# Puzzle Games: A Metaphor for Computational Thinking

**Bobby Law**
**Glasgow Caledonian University, Glasgow, Scotland**
Robert.law@gcu.ac.uk

**Abstract:** Introductory programming courses often concentrate on teaching students the syntax for a specific programming language and the relevant constructs for implementing sequence, selection, and iteration. A necessary component of teaching programming, a fundamental flaw is the omission of a suitable strategy for teaching students problem solving. The ability to manipulate a programming language from a syntactical perspective leaves the students with a cursory awareness but does not necessarily mean that given a complex problem they will have the ability to produce a suitable solution. Thus it is important to embed within the teaching of introductory programming an integrated approach to problem solving. This can be a challenge within itself as students are not always receptive to the traditional approach of defining the problem, planning the solution, coding and testing the program. Present such problems as a paper and pen based exercise, couple this with the student's perceptions of these problems as irrelevant and a recipe for lack of engagement ensues. Teaching introductory programming to Game Software Development students lead to an interesting observation; while playing various genres of game, when encountering a puzzle, they appeared to consider their options as if mentally processing a strategy or solution before proceeding. Thus a hypothesis was derived that the use of cerebral based puzzle games could be used to introduce computational thinking, its four key techniques and the mapping between these techniques and the basic programming building blocks in a fun and engaging manner far removed from the traditional approach. Game Software Development students were already, unwittingly, implementing the techniques of decomposing the problem, identifying patterns, abstracting out relevant information and developing suitable algorithms to solve the puzzle. This paper will present the initial research identifying the underlying pedagogical issues associated with the use of game based learning in respect of computational thinking and the genres of game considered for use and the pedagogical constructs they exhibit.

**Keywords:** Game based learning, computational thinking, problem solving, programming

## 1. Introduction

Programming and problem solving are inextricably linked, however, Michaelson (2015) proffers that one of the "oldest" approaches to learning programming is "Programming language oriented." A technique of writing code he likens to hacking which does not scale well.

The use of game based learning and in particular puzzle games helps to reinvigorate the link between problem solving and programming in a fun way. Hence making the learner realise that programming can be fun and will provide for a more efficacious and viable learning experience (Čisar et al. 2014).

As educators, do we teach students how to contemplate problem solving? Michalewicz & Michalewicz (2008) suggests not. Michalewicz & Michalewicz (2008) firmly believe that the lack of problem solving skills exhibited by students "is the consequence of decreasing levels of mathematical sophistication in modern societies." Although many different "thinking" strategies have been trialled these have not been rooted in mathematics and as such do not lend themselves to the solving of problems but rather the discussion of problems (Michalewicz & Michalewicz 2008).

It would be appropriate at this juncture to define the terminology used within the paper allowing the context to be set. The following terms will be defined within the subsequent paragraphs: puzzle, computational thinking, problem solving and algorithm.

Is it possible therefore to define the term puzzle? Unfortunately there appears to be no academic consensus on the definition of the term puzzle to the point that exercise and problem are used interchangeably with the term puzzle (Dasgupta et al. 2013). Badger et al. (2012) offers a usable definition of a puzzle as offering a "significant intellectual challenge."

Wing (2014) defines computational thinking as "the thought processes involved in formulating a problem and expressing its solution(s) in such a way that a computer—human or machine—can effectively carry out." The definition hits upon a key concept that students universally struggle with - the conveying of a solution in a way that they can ultimately convert into the required coding constructs for a programming language.

No universal academic definition of the term problem solving exists (Gomes & Mendes 2007). For the purposes of this paper, problem solving will be seen as applying previously learned material to derive a suitable solution (Gomes & Mendes 2007).

The term algorithm is commonly used within Computer Science education but for the avoidance of doubt it will be defined as "a step-by-step procedure which, starting with an input instance, produces a suitable output." (Edmonds 2008).

## 2. Literature Review

In this section the paper will examine the concepts of puzzle-based learning, computational thinking, problem solving and algorithms in relationship to each other and how these concepts can be applied to the teaching of introductory programming courses.

