CITRENZ
COMPUTING AND INFORMATION TECHNOLOGY RESEARCH AND EDUCATION NEW ZEALAND

Journal of Applied Computing and Information Technology

ISSN 2230-4398, Volume 12, Issue 1, 2008

Incorporating the NACCQ publications:
Bulletin of Applied Computing and Information Technology, ISSN 1176-4120
Journal of Applied Computing and Information Technology, ISSN 1174-0175

:: Refereed Article A4:

Using game scenarios for teaching novice programmers

Minjie Hu
Tairawhiti Polytechnic, New Zealand
min@tairawhiti.ac.nz

Hu, M. (2008). Using game scenarios for teaching novice programmers. *Journal of Applied Computing and Information Technology*, 12(1). Retrieved June 2, 2015 from http://www.citrenz.ac.nz/jacit/JACIT1201/2008Hu_GameScenarios.html

Abstract

As programming educators we need to find ways to engage our students.

The students we see today have been called the Nintendo generation. Such students are continually exposed to fast-paced sound, graphics, animation and games. It can be argued that these are the kinds of things that Nintendo generation students want to develop when learning computer science. As a result, computer programming educators have started to use games to engage and motivate students who are learning programming.

However, there are difficulties in teaching novices to program using games. In many cases, it is too complicated for novices to begin programming with the extensive packages, libraries, and available object oriented languages when they are required to develop games. Moreover, the games development may seem trivial to the Nintendo generation if we do not include artificial intelligence (AI). Unfortunately, AI algorithm development is not appropriate for novices who are still trying to grasp the simple syntax and semantics of programming.

This paper reports on research that explores how educators can motivate students to learn programming by using simple game scenarios. A revised version of Bloom's taxonomy is employed as a framework to aid in the creation of teaching resources that utilise game scenarios as exemplars, exercises and assessments. Finally, some recommendations are made on how the teaching of programming might be improved through a game approach to teaching and learning.

Introduction

Like many tertiary education institutes, Tairawhiti Polytechnic has students with a wide range of academic and cultural backgrounds. Previously, various strategies for the teaching of programming courses have been reported (Hu, 2003). These strategies were designed to meet both the student's expectations and the course objectives in terms of teaching resources, teaching style and assessment approach. One of the significant changes made was to teach introductory programming using MicroSoft (MS) Visual Basic (VB) instead of C++. A recent National Advisory Committee on Computing Qualifications (NACCQ) moderation report found that VB is the most popular language for teaching first year programming courses at Institutes of Technology and Polytechnics (ITP) in New Zealand. The primary benefit of adopting

VB is that a novice programmer can rapidly learn enough VB to produce impressive looking graphical user interface (GUI) programs (Tanall & Davey, 2001). It has been claimed that VB is the simplest programming language for developing the common Windows applications and that the quality of applications developed in VB can be as high as those developed in C++ (Sink, 2002).

Earlier research (Hu, 2004) has shown that teaching VB using the dynamic visualization of flowcharts and code makes it easier for students to obtain immediate feedback on variables based on step-by-step execution under program control. Additionally, it was found that using dynamic visualization helps to establish a degree of self-confidence in students who are learning programming. Further research (Hu, 2005) found that it is beneficial to integrate VB with other computer subjects such as database development, data modeling, web application development and MS Office applications. Moreover, it was found that it is relatively easy to integrate VB components with components developed in other programming languages (Hu, 2006). For the reasons outlined above, VB has been applied as an instrument for teaching novice programmers at Tairawhiti Polytechnic for several years.

Guzdial and Soloway (2002) argue that computer science educators are guilty of employing an out-dated view of computing and teaching much as they learned it, by laying out complex patterns of abstract decisions and computations, manipulating invisible data structures, and then printing a number or a phrase.

In a response to the demands of the new Nintendo generation, computer educators have begun to investigate a game-based approach to the teaching and learning of introductory programming. Kearney and Skelton (2003) discussed an approach to bringing games programming into their classroom in an engaging, challenging and pedagogically valid way. More recently, Haden (2006) described a second year programming paper teaching traditional programming skills through games programming.

This paper, which is an updated and expanded version of an earlier conference paper (Hu, 2007), reviews the current literature in the field and then discusses how to integrate simple game scenarios into the teaching of computer programming. These scenarios act as a way of illustrating basic programming concepts, using VB, in order to motivate students to learn programming. Anderson et al.'s (2001) revised Bloom's taxonomy is used as a framework to create game scenario teaching resources. Finally, some recommendations are made on how the teaching of programming might be improved through a games-based approach to teaching and learning.

