Simulation & Gaming

http://sag.sagepub.com

Advances in Games Technology: Software, Models, and Intelligence

Edmond Prakash, Geoff Brindle, Kevin Jones, Suiping Zhou, Narendra S. Chaudhari and Kok-Wai Wong

Simulation Gaming 2009; 40; 752 originally published online Oct 29, 2009;
DOI: 10.1177/1046878109335120

The online version of this article can be found at:
http://sag.sagepub.com/cgi/content/abstract/40/6/752

Published by:

http://www.sagepublications.com

On behalf of:

Association for Business Simulation & Experiential Learning

International Simulation & Gaming Association

Japan Association of Simulation & Gaming

North American Simulation & Gaming Association

Society for Intercultural Education, Training, & Research

Simulation & Gaming
40(6) 752–801

DOI: 10.1177/1046878109335120
http://sg.sagepub.com
SAGE

Advances in Games Technology: Software, Models, and Intelligence

Edmond Prakash,[1] Geoff Brindle,[1] Kevin Jones,[2] Suiping Zhou,[2] Narendra S. Chaudhari,[2] and Kok-Wai Wong[3]

Abstract

Games technology has undergone tremendous development. In this article, the authors report the rapid advancement that has been observed in the way games software is being developed, as well as in the development of games content using game engines. One area that has gained special attention is modeling the game environment such as terrain and buildings. This article presents the continuous level of detail terrain modeling techniques that can help generate and render realistic terrain in real time. Deployment of characters in the environment is increasingly common. This requires strategies to map scalable behavior characteristics for characters as well. The authors present two important aspects of crowd simulation: the realism of the crowd behavior and the computational overhead involved. A good simulation of crowd behavior requires delicate balance between these aspects. The focus in this article is on human behavior representation for crowd simulation. To enhance the player experience, the authors present the concept of player adaptive entertainment computing, which provides a personalized experience for each individual when interacting with the game. The current state of game development involves using very small percentage (typically 4% to 12%) of CPU time for game artificial intelligence (AI). Future game AI requires developing computational strategies that have little involvement of CPU for online play, while using CPU's idle capacity when the game is not being played, thereby emphasizing the construction of complex game AI models offline. A framework of such nonconventional game AI models is introduced.

[1]Manchester Metropolitan University, Manchester, UK
[2]Nanyang Technological University, Singapore
[3]Murdoch University, Murdoch, Western Australia, Australia

Corresponding Author:
Edmond Prakash, Department of Computing and Mathematics, Manchester Metropolitan University, Chester Street, Manchester, M1 5GD, UK
Email: e.prakash@mmu.ac.uk

Keywords

artificial intelligence, behavior modeling, board game, character modeling, conversational avatars, environment modeling, game AI, game development, game editor, game engine, game software, gardening game, genetic algorithms, hybrid soft computing, memetic algorithms, neural network optimization, programmable tournament, racing game, small world models, terrain modeling

This article is organized as follows. First, we present some simple and effective topics in software games engineering. Then we look at some of the state-of-the-art game engines available for game development. A survey of the major issues that have occupied game programmers, in relation to the problems of terrain generation, over the past 10 years, is presented, and then we present current trends as a prelude to the development of a general purpose reusable terrain function library.

We then describe how to generate realistic human behaviors. This is a challenging and interesting problem in many applications such as computer games, military simulations, and crowd animation systems. First, we propose a generic behavior modeling framework that reflects the major observations on human behavior in daily-life situations. Next, we describe a case study to illustrate how this modeling framework can be used in various crowd simulations.

Personalization in games can be in terms of difficulty levels, game resources, emotion, characters, and so forth. In game design, the first area can be easily identified and addressed. Normally, during this first stage, the genre of the game will be determined. However, the second area of personalization is sometimes difficult. This is because of the differences of players in terms of their personality, background, culture, skill, and learning ability.

Currently, game artificial intelligence (AI) is used for modeling various aspects of nonplayer characters. Although such applications (e.g., behavior modeling) have been very successful, the deployment of game AI for serious games having learning as main component poses basic questions, such as the representation of an explanation capability as perceived by the human player. In this article, we propose a general framework for the development of such an explanation capability. Various soft-computing models have complementary capabilities. This motivates us to propose a framework for their integration. We identify the need for future game AI engines with such capabilities.

Simple and Effective Topics in Software Games Engineering

Over the past few years, we have introduced several simple but effective new topics into the teaching of software programming and other similar courses. These new topics are industry case study, open-source game editors, programmable tournament, and project-based development. The industry case study allows the students to understand the myriad factors in the technical and business sides of a real games company; the case study we use is id Software, a standout small company that is one of the true leaders in the games industry. The open-source game editor abstracts the programming

details of the platform away from the game, and provides an easy and fast way of creating the domain and objectives of game. Thus, the students can focus their learning on the story line, game boarding, and psychologically manipulating the player. The programmable tournament allows the students to devise and build game strategy in a multiplayer arena. This is quite effective as a tutorial for assessing the students' problem solving and programming skills. Finally, the project demands that the student study and report on the approach to or method of developing games. These new topics are excellent for marrying games development with software engineering, and provide for a wide range of learning and criteria for assessment of the students.

This section discusses the new topics introduced into the game programming and other similar courses in a computer science/engineering program. The topics are simple and easily adapted for most forms of games courses, yet they are also extremely effective in presenting meaningful depth and breadth to any software games engineering course.

Overview of Games Teaching

Software games are quite expansive in concepts, content, and conduct. Hence, teaching games development involves many different learning fields for the students. For instance, there are the technical aspects of games, including computer graphics, mathematics, optimized programming in 3D, algorithms, and grid programming. There are also the media and art design aspects, including character modeling, animation, plot and story line, character development, and atmosphere. There may also be aspects of psychology, including human condition, inflicting of terror, intensity of warfare, the thrill of science fiction, and player addiction.

Therefore, context of the teaching and learning is the key driver of what topics are taught in any games courses. Different schools will focus their teaching of software games on those aspects that align with their core competencies. For instance, the games courses in an arts school will differ substantially from that in a computer engineering school. However, this results in narrow and potentially inaccurate view of games development. Rarely would a student see a more well-round picture of industrial gaming, because there may be little cross-pollination of courses between the schools, and little opportunity for students to take courses in both schools.

With a view to broadening the learning of computer engineering students, the authors introduced new topics that were slightly different to the standard fare in the games engineering course. These were business case study, open-source game editors, programmable tournament, and project-based development. It was discovered that students with engineering background did appreciate, and benefit greatly from, the new content.

Topic 1: Industry Case Study

The motivation for considering a business case study in an engineering school course arose from the queries of several students on how to start up a games company. Few lecturers at university have had such privilege, so the next best thing seemed to be

studying selected gaming companies in the industry. With many such companies in the market today, the search was limited to small privately owned companies; it was natural that the choice of case study narrowed down to id Software, Inc (id, 2008). It is a very small company that earns a lot of profits and brings real innovation to the games industry; in other words, it is an industry leader.

As the case study was being assembled for the course, it was realized that there was far more to the company and the available material than just business, although the sales and marketing was fascinating enough. There were legal issues surrounding the content warning, ethical issues about the games being produced, and the "geek" employee environment and interpersonal relationships. There were also a wealth of technical information, provided through a decade of blogging by the chief software programmer (Carmack, 2008), and a comprehensive technical wiki (DooM, 2008). The case study easily encompasses a week of learning.

Topic 2: Open-Source Game Editor

The overriding constraint of a games development course has to be time. Building a viable and effective game involves so much effort and so many different activities that it really limits the extent of learning by the student. One way to help focus the learning of students is to use the basic framework of an existing game and allow the student to build their own details of gameplay. This is possible through open source game editors; there are several on the market today, thanks to a decade-old business innovation by id Software.

With an open-source game editor such as DeePsea (SBSoftware, 2008), the student can quickly produce a large set of interconnected game levels, with intricate rooms and game areas, and a myriad of characters. The user (of the editor) is provided with menu pick-lists for just about every aspect of game building, including textures, room configurations, object shapes, and character avatars; the editor provides the "glue" to position everything in the game world, including the sequencing of the levels. With a code plug-in and integral compiler, there is no programming required. Now, the student can concentrate on aspects of game development not normally touched on in engineering schools, namely story boarding, gameplay (plot, story context, and character development), and psychology (stealth and suspense, riddle solving, drama, and intensity). The students can produce a working game that is exciting and rich, within 2 weeks complete with story boards and a comprehensive story line.

Topic 3: Programmable Tournament

In an international programming competition, the authors was introduced to an unusual form of game that involved a programmable interface for different teams to realize different game strategies in different programs (CodeRuler, 2008). These programs were run together in a series of tournaments, and the program with the best strategy would triumph over the others. The authors obtained the Alphaworks software

(a free download from IBM (Alphaworks, 2008), and assigned the building of team programs to the students.

The assignment was primarily geared to the students devising good algorithms and writing optimized code—each turn of a player in the tournament lasts only 1 second, so the code has to have low complexity and be optimized. Additionally, some learning is gained from the students writing reports on the strategies they devised (Raveendran, 2007). Finally, the competition between student teams is just plain fun. This activity may encompass at least a week of learning, and may generate a variety of reports and document submissions.

Topic 4: Project-Based Games Development

In addition to the 13-week programming course, software game topics may be learned in project-based activities such as a final year project and university research programs for advanced students. These projects are far longer in duration (6 to 9 months) and provide an excellent opportunity for the student to have a far richer experience in delving into games programming and development.

A common project is for the student to build a game with lots of functionality and attractive graphics (Foo, 2005). Other less common, but nevertheless meaningful, project goals are for the student to report on management of a games project, methodology for development of games (Foo & Kevin, 2005), and development of games engines (Medzseva, 2008).

Summary

Generally, games programming and development courses are limited to the core competencies of the particular school teaching the courses. This provides the students with an incomplete understanding of games development. We have discussed four simple topics that may be incorporated into an engineering schools software games course, and that may effectively increase the learning of the students to make them more effective games developers after graduation.

Game Development Using Game Engines

In this section, first we look at some of the state-of-art game engines available for game development. Then, we focus on two case studies of student-developed game projects using XNA. XNA is a programmable environment from Microsoft and is used as a dual development platform for Xbox and PC.

Game Engines

Games technology has seen tremendous growth recently. Game software development has also seen rapid changes with the introduction of new game engines. Even older

game software have new features being added to them to make the engines more powerful. We take a look at some of the advances available today that help in game software development. Game development involves two distinct processes. In the first process, known as the *game construction process*, the game developer builds the game as an offline process using the game engine. The second process known as the *run-time process* is when the game is played interactively with the help of the game engine. From a software perspective, the *game construction process* prepares the game data and identifies the necessary structures in the game data. These efficient structures that have been identified as part of the construction process are then reused during the *run-time process* when the player interacts with the game.

Developers either rely on an existing game engine to build a game or in some cases build a whole engine from scratch. At the top level a game engine is a software architecture that fits together seamlessly several components such as: player interaction, modeling, animation, graphics, physics, AI, audio, and networking. Each one of these components can by itself be a powerful utility or library that helps in game development, both in the *game construction process* and the *run-time process*. A state-of-the-art game engine supports scene management with complex terrain, landscapes, or closed buildings. By using efficient scene management techniques it ensures that the scene can be navigated and rendered in real-time. The engine's networking feature supports consistent scene representation and rendering for multiple players on a network. The realistic behavior of the objects and characters in the game is enhanced by the game engine's AI, physics, and graphical rendering features. The player's immersion is further enhanced by the novel user-interface features and audio features of the game engine. Hence, a game developer needs to master these features first to build a game as part of the *game construction process* and to efficiently play the game during the *run-time process*.

Distinct Features of Game Engines

There are several game engines available to developers. Some are open-source engines, whereas others are commercial. We highlight the distinct features of some of the engines here.

The Doom 3 (2008) game engine technology pushed the frontiers in three distinct areas: (a) Unified lighting and shadowing; (b) complex animations and scripting that show off real-time, fully dynamic per-pixel lighting and stencil shadowing; and (c) graphical user interface (GUI) surfaces that add extra interactivity to the game. The key advance of the Doom 3 graphics engine is the unified lighting and shadowing. Rather than computing or rendering lightmaps during map creation and saving that information in the map data, most light sources are computed on the fly. This allows lights to cast shadows even on nonstatic objects such as monsters or machinery, which was impossible with static lightmaps. A shortcoming of this approach is the engine's inability to render soft shadows and global illumination. To increase the interactivity with the game world, id designed hundreds of high-resolution animated screens for

in-game computers. The crosshair acts as a mouse cursor over the screens allowing the player to use a computer in the game world. This allowed an in-game computer terminal to perform more than one function, such as a readily apparent door-unlocking button, combined with a more obscure function allowing an astute player to unlock a nearby weapons locker. Other important features of Doom 3 engine are normal mapping and specular highlighting of textures, realistic handling of object physics, dynamic, ambient soundtrack, and multichannel sound.

The Halo 3 (2008) engine is a proprietary, in-house graphics engine used to make Halo 3. It employs advanced graphics technologies such as high dynamic range, global lighting, and depth of field effects within cut scenes. The dynamic objects in the game cast real-time shadows on themselves and the environment around them, including the game's plant life. Halo 3 uses normal, bump, and parallax mapping to give surfaces more detail without dramatically increasing the number of polygons.

Microsoft's XNA is a new software package that provides game developers the tools to make a game. It hides the lower level, more hardware-specific code and allows developers to concentrate on making their game. The Microsoft XNA Game Studio provides a set of tools, complete with a managed runtime environment, facilitates computer game design, development, and management that brings all aspects of game production into a single system. Novices in the game industry such as hobbyists, students, and indie developers will be able to get their first taste of building video games. Moreover, XNA enables them to bring their creative game ideas to life while nurturing game development talent, collaboration, and sharing that will benefit the entire industry (Goth, 2006). Game content development costs continue to rise especially for major titles. Microsoft XNA allows programmers to integrate artistic elements, physics, and audio early in the development process so that they can see the game as a whole soon after the initial storyboard game concepts.

The CryEngine2 is an extended version of the CryENGINE, the game engine behind FAR CRY. CryEngine2 is currently being used for Crytek's newly released game, CRYSIS, and is planned for use in a massively multiplayer online role-playing game (MMORPG), ENTROPIA UNIVERSE, which uses a real cash economy.

In this section, we have presented some of the development options available for game developers. The challenges become nontrivial when the games are developed for online deployment (Morgan, 2009). These game engines and others present the developer with two conflicting interests. On the one hand, the rich features in the game engines definitely help in producing games that can seamlessly use the features to make better games. On the other hand, these features pose a big learning curve and can be used only by advanced developers. We present the development of two games using XNA.

