doi: 10.14355/jitae.2014.0302.01

# Evaluating the Effectiveness of a 3D Visualization Environment While Learning Object Oriented Programming

Arwa A. Allinjawi*[1], Hana A. Al-Nuaim[2], Paul Krause[3]

[1, 2] Computer Science Department, Computing and Information Technology College, King Abdulaziz University, P.O.Box 80200, Jeddah 21589, Kingdom of Saudi Arabia.
[3] Department of Computing, University of Surrey, Guildford, Surrey, GU2 7XH, United Kingdom.

*[1]aallinjawi@kau.edu.sa; [2]hnuaim@kau.edu.sa; [3]p.krause@surrey.ac.uk

Received 6 October 2013; Revised 29 January 2014; Accepted 27 February 2014; Published 27 May 2014

***Abstract***

Students often face difficulties while learning Object Oriented Programming (OOP) concepts. Many researchers have proposed different approaches to improve the teaching and learning of OOP concepts. One possible method is to engage the students with stimulating 3D visualization environments to reduce the complexity while enhancing understanding of concepts. The visualization environments may improve programmer productivity and the OOP learning outcomes. However, still many researches' conclusions were based on subjective assessments, where Computer Science (CS) lacks standard assessment methods for educators to measure their students' learning outcomes. In this context, the purpose of this research is to illustrate a demonstration experiment using the Achievement Degree Analysis (ADA) approach to statistically evaluate the effectiveness of the visualization environment—ALICE—that is hypothesized to improve novice programmers' understanding of OOP concepts at King Abdulaziz University's (KAU), CS department female section, in Saudi Arabia. We focused on a specific intervention in OOP, but the experimental method could be applicable in a range of domains.

***Keywords***

*Object Oriented Programming; Achievement Degree Analysis Approach; ALICE Visualization Tool*

## Introduction

Programming is a fundamental skill that all Computer Science students are required to learn. Programming requires the correct understanding of abstract concepts, and the theory of Object Oriented Programming (OOP) is based on a representation of the real world, employing abstraction principles and basic abstract operations (Oliveira, Conte and Riso, 1998). However, abstraction is a very complex concept to master; therefore, many students find programming, and especially OOP, difficult to learn and thus hard to maintain an interest in. This can lead to high rates of failure or dropout (Esteves, Fonseca, Morgado and Martins, 2011; Jenkins, 2002; O'Kelly and Gibson, 2006). As a result, lecturers of introductory programming courses are faced with the challenge of helping students learn to program and understand how their program works (Dann, Cooper and Pausch, 2000). At the same time, they must try to make programming fun by creating activities that can help students enjoy the process of learning. One possible method to make lectures on programming more effective is to have a stimulating visualization integrated program development environment to support innovative instructional methods for teaching novice programmers (Dann, et al., 2003), help the students to fully engage with the environment, encourage them to practice more (Ragonis and Ben-Ari, 2005), reduce the complexity while enhancing understanding of concepts, and make the invisible visible (Chuda, 2007).

Due to programming difficulties, many researchers have presented subjective assessments for diagnosing learning problems to improve the teaching of programming in Computer Science higher education. Moreover, many researchers have proposed different approaches, such as using visualization tools and engaging students with these tools, to improve the teaching and learning of programming concepts (Lahtinen, Ala-Mutka and Järvinen, 2005).

Rowe and Thorburn (2000) developed Vince, a web-based tool to help students understand the execution of C programs. The tool visualizes the workings of a C program step by step, showing a positive effect on students' learning, and it was considered an effective supplement for an introductory programming course. Dann, Cooper, and Pausch (2001) described an approach to introducing recursion by using ALICE, the visualization tool, as part of a course at Ithaca College in New York for programmers with no previous programming experience. Dann et al. concluded that using ALICE offers computer science instructors an approach to introducing fundamental concepts to novice programmers that allows them to quickly identify and learn from mistakes. Rodger (2002) developed a course, CPS49S Animation and Virtual Worlds for non-CS majors, at Duke University to teach students CS concepts and programming through simple animation and 2D and 3D virtual worlds. The course consisted of five main units: HTML; a scripting language (JAWAA); a programming modeling environment (StarLogo); an interactive programming environment (ALICE); and a simple programming language (Karel++). The researcher concluded that ALICE was the most popular tool. Boyle (2003) developed materials for visualizing certain programming concepts and describes them as pedagogically rich compound learning objects. Each object is designed towards a certain learning goal. The approach combines together several features visualizing the execution of, e.g., a "while" or a "loop", and giving students questions and tasks relating to the concept. The benefit of this approach is that these kinds of objects can be easily adapted for use and incorporated into education. Cooper, Dann, and Pausch (2003a) presented ALICE for an objects-first strategy, which emphasizes the principles of OOP and design from the beginning of the teaching process (CC2001, 2001). They concluded that ALICE was quite useful in teaching objects-first strategy to help students master the complexities of OOP, it provided stronger object visualization, and a flexible, meaningful context for helping students to see object-oriented concepts. Yadin (2011) believed that using visual environments may help the students to understand some of the abstract concepts related to programming and problem solving and decrease the dropout rates. He used Guido Van Robot (GVR), a Python-based implementation of Karel. GVR showed successful results, and the number of failing students was reduced.

Although previous researchers and others have proposed visualization environments to improve the teaching of programming, the statistically valid evaluation of an intervention that actually does resolve a specified learning difficulty is still limited. Therefore, the objective of this research study is to perform the achievement degree analysis (ADA) approach (Allinjawi, Al-Nuaim, & Krause, 2014), which assesses the students' learning outcome of specific concepts with the OOP behavior, on a group of female students studying OOP at King AbdulAziz University in Saudi Arabia. The study engaged them with the ALICE visual tool training to assess whether it made a significant difference in the level of understanding across individual OOP concepts.

