Automata and NPC control in video games

Denis Kuperberg, CNRS, LIP, ENS Lyon



1. INTRODUCTION

The goal of this paper is to give a window into some real-world application of automata, and to wonder how our beloved models are actually used in some specific contexts. I took the example of how Non-Player Characters (NPCs) are controlled in video games. To some extent, this also applies to the field of robotics.

When designing and studying automata, we are interested in properties such as expressivity of various formalisms, and performing some operations at minimum cost. We will investigate here what properties are prioritized for models describing NPC behaviors. The reader must keep in mind that the role of automata here will be different from the one they play in formal language theory. Instead of being used to represent regular languages, they will serve as abstractions of program architectures.

This article will mainly be a very partial history of how this field evolved in the last few decades, and how practical considerations affected design choices, stabilizing into specific architectures that are widely used today. Pointers to more extensive literature will be provided as we go.

Although the field was historically named "Artificial Intelligence", it is almost entirely disconnected from the current use of the term, as we will see in Section 4.

2. AUTOMATA FOR BEHAVIOR DESCRIPTION

2.1. What are NPCs

Let me first describe the problem at hand, in particular for those not familiar with electronic gaming. In a variety of video games, especially those where you control a character evolving in a populated world, you will encounter characters that you may interact with in various manners (fighting, conversations, trade...). The generic name for these characters is "Non-Player Characters" (NPCs), by contrast with your character or those controlled by other humans in multiplayer online games. This term of NPC encompasses enemies in shooting games, peaceful villagers in role-playing games, opponents in racing games, etc.

Since the beginning[1] of commercial video games in the 1970s, a major challenge of the industry has been to make these NPCs as realistic as possible, in order to provide immersion and challenge to the player.

[1]The first commercial video game is Computer Space (1971).

Let us see how automata offer a natural way to model NPC behaviors, and how they were actually used in early examples.

2.2. Pac-Man

Although not the first appearance of NPCs, the 1980 Pac-Man game is an emblematic example that will likely sound familiar to most readers.

In this game, as described in [Pittman 2009], you control Pac-Man, a round yellow character that must eat dots in a maze, while avoiding four ghosts chasing him. The ghosts alternate between a Scatter behavior where they randomly explore the maze, and a Chase behavior where they actively move towards you[2]. This alternation is predefined, with fixed time intervals for each behavior. When Pac-Man eats one of the bonuses present on the level, he becomes able to eat the ghosts for a fixed period of time.

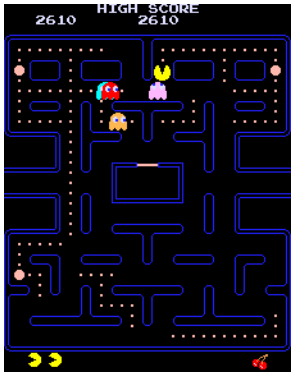


Fig. 1: Pac-Man screenshot

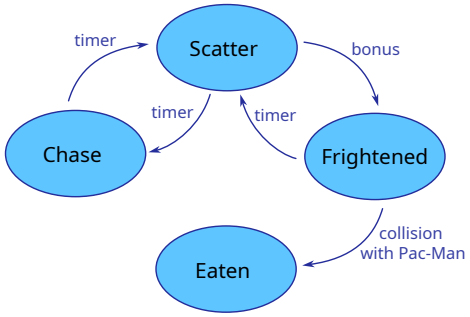


Fig. 2: Ghost automaton

We can thus represent the behavior of each ghost with the simple 4-state automaton from Figure 2, where each state is a possible behavior.

We see how natural the automaton model is in this case. Since we can consider that the automaton also outputs the wanted behavior, we can rather consider it a Mealy Machine. To comply with the terminology used in the industry, let us stick to the more agnostic term of Finite State Machine (FSM).

2.3. Next generation FSMs: Doom and Goldeneye

Let us advance a bit to two other emblematic examples: Doom (1993) and Goldeneye 007 (1997). Both games are First-Person-Shooters: they let you adopt the point of view of your character in a 3D environment, and the primary goal is to kill enemies, usually with various types of guns.

Both games became landmarks for what can be achieved with FSM-controlled NPCs, and were acclaimed for the immersive behaviors of the enemies. The goal of this section is not to give a thorough description of the details, but rather to give a feel of the complexity of NPC behaviors in these games. If you wish to know more about the specifics, see [Thompson 2022; Thompson 2019].