Ghoulash Using XNA

First, we discuss the making of a game that is multiplayer via the Internet. We pay particular attention to adapting to the new software and the obstacles faced in creating the game and getting it online. XNA software is used in conjunction with Visual Studio

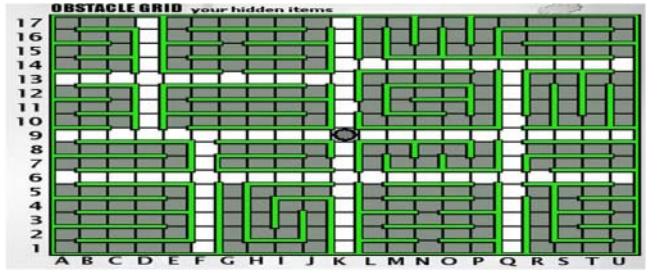

Figure 1. A typical game grid

to allow the creation of games. XNA can compile higher level programming language such as C# to produce executable code for Windows platforms and the Xbox 360.

We decided to implement a game that exists as a pen-and-paper game called Ghoulash (www.ghoulash.com; Figures 1, 2, and 3). The game is a more complex version of the pen-and-paper game Battleships. Two players each have 2 grids, on one grid they hide items which will affect their opponent, and on the other grid they track their own movements. The two grids are identical and show buildings joined by streets.

When it is the players' turn, they track their movements on their grid and will continue moving until they encounter an item. When it is the opponents turn they will track their movement on the grid with the items hidden and will be alerted when an item is encountered. The game is turn based with a player alternating between the roles of player and games master as they navigate between their opponent's hidden items and their opponent navigates through the player's secret items.

The game starts (after connecting to a player) in Item Placement mode. Here the two players hide all their items for the other player. Here is a list of all items:

- **Ghouls:** Can only be placed in buildings (shaded areas) and only 3 in any one building. If encountered a Ghoul battle occurs
- **First Aid Kits:** Can only be placed in buildings and one per building. These recover all health though can tactically be left behind
- **Holes:** Can be placed anywhere and makes the player miss the next turn
- **Debris Zones:** Can be placed anywhere and inflict one wound on the player
- **Food:** The object of the game is to find the food and return to the center square

Though the number of items and the items themselves change within the paper-based game, in our implementation there are 12 Ghouls, 5 First Aid Kits, 10 Holes, 10 Debris Zones, and 1 Cache of Food. We have also implemented two scenarios (two-level layouts) but both have the same number of items.

Health and ghoul battles. Each player has a wound meter consisting of 15 units. If they lose all 15 units within a game, they die, thus making the other player the winner. Debris inflicts one unit of damage and remains active throughout the whole game. Holes do not

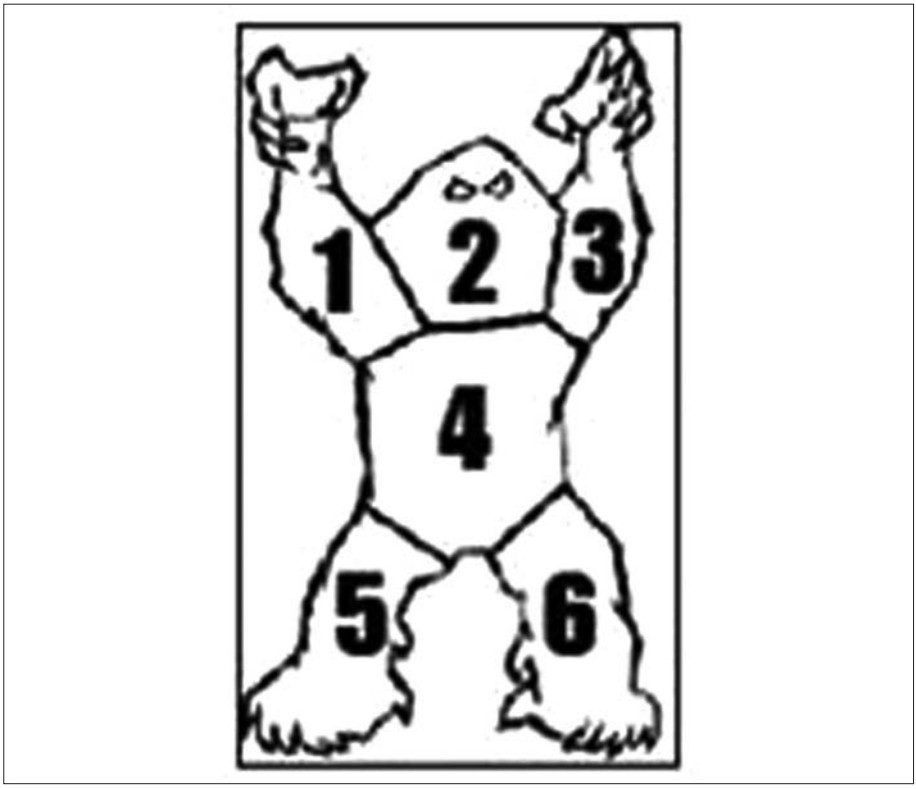

Figure 2. Sections of the ghoul

Figure 3. The sprites that make up a player's movement

inflict damage; however, the player loses a turn but they also remain active throughout the game. First Aid Kits heal all wounds; a kit can only be used once and can be left behind for tactical purposes. Ghoul battles occur when a player finds a ghoul. When a player finds a Ghoul the opponent secretly chooses a weak spot. The player must then shoot at a section of the Ghoul. If the weak spot is hit, the Ghoul dies and is dead for the remainder of the game. If the player misses, they incur one wound and must continue shooting until the Ghoul dies meaning a worst case scenario of five wounds incurred.

Winning. A player can win in two ways. By finding the food and returning to the center square or by surviving longer than their opponent. If the opponent dies, they forfeit the game. Players needed to be moved using the keyboard and items needed to be placed using the mouse. The look and feel of the game and its UI needed to be addressed. The original game has a very clear look that we captured by using the original artwork with permission. There is also a lot of UI (one A4 page) that we had to fit on screen.

Internet connectivity. The game was always intended to be online and was aimed at a wide audience so we needed a simple and convenient way of offering connectivity. We also needed an efficient way of sending data between players to keep the connection functional. The two players connect to each other directly using sockets. A player can either host a game or join a game being hosted by a friend. This requires the host to give their IP address to a friend for them to connect. The process of sharing an IP address would be too tricky a concept for small children so we created a friend code. This converts the numbers 0-255 into a 5-6 letter word effectively turning an IP address into a much easier to remember 4-word friend code. As an example, we currently have an IP address 80.2.21.243, which translates to "Magnet Plane Jelly Robot"—a code which is much easier to remember and share with friends.

Development. Coding within XNA was quite simple. The initial template available when starting a new project in XNA is very easy to understand and to get started with. Collision detection is easy to model as the game is 2D and the wall positions are known prior to the start of the game. The logic and program flow all works well, the inclusion of new graphics and fonts is simple to achieve and as a developer you really can concentrate on the game and its content.

Discussion. XNA is a must for hobbyist developers or educational institutions looking for a tool to teach programming to their students. With XNA, games can be generated quickly and easily, and there is also the scope to create much more advanced games, always without the requirement to get involved in the more complex, lower level code that often puts off many programmers and can slow down a project.

Our game is available from http://www.quixity.com/cdodd/ along with documentation. Future expansion will involve improving the Internet code and developing more scenarios and more complex items. Game balancing is also an issue that could be looked at further and possibly the development of an AI opponent for those who wish to play offline. The release of the new version of XNA will no doubt result in code updates and improvements.

BUMPER Racing in XNA

The second game in our XNA case study is BUMPER. It is a chasing and racing game. A racing starter kit has been used that is available from the XNA Creators Club (2007), so that most of the core parts such as graphics and audio are already in place and ready to use. *The Racing Game Starter Kit* provides a complete starter XNA Game Studio Express game, including source code and game content such as models, textures, and sounds. The starter kit demonstrates several features that include advanced graphics

using shaders and postprocessing effects; game state management through UI and gameplay screens; and simple automobile physics, including collision detection. To build a new game, the starter kit is added to the project's template.

Storyline. The patrol radio reports that a lawbreaker has escaped to the race track. It is now the job for the player to use the nearest patrol car called Bumper to capture the lawbreaker. The criminal's car will overtake the player from behind. However, the criminal's car is made with a special armor that makes it hard to block its way or to puncture its tires. The only hope is on Bumper that is made with special material that is hard as steel and has a sporty look too. The player has to drive Bumper to bump the car to wreck its armor so it can be captured. However, it must be done quickly before the car reaches the end of the race track.

Gameplay, logic, and rules. The game is presented in third person view, which means that the player can see the model of the Bumper from the back and side view and has the wider view. A special material made from steel is attached on the front of Bumper to make it withstand the collision. Therefore, Bumper can hit the car as many times and as hard as it can to destroy the car before the car crosses the finishing line. When Bumper hits the car, the collision will be rendered with sound and particle emissions. The armor indicator will be reduced depending on how hard the collision is. If the opponent car is controlled by the computer, it is controlled with AI features that enable the car to follow the track with varying speed and control.

Game elements. The game elements are the environment, cars, score, audio effects, and visual effects. The chase takes place in mountainous environment in hot sunny weather with the long race track that has been build for the hot pursuit. Bumper is the main car controlled by Player 1 to pursue the criminal's car. The car is attached with special black steel in front of it. The criminal's car is the car that has been chased by Bumper. It will avoid itself from getting hit by Bumper until it cross the border at the end of the track. The score is based on the time taken by Bumper to destroy the armor of the criminal's car. The quicker it destroys the armor, the higher the Bumper's points, and lower is the criminal's (second player) points. When the player passes every check point, additional points are gained. Particle emission produced shows the dust and impact of the collision whenever the Bumper hits the criminal's car. Camera transformation is used to facilitate the spring game physics to enhance the car suspension. Some of the audio effects include running engine, tires screeching, car collision, and armor. In addition clips have also been included for sound and music such as in Main Menu, Time Critical, and Win or Lose screens.

Development. We used the game engine provided by the Racing Game Starter Kit (Center, 2007) We present a summary of the implementation (Figure 4). The landscape constructor loads the height map data from the level file. The car, buildings and tree models in the .x format are loaded. The programming interface is used to take advantage of hardware acceleration capabilities to display 3D objects. The features of the physics engine are used to add realism to the game.

Discussion. What can be seen in the BUMPER game development is that XNA provides a game components pluggable mechanism, where it permits us to create our own

Figure 4. The result of the unique matrix implementation

game components, thus making the contents reusable and easy to just plug them with the game. Furthermore, it provides many components that can be used to get the game running on the Xbox platform. This game is extendable to a multiplayer version that enables the user to play this game online and share their game content. The initial idea of implementing multiplayer game is to create an array of player objects. Finally, to ensure the content reusability and extensibility, the game should have its own editor such as a track editor and player's car customization. Track editor enables either users or developers to create tracks based on their own interests or enable the users or developers to upload their own track data to be rendered in the game. In other words, by engaging the player in game content development, game development benefits both from the game developer's perspective and also from the players' perspectives.

Summary

In this section, we have presented two case studies on game development using the XNA environment. The results show that training in game software development at

the university level has come a long way. Several powerful game engines that are available today for game development can be used in the training. Game developers hence have a choice to use public domain engines or commercial engines. The game engines vary in their capabilities. Students who are trained using powerful game engines are ready to take on responsibilities when they start work in a game development company or in some cases are also ready to start up their own game development enterprise.

Terrain Modeling and Rendering

Terrain is a key component for many computer games, in particular real-time strategy, first person shooters, and role-playing games. The popularity of massive multiplayer online games (MMOGs) with large numbers of participants has in the past few years focused attention in particular on the problems of performance and quality when modeling and rendering very large terrains. This report surveys the major issues that have occupied game programmers, in relation to the problems of terrain generation, over the past 10 years and looks at current trends as a prelude to the development of a general purpose reusable terrain function library. A prototype system developed as a final-year student project is also described.

Terrain generation has proved problematic from the very beginning. In 1998, the game INCOMING (Rage Software Ltd) used a reduced far clip plane to limit the number of polygons that needed to be generated during the rendering stage. Popping, as new areas of the world came into view, made the visual effect less than satisfactory. The Silent Hill series of games (1999) employed a similar technique but added thick fog as shown in Figure 5 to hide the popping on the horizon. This actually lent a particular trademark atmosphere to these games but only provided a stop-gap solution. Since that time, the problem of generating the massive number of triangles usually associated with a terrain has been approached using level of detail (LOD) techniques whereby the number of triangles in the terrain mesh is systematically reduced. In general, this is achieved by drawing lots of triangles in areas of the mesh that are close to the camera and fewer triangles in those areas that are more distant. This approach works well and is the basis for majority of existing terrain renderers whether they are for games or indeed other application areas. The use of LOD techniques does, however, introduce a number of issues and a number of different approaches to LOD rendering have been suggested. The problems of LOD are the main focus of this section.

Level of Detail

Level of detail applied to terrain generation is inherently more complicated than with other games objects because the terrain is a single entity so that detail level needs to change across a single object. Some parts of the mesh are close to the viewer and therefore require substantial detail whilst other parts can be adequately represented with just a few triangles. The problem in terrain modeling and rendering therefore becomes one of eliminating triangles in selected parts of the mesh. However, some parts of the mesh

Figure 5. Fog effect on terrain in the game Silent Hill

have lots of height variations or local bumpiness where reducing the number of triangles would lead to large differences in the original landscape through approximation and this needs to be avoided. Pajarola and Gobbetti (2007) developed a LOD technique where triangles have in general been removed according to distance from the viewpoint. However, distance is not the only criterion. Some relatively distant areas maintain a dense mesh to capture substantial local variations in elevation at those locations.

The history of terrain modeling and rendering is characterized by a large number of different approaches to achieving LOD in a terrain mesh. Each of these techniques can achieve an improvement in performance but cannot be judged and evaluated solely on the issue of performance because each has implications for image quality, ease of implementation, and so on.

In all cases, LOD algorithms selectively remove grid points from the original height map in a dynamic fashion so that fewer triangles need to be drawn. In general, triangles with less importance are omitted from the grid. The most common way of deciding which triangles should be left out is via the application of an error metric at specific points in the mesh. This takes into account not just distance but also changes in elevation per unit area. Some techniques apply LOD on a per triangle basis whereas others reduce the density uniformly across a block of grid positions.

Issues in LOD Terrain Modeling and Rendering

The major issues in terrain modeling and rendering are the size of the terrain, the performance and quality of terrain model representation and rendering, as well as the ease of implementation.

