Towards a Definition of a Computer Game

Jouni Smed

University of Turku, Department of Information Technology,
Lemminkäisenkatu 14 A, FIN-20520 Turku, Finland

Harri Hakonen

University of Turku, Department of Information Technology,
Lemminkäisenkatu 14 A, FIN-20520 Turku, Finland

Turku Centre for Computer Science
TUCS Technical Report No 553
September 2003

ISBN 952-12-1217-9
ISSN 1239-1891

Abstract

This paper approaches computer games from three perspectives: First, by defining the properties common to all games. Second, by fitting computer games into Model–View–Controller architectural pattern and discerning common software components. Third, by listing features that players expect from an enjoyable computer game.

Keywords: computer games, model–view–controller, games, entertainment industry, ludology

TUCS Laboratory
Algorithmics laboratory

In memory of Timo Kaukoranta.

1 Introduction

Games have always been a popular pastime, but with the advent of computer games they have become even more pervasive. Despite all this progress, we may still stop and ask what makes a game. Because computer games are a subset of games, everything we can say about games in general applies also to them. Nevertheless, computer games are also computer programs, and, therefore, lessons learnt in software construction can be applied to them. A third perspective to computer games is subjective and it concerns finding out what features the players expect from a computer game

To answer these questions—and perhaps to raise some more—we begin in Section 2 by analysing the structure of games in general. In Section 3, we turn our focus on computer games and try to recognize their common software components. In Section 4, we present a list of sought-after features that a computer game should have; although we align ourselves with the players' position, we try to form our statements so that they can be considered when designing the implementation. The concluding remarks appear in Section 5.

2 Defining a game

J. Huizinga in his classical work *Homo Ludens* (1938) gives the following definition for play [Hui55, p. 132]:

> [Play] is an activity which proceeds within certain limits of time and space, in a visible order, according to rules freely accepted, and outside the sphere of necessity or material utility. The play-mood is one of rapture and enthusiasm, and is sacred or festive in accordance with the occasion. A feeling of exaltation and tension accompanies the action, mirth and relaxation follow.

A dictionary defines 'game' as 'a universal form of recreation generally including any activity engaged in for diversion or amusement and often establishing a situation that involves a contest or rivalry' [Enc03]. A game seems to involve three components:

- players who are willing to participate the game (e.g., for enjoyment, diversion or amusement),
- rules which define the limits of the game, and
- goals which give arise to conflicts and rivalry among the players.

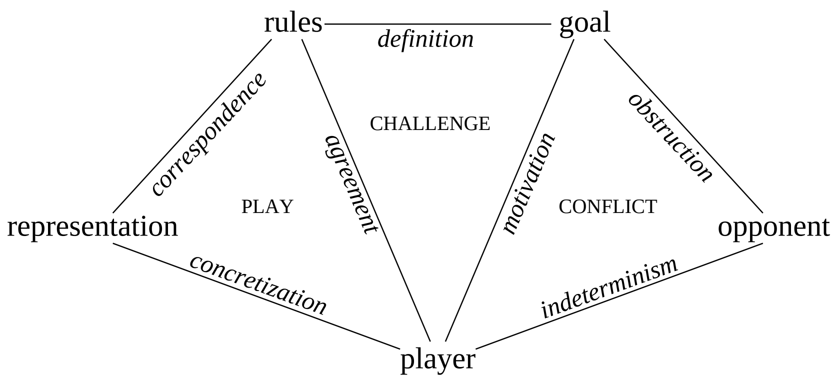

Figure 1: Components, relationships, and aspects of a game.

Although most of the definitions encountered in the literature recognize at least these three aspects, we can make more subtle distinctions between them. Figure 1 illustrate the components and relationships present in a game. The relationships form three aspects:

Challenge Rules define the game and, consequently, the goal of the game. When players decide to participate in the game, they agree to follow the rules. The goal motivates the players and drives the game forwards.

Conflict The opponent (which can include unpredictable humans and unpredictable random processes) obstructs the players from achieving the goal. Because the players do not have a comprehensive knowledge on the opponent, they cannot determine precisely the opponent's effect on the game.

Play The rules are abstract but they correspond to real-world objects. This representation concretizes the game to the players.