Background

Why games programming?

In order to improve student engagement, computer educators need to understand the way in which students are motivated to learn. Motivation is defined as "the complex forces, drives, needs, tension states, or other mechanisms that start and maintain voluntary activity directed toward the achievement of personal goals" (Hoy & Miskel, 1982). It is also described in three general forms, namely extrinsic (based on future reward), intrinsic (based on the subject itself) and social (based on the influence of a third party or family) (Feldgen & Clua, 2004; Jenkins, 2001).

Several studies have been undertaken recently that attempt to identify the key motivator for students to learn programming. Guzdial and Soloway (2002) found that using games is an intrinsic motivator for students who are learning to program. Xu (2006) claims that using games to teach programming is similar to two people holding a dialog; if the topic is interesting, they will talk for longer. It can be argued that, in the same way, games stimulate students to continue to communicate with the computer in the form of programming.

Perhaps, the key to the success of games in engaging students is that games encourage an element of fun, imagination and creativity that is lacking in more traditional business orientated teaching examples and assessments (O'Kelly & Gibson, 2006). It can therefore help to transform fragile knowledge into concrete skills that

can be applied in new and different situations. Additionally, it has been observed that game programming encourages students, in particular novices, to experiment, invent, and modify programs (Becker, 2001; Kolling & Henriksen, 2005). Finally, it has been reported that the interest a game programming course provides can improve retention and attendance rates (Feldgen & Clua, 2004; Haden, 2006).

Many computer programming teachers lament the fact that their students are poor problem solvers. Becker (2005) identified three ways in which to teach students to become better problem solvers. One of these ways is to teach with diversity in order to address students' individual learning styles. She found that teaching programming using games supports a variety of learning styles. However, integrating game scenarios into the instructional design for teaching novices to program is itself still an unexplored area.

Models for the integration of games

Many educators (Kearney & Skelton, 2003; Kolling & Henriksen 2005; Masuch & Freudenberg, 2002; Motta & Lima, 2006; Parberry, Kazemzadeh, & Roden, 2006; Purewal & Bennett, 2006) teach game programming in second-year advanced programming courses in which the students already have knowledge of object-oriented (OO) principles and Artificial Intelligence (AI). The choice for timing of such a course is made because it is believed that in order to provide an engaging programming course, prior programming experience and a reasonable level of conceptual sophistication are required (Distasio & Way, 2006). Such courses make use of advanced packages and libraries for game development. Software that has been reported as being used in these course includes RoboCode, Jsoccer, DirectX, Director, jogl, open GL, Sage and Shark-3D.

Other educators have integrated games into their introductory programming courses (Cliburn, 2006; Feldgen & Clua, 2004), creating console-based games using Pascal and C++. However, the console-based games have less appeal and limited user interaction and are not believed to motivate the current generation of students.

Alice may be used to bridge the gap between the previous approaches to integrating games into novice programming courses. Alice was developed at Carnegie Mellon University and has been successfully utilised as an introductory programming tool in various universities (Klassen, 2006; Zaccone, Cooper, & Dann, 2003). It enables novice programmers to develop interesting 3D animations and games by drag-and-drop. The students are free from syntactic concerns because the environment ensures the correctness of the syntax and they are therefore free to focus on the semantics and OO principles. Klassen (2006) reports that the students found Alice to be a fun way to learn but they were concerned about the amount of time they had to spend in order to generate a 3D picture. The students expressed the view that the time could be better utilised by learning more pure programming concepts. After using Alice for three semesters, Klassen (2006) concluded that Alice did not serve the goal of providing a solid foundation of programming concepts within the first semester of programming. Moreover, Distasio and Way (2006) believe that while Alice is fun, using it for games programming requires significant overheads in terms of time and experience.

There are numerous game scenarios to choose from. How can we as educators adopt and adapt them to meet the learning outcomes for a first year programming course?

Designing the Course

The following research utilizes game scenarios and Anderson et al.'s (2001) revised Bloom's taxonomy to design instructional resources. The notion of using Bloom's (1956) taxonomy to enable the design of course objectives and items for the assessment of novice programmers is not new. However, the taxonomy provides a useful framework that with careful application ensures that the resources developed for a course do the job that they were intended to do. Lister (2001) provided guidelines on the use of Bloom's taxonomy in designing assessments and learning objectives for first year programming courses. The Bracelet Project group (Whalley, et al., 2006) applied the revised Bloom's taxonomy as a way of generating multiple-

choice questions that assess novice program reading and comprehension skills.

