An investigation of a game generator tool to teach recursion

Shakeel Mohamed
CS Honours UCT
mhmsha056@myuct.ac.za

ABSTRACT

Recursion is an important and powerful computational problem-solving tool however many students find it hard to understand conceptually and hard to utilize in a practical coding environment when applied to a problem. This literature review aims to investigate a game generation software tool that can produce a game, based on specific input parameters, which will serve as an interactive and fun educational assignment generator to teach the concept of recursion to those being introduced to the topic, with particular focus on computer programming. Using a game as a form of an alternative assignment may help facilitate the learning experience and provide learners with a deeper and more intuitive understanding of recursion. We aim to investigate the best way of structuring a game that has a balance between actual programming skills (ie. the ability to code recursively) and understanding given recursive code. The purpose of the game is to promote the understanding of the topic of recursion and encourage self-learning.

1. INTRODUCTION

Recursion is one of the most important programming concepts in computer science. It allows the creation of extremely simple algorithmic solutions to certain problems that would otherwise be unsolvable or inefficient with any other type of approach. It is a fundamental concept in computer science, whether understood as a mathematical concept or programming technique [24]. It is regarded as a challenging topic to learn for students being introduced to the world of computer science. Educators often find it a difficult topic to teach as well [18]. Most people are first introduced to an iterative or loop based style of programming to solve problems, which share many similarities with recursion and this often bewilders the beginner. It's applications in computer programming cannot be understated and many students fail to grasp the concept as it is taught in lectures and textbooks and thus may find it difficult to cope with more advanced topics taught later in many courses. Many classic examples are taught such as factorial, Fibonacci numbers, towers of Hanoi and binary search. However simply learning the code or algorithms do not promote or illustrate the concept of recursive thinking in a visual or interactive way. Visualization is a highly efficient method for demonstrating difficult to learn concepts. It is well known that concrete conceptual models are better than abstract conceptual models [23]. Being able to see and understand when and how each recursive call is executed can be invaluable to one's understanding of recursion but this can be taken further.

Using games to teach subjects and programming concepts is not uncommon and has proven to be a fun and engaging way to promote self -learning [3]. Teaching programming through a game is something that has been explored however very few are specific to recursion [21, 22, 3]. Few attempts have been made to gamify the concept of recursion especially when the payoff for achieving this can be tremendous [18]. Being able to easily teach such a challenging topic and promote interest in it could substantially reduce the initial confusion many people have and remove the misconception that recursion is a daunting method of solving problems where an iterative solution could be easier (and possibly slower).

2. VISUALIZING RECURSION

Recursion is a method of problem-solving that involves the decomposition of a problem into subproblem(s) of the same nature until a base case is reached. The composition of these problems solves the original. Many students fail to understand the "passive flow" of recursion and the use of the stack for backtracking [11]. More specifically, they visualize recursion as a loop structure and each recursive call as iteration, which is not true. It is well known that visualization can play an important role in understanding abstract concepts [23]. Visual analogies of recursion do exist and research into real-life examples of recursion has been done as a number of papers have been written in this regard. Examples such as parking cars [1], delegating tasks [2] and the Cargo-Bot game [3] aim to give context to recursion in the real world. Pirolli and Anderson [4] claim that the lack of analogies for recursive problems is what makes it difficult to learn. Kurland and Pea [5] discovered that students often develop an incorrect mental model of recursion through standard classroom and textbook learning. Kahny and Eisenstadt [19] examined novices' judgments of given recursive programs and concluded that they developed one of several mental models of recursion, which they named "copies", "loop", "odd", "null", and "syntactic magic". All of these models except for the "copies model" are regarded as incorrect models of recursion.

Many conceptual and concrete models have been used in introducing recursion. Some have been listed below.

- Russian Dolls [6]. A Russian Doll can be taken apart into many smaller dolls of the same shape. It illustrates the process of taking a smaller version of itself (the problem) until the base case is reached (the final doll that does not contain another).

- Process Tracing. This approach focuses on tracing the process generated by recursive functions.

- Stack Simulation [8]. Calls to functions are traced with explicit reference to the system stack mechanism that is used when implementing recursion in a programming language.

- Mathematical Induction [9]. This approach introduces recursion in terms of the mathematical basis for its correctness. ie. Proof by induction.

