AIsHockey—A Platform for Studying Synthetic Players

Jouni Smed, Timo Kaukoranta
Department of Information Technology and Turku Centre for Computer Science (TUCS), University of Turku, Lemminkäisenkatu 14 A, FIN-20520 Turku, Finland
jouni.smed@cs.utu.fi, timo.kaukoranta@cs.utu.fi

Harri Hakonen
Oy L M Ericsson Ab, Telecom R&D, Joukahaisenkatu 1, FIN-20520 Turku, Finland
harri.hakonen@lmf.ericsson.se

Abstract

We introduce a computer game platform, AIsHockey, which is based on the real-world game of ice hockey. The platform allows us to implement and study autonomous, real-time synthetic players (i.e., computer-controlled actors in a game). By applying the Model-View-Controller architectural pattern we define the role of a synthetic player and recognize its responsibilities and interfaces in the software. We describe the AIsHockey system and discuss our experiences and observations in educational and research perspectives.

1. Introduction

Whilst modern computer games (CGs) already excel in the areas of graphics and sound, most players still find the artificial intelligence (AI) embedded in the games lacking. The computer-controlled opponents or bots can be challenging at first but soon the human player begins to notice patterns in their behavior and can easily exploit their weaknesses. The gaming industry is acutely aware of the problem, and the emphasis is gradually shifting towards AI development. The companies allocate more resources—time, money and personnel—to AI development, and during the last six years the share of processing power reserved for AI programs has risen from less than five percent to over twenty percent [15].

The AI problems encountered in CGs cover a gamut of topics from simple pathfinding to complex decision-making problems [11, 16]. To avoid ambiguousness we use the term *synthetic player* to denote the humanlike aspects of AI in CGs. This emphasizes that the computer is a fellow participant in the game, and, obviously, this is the goal AI in CGs. Apart from games requiring simple hand-eye coordination, people usually want to play with or against someone who they feel is a worthy ally or adversary. The grand challenge for the future is to develop synthetic players to fulfill these expectations.

Academic AI researchers have awaken to the problems and possibilities of CGs [9, 13]. Commercial CGs provide a cheap, tested, and widely-spread environment for trying out AI techniques. Moreover, CG environment has one unique feature: it is not a simulation of a problem domain but the problem domain *per se* [13]. This means that we can omit problems originating from the real world such as noisy sensor data or limitations of motor functions. Consequently, while the related academic work done on military simulations [14] or robotics [1] is valuable, CGs present the AI problems in a much purer form.

The motivation behind the *AIsHockey* platform presented in this paper stems from this observation. Although it is based on a real-world game of ice hockey, it is not a simulation ice hockey but a game itself. Because it is abstracted and simpler, it allows us to concentrate on synthetic players. Simply put, we are not interested in how to skate realistically or what the player can see at a given moment. Instead, we want to study how to form a cooperating team from a group of autonomous synthetic players.

The plan of the paper is following. We begin with an analysis of the architectural structure of CGs in Section 2. It is needed to explicate the relation of synthetic players to the software as a whole. We realize this by using the Model-View-Controller architectural pattern. In Section 3, we describe the rules and structure of the AIsHockey platform. We also give an example of a simple synthetic player. This is followed by a discussion in Section 4, where we summarize our experiences and observations from the team development. Concluding remarks appear in Section 5.

2. 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

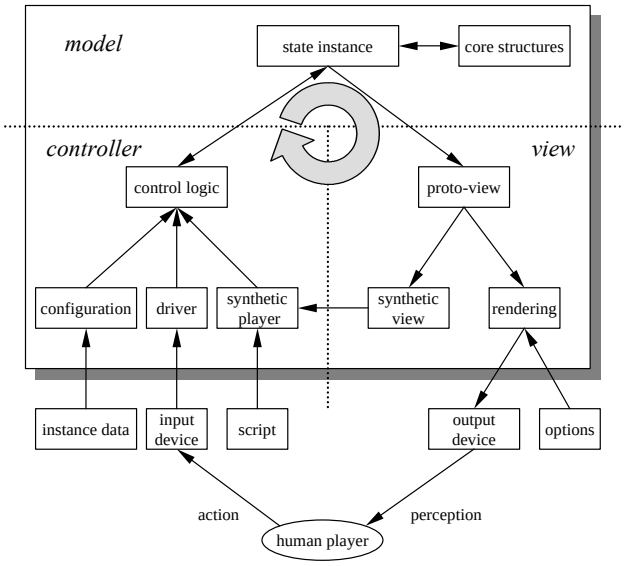

Figure 1: Model, View and Controller in a computer game.

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, although the opponent is a computer program. Incidentally, we can discern three roles for a computer program in a game:

1. coordinating the game process,
2. illustrating the situation, and
3. participating as a player.

By definition a CG must act out one or more of these roles.