Many studies used ALICE to prepare students for introductory CS, especially for those who are considered at-risk of failure (Cooper, et al., 2003a; Dann, et al., 2003; Moskal, Lurie and Cooper, 2004), those who want to teach an object-first approach (Cooper, et al., 2003a), those who want to teach particular concepts such as recursion (Dann, et al., 2001), or those who want to help the instructor to teach the basics of the object-oriented paradigm (Aktunc, 2013; Cooper, et al., 2003a).

## ALICE Programming Environment

ALICE is a 3D interactive graphics programming environment built by the Stage 3 Research Group at Carnegie Mellon University under the direction of Randy Pausch (Pausch, et al., 1995).

The goal of ALICE is to introduce and easily explain OOP concepts (for example, a fundamental introduction to objects, methods, decision statements, loops, recursion, and inheritance) to novices by developing motivating 3D environments, such as creating animation for telling a story, playing an interactive game, or creating a video to share on the web. In ALICE, students can see their programs in action, understand the relationship between the programming statements and the behavior of objects, and then gain experience with all the programming constructs typically taught in an introductory programming course (Cooper, Dann and Pausch, 2003b; Dann, et al., 2000; Dann, Cooper and Pausch, 2005).

The program code the students create while using ALICE corresponds to standard OOP language statements, such as Java, C++, and C#. Fig 1 illustrates the five regions in the ALICE colorful interface that students engaged with to create a program that generates a 3D animation scene. These regions are:

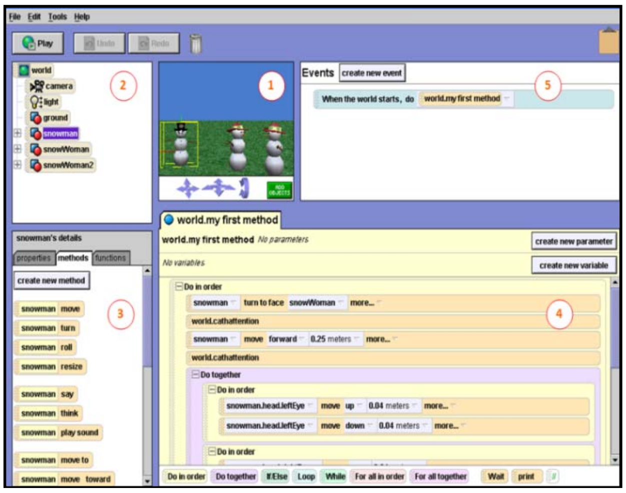


FIG 1. ALICE INTERFACE

1. *Scene Window:* allows the students to add objects to the scene by dragging them from a web or local-based object gallery. ALICE has many web and local galleries that allow students to create and manipulate interesting 3D objects (such as an ice skater from People gallery, or a frog from the Animals gallery, as shown in Fig 2), and create their virtual world based on these predefined objects. In addition, the scene window also allows students to position and resize objects to create an initial state for their virtual worlds.

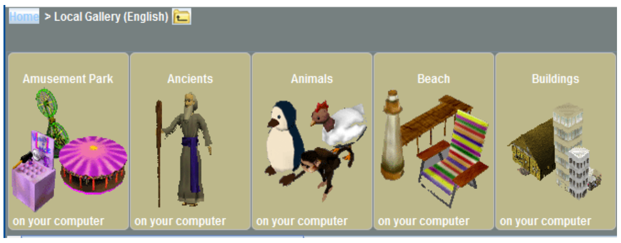


FIG 2. ALICE LOCAL GALLERY 3D OBJECTS

2. *Object Tree:* provides a hierarchical list of the representing objects and their subparts, as shown in Fig 3.

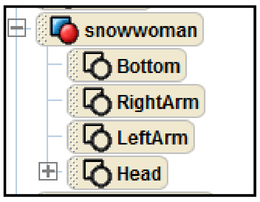


FIG 3. ALICE OBJECT'S SUBPARTS

3. *Object Details Area:* provides additional information about the object, such as the object's properties (color, opacity), the methods that the object can perform (move, turn), and the questions (functions) the object can ask about the state of the world (distance to, width). Students can change the properties of their objects, use the available object method (such as skate and spin as shown in Fig 4) to create highly visual exciting animations, or build a new object method from scratch.

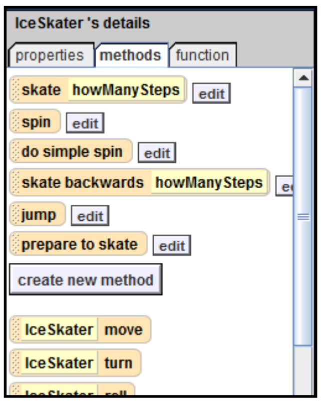


FIG 4. ALICE'S METHOD EDITING INTERFACE

4. *Editor Area:* allows students to create, view, and edit their code. Students can drag methods and functions from the object details area and program control structures statements (If/Else, While, DoTogether, etc.) from the bottom line of the editor area to implement the code, as shown in Fig 5. Students can create their programs by dragging and dropping the statement into the main window, "world.my first method" as in Fig 5, instead of having to write code. The drag and drop mechanism eliminates the frustration of the syntax errors that are known to discourage new programmers (Nguyen, Nandi and Warburton, 2011) and allows the students to focus on the concepts (Daly, 2011).