- Structure Template [10]. This model provides novice programmers with samples of recursive programs and describes the base cases and recursive cases.

3. EDUCATION USING GAMES

There is an increasing interest and demand for games or elements of games to support education. There is a "new" generation of learners. By new generation we refer to those students who have daily access to interactive 3D games and who spend a significant amount of time exploring them. The gamification of education is a topic that has sparked a lot of interest at all levels of education and for good reason. Gamification, is the use of game design elements in non-game contexts. The concept of gamification is different from that of an educational or serious game. While the latter describes the design of full-fledged games for non-entertainment purposes, "gamified" applications merely employ elements of games. Games promote self-learning, deeper understanding and exploration in the topic. Oftentimes educators do not see games as a learning tool as there is a stigma attached to it where games are simply a past-time or a distraction. While this can be true, it has also been proven that when incorporated into educational topics, games serve as a tool to motivate learners to engage in the material taught to them [26]. The current generation of learner wants to be challenged and often being seated in a 45-minute lecture simply listening or reading a textbook simply does not provide this [27]. It's not interactive enough. Games are empowering, motivating and value the efforts of the player. It's a platform for problem solving where a goal is clearly defined and obtainable. The reward for completing the goal and "beating" the game contributes to this engagement. Unfortunately, creating highly engaging, full-blown instructional games can be difficult, time consuming and costly [16]. For the field of computer science however, there can be a big divide between the regular assignments' students receive where they are required to code in a real IDE and games designed to teach programming which don't require any code at all such as Cargo-bot [3]. Because of this, educational games remain as forms of extra practice rather than alternative assignments. This wouldn't usually be a problem but students would more likely engage in material that has an effect on their grades. Extra practice is something often disregarded.

4. GAMES AND PROGRAMMING

Animating the evaluation of programs, visual programming languages and games to teach programming do exist and are known to significantly help with the learning process [23], however games to teach recursion specifically haven't been explored. Designing educational games requires a different focus than general game design; otherwise, we may fall into the trap of designing fun games with no learning value. Then the designers create the code that best illustrates the target concept. Only after the concept and the target code are designed do the designers begin to develop a game that wraps the concept and the code in a game mechanic that works as a metaphor for the entire game as well as the coding concept. After the overall game concept is defined, the designers then tailor the game instructions to support students in writing code and learning concepts. Game instructions include both how to play and write in-game code as well as educational instruction and information about the game content. Scaffolding code, that is, pre-written code provided to help students get started, is designed at this point as well, although we try to limit the overall amount provided to reduce complexity for introductory students [25]. The alternative is to simply provide the entire recursive code and require the student to interpret it correctly to solve the maze or puzzle presented to them, which is a visual representation of the code provided. These test the students understanding rather than their ability to write correct code. A balance needs to be found between these.

Recent societal changes have caused both a need and an opportunity for a new approach to teaching programming. The need is caused by plummeting enrolments in computer science. Between 2000 and 2005 there has been a 60 - 70% reduction in incoming freshman computer science majors [12]. This drop makes retention especially important. The goal is thus to attract and retain majors without "watering down" the technical content of CS classes.

Since the early 1990s there have been multiple efforts to use end user video game creation in an effort to teach programming. Examples of these game creation tools include Alice, Scratch, and AgentSheets [13,14,15]. Scratch in particular is interesting as it makes use of colourful coding "blocks" as opposed to actual textual code, which is visually appealing. Users are able to intuitively connect blocks to produce executable code. These blocks represent various coding constructs such as loops and variables, but also sprites and characters that can "act out" the code constructed.

MUPPETS (Multi-User Programming Pedagogy for Enhancing Traditional Study) is a game where students develop and interact with visible 3D objects in the game world [20]. Using Java, students can edit existing code, create new code, compile and run

their code and have direct feedback in the form of compilation errors or, in the case of the code being correct, have the changes appear in the game. The problem with this approach is that it barely differs from traditional coding assignments and simply adds a visualization to the students completed code.

5. GAME GENERATION

Previous approaches to automated game generation have focused on hand-designed game-specific mechanics and generating content for fixed sets of hard-coded mechanics. As a result, most prior work on game generation revolves around selecting and assembling components from human provided knowledge and content [3].

