A Framework for Teaching Novice VB Programming Using Motivational Game Scenarios

Minjie Hu
Tairawhiti Polytechnic
PO Box 640, Gisborne, NZ
min@tairawhiti.ac.nz

Abstract

As programming educators we need to find a way 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 OO languages when they are required to develop games. Moreover, the games development seem trivial to the Nintendo generation if we do not include 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 attempts to explore how educators can motivate students to learn programming by using simple game scenarios. The revised Bloom's taxonomy is employed as a framework to aid in the creation 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.

Key words: games, novice programmers, Bloom's Taxonomy, Visual Basic

1. 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 MS Visual Basic (VB) instead of C++. A recent 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 GUI program (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).

Recent research has shown that teaching VB using the dynamic visualization (Hu, 2004) of flowcharts and code makes it easier for students to obtain immediate feedback on variables based on the step-by-step execution under program control. Additionally, it was found that using dynamic visualization helps establishing a degree of self-confidence in students who are learning programming. Further research 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 (Hu, 2005). 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 programming 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 & Skelton (2003) discussed an approach to bringing games programming into their classroom in an engaging, challenging and pedagogically valid way.

More recently, a second year programming paper teaching traditional programming skills through games programming was described (Haden, 2006).

This quality assured paper appeared at the 20th *Annual Conference of the National Advisory Committee on Computing Qualifications* (NACCQ 2007), Nelson, New Zealand. Samuel Mann and Noel Bridgeman (Eds). Reproduction for academic, not-for profit purposes permitted provided this text is included. www.naccq.ac.nz

This paper 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. The revised Bloom's taxonomy (Anderson, et al. 2001) 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.

2. Background

2.1 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 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) (Jenkins, 2001; Feldgen & Clua, 2004).

Several studies have been undertaken recently that attempt to identify the key motivator for students to learn programming. Guzdial & Soloway (2002) found that using games are 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, key to the success of games 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 (Kolling & Henriksen, 2005, Becker, 2001). Finally, it has been reported that the interest a game programming course provides has been shown to improve retention and attendance rates (Feldgen & Clua, 2004; Haden, 2006).

Many computer programming teachers lament at 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. Becker found that teaching programming using games supports a variety learning styles. However, integrating game scenarios into the instructional design for teaching novices to program is itself still an unexplored area.

2.2 Models for the Integration of Games

Many educators (Motta, Jr & Lima, Jr, 2006, Purewal, Jr & Bennett, 2006, Parberry, Kazemzadeh, & Roden, 2006, Kolling & Henriksen 2005, Kearney & Skelton, 2003, Masuch & Freudenberg, 2002) 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 is required (Distasio & Way, 2006). Such courses make use of different or advanced packages and libraries for game development. Software that has been reported as being used in these course are RoboCode, Jsoccer, DirectX, Director, jogl, open GL, Sage and Shark-3D.

Other educators have integrated games into their introductory programming course (Feldgen & Clua, 2004, Cliburn, 2006). But, they create console-based games by Pascal and C++. 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 by Carnegie Mellon University and has been successfully utilised as an introductory programming tool in various universities (Zaccone et. al., 2003, Klassen, 2006). 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. The students found Alice to be a fun way to learn (Klassen, 2006). But they were concerned that 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 concluded that Alice didn't serve the goal of providing a solid foundation of programming concepts within the first semester of programming. Moreover, Distasio and Way (2006) also 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. But how can we as educators adopt and adapt them to meet the learning outcomes for a first year programming course?

3. Designing the Course

The following research utilizes game scenarios and the revised Bloom's taxonomy (Anderson, et al. 2001) to design instructional resources.

The notion of using Blooms 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 applications 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 of Educational Objectives (Bloom, 1956) to the design of assessments and learning objectives for first year programming courses. The Bracelet Project group (Whalley, et. al. 2006) applied the revised Bloom's Taxonomy (Anderson et. al, 2006) as a way of generating multiple-choice questions that assess novice program reading and comprehension skills. To this end the key tools used in this 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 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. This forms the framework unifying the elements of the course. The key progression of the course is as follows (the Blooms' 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 example, presented in 1, the students modify the example by adding similar code to familiarise and then to "*remember*" or recall what they have learned.

