Automatic Generation and Grading of Programming Exercises

By Andy DuFrene
Computer Science Department
College of Engineering
California Polytechnic State University

June 10, 2016
Abstract

In our current age of technological advances, and rising education costs, it is becoming increasingly beneficial to use technology to aid in education, usually through automated lectures or grading. One missing feature from this automated education is complex grading and problem generation. The main objective of this project is to create a program that can automatically generate and grade problems of varying difficulties based on the topic of programming for loops. The project solution was created as a program designed for use in Dr. Clinton Staley’s Intelligent Homework System, which is used precisely for automated exercise grading and generation. Ultimately, the created program operates as expected, and is capable of generating a limited number of for loop exercises across five different difficulty levels. This project is a nice improvement on current automated exercises that will hopefully improve the understanding of for loops in entry-level computer science students.
# Contents

1 **Introduction** 4

2 **Background** 4
   2.1 Massive Open Online Courses 4
   2.2 Automated Education Through Video Lectures 5
   2.3 Automated Exercise Grading and Generation 6
   2.4 Intelligent Homework System 7

3 **Description** 7
   3.1 Shapes 8
   3.2 Language Independence 9
      3.2.1 Shapes - Independent 9
      3.2.2 Statements - Dependent 10
   3.3 Grader Interface 11
   3.4 Grid Examples 13
      3.4.1 Intro Grids 13
      3.4.2 Easy Grids 14
      3.4.3 Medium Grid 16
      3.4.4 Hard Grid - Star and Flags 17
      3.4.5 Expert Grid - Game Art 20

4 **Evaluation** 21

5 **Conclusions** 22
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sample grid for for-loop problems</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>Sample empty code blocks that students are required to complete</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>Sample instructions for creating a grid</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>A python for loop entry</td>
<td>11</td>
</tr>
<tr>
<td>5</td>
<td>A C-based for loop entry</td>
<td>11</td>
</tr>
<tr>
<td>6</td>
<td>Grid showing marks where student’s grid differs</td>
<td>12</td>
</tr>
<tr>
<td>7</td>
<td>A horizontal line</td>
<td>13</td>
</tr>
<tr>
<td>8</td>
<td>A vertical line</td>
<td>13</td>
</tr>
<tr>
<td>9</td>
<td>A diagonal-up line</td>
<td>14</td>
</tr>
<tr>
<td>10</td>
<td>A diagonal-down line</td>
<td>14</td>
</tr>
<tr>
<td>11</td>
<td>A rectangle</td>
<td>14</td>
</tr>
<tr>
<td>12</td>
<td>A right triangle facing left</td>
<td>15</td>
</tr>
<tr>
<td>13</td>
<td>A right triangle facing right</td>
<td>15</td>
</tr>
<tr>
<td>14</td>
<td>An upside-down right triangle facing left</td>
<td>15</td>
</tr>
<tr>
<td>15</td>
<td>An upside-down right triangle facing right</td>
<td>15</td>
</tr>
<tr>
<td>16</td>
<td>A rectangular outline</td>
<td>16</td>
</tr>
<tr>
<td>17</td>
<td>An isosceles triangle</td>
<td>16</td>
</tr>
<tr>
<td>18</td>
<td>An upside-down isosceles triangle</td>
<td>16</td>
</tr>
<tr>
<td>19</td>
<td>A trapezoid</td>
<td>17</td>
</tr>
<tr>
<td>20</td>
<td>An upside-down trapezoid</td>
<td>17</td>
</tr>
<tr>
<td>21</td>
<td>A six-pointed star</td>
<td>17</td>
</tr>
<tr>
<td>22</td>
<td>Greek flag design</td>
<td>18</td>
</tr>
<tr>
<td>23</td>
<td>Icelandic flag design</td>
<td>18</td>
</tr>
<tr>
<td>24</td>
<td>Bahamas flag design</td>
<td>19</td>
</tr>
<tr>
<td>25</td>
<td>Space invader enemy</td>
<td>20</td>
</tr>
<tr>
<td>26</td>
<td>A mario goomba enemy</td>
<td>20</td>
</tr>
<tr>
<td>27</td>
<td>A mario mushroom</td>
<td>20</td>
</tr>
<tr>
<td>28</td>
<td>Pacman ghost looking up</td>
<td>21</td>
</tr>
<tr>
<td>29</td>
<td>Pacman ghost looking down</td>
<td>21</td>
</tr>
<tr>
<td>30</td>
<td>Pacman ghost looking left</td>
<td>21</td>
</tr>
<tr>
<td>31</td>
<td>Pacman ghost looking right</td>
<td>21</td>
</tr>
</tbody>
</table>
1 Introduction

