Improving Student Understanding, Application and Synthesis of Computer Programming Concepts with Minecraft

Brett Wilkinson, Neville Williams, Patrick Armstrong

Flinders University of South Australia, Australia

0477

The European Conference on Technology in the Classroom 2013

Official Conference Proceedings 2013

Abstract

Ensuring student acquisition, comprehension and synthesis of complex yet abstract concepts within computer programming teaching is a difficult process, especially with younger students. The syntax and semantic fundamentals within programming languages for many students emulate the complexities of learning a foreign written or spoken language. An additional difficulty with learning a programming language is the development environment; learning to use an integrated development environment (IDE) can sometimes be as complicated as grasping the basics of a language.

This paper discusses a learning structure where the students are introduced to programming fundamentals within a familiar and enjoyable development environment – the commercial Minecraft game. Students typically have an awareness of gameplay, controls or at least the goals of the game, thereby removing some of the additional learning requirements of the development environment. The use of the game provides motivation for the student to not only learn but also apply and synthesise their knowledge, thereby achieving the higher orders of Bloom's Taxonomy. Providing a correlation between learning environment and personal game experience, students developed skills that they identified as enhancing their own game experience and showed some desire to replicate and expand on the content delivered.

Foundational skills are developed focussing on variables, assignment, selection and looping within a widely used scripting language. This program has been used to promote programming and computer science to students across a range of ages (8 – 16 year olds), skill levels (typically developing and gifted) and from various educational sectors (state, religious, and Montessori).

Keywords: Computer programming education, development environments, Blooms taxonomy, eLearning, games, emerging technologies.

Introduction

Teaching students the fundamentals of programming can be a difficult task. Compounding this can be a number of factors ranging from age, culture, socio-economic background, capacity, development environment, and motivation. The ability to keep a student interested in what can sometimes be described as stale syntax and semantics can be challenging. By presenting the fundamentals of programming in an accessible, entertaining forum that students have a vested interest in, can potentially aid in not just the understanding of the language but also in its application and evolution into creative expression and individual investigation.

This paper presents a summary of one of a range of workshops that are conducted as outreach activities promoting computing and programming. The tasks that the students are guided through are identified and the results of the process, in the form of participant feedback, are presented. By identifying the creativity and application of ideas presented, the paper will demonstrate the benefit of an easy to use development environment as well as a motivation for using the application.

The paper will survey relevant literature that has guided the development and philosophy behind the workshop. Discussion will continue by identifying the features of the world and the workshop tasks. The presentation of participant feedback and student progression follows and provides justification for the types of cognitive development, retention and adaptation as identified in traditional pedagogical systems.

Background

Computer games have grown to become a fundamental part of many people's social environment. Educators already make use of "world to the desktop" environments that include web browsers, document editors, information and collaboration. However there exists another interaction model within video games (Oblinger, 2006). Rather than static views, games provide users with a different experience, one that is immersive for the user and where the user interacts with abstracted agents and artifacts.

Work has also shown that when offered the choice between a useful but staid technology students will gravitate towards more hedonistic "enjoyable" experiences over of the more "useful" alternative (Van der Heijden, 2004). A key factor is the perception of ease of use for the student, a task that is perceived as enjoyable benefits from an increase in willingness of the student to use the system. This suggests that immersing students in a "fun" activity, in this case playing Minecraft, can lead to a quicker and more organic acceptance of the learning environment. Additionally the increase in time spent playing video games directly affects the time students dedicate to other tasks, especially in relation to a student's willingness to attend classes / perform homework. Video games not only displace older activities like watching television but also show the impact playing can have in relation to time spent on education tasks (Ward, 2012). A way of using this "game time" without sacrificing "education time" presents an opportunity to explore.

One video game possibility, Minecraft, has already seen investigation as a tool to aid education. Work has been done to explore the use of Minecraft as a catalyst for generating interest in STEM topics (Van der Heijden, 2004), as a way of introducing young children to programming using a block based visual mod for the game (Zorn et al.), and a school in Sweden has made use of the game compulsory as a tool to aid thinking and collaborative skills of young high school students (Gee, 2012).