### 2.1 Puzzle-based learning

In an educational sense Michalewicz & Michalewicz (2008) suggest that educational puzzles conform to four principles: generality, simplicity, Eureka moment and entertainment factor. Badger et al. (2012) state that generality is not unique to puzzles and can also be associated with problems. Interestingly Michalewicz & Michalewicz (2008) openly admit that not all of the four principles need be met. The two principles that stand out are the Eureka moment and the entertainment factor. The Eureka moment should offer a sense of frustration, relief (when the puzzle solution is derived) and reward (Michalewicz & Michalewicz 2008). This is somewhat akin to the nature of puzzles found in many popular console games. Without some kind of entertainment factor the puzzle may lose its lustre and thus interest in solving it will wane (Michalewicz & Michalewicz 2008). Another differentiating feature of a puzzle is the fact that it is self-contained i.e. all the information required to solve the puzzle is supplied within it (Badger et al. 2012).

Research by Falkner et al. (2010) suggests that leading technology companies see a relationship between puzzle solving and the ability to solve real world problems. As such they have drawn parallels between the categories hoped for in students with their associated forms of learning. Figure 1 below illustrates this (Falkner et al. 2010).

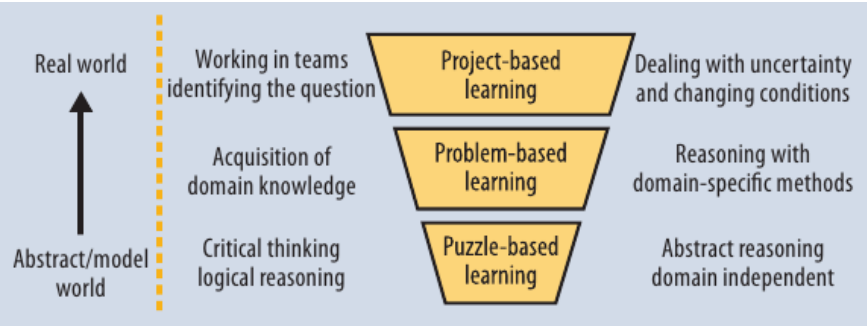


**Figure 1:** Skills categories and associated forms of learning

Figure 1 shows puzzle based learning offering a solid foundation on which to build and develop further problem solving skills which will benefit students and employers.

Badger et al. (2012) proffers that solving puzzles and problems help students:

- Learn to take personal responsibility.
- Adopt novel and creative approaches, making choices.
- Develop modelling skills.
- Develop tenacity.
- Practice recognition of cases, reducing problem situations to exercises.

In relation to programming puzzle based learning helps cement the idea that there can be more than one solution to the puzzle (Badger et al. 2012). Thus more than one way to program a solution, albeit, some programming solutions will be more efficient than others.

### 2.2 Computational Thinking

Cheng et al. (2012) make the bold statement that computational thinking is a "fundamental skill" required by "everyone in the 21st Century." With this in mind it seems a valuable life skill for all students. Wing (2014) suggests that computational thinking encompasses "problem formulation" and not just "problem solving".

Computational Thinking has a number of elements but four can be thought of as its cornerstones: decomposition, pattern recognition, abstraction and algorithm design (Bitesize n.d.). Decomposition involves reducing the problem into smaller more manageable chunks; pattern recognition involves looking for trends, similarities and patterns; abstraction involves removing irrelevant detail and focusing in on the important points; algorithm design involves creating the step by step rules which when followed will solve the problem.

Figure 2 below presents the idea of the cornerstones and the importance of applying them correctly.