Consider, for example, the game of Poker. The players agree to follow the rules, which state what cards there are in a deck, how many cards one can change, and how the hands are ranked. The rules also define the goal, having as good hand as possible, which is the player's motivation. The other players are opponents, because they try to achieve a better hand to win. Also, the randomness of the deck opposes the player, who cannot determine what cards will be dealt next. The game takes a concrete form in a deck of cards, which represent the abstractions used in the rules.

To clarify, let us contrast the difference between games and other pastimes [Cra84, §1]:

- A *puzzle* includes only the play and challenge aspects. Because it lacks interactive elements, it presents no conflict. Once a puzzle has been solved, its interest usually vanishes. Although puzzles are not games, games can include puzzles as subtasks.
- A *story* is a linearly ordered sequence of events set by a storyteller, whereas a game allows the players to have a choice of the sequence of events. A play of a game can be told afterwards as a story, but a game cannot be constructed from a story alone.
- A *toy* can be identified with the representation component. It is only a part—but an essential one—of a game.

A game play includes also subjective elements such as an immersion to the game world, a sense of purpose, and a sense of achievement from mastering the game. One could argue that the sense of purpose is essential for the immersion. What immerses us into a game (as well as into a book or a film) is the sense that there is a purpose or motive behind the surface. In a similar fashion, the sense of achievement is essential for the sense of purpose (i.e., the purpose of a game is to achieve goals, points, money, recognition etc.). From a human point of view, we get satisfaction in the process of nearing a challenging goal and finally achieving it.

Although this line of thought is fascinating, we shall not pursue into a philosophical discussion on games in general but turn our focus to a subset of games, namely *computer games*. Computer games have a forty-year history, since *Spacewar* is widely regarded as the first proper computer game [Gra81]. During that time, on the outside, everything (especially technological aspects) seems to have changed, but a closer inspection reveals that the basic concepts have remained the same.

3 Anatomy of computer games

Let us define a computer game as a *game that is carried out with the help of a computer program*. This definition leaves us some leeway, since it does not implicate that the whole game takes place in the computer. For example, a game of Chess can be played on the screen or on a real-world board, regardless whether the opponent is a computer program. Incidentally, we can discern three roles for a computer program in a game:

1. co-ordinating the game process,
2. illustrating the situation, and

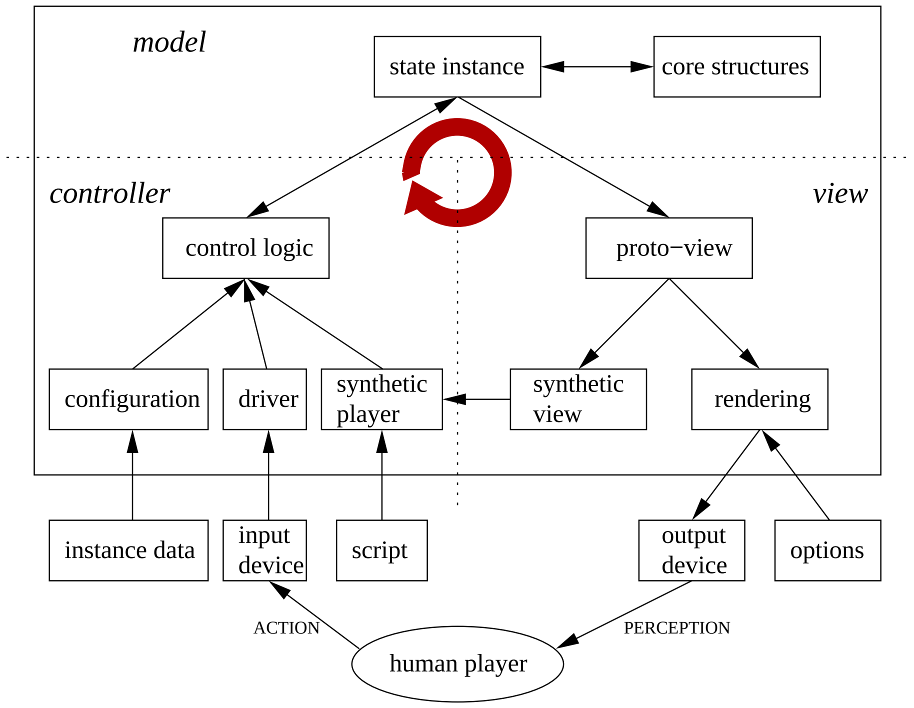

Figure 2: Model, View and Controller in a computer game.