The Cognitive Process	Definition by Anderson et al. (2001)	Interpretation by Thompson et al. (2008)	The Framework of Course Progression
1. **Remember**	Retrieving relevant knowledge from long-term memory.	Recalling any material explicitly covered in the teaching programme.	Implementing examples
2. **Understand**	Constructing meaning from instructional messages, including oral, written, and graphical communications	Translating an algorithm from one form of representation to another form	Explaining examples
3. **Apply**	Carrying out or using a procedure in a given situation.	The process and algorithm or design pattern is known to the learner and both are applied to a familiar or unfamiliar problem	Doing a series of similar exercises based on examples
4. **Analyse**	Breaking material into its constituent parts and determining how the parts relate to one another and to an overall structure or purpose	Being able to break a programming task into its component parts	Doing assignment based on exercises
5. **Evaluate**	Making judgements based on criteria and standards	Determining whether a piece of code satisfies the requirements through defining an appropriate testing strategy	Doing assignment based on exercises
6. **Create**	Putting elements together to form a coherent or functional whole; reorganizing elements into a new pattern or structure	Constructing a code segment or program either from an invented algorithm or through the application of known algorithms in a combination that is new to the students.	Doing assignment based on exercises

Table 1. Revised taxonomy and framework

The key tools used in our course are game scenario based resources, VB with dynamic visualization, and the MS VB.NET integrated development environment (IDE). In addition the course makes use of a textbook by Crews and Murphy (2004).The purpose of our course is not only to deliver the contents of the course, but also to produce students that are mature, self-motivated and independent learners. To this end, the key steps that students progress through during the course are designed so that the course as a whole holds to the principles of the Learning Maturity Model (Thompson, 2006).

Thompson (2006) suggested that a student's learning processes is a series of transformations from a novice to a matured learner. In this case a student can be identified as having reached a suitable level of learning maturity when they have achieved the capability to repeatedly and reliably meet learning outcomes. He applies the capability maturity model to a learning maturity model and divides the process into different levels.

The approach throughout our course follows a developmental model that begins at lower-order thinking and slowly transitions to higher-order thinking. Based on the interpretation of the revised taxonomy in the domain of teaching programming in the cognitive dimension (Thompson, Whalley, Luxton-Reilly, Hu & Robbins, 2008), it forms the framework unifying the elements of teaching, learning and assessing. In Table 1, the columns list the cognitive processes, the definitions by Anderson et al. (2001), the interpretations by Thompson et al. (2008) and the framework of course progression in terms of examples, exercises and assignment.

The key progression of the course is as follows (the level achieved by each element is indicated in italics):

1. The tutor explains and guides students through the study of the sample code for a game. Students begin to comprehend or "understand" the programming language, its syntax and semantics.
2. Following on from the explanation, presented in 1, the students implement the example to familiarise and then to "remember" or recall what they have learned.

3. Doing further modification by adding similar code based on the example presented in 2, students learn how to "apply" their newly gained knowledge to different situations in several similar exercises.
4. Designing a variation on the exemplar game, students start to learn how to "analyze" the requirements and existing design for the assignment.
5. Testing and comparing the provided code with their own code for the assignment, students learn how to "evaluate" programs.
6. By adding new functionality that improves the game, students learn how to "create" and develop programs from the assignment.

The Course Progression

At the very beginning of a GUI/games-first style approach to teaching programming, there are several core concepts that need to be covered above and beyond a traditional non-GUI approach. These include event-driven programming and simple GUI components such as forms, buttons, labels and textboxes. To introduce students to these things we start with a simple game called "Dice". The coding of this game is modeled by the tutor in a stepwise manner and used to discuss some basic programming concepts and is subsequently provided to students as their first piece of example code.

A simple first dice game

The first application that students are exposed to is a simple dice game. This game is presented to the students using a staged set of resources that include examples and exercises. This first game fulfills the course progression steps 1 and 2.

Event-driven, controls and variables

Example 1: Roll a Die.
Write a program to stimulate a roll of a die.

This program displays a randomly generated number between one to six at the click of a button. Before the tutor starts coding, they explain how a button may be used as a trigger to start the program and how a label may be used to display information on the form. A projector is used to allow students to view the process of programming. The code developed is as follows:

```
Dim Dice1 As Integer

Randomize()

Dice1 = Int(Rnd() * 6) + 1

Label1.Text = Dice1
```

From this small piece of code, the students have been taught the basic concepts related to the nature and use of data type, variables, VB built-in functions, processing and outputting data. Through the demonstration they experience first hand that without Randomize() the dice always generates the same sequence of numbers. As a result they are introduced to the concept of pseudo-random number generation using a concrete example.

