AI Architecture of Non-Player-Characters in Games: A comparative view from Rule-Based to Utility Based Approaches.

Hossam Khalifa
Gina Cody School of Engineering and Computer Science
Concordia University

CONTENTS

I Introduction 1

II NPC AI Architectures 2
II-A Rule-Based Systems . . . 2
II-B Finite State Machines (FSM) . . . 2
II-C Behavior Trees . . . 3
II-D Utility-Based . . . 4

III Layered Supportive Elements 5
III-A Adaptive/Learning . . . 5
III-B Advantaged . . . 6

IV Comparative Analysis 6
IV-A Benefits and downsides . . . 6
IV-A1 Rule-based systems . . . 6
IV-A2 Finite State Machines . . . 7
IV-A3 Behavior Trees . . . 7
IV-A4 Utility Based AI . . . 7
IV-B Ranking . . . 7

V Genre Specific Deployment 8

VI Conclusion 8

References 9

LIST OF FIGURES

1 Rule Based Systems implementation Graph visualization . . . 2
2 FSM visualization for AOE2 unit states . . . 3
3 Behavior Tree simple example . . . 4
4 UtilityBased visualization for decision making process . . . 5
5 Adaptive Layer visualization atop of existing unit FSM example . . . 6

AI Architecture of Non-Player-Characters in Games: A comparative view from Rule-Based to Utility Based Approaches.

***Abstract*—This report explores basic implementations of NPC AI architectures to provide a comparative analysis for which of them provide the best general use deployment. The evaluation criteria used consider both developer costs and challenges in implementation as well as player expectations in NPC behaviors to accurately determine which architecture is most useful for general use. Additionally, using the highlighted and explored complexities, advantages, and disadvantages of each method, a genre specific deployment is identified for each to maximize their potential.**

Utility based AI ranked overall as the most suited for general purpose deployment or general use. This was due to its highly extensible and modular design that allows easy updates over the course of development. Additionally, utility based AI offers the option of adding non-determinsim for unpredictable behaviors while maintaining relatively realistic behavior through the use of the utlity scores. While utility based AI has its weaknesses, it was the best option for general use.

I. Introduction

Non-Player-Characters (NPCs) in games have a wide variety of definitions depending on the context. Some of the public often incorrectly assume that NPCs refer exclusively to roaming characters in open-world games that are not controlled by the player. However, in this report, NPCs are defined more broadly. NPCs discussed here include all the AI opponents in games, roaming characters within worlds, and even occasionally units of the player in some games. The AI architecture refers to all systems that facilitate decision-making not explicitly provided by a human player. It should also be noted that the definition of AI here is not the standard understanding for normal cases. The AI architecture here is the mechanism by which decisions are made within the games and not exactly neural networks or deep learning cases, although those may fall within this broader definition.

This AI definition notably does not include the pathfinding algorithms each NPC uses such as well-known methods like A* or Dijkstra's algorithm. While these algorithms play a critical role in NPC movement, they address a functionally different problem, specifically determining optimal paths. As such, for this report where the focus is high-level decision-making behavior, pathfinding is excluded as it is considered out of scope.

Given these stated definitions, it is easy to see why NPC AI architecture matters greatly in game development. NPCs can range from what amounts to decorative props in the world with extremely limited interactions to detailed and immersive interactive elements [1]. These differences affect the game design through difficulty balancing, where artificial difficulty resulting from poor design can cause players to lose immersion. Another element deeply affected by NPC AI is game realism. In 1955, Johan Huizinga used the term "Magic Circle", later used by many game scholars, to describe the confined world of the game. Huizinga described games as occurring in a separate temporary world and in turn states that "the least deviation from it "spoils the game"" [2]. This is an accurate way of describing the importance of NPC behavior for game immersion.

Reduced immersion and challenges encountered due to simple bad game design can cause players to fully drop a game [1], showcasing the importance of good NPC design.

Given the wide spectrum of NPC behavior, numerous methods exist to implement NPC AI with different architectural approaches. Each of these different architectures offers trade-offs in scalability, performance, complexity, and behavioral realism. The architectures examined in this report include widely used categories such as rule-based systems, finite-state-machines (FSMs), behavior trees, and utility-based AIs. Supporting layers such as adaptive systems and advantaged AI are briefly discussed at a higher level to illustrate their role in modifying core AI behavior, but the report's primary focus remains on the comparative analysis of the different core architectures.

