1.1

Common Game AI Techniques

Steve Rabin—Nintendo of America Inc.

steve@aiwisdom.com

The emerging field of game artificial intelligence (AI) has made significant progress over the last decade. From the flurry of information shared through conferences, magazines, Web sites, and books, there has evolved a loose consensus on game AI techniques that are commonly used in practice, and AI techniques that look promising for the future. This article and the following article ("Promising Game AI Techniques") attempt to survey these algorithms and architectures so that you can easily see where the field stands and where it is going.

A main goal of these two articles is to boil down each technique into a couple of descriptive paragraphs. By forcing each topic into such a confined space, the crust is scraped off and you can clearly see what these techniques are all about. Each explanation is followed by references so that you can further explore the details.

A* Pathfinding

A pathfinding* (pronounced A-star) is an algorithm for finding the cheapest path through an environment. Specifically, it is a directed search algorithm that exploits knowledge about the destination to intelligently guide the search. By doing so, the processing required to find a solution is minimized. Compared to other search algorithms, A* is the fastest at finding the absolute cheapest path. Note that if all movement has the same traversal cost, the cheapest path is also the shortest path.

Game Example

The environment must first be represented by a data structure that defines where movement is allowed [Tozour03a]. A path is requested by defining a start position and a goal position within that search space. When A* is run, it returns a list of points, like a trail of breadcrumbs, that defines the path. A character or vehicle can then use the points as guidelines to find its way to the goal.

A* can be optimized for speed [Cain02, Higgins02b, Rabin00a], for aesthetics [Rabin00b], and for general applicability to other tasks [Higgins02a]. Variations like D* attempt to make path re-planning cheaper [Stentz94].

ON THE CD

As a bonus, an A* tutorial used at the Full Sail School of Game Design & Development is included on the accompanying CD-ROM [Mesdaghi04]. The tutorial walks you through different algorithms such as Breadth-First, Dijkstra, Best-First, and finally A* in order to give you a deep understanding of how A* works, and why it is better than the other planning algorithms.

Further Information

Google Search: <"A*" path finding pathfinding>
***Game Programming Gems*:** [Rabin00a, Rabin00b, Stout00]
***AI Game Programming Wisdom*:** [Cain02, Higgins02, Matthews02]
***AI Game Programming Wisdom 2 CD-ROM*:** [Mesdaghi04]

Command Hierarchy

A *command hierarchy* is a strategy for dealing with AI decisions at different levels, from the general down to the foot soldier. Modeled after military hierarchies, the general directs the high-level strategy on the battlefield, while the foot soldier concentrates on individual combat. The levels in between deal with cooperation between various platoons and squads. The benefit of a command hierarchy is that decisions are separated at each level, thus making each decision more straightforward and abstracted from other levels.

Game Example

A command hierarchy is often used in real-time strategy or turn-based strategy games where there are typically three easily identifiable levels of decisions: overall strategy, squad tactics, and individual combat. A command hierarchy is also useful when a large number of agents must have an overall coherency.

Further Information

Google Search: <AI command hierarchy -class>
***AI Game Programming Wisdom*:** [Reynolds02]
***AI Game Programming Wisdom 2*:** [Kent04]

Dead Reckoning

Dead reckoning is a method for predicting a player's future position based on that player's current position, velocity, and acceleration. This simple form of prediction works well since the movement of most objects resembles a straight line over short periods of time. More advanced forms of dead reckoning can also provide guidance for how far an object *could have moved* since it was last seen.

Game Example

In a first-person shooter (FPS) game, an effective method of controlling the difficulty level is to vary how accurate the computer is at "leading the target" when shooting

projectiles. Since most weapons don't travel instantaneously, the computer must predict the future position of targets and aim the weapon at these predicted positions. Similarly, in a sports game, the computer player must anticipate the future positions of other players to effectively pass the ball or intercept a player.

Further Information

***AI Game Programming Wisdom*:** [Stein02]
***AI Game Programming Wisdom 2*:** [Laramée04]

Emergent Behavior

Emergent behavior is behavior that wasn't explicitly programmed but instead emerges from the interaction of several simpler behaviors. Many life forms use rather basic behavior that, when viewed as a whole, can be perceived as being much more sophisticated. In games, emergent behavior generally manifests itself as low-level simple rules that interact to create interesting and complex behaviors. Some examples of rules are seek food, seek similar creatures, avoid walls, and move toward the light. While any one rule isn't interesting by itself, unanticipated individual or group behavior can emerge from the interaction of these rules.

Game Example

Flocking is a classical example of emergent behavior in games [Reynolds87, Reynolds01]. Another example is insect-like behavior described in this book by Alex Darby for racecar applications [Darby04] and Nick Porcino for general-purpose creatures [Porcino04].

Further Information

Google Search: <emergent behavior alife>
Internet: [Alife03, Reynolds87, Reynolds01]
***AI Game Programming Wisdom 2*:** [Darby04, Porcino04]

Flocking

Flocking is a technique for moving groups of creatures in a natural and organic manner. It works well at simulating flocks of birds, schools of fish, and swarms of insects. Each creature follows three simple movement rules that result in complex group behavior. It is said that this group behavior *emerges* from the individual rules (emergent behavior). Flocking is a form of artificial life that was popularized by Craig Reynolds' work [Reynolds87, Reynolds01].

The three classic flocking rules devised by Reynolds are:

- **Separation:** Steer to avoid crowding local flockmates.
- **Alignment:** Steer toward the average heading of local flockmates.
- **Cohesion:** Steer toward the average position of local flockmates.

Game Example

Games typically use flocking to control background creatures such as schools of fish or swarms of insects. Since the path of any one creature is highly arbitrary, flocking is typically used for simple creatures that tend to wander with no particular destination. The result is that flocking techniques, as embodied by the three core rules, rarely get used for key enemies or creatures. However, the flocking rules have inspired several other movement algorithms, such as formations and swarming [Scutt02].

Further Information

Google Search: <flocking boids>
Internet: [Reynolds87, Reynolds01]
***Game Programming Gems*:** [Woodcock00]
***AI Game Programming Wisdom*:** [Scutt02]

Formations

Formations are a group movement technique used to mimic military formations. Although it shares similarities to flocking, it is quite distinct in that each unit is guided toward a specific goal location and heading, based on its position in the formation.

Game Example

Formations can be used to organize the movement of ground troops, vehicles, or aircraft. Often, the formations must split or distort themselves to facilitate movement through tight areas [Pottinger99a, Pottinger99b].