Exercise 1: Roll Two Dice.

Students now add their own code to the program of Example 1. This is a small scale exercise that fulfils step 2 of the developmental model.

The students are required to write a program that simulates the roll of two dice and displays the result. This exercise requires them to essentially duplicate the code, but in order to do this correctly they must be able to read and comprehend the example code from step one. The students are instructed to use the debugger to test their code and track variable values.

The key to the students' learning in this step is the use of dynamic visualisation (Hu,

2004) and a debugger to help students understand the programming concepts and correct errors in their code. Figure 1 illustrates the dynamic change of values for each variable when the code runs step-by-step.

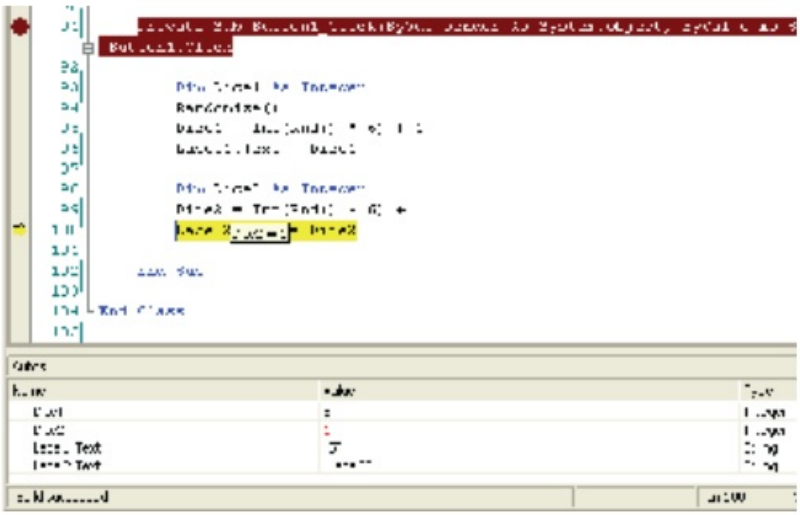

Figure 1. The dynamic change of variable values

It has been observed that students are more interested and are willing to participate in this practice because they are eager to test their own code in the IDE. Because the students are engaged and enthused by this small exercise they are more motivated to move on to next step.

More control objects and data types

Example 2: Add a Picture for Dice.

Modify Example 1 to add an image to the Dice, which matches the number in the label for the game GUI (Figure 2).

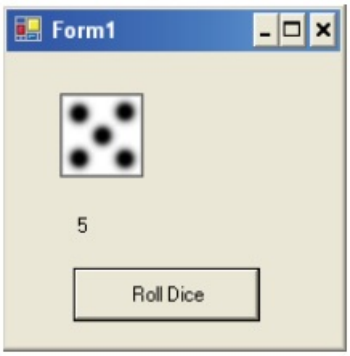

Figure 2. The dice exercise using an image

Six images are used to represent the six numbers or sides of each dice. Initially we avoid introducing students to the complexities of a selection statement. Instead of using a nested IF or CASE statement to select each picture in the first version we simply use a concatenated string to select the image that corresponds to the randomly generated number. This example provides an opportunity for the tutor to discuss how different data types can be combined.

Exercise 2: Add Two Pictures for Both Dice.

Similarly, students are required to extend the latest example code in order to create a second die (Figure 3).

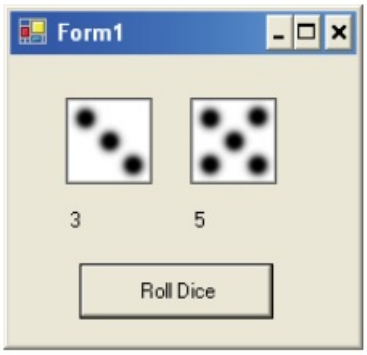

Figure 3. Exercise 2, two dice

So far, students have learned how to use various VB controls on a VB Form. Moreover, they should now understand data types, variables and the notion of sequential programming. The examples give students the opportunity to read and comprehend VB code. The exercises reinforce and further their understanding of the programming concepts taught during the development of Example 2.

Selection, local and module level variables

We have observed that once students have got the dice displayed on the screen they are keen to learn how to extend the application into a playable game.

Example 3: A First Simple Playable Game.

The one and only rule for the dice game is that if the two dice numbers are the same, the player wins.