Let me reproduce the FSM used in Doom, taken from [Thompson 2022]:

[2]Each ghost has a specific way of doing that, but these details are not necessary for the current exposition.



Fig. 3: Doom (1993)



Fig. 4: Goldeneye 007 (1997)

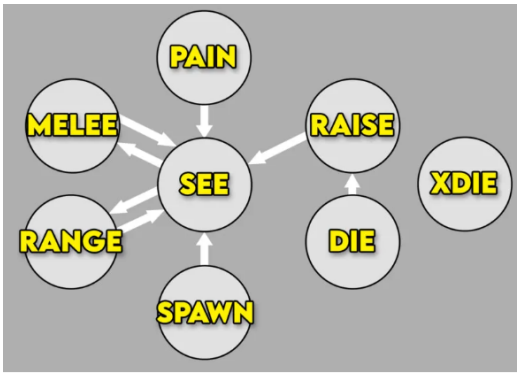


Fig. 5: Doom FSM

We will not expand here on the behaviors corresponding to these states, the idea is just to give a feeling of the shape of the FSM. Of course, as in Pac-Man, this FSM alone does not contain all the information about NPC behavior, as each state actually contains a specific subroutine, and transitions between states are triggered by various events. We can for example see that the state XDIE, representing a particular way for the NPC to die, appears to be disconnected from the rest. This is because this state is entered upon specific external conditions, which are independent of the current state and usual triggers. This FSM is thus only a rough abstraction of the actual specification.

Remark that these FSMs are not explicitly encoded as data structures in the game programs, they rather describe the structure of function calls. In Goldeneye 007, a scripting language with custom high-level commands was created for this purpose [Knorr 2022], including functions such as:

```
aiKneel()
aiLookAtBond()          // Look towards Bond then return to stand
aiSurprisedByBond()     // Act surprised by seeing Bond
aiOpenDoor(special)     // Open the door (even if locked)
```

The scripting language combines roughly 250 custom commands [kholdfuzion 2026], giving rise to elaborate behaviors. These commands include actions (`Kneel`, `PointAtBond`, `LookSurprised`, `Sidestep`, ...) as well as triggers (`IFISeeBond`, `IFIHeardBond`, `IFISeeSomeoneShot`, `IF_GAS_IS_LEAKING`, ...).

Thus, in this context, FSMs are essentially the relevant abstraction of GoTo programs. And as GoTo programs in general, the industry realized that it was not an ideal architecture.

Although games such as Doom and Goldeneye set a new standard for NPC behavior, the industry soon began to explore new paradigms that were better suited to the task.

3. THE NEW STANDARD: BEHAVIOR TREES

In order to address the specific needs of NPC behavior design, and later robotics, the structure of Behavior Trees (BTs) started to emerge in the early 2000's [Isla 2005], and progressively became the new standard [Colledanchise and Ögren 2018].

We will start by defining formally these behavior trees, and then explore the reasons why they advantageously replaced FSMs in this context.

The book "Behavior Trees in Robotics and AI: An Introduction" [Colledanchise and Ögren 2018] provides a detailed exposition of BTs, destined to a wide audience.

3.1. What is a Behavior Tree?

This is only intended as brief introduction to the model. For a more thorough account of the history of use of BTs, the interested reader is encouraged to consult the survey [Iovino et al. 2022] or the book [Colledanchise and Ögren 2018].

A behavior tree (BT) is a tree with the following types of nodes:

◇ Condition nodes, labelled with events that can be detected.
⬭ Action nodes, labelled with instructions that can be sent to the agent.
▭ Control nodes that come in two kinds: Fallback (**?**) and Sequence (→).

Only control nodes can have children, conditions and actions are leaves of the tree. The children of a control node are ordered from left to right.