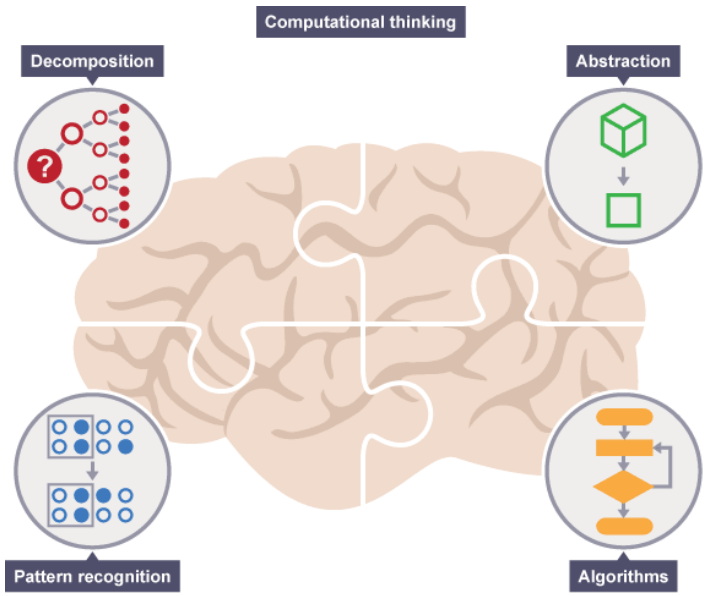


**Figure 2:** Cornerstones of computational thinking (Bitesize, n.d.)

Kazimoglu et al. (2012) contend that computational thinking with respect to computer science education can be categorised by five fundamental skills: "problem solving", "building algorithms", "debugging", "simulation" and "socialisation". Moursund (2006) believes that computational thinking encompasses the idea of "modelling and simulation", both mentally and with the use of a computer.

### 2.3 Problem Solving

In many instances the graduates produced by Universities may well have many skills but it has been mooted that they lack experience of problem solving and therefore "little experience in solving ill-structured, open-ended problems" (Steinemann 2003).

Students learning programming often find themselves in a situation where the problem domain can be too wide and as such they can be intimidated by the problem. Moursund (2006) observes that a facet of problem solving is identifying when the problem is too vague and attempting to remove the ambiguity in order to convert it into a well-defined problem.

Problems can be thought of as straddling two categories: well-structured or ill-structured. Well-structured problems have a set answer in contrast to ill-structured problems which can have an undefined goal or inadequate material. Interestingly it has been suggested that ill-structured problems provide the student with the prospect of using diverse problem solving stratagems (Kiili 2005). One difficulty students encounter when problem solving is defining the steps required to solve the problem and the order these steps should be placed to create the solution (Bachu & Bernard 2014). It is noted by Chao (2016) that programmers are required "to use both problem solving strategies and program development skills".

### 2.4 Algorithms

Wing (2014) describes an algorithm as "an abstraction of a process that takes inputs, executes a sequence of steps, and produces outputs to satisfy a desired goal." Edmonds (2008) suggests that students find the abstract thinking involved in the production of algorithms a difficult task emphasising that "*code* is an implementation of an algorithm." Something that is not always clear to students. This view is echoed by Chao (2016) suggesting that students "lack strategies" for creating algorithms.

Levitin (2005) asserts that the use of puzzles allows the student to apply a level of abstract thinking detached from any form of programming language and strategies for designing algorithms can be considered as general problem solving tools.

### 2.5 Game Based Learning

Game Based Learning is a form of active learning (Bodnar & Clark 2014) and Kiili (2005) states that games provide a "meaningful framework" for posing problems for students to solve. In fact he suggests that games are composed of smaller "casually linked problems" the nature of which can vary and that a problem can be anything that hinders the player before the ultimate goal of completion.

Games can be said to appeal to a wide range of individuals and as such research suggests that they can be used for teaching programming producing "positive effects on students." (Kazimoglu et al. 2012) Kazimoglu et al. (2012) have identified two methods for enabling computational thinking and learning programming: "learning through the exercise of designing games" and "learning through game-play".

The use of commercial off the shelf games as a learning vehicle in the classroom is supported by Van Eck (2006) due to its practicality, however, he warns that there can be drawbacks as this type of material is not designed as teaching material and an analysis of the chosen game with respect to the content to be taught is required.

Games also provide a safe place to fail and retry without a fear of any stigma attached to failing in the normal educational environment hence allowing the student to learn from their mistakes fostering reflection, an essential part of learning (Bodnar & Clark 2014).

Other contributing factors such as motivation and engagement influence the student's problem solving outcomes such that the games being played should offer the student complexity for engagement, autonomy for decision making and attainable challenges for goal achievement (Eseryel et al. 2014). In particular Ross (2002) suggests that puzzle based games are good for enhancing both the student's strategic thinking and motivation.

### 2.6 Summary

The literature reviewed certainly points towards a strong case for the inclusion of puzzle based games for the teaching of programming. It can be concluded that games do provide a good environment for developing "thinking and problem-solving skills" that will benefit the student and help them "gain in mental maturity." (Moursund 2006) Van Eck (2006) makes a valid point when he dispels the idea "that all games are good for all learners and for all learning outcomes".