However, it is important to note that these explanations detailed in the following sections only discuss basic implementations of these architectures. Coding and software engineering are highly diverse with regards to architectures and implementations as many different modifications can be made for each code to fit requirements needed for its specific project. Discussing all the types of architectures with all their possible alterations and different inner algorithms would make comparisons unfeasible. Therefore, the implementations discussed and the following comparisons are performed using basic ideas with flexible alterations to the concepts.

The report provides a comparative view of these architectures based on defined evaluation criteria to assess suitability for general use and genre specific architecture suitability. While the discussion of this report focuses on the architecture implementations in games, the architectures are used throughout many different real-world implementations, such as robotics,

software systems, and modern assistive AIs like Siri, where similar trade-offs apply.

II. NPC AI Architectures

There are many architectures and methods for implementing NPC AI with varying degrees of complexity and each serving different uses with desired trade-offs. These architectures are not always mutually exclusive and can be used in tandem. While game companies and developers do not always offer expanded proofs on what architectures are used in their games, these can be inferred through mapping the observed behaviors within these games to the widely used definitions for each architecture. Each of the following architectures will be analyzed thoroughly with certain examples provided from inferred implementations in games as a precursor to a general comparative analysis and then a genre specific recommendation.

A. Rule-Based Systems

This system can be described as a foundation for decision making for NPC AIs. This architecture style was the dominant approach during the 1970s and early 1980s, and has been used intermittently in gaming the last 15 years [3]

The rule-based system offers decision making through pure simple conditioning utilizing if-then triggers. While other systems also utilize conditional logic, the rule-based systems solely utilize those. As described in "Artificial Intelligence for Games", this architecture is structured into two main parts: a database containing all the game info relevant to the rules, and a stored set of if-then rules [3].

Implementations of this architecture, like most others, all have varied elements and styles that also affect scalability and maintainability. Small games with a small rule set can afford to utilize regular status checks with simple prepared if-then blocks instead of arranging them into a set.

Other implementations provide a separation of concerns and a middleman by adding a so called "arbiter" that decides which of the triggered rules in the set fires. Without this "Arbiter", the general assumption is that the first rule triggered is the action which fires [3, Ch. 5 p. 418]. This can lead to reduced flexibility or varied gameplay.

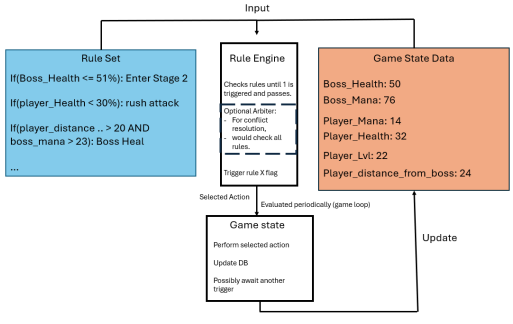


Fig. 1. Rule Based Systems implementation Graph visualization

Figure 1 provides a possible implementation. This is not the only style, as is the case with almost all coding implementations, but offers a base standard to visualize the system. Some games could add a connection from the game state to the rule set by adding or removing rules over the course of gameplay to increase variability.

In worst case scenario, this specific rule-based system implementation provides decision making through iterating over the set of rules n, where each rule may require scanning the entire game state data, of size m, to provide a single evaluation O(n*m). Over j repeated evaluations throughout the game, the overall complexity is O(n*m*j).

In practice, not all rule checks are triggered where there is no arbiter, such as the case in figure 1, with early stoppage for first passing rule and many rule conditions operate on constant-time state checks, making the average case for performance closer to O(n) per evaluation and overall O(n*j).

Memory expenditure for this architecture is generally low as most implementations don't require storing data over time or viewing historical data but rather operate in the current game state. A memory bottleneck here is unlikely unless in the case of an unusually large rule set or requirements to maintain prior world state data.

Games with constrained and turn-based nature often offer a good environment for the utilization of Rule-based systems in AI opponents. The restricted and reduced game state data makes creating rules for these games easier without the need to resort to overcomplicated rules or utilization of advantaged AI methods (discussed later in this report).

A representative example can be found in certain implementations of the game Monopoly. The AI opponent is constrained in possible moves since they utilize the same random dice rolls, follow turns of the game, and simply buy property. The decision to buy property can be easily handled with a rule set utilizing the price of the property, the possible returns (property rent), and the current cash available to the AI.

This illustrates how rule-based systems can be effective and efficient in environments with limited decision space.