3. Doing further similar assignment based on the exercises presented in 2, students learn how to "*apply*" their newly gained knowledge to different situations.

4. Designing a variation on the exemplar game students can learn how to "*analyze*" the requirements and existing design.

5. Testing and comparing the provided code with their own code, students learn how to "*evaluate*" programs.

6. By adding new functionality that improves the game, students learn how to "*create*" and develop programs.

4. 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 from 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 as discussed below and is subsequently provided to students as their first piece of example code.

4.1 A Simple First 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.

4.1.1. Example 1: Roll a Dice

In the first example a program is written that stimulates a roll of a dice. 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 shown in figure 1.

```
Dim Dice1 As Integer
Randomize()
Dice1 = Int(Rnd() * 6) + 1
Label1.Text = Dice1
```

Figure 1. The First Program

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.

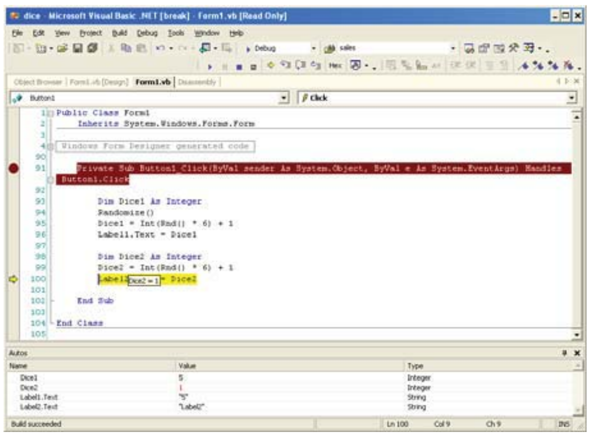
4.1.2. <u>Exercise 1: Roll Two Die</u>

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

The students are required to write a program that simulates the roll of two die 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.

Key to the students learning in this step is the use of dynamic visualisation (Hu, 2004) and a debugger to assist students understand the programming concepts and correct errors in their code. Figure 2 illustrates the dynamic change of values for each variable when the code runs step-by-step.

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 more motivated to move on to next step.


dice - Microsoft Visual Basic .NET [break] - Form1.vb [Read Only]

```
Public Class Form1
    Inherits System.Windows.Forms.Form

Windows Form Designer generated code

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
Button1.Click

        Dim Dice1 As Integer
        Randomize()
        Dice1 = Int(Rnd() * 6) + 1
        Label1.Text = Dice1

        Dim Dice2 As Integer
        Dice2 = Int(Rnd() * 6) + 1
        Label2.Text = Dice2

    End Sub

End Class
```

Dice2 = 1

Autos

```
Name          Value      Type
Dice1         5          Integer
Dice2         1          Integer
Label1.Text   "5"        String
Label2.Text   "Label2"   String
```

Figure 2. The dynamic of variable values

4.1.3. <u>Example 2: Add a Picture for Dice</u>

This example code builds on Example 1 and shows the students how to add images to their game GUI.

The tutor writes a program that displays a dice with picture and number by modifying Example 1. They are careful to point out to the students that the picture box and the label should represent the same number (Figure 3).

Six images are used to represent the six numbers or sides of each dice. The images are named from Side1.gif to Side6.gif.

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 ("Side" & CStr(Dice1) & ".gif") to select the image that corresponds to the randomly generated number.

This example provides an opportunity for the tutor to discuss how different data type can be combined.

In this example a single statement is added to the code in Example 1, the result is shown in Figure 3.

```
PictureBox1.Image = Image.FromFile ("Side"
                    & CStr(Dice1) & ".gif")
```

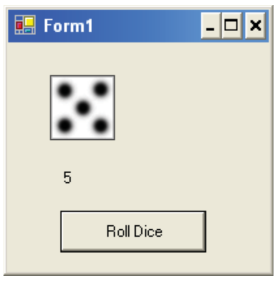


Figure 3. The dice exercise using an image.