Height map based mesh. The games terrain mesh is normally described using a grid of height maps. Although terrain synthesis using techniques such as fractal iteration can be very effective, most games use a predefined 2D array of elevation values linked to a regular grid whose spacing determines the size of the terrain in game space. Height maps are preferred mainly because specific terrain features are often required for the game play and fractal techniques provide a generic solution where individual features are not easily generated or controlled. On the other hand, a height map can be generated via a simple paint program using a color to height association. Moreover, a number of sophisticated utility programs are now available, such as Terragen and L3DT, which allow the designer to use high-level tools and algorithms to build terrains represented by means of a height map. These systems permit a variety of useful features such as procedural textures and light maps to be generated automatically using predefined texture tiles, ray casting to generate soft shadows and erosion features. Fractal-based terrain synthesis techniques can be added to a primary height map representation.

Performance. Rendering speed is the most important issue in terrain modeling and rendering. As games become increasingly sophisticated terrains are getting larger in size to accommodate the game play. Consequently, it becomes difficult to render a terrain and take care of all the other game activities at the 30 frames per second rate needed to maintain a steady animated image. This problem is shared with other terrain applications such as geographic information system (GIS) where large data sets of whole countries or planets need to be rendered and indeed many of the techniques used by game programmers have been borrowed from other application areas. Over the past decade, a number of LOD techniques have been developed and used in games. These have largely been CPU based. However, with the development of programmable graphics boards, CPU-based techniques are now being proposed. The brute force method even when CPU based is seldom adequate. Even when high frame rates can be maintained a uniform mesh can cause aliasing problems as in texture mapping where several triangles map into one pixel.

Quality. This is perhaps of less importance to the game programmer than performance. Nevertheless, there are some issues. When level of detail algorithms are used the true elevation grid is approximated to a greater or lesser extent. This loss of detail may lead to a situation where the illusion of reality is compromised or where key features of the game play become less effective. The degree of approximation and its distribution does differ depending on the LOD algorithm chosen.

Terrain deformation: The TREADMARKS (Longbow, 2000) game pioneered the use of run-time modifications to a terrain. Shells hitting the terrain would cause craters and these craters would become an obstacle to vehicles and an integral part of the game play. Davison and Tang (2001) point out the potential of deformable terrain; raising or lowering areas, creating chasms and earthquakes can become important features of the game play and of realism. In TREADMARKS, the terrain texture could also be changed to show track of vehicles, burn marks, and so on. This is potentially a very important facility for game designers in real time strategy games in particular.

Ease of implementation. One of the problems that LOD techniques gives rise to is the algorithm complexity and implementations that require extensive development time.

Game developers usually work to tight deadlines so there is a tendency to favor simpler algorithms as long as they do the job. This will be less of an issue with a terrain engine library because the functions will be reused.*Dealing with large terrains.* Several authors point to the importance of paging. Terrains are often extensive and it is not possible to hold the whole scene in memory at one time. A terrain modeling representation ideally needs to be set up in such a way as to allow the streaming of grid data.

Tile-Based Approaches

An approach that has been used for many years in games programming involves dividing the mesh into fixed-size blocks. At program initialization, each grid block is processed to uniformly merge triangles and provide new grid blocks at a series of lower detail levels. This is a relatively simplistic approach but one which is easy to implement and potentially very fast. Snook (2003), for example, describes an interlocking tiles approach.

Levels of detail are precomputed and stored in the graphics hardware using static vertex buffers. The use of static vertex buffers gives a significant boost in performance. Separate edge tiles are stored to solve the problem of gaps arising between tiles. The method lends itself very well to large terrains; the tile-based approach means that tiles can be streamed in from disc as required and the whole terrain does not have to be held in memory at one time. The major problem with the tile-based approach is loss of accuracy because local undulations in terrain may be lost. Clearly, the extent of this problem will depend on the size of the tile but with more tiles comes greater complexity and loss of performance. The tile approach was also used by McNally in the game TREADMARKS though he does combine it with a bin tree approach described later to gain greater accuracy.

Quad Trees

The quad tree approach aims at higher accuracy. Lindstrom et al. (1996) start with the original grid structure and progressively merge tiles from the bottom up to form a quad tree structure. The bottom-up approach is chosen because it is assumed that the target level of detail will be closer to the maximum detail level than the lowest and the algorithm will therefore run more efficiently.

At initialization, height field simplification is carried out whereby triangles are recursively merged until a desired level of detail is reached. For each possible merge, an error metric is stored. This metric is the difference between the height of the hypotenuse midpoint on the parent triangle and the underlying mesh grid point.

The error metric measures the degree of approximation involved when a merge is performed. At run time, each node of the tree is processed and this error metric is combined with the viewpoint distance to determine if the merge should be performed or not. A threshold value must be decided in advance and this will be set at a level that reflects the capability of the hardware and the proportion of machine cycles available

for terrain generation. Each time the camera moves the mesh needs to be regenerated. However, because the camera moves relatively slowly per frame only small changes in the mesh structure will be necessary. This technique has been used extensively, for example, in flight simulators and in games such as COLIN MCRAE RALLY (Codemasters, 1998).

The advantage of using quad tree structures is that a relatively stable number of triangles are required frame to frame and that the level of detail changes smoothly across the mesh; there are no large jumps in level of detail. Accuracy/quality is achieved compared with a tile-based approach because large elevation variances within a small area are taken into account even when an area of the mesh is in the distance. Nevertheless, the mesh must be recomputed by traversing the tree each frame even though only small changes may result. The most significant drawback, however, is that this method does not exploit modern hardware capabilities.

Roam Bin-Tree Approach

Another very influential continuous level of detail approach is the Roam algorithm (Duchaineau et al., 1997). This method is noteworthy for the frame coherence features it incorporates. It also offers per triangle level of detail but on this occasion using a top down approach. A binary tree is built by taking the original grid and dividing it into two triangles. Each triangle then forms the root of a binary tree. The tree is formed by splitting the parent triangle into two by joining the midpoint of the hypotenuse to its opposing corner. As with quad trees, an error metric is stored at each node indicating the disparity between the elevation value of the hypotenuse midpoint and the grid point that lies beneath it.

This technique allows view frustum culling against each triangle as we descend the tree. Roam maintains two queues held as linked lists—namely, the split and merge queues. When a triangle is split it is placed on the merge queue and when a triangle is merged it goes onto the split queue. These queues are sorted by priority (based on the error metric). A new mesh can be generated from the old mesh by moving through these lists merging and splitting until the error threshold is reached. Triangles can be split and merged either until a set number of triangles have been included in the mesh or until a certain frame rate has been achieved the best detail in the time allowed.

In the original Roam, merge/split queues are used to provide frame coherence but game programmers have generally used a split only version of the algorithm to reduce complexity. In this scenario, a new tree is generated for each frame. The split-only approach facilitates terrain deformation because the tree will always need to be regenerated if the geometry of the underlying grid changes.

There have been claims that this top-down bin-tree approach is faster than the bottom-up quad-tree approach but this is debatable and doubted by some commentators. Nevertheless, Roam has proved to be a popular algorithm for games programmers.

Geo Clip Maps

Recent developments in terrain rendering have promised the possibility of rendering very large terrains in real time using Geo Clip Maps (Losasso & Hoppe, 2004). With this approach most of the work is moved away from the CPU to the programmable graphics board. Losasso and Hoppe reports 90 frames per second for flyover of 20 billion sample grid. The algorithm is analogous to the use of MipMaps in texture filtering. A series of grids each a quarter the size of its predecessor is produced from its predecessor by removing grid points. Several regions of are then defined as rings surrounding the viewpoint. Each region links to a particular level of detail in the pyramid based on its distance from the viewpoint. Lower levels (high detail) are selected for the central region surrounding the viewpoint. Higher levels (low detail) of detail are then selected as we move out through the rings.

Of course the position of the rings on the underlying mesh change as the camera moves across the terrain so that the mesh structure has to be reformulated from the pyramid each frame. This, however, can be achieved on the graphics board using vertex Shader programming. The grid is initially loaded into a vertex buffer and elevations are read into the pixel shader as a texture map. The pyramid is then automatically generated by the hardware using its MipMapping capabilities. The fact that the latest boards allow the vertex Shader to access textures in the pixel Shader provides a mechanism for integrating these data structures at run time and recalculating the rings.

The results obtained from this technique show that performance is indeed exceptional but there are drawbacks:

- The pyramid structure means a doubling of memory requirements for the grid.
- The method is akin to a tile based approach in that large areas are fixed to a particular detail level even if there are large differences in elevation within that area. In other words it does not have the per triangle accuracy of the quad- and bin-tree approaches.
- The method relies on the grid being stored in graphics board memory and this has consequences if we wish the terrain to be modified in the middle of the game play. The whole grid would need to be resent down the bus with its associated low transfer rates.

Geomancy

We use OpenGL extensively for teaching in the game program. There are many advantages to this approach in that it provides students with an industry standard skills and an appreciation of key problems and issues in games programming. OpenGL is, however, a relatively low-level graphics programming language and it is only possible for students to develop fragments of a game during the time available. Geomancy was developed as a final-year project by Matthew White (2007) to provide a high-level library of terrain functions that could be invoked from an OpenGL program. A rendered snapshot of the terrain is shown in Figure 6.

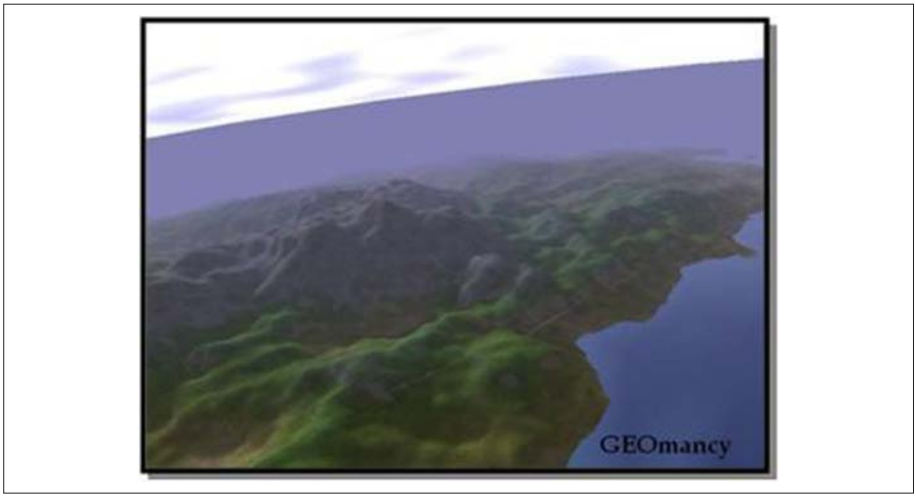

Figure 6. A Geomancy-generated terrain

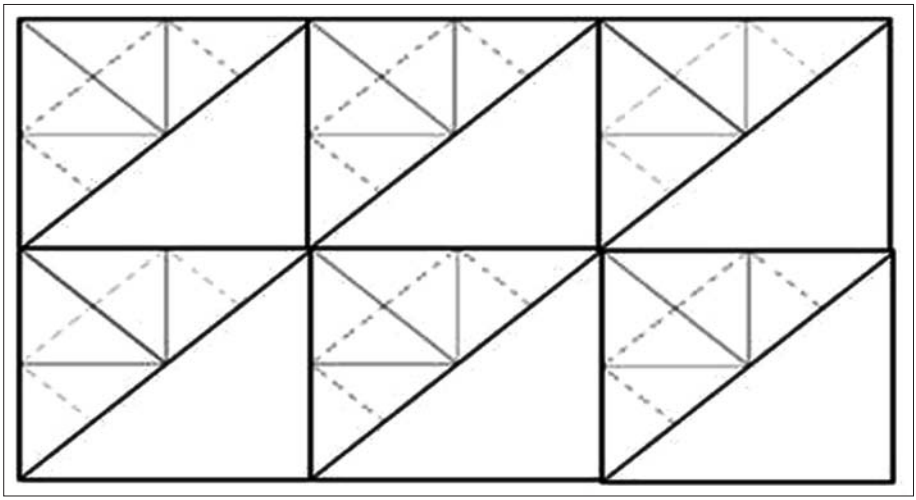

Figure 7. Geomancy tile system

Geomancy uses a hybrid approach that attempts to combine the performance and flexibility of a tile-based approach with the accuracy of a per triangle Roam-based algorithm. The terrain is divided into a set of tiles at initialization (as shown in Figure 7) and a bin tree constructed separately for each of these tiles. Roam split and merge queues are used to provide frame coherence.

Table 1. Performance Data for Geomancy

Error Metric (Screen Pixel)	Minimum Frame Rate	Maximum Frame Rate	Average Frame Rate
0.2	73.6	80.2	76.9
1.0	69.8	80.3	75.1
3.0	82.3	89.1	85.7
5.0	87.4	92.9	90.2
7.0	91.3	97.5	94.4

For each tile, a full detail grid is loaded into a vertex buffer and stored in graphics memory. The triangles needed for a particular computed level of detail are then identified by an index buffer, which also resides on the graphics card. At run time, the bin trees are modified following movement of the camera and changes in the bin tree are computed and used to update the index buffers. There is an index buffer for each tile but because per frame changes are small in Roam when split merge queues are used then not all index buffers need to be sent when a new frame is composed.

The tile-based approach means that paging for large meshes can be added as a facility in the future.

The Geomancy engine has per triangle accuracy, and is fast (see performance shown in Table 1) because of the use of Roam split/merge queues and the use of vertex arrays. The use of tiles segments the terrain to minimize transfers from memory to the graphics card. The engine has a high development time overhead but as the engine is a reusable component for student development purpose this is a price that is affordable. The main drawback of the engine is the need to replenish vertex buffers if and when the terrain grid is modified. At the present time any attempt to exploit the graphics card in a significant way to enhance performance faces this problem. There is a fundamental issue in providing both terrain deformation capability and speed given current technology but the tile-based approach provided a compromise in that small-scale deformations will only affect a few tiles. The segmentation of the terrain in this way means that updating the on board vertex arrays can be limited in extent.

Summary

Over the past decade, a variety of approaches have been devised to render large game terrain accurately. This section has reviewed the development of terrain algorithms over this period. We have identified a set of issues of concern to those developing game terrain engines and looked at how these issues have been addressed by a variety of techniques. At the present time there are two competing paradigms—one is based on tree structures, which permit a continuous, per triangle, LOD. This provides a high-quality solution but requires the maintenance of relatively complex data structures and places a heavy computational load on the CPU whereas the graphics card is, in

contrast, underused. The second and more recent development uses a series of discrete levels of detail managed very largely by the CPU. This latter approach offers impressive frame rates over very large terrain but loses per triangle accuracy and does not lend itself easily to terrain modification. As graphics hardware becomes more sophisticated it is likely that per triangle accuracy based on more complex data structures will become available to Shader programs. Until then developers need to carefully consider the requirements of their engine. Does the need for fast movement over massive terrains take precedence over accurate local details and terrain modification? Our survey suggests that a single general-purpose terrain engine is not currently feasible.