As technology improves, many aspects of society are becoming automated. A prime opportunity for this automation presents itself in teaching programming. Many programming courses follow an automated model, where lectures are pre-recorded so students can learn or review on their own schedule, or where exercises designed to reinforce material are automatically generated and graded. Due to their rather simplistic nature, pre-recorded lectures offer little room for improvement in terms of automated education. However, automatic exercise grading has progressed very little beyond small scripts diffing program outputs, and automatic exercise generation is almost nonexistent. This project aims to create a simple program that can generate and grade basic problems of varying difficulties. Specifically, this project will focus on exercises about for-loops, an introductory programming concept. The program will be integrated with the Intelligent Homework System[1] created by Dr. Clinton Staley, which will be talked about in more detail in Section 2.4.

2 Background

2.1 Massive Open Online Courses

Massive Open Online Courses, or MOOCs is an initiative to create free, online courses taught by accredited instructors, that a student can participate in to learn new skills, either for career development or just for fun. The primary use case for MOOCs in recent years has been as a university supplement or substitution. Many universities have been working towards offering MOOCs that can be taken for credit at that university [2] [4], allowing incoming students to get started on the degree early, or allowing people to try out a degree before committing fully to it [4]. California’s university system has taken advantage of MOOCs to offer credit in classes at other California University campuses [5]. This movement is intended to provide a solution to budget cuts to the state’s education funds by allowing a class to be taught at a single university, and allowing other universities to offer credit for taking that class online, thus eliminating the need for duplicate classes and teachers. Unfortunately, the project has not been utilized enough, and many students who start an online course fail to complete the course; studies have shown that as few as 6 percent of students will complete an online course offered at a university [5]. Additionally, there has been much debate on whether a MOOC can truly offer the same level of education as an in-person class, where a physical instructor is more capable of recognizing student difficulties and can tailor the subject content to the students’ needs [6]. Ultimately, if MOOCs can become successful in the university system, then higher education will become much more available to all people, regardless of their background and social status.
2.2 Automated Education Through Video Lectures

The simplest form of automated education is a pre-recorded video lecture on a topic. A prime example of this exists in MIT’s OpenCourseWare [7] which consists of video recordings from many of MIT’s classes over the past 17 years (beginning in 1999 [8]). OpenCourseWare (OCW for short) allows students to freely watch lectures on their own schedule, and most courses provide some form of assessment that allow the student to gauge their own progress on the topic. The assessments must be manually checked for correctness by the student against a set of supplied solutions. Additionally, not every set of assessments comes with a solution set [9]. Another automated education venture led by MIT is called edX [10]. EdX is similar to OCW in that it consists of sets of recorded lectures that allow a student to complete a course on his or her own schedule. However, edX is an improvement over OCW in a couple ways: edX features content from many different institutions and companies, whereas OCW is entirely composed of lectures from MIT, and edX has real support for automated grading of problems and quizzes given during the course. Instead of requiring the student to manually check against a given set of solutions, edX has a web interface for asking questions that also grades the student without necessarily giving away the answer if the student is wrong. This is very helpful for programming questions. EdX supplies an “External Grader” interface [11] that allows courses to ask for a programming solution, and then process and run the provided program as the student’s response to a problem. EdX is limited a bit in how the code graders only support python and MATLAB as programming languages.