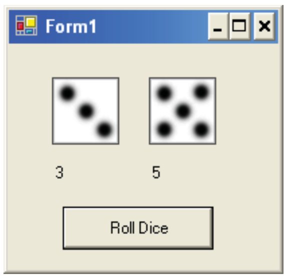


Figure 4. Exercise 2, two die

4.1.4. <u>Exercise 2: Add Two Pictures for Both Dice</u>

In this exercise once again the students extend the latest example code. The students use the same code as generated in Example 2 to create a duplicate dice (Figure 4).

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.

4.1.5. Example 3: A First Simple Playable Game

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

The one and only rule for the dice game is that if the two dice numbers are the same, the player wins. Thus, the concept of a selection statement is introduced in order to complete this game. A message box is used in the game as a way of notifying the current game status to the player.

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 die). An Exit button is added that triggers the creation of a modal message box that confirms the termination instruction. Sample screen snapshots of the working Example 3 are provided in Figure 5.

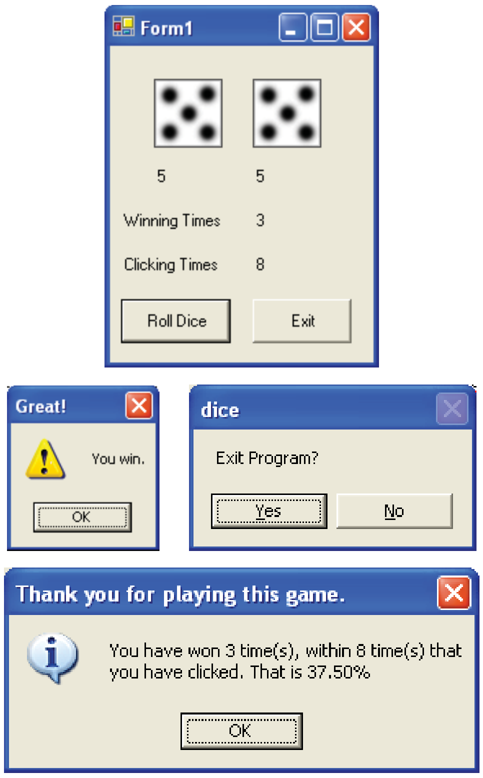

Form1
5 5
Winning Times 3
Clicking Times 8
Great!
You win.
dice
Exit Program?
Thank you for playing this game.
You have won 3 time(s), within 8 time(s) that you have clicked. That is 37.50%

Figure 5. 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 (Figure 5, top). 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`).

Various message boxes are used to provide information to the player. And students learn about another built-in function allows for percentages to be formatted in a string. The Example 3 code is provided in Figure 6.

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 size and font 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.

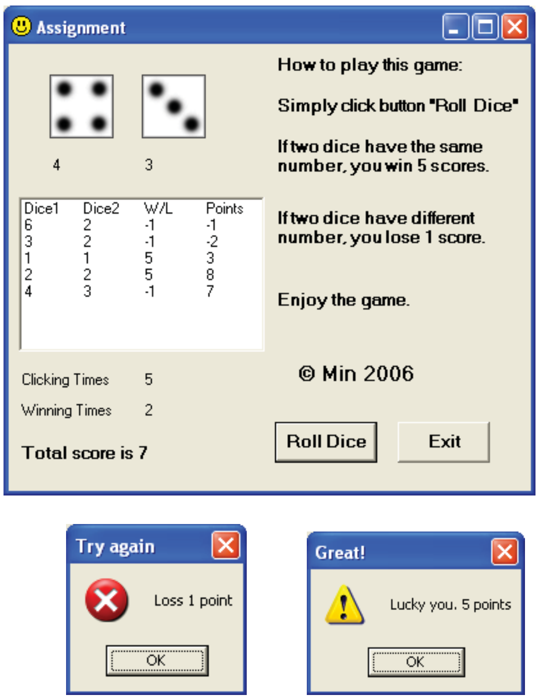
```
Dim w, count As Integer
Private Sub Button1_Click(…) …
……
……
count = count + 1
If Dice1 = Dice2 Then
MsgBox("You win.",
MsgBoxStyle.Exclamation, "Great!")
w = w + 1
End If
End Sub

Private Sub Button2_Click(…) …
Dim Msg, bye As String
Msg = ("You have won " & w
Msg = Msg & " time(s), within "& count
Msg = Msg & " time(s) that" & vbCrLf
Msg = Msg & "you have clicked. That is
" & FormatPercent(w / count)
bye = "Thank you for playing this
game."

If MsgBox("Exit Program?",
MsgBoxStyle.YesNo) = MsgBoxResult.Yes
Then
MsgBox(Msg, MsgBoxStyle.Information,
bye)
Close()
End If
End Sub
```