Melero et al. (2011) point towards the use of puzzle games in the teaching of programming suggesting that code segments can be designed as specific "puzzle pieces" that can be ultimately collated and joined to derive the solution for a proposed program.

Levitin (2005) states that creativity and problem solving are enhanced through the use of puzzles and that puzzles generate more interest for the student thus making them engage with the assigned task. It should also be noted that the definition for Computational Thinking "is defined abstractly at best and covers a wide variety of skills." (Kazimoglu et al. 2012)

## 3. Games

Van Eck (2006) notes three paths that can be traversed when incorporating games into the curriculum. These are: "have students build games from scratch; have educators and/or developers build educational games

from scratch to teach students; and integrate commercial off-the-shelf (COTS) games" This section will investigate both the educator/developer educational games and the off-the-shelf games.

### 3.1 Educator/Developer built Educational Games

This section will consider three games that have been specifically developed as an aid to teaching programming through the use of problem solving. The games discussed in this section have been chosen based on their relevance and coverage of programming topics. The criterion used to select the games was based on a minimum implementation of sequence, selection and iteration and as a secondary consideration a game like user interface. The games presented are by no means an exhaustive list.

The first example is called CodeCraft (Ventura et al. 2015) The concept behind CodeCraft is to provide the student with "a fun, dynamic, and engaging game environment" by offering programming puzzles that have been developed to introduce programming concepts in a staged and measured approach. This is a 3D game environment based on the premise of using a state machine to program the steps required by the robot to solve the presented puzzles. CodeCraft provides five levels of progressive difficulty and doesn't assume that the student has any formal programming knowledge with levels 4 and 5 having multiple solutions. CodeCraft has been designed to teach students: sequence, selection, iteration, variables, collections and computational thinking (Ventura et al. 2015).

The interesting aspect of the puzzles is the use of constraints. These constraints are either environmental or programmatic. Programmatic constraints limit the programming constructs that are available to be used. This requires the student to be more creative in their use of the available constructs when creating their solution. The student uses either action-related or sensory actions to allow the robot to interact with its environment (Ventura et al. 2015).

To solve a puzzle the student uses a state machine interface to connect actions, events and states in an appropriate sequence. The use of a state machine negates the need to have any previous knowledge of a programming language thus there is no requirement for the student to be burdened by the programming language specific syntax. Figure 3 shows an example puzzle and attempted state machine (Ventura et al. 2015).

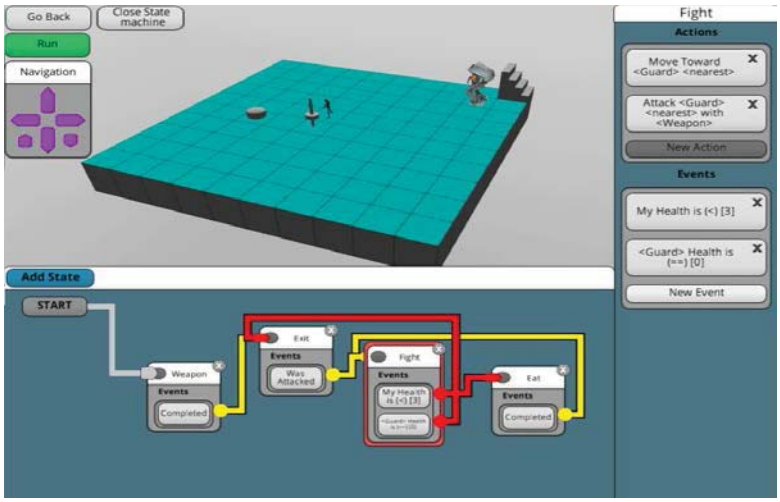


**Figure 3:** an example of CodeCraft (Ventura et al. 2015)

From the perspective of computational thinking this system offers a good environment for the student to learn problem solving while introducing the fundamental building blocks of programming without the student being encumbered by any specific programming language and its inherent syntax.

Kazimoglu et al. (2012) have produced their own software called Program Your Robot designed to teach introductory programming constructs based on controlling a robot to solve puzzles while introducing the programming concepts of sequence, selection, iteration and functions/methods. The authors of this software see it as a serious game introducing students to "algorithm building, debugging and simulation." The game

consists of the student constructing a "solution algorithm" which will help the Robot reach the designated point of each platform. In a similar vein to CodeCraft the solution algorithm is built from two categories of commands: action commands and programming commands. Programming commands are used to implement the programming constructs of sequence, selection and iteration. An interesting concept introduced in this version of the game is the ability to "create repeatable patterns" through the use of functions.