B. Finite State Machines (FSM)

Finite state Machines are not exact decision-making systems but rather a way to encapsulate possible decision making and outcomes available to an entity during a specific period or condition. In a basic state machine, actions or behaviors are associated with a state and carried out continuously by the character in that state [3, p. 318].

This approach offers flexibility through multiple different behaviors and actions available to the units/NPCs depending on the current state as opposed to a static rule set. However, the internal logic used to dictate the actions to take within a state are most often implemented via conditional checks (e.g. if-guards) or time-based conditions.

FSMs are relatively simple to design and visualize. They rely

on three parts, the number of states, number of transitions between those states, and the set of actions to take within those states. While FSM behavior can still be replicated via other methods like decision tree, in most cases, it is better to use an FSM designed for this purpose [3, p. 357].

As noted in Artificial Intelligence and games, FSMs have been proven to work well with games, dominating the decision making processes of NPCs up until the mid 2000s [4, p. 91].

They are also extremely versatile in their implementation method, with it being rare for 2 developers to utilize the same techniques, making it difficult to represent FSMs with one example [3, p. 325].

A simple hypothetical FSM controller for Age of Empires 2 inspired by unit behavior is represented in figure 2 below. In this strategy game, units can be placed in one of 4 behavioral states each dictating different behavior in the absence of user inputs. User provided commands often take priority over these behavior states but during the absence of these commands, the regular behavior from these states can be observed.

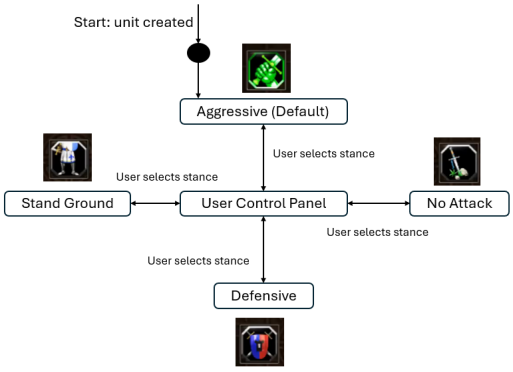


Fig. 2. FSM visualization for AOE2 unit states

In this implementation, no explicit entry and exit actions were observed for the states. The start begins at the aggressive since it is the default set option for most unit creations in the game. Additionally, the transitions are all graphed as going to the central control panel to simplify the graph visually since all the states are interconnected. Transitions from any one state to another is possible at a user's request/selection. As a result, the state management is simplified and centralized using a user control panel which, stores current state, handles user state requests, and performs the transitions accordingly.

The *Aggressive* state was the default provided to all units, and its behavior was to stay idle until an enemy target appears in sight. After target sighting, the behavior moves to following them to attack regardless of direction or distance from starting area so long as they remain in sight.

This contrasts the *Defensive state* where the unit behavior was observed to stop targeting enemy units when leaving initial standing area by too much distance and return close to the initial starting area. Both the aggressive and defensive behaviors are easily observable in [5, 1:04-2:00].

The *Stand Ground* state restricts movement allowing units to attack only enemies in range. It is ideally used and helpful for grouped ranged units. Finally, the *No Attack* prevents units from initiating or retaliating in combat without explicit user commands. These two states' effects are easily inferable from without needing explicit observation to identify their key behaviors like the defensive state.

This implementation provided simple, observable change in unit behavior making them more easily controllable. For this simple implementation of FSM, the memory cost is O (1) per unit since only need to store the current state with the worst-case time complexity being O(n) where n is the number of transitions available.

Overall, FSM performance scales primarily with the number of active entities rather than the complexity of individual state logic, making it suitable for real-time game environments with large numbers of units.

Hierarchical Finite state machines (HFSM) extend basic FSM by introducing additional layers within states enabling more complex behaviors while simultaneously improving the organization of state transitions. This enables increased complexity that helps in addressing limitations of regular FSM, specifically in system requiring handling of interruptible behaviors (such as alarm implementations). HFSM allow what is an approximate of being in more state than once at a time using "history states", which allow previously active substates to be resumed [3, p. 328].

In 2006 developer Jeff Orkin discussed the utilization of HFSM in the game F.E.A.R [6], [7]. While it was not explicitly called by the name HFSM, the implemented method was practically identical.