Figure 6. Example 3, the playable game

4.1.6. 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 die have the same number, the player wins 5 points. If the die have different numbers, the player losses 1 point. The result is required similar as Figure 7.



Assignment

How to play this game:

Simply click button "Roll Dice"

If two dice have the same number, you win 5 scores.

If two dice have different number, you lose 1 score.

Enjoy the game.

© Min 2006

4 3

Dice1	Dice2	W/L	Points
6	2	-1	-1
3	2	-1	-2
1	1	5	3
2	2	5	8
4	3	-1	7

Clicking Times 5

Winning Times 2

Total score is 7

Roll Dice Exit

Try again

Loss 1 point

OK

Great!

Lucky you. 5 points

OK

Figure 7. Sample result of assignment

Concurrent with the students undertaking the assignment they are expected, in class, to complete non-game based exercises are also provided 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 Section 3).

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. We have found that supplying an executable is better than supplying a written specification of the assignment. We have found that a picture is indeed worth a thousand words. Hence it might be argued that a playable sample game is worth more than thousand pictures. By providing a playable game we make it easier for students to understand what they need to achieve, especially for those for whom English is as second language. 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.

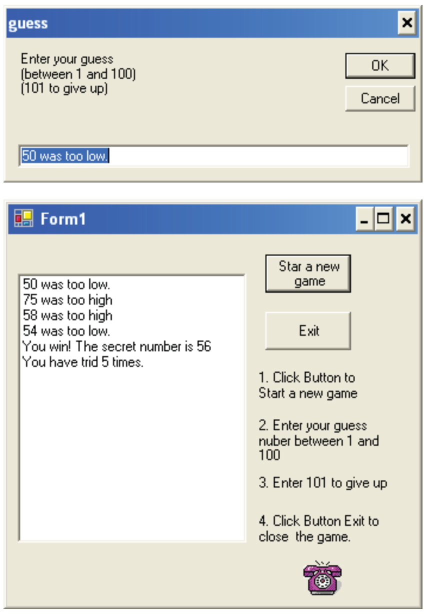
Through the three stages of example, exercise and assessments, the dice game program covers the six cognitive process of revised Bloom's taxonomy. In this limited content of knowledge, students are on their way to becoming a more mature learner.

5. Games as a Common Thread in a Programming Course

While using game scenarios to stimulate students to learn programming, the author became impressed and encouraged by the fact that students were continuing to use the game scenarios in their next programming course.

5.1 The Guess A Number Game

The guess number game (Figure 8) is used to introduce repetition statements (looping constructs). In this game the player enters a number and is trying to guess what the number is 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.



guess

Enter your guess
(between 1 and 100)
(101 to give up)

OK
Cancel

50 was too low.