However the increase in modifications that expand the Minecraft game world ("mods") have increased the depth of possibilities for the game and one that stands out as a possible tool for engaging and teaching detailed programming is ComputerCraft (dan200, 2013). ComputerCraft uses the LUA scripting language, a dynamic language designed to be small, portable and fast to control "turtle" objects that can be instructed to move, build and interact deeply with the world (Ierusalimschy et al., 2007). Code can be written externally and imported in or internally using the games own simple editor allowing the use of IDE's or editors. The ComputerCraft community has already used the mod to not only perform standard game play tasks but also to develop custom libraries, create ComputerCraft operating systems with functional GUI elements and complex GPS structures using turtles.
However one of the issues with independently developed software is an adherence to educational pedagogy. To frame Minecraft and ComputerCraft as an educational tool we looked to apply the revision of Blooms Taxonomy (Bloom et al., 1956) to the game workshops in an attempt to show its potential as educational software and how the workshop uses the game to achieve educational goals.

Published in 1956 Bloom's original taxonomy looks to classify thinking behavior with regard to the act of learning. The original taxonomy consisted of three domains:

- Cognitive Domain: knowledge based domain with six sub levels
- Affective domain: attitudinal based with five sub levels
- Psychomotor – physically based, Bloom provided no sub classes however later work looked to classify education objectives within the psychomotor domain to bring it in to line with the other two areas (Simpson, 1966).

As we seek to validate learning outcomes of an educational process the cognitive domain is the most relevant and Blooms original taxonomy sought to define the educational categories of the Cognitive domain as:

I. Knowledge
II. Comprehension
III. Application
IV. Analysis
V. Synthesis
VI. Evaluation

For educators often the most important fields in learning outcomes concern an overlap of the fields from Application to Synthesis, and the taxonomy views the progression through the domain as hierarchal, the student reaches Evaluation only by progressing through the previous five categories. As curriculum advances have focused more on the later portions and the sub categories of the knowledge field it has been necessary to revise Blooms taxonomy to more closely align with advances in developmental psychology. To this end a revised edition of Blooms taxonomy has developed to allow the application of the taxonomy in a two dimensional way (Anderson et al., 2005).

In this revision, language and concepts of the original taxonomy are updated, but the dimension of the original is also expanded to provide a two dimensional framework. As educational outcomes are often associated with dual noun/verb objectives, by expanding the taxonomy to allow educators to separate the act and the process a more nuanced goal can be achieved (Krathwohl, 2002). This allows separation of the taxonomy into cognitive and knowledge domains.

This revised taxonomy now employs the Knowledge domain as one that is independent of subject matter, made up of three of Blooms original fields and one new:

- Factual knowledge: knowledge of terminology, specific details and elements of the task
- Conceptual knowledge: how elements of the task behave and function within larger structures
- Procedural knowledge: "how" to do something, the techniques and algorithms involved
- Metacognitive knowledge: the new addition focuses on the "knowledge of knowledge", awareness of one's own cognition. Specifically strategic knowledge, knowledge about cognitive tasks and self-knowledge (Pintrich, 2002).

The cognitive domain remains similar to Blooms original six fields, modified to more closely represent the "verb" nature of the domain:

I. Remember
II. Understand
III. Apply
IV. Analyze
V. Evaluate
VI. Create

The revision retains a loose hierarchal structure, latter does not require former but remember is viewed as less complex than understand, in turn less complex than apply etc. and some rankings are swapped; Create (Synthesis) and Evaluate.

One of the key benefits for this two dimensional approach is the way it lends itself neatly to a table format running the Knowledge dimension against the Cognitive. Table 1 provides an example of the type of use of the two dimensional representation for learning outcomes using the revised taxonomy. The educational developers would be able to identify the tasks at each intersection.