Further Information

Google Search: <military formations>
Internet: [Pottinger99a, Pottinger99b]
***AI Game Programming Wisdom*:** [Dawson02]

Influence Mapping

An *influence map* is a method for viewing the distribution of power within a game world. Typically, it's a two-dimensional (2D) grid that is superimposed onto the landscape. Within each grid cell, units that lie in the cell are summed into a single number representing the combined influence of the units. It is assumed that each unit also has an influence into neighboring cells that falls off either linearly or exponentially with distance. The result is a 2D grid of numbers that gives insight into the location and influence of differing forces.

Game Example

Influence maps can be used offensively to plan attacks; for example, by finding neutral routes to flank the enemy. They can also be used defensively to identify areas or positions that need to be strengthened. If one faction is represented by positive values and the other faction is represented by negative values within the same influence map, then any grid cells near zero are either unowned territory or the "front" of the battle (where the influence of each side cancels each other out) [Tozour01].

There are also nonviolent uses for influence maps. For example, the *Sim City* series offers real-time maps that show the influence of police and fire departments placed around the city. The player can then use this information to place future buildings to fill in the gaps in coverage. The game also uses the same information to help simulate the world.

Further Information

Google Search: <"influence mapping" code>
***Game Programming Gems 2*:** [Tozour01]
***AI Game Programming Wisdom*:** [Woodcock02]
***AI Game Programming Wisdom 2*:** [Tozour04b, Sweetser04]

Level-of-Detail AI

Level-of-detail (LOD) is a common optimization technique in 3D graphics where polygonal detail is only used when it can be noticed and appreciated by the human viewer. Close-up models use large numbers of polygons, while faraway models use fewer polygons. This results in faster graphics processing since fewer polygons are rendered, yet there is no noticeable degradation in visual quality. The same concept can be applied to AI where computations are performed only if the player will notice or appreciate the result.

Game Example

One approach is to vary an agent's update frequency based on its proximity to the player. Another technique is to calculate paths only for agents that the player can see; otherwise, use straight-line path approximations and estimate off-screen movement. This technique becomes important when there are more than several dozen agents in a game and collectively they use too much processing power. This often occurs with RPG, RTS, strategy, and simulation games.

Further Information

Google Search: <"level of detail" AI>
***AI Game Programming Wisdom*:** [Brockington02a]
***AI Game Programming Wisdom 2*:** [Fu04]

Manager Task Assignment

When a group of agents tries to independently choose tasks to accomplish, like selecting a target in battle, the performance of the group can be rather dismal. Interestingly, the problem can be turned around so that instead of the individuals choosing tasks, a manager has a list of required tasks and assigns agents based on who is the best suited for the job. Note that this is very different from having the manager run through the list of individuals and assign tasks. Task assignment considers the tasks themselves first and uses them as the basis for prioritizing. This avoids duplication of tasks, and the best candidate for a task is always chosen. This type of tactical planning is more deliberate than the emergent coordination that can be achieved with a blackboard architecture. However, the resulting plan might not be as optimal as performing an exhaustive planning search [Orkin04a].

Game Example

In a baseball game with no runners on base, it might be determined that the first priority is to field the ball, the second priority is to cover first base, the third priority is to back up the person fielding the ball, and the fourth priority is to cover second base. The manager can organize who covers each priority by examining the best person for the job for a given situation. On a soft hit between first and second base, the manager might assign the first baseman to field the ball, the pitcher to cover first base, the second baseman to back up the first baseman fielding the ball, and the shortstop to cover second base. Without a manager to organize the task assignment, it can be significantly harder to get coherent cooperation out of the players using other methods.

Further Information

***AI Game Programming Wisdom 2* CD-ROM:** [Rabin98]

Obstacle Avoidance

A* pathfinding algorithms are good at getting a character from point to point through static terrain. However, often the character must avoid players, other characters, and vehicles that are moving rapidly through the environment. Characters must not get stuck on each other at choke points, and they must maintain enough spacing to maneuver when traveling in groups. *Obstacle avoidance* attempts to prevent these problems using trajectory prediction and layered steering behaviors [Reynolds99].

Game Example

In an FPS game, a group of four skeletons wants to attack the player, but must first cross a narrow bridge over a river. Each skeleton has received a route to the player through the navigation system. The skeleton closest to the bridge has a clear path across. The second skeleton predicts a collision with the first, but sees space to the right, which is still within the boundaries of the path across the bridge. The last two

skeletons predict collisions with the first two, so they slow their rate of travel to correctly queue up behind the first two.

Further Information

Internet: [Saffiotti98]
***AI Game Programming Wisdom 2* CD-ROM:** [Reynolds99]

Scripting

Scripting is the technique of specifying a game's data or logic outside of the game's source language. Often, the scripting language is designed from scratch, but there is a growing movement toward using Python and Lua as alternatives. There is a complete spectrum for how far you can take the scripting concept.

Scripting influence spectrum:

Level 0: Hard code everything in the source language (C/C++).
Level 1: Data in files specify stats and locations of characters/objects.
Level 2: Scripted cutscene sequences (noninteractive).
Level 3: Lightweight logic specified by tools or scripts, as in a trigger system.
Level 4: Heavy logic in scripts that rely on core functions written in C/C++.
Level 5: Everything coded in scripts—full alternative language to C/C++.

Commercial games have been developed at all levels of this spectrum, with the oldest video games at level 0 and games such as *Jak and Daxter* at level 5 (with their GOAL language based on LISP). However, the middle levels are where most games have settled, since the two extremes represent increased risk, time commitment, and cost.

Game Example

Programmers must first integrate a scripting language into the game and determine the extent of its influence. The users of the scripting language will typically be artists and level designers. The written script will either be compiled into byte code before actual gameplay or interpreted on the fly during gameplay.

Advantages of scripting:

- Game logic can be changed in scripts and tested without recompiling the code.
- Designers can be empowered without consuming programmer resources.
- Scripts can be exposed to the players to tinker with and expand (extensible AI).

Disadvantages of scripting:

- More difficult to debug.
- Nonprogrammers are required to program.
- Time commitment to create and support scripting language and complementary debugging tools.

Further Information

Google Search: <"scripting language" games AI>
***AI Game Programming Wisdom*:** [Barnes02, Brockington02b, Berger02, Poiker02, Tozour02]
***AI Game Programming Wisdom 2*:** [Herz04a, Herz04b, Kharkar04, Orkin04b, Snavely04]

State Machine

A *state machine* or *finite-state machine* (FSM) is a widely used software design pattern that has become a cornerstone of game AI. An FSM is defined by a finite set of states and transitions, with only one state active at any one time. In common practice, each state represents a behavior, such as `PatrolRoute`, within which an agent will perform a specific task. The state either polls or listens for events that will cause it to transition into other states. For example, a `PatrolRoute` state might check periodically if it sees an enemy. When this event happens, it transitions into the `AttackEnemy` state. An exhaustive explanation of FSMs for games, along with common enhancements, can be found in this book [Fu04].