Figure 4 below gives an example of the programming learning environment for the game. On the right hand side of the screen the student has a main method, and two functions. On the left hand side of the screen the student has the two command categories of action and programming. The student can drag and drop any of the action and or programming commands to fill the empty slots within the main method or the two functions. The main method is the primary control mechanism for controlling the robot. When the student has determined that they are finished building their solution algorithm they can use the Run or Debug buttons located at the bottom left of the screen to execute their algorithm. The debug button allows the student to test their algorithm receiving feedback in the form of error/warning messages (Kazimoglu et al. 2012).

As the student progresses through the levels the need to use functions becomes more apparent as the number of free slots in the main method is not enough to successfully complete the level hence reinforcing the need for reusable solution algorithms.

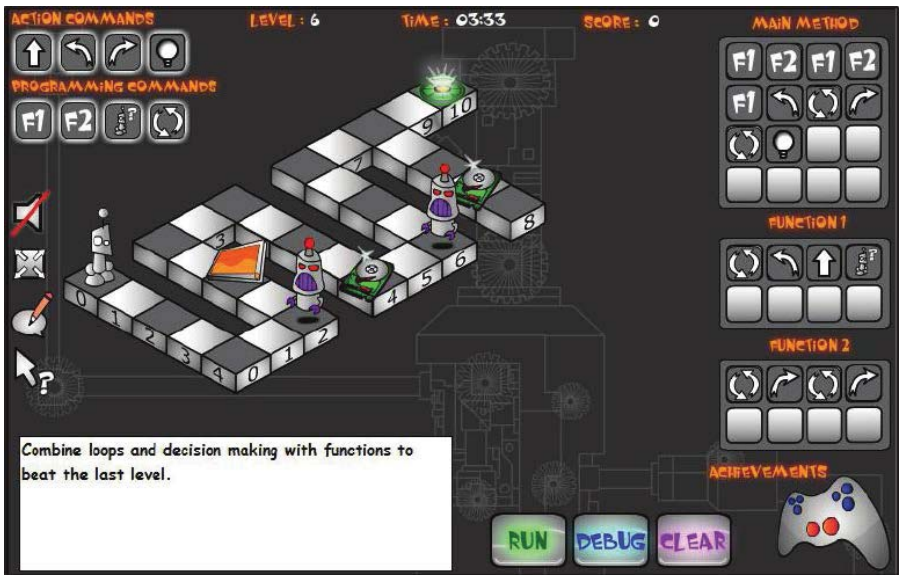


**Figure 4:** Introductory Programming environment Kazimoglu et al. (2012)

As with CodeCraft Program the Robot does not use any explicit programming language or syntax instead relying on the production of solution algorithms thus aligning well with the concept of computational thinking. As an aside the game implements a scoring system which attempts to reward the student with more points the neater their code is.

Like Kazimoglu et al. (2012) Chao (2016) has also developed a game based environment for teaching introductory programming concepts of sequence, selection, simple iteration and nested iteration which employs the use of a robot farmer. The premise of this game is to create a set of instructions that will help the robot navigate the grid avoiding the stones and meeting the objective of picking the flowers for the farmer.

The student uses “instruction cards” which are placed in the relevant cells of the grid to provide the robot with the necessary instructions to complete its task. The student populates the instruction cards using the commands from the “instruction library” which can be selected or dragged and dropped in the “Program Composer” to build a solution. These instruction cards form a “complete computer program” for the robot to achieve its goal (Chao 2016).

Figure 5 below gives an example of the game showing the positioning of instruction cards indicating the decomposition of the problem domain into two sub goals.

