Available online at www.sciencedirect.com
ScienceDirect
Information and Software Technology 49 (2007) 445–454
INFORMATION AND SOFTWARE TECHNOLOGY
www.elsevier.com/locate/infsof

Evaluation of object-oriented design patterns in game development

Apostolos Ampatzoglou, Alexander Chatzigeorgiou *

Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece

Received 19 January 2006; received in revised form 30 May 2006; accepted 5 July 2006
Available online 22 August 2006

Abstract

The use of object-oriented design patterns in game development is being evaluated in this paper. Games' quick evolution, demands great flexibility, code reusability and low maintenance costs. Games usually differentiate between versions, in respect of changes of the same type (additional animation, terrains etc). Consequently, the application of design patterns in them can be beneficial regarding maintainability. In order to investigate the benefits of using design patterns, a qualitative and a quantitative evaluation of open source projects is being performed. For the quantitative evaluation, the projects are being analyzed by reverse engineering techniques and software metrics are calculated.

Keywords: Game development; Design patterns; Software evaluation; Software metrics

1. Introduction

Recently, games have become one of the most profitable factors in the software industry. More specifically, during the last few years the game industry has been considered to produce revenue greater than the movie industry and its development rate has been one of the most fast growing in the United States economy [19,24]. Furthermore, game design and the methods used for easier and more efficient development constitute a very interesting open research field [4]. It goes without saying that computer games play a very important role in modern lifestyle. Therefore, it is no longer necessary to explain what a computer game is. On the other hand, it is not so obvious why game research is an extremely interesting field and simultaneously why game development is a very complicated task to accomplish.

The answer to the first question has many levels. As mentioned above, even though game development is a very strong industry, the research on this field is in its infancy. This fact leads game programming professionals to demand better developing methodologies and software engineering techniques [11]. Furthermore, games are the first and sometimes the only market for advanced graphics techniques to demonstrate the quality of graphics they produce [16,22]. It has been acknowledged [21] that game industry draws on research from academia, corporate R&D labs and in-house work by game developers. Several papers pointing out the need for transfer from research to industry appear at SIGGRAPH [25] conferences.

In order to prove what a complex task game development is, we will present the minimum personnel that a typical such company needs. The discrete roles of personnel do not prove the complexity of the task itself, because this is a common tactic for software development teams. The distinction between games and other forms of software is that, in games, the development groups consist of people with different fields of expertise. First of all, a script writer is required; this person will write the game script and fill in a document usually called "concept paper" [15]. The lead game designer will convert information from the concept paper into another called "design document" which will be the guide throughout the development process. Apart from that, the company employs a group of programmers with several skills and expertise, such as engine and graphics programmers, artificial intelligence programmers, sound

* Corresponding author. Tel.: +30 2310 891886; fax: +30 2310 891875.
E-mail addresses: ampatzoglou@doai.uom.gr (A. Ampatzoglou), achat@uom.gr (A. Chatzigeorgiou).

0950-5849/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2006.07.003

programmers and tool programmers. In collaboration with the sound programmers, the game development company will hire a musician and a sound effects group. In addition, the art of the game will be created by graphic artists, such as the character artists, the 3D modelers and the texture artist. Finally, the company must hire testers who would play the game in order to find bugs and make suggestions for changes in gameplay, graphics and the story [10,19,27].

Game programming is a main course in many Universities, at one or two semesters, and there are quite a few Master of Science programmes related to that field, but not many PhD theses on this subject. This fact reveals the industry need for game development methodologies but also the lack of relevant scientific research. A speculation about the absence of scientific research is that games are widely considered a "soft skill" topic. During the last few years this situation has slightly changed with a few publications about game design patterns and the use of game engines in creating virtual worlds and GIS applications. The ways and the extend of teaching game programming in graduate and postgraduate studies has also been examined in a few papers [18,19,24].

In the next sections, the way object-oriented design patterns could be used in computer games and an extended evaluation of their benefits and drawbacks are being examined. More specifically, in chapter 2 a short introduction to general game architecture is being presented. In chapter 3, there is a brief literature review of object-oriented design patters and game mechanics design patterns. In addition, four examples of how object-oriented design patterns could be used are discussed. In chapter 4, two real open-source games are being evaluated. Finally, future research plans and conclusions are being presented.

2. Game architecture

One of the most interesting aspects of game research is the architecture that the developer will use. In recent papers, there are a few references to the modules that the programs are being decomposed to, however, without extensive discussion of maintainability and code reusability issues. Such issues have been examined in detail in classical object-oriented programming, but those ideas are extremely immature in game programming.

Designing and programming large-scale software is a very complicated job that requires many human work hours. Consequently, software is usually divided, logically, into subprograms that are autonomously designed, programmed and tested by separate programmers' groups. These subprograms are called modules. Decomposing software into modules is an important decision that plays a main role in the architecture and further design of the program. In this section, the modules proposed for games are examined and briefly discussed.

In [3], Bishop et al., described a general game's architecture as shown in Fig. 1. This schema presents an interactive game's vital modules. The items with solid outlines

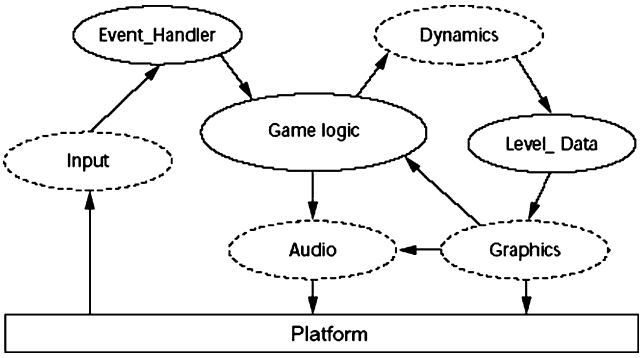

Fig. 1. Generic game architecture [3].

are essential to every game while the dashed outlines refer to modules that are found in more complicated and demanding games. The game logic is the part that holds the game's story. The audio and graphics are the modules that help the writers narrate the story to the player. The event-handler and the input modules, supply the game logic with the player's next action. The level data module is a storage module for details about static behaviour and the dynamics module configures dynamic behaviour of game's characters.

3. Design patterns

With the term design patterns one refers to identified solutions to common design problems. The notion of patterns was first introduced by Christopher Alexander, who identified and proposed solutions to common architectural problems. In his work he dealt with the question whether quality in architecture can be objective. By examining several architectural artifacts he discovered that "good" quality designs shared some common characteristics, or shared "common solutions to common problems" [1]. Patterns can also be used in software architecture and, if applied properly, they increase the flexibility and reusability of the underlying system. Object-oriented design patterns specify the relationships between the participating classes and determine their collaboration. Such solutions are especially geared to improve adaptability, by modifying the initial design in order to ease future changes [12]. Each pattern allows some aspect of the system structure to change independently of other aspects. In [20,29] the authors investigated the effect of design patterns on comprehensibility and maintainability. Their experiment analyzed the consumed time and the correctness of the outcome for the implementation of a given set of requirements employing systems with and without design patterns. The results have indicated that some patterns are much easier to understand and use than others and that design patterns are not universally good or bad. However, it is implied, that if patterns are used properly and in appropriate cases, they prove extremely beneficial regarding maintainability and comprehensibility.