Form1

```
50 was too low.
75 was too high
58 was too high
54 was too low.
You win! The secret number is 56
You have trid 5 times.
```

Star a new game

Exit

1. Click Button to Start a new game

2. Enter your guess nuber between 1 and 100

3. Enter 101 to give up

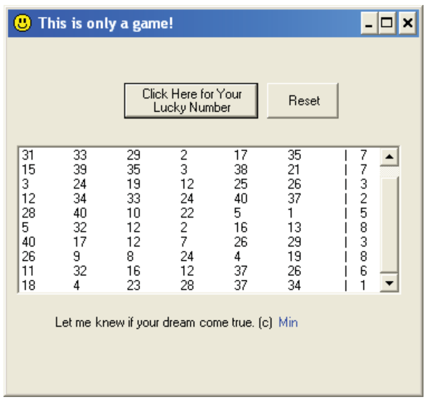
4. Click Button Exit to close the game.

Figure 8. Guess number game

5.2 The Lotto Game

This game simulates a lottery draw (Figure 9) 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.

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 (Section 5.1). The students find that they need to use dynamic visualization in order to identify and fix the bug.



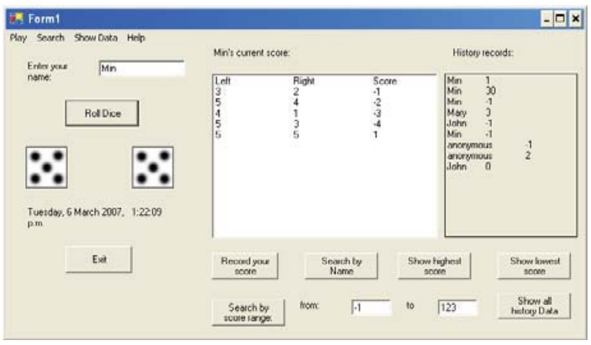
```
31    33    29    2     17    35    |  7
15    39    35    3     38    21    |  7
3     24    19    12    25    26    |  3
12    34    33    24    40    37    |  2
28    40    10    22    5     1     |  5
5     32    12    2     16    13    |  8
40    17    12    7     26    29    |  3
26    9     8     24    4     19    |  8
11    32    16    12    37    26    |  6
18    4     23    28    37    34    |  1
```

Let me knew if your dream come true. (c) Min

Figure 9. Lotto game

5.3 An Improved Dice Game

An improved version of the original dice game is now taught (Figure 10). 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 simple GUI design to trigger a certain amount of procedures and functions.



Enter your name: Min

Tuesday, 6 March 2007, 1:22:09 p.m.

from: -1 to 123

Min's current score:

```
Left    Right    Score
3       2        -1
5       4        -2
4       1        -3
5       3        -4
5       5        1
```

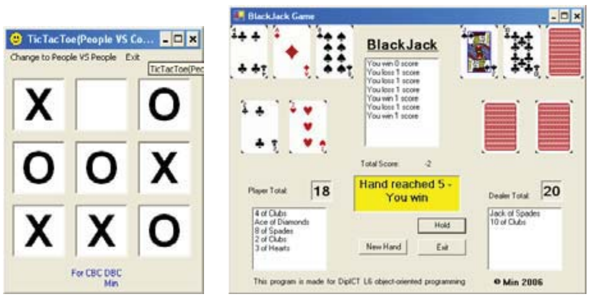
History records:

```
Min        1
Min        30
Min        -1
Mary       3
John       -1
Min        -1
anonymous       -1
anonymous       2
John       0
```

Figure 10. An improved dice game

5.4 Tic-Tac-Toe, Black-Jack & O.O.

With the success the author has had in using game scenarios to motivate students to learn programming, he has recently prepared games for teaching in a follow-on, Level 6, programming course. In this course the games Tic-Tac-Toe and Black Jack (Figure 11) are used in the introduction of object-oriented programming for the Diploma in ICT. In this course the VB Array and OOP are used in both games. Unlike the previously described games, for these more complicated ones, the author provides part of the code for students to start with. For example, we provide coding for People vs. Computer in the game of Tic-Tac-Toe. Through this, students need to complete the function for People vs. People. For Black Jack, we only provide coding for List Box, not for pictures. Students need to find out how the instance matches to the picture. They also need to provide a function to bid for double score.



```
You win 0 score
You loss 1 score
You loss 1 score
You loss 1 score
You win 1 score
You loss 1 score
You win 1 score
```