3. participating as a player.

This role division resembles closely the *Model–View–Controller* (MVC) architectural pattern for computer programs. MVC was originally developed within the Smalltalk community [KP88] and later on it has been adopted as a basis for object-oriented programming in general [GHJV95]. The basic idea is that the representation of the underlying application domain (Model) should be separated from the way it is presented to the user (View) and from the way the user interacts with it (Controller). Figure 2 illustrates the MVC components and the data flow in a computer game.

The Model part includes software components which are responsible for the co-ordination role (e.g., evaluating the rules and upholding the game state). The rules and basic entity information (e.g., physical laws) form the core structures. It remains unchanged while the state instance is created and configured for each game process. The core structures need not to cover all the rules, because they can be instantiated. For example, the core structures can define the basic mechanism and properties of playing cards (e.g., suits and values) and the instance data can provide the additional structures required for a game of Poker.

The View part handles the illustration role. A proto-view provides an interface into the Model. It is used for creating a synthetic view for a synthetic player or for rendering a view to an output device. The synthetic view can be preprocessed to suit the needs of the synthetic player (e.g., board coordinates rather than an image of the pieces on a board). Although rendering is often identified with visualization, it may as well include audification and other forms of sensory feedback. The rendering can have some user-definable options (e.g., graphics resolution or sound quality).

The Controller part includes the components for the participation role. Control logic affects the Model and keeps up the integrity (e.g., by excluding illegal moves suggested by a player). Human player's input is received through an input device filtered by a driver software. The configuration component provides instance data, which is used in generating the initial state for the game. The human player participates the data flow by perceiving information from the output devices and generating actions to the input devices. Although the illustration in Figure 2 includes only one player, naturally there can be multiple players participating the data flow, each with their own output and input devices. Moreover, the computer game can be distributed among several nodes rather than residing inside a single node. Conceptually, this is not a problem since the components in the MVC can as well be thought to be distributed (i.e., the data flows run through network rather inside a single computer). In practice, however, the distributed computer games provide their own challenges [SKH02].

A synthetic player is a computer-generated actor in the game. It can be an opponent, a non-player character which participates limitedly (like a supporting actor), or a *deus ex machina* which can control natural forces or godly powers and thus intervene the game events. The more open the game world is, the more complex the synthetic players are. This trade-off between the Model and the Controller is obvious: If we remove restricting code from the core structures, we have to reinstate it in the synthetic players. For example, if the players can hurt themselves by walking into fire, the synthetic player must know how to avoid it. Conversely, if we rule out fire as permitted area, path finding for a synthetic player becomes simpler.

As we can see in Figure 2, the data flow of the human player and the synthetic player resemble each other. This allows us to project humanlike features to the synthetic player. We can argue that, in a sense, there should be no difference between the players whether they are humans or computer programs; if they are to operate on the same level, both should ideally have the same powers of observation and the same capabilities. Still, synthetic players usually cheat, and this has been the norm for a long time. Generally, the reason is obvious: a computer program is no match for human ingenuity, and, hence, it gets the benefit of the home turf. This is understandable—and we may even forgive it when it seems fair—but, ideally, the synthetic players should be in a similar situation as their human counterparts.

4 Sought-after features

In this section, we aim to provide a list of features that make a computer game interesting for a player. However, as we observed earlier, defining what makes a game enjoyable is subjective, and, therefore, we can argue only ostensively by giving examples from existing games. Our list is far from complete and open to debate.

Let us turn the discussion around and ask what makes a bad computer game. It can be summed in one word: *limitation*. Of course to some extent limitation is necessary—we are, after all, dealing with limited resources. Moreover, the rules are all about limitation, although their main function is to impose a goal. The art of making games is to balance the means and limitations so that this equilibrium engrosses the human. How limitations manifest themselves in the program code? The answer is the lack of parameters: The more things are hard-coded, the less there are possibilities to add and support new features. Rather than closing down possibilities, a good game should be open and modifiable for both the developer and the player as we shall see.

