

Computer Game-Flow Design

M. J. TAYLOR, D. GRETTY, and M. BASKETT
School of Computing and Mathematical Sciences,
Liverpool John Moores University, Liverpool, U.K.

Computer games are a rapidly growing segment of the entertainment industry. Design and development of modern computer games can be a complex activity involving many participants from a variety of disciplines. However, computer game design approaches typically appear to be less formalised than those used for other types of software systems. In this article we describe an approach to computer game-flow design intended for the design of individual game levels within a computer game and demonstrate its application in practice.

Categories and Subject Descriptors: I.6 [Computing Methodologies]: Simulation and Modeling

General Terms: Design, Performance

Additional Key Words and Phrases: Computer games, design

INTRODUCTION

Computer games are a hugely popular and successful application of computer technology [Jones 2000]. Modern computer games involve the development of large software systems that can be composed of hundreds of thousands, or even millions of lines of code [Bethke 2003]. Gold [2004] commented that the development of computer games is different from other types of software development because computer games include far more artistic content than other types of software systems. In addition, most computer games have a control system that is quite different from other software applications. This is particularly so for console games where the player has a control pad with ten or so buttons and one or more analogue joysticks. Much work will go into controlling what can often be a rich and complex system with a particularly limited control set. McConnell [2001] noted that the development of computer games involves teams of developers from a variety of disciplines, including analysts, designers, coders, testers, QA staff, project management staff, and also lots of nontraditional software personnel like writers, artists, and musicians.

Kanev and Sugiayama [1998] stated that when designing computer games, new game concepts can be difficult to communicate. Bethke [2003] advocates the use of development methods for computers game design, since systematic and repeatable methods allow the retention of what worked and the improvement of what did not work well. However, Natkin and Vega [2004] commented upon the absence of established methodologies for computer game design. Bethke [2003] stated that it is understandable that game development companies are generally poor at enforcing a strong software development process. First, software companies are in general poor at the development

Authors' address: School of Computing and Mathematical Sciences, Liverpool John Moores University, Liverpool, L3 3AF, UK; email: mjtaylor@livjm.ac.uk

Permission to make digital/hard copy of part of this work for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication, and its date of appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee. Permission may be requested from the Publications Dept., ACM, Inc., 1515 Broadway, New York, NY 10036, USA, fax:+1(212) 869-0481, permissions@acm.org

© 2006 ACM 1544-3574/06/001-ART3A \$5.00

process; second, the computer game industry holds creativity sacred; and third, the games themselves are continually becoming larger, faster, and more complex. The result is that managers who may have had hands-on experience in creating a computer game five years ago, will now have a misguided interpretation of the process.

In this article we demonstrate how use-case diagrams (from UML, the Unified Modelling Language) [Booch et al. 1999] can be extended and adapted by incorporating aspects of decision trees to provide a means to design detailed computer game-flows that are not only useful to game programmers, but also to other professionals involved in computer game development (e.g., story, level, and character designers, 3-D modelers, artists, animators, and musicians). The computer game-flow design approach described in this article is intended to model the individual game levels within a computer game.

LITERATURE REVIEW

Onder [2002] describes the use of story-beat diagrams, which are a collection of ovals and arrows that help show the flow (or alternate flow) of a computer game story at a high level. Story-beat diagrams show how the player can move through a computer game. For each scene in the story-beat diagram, Onder [2002] describes the need to record the scene location, scene description, the object and cast members in the scene, and an event table that includes action / result entries that describe what happens in response to each player inside the scene. Rollings and Morris [2004] suggest the use of token interaction matrices for designing computer games. A token interaction matrix is a chart of all the interactions that take place in a computer game or a segment of a computer game. A token is defined as a discrete game element that is directly or indirectly manipulated by the game player. Lewinski [2000] describes the use of flowcharts for high-level design of computer games. The flowcharts simply show how each scene or mission in a computer game (represented by an ellipse) flows to the next scene or mission by means of an arrow. Rouse [2001], however, suggests that flowcharts are not actually all that useful in the game design process, other than to communicate the progress of the game play to other members of the development team. Instead, Rouse [2001] supports the use of storyboards (as used in film and television production) for mock-ups of how the game will appear to the player. In addition, Rouse [2001] encourages the production of technical computer design documents such as the overall code structure, the major classes to be used, and pseudocode for more detailed program design. LaMothe [2002] advocates state transition diagrams for game loops to assist game programming activities, and for decision trees to develop artificial intelligence coding for computer games.