Further Information

Google Search: <finite state machines AI>
***AI Game Programming Wisdom 2*:** [Fu04]

Stack-Based State Machine

A *stack-based state machine* is a technique and design pattern that often appears in game architectures. Also sometimes referred to as *push-down automata,* the stack-based state machine can remember past actions by storing them on a stack. In a traditional state machine, past states are not remembered, since control flows from state to state with no recorded history. However, it can be useful in game AI to be able to transition back to a previous state, regardless of which state it was. This stack concept can be used to capture previous states, or even entire state machines.

Game Example

In a game, this technique is important when a character is performing an action, becomes interrupted for a moment, but then wants to resume the original action. For example, in a real-time strategy game, a unit might be repairing a building when it gets attacked. The unit will transition into an attack behavior and might destroy the enemy. In this case, the conflict is over and the unit should resume its previous activity. If past behaviors are stored on a stack, then the current attack behavior is simply popped from the stack and the unit will resume the repair behavior.

Further Information

***AI Game Programming Wisdom 2*:** [Fu04, Tozour04c, Yiskis04a]

Subsumption Architecture

A *subsumption architecture* cleanly separates the behavior of a single character into concurrently running layers of FSMs. The lower layers take care of rudimentary behavior such as obstacle avoidance, and the higher layers take care of more elaborate behaviors such as goal determination and goal seeking. Because the lower layers have priority, the system remains robust and ensures that lower layer requirements are met before allowing higher level behaviors to influence them. The subsumption architecture was popularized by the work of Rodney Brooks [Brooks89].

Game Example

Subsumption architectures have been used in many games, including the *Oddworld* series of games, *Jedi Knight: Dark Forces 2*, and *Halo: Combat Evolved.* The architecture is ideally suited for character-based games where movement and sensing must coexist with decisions and high-level goals.

Further Information

Google Search: <subsumption brooks>
***AI Game Programming Wisdom 2*:** [Yiskis04b]

Terrain Analysis

Terrain analysis is the broad term given to analyzing the terrain of a game world in order to identify strategic locations such as resources, choke points, or ambush points [Higgins02c]. These locations can then be used by the strategic-level AI to help plan maneuvers and attacks. Other uses for terrain analysis in a real-time strategy game include knowing where to build walls [Grimani04] or where to place the starting factions. In a first-person shooter (FPS) game, terrain analysis can assist the AI in discovering sniper points, cover points, or where to throw grenades from [Lidén02, Reed03, Tozour03b, van der Sterren00]. Terrain analysis can be viewed as the alternative approach to "hard-coding" regions of interest in a level.

Further Information

Google Search: <"terrain analysis" AI>
Internet: [van der Sterren00]
***Game Programming Gems 3*:** [Higgins02c]
***AI Game Programming Wisdom*:** [Lidén02]
***AI Game Programming Wisdom 2*:** [Kent04, Reed04, Tozour04b]

Trigger System

A *trigger system* is a highly specialized scripting system that allows simple if/then rules to be encapsulated within game objects or the world itself. It is a useful tool for level designers since the concept is extremely simple and robust. Often, it is exposed through a level design tool or a scripting language.

Game Example

A designer might put a floor trigger in the middle of a room. When the player steps on the floor trigger (the condition), the designer might specify that a scary sound effect is played and that a dozen snakes drop from the ceiling (the response). In this way, a trigger system is a simple way to specify scripted events without designing a complex scripting language. As an example, the level editor for *StarCraft* allowed users to define their own missions with a Windows-based trigger system tool.

Further Information

Google Search: <"trigger system" scripting>
***Game Programming Gems 3*:** [Rabin02]
***AI Game Programming Wisdom*:** [Orkin02]

Acknowledgments

Several people helped contribute to the ideas and knowledge within this article. In particular, Mark Brockington, Daniel Higgins, Lars Lidén, John Manslow, Syrus Mesdaghi, Jeff Orkin, Steven Woodcock, and Eric Yiskis helped a great deal.

References

[Alife03] International Society of Artificial Life, *www.alife.org*.
[Barnes02] Barnes, Jonty, and Hutchens, Jason, "Scripting for Undefined Circumstances," *AI Game Programming Wisdom*, Charles River Media, 2002.
[Berger02] Berger, Lee, "Scripting: Overview and Code Generation," *AI Game Programming Wisdom*, Charles River Media, 2002.
[Brockington02a] Brockington, Mark, "Level-of-Detail AI for a Large Role-Playing Game," *AI Game Programming Wisdom*, Charles River Media, 2002.
[Brockington02b] Brockington, Mark, and Darrah, Mark, "How *Not* to Implement a Basic Scripting Language," *AI Game Programming Wisdom*, Charles River Media, 2002.
[Brooks89] Brooks, Rodney, "How to Build Complete Creatures Rather than Isolated Cognitive Simulators," *Architectures for Intelligence*, Lawrence Erlbaum Associates, Fall 1989, available online at *www.ai.mit.edu/people/brooks/papers/how-to-build.pdf*
[Cain02] Cain, Timothy, "Practical Optimizations for A* Path Generation," *AI Game Programming Wisdom*, Charles River Media, 2002.
[Darby04] Darby, Alex, "Vehicle Racing Control Using Insect Intelligence," *AI Game Programming Wisdom 2*, Charles River Media, 2004.