Game generation can be very useful especially for generating problems because most of the time, the output will be unique and have a unique solution. This puts emphasis on the understanding of the core concept used to solve these problems as opposed to memorization of the process. This can be particularly useful when assigning generated problems to learners as generally no two learners will have the same problem at hand. Of course, this depends on the complexity of the game generation algorithm and input parameters.

A prime example of a game generation tool is Game-o-matic [17], which creates games that represent real life ideas. It makes use of an input map system and networks of nouns connected by verbs to generate a game with simple arcade mechanics.

A game generation tool would require various input parameters that will limit the complexity of the game level generated. In the context of generating a game to teach recursion, the aim is to develop a tool which, given input specifications, will generate a level or maze-like problem that the player will need to solve or navigate using recursive thinking and a recursive solution.

5.1 MAZE GENERATION

There are two basic ways to generate mazes. Algorithmic and non-algorithmic. Algorithmic methods create mazes according to a predefined step order. One or more steps need to be randomized. That is, the function which decides the step needs to return a random result. Graph based maze generation algorithms create mazes by building a spanning tree. The end of this process generates a spanning tree corresponding to the maze created. Kruskal's algorithm and Prim's algorithm are examples of these. The size and shape of the maze should also be taken into account and can be specified as an input parameter when generating the maze.

6. DISCUSSION

It's clear that games and gamification in education is the way to go when it comes to promoting learning content and maintaining enthusiasm and interest. Visualization is the key factor in engaging the learners with the abstract concepts taught. There are many examples of teaching programming using games and visualization and we know that they are useful for engaging students in the material but it's interesting that recursion, a challenging topic, hasn't really been attempted to be gamified or incorporated into a game where the payoff for achieving this could be huge. Recursion is often taught in the early stages of CS education and provides a basis for more challenging topics, thus it's very important that students are able to easily understand it. This also promotes the retention of students in the field and will likely improve pass rates. There appears to be two approaches to this problem. The first approach involves a hands-on approach from the student where they are expected to write recursive code in order to solve a specific problem, maze or puzzle. An example of this is the game "EleMental: The Recurrence" [25]. In EleMental, a code editor is provided as well as a simulated game world that visualizes the execution of the written code. The player is expected to edit the pre-written code in order to traverse a tree in game. The problem with this approach is that it puts too much emphasis on the written code and not so much the game itself. You aren't actually playing the game, rather watching the execution of your program visually. The other approach involves providing recursive code for the student to understand and be able to implement in a game environment such as a maze or puzzle. The player would be expected to control a character in the game world and solve the maze or puzzle using movement controls of some kind in order to match the execution of the pre-written recursive code. The latter approach is more game-focused and involves more playability where the former is more technical. A balance between these approaches needs to be found where playability is present but the game isn't completely abstract and far removed from what would be expected from students in the real world.

7. SUMMARY

There is a clear need for alternative methods of teaching in the computer science field. Lectures are the current standard but there are more ways of engaging students in learning material. Although we know gamification is the way to go when it comes to education, we aren't quite there yet and there are many topics that haven't been explored in a gaming context. Recursion being one of them. Recursion, being one of the first concepts students learn in the early parts of any CS course, is a prime target for gamification as the understanding of it is extremely useful for understanding more advanced topics taught later. There is a distinct lack of research in this area. Teaching programming using games has been proven to be beneficial so the question remains why recursion as a concept does not have enough research done in teaching it using games or incorporating game elements into the teaching process. This literature review aimed to point out that there is a lack of research in this area and where there is, it's not very good or in depth.

We need to find better ways of teaching this challenging topic and a game seems to be the best way of doing so. We know games resonate with students and greatly enhance the learning experience so it seems like a good start. Of course a single game

is not going to be the entire solution to the difficulties students face with this concept, but the benefits for designing a game that's enjoyable and promotes the learning of the concept can have great advantages for students and educators in and out of the classroom.