Here is a simple example of a BT describing how an academic researcher might go about his day:

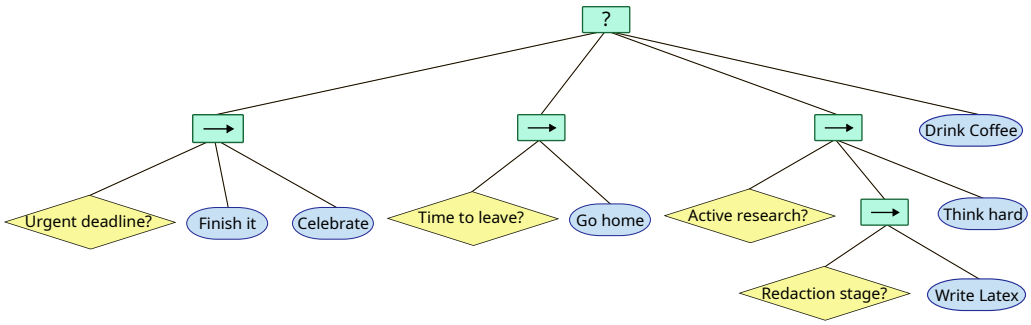


Fig. 6: A toy BT example

Each node can be executed, and may return three different reports: Running, Success, Failure. The report is sent to the parent node.

When a node is executed (this is often phrased as receiving a "tick"), the following is performed, depending on the nature of the node. In all cases, Running is returned while waiting for Success or Failure to be returned.

Condition: The condition label is checked, and Success or Failure is returned depending on the truth value of the condition.

Action: The action label is performed, and Success or Failure is returned depending on whether the action was successful. This is assumed to be part of the interface of possible actions.

Fallback (**?**): Children from left to right are executed while they return Failure, and the node returns Success when one of the children has returned Success, or Failure after the last child failed.

Sequence (→): Children from left to right are executed while they return Success, and the node returns Failure when one of the children has returned Failure, or Success after the last child succeeded.

For simplicity of the exposition, parallel nodes were omitted here. These nodes are able to execute all children in parallel, returning Success when enough of them succeeded.

3.2. Advantages

Let us now describe the many advantages of BTs compared to the FSMs of the previous section.

First of all, the fixed interface Running/Success/Failure allows for modularity. Each action node can be replaced with a more detailed BT refining the behavior, without having to touch the outer structure. For instance the node "Think hard" from Figure 6 could be replaced by Fallback(Try option 1, Try option 2, Try option 3). The tree structure therefore naturally allows for splitting a task into subtasks, with a level of detail that can be later adjusted if needed.

Moreover, the type of nodes (Condition, Sequence, Fallback) carry very natural ideas that will be present in almost any high-level behavior description. In particular this allows to natively implement priorisation of tasks: the more to the left a child is, the higher its priority.

The model also allows easy maintainability and updating. Consider a new feature added to an NPC, such as being able to help another neighbouring NPC. This new behavior can simply be added at the wanted level of priority as a child of a Fallback or Sequence node, while in a FSM, you would need to add edges from many unrelated nodes to implement a specific priority level, and it would be very hard to make sense of the resulting structure.

Coming back to the analogy of FSMs as GoTo programs, the Fallback and Sequence nodes of BTs can be compared to replacing arbitrary GoTo statements with `for` and `while` loops, greatly helping the intelligibility of programs. In another direction, the modular aspect of BTs with a clear interface (Running/Success/Failure) gives them an Oriented-Object flavour. We can therefore observe that the innovations of BTs share similarities with the paradigm shifts that occurred in programming languages.

We can however notice that even though FSMs were not retained as a model of program architecture for NPCs, techniques stemming from automata theory can still be used to analyze BTs, for instance [Biggar and Zamani 2020] uses Linear Temporal Logic to verify properties of BTs.

3.3. Examples from the industry

Interestingly, although there is a consensus that BTs are now the standard for NPC control, it is very difficult to find a publicly available example to illustrate this, i.e. a BT that was used in a commercial game. Even when BTs are shown in game developer conferences or papers, such as in [De Pascale and Vehkala 2013] (Figure 7), they are usually toy BTs whose purpose is to illustrate principles, rather than revealing actual implementations. This is probably mainly due to the proprietary nature of the game industry.