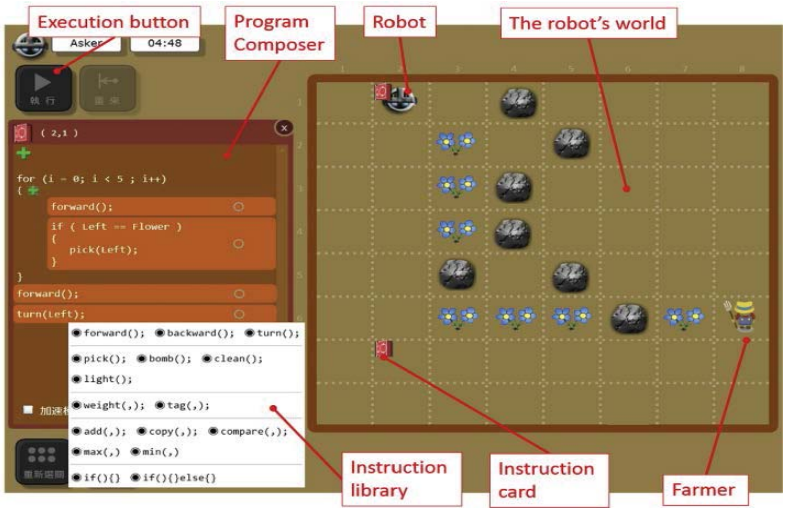


**Figure 5:** Example of Programming Learning Environment Chao (2016)

Chao (2016) game environment differs from Kazimoglu et al. (2012) as it shows the student C/C++ like coding language and syntax and as such this is not as pure an implementation of computational thinking as the previous examples but it still implements puzzle based learning, problem solving, algorithms and game based learning.

### 3.2 Commercial off-the-shelf Games

This section will consider three commercial off-the-shelf games that have been developed as puzzle/logic games for mass consumption and not for teaching as such. The games discussed in this section have been chosen based on their relevance and coverage of programming topics. The criterion used to select the games was based on a minimum implementation of sequence, selection and iteration and as a secondary consideration a game like user interface. The games presented are by no means an exhaustive list.

One example of a commercial off the shelf game that has a strong emphasis on problem solving is Portal and its successor Portal 2 as identified by Shute & Wang (2015) highlighting the fact that the game has "goals and complicated scenarios" requiring the player to discover "new knowledge". Figure 7 gives an example of the type of puzzle available in Portal 2 where the cubes are being used to redirect the laser beams.

Portal consists of a series of puzzles where the student is required to warp time and space with portals in order to solve them. Portals can be placed on a variety of surfaces. As the game progresses various gels are introduced including Repulsion, Propulsion and Conversion gel. These gels offer the student the ability bounce, speed up and turn any surface into a portal compatible surface. Two portals are required to solve puzzles and on completion the student is taken to the next puzzle.

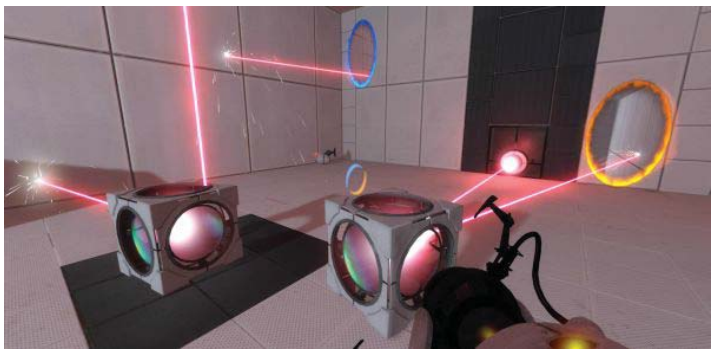

**Figure 6:** Example of a puzzle in Portal.

Another game in the same vain as Portal is The Talos Principle which is a first person puzzle game which offers the player a non-linear world in which the player can create their own path through the game making their own decisions and solving puzzles as they go.

The logic based puzzles present the student with the goal and all the tools needed to get to the goal. The student is required to determine the steps needed to solve the puzzle and thus achieve the goal. Each puzzles title indicates its unique setup and offers a hint to its possible solution.

Figure 7 shows the similarity between Portal 2 type puzzles as seen in Figure 6. Here the student must be careful not to let the red and blue beams intersect when pointing them at the sockets on the wall.

**Figure 7:** Example of a puzzle in The Talos Principal

Again this type of game offers much in the way of problem solving and computational thinking but may well be a bit more obtuse when drawing parallels with programming constructs.

The World of Goo is a physics based problem solving game that requires players to solve innovative and challenging problems (Shute & Kim 2011) requiring the player to use a number of techniques including problem solving.

