
Smarter games are
making for a better user experience.
What does the future hold?

AI in Computer Games

ALEXANDER NAREYEK, GUEST RESEARCHER, CARNEGIE MELLON UNIVERSITY

If you've been following the game development scene, you've probably heard many remarks such as: "The main role of graphics in computer games will soon be over; artificial intelligence is the next big thing!" Although you should hardly buy into such statements, there is some truth in them. The quality of AI (artificial intelligence) is a high-ranking feature for game fans in making their purchase decisions and an area with incredible potential to increase players' immersion and fun.

If you've ever studied AI, however, you likely paint yourself a misleading picture of the AI methods used in games. Game AI has hardly anything to do with what is called artificial intelligence in academia. After a brief discussion of the role of AI in game development, I will provide an overview of the current state of the art, discuss the future of this game development area, and provide some links to further information.

THE ROLE OF AI DEVELOPMENT IN GAMES

Let's begin with the general set-up of AI development in games. The rampant progress of technology makes nearly every game a new beginning. Even though some basics of the game engine will probably stay the same during a game's development, constant feature and schedule revisions will make creating a subsystem such as AI something like shooting at a quickly moving target. AI is very dependent on concrete details of the game environment, which is the main reason why it's often added as one of the last subsystems. In fact, early tech demos rarely feature it.

AI in Computer Games

There are other reasons why AI is usually shifted to the end of the development process: Customers value great AI, and bad AI behavior can lead to negative reviews in the media. A game has to generate money in the end, however, and AI simply does not have the highest priority from a marketing point of view. Humans are very visual animals, and a beautiful sunset is much easier to sell than any particularly clever reasoning capabilities of an opponent.

Artificial intelligence
is very dependent on concrete details of the game environment.

In addition, early milestone demonstrations for the publisher, press presentations, and other hype-generating events do not promote inclusion of a globally/consistently good AI, but instead focus on one or two "absolutely unexpected but genius outcomes of revolutionary new and complex AI procedures" (did you spot the ironic tone?) that provide the necessary "wows." Although long-term fun with the game certainly is important as well, market mechanisms will make it very difficult for AI to receive equal ranking with features such as graphics and physics. Things might get even more difficult if the games market should finally turn into a real mass market. Markets such as persistent online game worlds, on the other hand, may increasingly promote a focus on consistently good AI because players continuously evaluate these games, gaining much more insight on mechanics, and they can continuously decide to pay or quit.

Surveys indicate that the percentage of CPU (central processing unit) cycles that developers are allowed to burn on AI computations is steadily growing. This might be because the speed of graphics cards has been increasing much faster than that of the CPU, which frees up lots of resources. Anyway, these additional resources are much needed for AI computations and open up many possibilities for more sophisticated AI.

GETTING TECHNICAL

AI techniques can be applied to a variety of tasks in modern computer games. A game using probabilistic networks to predict the player's next move in order to precompute graphics may be on a high AI level. Although AI must not always be personified, the notion of AI in computer games is primarily related to guiding nonplayer characters (NPCs).

But how does the player of a computer game perceive the intelligence of an NPC? This affects features well beyond obvious issues such as goal-related behavior. Important dimensions also include physical characteristics, language cues, and social skills. For example, a good-looking and sympathetic NPC is likely to be considered more intelligent. I will, however, focus in the following discussion on "core AI"—computing an NPC's actions. In many cases, developers also subsume collision detection under AI. (In my opinion, this is the responsibility of the physics engine, so I will not cover that topic here.)

I should mention that the goal in game AI is not to compute the most optimal behavior for winning against the player. Instead, the outcome should be as believable and fun as possible. Measures such as cheating are absolutely acceptable as long as the "suspension of disbelief" is retained. It does not really matter if real insight is behind the characters' actions. In many cases, too much autonomy and unpredictability are, in fact, undesirable: Who guarantees that the result is enjoyable? And you will most likely have a hard time selling highly unpredictable outcomes to your quality assurance (QA) department.