REFERENCES

1. Wirth, M.: Introducing Recursion by Parking Cars. SIGCSE Bulletin 40(4), 52–55 (2008)
2. Edgington, J.: Teaching and Viewing Recursion as Delegation. J. Computing Sciences in Colleges 23(1), 241–246 (2007)
3. Tessler, J., Beth, B., Lin, C.: Using Cargo-Bot to Provide Contextualized Learning of Recursion. In: Proceedings ICER 2013, San Diego, pp. 161–168. ACM (2013)
4. Pirolli, P.L. and Anderson, J. R'. The role of learning from examples in the acquisition of recursive programming skills. Canadian Journal of Psychology 39 (1985), 240-272.
5. Kurland, D. M. and Pea, R. D. Children's mental models of recursive LOGO programs. In Proceedings of the 5th Annual Conference of the Cognitive Science Society (1983), Session 4,,1-5.
6. Dale, N. B., & Weems, C. Pascal (3rd ea.). Lexington, MA: D. C. Heath, 1991.
7. Greer, J. E. Empirical Comparison of Techniques for Teaching Recursion in Introductory Computer Science. Ph.D. dissertation, The University of Texas at Austin, 1987.
8. Aho, A. V. and Ullman, !. D. Foundations of computer Science. W. H: Freeman and Company, New York, NY,1992.
9. The role of learning from examples in the acquisition of recursive programming skills. Pirolli, P.L. and Anderson, J. R'. 1985.
10. Tamarisk Lurlyn Scholtz and Ian Sanders. Mental models of recursion: Investigating students' understanding of recursion. In *Proceedings of the Fifteenth Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE '10, pages 103–107, New York, NY, USA, 2010. ACM.
11. Vegso, J, Drop in CS Bachelor's Degree Production. Computing Research News, Vol 18, No 2, March 2006,
12. Cooper, S., Dann, W., Pausch, R., Teaching Objects-first In Introductory Computer Science, In Proc. SIGCSE 2003, Reno, Nevada, USA, 2003
13. Peppler, K. & Kafai, Y. B., Collaboration, Computation, and Creativity: Media Arts Practices in Urban Youth Culture. In C. Hmelo- Silver & A. O'Donnell (Eds.), In Proc. Computer Supported Collaborative Learning, New Brunswick, NJ, USA, 2007
14. Repenning, A., Excuse me, I need better AI! Employing Collaborative Diffusion to make Game AI Child's Play. In Proc. ACM SIGGRAPH Video Game Symposium, Boston, MA, USA, ACM Press, 2006
15. The Gamification of learning and instruction: Game-based methods and strategies for training and education, John W. Rice (Department of Learning Technologies, University of North Texas, Denton, TX, USA)
16. Game-O-Matic: Generating Videogames that Represent Ideas Mike Treanor, Bryan Blackford, Michael Mateas and Ian Bogost
17. Eagle, M., & Barnes, T. (2009). Experimental evaluation of an educational game for improved learning in introductory computing. ACM SIGCSE Bulletin, 2009, 321-325.
18. Kahney, H. and Eisenstadt, M., "Programmers' mental Models of their Programming Tasks: The Interaction of Real World Knowledge and Programming Knowledge", Proceedings of the Fourth Annual Conference of the Cognitive Science Society, pp. 143-145, 1982.
19. BIERRE , KEVIN J. AND ANDREW M. PHELPS. The use of MUPPETS in an introductory java programming course, SIGITE 2004, October 28-30, 2004, Salt Lake City, UT, USA.
20. A serious game for developing computational thinking and learning introductory computer programming. Cagin Kazimoglu, Mary Kiernan, Liz Bacon, Lachlan Mackinnon 2012
21. Teaching Recursion in a Procedural Environment - How much should we emphasize the Computing Model? David Ginat. Eyal Shifroni
22. Do Algorithm Animations Assist Learning? An Empirical Study and Analysis. John Stasko, Albert Badre. Clayton Lewis. 1993.
23. McCracken, D.D. Ruminations on Computer Science Curricula. Communications of the ACM. 30, 1: (January 1987), 3-5.
24. EleMental: The Recurrence Andrew Hicks, Katelyn Doran, Graduate Advisor: Amanda Chaffin, Mentor: Dr. Tiffany Barnes
25. Serious Games: Games that educate, train, and inform. David R. Michael. Sandra L.Chen 2005
26. Are Just-In-Time Lectures Effective At Teaching? RB Dannenberg, P Campell 1997