Chapter 23
Understanding Computational Thinking before Programming:
Developing Guidelines for the Design of Games to Learn Introductory Programming through Game-Play

Cagin Kazimoglu
University of Greenwich, UK

Liz Bacon
University of Greenwich, UK

Mary Kiernan
University of Greenwich, UK

Lachlan MacKinnon
University of Greenwich, UK

ABSTRACT

This paper outlines an innovative game-based approach to learning introductory programming that is grounded in the development of computational thinking at an abstract conceptual level, but also provides a direct contextual relationship between game-play and learning traditional introductory programming. The paper proposes a possible model for, and guidelines in support of, this games-based approach contextualized by an analysis of existing research into the issues of learning programming and game based learning approaches. Furthermore, the proposed game-based learning model focuses not only on procedural and applied knowledge and associated skills acquisition in computational thinking, but also provides contextualised theoretical knowledge on Computer Science concepts. By way of illustration, the authors introduce a game prototype currently being developed to combine a puzzle solving game-play that uses Computer Science concepts as the game elements.

INTRODUCTION

Computer Science (CS) is now an intrinsic part of our lives and one could argue that a world without computers would be unthinkable. Yet there is a consensus that CS has serious conundrums, such as attracting students, low retention rates and low motivation for learning programming despite the continuing growth of the IT industry (Beaubouef & Mason, 2005; Kinnunen & Malmi, 2006; Fletcher & Lu, 2009). It is widely accepted that motivation and involvement are imperative in retaining stu-

DOI: 10.4018/978-1-4666-1864-0.ch023

dents in CS (Guzdial, 2004; Beaubouef & Mason, 2005) and in order to do this we need to engage students more in the process of learning programming by building more effective mechanisms and tools for the development of programming skills. However, this is not an easy task, and one of the core aims of learning programming should be to constantly highlight that programming is not only coding but also thinking computationally and acquiring skills to develop solid solutions through understanding of concrete problems. Wing (2006) describes Computational Thinking (CT) as a problem solving approach that combines logical thinking with CS concepts, and that it can be used to solve a problem in any discipline regardless of where the problem lies. CT does not propose that problems need to be solved in the same way a computer tackles them but rather it encourages the use of critical thinking using CS concepts. Surprisingly, most students are not conscious of this and when they undertake a project the reaction of the majority of them is to start coding immediately, skipping the crucial steps of design and the need to apply CT (Rajaravivarma, 2005). Thus, learning programming becomes a demanding task that requires CT to describe a problem and propose a solution, followed by the need to design and code in order to convert the solution into the syntax of a programming language. Recent studies in this field address this problem by highlighting the necessity to become trained in thinking computationally before learning programming, and conclude that the education of programming along with the theory of computing needs to be represented in a way that would make sense to students within the CS discipline (Wing, 2008; Guzdial, 2008; Fletcher & Lu, 2009; Qualls & Sherrell, 2010). Currently, students are introduced to CT at the same time and in the same setting as they are introduced to their first programming language, and this creates a problem for learning programming (Fletcher & Lu, 2009).

There has been a huge amount of research undertaken to propose techniques and tools to make learning programming easily accessible to students. Many of these approaches involve innovative teaching styles (such as active learning, visualization, pair programming). Kazimoglu et al. (2010a) discuss computer video games (henceforth referred to as games) and game-like environments as one strategy to use as a motivational tool to engage students in learning programming. Moreover, various studies, despite relatively low participation in their assessments, point out that a game based learning (GBL) environment for learning programming is more enjoyable and motivational to students than a traditional teaching environment within the CS discipline (Eagle & Barnes, 2009; Yeh, 2009). However, the lack of empirical evidence supporting the approach, and in some cases the absence of curriculum specific learning outcomes, lead us to the problem that there is not enough evidence to prove games are an educationally effective solution (Kazimoglu et al., 2010b).

These studies indicate that there is still a gap in the body of knowledge for GBL with regard to how learning computer programming and developing CT should happen, particularly through game-play. Most of the literature in this field examines either a drill and practice approach (Graven & MacKinnon, 2008; Yeh, 2009) or runs assessments on the existing visual programming environments depending on game design principles (Maloney et al., 2008; Resnick et al., 2009; Wu et al., 2010). Only a small number of studies review how to learn programming through game-play without a drill and practice approach and/or a visual programming tool (Long 2007; Muratet et al., 2009). None of these studies provide sufficient information to be regarded as guidelines that would enable researchers to develop similar GBL environments for learning programming. Despite the huge amounts of work in GBL, currently there are only a small number of games available specifically designed for learning programming through game-play. Moreover, existing guidelines in GBL mostly consider how to use, adapt and assess games in