracle, University of Kent, 16 Mar. 2015. http://www.greenfoot.org/doc/tut-2

[24] Lenton, Joseph. "Tutorial 3: Detecting and Removing Actors, and Making Methods." *Greenfoot*. Oracle, University of Kent, 16 Mar. 2015. http://www.greenfoot.org/doc/tut-3

[25] Lenton, Joseph. "Tutorial 4: Saving the World, Making and Playing Sound." *Greenfoot*. Oracle, University of Kent, 16 Mar. 2015. http://www.greenfoot.org/doc/tut-4

[26] Lenton, Joseph. "Tutorial 5: Adding a Randomly Moving Enemy." *Greenfoot*. Oracle, University of Kent, 16 Mar. 2015. http://www.greenfoot.org/doc/tut-5

[27] "Lesson 8: Object-Oriented Programming." *Oracle*. Oracle, 19 Feb. 2015. http://www.oracle.com/technetwork/java/oo-140949.html

[28] "NYC Schools to Use Video Games to Teach Computer Coding." *University of Colorado Boulder*. Regents of the University of Colorado, 29 July 2014. 21 Feb. 2015. http://www.colorado.edu/news/releases/2014/07/29/nyc-schools-use-video-games-teach-computer-coding

[29] "Operators." *Cplusplus.com*. C++ Resources Network. 2013. 22 Mar. 2015. http://www.cplusplus.com/doc/tutorial/operators/

[30] "Package java.awt." *Oracle*. Oracle, 2014. Web. 22 Mar. 2015. http://docs.oracle.com/javase/7/docs/api/java/awt/package-summary.html

[31] "Pascal Goto Statement" *Tutorials Point: Simple Easy Learning*. 05 Mar. 2015. http://www.tutorialspoint.com/pascal/pascal_goto_statement.htm

[32] Repenning, Alexander. "Conversational Programming." *AgentSheets*. AgentSheets, Inc., 2 Mar. 2015. http://www.agentsheets.com/Documentation/windows/Reference/conversational_programming.html

[33] Repenning, Alexander. "Scalable Game Design." University of Colorado. 2 Mar. 2015. http://www.agentsheets.com/about_us/press-material/documents/ScalableGameDesign.pdf

[34] Repenning, Alexander. "Visual AgenTalk®." *AgentSheets*. AgentSheets, Inc., 02 Mar. 2015. http://www.agentsheets.com/products/vat/index.html

[35] "Repetition Structures: Do-while and for Statements." *Repetition Structures: Do-while and for Statements*. Pui Ching Middle School. 04 Mar. 2015. http://www.puiching.edu.hk/~wtchung/trace/CPP/cpp11.htm

[36] "Repetition Structures: While Statement." *Repetition Structures: While Statement*. Pui Ching Middle School. 04 Mar. 2015. http://www.puiching.edu.hk/~wtchung/trace/CPP/cpp10.htm#Prog18

[37] "Scalable Game Design." *Outreach and Engagement: University of Colorado Boulder*. Regents of the University of Colorado, 02 Mar. 2015. https://outreach.colorado.edu/programs/details/id/172

[38] Schmalz, Mark S. "Pascal -- Data Structures." *University of Alberta*. University of Alberta, COPLAC, 24 Feb. 2015. http://www.augustana.ab.ca/~mohrj/courses/common/csc370/lecture_notes/pascal.html

[39] Schmalz, Mark S. "PASCAL Programming: Selection and Iteration Structures." *University of Florida: Department of Computer and Information Science and Engineering*. Department of Computer and Information Science and Engineering, 24 Feb. 2015. https://www.cise.ufl.edu/~mssz/Pascal-CGS2462/ifs-and-loops.html

[40] "Scratch Workshop: Introduction to Programming." *UT Dallas Computer Science Outreach*. 02 Mar. 2015. http://www.utdallas.edu/~veerasam/scratch/

[41] Sklenar, Jaroslav. "INTRODUCTION TO SIMULA." *University of Malta*. University of Malta, 24 Feb. 2015. http://staff.um.edu.mt/jskl1/talk.html

[42] Stroustrup, Bjarne. "Some Information about Bjarne Stroustrup." *Welcome to Bjarne Stroustrup's Homepage!* Bjarne Stroustrup, 29 Jan. 2015. 04 Mar. 2015. http://www.stroustrup.com/bio.html

[43] "Structured Programming." *Cal Poly: San Luis Obispo*. Cal Poly: San Luis Obispo, 21 Feb. 2015. http://users.csc.calpoly.edu/~jdalbey/308/Resources/StructuredProgramming

[44] "Summary of Objectdraw API." *Pomona College*. Pomona College, 21 Feb. 2015. http://www.cs.pomona.edu/classes/cs51/handouts/objectdraw-api.html

[45] "Summer – K12 – The University of Texas at Dallas." *Summer – K12 – The University of Texas at Dallas*. StateFarm, University of Texas at Dallas, 01 Mar. 2015. http://www.utdallas.edu/k12/summer/

[46] "Using Gotos." *Steve McConnell*. Construx, 05 Mar. 2015. http://www.stevemcconnell.com/ccgoto.htm

[47] Vennapoosa, Chandra. "The History of Object Oriented Programming | IT Training and Consulting – Exforsys." *IT Training and Consulting Exforsys*. Exsforsys, Inc., 19 Feb. 2015. http://www.exforsys.com/tutorials/oops-concepts/the-history-of-object-oriented-programming.html

[48] Wright, Steve. ""In the Zone": Enjoyment, Creativity, and the Nine Elements of "flow""*Meaning and Happiness.com*. Meaning and Happiness.com, 3 Mar. 2015. http://www.meaningandhappiness.com/zone-enjoyment-creativity-elements-flow/26/