The concept of a selection statement is introduced in order to complete this game. The GUI should display the number of times the player has won and the number of times the game has been played (the number of rolls of the dice). An Exit button is added that triggers the creation of a modal message box that confirms the termination instruction. A sample screen snapshot of working Example 3 is provided in Figure 4.

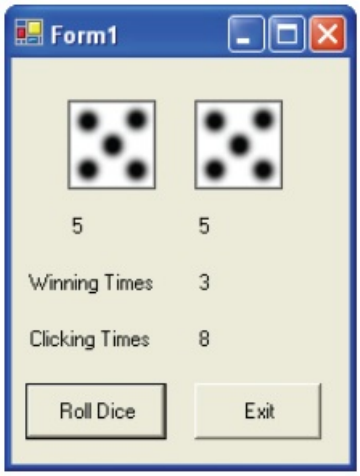

Figure 4. The first simple playable game

Firstly, students are required to apply their current programming knowledge in order to display the value of a variable that counts the number of times the game has been played. The line of code that increments the counter is introduced as count = count + 1. However, when this version of the code is run students discover that they always get the same value due to the use of a local variable. In this way the idea of a module variable can be introduced. By comparing the two different kinds of variables, students learn about the scope of each variable and also how to avoid or identify a similar mistake in the future.

Next the students learn how to use IF statements to set up the game and to exit the

game. Additionally they learn how to use an IF statement to count the times that the player has won the game (w = w + 1). The Example 3 code is as follows:

```
Private Sub Button1_Click(...) ...

.....

        count = count + 1

        If Dice1 = Dice2 Then

                MsgBox("You win.")

                w = w + 1

        End If

End Sub
```

The code is now getting more complicated. Students are positively keen to learn because they expect their game is getting better and better. They are enthusiastic to try adding extra features to it, such as wallpapers for the Form, colours, different sizes and fonts for words and even music or sound for winning. The programming class is now more eager to do the tasks and no longer requires persuasion from the teacher.

The dice game as an assessment item

Since games were introduced as examples and exercises students have also expected to have a game program as part of their assignment. However, our goal in using games is not to teach game development. We want students to learn core-programming principles. The dice game scenario is not only used to stimulate students to learn programming but is also used as a foundation for a course assignment. The assignment is designed in such a way that the students reach stage 3 in the course progression.

A more complicated IF statement is required in the assignment to improve the dice game. Students are also required to use a different layout for game output. The Assignment asks students to write a program to simulate rolling two die. If the dice have the same number, the player wins 5 points. If the dice have different numbers, the player loses 1 point. The result required is as shown in Figure 5.

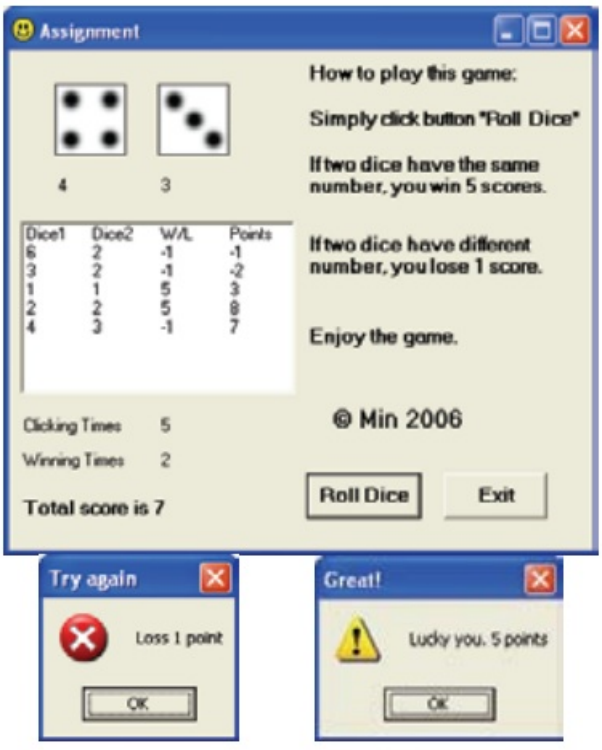

Figure 5. Sample result of assignment

Concurrent with the students' undertaking the assignment they are expected to complete non-game-based exercises, such as individual wage, tax calculation and total accumulation. These exercises continue the development of the students as independent learners and focus on stage 3 of the course progression (see Table 1).