Other prevalent examples of computer science education through video lectures can be found on websites like Udemy [12], Udacity [13], or Academic Earth [14]. These sites are similar to MIT’s OCW in format; they consist of a set of video lectures and optionally a set of review exercises and accompanying solutions. Of the three education sites presented here, Academic Earth is the most unique in that it operates as more of a hub for finding free online courses and will usually point the user to a different website for taking the course. Udacity and Udemy on the other hand will host video lectures on their own sites, and are more focused on building a brand that would allow their site to become synonymous with a university in terms of education quality. Udacity in particular is focused on creating what they term a “Nanodegree” [15]. This degree is earned by completing a set of courses created by high-profile tech companies, and is intended to be a replacement for a university degree. Udacity even offers a “Nanodegree Plus” [16] program that guarantees a job within six months of graduation. These types of programs differ from the previously mentioned OCW model in that they are not free, and they enforce a schedule on the student, instead of allowing the student control over the pacing of the course. However, all of these education programs are redefining education [17] through their use of an online course structure and automated video lectures.
2.3 Automated Exercise Grading and Generation

While there are many existing implementations of automated instruction, automated exercise generation and grading is a much more immature problem. Automatic exercise grading has many more examples, and is a problem that has been mostly solved, whereas automated problem generation is a much more open problem. In the context of computer science education, automated grading usually takes the form of compiling and running the student’s program, then running the program through some test cases, which may or may not be hidden from the student, and then grading the problem attempt based on the result of the test cases. The test cases can be as simple as checking that the program prints out an expected string, or can be as complicated as running a full suite of unit tests against the student’s program. An example of an online computer science related education with automatic exercises is Codecademy [18].

Codecademy specializes in teaching courses related to web development, and uses an automated education model where each topic is taught through a series of interactive mini-lessons, in which the student needs to complete some task, given a web-based terminal interface or text editor. Codecademy grades each exercises by running small test cases against whatever the user was expected to produce. For example, one of the lessons requires the student to create a new git repository from a terminal session, and the student is graded based on the output of a small ruby script that checks for a .git directory after any terminal command is run. Each exercise has a custom testing script that indicates if the user has passed the current exercise and can move on to the next problem. Codecademy doesn’t have automated exercise generation; a student can repeat a lesson exercise pair and receive the same problem and test every single time.

Another solution for automatic grading exists at Codeboard.io [19]. Codeboard.io acts solely as an online exercise grading platform, making it ideal for being paired with the automated instruction websites. In fact many edX courses use Codeboard.io as a platform for assessing the students in the online course. Unlike Codecademy, which has both instruction and assessment together, Codeboard.io operates as a repository for exercises and provides a web IDE interface that allows the students to attempt the problem from the web browser. It allows an instructor to upload a problem and accompanying test cases that will be run against the student’s solution for grading. The instructor also has the option to create public problems, or restrict problem access to a set of users, likely students in a class. Again, Codeboard.io has no form of question generation, and relies on manual question creation and uploading, and will provide the same problems to students every time they access the exercise.

Examples of online programming exercise generation are very scarce, and most are merely crowd-sourced. Codewars [20] is a web application where users submit problems, termed “kata”, and attempt to solve other users’ submitted kata. This acts as question generation for the site, and a user can attempt different problems as new exercises are created. The downside to this crowd-sourced approach is that questions won’t be testing exactly
the same concepts, and the user can’t repeatedly attempt a similar question to reinforce understanding of a concept. Additionally, Codewars isn’t associated with any instruction, and merely operates as an outlet for testing and reinforcing programming ability.

2.4 Intelligent Homework System

The Intelligent Homework System [1], or IHS for short, is a system that automatically generates and grades problems related to specific topics. IHS is owned and maintained by Dr. Clinton Staley [21] and is actively used in his classes taught at California Polytechnic State University, as well as in online courses he teaches on his own [22]. The intent of IHS is to repeatedly provide students with similar, yet different problems related to a specific programming topic, allowing a student to attempt a topic multiple times, and ensuring that simple memorization of a solution doesn’t guarantee success.