Siang and Rao [2004] and Bethke [2003] describe an approach for designing computer games using use-case diagrams and class diagrams from the Unified Modelling Language (UML) [Booch et al. 1999; Bennett et al. 2002]. A use-case is a description of a set or sequence of actions that a system performs; it yields an observable result of value to a particular actor. A use-case is rendered as an ellipse with solid lines, usually including only its name. A use-case diagram shows a set of use-cases and actors and their relationships [Booch et al. 1999]. However, Biddle et al. [2002] commented that debate on the benefits of use-cases, and indeed what a use-case really is, is still ongoing. Variations of the basic concept have been developed, particularly to support specific aspects of software development such as task scripts. A class is a description of a set of objects that share the same attributes, operations, relationships, and semantics. A class is rendered as a rectangle, usually including its name, attributes, and operations. A class diagram shows a set of classes, interfaces, and collaborations and their relationships [Booch et al. 1999]. Penton [2003] states that one of the first things to do when designing

a computer game is to establish the major classes that will be used in the game. Rucker [2003] proposed the use of a larger subset of UML for developing computer games, including component diagrams for the dependencies of the source code files, activity diagrams for program execution flowcharts, sequence diagrams for the interaction of program objects, in addition to use-case diagrams and class diagrams.

However, the approaches described by Siang and Rao [2004], Bethke [2003], and Rucker [2003] apply use-case diagrams in a very high-level manner, and clearly show the high-level actions that the game player may undertake. Similarly, the class diagrams are used at a very high level to describe how the high-level actions relate to one another. The approaches described by Siang and Rao [2004], Bethke [2003], and Rucker [2003] do not explicitly model the flow of a computer game within a given game level.

Gold [2004] suggests a diagrammatical representation of the route through a complete game level as a topological map or graph, with nodes representing scenes and edges representing the transitions between them. However, Gold [2004] advocated that such maps be used for scenic (noninteractive) objects.

RESEARCH RESULTS

Computer Game-Flow Design

The aim of the computer game-flow design approach described here is to provide a detailed design technique for modeling individual computer game levels. Although storyboards can be used to design game levels, they are mainly a visual design tool, and provide little in the way of concrete guidance for game programmers, since storyboards mainly concentrate on artistic interpretations of the scenes within a given game level. The game-flow design approach described here aims to outline the flow of game play between scenes and within scenes in a given game level. The techniques described in this article can be used in conjunction with storyboards, which add a visual representation of the backgrounds and characters in each scene within a given game level.

Use-case diagrams from the Unified Modelling Language (UML) [Booch et al. 1999] can be used to show the objects involved in a given transaction or process; however, use-case diagrams are often represented as a series of simple single-level actions, which can be somewhat limiting for complex computer game design. For example, the use-case diagrams for the Starfleet Command 3 and Pacman games provided by Bethke [2003] and Siang and Rao [2004], respectively, do little more than indicate the main actions that a player could undertake, like “View current crew assignments” or “Eat power pellet.”

Extending and adapting use-case diagrams by incorporating elements of decision trees and identifying directionality of flow can produce a modeling technique that we call “Computer game-flow design,” which can model the nature of a given game level within a computer game and provide easy to interpret guidance for game programmers, artists, animators, and musicians. Computer game artists can benefit from the above approach, since all the backgrounds and characters within a given game level are defined in the game-flow design diagrams for scenes in a given game level. Animators can benefit from the approach because all the interactions relating to the objects in a given game level (mobile game objects such as opponents; fixed game objects such as doors) are defined. Musicians can benefit, since the soundtracks required in the scenes in a given game level are defined, and in addition, by being aware of what will happen in the scenes within a given game level, musicians can also be aware of the tempo and dramatic impact required of game play music.