Human Behavior Modeling for Crowd Simulations

In this section, we describe how to generate realistic human behaviors, which is a challenging and interesting problem in many applications such as computer games, military simulations, and crowd animation systems. First, we propose a generic behavior modeling framework that reflects the major observations on human behavior in daily-life situations. Next, we describe a case study to illustrate how this modeling framework can be used in various crowd simulations.

Human crowd is a fascinating social phenomenon in nature. In some situations, a crowd of people shows well-organized structure and demonstrates tremendous constructive power. In other situations, people in a crowd seem to abandon their social norms and become selfish animals. Numerous incidents with large crowd have been recorded in human history, and many of these incidents have led to severe casualties and injuries (CrowdDynamics, 2007). How to predict and control the behavior of a crowd on various events is an intriguing question faced by many psychologists, sociologists, and computer scientists. It is also a major concern of many government agencies.

To set the scope of our work, we first classify the research in crowd behavior modeling along two dimensions, namely *the size of the crowd* and *the time-scale of the underlying process involved*. Along the first dimension, existing research in crowd behavior modeling can be broadly classified into two categories. The first category targets huge-sized crowds with thousands or even hundreds of thousands of individuals. Because of huge size of the crowd, research in this category usually treats the crowd as a whole and focuses on the global trend of the crowd on some events. The second category targets small- to medium-sized crowd with tens to hundreds of individuals. The relative small size of the crowd allows research in this category to model the behavior of the individuals in the crowd. In this article, we call the research in this category *individual-based approach*.

Along the second classification dimension, the work in crowd behavior modeling can be classified into two categories. The first category put emphasis on the *long-term crowd phenomena*. The underlying processes for these phenomena have a relative long time period. In contrast, research on short-term crowd behavior investigates how a crowd will respond to various events such as emergencies and threats, *given the crowd composition and the social, psychological, physical characteristics of the individuals*

and groups in the crowd. The time scale of the underlying process of interest is usually of the order of minutes to hours.

Our work is on the investigation of the short-term behavior of a medium-sized crowd. Typical applications of our research include the investigation of the pedestrian behavior in a downtown area and the investigation of the fire evacuation process in a shopping center. We adopt the individual-based approach, and aims to investigate how the crowd process is determined by the individuals' behavior and various social/psychological factors.

A key issue in the individual-based approach is how to model the decision-making process of the individuals in a crowd. These different decision-making methods have been successfully used in different applications. However, most of these methods are focused on the mathematical and computational framework rather than on the imitation of the way that real humans make decisions. In our work, we propose a human behavior modeling framework that naturally reflects the human decision-making process.

Next, we present our behavior modeling framework. Then we describe the navigational behavior model, which is an important component of the proposed framework. Finally, we present the simulation case studies.

Behavior Modeling Framework

We believe that the modeling of human behavior should start with some basic observations on how humans behave in various daily-life situations. Based on these observations, a generic modeling framework could then be developed by imitating how humans behave in these situations in a simulated environment.

Observation 1: Humans are social animals. The social aspect and the animal aspect of a human being are inhibitory to each other. By the animal aspect of a human being, we refer to his or her basic needs and behavioral features *as an animal*. Just like an animal, a human being needs to drink when thirsty and to eat when hungry. He or she can also move, run, cry, fight, even kill, and so forth. By the social aspect of a human being, we refer to an individual's social needs and behavioral features *as a social entity* in the society. In normal situations, the animal aspect of a person is often inhibited by various social norms. However, when a person has a strong physical need such as seeking comfort or places when threatened, the animal aspect of the person may show up. Therefore, given the context of a situation, the relative strength of the social aspect and animal aspect plays a key role determining in how a person will behave.

Observation 2: A person's behavior is largely determined by his or her experiences rather than by some complex decision rules. It is fascinating to observe that humans are able to handle various everyday tasks with ease, although many of these tasks seem to be quite challenging from a computational intelligence point of view. We have heard frequently of something like "I act in this way simply because I used to do so" as the reply when we ask a person the reason that he or she reacts to certain events. People often need to make everyday decisions under time pressure and with incomplete information. For

such situations, we believe that the *naturalistic decision making*, for example, recognition-primed decision (Klein, 1999) is a good model of people's decision-making process that emphasizes the role of a person's experience in determining his or her behavior. Therefore, considerable effort should be put on representing an individual's experience and the recognition process.

Observation 3: An individual may behave quite differently if he or she is put in a group/crowd. A person's behavior is greatly affected by his social groups such as family, friends, colleagues, and so on. While in a group, a person tends to seek approval from his or her group members regarding how to behave in a given situation. Thus, people in a group tend to behave in a similar manner (Festinger, 1954). Consider an individual in a crowd attempting to evacuate from a firing building. If this person is acting alone, he or she may just try to escape immediately from the building. However, if the same person is with a group of friends, he or she may need to *wait till consensus opinions are reached* among the group members regarding whether to evacuate before escaping *together* with other members in the group. Therefore, social psychological and social organizational factors need be investigated.

Observation 4: A human's decision-making process consists of multiple layers of microlevel/macrolevel interactions. One of the major reasons that human beings are very efficient in handling various everyday situations is that they usually adopt the *divide-and-conquer* strategy in decision making. To achieve a certain goal, a person usually divide the whole task into multiple related subtasks, then find out ways to complete each of these subtasks one by one in a serial manner. Here, we refer to the dividing of the whole task into multiple related subtasks as macrolevel decision, and refer to finding out ways to complete the subtasks as microlevel decisions. Intuitively, macrolevel decision is to determine *what to do*, whereas microlevel decision is to determine *how to do*. A subtask may be further divided into some smaller tasks until these smaller ones can be easily executed by the person. Therefore, a layered behavior modeling architecture is needed to reflect this macrolevel/microlevel relation of a human's decision-making process.

Based on the above observations, we propose a generic behavior modeling framework as illustrated in Figure 8. The framework reflects the *perceive-decide-act* paradigm and emphasizes the role of experiences in human decision making. The sensory information about the virtual world needs to be processed by the agent's perception system, which is influenced by the agent's attention and general knowledge about the world. According to social schema theory (Augoustinos & Walker, 1995), how a human perceive a situation heavily relies on various social schemas in his or her long-term memory.

To make sense of a situation, the perceived cues of the current situation are compared with the situation patterns in an agent's experience repository. An agent's experiences are represented by a set of <situation, action> pairs. Then the action corresponding to the situation pattern that is closest to the perceived situation cues will be selected and passed to the *Action* module for execution. The selected action is often at high level, that is, it tells the agent *what-to-do* rather than *how-to-do*.

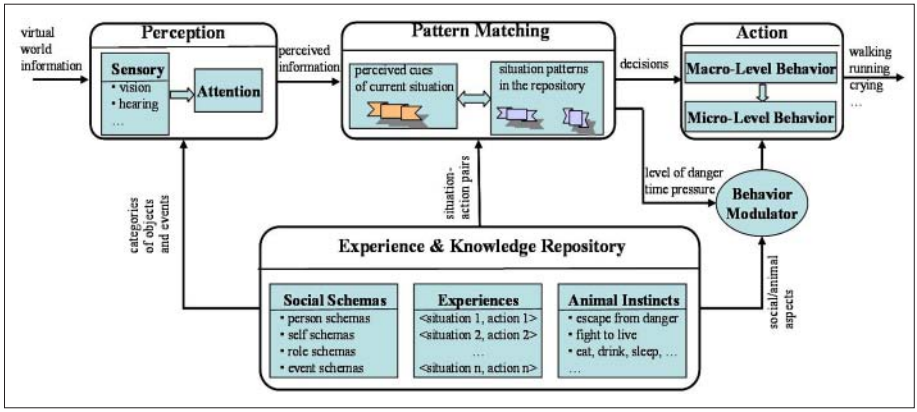

Figure 8. A generic behavior modeling framework

Thus, the execution of a decision often involves the interaction between macrolevel and microlevel behaviors.

The proposed framework is generic in the sense that it reflects the major observations that we have made on human behavior in daily-life situations. In particular, it emphasizes the role of a human's experience and social relations in determining his or her behavior.

Navigational Behavior Model

An important aspect of a person's every behavior is how he or she moves around in the world. In fact, much existing work in crowd simulation focuses on the navigational behavior of the individuals in the crowd. In this section, as an example to illustrate how the interaction between macrolevel and microlevel behaviors in the *Action* module of the proposed framework could be modeled in practice, we describe a two-level navigation model for crowd simulations.

The model consists of a *macrolevel* and a *micro-level* navigation module. The macrolevel navigation is responsible for planning a path from an agent's current position to a destination that is free from static obstacles. The microlevel navigation handles the routine movement between the subgoals of the planned path.

The macrolevel navigation module is executed whenever the destination of the agent has changed or if there is a change in local conditions. In our proposed model, we have used A* search algorithm as our primary path planner although other graph search techniques can also be used. Different heuristics can be used for our path planner for different application areas. The value of the heuristics may change over time as the agent moves along the planned path. Thus, an agent may replan its path if the newly calculated heuristics value is much higher than the original value. If no replanning is

needed, the agent will continue with its current path while relying on the microlevel navigation module to pass through the crowd along the planned path.

The microlevel navigation module is responsible for steering the agents away from potential collision with other dynamic objects. Although collision avoidance is one of the most common navigational behavior in people's day-to-day life, its importance is often overlooked when modeling autonomous agents. We feel that current collision-avoidance models are not adequate for modeling various important factors in real-life collision avoidance situations, such as a person's patience, social norms, experiences, and so on. Koh and Zhou (2007) have proposed an extensible collision-avoidance model for autonomous agents. The model allows individual agents to use their own frame of reference and reason on the gathered information. The choice of action for collision avoidance reflects the experience, capability, and preference of the agent. The model also allows other mechanisms to be implemented on top of our basic model to select a suitable steering that best fits an agent's needs and constraints.

Case Study

To show how the framework can be used to model crowd behavior for different application areas, we have conducted various case studies. In this section, we summarize some major findings in the simulation of pedestrian behaviors.

Pedestrian simulation has many applications in computer games, military simulations, and crowd animation systems. The focus of pedestrian simulations is usually on the navigational behavior of the pedestrians in urban environments. Thus, we only describe how realistic navigational behavior can be achieved using our proposed behavior modeling framework.

Our pedestrian behavior model has been prototyped using the *Game Studio A7* game engine. We have tested the performance of the prototype with various scenarios. Here, we summarize some major results of these tests.

Basic navigation capabilities. Figure 9 shows various behaviors that can be achieved by using only our microlevel navigation module. The *side-stepping* behavior as shown in Figure 9a is one of the most common response for pedestrians when a collision is predicted. The *overtaking* behavior as show in Figure 9b occurs when one of front agents moves slower than the minimum acceptable speed of the agent walking behind. This is another common navigation behavior for real pedestrians. The *following* behavior as shown in Figure 9c occurs when the agents are not in a hurry or it takes more effort than desired to overtake other agents. The separation behavior as shown in Figure 9d is another natural behavior for pedestrians.

Navigation in a complex environment. To navigate in a complex environment, the macrolevel navigation module is needed to work together with the microlevel navigation module. Figure 10 shows the results. In scenario (a), we let the agents move from left to right. In scenario (b), we have the agents moving in four different directions. The agents are able to follow their lanes in the intersection area, avoid other agents, and finally reach their destinations.

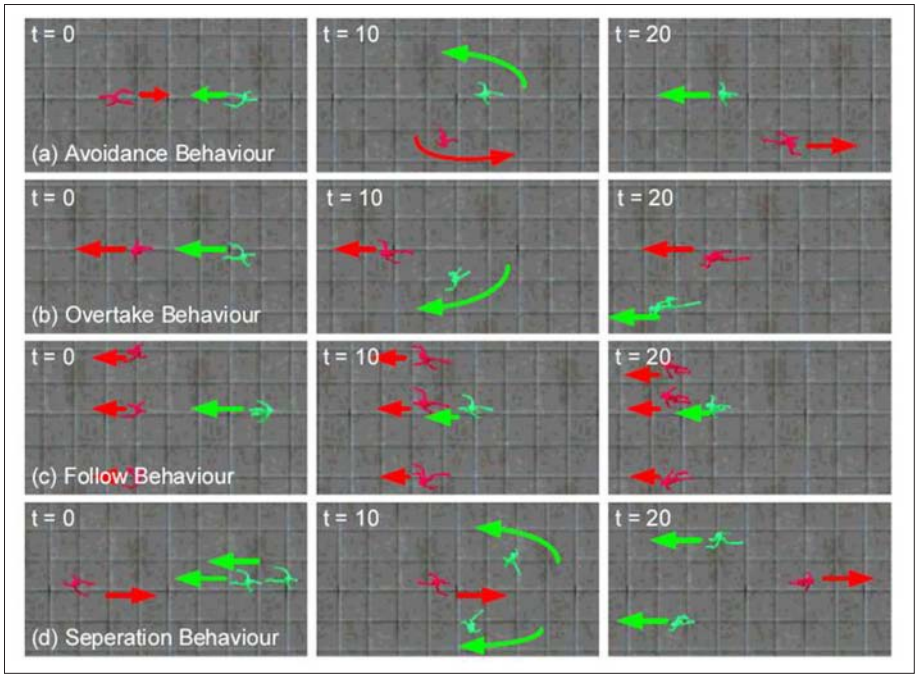

Figure 9. Different navigational behaviors generated by the microlevel navigation module

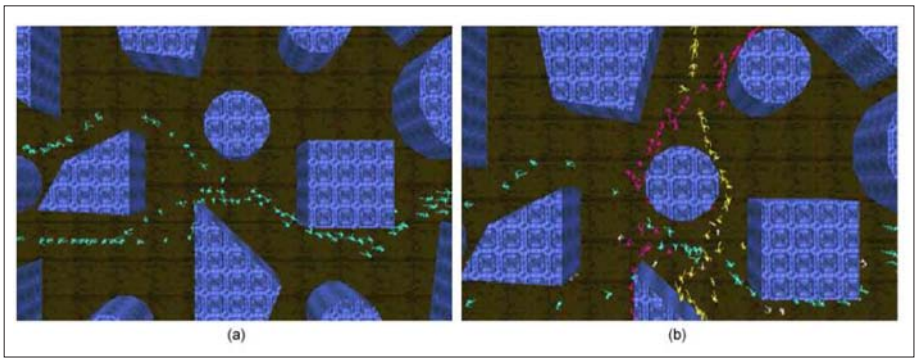

Figure 10. Pedestrian navigating through a complex environment with (a) one-directional flow and (b) four-directional flows

Summary

How to generate realistic human behaviors is a challenging and interesting problem in many applications. A generic behavior modeling framework has been proposed based

on some key observations on how real humans behave in everyday situations. We emphasize the role of human's experience in determining their behaviors in various situations. In addition, various social and psychological factors need to be considered. These factors may determine the relative strength of a person's social aspect and animal aspect. That is, they help to determine whether a human being will act according to his or her social role in the society or his or her social group, or will act like an animal.