IHS works with a series of grader modules, where each grader is responsible for creating and grading problems about a certain topic. A grader module is a java implementation of a provided grader interface, that will then be installed into the system and be ready for use. The student interacts with IHS through a web interface that will prompt the student to input their answer to a question, which may be a simple multiple choice question, or may require the student to write code in a certain language, which IHS will compile and evaluate.

3 Description

In this project, I implemented an IHS grader module about for-loops targeted at entry-level computer science students. The grader allows students to choose between five different difficulties, Intro, Easy, Medium, Hard, and Expert. Each difficulty will present the student with a number of problems where each problem is composed of a two-dimensional grid with certain cells colored to form a pattern, shown in Figure 1.

Students are presented with a partially completed code block, shown in Figure 2 where they must complete statements in a for-loop so each loop will correctly color a blank grid in order to match the provided sample grid.

The student will have 3 attempts to complete each problem, and if they fail any problem in the difficulty, they fail the entire difficulty level.
3.1 Shapes

The core object used in creating each grid is a shape; a grid is created by coloring an ordered list of shapes on top of one another. A shape in this project is defined as an object which knows how to: create a description of itself, this is used when providing the student with written instructions on how the sample grid was created, shown in Figure 3 color itself on an input grid using an input color, which is used to create the sample grid for each problem; create the code needed to draw itself on a grid in all supported programming languages, which is used to both create the initial blank code UI for the student as well as respond to a problem; and finally a shape knows how to create a bounding box of itself, allowing clients of the shape to request how large the shape is. All shapes are created with a random size and a random color when the problem is created. This is achieved through another interface which is used to create the shapes, given the width and height of the grid as well as a java Random object. This interface is informally referred to as the shapes creator interface.
A shapes creator is defined as an object that can create an ordered list of shapes, which will be used for coloring a grid. Each shape usually defines its own creator object that can create an instance of itself when requested. While this is not required by the interface of a shape, it is usually convenient to provide this. For complex grids, (see Section 3.4.4) a custom shapes creator object is needed, which will create several different shapes that will be colored different colors. For simple grids a single shape’s creator is usually enough.

For coloring shapes on a grid, each shape is given a single color used for coloring itself. This presents an issue when a grid design requires multiple shapes of the same color, as is the case for Hard and Expert difficulty grids (Section 3.4.4). For cases like this, a special implementation of the shape interface exists called a CompoundShape. A CompoundShape is a shape that is composed of a list of “sub-shapes” which will all be colored the same color. In this implementation, all shape responsibilities are delegated to the children shapes, which allows us to treat the collection of similarly-colored shapes as a single shape for the purposes of coloring, creating a description, and responding to a problem.

3.2 Language Independence

One of the challenges in this project was ensuring that the problems could be created and responded to in different programming languages. In order to solve this, there needed to be a clear separation between language dependent objects and language independent objects when creating a grid and associated problem.

3.2.1 Shapes - Independent

The shape objects that make up the grid are intended to be language independent; coloring the grid does not require any specific language knowledge, and each shape can create language specific questions and responses for each language that the grader supports. This is accomplished through a derivative of the visitor pattern, where the shape can be used...
as a visitor object and the language object can accept the shape and request the language-specific information from the shape. Once language-specific information is needed, such as code for both creating and answering the question, (the problem code is actually used for both use cases, and will be discussed in Section 3.2.2) any future information must be language-specific as well, thus crossing the information separation boundary.

3.2.2 Statements - Dependent

Information that is language dependent is represented by a statement interface. A statement is defined by its ability to add its information to a code entry UI element; answer the UI element it created, either correctly or incorrectly depending on what is requested; and add any variable declarations needed to a variable declaration object (which is essentially a specialized Map object). Adding variable declarations will be an empty operation for languages that do not require pre-declarations of variables. Because a statement is required to create and answer the UI for a problem, a shape only needs to transform itself into a list of statements depending on the language, in order to switch from language independent information to language dependent information. For the purposes of this project, only a few statement implementations were necessary. These include a method call statement for coloring the grid (this statement creates fixed UI elements, and does not need to do anything to respond to the UI or add declarations), an assignment list statement (representing assigning a list of variables to a list of values, this statement also does not perform any response to the UI), a comment statement (a bit of a misnomer, but represents writing a comment in a language, and also does not perform any response), and finally a for statement.