C. Behavior Trees

Behavior Trees are a practical hierarchical structure composed of a finite set of behaviors, designed to be scalable [8, p. 2] and to increase modularity in NPC control systems [9, p. 4]. Behavior trees, true to their name, implement a tree structure with a directed rooted tree where internal nodes control the flow and leaf nodes are execution nodes or executable behaviors. They typically employ a depth first search (DFS) search method to go through the tree and evaluate nodes from the root downward each tick. [8, p. 2] Ticks are updates where the behavior trees reperform the checks to determine the NPC actions. They are often done every other frame.

Damian Isla, in his 2005 GDC talk on AI implementation in Halo 2, defined behaviors as programs that take temporary control of an actor in order to accomplish a task [10], [11, 15:25]. This definition is useful for understanding the role of behavior nodes in game AI systems.

In their basic classical form (a sufficient example for this report), behavior trees consist of four main types of control flow nodes: sequence, fallback, parallel, and decorator.

Sequence nodes—often represented by an arrow, as shown in Figure 3—execute their children in order and return success only if all child behaviors succeed.

Fallback nodes (also called selector nodes), typically represented by a question mark as seen in Figure 3, return success if any one of their child nodes succeeds.

Parallel nodes execute all children simultaneously and use a developer-defined success threshold m: if at least m out of n children succeed, the node returns success.

Finally, decorator nodes apply user policy to a single child node to either assess the result or condition their execution. Common standard policy examples for these nodes exist such as the inverter decorator which flips the return value of the child nodes.

For executable nodes or leaf nodes, there are two types: condition nodes and action nodes. As their names suggest, condition nodes perform what amounts to if-guards while action nodes execute behaviors. Actions return success if they complete, and failure if impossible to complete. Conditions simply return their Boolean logic outcome.

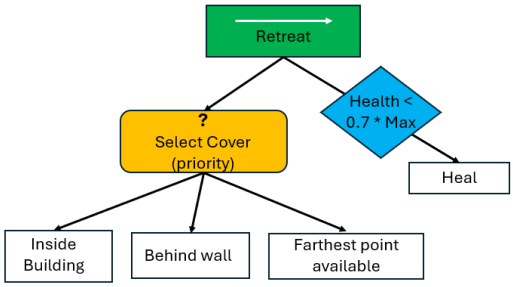


Fig. 3. Behavior Tree simple example

BTs are dominant in the game development scene with successful application games such as Halo 2 (Microsoft Game Studios, 2004) and Bioshock (2K Games, 2007) [4]. This is because BTs offer relatively easy to design, read, and test structure with highly modular behavior and actions.

Consider the example in Figure 3. The path is easy to traverse and follow. Additionally, if a developer needed to add another action to take in the retreat tree, there would be no need to alter or rewire existing connections.

With regards to time complexity, the worst-case scenario would be that the behavior tree had to run every single node in the tree (fallback child nodes all fail or sequence nodes all pass). This would still overall result in a O(n) where n is the number of nodes in the tree. In average this would not always be the case since early children of sequence nodes or fallback nodes can result in early stoppage due to the failure or success of a single node respectively.

Similarly, the memory complexity is O(n) as the whole graph has to be stored within the memory regardless of which nodes ran each tick.

D. Utility-Based

Utility Based AI architectures include the element of scoring actions to define the optimal decision to make. It can be summed as "using numbers, formulas, and scores to rate the relative benefit of possible actions" as Dave Mark described it in his GDC talk in 2013 [12, 34:00 -]. Utility based AI can complement other decision-making architectures or be implemented as a stand-alone reasoner, which is the focus of this part here in discussing it as a separate architecture.

The utility-based approach is appreciated by developers as a highly modular, reusable architecture with dynamic factors and easily extensible considerations. This approach has seen many widespread use in successful games such as Red Dead Redemption by Rockstar games 2010, and Killzone 2 by Sony in 2009. Furthermore, the architecture was used for a wide variety of decision-making tasks from Boss AI decisions to NPC dialog selection [4, p. 69].

The following is a brief explanation of the process of decision making via a utility-based system that will be expanded on later for each step:

1) Take environment, situation, and unit data
2) Normalize that data
3) Apply response curves for each Actions axis
4) Aggregate the axis values to define the action overall score
5) Apply the corresponding Prioritization category for each action to the scores
6) Choose the highest scored action
7) Repeat decision making

This is basically the process the NPC AI takes for each decision-making moment. Figure 4 shows a visualization of the process. The process might seem complex with vague or unfamiliar wording, but this method of implementation is relatively intuitive and flexible to developer needs.