Total Score: -2

Player Total: 18

Hand reached 5 - You win

Dealer Total: 20

```
4 of Clubs
Ace of Diamonds
8 of Spades
2 of Clubs
3 of Hearts
```

```
Jack of Spades
10 of Clubs
```

This program is made for DipICT L6 object-oriented programming © Min 2006

Figure 11. Tic-Tac-Toe and Black Jack games

The teaching and learning practice for this course 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 as described in Section 4 to 5.3. For each game, various examples, exercises and assessments are designed to cover the six cognitive processes of revised Bloom's taxonomy. By repeating these three stages for each game, students not only complete the course content but have also progressed from a novice to a more mature and independent learner.

6. Results

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 has become 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 to next step.

Through game programming, students have a more positive attitude to programming in general. They now undertake both game and non-game programming equally well. Some students even find that they are spending less time to complete non-game programming than the game ones. The number of students who fail their assignment has dropped to zero since we fully introduced games in 2006 (Figure 12). Although these figures are not conclusive, and the performance of students in future semesters will need to be monitored, it shows a positive trend and is a very encouraging result.

Year	No. of Re-sit	Percentage
2004	4	44%
2005	2	25%
2006	0	0%

Figure 12. Number and percentage for re-sit

During the 2006 NACCQ moderation, there were nine ITPs submitted the module PP590 Programming Concepts and Tools.

> *"All used Visual Basic…. All used appropriate assessments".*

The mixture of game and non-game programming assessments from this research was recognized by the following comment:

> *"in most cases good practical programming modules were moderated with Tairawhiti's 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 adopting 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 their confidence. When next module using C# starts, students are interested in implementing the dice game by themselves in the new language after learning the new language syntax and grammar. We found during the process of interviewing the students that some of the students opt to use VB for the writing applications in their other courses. For example students have chosen to adopt VB to create a prototype for an assignment in the Prototyping course. Other students comment programming is one of the most interesting and creative courses that they have studied.

7. Conclusion

The purpose of this research was to use game scenarios to motivate novice 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. Game, VB and Bloom's taxonomy proved to be rich sources of ideas to create 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 way they are teaching. Game scenarios provide a common thread throughout the whole teaching process leading students from a low-order thinking to a higher-order thinking.

8. Acknowledgement

The author would like to thank Dr. Jacqueline Whalley, Computing and Mathematical Sciences, Auckland University of Technology, for her productive comments and reviews.

9. References

Anderson, L., Krathwohl, D., Airasian, P., Cruikshank, K., Mayer, R., Pintrich, P., Raths, J., and Wittrock, M. (2001): A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives, Allyn & Bacon.

Becker, K. (2005): Games and Learning Styles, Proc. ICET2005, Calgary, Alberta, Canada

Becker, K. (2001): Teaching with games: The minesweeper and asteroids experience, Journal of Computing in Small Colleges 17(2), Dec, pp 22-32

Bloom, B. S. (1956): Taxonomy of Educational Objectives Handbook 1: Cognitive Domain. New York: Longman, Green & Co.

Crews, T. and Murphy, C. (2004) Programming right from the start with Visual Basic.NET, Prentice Hall

Distasio, J. and Way, T. (2006): Exploring computer science concepts with a ready-made computer game framework, Retrieved 17 March, 2007, from http://arxiv.org/ftp/cs/papers/0609/0609070.pdf

Feldgen, M. and Clua, O. (2004): Games as a motivation for freshman students to learn programming, Proc. 34th ASEE/IEEE Frontiers in education conference, S1H 11-16.