The executable (.exe) file of a sample game is made available to students so that they may play, experiment with, and understand the requirements of the assignment. By providing this playable file, we make it easier for students to understand what they need to achieve, especially those for whom English is as second language. We have found that supplying an executable file is better than only supplying a written specification of the assignment. If a picture is indeed worth a thousand words, then it might be argued that a playable sample game is worth more than thousand pictures.

After completing this assignment successfully we believe that students have learned how to analyze and evaluate an existing GUI application. They have also learned how to create their own project from an existing one and how to test and evaluate their software independently. Through the three stages of example, exercise and assignment, the dice game program covers the six cognitive process of revised Bloom's taxonomy. With this limited content of knowledge, students are on their way to becoming a more mature learner.

Games as a Common Thread in a Programming Course

While using game scenarios to stimulate students to learn programming, we became impressed and encouraged by the fact that students were continuing to use the game scenarios in their next programming course. The framework of three stages of example, exercise and assignment continue throughout the programming course with various game scenarios.

Simple animation game (IF statement)

An animated object is firstly introduced moving from left to right hand side (see figure 6) (stage 1 of the course progression: explaining example for understanding). After explanation, students were firstly asked to implement the example (stage 1 of the course progression: remembering the teaching) and then to change the speed of the moving object (stage 2 of the course progression: applying the learning to different exercises). In the further exercises, students were also asked to make the object move from right to left, then make the object continually move from one end to the other, and finally make the object move from the top to bottom. Thus, the object is moving like a bouncing ball. Finally, an iterative control by mouse is added to complete it as an interactive ball game.

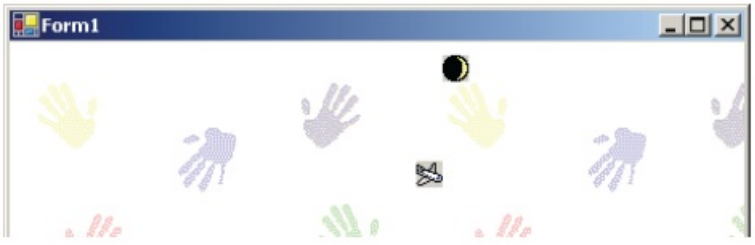

Figure 6. Animated objects

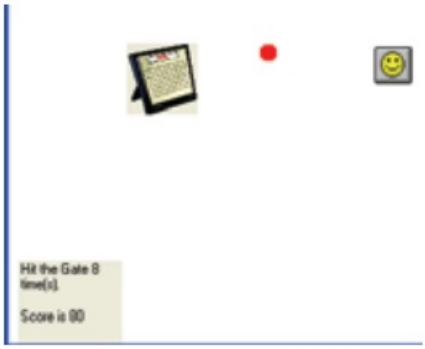

Figure 7. Ball game

After all the exercises, which relate to applying the IF statement to various situations, the assignment asks the students to create a playable interactive ball game (see Figure 7) (stage 3 of the course progression: creating a playable game for the assignment). Students create their own game with different rules for scoring when playing the ball, based on the analysis and evaluation of the examples and exercises (stage 3 of the course progression: analyzing and evaluating examples and exercises for the assignment).

Similarly, after introducing each of the following examples in the contents of programming course, students are required to implement the example and then to do further exercises, which are modified from the examples. Finally, the assignment is to create a playable game based on the examples and exercises.

Bubble and Guess game (FOR loops and subroutines)

Drawing a picture in the graphics device interface (GDI) was also introduced when the ball game was implemented. Drawing a number of bubbles leads to the use of the repetition statement FOR-NEXT and the Random function (see Figure 8).

Figure 8. Bubbles by FOR loop

Guess a Number game (WHILE loops and subroutines)

The Guess a Number game is used to introduce WHILE-Loop statements (Figure 9). In this game the player is trying to guess a number that has been generated randomly by the computer. In the game scenario students learn how to combine loop and IF statements. They also learn how to write cohesive functions by making use of sub-procedures and function procedures.