The first step is often the commonality between all decision-making processes. To make a realistic NPC AI, it needs to be one that responds to the environment changes in the world around it. Normalizing the data however is a step that is not always common or needed for other architectures.

Normalizing the data is used to ensure that all the resulting axes fall within a standard range which is typically from 0 to 1. This is done so that comparisons are easier to understand from a developing standpoint and for more fair comparison in utility score. Not all data points in games are similar in values; health could go up to thousands while mana points for example are not as plentiful. Normalizing their values simplifies the task by having similar values provide the same meaning and influence the decision making equally.

For numerical inputs that do not have a set maximum or non-numerical inputs, standardization takes a few more steps than simple normalization (e.g. dividing by max value).

With non-numerical values such as Booleans and Enums, the standardization can be performed via arbitrarily assigned

values corresponding to each possible Boolean or enumeration. This converts the non numerical values to workable data that can still be useful across and derive different scores for different actions to take.

Clamping is often the go to practice in the case of numerical values whose values do not possess exact minimums or maximums such as current accumulated gold or debt. As discussed by David mark in his "Architecture Tricks" GDC 2013 talk [12, 44:10 - 45:00], clamping is the practice of constraining values to certain ranges. So, if the data point value is higher or lower than the range maximum and minimum, the maximum and minimum values are the ones used. In some cases, simpler capping may be applied when only the upper bound is needed. For example, distance values are capped at a level where additional distance no longer affects decision making.

Response curves, in this context of NPC AI development, are mathematical representations of the relationship between a data point and an abstract value. So, for example, how high an NPC anxiety value is as the data point, and the "need to go back to base", abstract value. These representations help visually map expected responses and reactions across different states in the game. Additionally, response curves provide the added realism and flexibility of offering nonlinear options for the relations between the abstract values and the data points. Human behavior, which is what most NPC AI tries to emulate, is not linear and so simple conditions don't always offer the capability to emulate that behaviour. The response curve values are what scores are attributed to the axes of possible actions to take.

Each action has multiple separate axes within. They can be likened to considerations humans take prior to choosing between each action. One doesn't only consider how hungry they are before eating but also how much money they have. This is the same here, so once these axes have gotten their scores from 0 to 1 with 1 being that this is the optimal situation to accept this action, the scores of each of the axes are aggregated to find the overall action utility. Aggregation is often performed via multiplication in basic utility systems, but other systems can employ different methods such as addition with assigned weights for each axis. This is done for each possible action the AI is programmed for.

Prioritization is a method to support decisions for certain actions over others regardless of circumstances. Urgent actions may need a higher effect on decision making compared to regular idle actions. These priorities can be placed via developer testing to check which weights provide the desired reactions from NPC AIs.

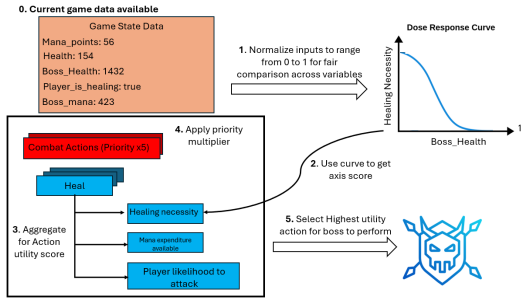


Fig. 4. UtilityBased visualization for decision making process

The memory complexity of this proposed system equates out to O(n) where n is representative of the number of stored facts or state variables required for evaluation. There is no need for storing any element's history or past lookups. The time complexity evaluated per action evaluates to O(m) where m is the number of axes for that one action. When evaluating all possible actions or when performing one decision making process, the overall time complexity equates to O(n * m) where n is the overall number of possible actions while m represents the average number of axes/considerations per action.

A well-known NPC AI that can be interpreted as a utility-based system is the SIMS. Character behavior is observed to be driven by internal state variables representing character needs such as food, water, sleep etc. As explained in "Those Darned Sims: What Makes Them Tick?" [13] actions available in the environment "advertise" their effects to the character which produces scores for each action. The scores are then evaluated with the highest scoring action being selected.

This mechanism closely aligns with utility-based AI concepts, where actions are scored and then chosen based on those scores representing relative effectiveness. However, in developer talks, the system is more described in terms of satisfying internal state variables rather than explicitly defined AI architecture.

III. Layered Supportive Elements

This section describes systems that are applied atop the core AI architectures to modify or enhance the NPC behavior without fundamentally changing the underlying decision-making processes. These layered additions still factor in the overall build of the NPCs enough to warrant a deeper look into their mechanisms.