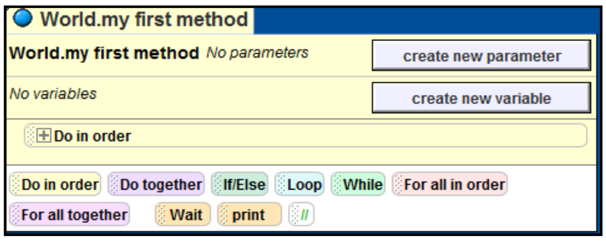

World.my first method
World.my first method No parameters
create new parameter
No variables
create new variable
Do in order
Do in order Do together If/Else Loop While For all in order
For all together Wait print //

FIG 5. ALICE EDITOR AREA

5. *Behaviors Area:* provides a mechanism to allow students to interact with ALICE virtual world using keyboard and mouse inputs as in Fig 6. The animations where characters play out a scene or a virtual world where objects respond to mouse and keyboard inputs are more attractive than text-based programs (Aktunc, 2013).

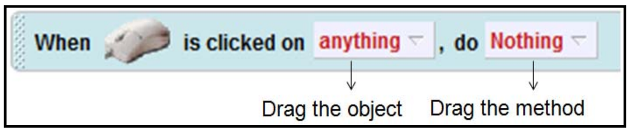


FIG 6. ALICE EVENT CREATION

The visual animated output of the program can be easily accessed and students can watch his/her animation anytime by using the play button (see Fig 7). Therefore, many studies use ALICE to motivate students and encourage them to program.

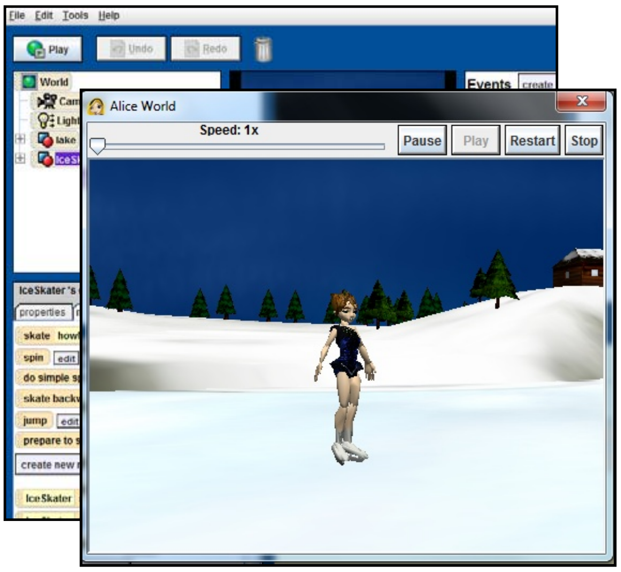


FIG 7. ALICE OUTPUT INTERFACE

Although ALICE has had many positive reviews within the literature, it has received a number of criticisms and some drawbacks of using it while learning programming (Cliburn, 2008; McKenzie, 2009; Nguyen, et al., 2011). ALICE, with its bugs, installation burdens, frequent crashes, and inconsistent behavior or messages could cause frustration and dissatisfaction (Nguyen, et al., 2011).

## The Methodology

This study would enable us to propose a method to statistically evaluate the effectiveness of ALICE as a given intervention to attain useful and deeper statistical results of whether ALICE is better at assisting students in some OOP concepts over other OOP concepts.

### *Objectives and Hypothesis*

The experiment was to assess the students' understanding of particular OOP concepts before and after use of ALICE, using the ADA approach. The formal question statement of this research is:

> Does engaging the students with a new visualization tool (ALICE) improve their OOP learning outcomes?

To formulate the hypothesis that we wanted to test in our experiment, the null hypothesis is:

> There will be no statistically significant differences in the students' improvements of learning skills in OOP between the experimental group who are engaged with the ALICE tool and the control group who are not engaged with the ALICE tool.

### *The Sample Description*

The cohort female students were registered for a traditional OOP class (CPCS 203) in the spring semester. The 128 students who were willing to be part of the experimental study were divided into two groups: the experimental group and the control group. Both groups were selected from a population who had studied a basic math course (Math 110) with a mean value of their scores across this course of 88.18% and standard deviation of 9.39. In addition, the OOP course was their second programming course, as the students had already passed a basic programming course (CPCS 202) with a mean value of their scores of 82.11% and standard deviation of 9.52. Only 12.5% of the population had animations or visualization software experiences, such as Flash, which were not required in the department curriculum.

The strategy to avoid a biased judgment across the sample of the control and experimental group was to select completely randomly from the population, and to show that each group was representative of the population (Fisher and Foreit, 2002). Consequently, students in the experimental or control groups were selected randomly from the students who were interested in attending the ALICE workshop. The number of students in each group was restricted by the capacity of the labs where the new intervention would take place. Twenty-nine students in the experimental group had, in addition to the traditional classes, been exposed to the visualization tool ALICE, and in contrast, 28 students in the control group had taken only traditional classes.

### *The Assessment Instrument*

Direct assessment is the key element to demonstrate the level of achievement within a course (Al-Bahi, 2007). The grades obtained by the students in the course's exams would show the levels of achievement of the course learning objectives. To help determine content mastery where a visualization tool is used, pre-testing and post-testing are particularly effective. Pre-test and post-test designs are widely used in behavioral research, primarily for the purpose of comparing groups, measuring the resulting change, and/or determining effects resulting from integrating new interventions (Dimitrov and Rumrill, 2003).