Movement: Pathfinding and Steering. NPCs have to move through the environment in an intelligent way—that is, not getting stuck by trees in their way, taking a possibly short route to the destination, and so forth. This

is one of the basics of game AI, and you would expect that this be properly solved in today's games. Not quite so, however. Though the major part of AI-development resources go into this area, its lack of quality is one of the top complaints. You might blame this on game developers assigned to this task who lack sufficient knowledge about AI, but the main reason is this: Given the highly limited computational resources, a sophisticated movement is a really hard thing to do! Add to this features such as complex environments with complicated terrain, dozens or hundreds of units for which this has to be computed in parallel, dynamically transformable terrain, and so on.

The so-called A* algorithm is the most common basic ingredient for computing a long-distance route for an NPC. The most suitable variant of this approach depends very much on the specifics of the game environment. The gaming literature is full of articles on this topic, and it is sometimes hard to maintain a perspective. Besides the general approach, it is also important to pay attention to implementation details, such as clever memory management. I will describe the basics of the A* algorithm here to give you an idea of the general approach.

The algorithm requires a definition of waypoints and their connections for a specific map/environment. For example, two rooms can have waypoints in their respective middles, and these waypoints are connected because it is easy to reach one room from the other via a passage. The waypoints with their connections span a net over the map, defining which regions/points can be directly reached from other regions/points. Given a starting point and a destination, the A* algorithm tries to find the shortest path along the waypoint connections. It stepwise explores the waypoints in increasing distance from the starting point along the possible connections until the destination waypoint is reached. The algorithm uses an estimation component, which has to provide an estimate for the distance between a point and the destination point. Thereby, the algorithm can focus its expansion of a possible path on the most promising connections.

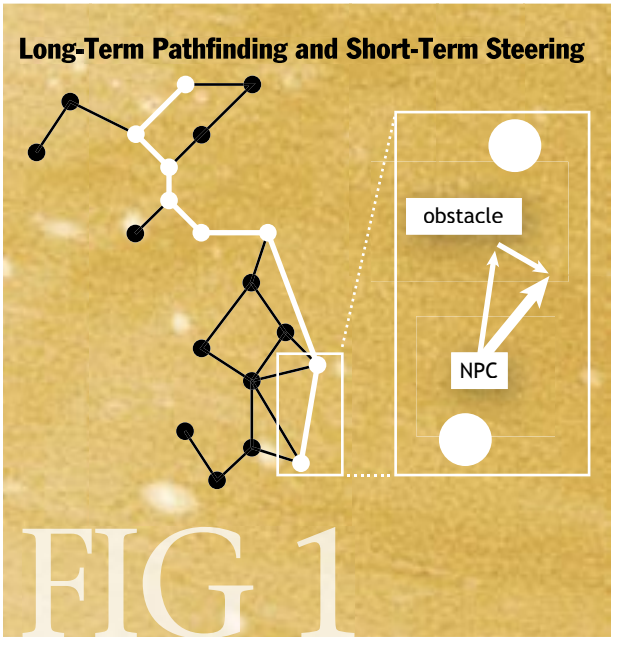
In many cases, a game applies pathfinding techniques at multiple granularity levels. For example, for long distances, a path of high granularity is computed first, and then the paths between the selected waypoints are computed with finer granularity. You can probably imagine how complicated things get with dynamically changeable terrain and so on.

To maneuver between connected waypoints, the game applies so-called steering methods. Obstacle avoidance, coordinated formation movement with team/group units, etc. are handled at this level (see figure 1).

Steering methods do not strive for a globally optimal behavior but compute an NPC's movements from a very limited perspective. In most cases, a vector with direction and speed/force is computed for each movement-relevant aspect, and these vectors are then combined to a single final vector. For example, one vector is directed toward the NPC's next waypoint, and an additional vector for each nearby obstacle along the way points orthogonally away from this obstacle. The vectors could be combined by a simple addition of all vectors, which produces a result vector that's then interpreted as acceleration and turn preferences. This is a simple example, but may give you an idea of how the process works.