Table 1: Example of the table for the Revised Blooms Taxonomy (Krathwohl, 2002)

	The Cognitive Domain					
The Knowledge Dimension	1.Remember	2.Understand	3.Apply	4.Analyze	5.Evaluate	6.Create
A. Factual Knowledge						
B. Conceptual Knowledge						
C. Procedural Knowledge						
D. Metacognitive Knowledge						

Lastly when employing the revised taxonomy additional care must be made to ensure that the information provided to students aligns with the educational goals sought. The educator needs to validly and reliably assess the higher order cognitive processes. One of the keys to link assessment directions and the assessment itself is the proposal of using the cognitive verb itself within the task specification (Airasian and Miranda, 2002) and care should be taken to avoid ambiguous definition that may cover multiple cognitive domains; "State the…", "List the…", "Demonstrate that...", etc. By aligning the language of the task with the objectives of the taxonomy it becomes easier for the educator to create relevant assessment tasks and also to evaluate the work the student has done with a clear view of the educational goals sought.

Workshop Background

The actual workshop that is employed has been provided to a large array of students (500+) from a range of backgrounds and interests. The workshop (or variations of it) has been delivered to children from the age of 8 through to 16 years of age. It has been used it to teach children from low socio-economic and disadvantaged backgrounds as well as the opposite end of the spectrum. The workshop has also been used to teach children who are part of a gifted and talented organization with both strong scientific and creative minds. The students who take part may be interested in engineering or computer science or they may be taking part as a wider outreach program and have little interest in programming or computers.

The initial intention of the workshop was to develop an outreach activity for the School of Computer Science, Engineering and Mathematics, at Flinders University that would motivate students to think differently about programming and computer science. This promotion of programming and the creativity and fun associated was used to demonstrate at a fundamental level the types of tasks a student programmer needs to consider as part of their studies.

As the activity was based in the Minecraft game there were a number of students who were interested in the game itself compared with the concepts we demonstrated. This interest in the game proved to be a significant motivator for the students to progress and complete the tasks.

Typically the workshop was run on a PC or laptop that may be used by an individual or a pair of students. A standard keyboard and mouse were used to provide the interaction, while low-end computing hardware was sufficient to effectively conduct the workshop. Depending on the computer networking resources available, all the students could play in the "one world" where they log into a hosted game, or they could use individual, isolated worlds deployed to each machine. The benefit of one shared world is the social and communication possibilities, while the isolated worlds remove competitive anxiety from the students.

Workshop Goals

The goal of the workshop was to provide students with an alternative introductory programming experience to standard practice. Many fundamental activities typically focus on text processing to explore language syntax and semantics concepts. By providing a graphical and entertaining medium it was hoped that the students would see this as an "easy" activity.

The tasks were designed to exercise the various levels of the cognitive domain of the revised Blooms Taxonomy. We sought to ensure the students could understand the material, this was achieved through demonstration, discussion and practical application. Once the students had been able to apply their knowledge we then had them analyse a complex, abstract problem

and evaluate potential solutions. Finally, the students were asked to take these developed skills and create their own implementations to prescribed problems.

This short process allowed the students to engage with all levels of the cognitive domain and thereby achieve better knowledge of the concepts. As discussed later in this paper, many of the students were able to continue this progression and apply and create additional programs for their own purposes.

The Minecraft World

The workshop was run in an already developed and populated Minecraft world. The world was not inhabited by any game enemies and there was no Minecraft game "goal" associated with the workshop. The world was relatively small with the player beginning in a light forest, with a multi-story school building to their right and a workspace to their left, see Figure 1 for a bird's eye perspective of the world. The red arrow head represents the players starting location. The brighter regions are areas lit by torches which are used (in combination with in-game signage) to guide the player to their appropriate destinations.