Parameterization indicates that reusable software components should be employed more often. Apart from graphics engines, game development needs also other game engine types such as network engines, simulation engines, and artificial intelligence engines. Because they are not yet widely available, their functionality is usually embedded in the game by bundling up separate low level function libraries. Application level parameterization enables reuse that eases the game development and diminishes the risks involved. In other words, it increases cost-efficiency on the long run. This applies also to the producing of innovative game mods and sequels by parameter variations.

4.1 Game world

A game is not a story: While a story progresses linearly, a game must provide an illusion of free will [Cra84, Cos02]. Obviously, the player must have a range of actions to choose from at each stage. More formally, let us consider the game as a directed graph where the game states are vertices and the possible actions edges (Figure 3). This means that the greater the outdegree (or fan-out) of a vertex is, the more freedom the player has. In this graph,

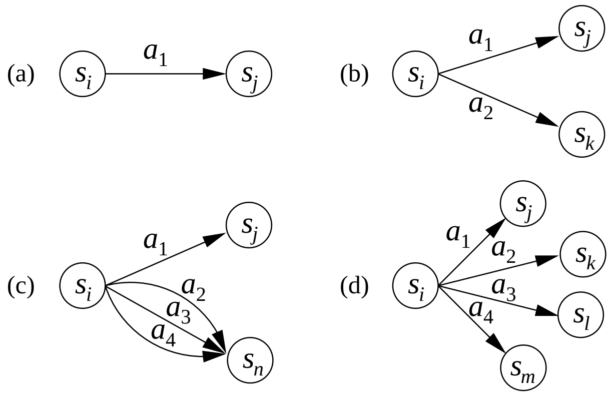

Figure 3: In a game graph, game states are represented as vertices and actions as directed edges. (a) A linear progression (e.g., a story) allows no diversion. (b) Outdegree is the number of edges leaving a vertex (in this case vertex s_i has an outdegree of 2). (c) Indegree is the number of edges entering the vertex (in this case vertex s_n has an indegree of 3). (d) Although the number of possible actions (i.e., the outdegree of vertex s_i) is the same as in the previous case, each action has now a unique response.

the uniqueness of a response can be measured as the indegree (or fan-in) of a vertex. For example, assume that the game plot is divided into chapters like in *Max Payne* or *Diablo II*. Typically, at this transition point the plot lines of the previous chapter are concluded, and many new plot alternatives are introduced. This means that in the graph the beginning state of the chapter has a large indegree and outdegree. The game properties can now be analysed with graph concepts (e.g., repetitiveness corresponds to cycles in the graph). An infamous example of a game graph with a small overall indegree is *Dragon's Lair*, where, at each stage, the players can choose from several alternative actions of which all but one will lead them to a certain death.

The game world should provide the player with as much freedom as possible, and the responses to the player's actions should be appropriate. An obvious way to implement an open game world is simulation. Computer games are often likened to simulations or virtual environments, but the classification is not clear-cut (see Figure 4): We may allow that, for example, Chess simulates warfare in a highly abstract manner but it is not so obvious whether Draughts is a simulation. Nevertheless, most games—and especially computer games—are simulations because a resemblance to the real-world objects or everyday world (even with a slight touch of fantasy) assists immersion. Indeed, entertainment industry has embraced military simulations

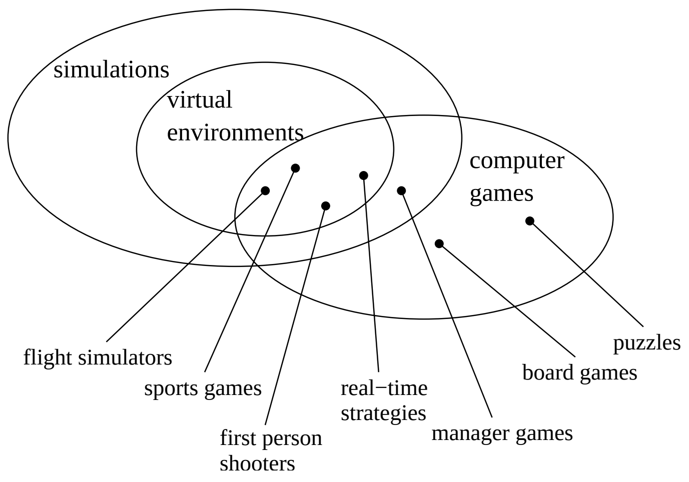

Figure 4: While virtual environments simulate (possibly real-world) environments, there are computer games that do not necessarily belong to simulations.

because of the realism they provide [CMZ01].