[Dawson02] Dawson, Chad, "Formations," *AI Game Programming Wisdom*, Charles River Media, 2002.
[Fu04] Fu, Dan, and Houlette, Ryan, "The Ultimate Guide to FSMs in Games," *AI Game Programming Wisdom 2*, Charles River Media, 2004.
[Grimani04] Grimani, Mario, "Wall Building for RTS Games," *AI Game Programming Wisdom 2*, Charles River Media, 2004.
[Herz04a] Herz, Alex, "Optimized Script Execution," *AI Game Programming Wisdom 2*, Charles River Media, 2004.
[Herz04b] Herz, Alex, "Advanced Script Debugging," *AI Game Programming Wisdom 2*, Charles River Media, 2004.
[Higgins02a] Higgins, Dan, "Generic A* Pathfinding," *AI Game Programming Wisdom*, Charles River Media, 2002.
[Higgins02b] Higgins, Dan, "How to Achieve Lightning-Fast A*," *AI Game Programming Wisdom*, Charles River Media, 2002.
[Higgins02c] Higgins, Dan, "Terrain Analysis in an RTS—The Hidden Giant," *Game Programming Gems 3*, Charles River Media, 2002.
[Kent04] Kent, Tom, "Multi-Tiered AI Layers and Terrain Analysis for RTS Games," *AI Game Programming Wisdom 2*, Charles River Media, 2004.
[Kharkar04] Kharkar, Sandeep, "A Modular Camera Architecture for Intelligent Control," *AI Game Programming Wisdom 2*, Charles River Media, 2004.
[Laramée04] Laramée, François Dominic, "Dead Reckoning in Sports and Strategy Games," *AI Game Programming Wisdom 2*, Charles River Media, 2004.
[Lidén02] Lidén, Lars, "Strategic and Tactical Reasoning with Waypoints," *AI Game Programming Wisdom*, Charles River Media, 2002.
[Matthews02] Matthews, James, "Basic A* Pathfinding Made Simple," *AI Game Programming Wisdom*, Charles River Media, 2002.
[Mesdaghi04] Mesdaghi, Syrus, "Path Planning Tutorial," *AI Game Programming Wisdom 2* CD-ROM, Charles River Media, 2004.
[Orkin02] Orkin, Jeff, "A General-Purpose Trigger System," *AI Game Programming Wisdom*, Charles River Media, 2002.
[Orkin04a] Orkin, Jeff, "Applying Goal-Oriented Action Planning to Games," *AI Game Programming Wisdom 2*, Charles River Media, 2004.
[Orkin04b] Orkin, Jeff, "Adding Error Reporting to Scripting Languages," *AI Game Programming Wisdom 2*, Charles River Media, 2004.
[Poiker02] Poiker, Falko, "Creating Scripting Languages for Nonprogrammers," *AI Game Programming Wisdom*, Charles River Media, 2002.
[Porcino04] Porcino, Nick, "An Architecture for A-Life," *AI Game Programming Wisdom 2*, Charles Rivers Media, 2004.
[Pottinger99a] Pottinger, Dave, "Coordinated Unit Movement," *Game Developer Magazine*, January 1999, available online at *www.gamasutra.com/features/19990122/movement_01.htm*
[Pottinger99b] Pottinger, Dave, "Implementing Coordinated Movement," *Game Developer Magazine*, February 1999, available online at *www.gamasutra.com/features/19990129/implementing_01.htm*
[Rabin98] Rabin, Steve, "Making the Play: Team Cooperation in Microsoft Baseball 3D," Computer Game Developers Conference Proceedings, 1998, available on the *AI Game Programming Wisdom 2* CD-ROM.
[Rabin00a] Rabin, Steve, "A* Speed Optimizations," *Game Programming Gems*, Charles River Media, 2000.
[Rabin00b] Rabin, Steve, "A* Aesthetic Optimizations," *Game Programming Gems*, Charles River Media, 2000.

[Rabin02] Rabin, Steve, "An Extensible Trigger System for AI Agents, Objects, and Quests," *Game Programming Gems 3,* Charles River Media, 2002.
[Reed03] Reed, Christopher, and Geisler, Benjamin, "Jumping, Climbing, and Tactical Reasoning: How to Get More out of a Navigation System," *AI Game Programming Wisdom 2,* Charles River Media, 2003.
[Reynolds87] Reynolds, Craig, "Flocks, Herds, and Schools: A Distributed Behavioral Model," *Computer Graphics, 21(4) (SIGGRAPH '87 Conference Proceedings),* pp. 25–34, 1987, available online at *www.red3d.com/cwr/papers/1987/boids.html*
[Reynolds99] Reynolds, Craig, "Steering Behaviors for Autonomous Characters," Game Developers Conference Proceedings, 1999, available on the *AI Game Programming Wisdom 2* CD-ROM.
[Reynolds01] Reynolds, Craig, "Boids," available online at *www.red3d.com/cwr/boids/*
[Reynolds02] Reynolds, John, "Tactical Team AI Using a Command Hierarchy," *AI Game Programming Wisdom,* Charles River Media, 2002.
[Saffiotti98] Saffiotti, Alessandro, "Autonomous Robot Navigation," Handbook of Fuzzy Computation, Oxford Univ. Press and IOP Press, 1998, available online at *aass.oru.se/Agora/FLAR/HFC/home.html*
[Scutt02] Scutt, Tom, "Simple Swarms as an Alternative to Flocking," *AI Game Programming Wisdom,* Charles River Media, 2002.
[Snavely04] Snavely, P.J., "Empowering Designers: Defining Fuzzy Logic Behavior through Excel-Based Spreadsheets," *AI Game Programming Wisdom 2,* Charles River Media, 2004.
[Stein02] Stein, Noah, "Intercepting a Ball," *AI Game Programming Wisdom,* Charles River Media, 2002.
[Stentz94] Stentz, Tony, "Original D*," *ICRA 94,* 1994, available online at *www.frc.ri.cmu.edu/~axs/doc/icra94.pdf*
[Stout00] Stout, Bryan, "The Basics of A* for Path Planning," *Game Programming Gems,* Charles River Media, 2000.
[Sweetser04] Sweetser, Penny, "Strategic Decision-Making with Neural Nets and Influence Maps," *AI Game Programming Wisdom 2,* Charles River Media, 2004.
[Tozour01] Tozour, Paul, "Influence Mapping," *Game Programming Gems 2,* Charles River Media, 2001.
[Tozour02] Tozour, Paul, "The Perils of AI Scripting," *AI Game Programming Wisdom,* Charles River Media, 2002.
[Tozour04a] Tozour, Paul, "Search Space Representations," *AI Game Programming Wisdom 2,* Charles River Media, 2004.
[Tozour04b] Tozour, Paul, "Using a Spatial Database for Runtime Spatial Analysis," *AI Game Programming Wisdom 2,* Charles River Media, 2004.
[Tozour04c] Tozour, Paul, "Stack-Based Finite-State Machines," *AI Game Programming Wisdom 2,* Charles River Media, 2004.
[van der Sterren00] van der Sterren, William, "AI for Tactical Grenade Handling," *CGF-AI,* 2000, available online at *www.cgf-ai.com/docs/grenadehandling.pdf*
[Woodcock00] Woodcock, Steven, "Flocking: A Simple Technique for Simulating Group Behavior," *Game Programming Gems,* Charles River Media, 2000.
[Woodcock02] Woodcock, Steven, "Recognizing Strategic Dispositions: Engaging the Enemy," *AI Game Programming Wisdom,* Charles River Media, 2002.
[Yiskis04a] Yiskis, Eric, "Finite-State Machine Scripting Language for Designers," *AI Game Programming Wisdom 2,* Charles River Media, 2004.
[Yiskis04b] Yiskis, Eric, "A Subsumption Architecture for Character-Based Games," *AI Game Programming Wisdom 2,* Charles River Media, 2004.