Figure 1. Bird's eye view of the world

To teach the programming concepts a freely available mod was used. The mod, called ComputerCraft, allows the player to place computing equipment within their Minecraft worlds. This computing equipment includes monitors, disk drives, floppy disks, modems, and "turtles"; robotic assets that can be programmed by the player, the name derived from constructs of programming language interfaces like Logo, (Solomon and Papert, 1976). The ComputerCraft mod allows the player to program some of these assets in a scripting language called Lua (Ierusalimschy et al., 1996). The LUA language can be somewhat forgiving with the syntax used and the structuring of the code. For novice programmers the roadblocks of syntax and structure tend to obscure the retention of the programming language concepts taught. When a syntax error occurs a novice programmer may not be able to identify the problem or the potential solution. By providing an instruction list and a well-paced development of skills the issue of syntax can be predominantly avoided while still allowing the student relative freedom to develop their own work.

The world was populated with the necessary components to complete the workshop tasks. In the work area there were a range of turtles that are preloaded with the necessary components to complete the tasks. Signs to instruct the student about the overall concept of the task, were

suitably placed within the world. The teacher then provided additional content to direct the creation.

In the school building there were a number of rooms to teach students about other aspects of programming as well as computer networks and gameplay. Behind the school building was a target range and castle defence area.

The world itself was set up for the specific task intended for the workshop but we encouraged the students to explore the potential uses of the techniques that were taught within their own personal games they played at home. The documentation and world itself was accessible to the students that attended the workshops so they were able to use this as a reference later if necessary.

Minecraft Workshop Activity

The players began their session in the light forest and followed the path indicated by the torches, Figure 2 shows the initial player view when first starting the workshop. This first task may seem redundant but it was found that some of the students had not played Minecraft before and so this presented the opportunity to describe the controls and functionality of the game itself.

Figure 2. Initial Player view of the world

Figure 3. Turtle operating system and place statement

Figure 4. Result of the place statement

Figure 5. Sequence of statements written to a file

Once at the work area the students were asked to represent a large number using a Montessori based tool: the golden beads. The students were told to start with the units. This introduced the students to the programming interface, the functionality of starting up the console in the turtle, creating the necessary program file and what a programming statement is, Figure 3 and 4 show screen shots of the command line interface and the result of placing a block. The students entered the command line environment, created a file and then programmed the necessary statements to instruct the turtle to move forward and place a block. They then

expanded on this by introducing 90^0 turns to place all the required units, Figure 5 demonstrates the list of statements for this functionality.

For the next task the students created the code to place sticks of ten blocks. This demonstrated the code required for repetition structures, Figure 6 and 7 show the code as well as the result of the execution. By extending the concepts covered in the previous step the students learning tasks were scaffolded to provide them with reference and structure. A discussion took place indicating how they could encapsulate the statements for placing blocks within a loop structure and repeat this process a number of predefined times. A breakout discussion occurred at this point identifying the usefulness of the loop and the potential resource and time savings afforded by such structures. The new program sequence was placed on a new turtle. The reasoning for this was to provide the student with the means to analyse the core functionality of their code without the distraction of other elements.

Figure 6. Example of the tens loop statement

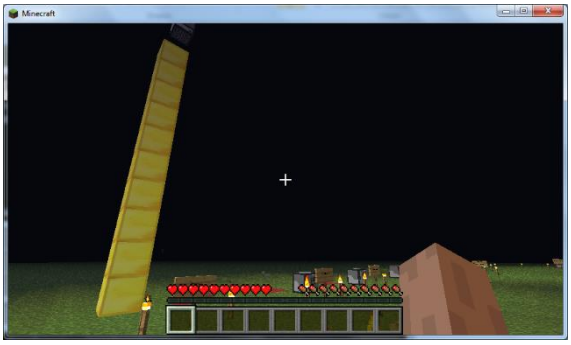

Figure 7. Result of the tens loop program

Once the student had practiced the creation of a tens stick, they were then asked to extend this functionality by creating hundreds squares. To complete this task the student was required to nest their loops. By nesting these structures they could essentially iterate through rows and columns and create a square rather than one column. Another pause was taken here to instruct the student, explaining the usefulness of this type of statement and the extensibility potential for these code structures.