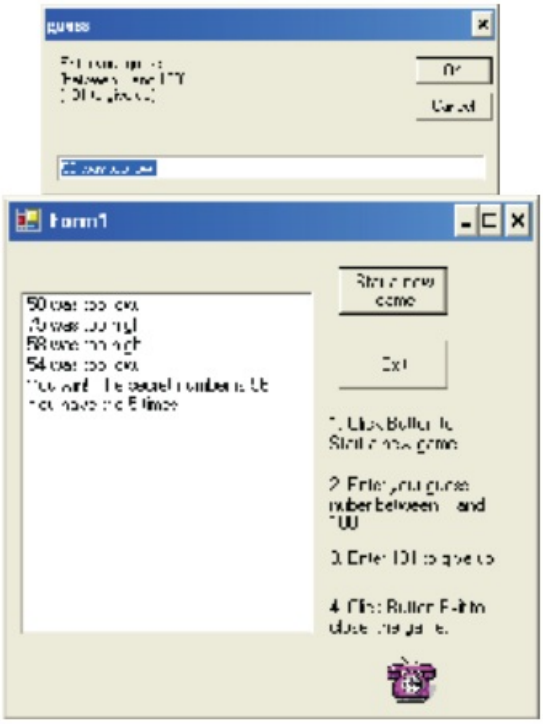

Figure 9. Guess number game

Lotto game (arrays, IF and loops)

This game simulates a lottery draw (Figure 10) and is used as a means of introducing the students to the basic concepts of an array data structure.

They learn about array indices, elements and potential applications.

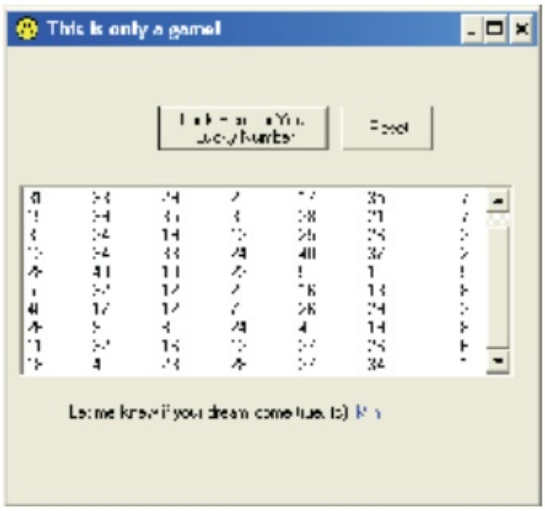

Figure 10. Lotto game

Students often rush into coding the procedure and make the mistake of creating more than one instance of the same number in a single lottery draw. To get rid of this bug they need to nest the Loop and IF statement as they did in the Guess a Number game. The students find that they need to use dynamic visualization in order to identify and fix the bug.

An improved Dice game (files and subroutines)

An improved version of the original Dice game is now introduced (Figure 11). In order to complete this version of the game, students are required to read and write the score in the text file. The game also needs an algorithm that sorts the scores and a search algorithm to find data within a certain range. Students also learn the GUI design to trigger a number of procedures and functions.

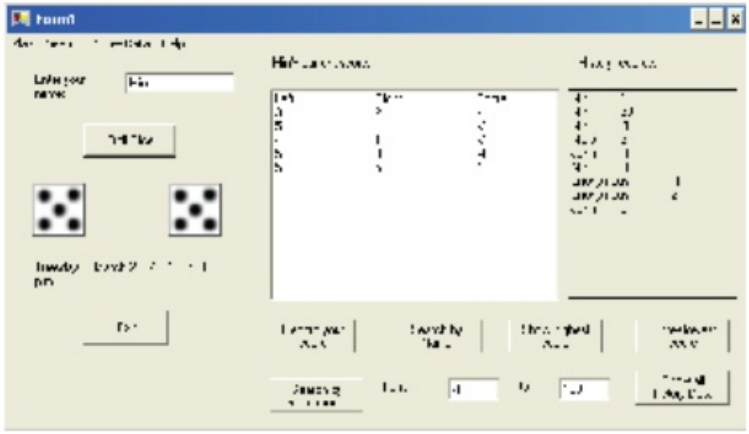

Figure 11. An improved Dice game

Dice game again (object-orientation)

Once again, the Dice game was used when OO was first introduced in the second year course (see Figure 12). Students were interested to learn how to use different programming styles to implement the same game. Through the comparison of two different approaches, they learned the advantage of reuse.

```
Public Class Dice
    Private m_die As Integer
    Public ReadOnly Property Die() As Integer
        Get
            Return m_die
        End Get
    End Property
    Sub Roll()
        Randomize()
        m_die = Int(Rnd() * 6) + 1
    End Sub
End Class
```

The development of the Dice game involved further concepts of class relationship (see Figure 13). Each player has two dice. When the player instance is removed, the dice instance is automatically destroyed. The composition relationship between player and dice classes are presented in the game scenario, which students are already familiar with.

Results

The teaching and learning practice for programming models the same approach to learning maturity and progression in developmental steps as described in this paper. The same model is also applied to developing the resources where the progression of example to exercise to assignment is followed.