As a final remark, we emphasize the social context when discussing human behavior models. It is impossible and also not necessary to build a model that covers all aspects in human behavior. To understand how a human behaves in certain situation, we must understand the situation and in the mean time, we must also understand the person regarding aspects like his physical capability, experience, personality, social relations that are deemed salient to the situation.

Player Adaptive Entertainment Computing

The entertainment computing area has gained much attention recently in the industry and in academia. Entertainment computing covers areas ranging from computer and digital games, mobile content delivery for entertainment, interactive media, entertainment robot, sociology and psychology in entertainment, and virtual/augmented reality for entertainment (Nakatsu & Hoshino, 2002). This area of the entertainment industry has become a highly competitive area. Recently, it can be observed that there is a shift to focus on the design of the entertainment media for individuals, so as to increase the perceived value.

Most companies are interested in their profit. Interactive media industry is no exception. The shift of the focus to develop something that can increase the perceived value by the user is similar to the customer relationship management (CRM), the term used in business world (Nykamp, 2001). One of the important models in CRM is personalization, which is to provide perceived value to customer when interacting with a business. In entertainment computing industry, the objectives of the designer or the developer are quite similar to that of the business, which is to increase the perceived value when interacting with the entertainment media. This is normally done by market research to identify the target group or preproduction concept pitch.

In this presentation, we present the concept of Player Adaptive Entertainment Computing (PAEC), which is to provide personalized experience for each individual when interacting with the entertainment media (Wong, 2007). For discussion and illustration of this concept, we narrow our focus on digital entertainment games. Two of the important areas in PAEC are to create specific targeted strategies to cater for individual game player, and also to perform personalization. Personalization in games can be in terms of difficulty levels, game resources, emotion, characters, etc. In game design, the first area can be easily identified and addressed in considerably easiness. Normally, during this first stage, the genre of the entertainment game will be determined. However, the second area of personalization is sometime difficult. This is because of the differences between players in terms of their personality, background, culture, skill, and learning ability.

PAEC Concept

PAEC basically focus mainly on providing improved perceived value to the users, in this article, the game players. There are three broad areas of focus in the PAEC: (a) the player, (b) the content, and (c) the experience. The interaction between the content and player is driven primarily by the value the player perceives from the experience. PAEC can be modeled based on the user perceived value as shown below:

$$\text{perceived value}_i = \frac{\text{experience} + \text{entertainment quality}}{\text{price}} \quad (1)$$

From the above model, we can see that the perceived value for player *i* has several components. The first component, experience, refers to the idea that players buy experience and not the particular entertainment products. To the extent, the content of a product enhances the experience and it then increases the player's perceived value. The entertainment quality also increases player's perceived value. Price is also a component of the perceived value. Different components of the player's perceived value provide opportunities for enhancement and management of the interactive content with individual players. From Equation (1), we can see that the perceived value is defined at the individual level (hence the subscript *i*). Therefore, it is important to identify the components of perceived value that are unique to each player or player base.

The most important part of the PAEC is related to the perceived value from the game player. As discussed, the experience and entertainment quality are two very important components in the perceived value. Normally, these two factors can be directly related to the factors such as fun, challenge, entertaining, interest level, and so on. As individuals are different, personalization becomes the important factor to improve this perceived value. Personalization for PAEC is defined as any set of actions that can tailor the entertainment media experience to a particular user or player.

Personalization

Personalization is normally referred to as any objects, activities, or interactive experiences that are customized and tailored to individual liking. The range of objects, activities, or interactive experiences includes phone covers, ring tones, images, fonts, media content, avatars, game play experience, and so on (Oulasvirta & Blom, 2007). Blom and Monk (2003) define personalization "as the process that changes the functionality, interface, information content, or distinctiveness of a system to increase its personal relevance to the individual."

The term *personalization* is also widely used in both conventional and electronic commerce. Web personalization, which is mostly used for online businesses, is defined as any set of actions that can tailor the Web experience to a particular user or customer

(Eirinaki & Vazirgiannis, 2003). Web personalization can improve the eCommerce experience by helping companies to maximize their customers' perceived value, which may translate to the increase of profit. Web personalization is used to provide personalized information, tailoring users' Web experience, updating users' interests, and so forth.

In Oulasvirta and Blom (2007), personalization can have a direct relationship with the psychological aspect of human needs. It is thus suggesting that when placed in the PAEC concept, the perceived value, which depends on the important component of experience and entertainment quality, could also largely depend on the background, culture, belief, and gender of the game player. It is thus important to explore ways to provide player adaptation in entertainment computing to improve the perceived value. This is thus one of the largest challenges facing the entertainment computing industry. In the following sections, we will explore what have been done to realize a bigger picture introduced in this PAEC concept.

Player Modeling and Player Centered Design

Player centered game design could be defined as the approach that aims to improve the game design and development from the individual players' perspective. A benefit of taking a player-centered design approach is that it should ideally result in enhanced game play experiences for players regardless of gender, age, or experience (Charles et al., 2005). Traditionally, to produce a game that is player centered is to perform market research. A large part of this work is data mining to realize the target game player base involving data collection and analysis. The outcome of the data mining is then used for game development (Kennerly, 2003). The analysis should help to determine what players want from a game. This information can drive the requirements for the game design based on what players want. Alternatively, it can also be a part of the preproduction process, where prototypes of game play can be developed. The players can then answer surveys to feedback into the game design process.

A common approach to player-centered game design is to conduct usability/play testing in prerelease versions of the game, which are often known as alpha or beta tests. These tests can involve a small selection of game testers. Larger multiplayer games in development often have large multiplayer beta test stage, where network problems are explored. There is another approach to this, known as postrelease, which is not very commonly used today. Normally, game player will translate the postrelease to poor game development, which could be damaging to the image of the company or the publisher. Some games provide the player the power to control their own game play experience to achieve the player-centered game design. Some of the examples include difficulty levels, customizing play scenarios (sandbox games, skirmish matches), or providing a range of different online arenas that players can select from with different game types/styles of play. Some also include SDK as well as map editors for player to create their own content.

On the other hand, some researchers proposed that player modeling allows game play experience to be enhanced and customized for individuals. Player modeling

involves creating a model of different types of players and grouping them based on how their experience from the game. This model is then used to adjust game play experiences to meet the players' preference. The ability to group or label players into appropriate/correct classes is always a difficult problem. Often the player types are too specific to a genre, culture, or platform or inversely they are too broad and do not provide enough variation to meet the needs of different players. Normally, to force a game player into a group, simple assumptions on demographics, gender, and player style are often made.

However, to satisfy the PAEC concept, player modeling and player-centered game design can only satisfy part of it. Although these two concepts can be used to provide generalized player base, it is still unable to satisfy the perceived value (Equation 1) at the individual level.

Adaptive Game Systems

Kaukoranta, Smed, and Hakonen (2003) define adaptive as having the ability to react and make appropriate reaction to the change in the interacting environment. Charles et al. (2005) define adaptive game systems as a method for creating dynamic heuristic models of player types that adjust themselves over time based on input and measurement from the individual players. Such an adaptive system can perform dynamic difficulty level adjustment, adapting the game play and style, identifying preferred game resources, and picking up patterns of exploitation. Charles et al. (2005) have proposed a framework for designing such adaptive game system (see Figure 11). Most of the research on adaptive game systems circles around the change of the difficulty level of the game plays.

One of the examples can be found in adventure game to dynamically adjust game play to the player's preference. If the game player has preference of using stealthy tactics, then the game can present different challenges that use and test their stealth skills.

When used in picking up patterns of exploitation in games, this could include identification of player tactics/strategies that are used more often than others to win. Often the player will continue to use the same strategy over and over to ensure a win; even if it negatively impacts the game play experience (becomes too easy).

With the advancement of game AI, adaptive game systems could be one of the solutions for producing a player-centered game. There are a number of research attempts to use game AI to manage the adaptive ability of the games (Ram, Ontañón, & Mehta, 2007; Spronck & Herik, 2004).

This concept of adaptive game systems can support most part of the PAEC concept; however, more needs to be done to fully realize the complete concept of PAEC. Game AI can definitely play a big part in this if it is used carefully. The major challenge is how to perform personalization in games in terms of difficulty levels, game resources, emotion, characters, user interaction, and so on, all at the same time.

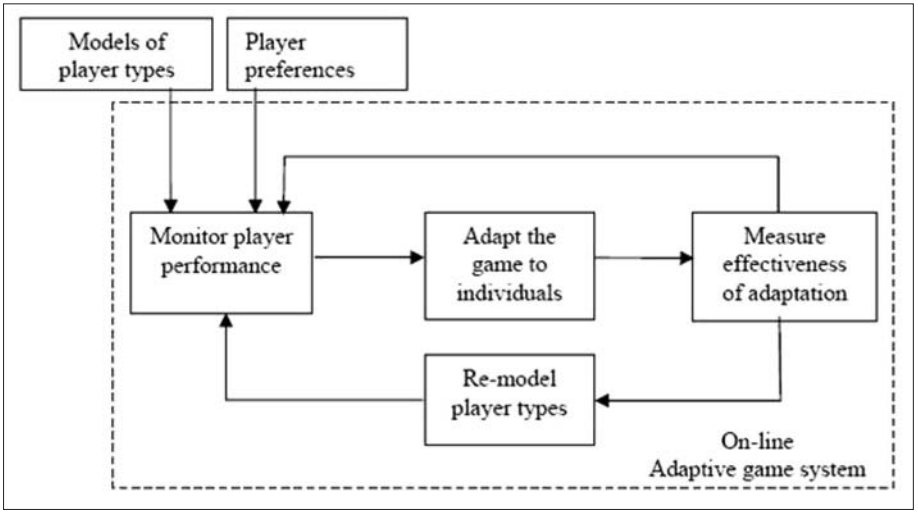

Figure 11. A proposed framework for designing adaptive game systems from Charles et al. (2005)

Cases

In games development, level editing is a challenging task. A level normally refers to a separate area in the game's virtual world, sometimes also known as a stage, course, or map. In most of the modern games, it is typically representing a specific location such as a building or a city. Level design is considered more of an art than a science. The level designer needs artistic skills and know-how, as well as extensive technical knowledge, to perform the tasks.

Besides focusing on the artistic side of the level design as mentioned above, another area that is known in early days for level design is the "level of difficulty." This is still an important area in level design for games, as this has direct effect on the game play experiences that engage and motivate the game players. If the games are too easy, the players may feel bored. If it is too difficult, the players may feel frustrated. Of course, most games allow players to adjust the difficulty level, such as setting to "Beginner," "Average," and "Advanced;" however, the experience is static and in the worst case, the "Beginner" or "Advanced" settings are subjective to an individual and are predictable. Will you play a game if you are "killed" in less than 30 seconds of the game play all the time?

Folmer (2007) captured the design patterns of adaptive difficulty level in games. A game basically should allow game player to select the difficulty level if the player thinks that it is too hard. The game should also cater to people with some disabilities using the accessibility selection. The game also used two methods to adjust the

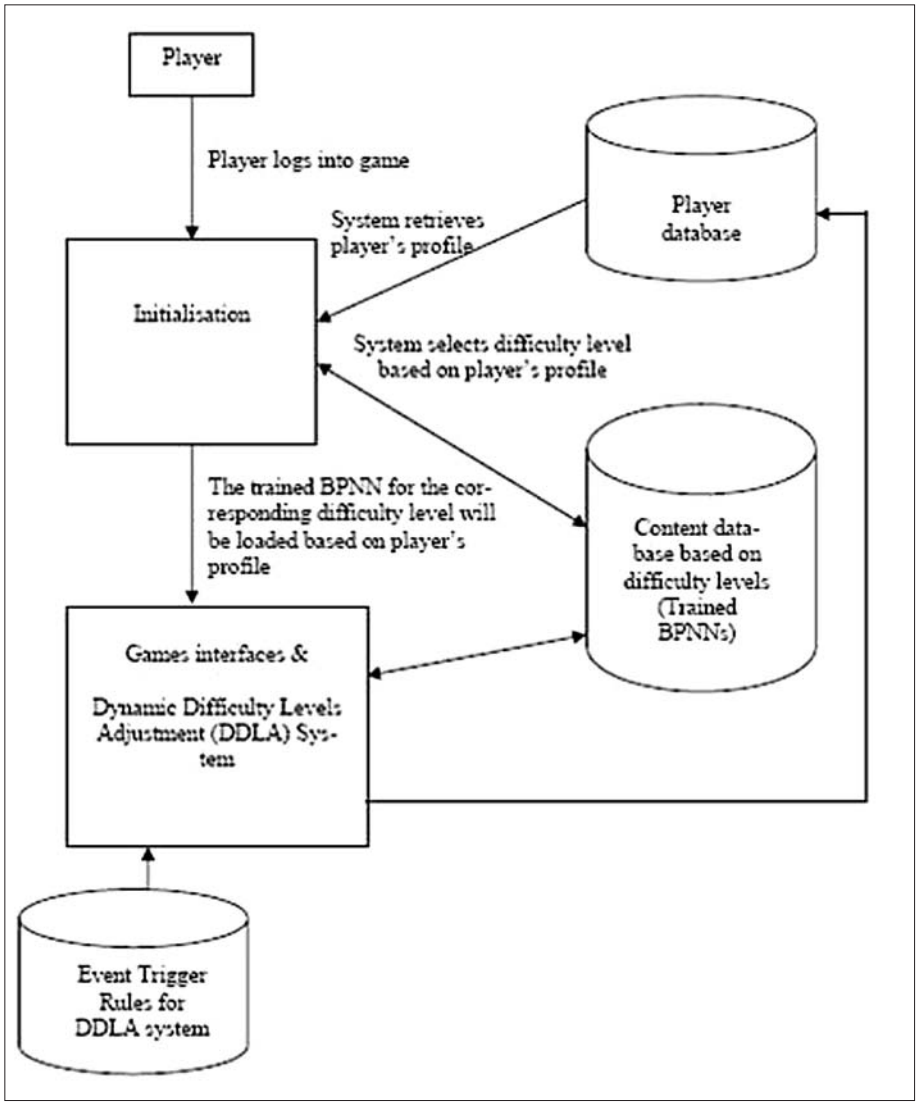

Figure 12. Adaptive difficulty level adjustment

difficulty level dynamically. The first solution could be to suggest a more appropriate difficulty level by observing the game play performance. Another solution is to automatically adjust the difficulty level by monitoring the game play performance online. In GOD OF WAR, they used the solution of suggesting a more appropriate difficulty level. In RESIDENT EVIL 4, the difficulty level is adjusted automatically based on

Figure 13. Virtualized personal digital assistant (PDA) for managing personalized game play experience

the player performance. SIN EPISODES offers more advanced dynamic difficulty level adjustment by monitoring the performance online. It then adjusts the difficulty level in terms of enemies' health ammo, armor, and the damage to a specific playing style.