1.2

Promising Game AI Techniques

Steve Rabin—Nintendo of America, Inc.
steve@aiwisdom.com

Some of the most promising game AI techniques have been lurking in the shadows for years. Many have been used in games, but most are still on the fringe of mainstream game development. What these techniques lack is not usefulness, but rather the impetus for game developers to take them seriously. Too often we become lazy and overly complacent with simpler techniques, failing to realize the potential that lies beyond our own comfortable knowledge. Current trends suggest that it is mostly academics who transfer into the game industry who push these newer techniques, but this needn't be the case. This article attempts to get the word out in the hope that you'll take up the challenge and personally push the limits of mainstream game AI.

Bayesian Networks

Bayesian networks allow an AI to perform complex humanlike reasoning when faced with uncertainty. In a Bayesian network, variables relating to particular states, features, or events in the game world are represented as nodes in a graph, and the causal relationships between them as arcs. Probabilistic inference can then be performed on the graph to infer the values of unknown variables, or conduct other forms of reasoning.

Game Example

One particularly important application for Bayesian networks in games lies in modeling what an AI should believe about the human player based on the information it has available. For example, in a real-time strategy game, the AI can attempt to infer the existence or nonexistence of certain player-built units, like fighter planes or warships, based on what it has seen produced by the player so far. This keeps the AI from cheating and actually allows the human to deceive the AI by presenting misleading information, offering new gameplay possibilities and strategies for the player.

Further Information

Google Search: <"bayesian networks" nets>
Internet: [IDIS99]
AI Game Programming Wisdom: [Tozour02]

Blackboard Architecture

A *blackboard architecture* is designed to solve a single complex problem by posting it on a shared communication space, called the *blackboard.* Expert objects then look at the blackboard and propose solutions. The solutions are given a relevance score, and the highest scoring solution (or partial solution) is applied. This continues until the problem is "solved."

Game Example

In games, the blackboard architecture can be expanded to facilitate cooperation among multiple agents. A problem, such as attacking a castle, can be posted, and individual units can propose their role in the attack. The volunteers are then scored and the most appropriate ones are selected [Isla02].

Alternatively, the blackboard concept can be relaxed by using it strictly as a shared communication space, letting the individual agents regulate any cooperation. In this scheme, agents post their current activities and other agents can consult the blackboard to avoid beginning redundant work. For example, if an alarm is sounded in a building and enemies start rushing the player, it might be desirable for them to approach from different doors. Each enemy can post the door through which it will eventually enter, thus encouraging other enemies to choose alternate routes. Within this book, Jeff Orkin describes many cooperation problems within FPSs that can be solved using this approach [Orkin04].

Further Information

Google Search: <blackboard architecture architectures>
***AI Game Programming Wisdom*:** [Isla02]
***AI Game Programming Wisdom 2*:** [Orkin04]

Decision Tree Learning

A *decision tree* is a way of relating a series of inputs (usually measurements from the game world) to an output (usually representing something you want to predict) using a series of rules arranged in a tree structure. For example, inputs representing the health and ammunition of a bot could be used to predict the probability of the bot surviving an engagement with the player. At the root node, the decision tree might test to see whether the bot's health is low, indicating that the bot will not survive if that is the case. If the bot's health is not low, the decision tree might then test to see how much ammunition the bot has, perhaps indicating that the bot will not survive if its ammunition is low, and will survive otherwise. Decision trees are particularly important for applications like in-game learning because (in contrast to competing technologies like neural networks) extremely efficient algorithms exist for creating decision trees in near real-time.

Game Example

The best known game-specific use of decision trees is in the game *Black & White* where they are used to allow the creature to learn and form "opinions" [Evans02]. In *Black & White,* a creature will learn what objects in the world are likely to satisfy his desire to eat, based on feedback it gets from the player or world. For example, the player can provide positive or negative feedback by stroking or slapping the creature. A decision tree is then created that reflects what the creature has learned from its experiences. The creature can then use the decision tree to decide whether certain objects can be used to satisfy its hunger. While *Black & White* has demonstrated the power of decision trees to learn within games, they remain largely untapped by the rest of the game industry.

Further Information

Google Search: <decision tree learning>
Google Search: <decision tree ID3 ID4>
***AI Game Programming Wisdom*:** [Evans02]
***AI Game Programming Wisdom 2*:** [Fu04]

Filtered Randomness

Filtered randomness attempts to ensure that random decisions or events in a game appear random to the players. This can be achieved by filtering the results of a random number generator such that non-random looking sequences are eliminated, yet statistical randomness is maintained. For example, if a coin is flipped eight times in a row and turns up heads every time, a person might wonder if there was something wrong with the coin. The odds of such an event occurring are only 0.4 percent, but in a sequence of 100 flips it is extremely likely that either eight heads or eight tails in a row will be observed. When designing a game for entertainment purposes, random elements should always appear random to the players.

Game Example

Simple randomness filtering is actually very common in games. For example, if a character plays a random idle animation, often the game will ensure that the same idle animation won't be played twice in a row. However, filtering can be devised to remove all peculiar sequences. For example, if an enemy can randomly spawn from five different points, it would be extremely undesirable for the enemy to spawn from the same point five times in a row. It would also be undesirable for the enemy to randomly spawn in the counting sequence 12345 or favor one or two particular spawn points in the short term, like 12112121. Although these sequences can arise by chance, they are neither intended nor anticipated when the programmer wrote the code to randomly choose a spawn point. In this book, Steve Rabin describes how to effectively filter random number sequences and provides an assortment of classes on the accompanying CD-ROM [Rabin04].

ON THE CD

Further Information

***AI Game Programming Wisdom 2*:** [Rabin04]

Fuzzy Logic

Fuzzy logic is an extension of classical logic that is based on the idea of a fuzzy set. In classical crisp set theory, an object either does or does not belong to a set. For example, a creature is a member of the set of hungry creatures or is not a member of that set (it is either hungry or not hungry). With fuzzy set theory, an object can have continuously varying degrees of membership in fuzzy sets. For example, a creature could be hungry with degree of membership 0.1, representing slightly hungry, or 0.9, representing very hungry, or any value in between.

Further Information

Google Search: <"fuzzy logic">
***Game Programming Gems*:** [McCuskey 00]
***AI Game Programming Wisdom*:** [Zarozinsky02]
***AI Game Programming Wisdom 2*:** [Snavely04]
***AI Game Programming Wisdom 2* CD-ROM:** [Zarozinsky04]

Genetic Algorithms

A *genetic algorithm* (GA) is a technique for search and optimization that is based on evolutionary principles. GAs represent a point within a search space using a chromosome that is based on a handcrafted genetic code. Each chromosome consists of a string of genes that together encode its location in the search space. For example, the parameters of an AI agent can be the genes, and a particular combination of parameters a chromosome. All combinations of parameters will represent the search space.