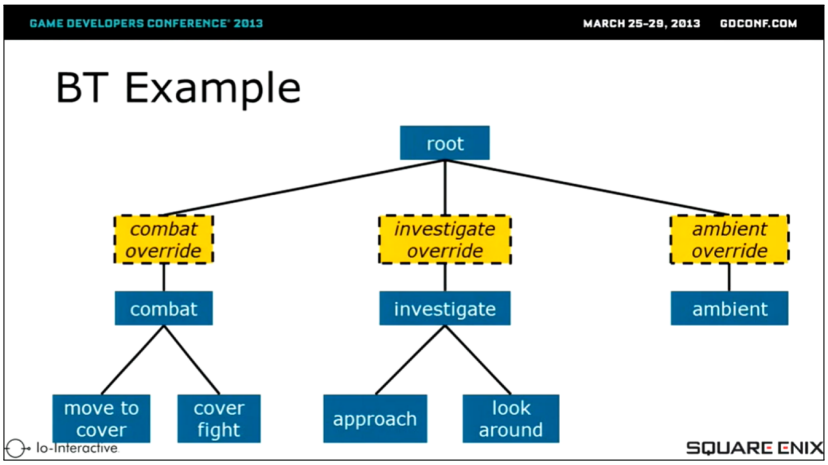


Fig. 7: Toy BT in a talk about the game Hitman: Absolution (2012)

3.4. Expressivity

The initial idea for this article was to compare formally the expressivity of this model with usual automata. Indeed, at first glance, it seems that BTs are losing some expressivity. For instance it looks like asking the NPC to oscillate between two behaviors is not possible.

This is indeed the case if we take BTs as a rigid model with no interactions with an outside environment except receiving the truth values of conditional tests and performing actions that have no impact on the future value of such tests. However, as soon as this last statement is violated, and an action can e.g. write on a variable that will be read in a condition, it is trivial to recover the expressivity of FSMs. Comparing the expressivity of FSMs, BTs, and other classical frameworks such as decision trees and teleoreactive programs (particular cases of BTs, introduced in [Nilsson 1994]) was done e.g. in [Biggar et al. 2021], but such comparisons amount to trivial reductions, that do not convey the relative merits of each approach. Notice that in the formalism chosen in [Biggar et al. 2021], "pure BT" with no external memory have the same expressivity as so-called "Reactive FSM", whose output only depends on the last letter of the input word.

4. WHY NOT NEURAL NETWORKS?

It might be puzzling why BTs are still very popular for NPC control today, in the age of neural networks capable of handling nearly any task thrown at them. To understand this, it is important to realize what problem we are trying to solve when designing such systems [Michael 2026].

4.1. Clarifying our goal

It would be natural to think that we want to optimize the performance of an NPC at a specific task: being a worthy fighting opponent, finding the optimal route from A to B, reacting efficiently when detecting an intruder, etc. But the ultimate goal that game developers are aiming at is not how good the NPCs are at such tasks, but how enjoyable it is for the player to play the game. We can anticipate that playing against "optimal" opponents that have superhuman capabilities would not constitute a pleasant game experience. Instead, the main quality that we look for in NPCs is their intelligibility. Their behavior must appear human-like, the reasons behind NPC actions should be easily interpretable, for different reasons:

— It contributes to the immersive feeling of the game.
— It allows the player to anticipate behaviors and plan strategies.
— It permits better control of the difficulty, by switching on or off certain elaborate behaviors.

In most cases, behavior trees are better suited to these goals than machine learning. By their nature, BTs resemble how human take decisions, and so they will lead to easily interpretable behaviors, while NPCs controlled by neural networks could exhibit mysterious behaviors that optimize the task at hand but fail to provide the user with a good game experience.

4.2. Exceptions: use of neural networks for NPCs

Some exceptions can however occur, e.g. in a racing game, minimizing the time to complete the race can be considered the only aspect of the game, both for the player and the NPC opponents, so it might be suitable to use machine learning with varying parameters to build opponents at all difficulty levels, and in particular to oppose champion players in competitive settings. This has been done for instance for the racing game Gran Turismo [Wurman et al. 2022].

Another instance where machine learning algorithms, in particular LLMs, have found a use is when NPCs are allowed to dialog freely with the player, without much of a relevance to the scenario. On the contrary, when dialog is important to the scenario, it must usually be scripted.

Some games also gave center stage to machine learning technologies, for example:

— Creatures (1996) is an interesting experiment where the learning ability of NPCs via basic neural networks is the main gameplay mechanism. It is credited as the first application of machine learning in an interactive simulation.
— AI Dungeon (2019) took advantage of the first generation of LLMs, here GPT-2, to propose a game primarily based on dialog.