Essentially the student is required to complete each level by building structures such as towers and bridges using Goo balls that are imbued with the laws of physics. A variety of Goo balls are available each with their own properties and uses. Some of the puzzles can pose quite a cerebral challenge and require precise execution as such this might dissuade some students from participating fully. Figure 8 shows an example of the type of puzzles in World of Goo where the player has to construct a tower using the Goo balls to connect the pipe.

Students apply the basic programming building blocks of sequence, selection and iteration when problem solving and building a solution in this game hence it does offer a degree of computational thinking but it will require the teacher to select the puzzle levels wisely as some can be very complicated.

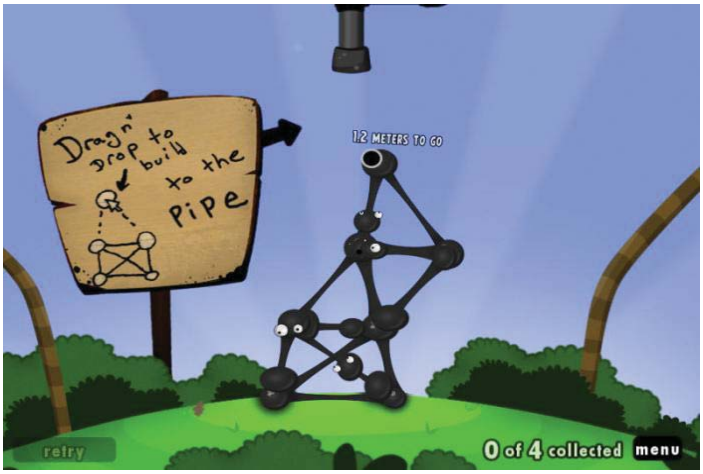


**Figure 8:** Example of a puzzle World of Goo

Van Eck (2006) suggests that the use of commercial off-the-shelf games can be more cost effective in relation to both money and time than the other options suggest at the start of section 3. He further suggests that continued use of commercial games will lead to commercial game makers being cajoled into serious games for education. As a caveat Van Eck (2006) points to the fact that the use of commercial games not a panacea as these games are produced first and foremost as entertainment and not to teach thus an analysis of content is required.

This section has attempted to bring together examples of both educationally derived games directed at the teaching of programming and commercial off-the-shelf games that could be adapted for use as computational thinking and problem solving aids.

## 4. Proposed Work

Teaching games programming to undergraduate students in their second year of study the intention is to have selected labs that will utilise the best aspects of the aforementioned games as a fun and interesting way to introduce computational thinking, problem solving and algorithms.

The strategy will be to start with Portal 2 puzzles to engage the student in the aspects of computational thinking, problem solving and algorithms getting the students to write down the steps taken to solve the problem. Following on from the use of Portal 2 the intention will be to utilise Kazimoglu et al. (2012) Program your Robot available at http://www.programyourrobot.org/ which will help reinforce computational thinking, problem solving and algorithms while enhancing the students ability to debug algorithms.

Gallagher & Prestwich (2013) applied cognitive task analysis to the levels of Portal 2 to determine gameplay and game design in doing so they developed a cognitive map/decision tree which encompassed a number of criterion. Three of the criteria are mechanical steps, cognitive steps and affordances. Mechanical steps involves recording the steps the student took manually or behaviourally to complete the task. Cognitive steps involves recording the cognitive steps the student undertook to complete the task. Affordances are the objects, tools and environment the student interacted with to complete the task. Although only a subset of Gallagher & Prestwich (2013) cognitive map/decision tree they will provide a starting point for formulating an understanding of the students thought process.

## 5. Conclusions

From the literature reviewed there seems to be a ground swell of opinion in favour of using puzzles and game based learning to teach programming at an introductory level. Commercial games such as Portal offer an engagement value that is sometimes lacking in educational software helping to enhance student thinking, problem solving and algorithm development in a non-educational context such that the student will be actively learning in a friendly non-judgemental environment.

## References

Bachu, E. & Bernard, M., 2014. Visualizing Problem Solving in a Strategy Game for Teaching Programming. In Proceedings of the International Conference on Frontiers in Education: Computer Science and Computer Engineering (FECS). p. 1.

Badger, M. et al., 2012. A guide to puzzle-based learning in STEM subjects. University of Birmingham: National HE STEM Programme.