By maintaining a population of chromosomes, which are continually mated and mutated, a GA is able to explore search spaces by testing different combinations of genes that seem to work well. A GA is usually left to evolve until it discovers a chromosome that represents a point in the search space that is good enough. GAs outperform many other techniques in search spaces that contain many optima, and are controlled by only a small number of parameters, which must be set by trial and error.

Game Example

Genetic algorithms are very good at finding a solution in complex or poorly understood search spaces. For example, your game might have a series of settings for the AI, but because of interactions between the settings, it is unclear what the best combination would be. In this case, a GA can be used to explore the search space consisting of all combinations of settings to come up with a near-optimal combination. This is typically done offline since the optimization process can be slow and because a near-optimal solution is not guaranteed, meaning that the results might not improve gameplay.

Further Information

Google Search: <"genetic algorithms" games>
***AI Game Programming Wisdom*:** [Laramée02a]
***AI Game Programming Wisdom 2*:** [Buckland04a, Laramée04, Sweetser04a, Thomas04]
***AI Game Programming Wisdom 2* CD-ROM:** [Buckland04b]

N-Gram Statistical Prediction

An *n-gram* is a statistical technique that can predict the next value in a sequence. For example, in the sequence 18181810181, the next value will probably be an 8. When a prediction is required, the sequence is searched backward for all sequences matching the most recent *n*-1 values, where *n* is usually 2 or 3 (a *bigram* or *trigram*). Since the sequence might contain many repetitions of the n-gram, the value that most commonly follows is the one that is predicted. If the sequence is built up over time, representing the history of a variable (like the last player's move), then a future event can be predicted. The accuracy of the predictions made by an n-gram tends to improve as the amount of historical data increases.

Game Example

For example, in a street fighting game, the player's actions (various punches and kicks) can be accumulated into a move history. Using the trigram model, the last two player moves are noted; for example, a Low Kick followed by a Low Punch. The move history is then searched for all examples where the player preformed those two moves in sequence. For each example found, the move following the Low Punch and Low Kick is tallied. The statistics gathered might resemble Table 1.2.1.

Table 1.2.1 Statistics Gathered from Past Player Moves

Player Sequence	Occurrences	Frequency
Low Kick, Low Punch, Uppercut	10 times	50%
Low Kick, Low Punch, Low Punch	7 times	35%
Low Kick, Low Punch, Sideswipe	3 times	15%

From Table 1.2.1, the computer would predict that the player's next move will be an Uppercut (with a 50-percent likelihood based on past moves). These statistics are quickly calculated on the fly when a prediction is requested. A moving window into the past can be used so as not to consider moves that occurred too long ago.

A very similar technique is to analyze the move history for the longest pattern match to the current situation [Mommersteeg02]. The assumption is that the longest match in the history is the best predictor of the future. Whether this is true depends on your application.

Further Information

AI Game Programming Wisdom: [Laramée02b, Mommersteeg02]

Neural Networks

Neural networks are complex nonlinear functions that relate one or more input variables to an output variable. They are called neural networks because internally they consist of a series of identical nonlinear processing elements (analogous to neurons) connected together in a network by weights (analogous to synapses). The form of the function that a particular neural network represents is controlled by values associated with the network's weights. Neural networks can be trained to produce a particular function by showing them examples of inputs and the outputs they should produce in response. This training process consists of optimizing the network's weight values, and several standard training algorithms are available for this purpose. Training most types of neural networks is computationally intensive, however, making neural networks generally unsuitable for in-game learning. Despite this, neural networks are extremely powerful and have found applications in the games industry.

Game Example

In games, neural networks have been used for gesture recognition in *Black & White*, steering racecars in *Colin McRae Rally 2.0*, and for control and learning in the *Creatures* series. Unfortunately, there are still relatively few applications of neural networks in games, as very few game developers are actively experimenting with them.

Further Information

Google Search: <neural networks games>
***AI Game Programming Wisdom*:** [Champandard02]
***AI Game Programming Wisdom 2*:** [Sweetser04b, Sweetser04c]
***AI Game Programming Wisdom 2* CD-ROM:** [Buckland04c, Fahey04]

Perceptrons

A *perceptron network* is a single-layer neural network, which is simpler and easier to work with than a multilayer neural network. A perceptron network is composed of multiple *perceptrons*, each of which can either have a "yes" or "no" output. In other words, each perceptron either gets stimulated enough to trigger or it does not. Since a perceptron can classify things as "yes" or "no," it can be used to learn simple Boolean decisions such as attack or don't attack. They take up very little memory and are easier to train than a multilayer neural network or a decision tree. It is important to note, however, that perceptrons and perceptron networks have some limitations and can only learn simple (linearly separable) functions.

Game Example

In the game *Black & White*, every desire of a creature was represented by a different perceptron [Evans02]. For example, a single perceptron was used to represent the desire to eat (or hunger). Using three inputs (low energy, tasty food, and unhappiness), a perceptron would determine whether a creature was hungry. If the creature ate and received either positive or negative reinforcement, the weight associated with the perceptron would be adjusted, thus facilitating learning.

Further Information

Google Search: <perceptron learning>
***AI Game Programming Wisdom*:** [Evans02]

Planning

The aim of *planning* is to find a series of actions for the AI that can change the current configuration of the game world into a target configuration. By specifying preconditions under which certain actions can be taken by the AI, and what the effects of those actions are likely to be, planning becomes a problem of searching for a sequence of actions that produces the required changes in the game world. Effective planning relies on choosing a good planning algorithm to search for the best sequence of actions, choosing an appropriate representation for the game world, and choosing an appropriate set of actions that the AI will be allowed to perform and specifying their effects.

Game Example

When the domain of a planning problem is sufficiently simple, formulating small plans is a reasonable and tractable problem that can be performed in real-time. For example, in a game, a guard might run out of ammo during in a gunfight with the player. The AI can then try to formulate a plan that will result in the player's demise given the guard's current situation. A planning module might come back with the solution of running to the light switch, turning it off to provide safety, running into the next room to gather ammo, and waiting in an ambush position. As game environments become more interactive and rich with possibilities, planning systems can help agents cope with the complexity by formulating reasonable and workable plans.

Further Information

Google Search: <AI planning>
***AI Game Programming Wisdom 2*:** [Orkin04, Wallace04]

Player Modeling

Player modeling is the technique of building a profile of a player's behavior, with the intent of adapting the game. During play, the player's profile is continuously refined by accumulating statistics related to the player's behavior. As the profile emerges, the game can adapt the AI to the particular idiosyncrasies of the player by exploiting the information stored in his or her profile.