The development of a computer game requires game programmers to generate functional programs; game artists and animators to use image processing / creation and

animation software tools; and for musicians, using MIDI composer or other software, to create game music. Ideally, game design documents should be relevant and applicable to all involved in the game development process, so that the different game elements produced by a variety of professionals will fit together appropriately.

The computer game-flow design approach described here is intended for the development of computer games where the player is “guided” through the scenes in each game level, or, by means of a limited number of possible paths, through the scenes in each game level. Such computer games can be thought of as story-driven, where events are scripted, drama or tension is built by a writer, and, if part of the story, the improbable can happen. The design approach described here is less applicable to computer games where the player has almost free control of the direction of the gameplay, e.g., SIMS [EA 2005]. Such computer games can be considered emerging games, where an environment and actors are specified, and as the player moves through the game environment, he or she interacts with the actors in the game.

The basic elements of our game-flow design diagram are directional arrows that show one-way or two-way motion through a game segment (or scene); rectangles to represent “fixed” game objects like doors, lifts, ladders etc.; ovals to represent “mobile” game objects such as opponents; and circles to represent events such as explosions. The directional arrows are labeled with a description of the background for that particular game segment or scene and the music or sounds to be played, and each object (fixed or mobile) is labeled with a description. The main line of the game-flow diagram shows the game objects with which the game player’s character must interact. Interactions with game objects above the main line of the game flow are optional. The interaction that the player has with each object (fixed or mobile) is described via pseudocode that can be used to model either the simple IF ... THEN ... ELSE type of finite state machine logic of simple interaction [LaMothe 2002] or the more complex mathematically formulated rules and formulas for artificial intelligence interactions [Buckland 2002]. Figure 1 gives an example of a computer game-flow design diagram for the first scene of the “On track” level of the Electronic Arts Medal of Honour: Frontline computer game [EA 2005]. The pseudocode that could be used to describe the interaction with the objects shown in Figure 1 follows (in high-level simplified form):

Silenced pistol and bullets:

```
IF player moves over silenced pistol and bullets
    THEN add silenced pistol and bullets to player inventory
```

Bedroom door:

```
IF player selects activate button
    IF door state = closed
        THEN display open door animation, play door open sounds
    IF door state = open
        THEN display close door animation, play door close sounds
```

Uniform and ID Card:

```
IF player selects activate button
    THEN add uniform and ID card to player inventory
```

Hall door:

```
IF player selects activate button
    IF door state = closed and uniform and ID card are in player
    inventory
        THEN display open door animation, play door open sounds
```

```

IF door state = closed and uniform and ID card not in player
inventory
    THEN display message "Uniform and ID card not obtained"
IF door state = open
    THEN display close door animation, play door close sounds
    
```

Wehrmacht officer 1:

```

IF uniform and ID card activated
    THEN display Identity confirmed animation, play identity
confirmation dialogue
ELSE combat function
    
```

Combat function:

```

IF player selects fire button and target direction correct
    THEN display wounded opponent animation, play wounded opponent
dialogue
IF hit by opponent
    THEN increment hit counter
IF hit counter > 10
    THEN display end game animation
    
```

The diagram shown in Figure 1, together with the relevant pseudocode (which can describe class methods in UML class diagrams and program methods in C++ objects), can be used by game programmers, artists, animators, and music composers to explicitly indicate what they need to do to develop scenes at a given game level and to show how their work will fit together. (The other scenes in the "On track" level of the Electronic Arts Medal of Honour: Frontline computer game are shown in Figures 2 to 8.)