The pre-test and post-test are considered as the course exam, and although it was not possible to use identical exams for both tests, the name of the "classes" and "methods" and the sequence of the questions were modified in the post-test. Other than that, both tests were similar in that they measured the same concepts with the same level of complexity for each question.

The pre-test and post-test consisted of five questions that examined the students' basic logical thinking skills in most OOP concepts with different weights, as shown in pre/post-Tests Item Relationship Table (TIRT) in Table 1. Following the TIRT in (Allinjawi, Krause, Tang and Al-Nuaim, 2011), each entry in the TIRT represents the degree of association between question $Q_n$ and concept $C_i$, which was calculated following a specific procedure described next. First, each statement "code" of the model answer of each question was related to specific concepts with a specific mark, depending on the question's total mark. It could be that one code is related to two or more concepts, which means the code's mark must be assigned to both concepts. Second, the total mark of each concept in $Q_n$ was normalized to be within a unified range from zero to 1. Zero indicates that the concept is not associated with the question, and 1 indicates full association. Since each question has a different mark, and our concern is the weight of each concept within the question rather than the weight of a question within the exam, we had to perform the normalization. The weight of each concept in the TIRT was generated based on the distribution of marks and the concepts to be assessed with weight of association. The last row in Table 1 represents the total association of each concept within the exam.

The *abstract* concept was not included in the pre-test and post-test because, at that time, students had not yet been exposed to the concept.

### *The Procedures*

The pre-test was held before the ALICE workshop, and the post-test was held after the workshop. In both tests, there was no time limit during the exams, because we were not assessing time management while answering the questions. The time set for the exams was not a hindrance for the students to answer the questions. In addition, the tests were open book because we were assessing problem solving skills, not memorizing skills. Open book tests are gaining popularity due to large initial benefits, as reducing the students' level of anxiety compared to close book tests (Agarwal, 2008; Gharib, Phillips, & Mathew, 2012; Theophilides & Koutselini, 2000). In examining problem solving skills and logical thinking, an open book test could be more appropriate for such an assessment test (Loi & Teo, 1999).

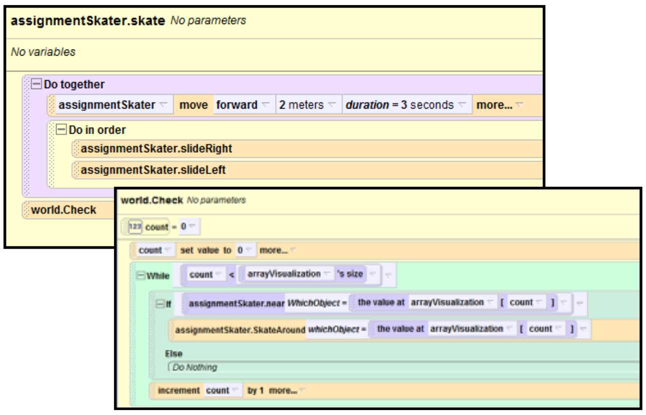
The ALICE workshop was conducted for eight 1.5-hour sessions and ran for three weeks. The workshop material was provided by Associate Professor Steven Cooper, one of the ALICE team members (Dann, et al., 2005). The availability of good support materials should lead to increased instructor satisfaction, which consequently would lead to more widespread use of visualization (Naps, et al., 2003).

We provided an example of one of the assignments the students had in the workshop to illustrate how students implement a program within the ALICE environment. For instance, students had to create a script of a skater, where "assignmentSkater" skates if the user presses the space button. This "assignmentSkater" must be imported from the original skater object they created to learn how to inherit and reuse available methods and learn how to add new features for this new imported object. While

TABLE 1 PRE/POST-TESTS ITEM-CONCEPTS RELATIONSHIPS TABLE (TIRT)

| | TIRT($Q_n$,$C_i$) | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Items | Variables | Methods | Class | Overload | Encapsulation | Constructors | Object | Inheritance | Aggregation | Dependency | Override | Polymorphism |
| | C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 | C11 | C12 |
| Q1 | 0.075 | 0.600 | 0.075 | 0.050 | 0.175 | 0.200 | 0.375 | 0.000 | 0.050 | 0.600 | 0.000 | 0.000 |
| Q2 | 0.000 | 0.292 | 0.000 | 0.000 | 0.000 | 0.333 | 0.000 | 1.000 | 0.000 | 0.000 | 0.250 | 0.000 |
| Q3 | 0.125 | 0.667 | 0.000 | 0.000 | 0.333 | 0.000 | 0.500 | 0.375 | 0.583 | 0.333 | 0.000 | 0.000 |
| Q4 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.667 | 0.000 | 0.000 | 0.333 | 1.000 |
| Q5 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.333 | 0.000 | 0.000 | 0.333 | 0.333 |
| Total_$C_i$ | 0.200 | 1.559 | 0.075 | 0.050 | 0.508 | 0.533 | 0.875 | 2.375 | 0.633 | 0.933 | 0.916 | 1.333 |

the "assignmentSkater" is skating, the students had to check if the "assignmentSkater" is close to any of the cones. If so, the "assignmentSkater" must skate around the closest cone. The students had to create the cones and combine them into one array, "arrayVisualization," to access each cone easily. Fig 8 illustrates the code of the "skate" and "check" method.



```
assignmentSkater.skate No parameters
No variables
⊟Do together
    assignmentSkater   move forward   2 meters   duration = 3 seconds   more...
    ⊟Do in order
        assignmentSkater.slideRight
        assignmentSkater.slideLeft
world.Check
```

```
world.Check No parameters
123 count = 0
count   set value to 0   more...
⊟While   count < arrayVisualization 's size
    ⊟If   assignmentSkater.near WhichObject = the value at arrayVisualization [ count ]
        assignmentSkater.SkateAround whichObject = the value at arrayVisualization [ count ]
    Else
        Do Nothing
    increment count by 1 more...
```

FIG 8. THE "SKATE" AND "CHECK" METHOD IN ALICE INTERFACE

To create a reliable scene and perform the required script, the students had to create and use functions, variables, control statements, methods, and other ALICE features.