Game Example

In an FPS, the AI might observe that the player is poor at using a certain weapon or isn't good at jumping from platform to platform. Information like this can then be used to regulate the difficulty of the game, either by exploiting any weaknesses or by shying away from those same weaknesses.

Further Information

Internet: [Beal02]
***AI Game Programming Wisdom 2*:** [Houlette04]

Production Systems

A *production system* (also known as a *rule-based system* or *expert system*) is an architecture for capturing expert knowledge in the form of rules. The system consists of a database of rules, facts, and an inference engine that determines which rules should trigger, resolving any conflicts between simultaneously triggered rules. The intelligence of a production system is embodied by the rules and conflict resolution.

Game Example

The academic community has had some success in creating bot AI for *Quake II* using the *Soar* production system [van Lent99], although the system requires upwards of 800 rules in order to play as a fairly competent opponent [Laird00]. Another applicable area is sports games where each AI player must contain a great deal of expert knowledge to play the sport correctly.

Further Information

Google Search: <"production systems" AI>
Internet: [AIISC03, Laird00, van Lent99]

Reinforcement Learning

Reinforcement learning (RL) is an extremely powerful machine learning technique that allows a computer to discover its own solutions to complex problems by trial and error. RL is particularly useful when the effects of the AI's actions in the game world are uncertain or delayed. For example, when controlling physical models like steering an airplane or racing a car, how should the controls be adjusted so that the airplane or car follows a particular path? What sequences of actions should a real-time strategy AI perform to maximize its chances of winning? By providing rewards and punishments

at the appropriate times, an RL-based AI can learn to solve a variety of difficult and complex problems.

Further Information

***AI Game Programming Wisdom 2*:** [Manslow04]

Reputation System

A *reputation system* is a way of modeling how the player's reputation in the game world develops and changes based on his or her actions. Rather than a single reputation model, each character in the game knows particular facts about the player [Alt02]. Characters learn new facts by witnessing player actions or by hearing information from others. Based on what the characters know about the player, they might act friendly toward the player or they might act hostile.

Game Example

In a cowboy gunfighter game, the player's reputation might be very important. If the player goes around killing people indiscriminately, others might witness the killings and relay the information to whomever they meet. This would give the player motivation to either play nice or to make sure there are no witnesses.

Further Information

***Massively Multiplayer Game Development*:** [Brockington03]
***AI Game Programming Wisdom*:** [Alt02]

Smart Terrain

Smart terrain is the technique of putting intelligence into inanimate objects. The result is that an agent can ask the object what it does and how to use it. For example, a smart microwave oven knows what it can accomplish (cook food) and how it should be used (open door, place food inside, close door, set cooking time, wait for beep, open door, take food out, close door). The advantage of such a system is that agents can use objects with which they were never explicitly programmed to interact.

The use of smart terrain is enlightened by *affordance theory*, which claims that objects by their very design allow for (or afford) a very specific type of interaction [Gibson87]. For example, a door on hinges that has no handles only permits opening by pushing on the non-hinged side. This is similar to letting the objects themselves dictate how they should be used.

Game Example

The term *smart terrain* was popularized by the very successful game *The Sims*.

In *The Sims,* the objects in the game world contain most of the game's intelligence. Each object broadcasts to agents what it has to offer and how it can be used. For example, an agent might be hungry, and food on the table will broadcast "I satisfy hunger." If the agent decides to use the food, the food instructs the agent how to interact with it and what the consequences are. By using this smart terrain model, agents are able to use any new object that is added into the game either through expansion packs or from Internet sites.

Further Information

Google Search: <"smart terrain" sims>
***AI Game Programming Wisdom 2* CD-ROM:** [Woodcock03]

Speech Recognition and Text-to-Speech

The technology of *speech recognition* enables a game player to speak into a microphone and have a game respond accordingly. In the games industry, there have been a few attempts to add speech recognition to games. The most notable are Sega's *Seaman* for the Sega Dreamcast and Nintendo's *Hey You, Pikachu!* for the Nintendo 64. While these first attempts were somewhat gimmicky, they serve an important role by feeling out the territory for viable speech recognition in games, both in terms of the current state of the technology and the possibilities for enhancing gameplay. Currently, any game developer can incorporate speech recognition into their PC game through Microsoft's Speech API [Matthews04a].

Text-to-speech is the technique of turning ordinary text into synthesized speech. This allows for endless amounts of speech without having to record a human actor. Unfortunately, at this point in time, virtually no games use text-to-speech technology, perhaps because it sounds rather robot-like. In practice, it's more effective to record a human voice, especially since most games have access to enough disk space to store high-quality audio samples. The quality of voice acting in games has also risen in recent years, which makes text-to-speech less appealing. However, for some games, it can be quite entertaining for the player to enter his or her name and have the game speak it. For the right game, text-to-speech can be a novel technology that can set the game apart. As with speech recognition, PC game developers can use Microsoft's Speech API to incorporate text-to-speech into their games.

Further Information

***AI Game Programming Wisdom 2*:** [Matthews04a, Matthews04b]

Weakness Modification Learning

Weakness modification learning helps prevent an AI from losing repeatedly to a human player in the same way each time. The idea is to record a key gameplay state that precedes an AI failure. When that same state is recognized in the future, the AI's behav-

ior is modified slightly so that "history does not repeat itself." By subtly disrupting the sequence of events, the AI might not win more often or act more intelligently, but at least the same failure won't happen repeatedly. An important advantage of weakness modification learning is that potentially only one example is required in order to learn [van Rijswijck03].

Game Example

Within a soccer game, if the human scores a goal against the computer, the position of the ball can be recorded at some key moment when it was on the ground before the goal was scored. Given this ball position, the game can create a gravity well vector field that will subtly draw the closest computer players toward that position. This particular vector field is then phased in whenever the ball appears near the recorded position in a similar situation (and phased out when the ball moves away). This example lends itself well to many team sports games such as soccer, basketball, hockey, and perhaps football. However, the general concept is very simple and can be applied to almost any genre.

Further Information

***AI Game Programming Wisdom* CD-ROM:** [van Rijswijck03]

Acknowledgments

Several people helped contribute to the ideas and knowledge within this article. In particular, John Manslow was a huge resource and influence. In addition, Jim Boer, Mark Brockington, Lars Lidén, Syrus Mesdaghi, Jeff Orkin, Steven Woodcock, and Eric Yiskis helped a great deal.