In many game development contexts, the core AI systems such as rule based logic and FSM are kept stable while layered additions modify their outputs. These modifications can include difficulty adjustments, or adaptive balancing.

A. Adaptive/Learning

In commercial game development, use of machine learning NPC systems is rare. Most shipped games rely on other AI

architectures due to constraints in performance, predictability, and development cost [14]. As a result, true standalone learning-based NPC AIs are mostly limited to research environments rather than deployed along with commercial games. An example of reinforcement learning NPC AI developed in research context can be found in Google DeepMind's AlphaStar [15], which demonstrates how self-play reinforcement learning can produce extremely high performance in controlled environments.

In practice, developers implement Adaptive AI in games as an extension layer applied to existing NPC AI architectures like FSMs or behavior trees, where decision making parameters are modified in response to the current state of the game or player behavior. The primary purpose is to increase realism, engagement, and replayability.

One of the most well-known examples available is in the game Left 4 Dead. Mike Booth, a lead designer in the game, expanded on the idea in his GDC talk in 2009 when talking about the game's "AI Director" [16]. The AI director alters the game state based on an estimation of the players' "Emotional intensity", inferred from in-game metrics such as damage taken and encounter pressure. The changes were done to keep the dramatic pacing through either increasing hostile mobs or reducing them based on this emotional intensity estimated [17].

This does not constitute learning or adaptive AI in the computational sense as explained earlier but instead showcases reactive systems that modify gameplay parameters atop of existing AI architecture to simulate adaptation.

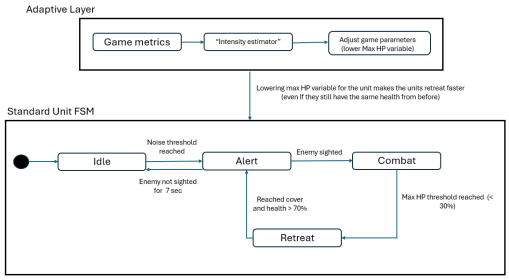


Fig. 5. Adaptive Layer visualization atop of existing unit FSM example

B. Advantaged

Advantaged AI is another example of layering atop existing AI architectures. This layer focuses on providing the NPCs systemic advantages over the player hence the term "advantaged AI". When developing more difficult levels for playing in many games, it is important to keep the challenging edge for players to provide them with a goal to beat. However, the difficulty for developers arises in providing an AI that can challenge top level players through simple behavioral scripting.

Many games resorted to simply allowing the AI to have certain advantages over players at certain difficulties. Age of Empires 2 1999 version is is well known for displaying this approach with cheating AI at the highest level of difficulty. The difficulty settings allow the AI to enable game cheats and utilize additional resources not present on the map to fight.

A developer known by the alias "Promi", who identifies himself in the interview as Marius Beck, discusses the AI difficulty scaling in a community interview in [18, 3:22 - 4:00].

IV. Comparative Analysis

Having examined the core architectures, their implementations, and commonly used layered additions, a comparative analysis can now be conducted. The evaluation criteria used are behavioral realism, unpredictability, computational efficiency, and development overhead. Development overhead includes elements such as costs, time, maintainability, and extensibility. These criteria reflect the practicality on development side while balancing them out with the desired behavioral results expected from the player base to provide a comprehensive fair ranking for implementation when selecting NPC AI architecture for general use.

A. Benefits and downsides

This section is to identify all the advantages and disadvantages or trade-offs present in each architecture. No architecture is perfect, but some are better than others. Documented limitations of each architecture are noted and factored in the ranking for general use. These descriptions also inform the genre-specific deployments recommendations in the following section.

1) Rule-based systems

A major benefit to rule-based systems is their simple and easy-to-implement approach for NPC AI. They rely on elementary if-then conditions in their basis which results in a low learning curve for basic implementations since rule writing is simple [19]. Furthermore, this architecture supports modular rule design, allowing individual behavior rules to be added and modified independently of others [20, Ch. 11].