The main difference between simulations and games is that games are goal-oriented. A flight simulator, for example, is not a game in itself but dog-fighting with a flight simulator is. In computer games, a usual approach is to include a story into the game and, as a consequence, limit the simulation. This game-as-a-story approach usually contains a linear—or at most a slightly diverse—plot, where the player has some freedom only between fixed entry points (i.e., the game graph converges to a predetermined state from time to time). Still, many games do not include a story-line nor impose a sequence of events. Granted, some of them can be tedious (e.g., *Frontier: Elite II* in which the universe is vast and devoid of action whereas in the original *Elite* the goal remains clearer)—but so are many games which include a story.

4.2 Synthetic players

For game developers, the border between the game world and the entities inhabiting it is often muddy. Earlier we separated them and called the computer-controlled entities synthetic players. Related research has been done on military simulations and within the artificial intelligence (AI) community [SBHS98, LvL01]. Four key features that a synthetic player must provide are [SKH03]:

- real-time response,

- distribution,
- autonomy, and
- communication.

In the traditional turn-based games, the computer opponent can think (almost) as long as it requires. Nowadays, games are mostly real-time programs, which puts a hard computational strain on the synthetic player. It can no longer delve into finding an optimal strategy but it should react immediately. Response is the key-word—even to such extent that game developers tend think that it is better to have armies of mindless bots than to grant them even a shred of intelligence. It seems as if we cannot achieve both real-time response and intelligent behaviour.

Distribution has become more important now that games using networking are more common. This can be a solution to the dilemma of real-time response and intelligence. Instead of running the synthetic players on one machine, they can be distributed so that the cumulative computational power of the networked nodes gets utilized. For example, *Homeworld* uses this technique and distributes the computer-controlled opponents among the participating computers.

Distribution begs the question how autonomous the synthetic players should be. As long as we can rely on the network there is no problem, but if nodes can drop out and join at any time, distributed synthetic players must display autonomy. This is not necessarily a drawback, because it can lead to a smaller and better design. Also, we must not forget that complex behaviour can emerge from seemingly simple autonomous agents [KES01].

Finally, if the synthetic players are to cross the gap of autonomy, they must start to communicate explicitly with each other. They have to inform others on their decisions, indicate their plans, and negotiate with each other—just like we humans do in the real world.

Ideally, a game comprising just synthetic players could be as interesting to watch as a movie or television show [CMC02]. In other words, if the game world is fascinating enough to observe, it is likely that it is also enjoyable to participate—which is one interesting factor in the god games like *The Sims*, where the synthetic players seem to act (more or less) with a purpose. Sometimes a game even gathers around a community that starts to tell stories of the things that synthetic players have done and to interpret them in human terms. A good example is *NetHack*, which, after nearly twenty years, remains a cornucopia of tales.

This brings us to the idea of a game within a game. Already back in the 1980s *Core War* demonstrated that programming synthetic players to compete with each other can be an interesting game itself [Dew84]. After that some games have tried to use this approach, but, by the large, AI programming games have been only by-products of 'proper' games. For example, *Age of Empires II: The Age of Kings* includes a possibility to create scripts for computer players. This has given a rise to a new kind of gaming, where programmers compete who creates the best AI script. The whole game is then carried out by a computer while the humans remain as observers. Although the programmers cannot affect the outcome during the game, they are more than just enthusiastic watchers: They are the coaches and the parents, and the synthetic players are the protégés and the children.

4.3 Multiplaying

What keeps us interested is—surprise. Humans are extremely creative at this, and it should be encouraged by computer games. Nowadays, networking has allowed the games to include an ever increasing number of human players. The possibility of having multiple players enriches the game experience—and complicates the design process [ZNR00]—because much of the story arises from the interaction between the players (synthetic or human). Indeed, sharing an experience with fellow players can make game play more pleasurable.