In Figure 12, we have presented a game AI framework that uses back-propagation neural networks and fuzzy rules.

One of the important areas, and one that is normally neglected, is designing games for people with special needs. We have designed a game prototype based on a role-playing computer game that uses both English and Auslan (sign language). The medium of communication is Auslan in the games. This game is based on the concept that a game can be designed for use as a leisure game and as a learning tool. It can be enjoyed purely for its entertainment factor without any obvious learning objective, or as a learning tool that has an entertainment factor to make the educational aspects more appealing. Of course, the game is to teach sign language as an underlying objective. A virtual personal digital assistant (PDA) as shown in Figure 13 will be used as a personalized tool to manage the learning and experience.

Another prototype that we have built is game play targeting seniors. The game concept is to allow seniors continue to enjoy gardening through virtual entertainment technology. Figure 14 illustrates the screen shots of this prototype. In this game, it will be very important that the personalized ability is incorporated. Some seniors may not be too comfortable with the keyboards, too slow to react to the on-screen interaction, and so on.

Another way to explore personalization is using intelligent conversation avatar. The intelligent conversation avatar will create their knowledge from the Web. It can be customized by allowing users to change the look and feel of the avatars. The content can then customize to lead the user during the interactive experience for different needs. Figure 15 shows the screen capture of such intelligent conversation avatars.

Figure 14. The gardening game for seniors

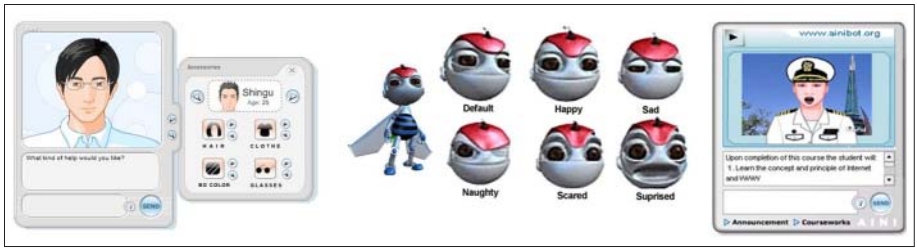

Figure 15. Examples of intelligent conversation avatars

Summary

We have proposed the concept of PAEC. From Equation (1), it can be observed that a large part of the factors that contributes to the perceive value of the entertainment media is individual. The article then discussed the different effort that can help to realize this concept. Game AI can also help realize this concept; however, more work needs to be done to fully capture the importance of the concept.

Future Game AI

Game AI has witnessed substantial growth in the last decade. Currently, game AI is used for modeling various aspects of nonplayer characters. Although such applications (e.g., behavior modeling) have been very successful, the deployment of game AI for serious games having learning as the main component poses basic questions, such as the representation of an explanation capability as perceived by the human player. In this article, we propose a general framework for the development of such an explanation capability. Our framework is based on the exploitation of the known results linking (formal) language models and (finite state) machine models in theoretical

computer science. It is a well-established fact that the Turing model, as a machine model (and its equivalent models such as Chomsky's phrase structured grammar model or recursively enumerable functions) is a *most general* model for computation. Many biologically inspired computing paradigms such as traditional neural networks, genetic algorithms, evolutionary computing frameworks, and so on, aim at addressing the problem of *automatic generation* of algorithmic solutions for many computationally difficult problems. Various soft-computing models have complementary capabilities. This motivates us to propose a framework for their integration. We identify the need for future game AI engines with such capabilities.

Game AI: Current State

In the past decade, the field of computer games has evidenced tremendous growth. The computer games have grown from text-based games to immersive worlds modeled using 3D graphics. Major components of current computer games include: user input (typically by some kind of game controller), graphics rendering, audio play, and game AI.

Traditionally, game AI has been used for various parts of the game that include unit movement, simulated perception, situation analysis, spatial reasoning, learning, group coordination, resource allocation, steering, flocking, target selection, and many more behavior modeling aspects (Tozour, 2002). Even context-dependent animation and audio use game AI. These game AI models used well-developed AI technologies such as rule-based systems, scripts, frames, Bayesian learning, Markov models, and neural networks. For many games, the game hardware includes processors with limited computational capability, and hence the efficient deployment of these techniques for game moves has remained the primary concern in the last decade.

Although soft-computing techniques have been deployed successfully for many embedded systems controllers that include processors with limited computing capability, their applications for computer games has been limited. How are computer games different from embedded systems using intelligent controllers? Well, various main systems of a computer game compete for the use of the same hardware resources. The major subsystems of a typical computer game, namely, graphics rendering and audio system are computer processor intensive. Because visual and audio interactivity has traditionally been very important for the success of computer games, especially in the past decade, game development decisions made by game developers typically allot more resources for graphic and audio systems leaving the game AI with the little processing time.

We also note that the existing game development engines provide extensive support for graphics (visual) and audio play. Recently, the applications of soft-computing techniques in the area of game AI are increasingly being reported, for example, Stanley, Bryant, and Miikkulaimen's (2005) real time neuroevolution and Louis and Miles's (2005) case injected genetic algorithms for learning to play computer games; however, these efforts are mainly at the research level in the field of game AI and have

not yet found significant place in the mainstream of game development process. One of the reasons for this state of affairs is the nonavailability of good soft-computing-based game AI engine. What factors should be considered for developing such software? In this article, we give the reader a bird's eye-view of the full perspective of capabilities and limitations of such an approach.

Soft-Computing Frameworks: Computational Capabilities

The field of soft computing, historically introduced by Zadeh (1994) as a problem solving paradigm, has now conventionally become a collection of more traditional techniques such as neural networks, evolutionary computing, fuzzy logic, and many novel techniques inspired from behavioral studies such as ant colony optimization, small world theory, theory of memes, and so on (Ovaska & Sick, 2006). These technologies have steadily changed the way we solve real world problems in science and engineering.

Problem solving using soft-computing techniques is different than problem solving using traditional computer algorithms. In the study of theory of algorithms, the notion of computationally difficult problems is formalized in terms of NP-complete problems. Soft-computing techniques have the ability to generate solutions for many computationally difficult problems. However, in the midst of deployment of soft-computing techniques to solve many such problems, it has also given rise to many fundamental questions that are of interest to the discipline of computer science. In this section, we focus on a few of such important questions.

Although many soft-computing techniques have attempted to give solutions to a specific problem, it is not clear how this approach generalizes for solving all computational problems.

We list four such questions below:

1. Is the given soft-computing technique *general enough*, in the sense that, *is it possible to express* any arbitrary computation in that technique?
2. Does the given soft-computing technique possess the ability of *automatically generating* a solution to any arbitrary problem for which algorithm is known to exist?
3. Does the given soft-computing technique possess the ability of generating automatically the *most efficient solution* to an arbitrary computable problem?
4. When the given soft-computing technique does not generate the most efficient solution, does it generate a *reasonably efficient solution*, with the *performance bound* on how far the resulting solution would be from the most efficient solution?

In general, these questions are very difficult to answer, and partial attempts for answering these questions have been done by a few researchers (e.g., Turchin's meta-computations (Turchin, 1993, 1996a) and super-compilations (Turchin, 1996b), and

investigations for cellular automata computations (Mitchell, Crutchfield, & Hraber, 1994). However, the traditional neural networks and theoretical computer science have, at least in their beginning stages, very strong links and overlaps. In fact, neural networks are the forerunners of the very first exhaustively studied model in theoretical computer science, namely finite automata. McCulloch and Pitts gave a model of finite automaton which is a network of idealized neuron-like elements. Now, neural networks have been used to represent many models like finite automaton, pushdown automaton, and Turing machines, and even to learn them from examples (Omlin & Giles, 1996; Siegelmann, 1999; Siegelmann & Sontag, 1991). Even more restrictive neural networks models, such as binary neural networks, have also been used to generate such models (Chaudhari & Dagdee, 2004; Chaudhari & Tiwari, 2004).

Another interesting field in soft computing is evolutionary computation. Evolutionary computing is based on the concepts of biological evolutionary theory that mimics the mechanics of reproduction, mutation, recombination, natural selection, survival of the fittest, and so forth. There are three basic kinds of evolutionary computations: (a) genetic algorithms (GAs), (b) genetic programming (GP), and (c) evolutionary algorithms (EAs).

In theoretical computer science, the Turing machine (TM) is (one of) the most general model for representing computations; in other words, any problem that is solvable by an algorithm has a TM representation; hence it is important to investigate whether soft-computing technologies are general enough to (a) simulate and (b) automatically construct (i.e., automatically learn) TMs from some high-level specification of the problem statement. The first question regarding the simulation has been settled by many researchers with a positive answer (Siegelmann, 1999; Siegelmann & Sontag, 1991). However, the automatic construction of arbitrary or even sufficiently larger interesting subclass of TMs using soft-computing frameworks remains a challenge. For automatic generation of the computational models, the fields of traditional neural network and other soft-computing techniques such as evolutionary computation need to be integrated.

This section is organized as follows. First, we give a brief introduction of the TM, neural network, GAs, and evolutionary computation as computational models. For explanations based on automatic generation of algorithmic solutions, many soft-computing models can be used. Next, we give some observations about specific models such as cellular automata, GAs, small world theory, and theory of memes. Then, we give one classification of computational capabilities based on the models in theoretical computer science and give a brief survey of research work about the integration of these models with other soft-computing models. Finally, we include the comments about the applications of soft-computing models to the field of game AI.

Basic Concepts

Turing machine. To represent any arbitrary computation, the formalism of TM is (one of the models) considered to be the most general model. Using the formalism

reported in Wood (1987), a deterministic TM can be specified by a sextuple (Q, Π, Γ, δ, s, f), where

Q is a finite set of states
Π is an alphabet of input symbols
Γ is an alphabet of tape symbols
δ is the transition symbol
s in Q is the start state
f in Q is the final state

The transition function has the following structure:

$$\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$$

where L denotes left move and R denotes right move of the tape head. A machine with a transition function δ of the above type, having two inputs (state, current memory tape symbol), and three outputs (next state, tape symbol to be written after the move, and whether the move is left move, or right move) is a five-tuple TM.

Neural networks. A neural network is an artificial system that aims to perform intelligent tasks similar to those performed by human brain (Pitts & McCulloch, 1947). A neural network stores its knowledge through learning within interneuron connection strengths known as synaptic weights. These networks have shown themselves to be adept at solving pattern classification, prediction, and etc. The most common neural network model is multilayer perception. A given feed-forward neural network can be trained using a learning algorithm by which it adapts to the network's connection weights so as to minimize the error over the pattern training sets. This success is hampered by the difficulties of initially defining the network's structure and training parameters and the other problems caused in the weight space. There are other factors which are to be decided, which include network architecture (number of hidden layer, number of hidden nodes, initial weights, etc.).

The back-propagation algorithm is a well-known method for training a multilayer feed-forward network but it follows gradient descent method that has a tendency to get stuck in a local minimum and thus the global minimum is not reached. The performance of the algorithm depends on the selection of the initial weights and on the parameter used.

The multilayer perception and other neural network models can be trained using a learning algorithm such as (error) back-propagation, steepest descent, least square error, GA, evolutionary computation, and so on. Using one of these algorithms, the weights are determined and the network is said to be trained for a set of data.

Genetic algorithms. GAs are based on the Darwinian-type survival of the fittest strategy with sexual reproduction, and Mendel's theory of genetics as the basis of biological inheritance (Goldberg, 1989). In these theories, stronger individuals in the population have a higher chance of creating offspring. Each individual in the population represents

a potential solution to the problem to be solved. GAs do not work with a single point on the problem space but use a set or population of points to conduct a search. This gives GAs the power to search multimodal spaces littered with local optimum points.

GAs can be used to train a multilayer perceptron in which weights form a parameter space. Although GAs have the advantage of not getting stuck in local optima, they have other problems. When the search space is very large then GA methods generally take a long time to converge to good quality solutions. The length of the search is due to the optimal generalization of the training process with no a priori knowledge about the parameter space.

Evolutionary computation. In the last decade, evolutionary computation has become a standard term to denote a very broad group of algorithms and techniques that are based on the principles of natural processes involving biological evolution. Evolutionary algorithms are mainly meta-heuristic and optimization methods that share some generic concepts borrowed from the natural process of biological evolution. Research in this area has mainly been focused on solving the problems that can be formulated as an exhaustive search over the space of all possible solutions. Using evolutionary computing frameworks, many approaches have been proposed in the last decade. Some approaches for global optimization algorithms include the approaches based on evolution of species (Davis, De Jong, Vose, & Whitley, 1999), immune system (Castro & Timmis, 2002), social behavior of ants (Bonabeau, Dorigou, & Theraulaz, 2000), and so on. Many variants of evolutionary computations are also studied by various researchers. For example, Boettcher and Percus (2001) proposed a new optimization algorithm that is based on the principles of natural selection, but it does not follow the basic GA framework for population reproduction. Their approach is one step toward integrating different models such as principles of self-organized criticality of Bak and Sneppen (1993) in broad evolutionary computation framework.

Some Soft-Computing Models for Automatic Generation of Explanations and Algorithmic Solutions

Cellular automata. Cellular automata are investigated for understanding the mechanisms and architectures of spatially extended and locally connected systems that perform *global computation*.

The biological phenomenon of reproduction motivated John von Neumann (1967) to conceptualize the model called *self-reproducing automaton*. He proposed this model to integrate the concept of universal computation, as represented by a TM, and the biological phenomenon of reproduction. His model was a 2D infinite grid of cells. On these cells any automaton, whose information is given on "input tape," could be constructed. Construction involves three concepts: construction arm, construction control, and construction site.