Bitesize, B., Introduction to Computational Thinking. Available at: http://www.bbc.co.uk/education/guides/zp92mp3/revision [Accessed April 2, 2016].

Bodnar, C.A. & Clark, R.M., 2014. Exploring the impact game-based learning has on classroom environment and student engagement within an engineering product design class. In Proceedings of the Second International Conference on Technological Ecosystems for Enhancing Multiculturality. pp. 191–196.

Chao, P.-Y., 2016. Exploring students' computational practice, design and performance of problem-solving through a visual programming environment. Computers \& Education, 95, pp.202–215.

Cheng, R., Annetta, L.A. & Vallett, D.B., 2012. Research Framework for Serious Educational Games: Understanding Computational Thinking in Pasteur's Quadrant. Journal of Information Technology and Application in Education, 1(4), pp.143–151.

Čisar, S.M., Pinter, R. & Cisar, P., 2014. Code hunt—"hunting" to learn programming. In Computational Intelligence and Informatics (CINTI), 2014 IEEE 15th International Symposium on. pp. 329–333.

Dasgupta, D., Ferebee, D.M. & Michalewicz, Z., 2013. Applying Puzzle-Based Learning to Cyber-Security Education. In Proceedings of the 2013 on InfoSecCD '13: Information Security Curriculum Development Conference. InfoSecCD '13. New York, NY, USA: ACM, pp. 20:20–20:26. Available at: http://doi.acm.org/10.1145/2528908.2528910.

Van Eck, R., 2006. Digital game-based learning: It's not just the digital natives who are restless. EDUCAUSE review, 41(2), p.16.
Edmonds, J., 2008. How to think about algorithms, Cambridge University Press.
Eseryel, D. et al., 2014. An Investigation of the Interrelationships between Motivation, Engagement, and Complex Problem Solving in Game-based Learning. Educational Technology & Society, 7(3), pp.42–53.
Falkner, N., Sooriamurthi, R. & Michalewicz, Z., 2010. Puzzle-Based Learning for Engineering and Computer Science. Computer, 43(4), pp.20–28.
Gallagher, P.S. & Prestwich, S., 2013. Cognitive Task Analysis: Analyzing the Cognition of Gameplay and Game Design.
Gomes, A. & Mendes, A.J., 2007. Problem solving in programming. The Proceedings of PPIG as a Work in Progress Report, pp.216–228.
Kazimoglu, C. et al., 2012. Learning programming at the computational thinking level via digital game-play. Procedia Computer Science, 9, pp.522–531.
Kiili, K., 2005. Digital game-based learning: Towards an experiential gaming model . The Internet and Higher Education , 8(1), pp.13 – 24.
Levitin, A., 2005. Analyze that: puzzles and analysis of algorithms. ACM SIGCSE Bulletin, 37(1), pp.171–175.
Melero, J., Hernández-Leo, D. & Blat, J., 2011. Towards the support of scaffolding in customizable puzzle-based learning games. In Computational Science and Its Applications (ICCSA), 2011 International Conference on. pp. 254–257.
Michaelson, G., 2015. Teaching Programming with Computational and Informational Thinking. Journal of Pedagogic Development, 5(1).
Michalewicz, Z. & Michalewicz, M., 2008. Puzzle-based learning, Hybrid Publishers.
Moursund, D.G., 2006. Introduction to using games in education: A guide for teachers and parents.
Ross, J.M., 2002. Guiding students through programming puzzles: value and examples of Java game assignments. ACM SIGCSE Bulletin, 34(4), pp.94–98.
Shute, V.J. & Kim, Y.J., 2011. Does playing the World of Goo facilitate learning. Design research on learning and thinking in educational settings: Enhancing intellectual growth and functioning, pp.359–387.
Shute, V.J. & Wang, L., 2015. Measuring problem solving skills in Portal 2. In E-Learning Systems, Environments and Approaches. Springer, pp. 11–24.
Steinemann, A., 2003. Implementing sustainable development through problem-based learning: Pedagogy and practice. Journal of Professional Issues in Engineering Education and Practice, 129(4), pp.216–224.
Ventura, M. et al., 2015. Development of a video game that teaches the fundamentals of computer programming. In SoutheastCon 2015. pp. 1–5.
Wing, J., 2014. Computational thinking benefits society. Social issues in computing.