Our goal in the workshop was to engage the students (the experimental group) with the ALICE environment, not teach them the concepts. This strategy was taken to avoid any confounding variable; any extra OOP concepts' illustration in the workshop could be the factor for their improvement not the ALICE environment itself.

## The ADA Approach and Results

The ADA approach (Allinjawi, Al-Nuaim, & Krause, 2014), is an incorporation of two available assessment approaches, the Hi-class approach (Al-Bahi, 2007) and the concept affect propagation approach (Chu, Hwang, Tseng and Hwang, 2006), with some modification to suit the OOP environment. The aim of ADA is to progressively diagnose the students' understanding within the context of an OOP course. The method will diagnose which particular OOP concepts are perceived as being particularly difficult to learn for each individual student.

The ADA is based on the concepts' association within each question and the students' answer grades of each question, which could be strongly subjective and based on the researcher's assessment judgment. In order to validate this approach, the concepts were reviewed and its association weights have been tested by a panel of experts to check if the concept weights were significantly correlated across the experts' content agreement and to provide content validity evidence. The Pearson's test showed a significant correlation ($p\text{-value} < 0.05$) between the concept association weights and those of the experts. In addition, the students' answer grades must be consistent with the experts' (raters) students' grading decisions. Thus, we asked the raters to correct the students answer sheets following the same model answer on each question. The internal consistency between the proposed rating decision and the raters' decisions was measured using Cronbach's alpha. The result showed that the alpha value of each question was >= 0.9; indicating that students' grading scores are consistent and agreed upon across the raters.

To perform the ADA approach within this type of experiment, first we had to represent the normalized students' gain scores in an answer grades table (AGT) (see Table 2). The $EX_K$ refers to a student from the experimental group, and $CO_K$ refers to a student from the control group, where in general the $S_K$ variable refers to a student from the sample. The gain score is the difference between the post-test and the pre-test scores. Using gain scores is one of the classical approaches of measuring of change with pre-test and post-test data (Dimitrov and Rumrill, 2003). The AGT shows that each entry, AGT ($S_k$, $Q_n$), is a gain value ranging from zero to ±1 for each student on each question that has been normalized using the following equation:

$$AGT(S_k, Q_n) = (\text{Post-test grade}(S_k, Q_n) - \text{Pre-test grade}(S_k, Q_n)) / \text{Total mark of } Q_n \quad (1)$$

TABLE 2 SAMPLE OF EXPERIMENTAL AND CONTROL GROUPS' NORMALIZED ANSWER GRADES TABLE (AGT) OF THE GAIN SCORES (N_EXP=29; N_CON=28)

| Exp. | AGT ( $EX_k$,$Q_n$) | | | | |
|---|---|---|---|---|---|
| Group | Q1 | Q2 | Q3 | Q4 | Q5 |
| EX1 | 0.075 | -0.125 | 0.042 | 0.000 | 0.333 |
| EX2 | 0.050 | 0.000 | 0.250 | 0.167 | 0.000 |
| EX3 | 0.150 | 0.000 | 0.167 | 1.000 | -0.333 |
| EX4 | -0.050 | 0.042 | -0.125 | 0.000 | 0.333 |
| EX5 | 0.125 | 0.125 | 0.000 | 1.000 | 0.000 |
| . | . | . | . | . | . |
| Con. | AGT ( $CO_k$,$Q_n$) | | | | |
| Group | Q1 | Q2 | Q3 | Q4 | Q5 |
| CO1 | 0.100 | 0.042 | 0.333 | 0.000 | 0.667 |
| CO2 | -0.025 | -0.083 | 0.125 | 0.000 | 0.333 |
| CO3 | 0.225 | 0.125 | 0.042 | 0.083 | 0.333 |
| CO4 | 0.100 | -0.083 | 0.083 | 0.000 | 0.333 |
| CO5 | 0.025 | 0.000 | 0.042 | 0.917 | 0.000 |
| . | . | . | . | . | . |

Then, we calculated the Achievement Degree Table (ADT) for every student with each concept, as shown in Table 3. The ADT has been calculated from the matrix of students' AGT ($S_k$, $Q_n$), in Table 2, and the TIRT ($Q_n$, $C_i$), in Table 1, following the equation below. $Q_{max}$ represents the total number of questions in the exam:

$$\text{ADT}(S_k, C_i) = \left(\sum_{n=0}^{Q\max} \text{AGT}(S_k, Q_n) * \text{TIRT}(Q_n, C_i)\right) / \text{Total\_ } C_i \qquad (2)$$