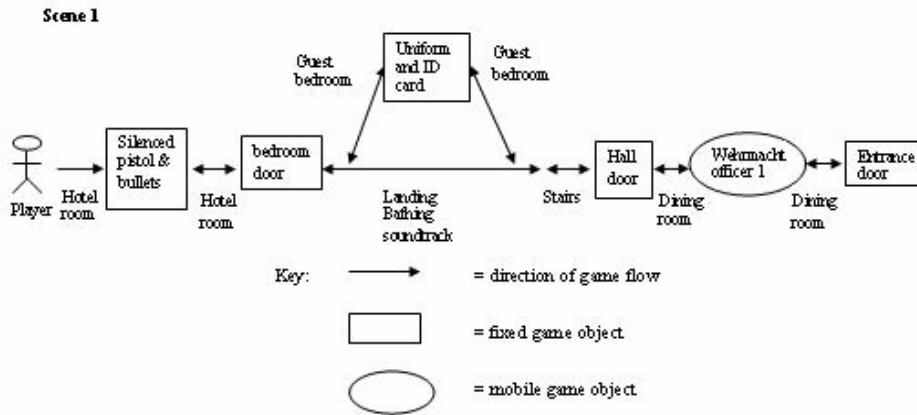
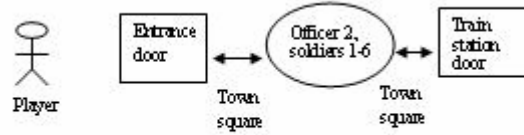


Fig. 1. Example of computer game-flow design diagram for scene 1 of "On track" level of Medal of Honour: Frontline computer game.

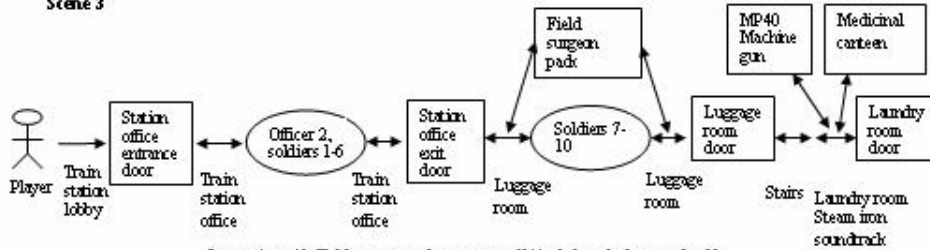
Scene 2



Interaction with Officer 2 and soldiers 1-6 can be via ID card or silenced pistol

Fig. 2. Scene 2 of “On track” level of Medal of Honour: Frontline.

Scene 3



Interaction with Field surgeon pack = restores 50% of player's character health

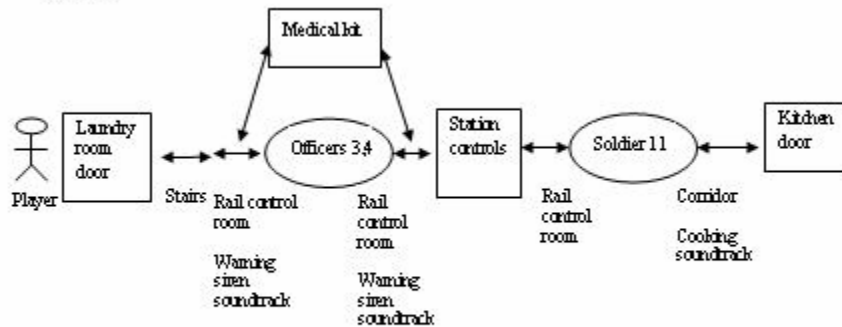
Interaction with Medicinal canteen = restores 10% of player's character health

Interaction with MP40 Machine gun = add to player inventory

Interaction with Luggage room door = once luggage room door has been opened, train station door is closed

Fig. 3. Scene 3 of game level.