However, this flexibility introduces a notable challenge. Rule-based systems do not inherently enforce an execution order, meaning that while rules can be added independently, conflicts may arise when multiple rules are eligible to fire simultaneously [3, Ch. 5]. This can also lead to limitations where certain rules are overshadowed by others depending on the evaluation order. Although this can be mitigated through the use of arbiters for conflict resolution, designing and implementing effective conflict resolution methods becomes more complex and costly as the rule set grows. Thus, it can be stated that the difficulty of maintaining correct, predictable, and desired behavior tends to scale with the size and condition density of the current rule set. Another downside is the deterministic nature of this architecture. Determinism is often a great point for systems as they prevent unexplained behaviors, however in gaming this is not the case. Should the NPC AI be fully

deterministic, players are bound to recognize patterns that reduce immersion and realism. While not using a rule-based system specifically, Pac-man was another game that suffered in gameplay due to determinism as it showed that there was an optimal strategy to beat the game. This reducing it to pattern memorization rather than reactive gameplay [21].

These limitations collectively restrict this architecture's ability to scale beyond simple behaviors without considerable increase in design complexity.

2) Finite State Machines

Finite state machines provide a structured approach for NPC AI by organizing behavior into a finite set of states. FSMs have the benefit of being easy to visualize with a distinct graphing style that helps document planned coding implementations.

FSMs provide the advantages of simplicity to code combined with intuitiveness and a lack of computational overhead. FSMs do not process complex reasoning, instead just following predefined conditional logic (e.g. if-then) for transitons between states [22, p. 43]. As a result, FSMs are widely used in game development due to their intuitive design and flexible implementation that it is rare for 2 developers to use the same techniques for their FSMs [3, Ch. 5.3].

These advantages however hide some unexpected limitations at first glance. First is the issue of maintainability and extensibility. Designing a game often requires many updates with frequently changing requirements. To quote Mat Buckland from his book *Programming game AI by Example*, "as more states and conditions are added, this sort of structure ends up looking like spaghetti very quickly, making the program flow difficult to understand and creating a debugging nightmare" [22, p. 46]. Another disadvantage is the determinism highlighted in the earlier section. Pac-man can be modeled as a FSM where ghosts transition between states such as chase and evade based on player actions and timing conditions [22, p. 45]. This resulted in the downside observed from gameplay being deterministic is attributed to FSMs [21].

3) Behavior Trees

Behavior trees were actually implemented as a way to increase the modularity for NPC AI architecture development from the previous available implementations such as FSMs [9, p. 4]. As noted by Bagnell et al., "The main advantage is that individual behaviors can easily be reused in the context of another higher-level behavior, without needing to specify how they relate to subsequent behaviors" [23]. This highlights the increased modularity as a key advantage to behavior trees.

In addition, the hierarchical structure provided by the behavior trees also allows for a major beneficial ability to decompose complex behaviors into smaller more manageable sub-behaviors. This improves overall readability and maintainability. Extensibility is also a major improvement. As opposed to earlier implementations, new behaviors can be added without having to heavily alter previous implementation [24, Ch. 4]

Despite the advantages of behavior trees in modularity and structure, they also introduce several limitations that would require additional complexity and changes to the basic style. One of these limitations is the potential of inefficient evaluation since behavior trees are ticked continuously. A notable example of this issue was encountered by the Halo 2 AI development team as explained by Damian Isla in his 2005 GDC talk [10]. Additionally, behavior trees do not inherently solve issues of optimal decision-making. While behavior trees provide structure, they do not evaluate or compare actions in terms of utility. As a result, in complex situations with many factors for consideration, behavior trees might produce behavior that appears limited in scope or realism.

4) Utility Based AI

Utility based AI approach is one where each possible action is assigned a numerical score representing its utility. This AI system evaluates scores for actions based on a configurable set of user-developed utility axes. This option offers the ability for each action to hold multiple considerations and calculate a score that represents the utility of that action. After comparing that action's utility score among other actions, the AI selects the highest utility score. This approach allows for multi-dimensional decision-making as opposed to simple one dimensional if-conditions enabling flexible and context sensitive decision making.

Key advantages for this AI architecture include its extensibility and modularity. Adding new axes to actions and fully new actions to the available sets do not affect the present implemented ones and simply provide more possiblities for the AI to consider [12, 57:37]. The mathematical nature of the decision making process for this architecture makes the decision making transparent, easy to visualize, and highly tunable as response curves providing perfect visualizations for expected behaviors. Furthermore, a random noise score can be applied to generate the desired level of non-determinism without requiring an additional layer of implementation such as adaptive AI.

Despte the architecutes's flexibility, utility based AI increses the difficulty of designing the effective utility functions accurately. Since behavior is entirely determined through the utility scoring system, poorly designed models are much easier to create. These can result in unimaginative and unrealistic behavior response. Significant testing and tuning is required before utility values reflect accurately in gameplay, especially in large systems. Another drawback is the increased computation in growing systems. Since in this model, all actions and their axes are calculated each evaluation, large systems can produce heavy computations requirements.