Team and formation movement can be incorporated in a similar way. For example, a general movement vector per team is introduced, which is combined into each team member's vector set, as well as a vector for each member that points toward the preferred position within the team. In many games and movies, flocking birds or fishes in the background are also realized by techniques such as these.

Finite State Machines and Decision Trees. FSMs (finite state machines) describe under which events/conditions a current state is to be replaced by another—for example, switching from an attack mode to an escape mode if the NPC is hit. It is mostly only a design concept—that is, the game has no general FSM interpreter, but the FSMs are realized by scripts and simple if-then statements.



AI in Computer Games

Figure 2 shows an example of an FSM. The boxes represent states, which involve specific scripts, animation schemes, etc. The starting state is "searching for player." The arrows show conditions under which the state of the NPC is changed, such as an incoming event when the player is sighted.

An FSM is a simple and powerful tool for modeling an NPC's behavior. There are extensions to cope with more complex behaviors, such as hierarchical FSMs, as well as nondeterministic variants to introduce random elements.

Decision trees conceptually are even slightly simpler than FSMs and represent branching structures that are often used to make high-level strategic decisions–for example, if a computer-guided opponent in a strategy game should prepare an attack or concentrate on resource gathering. The nodes in the tree are test conditions, which lead to different sub-trees. A final leaf node contains a resulting decision. Similar to FSMs, decision trees are conceptual tools and can be realized by simple if-then statements.

Other Approaches and AI Research. Many other techniques are applied to realize game AI. These include influence mapping, which is a technique for terrain analysis to identify boundaries of control or otherwise interesting points/areas/features of a map; and level-of-detail approaches for AI, which deal with the fact that there's not enough time available to compute AI details for every NPC and might, for example, assign time for rougher reasoning only for NPCs that are far away from the player.

Game AI spans a large array of tasks, and it is not easy to generalize the various approaches. They are most often highly tailored to the specific games and situations—for example, an algorithm to determine from which direction an enemy settlement should get attacked in the game Age of Mythology (Microsoft), or how a Counter-Strike (Microsoft) bot realistically aims when throwing a grenade. Most games feature powerful scripting languages