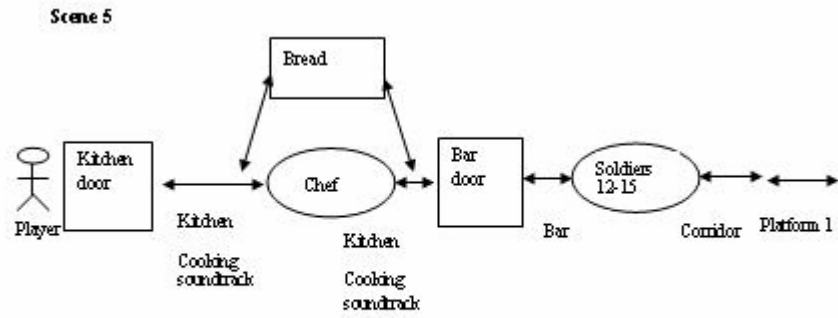
Scene 4



Interaction with Medical kit = restores 25% of player's character health

Interaction with station controls = destroy by firing at them. When destroyed, warning siren is ended.

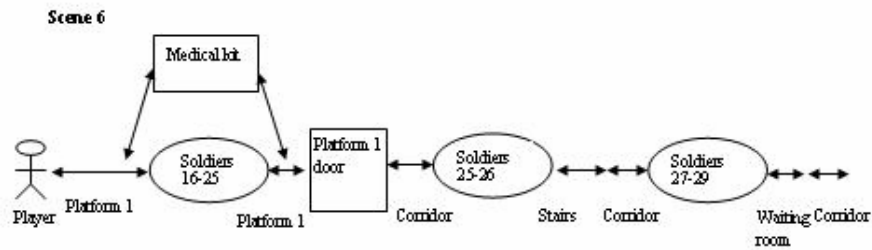
Fig. 4. Scene 4 of game level.



Interaction with chef = chef throws knives

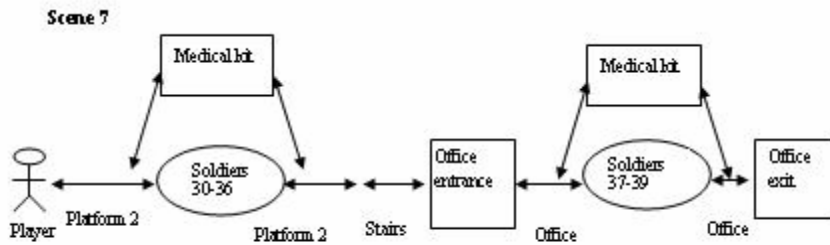
Interaction with bread = restores 5% of player's character health

Fig. 5. Scene 5 of game level.



Interaction with Medical kit = restores 25% of player's character health

Fig. 6. Scene 6 of game level.



Interaction with Medical kit = restores 25% of player's character health

Air raid soundtrack plays throughout this scene

Fig. 7. Scene 7 of game level.

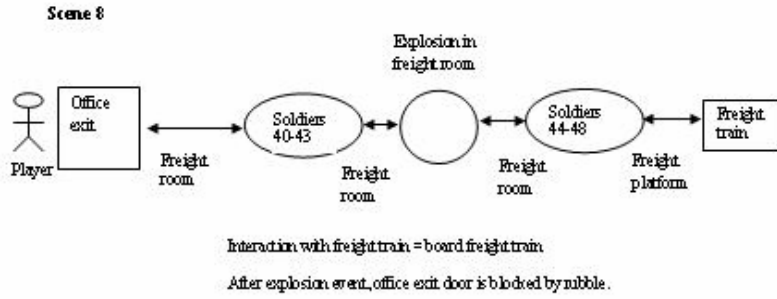


Fig. 8. Scene 8 of game level.

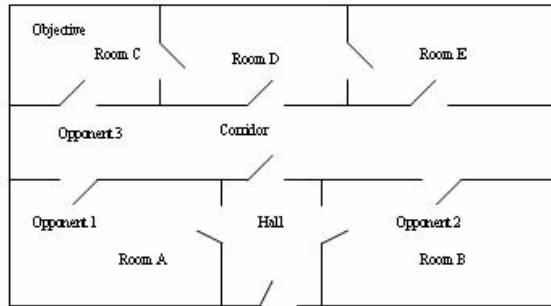


Fig. 9. Simplified map for a loosely scripted game.