Guzdial, M. & Soloway, E. (2002): Teaching the Nintendo generation to program, Communications of ACM, 45(4), pp 17-21

Haden, P. (2006): The Incredible Rainbow Spitting Chicken: Teaching Traditional Programming Skills through Games Programming, In Proceedings of the 8th Australian Conference on Computing Education, Hobart, Australia, pp. 81-89

Hoy, W. and Miskel, C. (1982): Education administration: Theory, research, and practice, pp137, Random House Inc

Hu, M. (2006): Component Programming, In Proceedings of the 19th Annual NACCQ, Wellington, New Zealand, pp. 127-134

Hu, M. (2005): ICE: Integrated Computing Education - An individual integrated computing teaching experience, Proceedings of the 17th Annual NACCQ, Tauranga, New Zealand, pp. 183-187

Hu, M. (2004): Teaching Novices Programming with Core Language and Dynamic Visualisation, In Proceedings of the 17th Annual NACCQ, Christchurch, New Zealand, pp. 95-104

Hu, M. (2003): A Case Study in Teaching Adult Students Computer Programming, In Proceedings of the 16th Annual NACCQ, Palmerston North, New Zealand, pp. 287-290

Jenkins, T. (2001): "Teaching programming: A journey from teacher to motivator", Retrieved 2 July, 2005, from http://www.ics.ltsn.ac.uk/pub/conf2001/papers /Jenkins.htm

Kearney, P. & Skelton, S. (2003): Teaching Technology to the Playstation Generation, Bulletin of Applied Computing and Information Technology, 1(2), Retrieved 6 Oct, 2006, from http://www.naccq.ac.nz/ bacit/0102/index.html

Klassen, M. (2006): Visual approach for teaching programming concepts, Proc. 9th International Conference on Engineering Education, San Juan, PR

Kolling, M. and Henriksen, P. (2005): Game programming in introductory courses with direct state manipulation, Proc. ITiCSE'05

Lister, R. (2001): Objectives and objective assessment in CS1, Proc. 32nd SIGCSE technical symposium on Computer Science Education, Charlotte, North Carolina, USA, pp 292 - 296

Masuch & Freudenberg, (2002): Teaching 3D computer game programming. Retrieved 2 Sept, 2004, from http://wwwisg.cs.uni-magdeburg.de/graphik/pub/files/ Masuch_2002_T3D.pdf

Motta, Jr. P. and Lima, Jr. H. (2006): Teaching computer programming: A game driven approach, Retrieved 5 March, 2007, from http://www.cin.ufpe.br/~sbgames/ proceedings/files/Teaching%20Computer%20Programming.pdf

O'Kelly, J. and Gibson, Jr. P. (2006): RoboCode & Problem-based learning: A non-prescriptive approach to teaching programming, Proc. ITiCSE'06

Parberry, I, Kazemzadeh, M, and Roden, R. (2006): The art and science of game programming, Proc. SIGCSE'06

Purewal, Jr. T. and Bennett, C. (2006): A framework for teaching polymorphism using game programming, Journal of Computing Sciences in Colleges, Volume 22, Issue 2, pp. 154-161

Sink, K. (2002): VB or C++: Which is better for DirectX Games? SAMS, Retrieved 26 June, 2006, from http://www.samspublishing.com/articles/ printerfriendly.asp?p=26257&rl=1

Tanall, A. & Davey, B. (2001): How Visual Basic entered the curriculum at an Australian university: An account informed by innovation translation, Proc. Informating Science, June 2001, pp. 510-517

Thompson, E. (2006): Using a subject area model as a learning improvement model, Proc. 8th ACE2006

Whalley, J., Lister, R., Thompson, E., Clear, T., Robbins, P., Kumar, A. and Prasad, C. (2006): An Australasian study of reading and comprehension skills in novice programmers, using the Bloom and SOLO taxonomies, In Proceedings of the 8th Australasian Computing Education Conference, pp. 243-252.

Xu, C. (2006): Why and how to teach game programming, In Proceedings 2006 International Conference on Frontiers in Education: Computer Science and Computer Engineering, Las Vegas, CSREA Press USA, pp. 215-220,

Zaccone, R., Cooper, S., and Dann, W. (2003): Using 3D animation programming in a core engineering course seminar, 33rd ASEE/IEEE Frontiers in education conference, 2, pp.14-17