This role division resembles closely the *Model-View-Controller* (MVC) architectural pattern for computer programs. MVC was originally developed within the Smalltalk community and later on it has been adopted as a basis for object-oriented programming in general [8, 5]. 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 1 illustrates the MVC components and the data flow in a CG.

The Model part includes software components responsible for the coordination 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 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 in a chess board). Although rendering is often identified with visualization, it may as well include audification and other forms of sensory feed-back. 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 1 includes only one player, naturally there can be multiple players participating the data flow, each with own output and input devices. Moreover, the CG can be distributed among several nodes rather than residing inside a single node (as illustrated). 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 CGs provide their own challenges [12].

A synthetic player is a computer-generated actor in the game. It can be an opponent (as in the game of chess), a non-player character (NPC) 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 tradeoff 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, pathfinding for a synthetic player becomes simpler.

As we can see in Figure 1, 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 no be 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 obser-

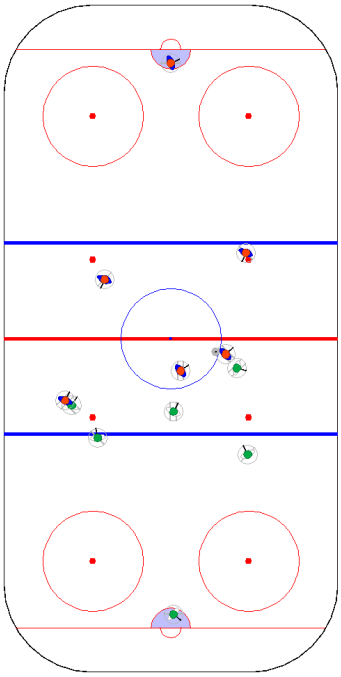

Figure 2: A screenshot from AIsHockey.

vation and the same capabilities. Still, synthetic players usually cheat, and this has been the norm for a long time [3]. 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.

3. Description of the system

AIsHockey[1] is a computer game based on ice hockey (see Figure 2). It complies the official IIHF rules [6] as carefully as possible. In addition to the measurements, the rules adopted to AIsHockey include face-offs, offsides and icings. However, all the rules regarding penalties are omitted; only interfering the opponent's goalie inside the opponent's goal crease is penalized, and it causes a face-off on the offending team's endzone.

The physical model implemented in the game engine obeys Newtonian particle physics on a two-dimensional plane. Particles are either circles, arches or lines, and they are associated with mass, size, position, velocity and acceleration. For example, a player is a circle with a radius of 0.35 m and a mass of 75 kg. Collisions between particles

[1]The AIsHockey platform is publicly available at the address http://staff.cs.utu.fi/staff/jouni.smed/aishockey/.

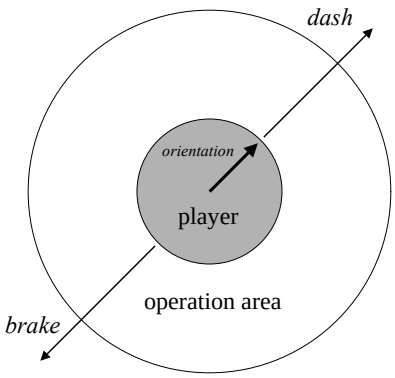

Figure 3: The player can move forwards (dash) or backwards (brake) according to its orientation. The player can shoot the puck only when it is inside the operation area.

are either elastic or inelastic. Players have an orientation according to which they can move forwards or backwards (see Figure 3). To simulate skating the friction depends on the player's orientation: along the orientation the friction is small but perpendicular to the orientation it is large. Because the player can change the orientation to any angle instantly, it can brake effectively by changing its orientation 90 degrees thus maximizing the friction. Player can shoot the puck (or a goalkeeper can keep the puck), if it resides within an operation area surrounding the player (radius 0.85 m). To communicate with other players, a player can send messages which all players (even the opponents) can receive.

AIsHockey draws inspiration from the simulation league of the RoboCup initiative, where autonomous client programs play soccer [1]. Nevertheless, AIsHockey is fundamentally a CG, whereas RoboCup is a testbench for robotics. The assumptions made in the simulation league yield to the restrictions of real-world robots (e.g., sensors and motor systems). As we emphasized earlier, AIsHockey concentrates on the implementation of synthetic players instead of robots. AIsHockey stands out also because the pace of the game is more intense than in RoboCup.