In a narrow, story-like plot line, multiplayer games become virtually impossible. One could argue that a single plot line has enough space for one controlling player at a time. However, even if the plot line itself seems quite narrow, the co-operative action around it can provide amusement. For example, the plot line of *Serious Sam: The Second Encounter* is a typical first-person shooter: Go through the levels and kill all monsters. Rarely enough, the same plot line can be played in both single-player and multiplayer mode. The combination of action, interaction, and humour compensates the narrowness that the plot line may have. Moreover, it demonstrates that collaboration can be as much fun as competing for oneself. In this sense, much can be gained from work done on collaborative virtual environments [BGRP01].

4.4 Customization

A good game has an intuitive interface that is easy to learn. Because players have their own preferences, they should be allowed to customize the user interface to their own liking. Moreover, the interface should adapt dynamically to the needs of a player. For example, in critical situations the player ought to be able to have more detailed control.

Personalization is one way to increase immersion of the game. This means the ability to modify game so that the players really feels that they themselves are in the game world. For example, *NHL 2001* allows the players to create their own ice hockey team with logos and customized players.

Likewise, living in a game world should feel natural. Tutorials are a convenient method for illustrating the interface and the game world to the player. Hence, it can reduce the learning curve significantly. To keep the game challenging as the player progresses, it should support different difficulty levels. When a player masters the game at some difficulty level, the next level must introduce new challenges.

4.5 Extensions

An enduring computer game is a platform for surprising innovations. Perhaps the genre closest to extendibility are the first-person shooters, where modifications (or 'mods') are a usual way to extend the original game. Quite aptly, the core of these games is called a game engine. For example, game engines of *Quake* and *Half-Life* are widely used in other commercial games. In real-time strategy games, typical mods include new maps and scenarios.

Another approach is the extension packs, which can be considered as mods provided by the original game developer. They usually include new levels, playing characters, and objects (e.g. *Diablo II: Lord of Destruction*, *Age of Empires II: The Conquerors Expansion*), and perhaps some improvement of the interface. Receiving player feedback through the Internet is extensively utilized when designing these extension packs.

It is important to recognize *a priori* what software development mechanism are published to the players and with what interfaces. The game developers typically implement special software for creating content for the game. These editing tools are a valuable surplus to the final product. If the game community can create new variations of the original game, longevity of the game increases. Furthermore, the inclusion of the developing tools is an inexpensive way—since they are already implemented—to enrich the contents of a game product.

4.6 Replaying

Once is not enough. We take pictures and videotape our lives. The same applies also to games. Traditionally, many games provide the option to take screen captures, but recently—especially in sports games—replays have become an important feature. Replaying can be extended to cover the whole game (e.g., *Age of Empires II: The Age of Kings* includes this option—although it was originally developed for debugging purposes). These recordings allow us to relive and memorize the highlights of the game, and we can share them with our friends and the whole game community.

5 Conclusion

We recognized components, relationships, and aspects common to all games. By fitting computer games into Model–View–Controller architectural pattern we discerned common software components. Finally, we listed features that a computer game should include to provide an enjoyable gaming experience, and concluded that a way to achieve them is to allow parameterization.

References

[BGRP01] Steve Benford, Chris Greenhalgh, Tom Rodden, and James Pycock. Collaborative virtual environments. *Communications of the ACM*, 44(7):79–85, 2001.

[CMC02] Fred Charles, Steven J. Mead, and Marc Cavazza. Generating dynamic storylines through characters' interactions. *International Journal of Intelligent Games & Simulation*, 1(1):5–11, 2002.

[CMZ01] Michael Capps, Perry McDowell, and Michael Zyda. A future for entertainment-defense research collaboration. *IEEE Computer Graphics and Applications*, 21(1):37–43, 2001.

[Cos02] Greg Costikyan. I have no words & I must design: Toward a critical vocabulary for games. In Frans Mäyrä, editor, *Computer Games and Digital Cultures Conference Proceedings*, pages 9–33, Tampere, Finland, June 2002.