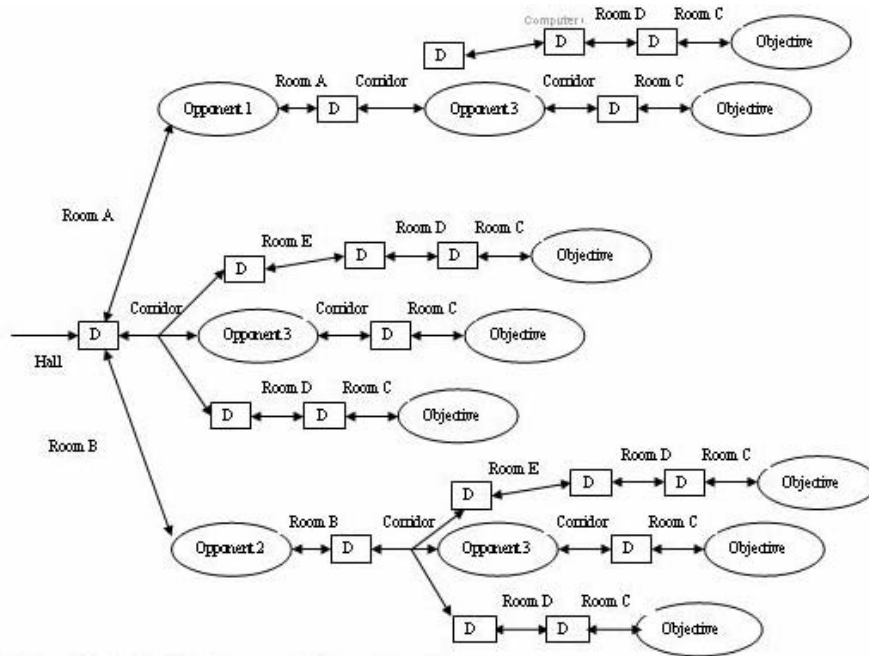


Fig. 10. Highly simplified game-flow design diagram for a loosely scripted game level.

Game-flow diagrams can be used to model the game play in tightly scripted games. Modeling game play in more loosely scripted games such as the Hitman and Driver series can be more complex. In a more loosely scripted game, the player may have a stated overall mission or objective (typically using text or dialogue or both). However, the player will have far greater freedom of passage and action through the game environment, compared to tightly scripted games like the Medal of Honour series or the Matrix. For example, in a given game level in Hitman 2, the player can choose a variety of paths through a building or area to reach the intended objectives. In addition, by using a map showing both the building or area layout and the location of opponents, the player can avoid or engage opponents by choosing an appropriate pathway. The player can also change clothes with opponents in order to alter the initial behaviour of future opponents; potentially, this can be done at different points in the game play of a given level. Modeling the game flow of a simple, loosely scripted game level with a map, as shown in Figure 9, would lead to a game-flow diagram as in Figure 10.

As can be seen from Figures 9 and 10, even with a simple, loosely scripted game environment, where opponents do not move and the player does not backtrack his or her movements, modeling the game flow in a given game level becomes more complex due to the possible number of paths through the game environment. Not only is extensive backtracking unwieldy to model, but holding state information such as the location of bullet-holes (which can provide players with a sense of continuity and immersion) also becomes computationally very expensive. The amount of state data that must be stored to permit backtracking can be a serious factor for the designer of console games, as the performance and memory available from the console is much more tightly constrained than, for example, a PC. Computer game-flow diagrams as described in this article are in practice perhaps more applicable to tightly scripted games, where the player has only a limited number of options at any particular point in the game play within a given game level.