TABLE 3 SAMPLE OF EXPERIMENTAL AND CONTROL GROUPS' ACHIVEMENT DEGREE TABLE (ADT) ACROSS EACH CONCEPT (N_EXP=29; N_CON=28)

| | ADT($S_k$,$C_i$) Gain | | | | | |
|---|---|---|---|---|---|---|
| | Variables | Methods | Class | Overload | Encapsulation | . |
| Exp. Group | C1 | C2 | C3 | C4 | C5 | . |
| EX1 | 0.054 | 0.023 | 0.075 | 0.075 | 0.053 | . |
| EX2 | 0.175 | 0.126 | 0.050 | 0.050 | 0.181 | . |
| EX3 | 0.160 | 0.129 | 0.150 | 0.150 | 0.161 | . |
| EX4 | -0.097 | -0.065 | -0.050 | -0.050 | -0.099 | . |
| EX5 | 0.047 | 0.072 | 0.125 | 0.125 | 0.043 | . |
| . | . | . | . | . | . | . |
| Con. | ADT($S_k$,$C_i$) Gain | | | | | |
| Group | C1 | C2 | C3 | C4 | C5 | . |
| CO1 | 0.246 | 0.189 | 0.100 | 0.100 | 0.253 | . |
| CO2 | 0.069 | 0.028 | -0.025 | -0.025 | 0.073 | . |
| CO3 | 0.110 | 0.128 | 0.225 | 0.225 | 0.105 | . |
| CO4 | 0.090 | 0.059 | 0.100 | 0.100 | 0.089 | . |
| CO5 | 0.035 | 0.027 | 0.025 | 0.025 | 0.036 | . |
| . | . | . | . | . | . | . |

The following example illustrates the production of an achievement degree ADT ($S_k$, $C_i$) of a particular student, such as EX1, followed by equation 2:

$\text{ADT}(EX_1,C_1) = (\text{AGT}(EX_1,Q_1)*\text{TIRT}(Q_1,C_1) + \text{AGT}(EX_1,Q_2)*\text{TIRT}(Q_2,C_1) + \text{AGT}(EX_1,Q_3)*\text{TIRT}(Q_3,C_1) + \text{AGT}(EX_1,Q_4)*\text{TIRT}(Q_4,C_1) + \text{AGT}(EX_1,Q_5)* \text{TIRT}(Q_5,C_1)) / \text{Total\_ } C_1;$ Giving:

$\text{ADT}(EX_1,C_1)=((0.075*0.075)+(-0.125*0.000)+(0.042*0.125) +(0.000*0.000)+(0.333*0.000))/0.200= 0.054$

Finally, we compared the experimental and control's mean of achievement degree across each concept, as in Table 4. An Independent sample t test assuming equal variance was conducted on the concepts that follow the normal distribution assumption, and a Two Independent sample test (Mann-Whitney U test) on the concepts that do not follow the normal distribution assumption to evaluate the hypothesis, with a level of confidence of 95% (alpha = 0.05). To reject the null hypothesis, the test probability (p-value) must be less than alpha. Table 4 illustrates the mean and standard deviation of the gain score for both groups across each concept. In addition, the p-value for each concept shown in Table 4 is > alpha, which leads us to accept the null hypothesis. This means that there was no statistically significant difference in the students' OOP improvement learning skill between the experimental group and the control group.

TABLE 4 THE ANALYSIS RESULT OF THE TWO GROUPS' ACHIVEMENT DEGREE ACROSS EACH OOP CONCEPT

| Concepts | Groups | Mean | SD | P- value |
|---|---|---|---|---|
| Variable | Experimental Group | 0.04 | 0.13 | 0.63 |
| | Control Group | 0.03 | 0.10 | |
| Method | Experimental Group | 0.04 | 0.11 | 0.29 |
| | Control Group | 0.01 | 0.09 | |
| Class | Experimental Group | 0.02 | 0.21 | 0.61 |
| | Control Group | -0.01 | 0.17 | |
| Overload | Experimental Group | 0.02 | 0.21 | 0.61 |
| | Control Group | -0.01 | 0.17 | |
| Encapsulation | Experimental Group | 0.04 | 0.13 | 0.65 |
| | Control Group | 0.03 | 0.10 | |
| Constructor | Experimental Group | 0.04 | 0.14 | 0.07 |
| | Control Group | -0.01 | 0.11 | |
| Object | Experimental Group | 0.04 | 0.13 | 0.60 |
| | Control Group | 0.02 | 0.11 | |
| Inheritance | Experimental Group | 0.16 | 0.14 | 0.12 |
| | Control Group | 0.10 | 0.12 | |
| Aggregation | Experimental Group | 0.05 | 0.15 | 0.94 |
| | Control Group | 0.04 | 0.11 | |
| Dependency | Experimental Group | 0.03 | 0.15 | 0.73 |
| | Control Group | 0.01 | 0.12 | |
| Overriding | Experimental Group | 0.19 | 0.16 | 0.41 |
| | Control Group | 0.16 | 0.16 | |
| Polymorphism | Experimental Group | 0.34 | 0.31 | 0.44 |
| | Control Group | 0.25 | 0.28 | |

## Conclusion

There are many cognitive, physiological, pedagogical, and educational factors that come into play when deciding why students fail or succeed in learning programming. Many case studies have been performed to identify the varied issues that impact the effectiveness of learning OOP.

An abundance of research has been based on the premise that visualization can significantly impact CS education. ALICE has been implemented in some university settings and has been reported to have many benefits, including presenting OOP concepts in an engaging and fun learning environment (Schultz, 2011). However, the very simplicity and ease of ALICE may cause its limitations. For instance, although the drag and drop feature eliminates the frustration of dealing with syntax, sometimes this feature makes students dissatisfied with the tool because they cannot write their own code and cannot learn the actual process of structured program writing (Nguyen, et al., 2011). In addition, despite the availability of built-in objects in ALICE's gallery, which could positively affect some students, some could see these objects as a limit on their use of imagination to build their own objects (Bishop-Clark, Courte, Evans and Howard, 2007).