The last task at this point was to have the students write the necessary code to define a cube of blocks that represented one thousand. By extending their understanding of statements, and loops the students were able to define complex nested structures.

The next guided task was to have the student program a turtle to create a castle. The castle was a simple construction of four walls with crenulations across the top. This task had the student write statements to prompt the player for input for the height and length of the castle walls. To achieve this the student developed an interaction script within their program that stored the user input in variables. The task introduced the use of selection to test whether an additional block was required to be added to the top of a wall section to provide the crenulation shape.

Dependent on the age of the students, the time required to complete the tasks and discuss aspects of the code typically took approximately one hour. Again depending on the requirement of the workshop the students were then provided with the opportunity to take the content they had worked with and apply it to a new scenario. The students were told to use the firing range behind the school and develop a bombing program that would allow their turtles to drop explosive TNT with the intention of destroying all the targets. It was typically

at this point where the students got to be creative and utilised the programming statements for destruction rather than creation. The students demonstrated an increased level of excitement and enthusiasm at this task, Figure 8 and 9 show the initial code developed by a student and the resultant destruction of the world.

The description of this workshop identified the core, fundamental requirements for developing code within the Minecraft game. Follow up workshops were conducted where additional features such as networking and exploration were described.

Figure 8. Initial code for the bombing task

Figure 9. Result of the bombing code

Post workshop

Many of the students that took part in the initial workshop returned for subsequent workshops. These follow-up workshops usually took place at least one week after the first session. Students had developed their own solutions and many were so excited by their work that they brought their own laptops to demonstrate their achievements. The complexity of their developments demonstrated not just the acquisition of the language but the application and ability to create. As Bloom's revised taxonomy indicates this creative process demonstrates a higher level of cognitive knowledge.

The students' drive to adapt the knowledge and create their own usages for the concepts taught was driven primarily by their enjoyment of the Minecraft game. By blending this enjoyment with a technical process the students were able to enhance their skills. In follow-up workshops many students identified their own strategies for solving the presented problems. It was soon apparent that the students had engaged with the material and were providing unique solutions that had not been considered by the teaching team.

Given the extensions that students were delivering themselves, the follow-up workshops began to evolve into challenge-based tasks. An abstract problem was presented to teams of students and they were required to design and implement an appropriate solution based on their own knowledge of the language and their usage of the development environment.

Many students indicated their enjoyment of the tasks and were keen to detail their plans for their own implementations and code design.

Student feedback

Students were asked about their experience both during and after the workshop discussed in this paper. Of the feedback collected (14% of all attendees), the majority of the responses to the question asking about their satisfaction with the workshop was positive. Many students expanded on this with comments suggesting how they may use these skills in their own games.

The students were also asked about their previous experience with programming; approximately 30% of the students had programmed before but this was usually with visual programming languages like Scratch or Lego NXT. A follow up question asked them to describe their perception of the functionality of the development environment. All of the students who had programmed before conveyed a confusion in the development environment controls and indicated that they did not use all of the features.

Students were asked whether before attending the Minecraft workshop they were considering a career in computing. Approximately 50% of the students (primarily female) were not considering computer science as a career choice. Comments from students who were not interested in computer science suggested that they were not aware of the creative and "fun" side of computing. The students were asked whether they had changed their mind as a result of the workshop and of those not interested 69% changed their mind.

The students were specifically asked about the knowledge they felt they had gained from the workshop. Questions asked them to describe their understanding of variables, loops and selection. In their own words the majority of students were able to describe what these structures were, their functionality and examples of their implementation.

An abstract problem was also presented to the students and they were asked to describe their own solution using "pseudo code". 88% of the responses defined an appropriate solution. These suggested solutions demonstrated the students' understanding, application and synthesis of the materials taught.