The game scenario has been applied to programming contents in terms of event-driven, control objects, variables, control flow, subroutine, array file and OO as described previously. For each game, various examples, exercises and assignments are designed to cover the six cognitive processes of the revised Bloom's taxonomy. By repeating these three stages for each game, students not only complete the course content but also progress from a novice to a more mature and independent learner.

Through observation of the activities in the computer room, we noticed that students are regularly seen playing and testing their games by themselves or with classmates and friends. They spend more time on programming both during and after class than before game scenarios were introduced. It is no longer necessary for us to check whether or not the students are on task, due to the fact that we have managed to capture the students' interest through introducing game scenarios in programming.

After each exercise, students are more than willing to learn a step further before the next topic is introduced. This is great feedback. It shows they have already understood how to apply existing knowledge and that they are ready to go on to next step.

A survey has been conducted based on the rating system of Leutengger and Edginton (2007). The rating system was {1 = no clue; 2 = so-so; 3 = think I understand; and 4 = mastered}. Table 2 presents our survey contents and results. The average score of all the items is 3.3 out of 4. This means students are generally confident about their first year's study of programming. It appears that the students understand practical items better than abstract or paper work items. More exercises are needed for the items that are rated less than the average score for the purpose of future teaching.

Students were also asked about the game scenario for teaching programming. The rating system was {1= hated using games; 2 = game scenario neither hurt nor helped; 3 = game scenario was good; 4 = game scenario made it great}. The average score is 3.0 out of 4.

The majority of students were choosing 3, while two students selected 4 and the remaining two chose 2. This means most students enjoyed using game scenarios in programming.

Through game programming, students have a more positive attitude to programming in general. They now undertake both game and nongame programming equally well. Some students even find that they are spending less time to complete non-game programming than the game ones.

Item	Rating
1. Variables	3.6
2. Input	3.5
3. Output	3.5
4. Selection	3.3
5. Loop	3.5
6. Procedure and Function	3.3
7. Array	3.1
8.File (Read / Write)	3.1
9. Algorithm	2.7
10. GUI	3.4
11. Flowchart	3.1
12. Hierarchy Chart	3.0
13. Desk Check	3.1
14. Debug and Test	3.6

Table 2. Survey result

During the 2006 NACCQ moderation, nine ITPs submitted the module PP590 Programming Concepts and Tools and the report stated that all used VB and all used appropriate assessments. The mixture of game and non-game programming assessments from this research was recognized by the following comments:

In most cases good practical programming modules were moderated with

> Tairawhiti standing out in both presentation and in the level of assessment for this 500 level programming module … A very good module for 500 level in both theory and practical.

We are now adaptihng a research-informed approach to our teaching and course development. This has lead to improved teaching and learning. Moreover, we have found that the attitude of students towards learning programming has improved, as well as their confidence. When the next module using C# starts, students are interested in implementing the dice game by themselves in the new language after learning its syntax and grammar. We found during the process of interviewing the students that some of them opt to use VB for writing applications in their other courses. For example, students have chosen to use VB to create a prototype for an assignment in the Prototyping course. Other students comment that programming is one of the most interesting and creative courses that they have studied.

Conclusion

The purpose of this research was to use game scenarios to motivate novices to learn programming principles rather than to teach them how to develop games. Apparently, there is no need to start from complicated packages and libraries. As a programming language, Visual Basic has clear advantages in terms of visualization and rapid development when compared with console-based programming of game applications. Games, VB and Bloom's taxonomy proved to be rich sources of ideas for creating our teaching resources in terms of examples, exercises and assessments.

We could use the analogy that a traditional approach to teaching and learning programming is like a very nutritional soup, good for you but lacks taste. Game scenarios provide the spice for our soup that is not only good for the students but makes the soup more delicious and attracts students to try it again and again. The experiential results indicate that game programming stimulates students' motivation to learn programming. It been discovered that through this new approach to teaching, students are now keen to learn programming and face problems in a positive manner.

This research has yielded more benefits than we originally expected. The teacher motivates the students to learn programming. The motivation of learning from students also encourages the teacher to improve the teaching. Game scenarios provide a common thread throughout the whole teaching process to lead students from a low-order thinking to a higher-order thinking.

References

Journal of Applied Computing and Information Technology (JACIT): ISSN 2230-4398
(Incorporating the Bulletin of Applied Computing and Information Technology, NACCQ: ISSN 1176-4120 and
Journal of Applied Computing and Information Technology, NACCQ: ISSN 1174-0175)

An Open Access Journal, DOAJ #22304398, (✓zotero)