3.1. Structure

AIsHockey is implemented with Java 2 Platform, Standard Edition, version 1.4 by Sun Microsystems. AIsHockey provides the synthetic players with a simple interface to the game engine. Each synthetic player is an instance of a Java class, which inherits methods for receiving information on the game state and for sending its own actions (e.g., moving, shooting and turning). Because each instance runs on its own thread, the players are independent and can be distributed among several nodes in a network. To enable team-

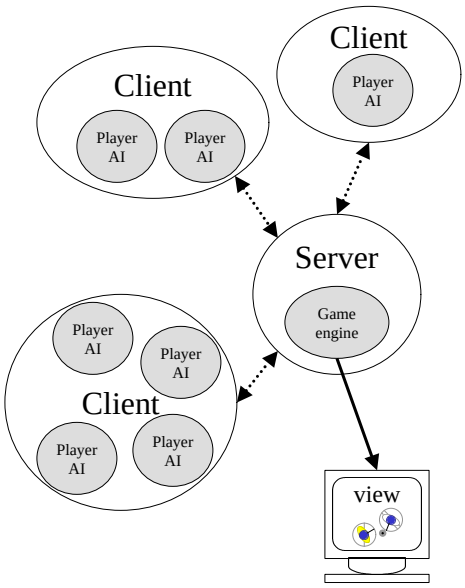

Figure 4: Synthetic players are distributed among the clients, while the game engine resides in the server.

work and cooperation the players can communicate in the game world through the server.

Figure 4 illustrates the underlying client/server network model. The game engine runs in a server, which sends update messages to the clients regularly. To reduce network traffic this is done by using multicasting. Each client has one or more synthetic players (i.e., Java class instances run on separate threads), which receive the update messages and, in return, send their commands to the server using unicast. The game situation can be displayed on screen or the game can generate a log file of the game events.

Let us now review the system briefly in MVC terms. The core structure defines the physical laws and the basic rules of the game. When system is started, the instance data (e.g., the duration of the game and the network addresses) are used to create a state instance (i.e., the game itself). The state instance creates a proto-view which is delivered over the network to the clients, which modify it to a synthetic view for the synthetic players. The proto-view can also be rendered to a display. A synthetic player sends commands to the control logic, which in turn updates the game state. Since we have reduced the role of the human player on purpose, human input is needed only for outside intervention (e.g., passing judgment on situations that the game engine cannot detect).

3.2. Implementing a synthetic player

The challenge of AIsHockey is to implement a collaborating team of autonomous, real-time synthetic players. As mentioned earlier, each player is defined by a Java class which inherits the class `AI` from the package `fi.utu.cs.hockey.ai` and implements the abstract method `react`. Simply put, the synthetic player is just the implementation of the this method.

The synthetic player receives information about the positions, orientations and messages of all players as well as the position of the puck. In addition, it inherits auxiliary methods (e.g., method `headingTo` returns the orientation towards a given player or position) and constants defining the measurements of the rink (e.g., the width of the goal or the position of face-off spots).

For interacting with the game world, the synthetic player has six methods: `dash`, `brake`, `head`, `shoot`, `keepPuck` (for goalkeepers only), and `say`. The parameters of moving and shooting are normalized to the range $[0.0, 1.0]$, the angle of orientation is given in radians, and the message is a long word (64 bits).

Let us give an example of a simple synthetic player:

```
import fi.utu.cs.hockey.ai.*;

public class MyAI extends AI
                  implements Constants {
    public void react() {
        if (isPuckWithinReach())  {
            head(headingTo(0.0,
                       THEIR_GOAL_LINE));
            brake(0.5);
            shoot(1.0);
            say(1050L);
        } else {
            head(headingTo(puck()));
            dash(1.0);
}   }   }
```

The behavior of the player should be obvious: If the puck is within the reach of the player, it changes its heading to point to the center of the opponent's goal. It brakes slightly, shoots the puck with full force, and sends a message to the other players. If the puck is not operable, the player heads towards the puck with full speed.

4. Discussion

In the development of AIsHockey we aimed at a platform that makes possible to study synthetic players. We wanted to strip away the unrelated parts to expose the underlying characteristics of a synthetic player. This process revealed us four key features that a synthetic player must provide:

- 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, CGs 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 keyword—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 behavior.

Distribution has become more important now that CGs 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* (Sierra Entertainment, 1999) 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 behavior can emerge from seemingly simple autonomous agents [7].

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.

4.1. Team development

The AIsHockey platform was used in a research seminar for graduate students, where they could experiment with the approaches presented in the scientific literature. It provided an entertaining way to address serious algorithmical and software design issues. The students were given free hands to choose any approach they think would be a basis for a winning team. We were surprised by the diversity and high quality of the presented solutions. Hardly no two students took the same approach, and everybody tried to be on the alert on what others were doing and how to react and counteract their efforts.

In the overall designs, we can easily recognize the classical three-level hierarchy of decision making. On the strategical level, there are the choices of how to win the game (e.g., whether to play offensively or defensively). On the tactical level, the choices concern carrying out the strategy the best possible way (e.g., whether to use man-marking defense or space-marking defense). On the operational level, the choices are simple and concrete (e.g., where should the player position itself and if it has the puck, whether to shoot it or pass it to another player). The problem is how to choose what do. It is fairly simple on the operational level—shoot if you have an opening, pass if you can do it safely—but it gets harder and harder as the level of abstraction raises.