The for statement is the most complex implementation of a statement, and is the only statement used in this project that provides an opportunity for student input in the UI, as well as being the only statement which adds a response to the UI. It comes in two forms, depending on the language. For pythonic languages, the for statement is represented by the beginning and end expressions of a range expression (the beginning and end expressions are typed as simple strings). This statement then creates a UI element containing the for keyword, any characters needed after the range expression, any statements nested in the for loop, and an entry where the student must create the range expression so the nested statements will color the grid correctly (see Figure 4).

When responding to the UI, the python for loop creates the range expression and fills in the entry with the expression. A bad response by the python for statement creates a loop which iterates over an empty list so the output grid is blank (see Section 3.3 for more detail).

The other type of for statement is a C-based for statement, used by languages that are C-based or are syntactically similar to C. This implementation contains three main properties that model the three properties of a C for statement: an assignment list statement used for initializing variables in the loop (talked about earlier in this section), a boolean expression which is the condition for when the loop run, and an update expression that is used to update
variables at the end of the each iteration. This statement creates a UI containing the for keywor
keyword and necessary parentheses and brackets, any statements included in the body of
the for statement, and an entry for the student to add initializations, loop conditions, and
variable updates (see Figure 5).

*/ Background Color / Outer Stripes */
for (y = 9; y < 28; ++y) {
    for (x = 3; x < 41; ++x) {
        colorGrid(x, y, 'M');
    }
}

Figure 5: A C-based for loop entry

When responding to the UI, the C-based for loop combines its initializations, condition,
and update properties into a syntactically-correct string and populates the for statement
entry. When responding incorrectly, C-based for statements create correct initializations
and updates, but use an always-false condition (see Section 3.3 for more detail).

3.3 Grader Interface

IHS provides a grader interface that allows third-party developers to contribute to the
overall system. The for-loop grader developed for this project was written to this grader
interface. The grader interface essentially requires definitions for creating the UI and state
for a problem; grading a problem attempt, updating the UI as necessary; and responding to
a problem, both correctly and incorrectly depending on which response is asked for. Each
difficulty level has a collection of shapes creator objects (see Section 3.1), and when asked to create a problem, the grader creates a random order of the difficulty’s creator objects, and sets up the problem state, consisting of the problem ordering, count of attempts remaining, and information about how to create the current grid. The initial problem contains the full list of remaining shapes creators, in a pre-randomized order.

When grading a problem, the grader creates the full program based on the student’s input and some hidden surrounding code, then runs the program and compares the produced grid with the provided sample grid. If the grids match, then the next question is pulled from the current question’s state and that question is created. This is repeated until there are no more questions remaining in the question’s state, upon which a final grade of full credit is recorded. If the grids do not match, then the student’s remaining attempts count is decremented, the problem is recreated, and the sample grid is marked with ‘X’s where the student’s grid differed. This can occur either where the student has colored a square incorrectly, or where a student has not colored a square that was expected to be colored, shown in figure 6. If the student’s grid differs and the student has no remaining attempts, then they have effectively failed the problem and receive a zero for the problem.

![Figure 6: Grid showing marks where student’s grid differs](image)

In order for the grader to respond to the current problem, each problem’s state is created with information about how to correctly answer the problem, as well as information about incorrectly answering the problem. In order to incorrectly answer the problem, while still producing syntactically correct code, a “bad” responder will populate all for statements so they do not run the body of the loop. For python and ruby, this produces for loops like:

```python
for x in []:
    # color grid
```

while for languages like java and C++, this produces loops that look like:
for (int x = 0; false; ++x) {
    // color grid
}

Java actually requires a small trick to generate no-op code; because the compiler will throw an error if it detects unreachable code, the created loop statement can not use a false constant, and must use a false variable that is hidden in the grader-supplied surrounding code. The question state is used to correctly answer the problem, as it is created with the information about how to write the correct code for the problem.