[Cra84] Chris Crawford. *The Art of Computer Game Design*. Osborne/McGraw-Hill, Berkeley, CA, 1984. Available at ⟨http://www.vancouver.wsu.edu/fac/peabody/game-book/Coverpage.html⟩.

[Dew84] A. K. Dewdney. Computer recreations: In the game called Core War hostile programs engage in a battle of bits. *Scientific American*, 250(5):14–22, May 1984.

[Enc03] Encyclopædia Britannica. Game. Encyclopædia Britannica Online, accessed Apr. 23, 2003. ⟨http://search.eb.com/eb/article?eu=36648⟩.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1995.

[Gra81] J. M. Graetz. The origin of Spacewar. *Creative Computing*, pages 56–67, August 1981. Available at ⟨http://www.wheels.org/spacewar/creative/SpacewarOrigin.html⟩.

[Hui55] Johan Huizinga. *Homo Ludens: A Study of the Play-Element in Culture*. The Beacon Press, Boston, MA, 1955.

[KES01] James Kennedy, Russell C. Eberhart, and Yuhui Shi. *Swarm Intelligence*. Morgan Kaufmann, San Francisco, CA, 2001.

[KP88] Glenn E. Krasner and Stephen T. Pope. A cookbook for using the model-view-controller user interface paradigm in Smalltalk-80. *Journal of Object-Oriented Programming*, 1(3):26–49, 1988.

[LvL01] John E. Laird and Michael van Lent. Human-level AI's killer application: Interactive computer games. *AI Magazine*, 22(2):15–25, 2001.

[SBHS98] Martin R. Stytz, Sheila B. Banks, Larry J. Hutson, and Eugene Santos, Jr. An architecture to support large numbers of computer-generated actors for distributed virtual environments. *Presence*, 7(6):588–616, 1998.

[SKH02] Jouni Smed, Timo Kaukoranta, and Harri Hakonen. Aspects of networking in multiplayer computer games. *The Electronic Library*, 20(2):87–97, 2002.

[SKH03] Jouni Smed, Timo Kaukoranta, and Harri Hakonen. AIsHockey—a platform for studying synthetic players. In Loo Wai Sing, Wan Hak Man, and Wong Wai, editors, *Proceedings of the 2nd International Conference on Application and Development of Computer Games*, pages 183–8, Hong Kong SAR, China, January 2003.

[ZNR00] José Pablo Zagal, Miguel Nussbaum, and Ricardo Rosas. A model to support the design of multiplayer games. *Presence*, 9(5):448–62, 2000.

Ludography

Blizzard North, *Diablo II.* Blizzard Entertainment, 2000.

Blizzard North, *Diablo II: Lord of Destruction.* Blizzard Entertainment, 2001.

David Braben and Ian Bell, *Elite.* Firebird, 1984.

Croteam, *Serious Sam: The Second Encounter.* Gathering of Developers, 2002.

DevTeam, *NetHack 3.4.1.* ⟨http://www.nethack.org/⟩, 2003.

A. K. Dewdney, *Core War.* 1984.

EA Sports, *NHL 2001.* Electronic Arts, 2000.

Ensemble Studios, *Age of Empires: The Age of Kings.* Microsoft Games, 1999.

Ensemble Studios, *Age of Empires II: The Conquerors Expansion.* Microsoft Games, 2000.

Frontier Developments, *Frontier: Elite II.* Gametek, 1993.

J. Martin Graetz, Stephen R. Russell, and Wayne Witanen, *Spacewar!.* 1962.

id Software, *Quake.* id Software, 1996.

Maxis, *The Sims.* Electronic Arts, 2000.

Relic Entertainment, *Homeworld.* Sierra Studios, 1999.

Remedy Entertainment, *Max Payne.* Gathering of Developers, 2001.

Sullivan Bluth, *Dragon's Lair.* ReadySoft, 1989.

Valve Software, *Half-Life.* Sierra Studios, 1998.

Turku Centre for Computer Science
Lemminkäisenkatu 14
FIN-20520 Turku
Finland

http://www.tucs.fi

University of Turku
- **Department of Information Technology**
- **Department of Mathematics**

Åbo Akademi University
- **Department of Computer Science**
- **Institute for Advanced Management Systems Research**

Turku School of Economics and Business Administration
- **Institute of Information Systems Science**