CONCLUSION

Computer game design is an emerging type of software development activity. Although early computer games may have required little in the way of game-flow design, modern three-dimensional computer games are becoming increasingly sophisticated, and require a more formalised design approach. In this article we have described and demonstrated a computer game-flow design approach that extends and adapts the widely used UML technique of use-case diagrams to make them more applicable to computer game design. This approach is pragmatic, given that much computer game programming is done with the object-oriented language C++, and that use-case diagrams are commonly used as a development technique in other types of software systems. In addition, the computer game-flow design approach discussed here provides a rich description of game flow for a given game level, which can be used by game programmers, artists, animators, musicians, and testers. In addition, our approach can be used as a communication avenue between these often distinct roles within a computer game development project.

Game-flow design diagrams are a high-level game design approach that can compliment storyboards and provide the detail that storyboards would find more difficult to describe explicitly, for example, the interaction between the game player's character and the fixed and mobile objects in the game.

REFERENCES

- BENNETT, S., MCROBB, S., FARMER, R. 2002. *Object Oriented Systems Analysis and Design Using UML*. McGraw Hill, London.

- BETHKE, E. 2003. *Game Development and Production*. Wordware Publishing, Plano, TX.
- BIDDLE, R., NOBLE, J., AND TEMPERO, E. 2002. Essential use cases and responsibility in object oriented development, In *Proceedings of the Twenty-fifth Australasian Computer Science Conference* (Monash University, Melbourne, Jan./Feb. 2002). In *Conferences in Research and Practice in Information Technology*, vol 4. M. Oudshoorn (ed.). Australian Computer Society, Sydney, Australia, 7-16.
- BOOCH, G., RUMBAUGH, J., AND JACOBSON, I. 1999. *The Unified Modelling Language User Guide*. Addison Wesley, Boston, MA.
- BUCKLAND, M. 2002. *AI Techniques for Game Programming*. Premier Press, Cincinnati, OH.
- EA. 2005. *Electronic Arts*. Computer Game Publisher, <http://www.eagames.com>.
- GOLD, J. 2004. *Object Oriented Game Development*. Addison Wesley, Harrow, UK.
- JONES, R. 2000. Design and implementation of computer games: A capstone course for undergraduate computer science education. *ACM SIG Computer Science Education Bull.* 32, 1 (2000), 260-264.
- KANEV, K. AND SUGIYAMA, T. 1998. Design and simulation of interactive 3D computer games. *Computers and Graphics* 22, 2 (1998), 281-300.
- LAMOTHE, A. 2002. *Tricks of the Windows Game Programming Gurus*. Sams Publishing, Indianapolis, IN.
- LEWINSKI, J. 2000. *Developer's Guide to Computer Game Design*. Wordware Publishing, Plano, TX, 160-163.
- MCCONNELL, S. 2001. Who needs software engineering. *IEEE Software* 18, 1 (2001), 5-8.
- NATKIN, S. AND VEGA, L. 2004. A Petri net model for computer games analysis. *Int. J. Intelligent Games and Simulation* 3, 1(2004). <http://www.scit.wlv.ac.uk/>
- ONDER, B. 2002. Writing the adventure game. In *Game Design Perspectives*. F. Laramée (ed.). Charles River Media, Hingham, MA. 28-43.
- PENTON, R. 2003. *Data Structures for Games Programmers*. Premier Press, Cincinnati, OH.
- ROLLINGS, A. AND MORRIS, D. 2004. *Game Architecture and Design*. New Riders, Indianapolis, IN.
- ROUSE, R. 2001. *Game Design, Theory and Practice*. Wordware Publishing, Plano, TX. 293-303.
- RUCKER, R. 2003. *Software Engineering and Computer Games*. Addison Wesley, London.
- SIANG, A. AND RAO, G. 2004. Designing interactivity in computer games: A UML Approach. *Int. J. Intelligent Games and Simulation* 3, 2 (2004). <http://www.scit.wlv.ac.uk/>
- VARANESE, A. 2003. *Game Scripting Mastery*. Premier Press, Cincinnati, OH.

Received October 2004; revised June 2005; accepted September 2005