### 3.4 Grid Examples

The following figures display examples of the various shapes and designs used for creating the grids which are used in the various problem difficulties. Each grouping of figures indicates possible grids for a problem, while all the groupings in a difficulty section indicate all the problems a student will have to answer. (e.g. In Section 3.4.1 a student will have to answer two problems, one problem on right-angle lines, and one on diagonal lines. A right-angle line problem can either be a horizontal or a vertical line, and a diagonal line problem can either be a diagonal-up line or a diagonal-down line.)

#### 3.4.1 Intro Grids

![Figure 7: A horizontal line](image)

Figure 7: A horizontal line

![Figure 8: A vertical line](image)

Figure 8: A vertical line

Right angle lines
3.4.2 Easy Grids
Right triangles

Figure 12: A right triangle facing left
Figure 13: A right triangle facing right
Figure 14: An upside-down right triangle facing left
Figure 15: An upside-down right triangle facing right
3.4.3 Medium Grid

Figure 16: A rectangular outline

Figure 17: An isosceles triangle
Figure 18: An upside-down isosceles triangle

Isosceles Triangles
3.4.4 Hard Grid - Star and Flags

Figure 19: A trapezoid

Figure 20: An upside-down trapezoid

Figure 21: A six-pointed star
Figure 22: Greek flag design

Figure 23: Icelandic flag design
Figure 24: Bahamas flag design
3.4.5 Expert Grid - Game Art

Figure 25: Space invader enemy

Figure 26: A mario goomba enemy

Figure 27: A mario mushroom enemy

Mario Sprites
4 Evaluation

In order to evaluate the success of this project, some metrics for success must be established. The first measurement of success is a very simple one; did the project get completed and does it work as intended. In this regard, the project was definitely successful. The project was ultimately completed and it functions as expected, successfully creating and grading for-loop related problems. A more complicated and subjective metric of success has to do with the grid designs created by the grader. One of the stipulations of this project was that it produce “interesting” grid designs, that would excite students and wouldn’t feel like busy work for the student. This metric is harder to evaluate, as there hasn’t been an opportunity to test and receive feedback from entry-level students regarding the problem and grid design choices. However, as a self-proclaimed advanced computer science student, I feel that most of the grid designs are “interesting”, especially the more difficult designs. I had
to find a compromise between introducing easy problems that may not be very interesting and creating challenging yet interesting designs. I also consulted a Graphic Communication student for feedback on how to create interesting designs, which I feel helped set the project up for success against the “interesting” design grid metric. The largest “failure” aspect of this project would be the timeline. I severely underestimated the amount of time this project would need originally, and the project took much longer than my original thoughts. Ultimately, the true metric of success for this project will be how effective it is at reinforcing introductory education of for loops. However, that information won’t be available until this grader has been used in an introductory programming course and students give feedback. Until that happens, I would declare this project a success.

5 Conclusions

This project was a great experience, especially for gaining familiarity working on a non-trivial project. I had multiple points during development when I needed to stop working and refactor the code I had already written. Whenever that happened, it would initially be frustrating, however the completed refactor was always very rewarding. As far as future work, I think that once students have used and provided feedback on the grader, there will probably be many suggestions and improvements to be done, however I don’t foresee these changes happening in the near future. This was also my first experience working on an education tool, so I would not be surprised if my grader implementation needs to be tweaked to reinforce certain concepts better. Ultimately, this was a great project and, while there were certainly a fair share of hurdles, the end result was a very rewarding program.
References

[1] https://softwareinventions.com/IHS


[10] https://www.edx.org/
[12] https://www.udemy.com
[18] https://www.codecademy.com
[19] https://codeboard.io/
[22] https://softwareinventions.com/EdSite/