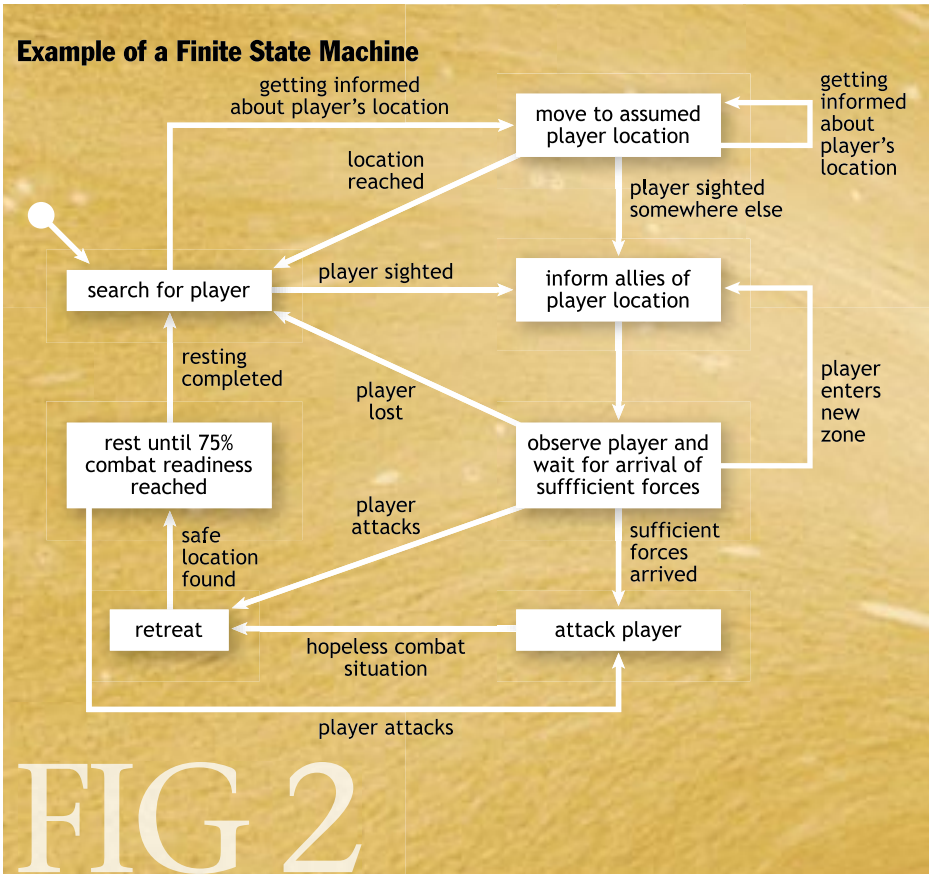


FIG 2

A*: The Tried and Tested Solution for Pathfinding

A* is an improved version of Dijkstra's shortest-path algorithm.[1,2] Though it can be used for a range of search problems, its primary application area is pathfinding. For those of you unfamiliar with the A* algorithm, here is a more detailed explanation.

As explained in the accompanying article, the map is represented by a set of location nodes/waypoints, some of which have connections of certain distances. Given a start node and a destination node, the algorithm has to find the shortest path between those two points.

The algorithm searches stepwise from the start node toward the destination. The algorithm maintains two lists: **open** and **closed**. The **open** list contains all new nodes that could be visited in the next step from the already-visited nodes. The **closed** list contains all nodes that were already visited. The **open** list is initialized with the start node. The algorithm has found the shortest path once the destination node is added to the **open** list. The **closed** list starts empty.

The nodes of the open list are ranked according to the formula f(n) = g(n) + h(n) where g(n) is the shortest distance along the already-visited connections from the start node to node n; and h(n) is an estimate of the remaining distance from node n to the destination node. It is important that the estimate is lower than or equal to the actual shortest distance along possible connections.

In each step of the algorithm, the node with the smallest f(n) is selected from the **open** list and moved to the **closed** list. All nodes that can be reached by a direct connection from the selected node and are not in the closed list—that is, they have not been visited before—are processed in the following way: If the node is not already in the **open** list, it is put there. If it is already in the **open** list, its f(n) needs to be recalculated. Figure 1 visualizes one step of the algorithm.

After the destination node has been reached, the actual path can be computed backward from the destination. To know which predecessor nodes to select from the **closed** list for chaining back to the start, a parent node is remembered for each visited node during the search for the destination. The parent of a node is the one that is selected when the node is added to the **open** list (or the node is already in the **open** list and a recalculation of f(n) yields a smaller value than before).

A nice tutorial with more details can be found at the Almanac of Policy Issues Web site.[3]

REFERENCES

1. Hart, P. E., Nilsson, N. J., and Raphael, B. A formal basis for the heuristic determination of minimum-cost paths. *IEEE Transactions on Systems Science and Cybernetics* 4, 2 (1968), 100–107.
2. Dijkstra, E. W. 1959. A note on two problems in connection with graphs. *Numerische Mathematik* 1 (1959), 269–271.
3. Lester, P. A* Pathfinding for Beginners. Almanac of Policy Issues: see http://www.policyalmanac.org/games/aStarTutorial.htm.