ALICE features could impact students' behavior or performance differently. These findings towards ALICE were based on subjective researches' assessments. Therefore, our intention in this research was to employ the ADA approach, which enables us to give more formal details of analysis and statistically valid objective conclusions about assessing the students' performance using ALICE.

The use of visualization tools might minimize the students' difficulties, and these students are likely to be much more engaged by smart phones and computer games than traditional introduction to computer science. However, the analysis results of the learning outcome between students who had used ALICE in an additional workshop together with their traditional classes, and students who were only taking the traditional classes, revealed no significant differences in their learning performance across all the tested OOP concepts.

The result of the comparison does not necessarily discount the use of ALICE as a tool, but it could reveal that the potential of the tool has not yet been maximized. The reason for the absent of significant difference could simply be a lack of sufficient data to show differences.

In addition, we believe that ALICE with its behavior is developed for the purpose of helping the student learn OOP concepts. Therefore, the failure to find a significant difference between the two groups in the experiment could be due to the lack of correlation between the problem-solving examples taught in the traditional classes and the examples given in the ALICE workshop. This argument is supported by Schultz's (2011) claim that according to the students' opinions, this disconnection could weaken the effectiveness of ALICE. Demonstrating initially most of the basic concepts of OOP in the traditional classes, then integrating the students with ALICE assuming that they will cognitively relate what they are learning in the workshop with what they learn in the traditional classes could distort the impact of ALICE. Therefore, in the future, combining the problem-solving exercises that are being presented in the classes with the fun and engaging environment of ALICE to teach basic OOP concepts may increase the students' cognitive learning skills.

Consequently, there is a need for both deeper qualitative and quantitative studies to evaluate ALICE interface features, and determine what features remain difficult to interact with.

## References

Agarwal, P. K. (2008). Examining the Testing Effect with Open- and Closed-Book Tests. *Applied cognitive psychology, 22*, 861-876.

Aktunc, O. (2013). A Teaching Methodology for Introductory Programming Courses using Alice. *International Journal of Modern Engineering Research (IJMER), 3*(1), 350-353.

Al-Bahi, A. M. (2007). *HI-CLASS – a Handy Instrument for Course Level Assessment.* Paper presented at the 2nd International Conference on Engineering Education & Training, Kuwait.

Allinjawi, A. A., Krause, P., Tang, L., & Al-Nuaim, H. A. (2011). *On Experimental Design for Diagnosing Student Learning Problems in Higher Education.* Paper presented at the FECS'11 - The 2011 International Conference on Frontiers in Education: Computer Science and Computer Engineering, Vegas, USA.

Allinjawi, A., Al-Nuaim H., Krause, P. (2014). An Achievement Degree Analysis Approach to Identifying Learning Problems in Object-Oriented Programming. *ACM Transactions of Computing Education (TOCE).* [In press]

Bishop-Clark, C., Courte, J., Evans, D., & Howard, E. V. (2007). A Quantitative and Qualitative Investigation of Using ALICE Programming to improve Confidence, Enjoyment and Achievment among Non-Majors *Educational Computing Research 37*(2), 193 - 207.

Boyle, T. (2003). Design principles for authoring dynamic, reusable learning objects *Australian Journal of Educational Technology 19*(1), 46 - 58.

CC2001. (2001). Computing Curricula 2001 Computer Science. Retrieved from: http://www.acm.org/education/curric_vols/cc2001.pdf

Chu, H.-C., Hwang, G.-J., Tseng, J. C. R., & Hwang, G.-H. (2006). A Computerized Approach to Diagnosing Student Learning Problems in Health Education. *Asian*

*Journal of Health and Information Sciences, 1*(1), 43 - 60.

Chuda, D. (2007). *Visualization in education of theoretical computer science.* Paper presented at the 2007 international conference on Computer systems and technologies, Bulgaria.

Cliburn, D. C. (2008). *Student opinions of Alice in CS1.* Paper presented at the Frontiers in Education Conference. FIE 2008., Saratoga Springs, NY.

Cooper, S., Dann, W., & Pausch, R. (2003a). *Teaching objects-first in introductory computer science.* Paper presented at the 34th SIGCSE technical symposium on Computer science education, Reno, Nevada, USA.

Cooper, S., Dann, W., & Pausch, R. (2003b). Using Animated 3D Graphics To Prepare Novices for CS1 *Computer Science Education Journal, 13*(1), 3 - 30

Daly, T. (2011). Minimizing to maximize: an initial attempt at teaching introductory programming using Alice. *Journal of Computing Sciences in Colleges, 26*(5), 23-30.

Dann, W., Cooper, S., & Pausch, R. (2000). *Making the connection: programming with animated small world.* Paper presented at the 5th annual SIGCSE/SIGCUE ITiCSEconference on Innovation and technology in computer science education, Helsinki, Finland.

Dann, W., Cooper, S., & Pausch, R. (2001). *Using visualization to teach novices recursion.* Paper presented at the 6th annual conference on Innovation and technology in computer science education, Canterbury, United Kingdom.

Dann, W., Cooper, S., & Pausch, R. (2005). *Learning to Program with Alice*: Pearson Prentice Hall.

Dann, W., Dragon, T., Cooper, S., Dietzler, K., Ryan, K., & Pausch, R. (2003). *Objects: visualization of behavior and state.* Paper presented at the 8th annual conference on Innovation and technology in computer science education, Thessaloniki, Greece.