Finally, let us mention that while a large part of this exposition applies as well to the field of robotics, this last section is more focused on the video game side. Indeed, the goals and challenges in robotics have allowed for larger adoption of machine learning techniques [Tang et al. 2025].

ACKNOWLEDGMENTS

Thanks to David Louapre who triggered my interest in the subject via a Youtube video [Louapre 2026], and to first readers Amina Doumane, Simon Iosti, Hugo Maussion, Léo Bendeks for their feedback. LLMs were used only for bibliographic search and basic Latex facilitation.

REFERENCES

Oliver Biggar and Mohammad Zamani. 2020. A Framework for Formal Verification of Behavior Trees With Linear Temporal Logic. *IEEE Robotics and Automation Letters* 5, 2 (2020), 2341–2348. DOI:http://dx.doi.org/10.1109/LRA.2020.2970634

Oliver Biggar, Mohammad Zamani, and Iman Shames. 2021. An Expressiveness Hierarchy of Behavior Trees and Related Architectures. *IEEE Robotics and Automation Letters* 6 (04 2021), 5397–5404. DOI:http://dx.doi.org/10.1109/LRA.2021.3074337

Michele Colledanchise and Petter Ögren. 2018. *Behavior Trees in Robotics and AI: An Introduction*. CRC Press, Boca Raton, FL. https://arxiv.org/abs/1709.00084

Maurizio De Pascale and Mika Vehkala. 2013. Creating the AI for the Living, Breathing World of Hitman: Absolution. (2013). https://gdcvault.com/play/1017802/Creating-the-AI-for-the, 19:36.

Matteo Iovino, Edvards Scukins, Jonathan Styrud, Petter Ögren, and Christian Smith. 2022. A Survey of Behavior Trees in Robotics and AI. *Robotics and Autonomous Systems* 154 (2022), 104096. DOI:http://dx.doi.org/10.1016/j.robot.2022.104096

Damian Isla. 2005. Handling Complexity in the Halo 2 AI. Game Developers Conference presentation and proceedings article. (2005). https://www.gamedeveloper.com/programming/gdc-2005-proceeding-handling-complexity-in-the-i-halo-2-i-ai

GitHub user kholdfuzion. 2026. Goldeneye 007 decompilation. https://github.com/n64decomp/007/blob/master/src/aicommands.def. (2026).

Alyse Knorr. 2022. Book Excerpt: Anti-Game Design and the Making of Goldeneye 007. Ars Technica article. (2022). https://arstechnica.com/gaming/2022/07/book-excerpt-anti-game-design-and-the-making-of-goldeneye-007/

David Louapre. 2026. L'intelligence artificielle dans les jeux vidéo. YouTube channel Science Étonnante. (2026). https://www.youtube.com/watch?v=svkQLQTrU8E.

Benjamin Michael. 2026. Game AI is not about Intelligence. Medium article. (2026). https://medium.com/@armoury_ale_0s/game-ai-is-not-about-intelligence-9fa6c563fd6a

Nils J. Nilsson. 1994. Teleo-Reactive Programs for Agent Control. *Journal of Artificial Intelligence Research* 1 (1994), 139–158. DOI:http://dx.doi.org/10.1613/jair.30

Jamey Pittman. 2009. The Pac-Man Dossier. Game Developers article. (2009). https://www.gamedeveloper.com/design/the-pac-man-dossier

Chen Tang, Ben Abbatematteo, Jiaheng Hu, Rohan Chandra, Roberto Martín-Martín, and Peter Stone. 2025. Deep Reinforcement Learning for Robotics: A Survey of Real-World Successes. *Annual Review of Control, Robotics, and Autonomous Systems* 8, Volume 8, 2025 (2025), 153–188. DOI:http://dx.doi.org/https://doi.org/10.1146/annurev-control-030323-022510

Tommy Thompson. 2019. The AI of Goldeneye 007 (1997). Game Developers article. (2019). https://www.gamedeveloper.com/design/the-ai-of-goldeneye-007

Tommy Thompson. 2022. The AI of Doom (1993). Game Developers article. (2022). https://www.gamedeveloper.com/game-platforms/the-ai-of-doom-1993

P.R. Wurman, S. Barrett, K. Kawamoto, and others. 2022. Outracing champion Gran Turismo drivers with deep reinforcement learning. *Nature* 602 (2022), 223–228. DOI:http://dx.doi.org/10.1038/s41586-021-04357-7