Already at an early stage, it became evident that the problem of forming a team boils down to whether the team is a collection of individuals or a central-controlled unit. Individuality relies on emergent behavior, which may be hard to achieve intentionally. This is, however, an easy way to proceed, since all players can run fairly similar programs and they can be built up gradually. On the other hand, if one opts for a hierarchy, where the decisions regarding the team are made centrally, problems begin early. Despite the problems, purposefulness can lead to effective results, since the team works clearly as a unit and the synthetic players can be highly specialized. Still, if such a rigid team encounters an opponent whose strategy is previously unknown, it can fail utterly due to the lack of adaptiveness.

During the development there was an interesting debate on whether the players should be memoryless. The platform does not provide any form of ready-made memory, but the input is like a continuous stream of still images of the situation in the rink. At some point there was put forth the proposition that there should be two separate leagues, one for forgetting and another for remembering players. On the whole, the advantages of memoryless players—namely, their simplicity, and they apparent effectivity against "smarter" players at the early stages—we tipped off as the benefits of higher level abstractive reasoning became evident. A simple example is the wall pass, which requires that the wall player remembers to pass the puck back.

Surprisingly, communication remained largely underused. Only at the last stages of development, when more edge was required, the benefits of expressing intentions and commanding others became apparent. However, the communication remained rather simple: the players invoke predefined offense or defense patterns or express their inner state.

As a whole, the development process demonstrates almost evolutionary features. It begins with a bunch of selfish and brutal players and sophisticates along the way into a reasoning and communicating team.

4.2. AI programming as a game

Ideally, a game comprising just synthetic players could be as interesting to watch as a movie or television show [2].

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* (Electronic Arts, 1999). The synthetic players seem to act 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* [10], 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 [4]. After that some games have tried to use this approach, but, by the large, AI programming games have been only byproducts of "proper" games. For example, *Age of Empires II: The Age of Kings* (Ensemble Studios, 2000) 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.

AIsHockey has the same effect. 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.

5. Conclusion

We presented a platform for implementing synthetic players. We discussed the role of a synthetic player in a CG by using the MVC architectural pattern, which is also the basis of the design of AIsHockey. We recognized four features—real-time response, distribution, autonomy, and communication—that a synthetic player should have, and observed their importance in the team development.

We would like to see this work as an opening for a new area of applicative research. Although this work concentrated on the sports game genre, the applicability of our approach to other CG genres is evident. One could say that the sports games exemplify the gameness in its purest form. Our approach may have been somewhat narrow, but, even to our own surprise, such a simple application as AIsHockey has given us new insights into the problems of computer games. And yet, it has continued to provide us with entertainment like a good game should.

Acknowledgments

The authors wish to express their gratitude to the students who participated the research seminar on artificial intelligence in computer games held in the fall 2002 for their involvement in developing the teams and their contribution to the discussion on synthetic players.

References

[1] M. Asada, H. Kitano, I. Noda, and M. Veloso. RoboCup: Today and tomorrow—what we have learned. *Artificial Intelligence*, 110(2):193–214, 1999.

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

[3] C. Crawford. *The Art of Computer Game Design*. Osborne/McGraw-Hill, Berkeley, CA, 1984. Electronic version available at http://www.vancouver.wsu.edu/fac/peabody/game-book/Coverpage.html.

[4] 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.

[5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1995.

[6] International Ice Hockey Federation. IIHF official rule book. Web page, Oct. 2002. http://www.iihf.com/hockey/rules/offrules.htm.

[7] J. Kennedy, R. C. Eberhart, and Y. Shi. *Swarm Intelligence*. Morgan Kaufmann, San Francisco, CA, 2001.

[8] G. E. Krasner and S. 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.

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

[10] NetHack 3.4.0 Home Page. Web page, Oct. 2002. http://www.nethack.org/.

[11] S. Rabin, editor. *AI Game Programming Wisdom*. Charles River Media, Hingham, MA, 2002.

[12] J. Smed, T. Kaukoranta, and H. Hakonen. Aspects of networking in multiplayer computer games. *The Electronic Library*, 20(2):87–97, 2002.

[13] R. St. Amant and R. M. Young. Artificial intelligence and interactive entertainment. *Intelligence*, 12(2):17–9, 2001.

[14] M. R. Stytz, S. B. Banks, L. J. Hutson, and E. Santos, Jr. An architecture to support large numbers of computer-generated actors for distributed virtual environments. *Presence*, 7(6):588–616, 1998.

[15] S. Woodcock. Game AI: The state of the industry 2001–2002. *Game Developer*, 9(7):26–31, July 2002.

[16] S. Woodcock. The Game AI Page. Web page, Oct. 2002. http://www.gameai.com/.