Dimitrov, D. M., & Rumrill, P. D. (2003). Pretest-posttest designs and measurement of change. *A Journal of Prevention, Assessment and Rehabilitation, 20*(2), 159-165.

Esteves, M., Fonseca, B., Morgado, L., & Martins, P. (2011). Improving teaching and learning of computer programming through the use of the Second Life virtual world. *British Journal of Educational Technology, 42*(4), 624-637.

Fisher, A., & Foreit, J. (2002). Designing a Study , Intervention Study Designs *Designing HIV/AIDS Intervention Studies*. Washington, DC: Population Council.

Gharib, A., Phillips, W., & Mathew, N. (2012). Cheat Sheet or Open-Book? A Comparison of the Effects of Exam Types on Performance, Retention, and Anxiety. *Psychology Research, 2*(8), 469-478.

Jenkins, T. (2002). *On the difficulty of learning to program.* Paper presented at the 3rd annual Conference of LTSN-ICS, Loughbourgh.

Lahtinen, E., Ala-Mutka, K., & Järvinen, H.-M. (2005). A study of the difficulties of novice programmers. *SIGCSE Bull., 37*(3), 14-18.

Loi, S. L., & Teo, J. C. C. (1999). The impact of open book examinations on student learning. *New Horizons in Education, 40*, 34-42.

McKenzie, W. B. (2009). Introductory Programming with ALICE as a Gateway to the Computing Profession. *Information Systems Education Journal, 7*(38).

Moskal, B., Lurie, D., & Cooper, S. (2004). Evaluating the effectiveness of a new instructional approach. *SIGCSE Bull., 36*(1), 75-79.

Naps, T., Cooper, S., Koldehofe, B., Leska, C., Röling, G., Dann, W., et al. (2003). *Evaluating the educational impact of visualization*. Paper presented at the Working group reports from ITiCSE on Innovation and technology in computer science education, Thessaloniki, Greece

Nguyen, T.-L., Nandi, D., & Warburton, G. (2011). *ALICE In Online And On-Campus Environments – How Well Is It Received?* Paper presented at the Information Systems Educators Conference, Wilmington North Carolina, USA.

O'Kelly, J., & Gibson, J. P. (2006). RoboCode & problem-based learning: a non-prescriptive approach to teaching programming. *SIGCSE Bulletin, 38*(3), 217-221.

Oliveira, C. A. d., Conte, M. F., & Riso, B. G. (1998). *Aspects on Teaching/ Learning with Object Oriented Programming for Entry Level Coursess of Engineering*. Paper presented at the International Conference on Engineering Education–ICEE.

Pausch, R., Burnette, T., Capehart, A. C., Conway, M., Cosgrove, D., DeLine, R., et al. (1995). Alice: Rapid Prototyping for Virtual Reality. *IEEE Comput. Graph. Appl., 15*(3), 8-11.

Ragonis, N., & Ben-Ari, M. (2005). A long-term investigation of the comprehension of OOP concepts by novices. *Computer Science Education, 15*(3), 203-221.

Rodger, S. H. (2002). *Introducing computer science through animation and virtual worlds.* Paper presented at the 33rd SIGCSE technical symposium on Computer science education, Cincinnati, Kentucky.

Rowe, G., & Thorburn, G. (2000). VINCE - an on-line tutorial tool for teaching introductory programming. *British Journal of Educational Technology, 31*(4), 359-369.

Schultz, L. (2011). Student Perceptions of instructional tools in programming logic: A comparison of traditional versus Alice teaching environments. *Information Systems Education Journal, 9*(1), 60-66.

Theophilides, C., & Koutselini, M. (2000). Study Behavior in the Closed-Book and the Open-Book Examination: A Comparative Analysis. *Educational Research and Evaluation, 6*, 379-393.

Yadin, A. (2011). Reducing the dropout rate in an introductory programming course. *ACM Inroads, 2*(4), 71-76.

**Arwa A. Allinjawi** received her Bachelors and M.Sc degree in Computer Science from King Abdulaziz University, Jeddah, Saudi Arabia, in 1999 and in 2006/2007. She is a Lecturer at Computer Science Department at King Abdulaziz University. Mrs. Allinjawi's research interests include improving the teaching and learning methods in Object Oriented Programming. The work reported in this paper is part of her Phd research at Surrey University, UK.

**Hana A. Al-Nuaim** received her Bachelors in CS from the University of Texas at Austin and Masters and DSc in Computer Science from George Washington University, USA. She is an associate professor in Computer Science and the Dean of King Abdulaziz University Women's Campuses, Jeddah, Saudi Arabia. Dr. Al-Nuaim was a former CS Department head and as the Vice Dean of e-Learning and Distance Education was part of the launch of the first higher education e-learning program in the kingdom for women. She has extensive faculty training background and referred papers for publications in HCI, user-centered design, usability, multimedia, e-learning, e- government and knowledge cities and has been involved in many Web-based research projects.

**Paul Krause** has a BSc from the University of Exeter, UK, in Pure Mathematics and Experimental Physics (1977) and a PhD from the University of Exeter in Geophysics (1985). His work now focuses on the theory and application of computer science. He is Professor of Software Engineering in the Department of Computing at Surrey University, UK. His research career has spanned 34 years, with contributions in the fields of Mathematics, Physics and Computer Science. Prior to this appointment he worked in the world class research establishements of the UK's National Physical Laboratory, Imperial Cancer Research Fund and Philips Research Laboratories, UK. Prof. Krause was elected a Fellow of the Institute of Mathematics and its Applications for his contributions to the theoretical foundations of Artificial Intelligence.