References

[AIISC03] "Working Group on Rule-Based Systems Report," *The 2003 AIISC Report,* AIISC, 2003, available online at *www.igda.org/ai/report-2003/aiisc_rule_based_systems_report_2003.html*
[Alt02] Alt, Greg, and King, Kristin, "A Dynamic Reputation System Based on Event Knowledge," *AI Game Programming Wisdom,* Charles River Media, 2002.
[Beal02] Beal, C., Beck, J., Westbrook, D., Atkin, M., Cohen, P., "Intelligent Modeling of the User in Interactive Entertainment," *AAAI Stanford Spring Symposium,* 2002, available online at *www-unix.oit.umass.edu/~cbeal/papers/AAAISS02Slides.pdf and www-unix.oit.umass.edu/~cbeal/papers/AAAISS02.pdf*
[Brockington03] Brockington, Mark, "Building a Reputation System: Hatred, Forgiveness and Surrender in Neverwinter Nights," *Massively Multiplayer Game Development,* Charles River Media, 2003.
[Buckland04a] Buckland, Mat, "Building Better Genetic Algorithms," *AI Game Programming Wisdom 2,* Charles River Media, 2004.
[Buckland04b] Buckland, Mat, "Genetic Algorithms in Plain English," *AI Game Programming Wisdom 2* CD-ROM, Charles River Media, 2004.

[Buckland04c] Buckland, Mat, "Neural Networks in Plain English," *AI Game Programming Wisdom 2* CD-ROM, Charles River Media, 2004.
[Champandard02] Champandard, Alex, "The Dark Art of Neural Networks," *AI Game Programming Wisdom*, Charles River Media, 2002.
[Evans02] Evans, Richard, "Varieties of Learning," *AI Game Programming Wisdom*, Charles River Media, 2002.
[Fahey04] Fahey, Colin, "Artificial Neural Networks," *AI Game Programming Wisdom 2* CD-ROM, Charles River Media, 2004.
[Fu04] Fu, Dan, and Houlette, Ryan, "Constructing a Decision Tree Based on Past Experience," *AI Game Programming Wisdom 2*, Charles River Media, 2004.
[Gibson87] Gibson, James, *The Ecological Approach to Visual Perception*, Lawrence Erlbaum Assoc., 1987.
[Houlette04] Houlette, Ryan, "Player Modeling for Adaptive Games," *AI Game Programming Wisdom 2*, Charles River Media, 2004.
[IDIS99] "Bayesian Networks," *IDIS Lab*, 1999, available online at *excalibur.brc.uconn.edu/~baynet/*
[Isla02] Isla, Damian, and Blumberg, Bruce, "Blackboard Architecture," *AI Game Programming Wisdom*, Charles River Media, 2002.
[Laird00] Laird, John, and van Lent, Michael, "Human-level AI's Killer Application: Interactive Computer Games," AAAI, 2000, available online at *ai.eecs.umich.edu/people/laird/papers/AAAI-00.pdf*
[Laramée02a] Laramée, François Dominic, "Genetic Algorithms: Evolving the Perfect Troll," *AI Game Programming Wisdom*, Charles River Media, 2002.
[Laramée02b] Laramée, François Dominic, "Using N-Gram Statistical Models to Predict Player Behavior," *AI Game Programming Wisdom*, Charles River Media, 2002.
[Laramée04] Laramée, François Dominic, "Advanced Genetic Programming: New Lessons from Biology," *AI Game Programming Wisdom 2*, Charles River Media, 2004.
[Manslow04] Manslow, John, "Using Reinforcement Learning to Solve AI Control Problems," *AI Game Programming Wisdom 2*, Charles River Media, 2004.
[Matthews04a] Matthews, James, "SAPI: An Introduction to Speech Recognition," *AI Game Programming Wisdom 2*, Charles River Media, 2004.
[Matthews04b] Matthews, James, "SAPI: Extending the Basics," *AI Game Programming Wisdom 2*, Charles River Media, 2004.
[McCuskey00] McCuskey, Mason, "Fuzzy Logic for Video Games," *Game Programming Gems*, Charles River Media, 2000.
[Mommersteeg02] Mommersteeg, Fri, "Pattern Recognition with Sequential Prediction," *AI Game Programming Wisdom*, Charles River Media, 2002.
[Orkin04] Orkin, Jeff, "Applying Goal-Oriented Action Planning to Games," *AI Game Programming Wisdom 2*, Charles River Media, 2004.
[Rabin04] Rabin, Steve, "Filtered Randomness for AI Decisions and Game Logic," *AI Game Programming Wisdom 2*, Charles River Media, 2004.
[Snavely04] Snavely, P.J., "Empowering Designers: Defining Fuzzy Logic Behavior through Excel-Based Spreadsheets," *AI Game Programming Wisdom 2*, Charles River Media, 2004.
[Sweetser04a] Sweetser, Penny, "How to Build Evolutionary Algorithms for Games," *AI Game Programming Wisdom 2*, Charles River Media, 2004.
[Sweetser04b] Sweetser, Penny, "How to Build Neural Networks for Games," *AI Game Programming Wisdom 2*, Charles River Media, 2004.

[Sweetser04c] Sweetser Penny, "Strategic Decision-Making with Neural Networks and Influence Maps," *AI Game Programming Wisdom 2*, Charles River Media, 2004.
[Thomas04] Dale, Thomas, "The Importance of Growth in Genetic Algorithms," *AI Game Programming Wisdom 2*, Charles River Media, 2004.
[Tozour02] Tozour, Paul, "Introduction to Bayesian Networks and Reasoning Under Uncertainty," *AI Game Programming Wisdom*, Charles River Media, 2002.
[van Lent99] van Lent, M., Laird, J., Buckman, J., Harford, J., Houchard, S., Steinkraus, K., Tedrake, R., "Intelligent Agents in Computer Games," *AAAI*, 1999, available online at *hebb.mit.edu/people/russt/publications/Intelligent_Agents_in_Computer_Games(AAAI99).pdf*
[van Rijswijck03] van Rijswijck, Jack, "Learning Goals in Sports Games," *Game Developers Conference Proceedings*, 2003, included on the *AI Game Programming Wisdom 2* CD-ROM.
[Wallace04] Wallace, Neil, "Hierarchical Planning in Dynamic Worlds," *AI Game Programming Wisdom 2*, Charles River Media, 2004.
[Woodcock03] Woodcock, Steven, "AI Roundtable Moderator's Report," *Game Developers Conference*, 2003, included on the *AI Game Programming Wisdom 2* CD-ROM.
[Zarozinsky02] Zarozinsky, Michael, "An Open-Source Fuzzy Logic Library," *AI Game Programming Wisdom*, Charles River Media, 2002.
[Zarozinsky04] Zarozinsky, Michael, "Free Fuzzy Logic Library," *www.loudert-hanabomb.com*, included on the *AI Game Programming Wisdom 2* CD-ROM.