Construction proceeds via construction arm, which is a propagating sequence of states moving through the construction site where the automaton is to be located.

Cellular automata model, mainly attributed as conceptualized by John von Neumann in the 1960s has many variations of the basic idea depending on the

phenomena they wish to emulate. One standard approach is due to Stephen Wolfram (2002). Cellular automata model is capable of *universal construction* in the sense that it can construct any automaton that has been specified on the "input tape," and it can replicate the behavior of a TM.

Global computations, cellular automata, and genetic algorithms. The concept of global computations can be illustrated by the following example. Consider the problem of "firing squad synchronization problem" (Mezoyer, 1987). Consider next a 1D array of cells as a squad of soldiers in a line, the leftmost of which is the "commander." Suppose that the commander gives an order to fire (at some time, say T_{start}). The problem is to construct an automaton that would, maybe after some time T ($T > T_{\text{start}}$), result in making all soldiers fire simultaneously; that is, all cells in cellular automaton representing the soldiers are in some prespecified state at time T. With the central command (much like a central processor with access of control line to every [memory] cell), this is a trivial task. However, models such as the cellular automation assume that only local communication (i.e., communication from each cell to its neighbors) is allowed. One approach to solve this problem is to first send commander's signal to all cells, and later allow interactions with each other and the array boundaries in some predetermined fashion. This high-level solution strategy involving

1. appropriate signal(s)
2. appropriate rule(s) for propagation of signal(s)
3. rule(s) of interaction after propagation phase is over

can be converted to the basic cellular automation operations on cell states. Although it is easy to come up with high-level solution strategy for this example, this approach does not generalize to many other problems. For specific and well-defined problems, such as optimization problems, solution space can be determined in exhaustively many (maybe exponentially large but finitely many) alternatives. For such problems, it is useful to automate the process of finding optimal solutions for global optimization problems involves using some kind of device that operates on rule tables, and then perform the inverse problem of inferring the high level rules from low-level solution(s). This approach has been proposed and investigated by Turchin (1996b) and Mitchell, Crutchfield, and Das (1997). Similar approaches have also been developed for GAs (Mitchell, 1996). An application of cellular automata–based approach for design of interesting games is reported in Henru, Chaudhari, and Prakash (2005).

Small world theory. Watts, Dodds, and Newman (2002) studied the dynamics of connected graphs and existence of short paths, the concepts that pioneered "small world theory." This theory has surprisingly many applications. As Watts points out in his popular book (Watts, 2003), the brain is a network of neurons; organizations are people networks; the global economy is a network of national economies, which are networks of markets, which are in turn networks of interacting producers and consumers. In fact, many more situations can be considered as the strategies for problem solving; such networks are observed in food webs, ecosystems, Internet, and even in the topics in a conversation, words in a language, and so forth.

The contributions to small world theory establish that, after certain conditions are satisfied, the local actions in such networks have global consequences, and the relationship between local and global dynamics depends critically on the network's structure. These models have been applied for many situations, for instance, the spread of infectious disease through a structured population, the evolution of cooperation in game theory, the computational capacity of cellular automata, and the synchronization of coupled phase-oscillators (Strogaz, 2003).

Klienberg (2000) investigates not just the existence of short paths but also deals with the problem of finding them (Kleinberg, 2000). Strogatz (2003) illustrates the use of these and related concepts for creation of order from such phenomena. Extension of Strogatz and Klienberg's ideas for automatic generation of algorithms is thus an exciting area.

Theory of memes. Historically, in 1904, the German biologist Richard Semon became famous because of his work *Die Mnemische Empfindungen in ihren Beziehungen zu den Originalenempfindungen*, which was translated in English in 1921 as *The Mneme.*

In the 1970s, Richard Dawkins (1976) coined the slightly different term *meme* to describe a unit of human cultural evolution similar to the gene, arguing that replication also happens in culture. We note that myths, inventions, language, and political systems as structures are made of memes, which are units of ideas, habits, skills, stories, customs, and beliefs that are passed from one person to another by imitation or teaching.

The theory of memes is one of the controversial ways of thinking about cultural evolution. In his book, Dawkins (1976) argues that the meme is a unit of information residing in the brain and is the mutating replicator in human cultural evolution. This created great debate among sociologists, biologists, and scientists of other disciplines, because Dawkins himself did not provide a sufficient explanation of how the replication of units of information in the brain controls human behavior and ultimately culture. Since the 1970s, the variants of this theory, for instance, Richard Brodies's *Virus of the Mind: The New Science of Meme* (1995), and the *Meme Machine* by Susan Blackmore's *Meme Machine* (1999) have been popular among researchers.

Meme has two important properties. Meme propagates in the neighborhood, and it influences its surroundings. We note that these properties of memes are in some sense similar to cellular automata; hence it is interesting to find out how results in cellular automata can be translated and expressed in the context of theory of memes.

Meme theory has given rise to new computational models, where the above phenomena are studied in an evolutionary framework by many researchers. This form of search algorithm may be regarded as a marriage between population-based global search and the local improvement made by each of the individuals. It has the potential of exploiting the complementary advantages of evolutionary algorithms (generality, robustness, global search efficiency), and problem-specific local search (exploiting application-specific problem structure, rapid convergence toward local minima). Such combinations of optimizers are commonly known as hybrid methods. In diverse contexts, hybrid evolutionary algorithms are also commonly known as memetic algorithms (MAs), Baldwinian evolutionary algorithms, Lamarckian evolutionary algorithms,

cultural algorithms, or genetic local search. Such methods are demonstrated to converge to high-quality solutions more efficiently than their conventional counterparts.

Approaches for Automatic Generation of Computational Models

Classification of computational capabilities. Based on the standard models in theoretical computer science, the ability of computational structure can be classified as follows:

1. *Table structure* (TS) given an input, the corresponding output is decided by the table (or a mapping, or a function, or simple translation), and it is returned by doing table search. However, it cannot search and return the patterns specified by regular expression.
2. *Deterministic finite automaton* (DFA) is a machine with finite states, and it can return the patterns matching the specified regular expression. However, it does not have "counting capability"; as a consequence, it cannot recognize the patterns such as sequence of matched parentheses, or patterns of palindromes.
3. *Pushdown automaton* (PDA) is a machine with one (potentially) infinite stack. It can recognize patterns of matched parentheses and palindromes, and it is useful for modeling of most of the programming language specification constructs. However, it cannot recognize patterns such as the following:
   - $\{ww : w$ is (finite) string of alphabet symbols$\}$, that is, "two copies of the same string, one followed by the other"
   - languages like $\{a^n b^n c^n : n \geq 1\}$
   - other more complex patterns for recognizing which we can write an algorithm.
4. *Turing machine* (TM) is the "most general" machine, in the sense that, for any algorithm, we can mechanically construct an equivalent TM; so it corresponds to the most general model of computation.

To retain the generality of "automatic generation" of algorithms, "learning" of function structure (or mapping, or table entries) is computationally least challenging. Automatic construction or *learning* of an algorithm in the above sense of generality) is most challenging, and the remaining models pose increasing order of challenge. As it turns out, from the theoretical point of view, there are tasks that TMs cannot handle; however, the famous Church–Turing hypothesis allows us to assume that, for any algorithm (computational procedure that terminates), there exists an equivalent TM.

It is therefore important for us to investigate the techniques for automatic construction of TMs. Siegelmann and Sontag (1991) and Siegelmann (1999), have proved that various neural network models with continuous nonlinearity in the neuron model have capability of simulating TMs. For restrictive neural network models such as binary neural networks, Chaudhari and Dagdee (2004) have proved that they have the capability of representing TMs.

In the remaining part of this section, we briefly give some approaches to achieve this goal using a few of the soft-computing frameworks.

Integrating various soft computing models. Because different soft computing models have complementary capabilities, their integration has been investigated very extensively in past few decades, and neuro-fuzzy approach has found a lot of applications. For example, in Chaudhari and Liew (2004), the application of neuro-fuzzy modeling for arrhythmia classification is reported. Many researchers have investigated the use of evolutionary computation for learning of the traditional neural networks. The weights of the neural network can be generated and optimized using evolutionary computation. One such approach is reported by Rajasekaran and Pai (1996). In such a typical approach, GA is used to generate and optimize the weights and the network structure in which the weights of the network are coded as strings. Each chromosome is evaluated for its fitness and new chromosomes are generated using the genetic operators like crossover and mutation. Thus, the chromosomes with good fitness are said to have survived and are present in the mating pool where as the bad chromosomes are discarded. Thus after generating several populations, the optimized weights for the given neural network is said to have achieved. Once the weights are generated, the neural network structure is to be optimized. Frank, Leung, Lam, Ling, and Tam (2003) investigated another approach for such integration.

Majoros (2002) reports interesting experiments to test the hypothesis that the bird house finch generates songs using a simple form of grammar model having representation that Majoros chose to express by a hidden Markov model. Majoros used the memetic evolution as applied directly to song elements, simulated by the GA, and found that such a model is very similar to those inferred from the house finch song. We note that, such approaches report the links between memetic models and simple computational models such as deterministic finite automatons. However, as we know, many features in natural languages are modeled using context free (and even context sensitive) grammars. Thus, the investigations of approaches based on memes theory to learn more powerful computational models such as pushdown automatons, and even TMs, gives rise to interesting and difficult research issues.

We observe that most of the existing attempts have not yet touched on the integration attempts of soft-computing models such as memes theory and small world models. Because such models have the capability of representing specific higher level information, it is useful to investigate integration of these models with other models as well.

Summary

Many soft-computing models such as neural networks, GAs, and so on, can simulate TMs. However, integration of many soft-computing methodologies is needed for the automatic construction of efficient solutions. To achieve such integration, in addition to standard evolutionary computation framework, we reviewed some novel soft-computing approaches such as cellular automata, GAs, small world theory, memetic evolution, and cultural evolution. We pointed out that researchers have already investigated the

integration of some of the models, for example, neural networks and fuzzy techniques; however, there are good opportunities for many more integrations. Such computational paradigms have an important role in future game AI engines and models.

Conclusions

The investigation of game development process models for games of various complexities has been, and still is, a very active research area, which is presented in the context of training in a computer engineering curriculum. In this article, we analyzed the game engines that are deployed for content generation. The challenges for online game development and deployment are addressed in Morgan (2009). Two case studies show the flexibility and the powerful features available for game developers. Continuous LOD triangle meshes continue to attract the attention for terrain modeling compared with fully irregular approaches or fully regular meshes. Geomancy terrain system shows easy implementation and real time performance that can be easily integrated into game engines.

A generic behavior modeling framework has been proposed based on some key observations on how real humans behave in everyday situations. We emphasized the role of humans' experience in determining their behaviors in various situations. In addition, various social and psychological factors need to be considered. These factors may determine the relative strength of a person's social aspect and animal aspect. That is, they help to determine whether a human being will act according to his or her social role in the society, his or her social group, or will act like an animal. As a final remark, we emphasize the social context when discussing human behavior models. It is impossible and also not necessary to build a model that covers all aspects in human behavior. To understand how a human behaves in certain situation, we must understand the situation and in the mean time, we must also understand the person regarding aspects such as his or her physical capability, experience, personality, and social relations that are deemed salient to the situation.

Another important area that we have presented is player adaptation, where a large part of the factors that contribute to the perceive value of the entertainment media is individual. We have then discussed the different efforts that can help realize this concept. Game AI can also help to realize this concept; however, more work needs to be done to fully capture the importance of the concept. Integration of many soft-computing methodologies is needed for the automatic construction of efficient solutions, which we believe will play an important role in future game AI engines and models.

Acknowledgments

We thank all our peer reviewers who have volunteered their time and experience. We would like to express our great appreciation to them for their valuable contribution, which helped to improve the quality of this article. The authors also thank the students who helped in the development of the prototype games described in this article at Manchester Metropolitan University, UK; Nanyang Technological University, Singapore; and Murdoch University, Australia.

Declaration of Conflicting Interests

The authors declared no conflicts of interest with respect to the authorship and/or publication of this article.

Funding

The authors received funding from their respective institutions to support this research.

References

Alphaworks. (2008). *CodeRuler: A Java-based, real-time competition game based on the Eclipse platform*. Retrieved October 9, 2009, from http://www.alphaworks.ibm.com/tech/coderuler/download

Augoustinos, M., & Walker, I. (1995). *Social cognition: An integrated introduction*. Thousand Oaks, CA: Sage.

Bak, P., & Sneppen, K. (1993). Punctuated equilibrium and criticality in a simple model of evolution. *Physical Review Letters, 7*, 4083-4086.

Blackmore, S. (1999). *The Meme machine*. Oxford, UK: Oxford University Press.

Blom, J., & Monk, A. (2003) Theory of personalization of appearance: Why people personalize their mobile phones and PCs. *Human Computer Interaction, 18*, 193-228.

Boettcher, S., & Persus, A. G. (2001). Optimization with extremal dynamics. *Physical Review Letters, 86*, 5211-5214.

Bonabeau, E., Dorigou, M., & Theraulaz, G. (2000). Inspiration for optimization from social insect behavior. *Nature, 406*, 39-42.

Brodie, R. (1995). *Virus of the mind: The new science of Meme*. Integral Press ISBN 0-9636001-25, Seattle, USA.

Carmack, J. (2008). John Carmack's blog at http://doom-ed.com/blog/category/doom-ed/john-carmack

Castro, L. N., & Timmis, J. (2002). An artificial immune network for multimodal function optimization, In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC'02* (Vol. 1, pp. 669-674). Hawaii.

Charles, D., McNeill, M., McAlister, M., Black, M., Moore, A., Stringer, K., et al. (2005, June). Player-centred game design: Player modeling and adaptive digital games. In *Proceedings of the DiGRA 2005 Conference* (pp. 285-298). Vancouver, British Columbia, Canada.

Chaudhari, N. S., & Dagdee, N. (2004). Turing Machine simulation using hard-limiter neurons. In *Proceedings of the International Joint Conference on Neural Networks* (Vol. 4, pp. 3269-3274). Budapest, Hungary.

Chaudhari, N. S., & Liew, C. S. (2004). The fuzzy c-means and neural network framework for arrhythmia classification. In F. K. Fuss & S. L. Chia (Eds.), *Proceedings of the First International BioEngineering Conference* (pp. 228-231). Singapore.

Chaudhari, N. S., & Tiwari, A. (2004). Extension of binary neural network for multi-class output and finite automata. In J. C. Rajapakse & L. Wang (Eds.), *Neural information processing research and development* (pp. 211-237). Berlin: Springer.