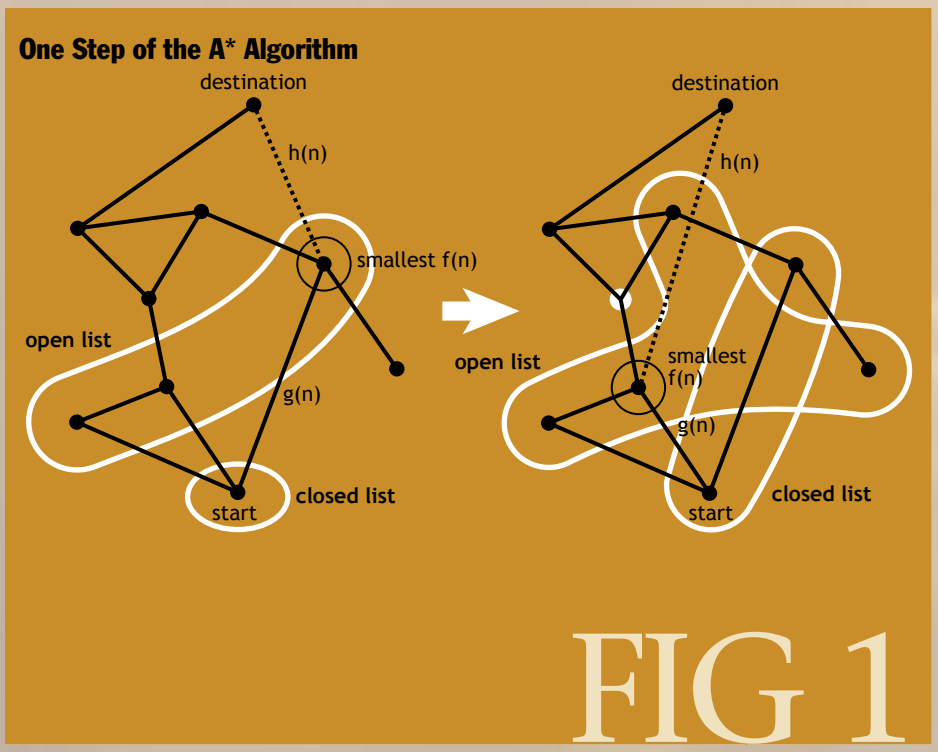


FIG 1

AI in Computer Games

that can produce individual AI behaviors also at a higher level, and some games even make them available to the players who can thereby rewrite parts of the AI. (For more on this subject, see Phelps and Parks' "Fun and Games with Multi-Language Development" on page 46 of this issue.)

I will not describe the previously mentioned approaches in any more detail and instead recommend the series *AI Game Programming Wisdom*[1, 2] and *Game Programming Gems*[3, 4] for more comprehensive coverage.

AI approaches from academia, such as genetic algorithms or neural networks, are hardly ever used in game development. Such approaches are generally believed to require too many resources for development, tuning, and testing. The general approaches must be heavily modified and specialized for a specific game to get anything acceptable; analyzing and understanding the reasons for the resulting behavior is complicated; they are hard to test thoroughly; and modifying them toward more enjoyable behavior is everything but easy as well. So far, very few games use academia-inspired technology, such as Creatures (CyberLife Technologies, 2001) or Black and White (Electronic Arts, 2001).

Unfortunately, AI research often focuses in a direction that is less useful for games. A* is the most successful technique that AI research has come up with—and nearly the only one applied in computer games. The research community is nearly exclusively concerned with tuning its approaches for computational efficiency and does not care about features such as dynamics, realtime, and software-engineering-related properties.

Bridging the gap between academic AI research and its distant cousin in the gaming world presents many challenges. The research domain continues to have reservations with respect to computer games as an application area, but, hopefully, the growing economic importance of the computer gaming field will continue to weaken those reservations. Games are slowly gaining respect in academics, and there are research groups being established now (including my own) with more viable approaches that focus on features that are more relevant in practice.

AI Integration. Central to the AI computation is not only how actions are determined, but also which information about the environment is available and how this can be accessed. Accessing coordinates of pathfinding waypoints may not be highly problematic, but in many cases there are complex interactions with the game world/engine. For example, if an NPC's FSM needs to react with a transition to an event that the player gets into the line of sight, will this line-of-sight condition be queried each AI cycle by the FSM code, or will a specific event callback be triggered by the game world? Can answering multiples of such AI queries be delayed by the game world and executed together to optimize processing times? Does the AI part have its own memory, and, thus, do changes only in the sensed data need to be provided? Such questions must be answered while designing a game AI, and the appropriate answers may even vary for its different sub-parts.