Approximately 30% of the students who completed the workshop presented in this paper answered questions in follow up workshops. This number is not a true indication of returning students but represents those who wished to provide additional feedback. These students were asked whether their knowledge of programming languages had increased. All of the students responded with yes.

The returning students were asked about their use of the development environment within Minecraft and while some indicated a level of complexity with starting the environment, 71% said it was easy to use. 91% of returning students said that the experience was enjoyable. 57% of returning students indicated that they had been developing their own code at home and had sought to expand their knowledge and skills.

Finally the returning students were asked whether they had considered looking at other more complex and robust programming languages. The same number of students who had worked at home reported that they were interested in developing their understanding and knowledge of additional programming languages.

Results from the initial survey and the follow-up survey, suggest that the students enjoyed the experience and increased their knowledge. The results are based off of workshop feedback forms that are designed to enable descriptive, subjective feedback rather than quantitative results.

Conclusion

While the workshop continues to be used as an outreach activity it has identified strategies and environments that benefit the learning objectives for young programmers. Students were motivated to participate in the lessons as they saw a direct relationship to their own personal

interests. The use of the ComputerCraft mod allowed most students to experience an engaging and familiar environment. Most students enjoyed the experience and felt their knowledge had increased.

It is expected that future work would look to assess pre and post experiment knowledge. This information would be used to indicate the level cognitive knowledge gained across a number of sessions. The workshop is also being developed into a series for use within school computing lessons.

Finally, the success of using such an approach to improve the retention and application of programming knowledge has been acknowledge by the comments from the students who participated as well as anecdotal and observational evidence.

References

AIRASIAN, P. W. & MIRANDA, H. 2002. The role of assessment in the revised taxonomy. Theory into practice, 41, 249-254.

ANDERSON, L. W., KRATHWOHL, D. R. & BLOOM, B. S. 2005. A taxonomy for learning, teaching, and assessing, Longman.

BLOOM, B. S., ENGELHART, M., FURST, E. J., HILL, W. H. & KRATHWOHL, D. R. 1956. Taxonomy of educational objectives: Handbook I: Cognitive domain. New York: David McKay, 19, 56.

DAN200. 2013. ComputerCraft | Programmable computers for Minecraft [Online]. Available: http://www.computercraft.info [Accessed 24 August 2013].

GEE, O. 2012. Swedish School Makes Minecraft a Must [Online]. Available: http://www.thelocal.se/45514/20130109 [Accessed 24 August 2013].

IERUSALIMSCHY, R., DE FIGUEIREDO, L. H. & CELES FILHO, W. 1996. Lua-an extensible extension language. Softw., Pract. Exper., 26, 635-652.

IERUSALIMSCHY, R., FIGUEIREDO, L. H. D. & CELES, W. 2007. The evolution of Lua. Proceedings of the third ACM SIGPLAN conference on History of programming languages. San Diego, California: ACM.

KRATHWOHL, D. R. 2002. A revision of Bloom's taxonomy: An overview. Theory into practice, 41, 212-218.

OBLINGER, D. 2006. Games and learning. Educause Quarterly Magazine, 29, 5-7.

PINTRICH, P. R. 2002. The role of metacognitive knowledge in learning, teaching, and assessing. Theory into practice, 41, 219-225.

SIMPSON, E. J. 1966. The Classification of Educational Objectives, Psychomotor Domain.

SOLOMON, C. J. & PAPERT, S. 1976. A case study of a young child doing turtle graphics in LOGO. Proceedings of the June 7-10, 1976, national computer conference and exposition. New York, New York: ACM.

VAN DER HEIJDEN, H. 2004. User acceptance of hedonic information systems. MIS quarterly, 695-704.

WARD, M. R. 2012. Does time spent playing video games crowd out time spent studying? . 23rd European Regional Conference of the International Telecommunication Society. Vienna, Austria.

ZORN, C., WINGRAVE, C., CHARBONNEAU, E. & LAVIOLA JR, J. J. Exploring Minecraft as a Conduit for Increasing Interest in Programming.

iafor