Codemasters, 1998, http://www.codemasters.com/

CodeRuler. (2008). *CodeRuler brief*. http://www.ibm.com/developerworks/java/library/j-coderuler/?Open&ca=daw-ja-dr http://www.alphaworks.ibm.com/tech/coderuler

Creators Club. (2007). XNA Creators Club. http://creators.xna.com

CrowdDynamics. (2007). *Crowd disasters*. Retrieved October 9, 2009 from http://www.crowddynamics.com/Main/Crowddisasters.html

Davis, L. D., De Jong, K., Vose, M. D., & Whitley, L. D. (Eds.). (1999). *The IMA volumes in mathematics and applications: Vol. 111. Evolutionary algorithms*. Berlin: Springer.

Davison, C. and Tang, W. Deformable Terrain Generation for Real-time Strategy Game, Proc. of Eurographics Short Presentations, Manchester, UK, 2001, pp. 1-8., 2001.

Dawkins, R. (1976). *The selfish gene*. Oxford, UK: Oxford University Press.

DooM. (2008). *Wiki for DooM*. Retrieved October 9, 2009, from http://doom.wikia.com/wiki/Entryway

Doom 3. (2008). *Doom3 game engine*. Retrieved October 9, 2009, from http://en.wikipedia.org/wiki/Doom_3

Duchaineau, M. A., Wolinsky, M., Sigeti, D. E., Miller, M. C., Aldrich, C., & Mineev-Weinstein, M. B. (1997, October). ROAMing terrain: Real-time optimally adapting meshes. In *Proceedings of IEEE Visualization* (pp. 81-88). Phoenix, AZ.

Eirinaki, M., & Vazirgiannis, M. (2003). Web mining for web personalization. *ACM Transactions on Internet Technology, 3*, 1-27.

Festinger, L. (1954). A theory of social comparison processes. *Human Relations, 7*, 117-140.

Folmer, E. (2007). *Designing usable and accessible games with interaction design patterns*. Retrieved April 10, 2008, from http://www.gamasutra.com/view/feature/1408/designing_usable_and_accessible_.php

Foo, J. W. (2005). *Strategem: A game of tactics*. SCE final year project, Nanyang Technological University, Singapore.

Foo, J. W., & Kevin, J. (2005, November). The inspiration model for games development. In *Proceedings of the Cyber Games Conference* (pp. 145-147). Singapore.

Frank, H. F., Leung, H. K., Lam, S., Ling, H. & Tam, P. K. S. (2003). Tuning of the structure and parameters of a neural network using improved genetic algorithm. *IEEE Transactions on Neural Networks, 14*, 79-88.

Goldberg, D. E.. (1989). *Genetic algorithms for search, optimization and machine learning*. Reading, MA: Addison-Wesley.

Goth, G. (2006). Next-gen game landscape extends beyond the console. *IEEE Distributed Systems Online, 7*(9), Art. No. 0609-o9004.

Ghoulash. (2007). *Ghoulash: The last game on earth*. http://www.ghoulash.com

Halo 3. (2008). *Halo 3 game engine*. Retrieved October 9, 2009, from http://en.wikipedia.org/wiki/Halo_3

Henru, R. A., Chaudhari, N. S., & Prakash, E. C. (2005, March). Emergent behavior, cellular automata, and our game. In *Proceedings of Cybergames: Intl. Workshop on Games Research and Development* (pp. 11-19). Singapore.

id. (2008). id Software Inc. www.idsoftware.com

Kaukoranta, T., Smed, J., & Hakonen, H. (2003). Understanding pattern recognition methods. *AI Game Programming Wisdom, 2*, 579-589.

Kennerly, D. (2003). *Better game design through data mining*. Retrieved October 9, 2009, from http://www.gamasutra.com/features/20030815/kennerly_01.shtml

Klein, G. (1999). *Sources of power: How people make decisions*. Cambridge: MIT Press.

Kleinberg, J. (2000). The small-world phenomenon: An algorithmic perspective. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing* (pp. 163-170). New York: ACM Press.

Koh, W. L., & Zhou, S. P. (2007, October). An extensible collision avoidance model for realistic self-driven autonomous agents. In *Proceedings of the 11th IEEE International Symposium on Distributed Simulation and Real-Time Applications* (pp. 7-14). Crete, Greece.

Lindstrom, P., Koller, D., Ribarsky, W., Hodges, L. F., Faust, N., & Turner, G. A. (1996, August). Real-time, continuous level of detail rendering of height fields. In *Proceedings of the 23rd ACM Annual Conference on Computer Graphics and interactive Techniques SIGGRAPH '96* (pp. 109-118). New Orleans, LA.

Longbow digital arts, Treadmarks, http://www.ldagames.com/treadmarks/.

Losasso, F., & Hoppe, H. (2004). Geometry clipmaps: Terrain rendering using nested regular grids. *ACM Transactions on Graphics, 23*, 769-776.

Louis, S. J., & Miles, C. (2005). Playing to learn: Case-injected genetic algorithms for learning to play computer games. *IEEE Transactions on Evolutionary Computation, 9*, 669-681.

Lotfi A. Zadeh, Fuzzy Logic, Neural Networks and Soft Computing, Communications of the ACM, 37(3):77-84, 1994.

Majoros, W. (2002). Syntactic structure in birdsong: Memetic evolution of songs or grammars? *Journal of Memetics: Evolutionary Models of Information Transmission, 6*. Retrieved October 9, 2009, from http://jom-emit.cfpm.org/2002/vol6/ majoros_w.html

McNally, S., "Treadmarks Engine (Binary Trees and Terrain Tessellation)," http://www.ldagames.com/.

Medzseva, R. (2008). *Design and development of generic 2D game engine (for a 2D Java game)*. SCE Final Year Project, Nanyang Technological University.

Mezoyer, J. (1987). A six states minimum time solution to the firing squad synchronization problem. *Theoretical Computer Science, 50*, 183-238.

Mitchell, M. (1996). *An introduction to genetic algorithms. Complex adaptive systems*. Cambridge: MIT Press.

Mitchell, M., Crutchfield, J. P., & Das, R. (1997). Evolving cellular automata to perform computations. In T. Bäck, D. Fogel, & Z. Michalewicz (Eds.), *Handbook of evolutionary computation* (pp. G5.1.1-G5.1.8). Bristol, UK/Oxford, UK: IOP/Oxford University Press.

Mitchell, M., Crutchfield J. P., & Hraber, P. T. (1994). Evolving cellular automata to perform computations: Mechanisms and impediments. *Physica D, 75*, 361-391.

Morgan, G. (2009). Are we there yet? Meeting the challenges of online game development. *Simulation & Gaming*. **[IN PRESS]**.

Nakatsu, R., & Hoshino, J. (2002). *Entertainment computing: Technologies and applications*. Dordrecht, The Netherlands: Kluwer Academic.

Nykamp, M. (2001). *The customer differential: The complete guide to implementing customer relationship management*. New York: AMACOM.

Omlin, C. W., & Giles, C. L. (1996). Constructing deterministic finite-state automata in recurrent neural networks. *Journal of ACM, 43*, 937-972.

Oulasvirta, A., & Blom, J. (2007). Motivations in personalisation behavior. *Interacting With Computers, 20*, 1-16.

Ovaska, S. J., & Sick, B. (2006). Fusion of soft computing and hard computing: Applications and research opportunities. In G. Y. Yen & D. B. Fogel (Eds.), *Computational intelligence: Principles and practice* (pp. 47-72). Piscataway, NJ: IEEE Computational Intelligence Society.

Pajarola, R., & Gobbetti, E. (2007). Survey on semi-regular multiresolution models for interactive terrain rendering. *Visual Computer, 23*, 583-605.

Pitts, W., & McCulloch, W. S. (1947). How we know universals: The perception of auditory and visual forms. *Bulletin of Mathematical Biology, 9*, 127-147.

S. Rajasekaran and G.A. Vijayalakshmi Pai, "Genetic Algorithm Based Weight Determination for Backpropagation Networks", Fourth International Conference on Advanced Computing (ADCOMP 1996), Bangalore, India, pp. 73-79, December, 1996.

Ram, A., Ontañón, S., & Mehta, M. (2007). Artificial intelligence for adaptive computer games. *Proceedings of the Twentieth International FLAIRS Conference on Artificial Intelligence (FLAIRS-2007)*. Menlo Park, CA: AAAI Press.

Raveendran, K. (2007). *A practical guide to CodeRuler*. PowerPoint presentation aired on Episode 7 of Media Corp's iWhiz at 20:00 hours on March 26, 2007.

SBSoftware. (2008). *DeePsea DooM game editor*. Author.

Starter Kits. (2007). Retrieved July 20, 2007, from http://creators.xna.com/Education/StarterKits.aspx

Siegelmann, H. T., & Sontag, E. D. (1991). Turing computability with neural nets, *Applied Mathematics Letters, 4*(6), 77-80.

Siegelmann, H. T. (1999). *Neural networks and analog computation: Beyond the Turing limit*. Boston: Birkhauser.

Snook, G. (2003). *Real-time 3D terrain engines using C++ and Direct X*. Charles River Media Games Development.

Spronck, P., & Herik, J. (2004). Game artificial intelligence that adapts to the human player. *ERCIM News,* 57.

Stanley, K., Bryant, B., & Miikkulainen, R. (2005). Real-time neuroevolution in the NERO video game. *IEEE Transactions on Evolutionary Computation, 9*, 653-668.

Strogaz, S. H. (2003). *Sync: The emerging science of spontaneous order*. Los Angeles: Hyperion.

Tozour, P. (2002). The evolution of game AI. In *AI game programming wisdom* (pp. 3-16). Hingham, MA: Charles River Media.

Turchin, V. F. (1993). Program transformation with metasystem transitions. *Journal of Functional Programming, 3*, 283-313.

Turchin, V. F. (1996a). Metacomputation: Metasystem transitions plus supercompilation. In O. Danvy, R. Glück, & P. Thiemann (Eds.), *Lecture notes in computer science: Vol. 1110. Partial evaluation* (pp. 481-510). Berlin: Springer.

Turchin, V. F. (1996b). Supercompilation: Techniques and results. In D. Bjørner, M. Broy, & I. V. Pottosin (Eds.), *Lecture notes in computer science: Vol. 1181. Perspectives of system informatics* (pp. 229-248). Berlin: Springer.

von Neumann, J. (1967). *Theory of self-reproducing automata*. Urbana: University of Illinois Press.
Watts, D. J. (2003). *Six degrees: The science of connected age*. New York: W. W. Norton.
Watts, D. J., Dodds, P. S., & Newman, M. E. J. (2002). Identity and search in social networks. *Science, 296*, 1302-1305.
White, M. (2007). Adapting Roam for use within a games application. In *Proceedings of Cyber Games 2007*, Manchester, UK.
Wolfram, S. (2002). *A new kind of science*. Champaign, IL: Wolfram Media.
Wong, K. K. (2007). Player adaptive entertainment computing. In *Proceedings of the 2nd International Conference on Digital Interactive Media in Entertainment and Arts (DIMEA '07)* (Vol. 274, p. 13). Perth, Australia.
Wood, D. (1987). *Theory of computation*. New York: Harper & Row.

Bios

Edmond Prakash is the Editor-in-Chief of the *International Journal of Computer Games Technology*. He currently leads the Research Group on Games Technology at the Department of Computing and Mathematics, Manchester Metropolitan University (MMU), Manchester, UK. Prior to joining MMU, he was an assistant professor of computer science, and he was also the research director of the gameLAB, School of Computer Engineering, Nanyang Technological University, Singapore. Contact: e.prakash@mmu.ac.uk

Geoff Brindle is a senior lecturer in the Department of Computing and Mathematics at the Manchester Metropolitan University (MMU) where he is Subject Leader for Computer Games Technology. Geoff has extensive experience in designing Computer Games Technology curriculum for the undergraduate and postgraduate programs at MMU. He has previously worked at the Ferranti Microelectronics Centre and the University of Manchester as a software engineer where he was concerned with the design and implementation of computer graphics and cad systems for electronic engineers. Currently, his main interests are in the areas of computer graphics algorithms and novel game interfaces. Contact: g.brindle@mmu.ac.uk

Kevin Jones is a full-time lecturer in the School of Computer Engineering; he joined Nanyang Technological University (NTU) in July 2002. His interests are in software development and implementation, and he is an expert in UML. Previous to NTU, Kevin was involved in the provision of adult education, both in the public and private sectors. He has developed professional training programmes for software consultants, software developers, and communication equipment technicians. He served in the Canadian Signals School as a Chief Instructor and Computer Course developer.

Suiping Zhou is currently an Assistant Professor in the School of Computer Engineering at Nanyang Technological University, Singapore. Previously, he worked as an engineer in Beijing Simulation Center, China Aerospace Corporation, and then joined Weizmann Institute of Science, Israel as a Post-doctoral fellow. He received his B.Eng., M.Eng. and Ph.D in Electrical Engineering from Beijing University of Aeronautics and Astronautics, P.R. China. He is a member of IEEE and his current research interests include: large-scale distributed interactive

applications (e.g., MMOGs), parallel/distributed systems, and human behaviour representation in modeling and simulation. He has published more than 60 peer reviewed articles in these areas. He is currently an associate editor of the International Journal of Computer Games Technology.

Narendra S. Chaudhari received his BTech (with distinction), MTech, and PhD degrees from the Indian Institute of Technology, Mumbai, India, in 1981, 1983, and 1988, respectively. After completing his BTech degree in electrical engineering, he worked on the design of microprocessor-based electronic controllers in the Electronic Controls Division of Larsen and Toubro Ltd, Mumbai, in 1981. He was a visiting professor in Southern Cross University, New South Wales, Australia, and Freie Universität, Berlin. Currently, he is with the School of Computer Engineering, Nanyang Technological University, Singapore, as an associate professor since 2001. His research interests are in the areas of neural networks, computational learning, and optimization algorithms. He is a Fellow of Institution of Electronics and Telecommunication Engineers, India, Chartered Engineer of Institution of Engineers, India, Senior Member of Computer Society of India, and member of many professional societies, including Singapore Computer Society, and Pattern Recognition and Machine Intelligence Association, Singapore. Contact: asnarendra@ntu.edu.sg

Kok-Wai Wong received the BEng (Hons) and PhD degrees from the School of Electrical and Computer Engineering, Curtin University of Technology, Western Australia, in 1994 and 2000, respectively. He is currently an associate professor in the School of Information Technology, Murdoch University, Murdoch, Western Australia. Prior to this appointment, he was an assistant professor at the School of Computer Engineering, Nanyang Technological University, Singapore. His research interests include intelligent data analysis, fuzzy rule based systems, computational intelligence, and game technology.