A problematic issue concerning AI integration is that the kinds of interfaces used to and from AI components are different from game to game, so far. The Artificial Intelligence Interface Standards Workgroup was recently formed to develop interface standards for basic game AI functionality such as pathfinding, steering, finite state machines, rule-based systems, goal-oriented action planning, and world interfacing. Hopefully, standardized interfaces will promote the reuse of AI components, unburdening game AI developers from worrying about low-level procedures and enabling them to focus on higher-level creative AI tasks.

FUTURE DIRECTIONS

With increasing CPU power available for AI computations, NPC behavior will become more and more sophisticated. Cheating, which is very annoying for the player if discovered, can be reduced, and NPC behavior gets much more believable. Advanced techniques will stepwise be introduced into games, such as goal-oriented action planning, which is already starting to make an appearance

in games that are coming out soon (even though in very simple forms).

The increasing complexity of AI technology will make it necessary to incorporate third-party middleware. Some companies already offer packages, but with limited success until now. Among the reasons for that is the lack of standard interfaces for basic AI routines, such as DirectX or OpenGL in the graphics area. This should soon change as the AI Interface Standards Workgroup begins to develop such interface standards.

Besides the technological challenges, however, we need to see more effort to make AI functionality available for the designers/artists. They often lack programming skills and need appropriate high-level tools to shape and control AI behaviors.

Looking further into the future, AI will be focused not on optimizing an NPC's behavior, but on the player's fun and experience in general. This reaches far beyond the guidance of single NPCs into learning what is fun for the player and shaping/changing the game experience accordingly—for example, whole cities and civilizations being simulated in a believable way, deep NPC characters, automated storytelling with dynamic tension and emotion planning for the player, and so forth.

It seems to be a fantastic perspective for AI in games. Just don't get carried away too much and assume that it will be the one and only determining factor for future games. Games feature many great technology directions, and AI is only one of them—of course, it is the most fascinating! Q

LOVE IT, HATE IT? LET US KNOW

feedback@acmqueue.com or www.acmqueue.com/forums

REFERENCES

1. Rabin, S., ed. *AI Game Programming Wisdom, Vol. 1*, 2002. (Hingham, MA: Charles River Media).
2. Rabin, S., ed. *AI Game Programming Wisdom, Vol. 2*, 2003. (Hingham, MA: Charles River Media).
3. Treglia, D., ed. *Game Programming Gems, Vols. 1–3*, 2000-2002 (Hingham, MA: Charles River Media).
4. Treglia, D., ed. *Game Programming Gems, Vol. 4*, 2004 (Hingham, MA: Charles River Media).

ALEXANDER NAREYEK received his Ph.D. from the TU Berlin. Since 2002, he has been on an Emmy Noether fellowship of the German Research Foundation (DFG) and is a guest researcher at Carnegie Mellon University. His main research interests include the generation and execution of behavior plans for goal-driven intelligent agents. He is active in the application area of computer games and serves as chairperson of the International Game Developer Association's (IGDA) Artificial Intelligence Interface Standards Committee (AIISC; http://www.ai-center.com/home/alex/).

AI in Games

Artificial Intelligence Interface Standards Workgroup of the International Game Developers Association
http://www.igda.org/ai/

Steven Woodcock's Game AI Resources
http://www.gameai.com

Game AI Articles and Research
http://www.aiwisdom.com

Amit Patel's Game Programming and A* Information
http://www-cs-students.stanford.edu/~amitp/gameprog.html#paths

Craig Reynolds' Resources on Steering
http://www.red3d.com/cwr/steer/

The EXCALIBUR Project (goal-directed action planning)
http://www.ai-center.com/projects/excalibur/

"Computer Games—Boon or Bane for AI Research?"
(An article about whether AI research makes relevant contributions)
http://www.ai-center.com/references/nareyek-04-gameairesearch.html

RESOURCES