B. Ranking

Based on the evaluation criteria defined earlier, each AI architecture can be ranked in suitability for general purpose NPC implementations. This ranking takes into account the advantages and disadvantages of each model discussed earlier and

provides a relative ranking. Note that the use of relative ranking here implies that this is not absolute evaluation of the capability of these systems/architectures of producing general purpose NPC but rather a comparison of them amongst each other. This ranking considers behavioral realism, computational efficiency, development overhead, and long term iterative development support (e.g. maintainability and scalability).

At the lowest ranking on this list is the rule-based systems. While they offer minimal development overhead with little need for skill and complex code, their limitations in long-term development support with their low maintainability and scalability hold them back significantly. Despite their computational efficiency, these systems would face large hurdles in tackling large or complex development needs with increased rule set sizes. Additionally, their deterministic nature limits the realism expected from their dynamic gameplay scenarios.

FSMs rank slightly higher than rule-based systems due to their readability, improved structural organization, and clearer execution flow. While FSMs are more structured, they still suffer from similar limitations shared with rule-based systems such as maintainability and predictability with their deterministic flow. FSMs do not provide great extensibility compared to other options available and the difficulty in increasing available states shows the lack of modularity. FSMs also do not offer the greatest transition capability between states with a limited set of transitions.

Behavior trees rank second overall in suitability for general purpose implementations. Their hierarchical design enables simplification of complex behaviors into simpler sub-behaviors which increases readability and maintainability over FSMs. Additionally, extensibility offered for long term iterative development greatly reduces developer overhead. However, their continuous evaluations which support potential inefficiencies in large systems and memory storage of the full trees regardless of run time do provide some limitations in large programs. Furthermore, the realism of this architecture is not optimal as the decision making is one dimensional in its considerations following predetermined conditions.

This leaves the winner being the utility based AI. It ranks the highest as its primary advantage lies in its complex decision making with multiple factors and great possibility for realistic behaviors. Although it introduces higher design complexity, the benefits are worth the added complexity with easy iterations and changes performed over the course of development due to the high modularity, flexibility, and scalability. An additional point in its favor is the ability to implement non deterministic behavior using random noise improving unpredictability without sacrificing realism or structure.

Overall, while each architecture has domain specific strengths, utility based AI offers the most balanced option for general usability overall with adaptability and high behavioral richness that can provide different responses for different desired implementations. Simpler architectures remain suited for more constrained or specific environments without added complexities to their implementations.

V. Genre Specific Deployment

This is a discussion of which genre best suits each of the 4 models outside of the general suitability.

For rule based systems, the best suited genre is the turn based and puzzle games. This is due to their limited environments and constraints on possible movements. This is a genre filled with rules and rule based systems perform greatly with no difference in their gameplay being distinguished from real human gameplay. Additionally the existence of the game rules already limits the rule set for the AI greatly since the game already introduces the limitations on its own. Human players abide by the same rules and the possible actions and strategies are finite. Creating a great strategy using simple rules is within bounds and perhaps even preferable.

FSMs are well suited AI architectures for games where NPC behavior can be cleanly divided into small states. This can include games such as platformers or early arcade style games. A rare but suitable genre for FSMs would be the stealth genre. The deterministic behavior in these genres is often appreciated as users need to adapt to the observed patterns in their opponents to win and unpredictable behavior can ruin gameplay. Furthermore, there are not many states for NPCs to go through in these stealth games or many interaction types with the user where realistic or unrealistic behavior matters much.

The most suited genre for behavior trees has to be among action games and shooters. NPCs require complex interruptible, priority driven behaviors that react reasonably to surrounding environment changes. The enemies need constant switching between behavior modes such as shooting, retreating, healing, and even surrendering. These games benefit from the continuous checks performed by the behavior trees and offer nearly realistic behavior in combat environments.

For utility based AI, any genre that requires highly realistic behavior is suitable and appropriate. Utility based AI can be used for all kinds of actions and even dialogue conversations. Open world RPGs are some of the best games to showcase the full capabilities of utility based AI to perform highly realistic reactions without scripting. This is even more crucial considering open world RPGs invite players to interact with them in highly imaginative and unpredictable ways where regular if-conditioning would fail to react or prepare for accordingly.

VI. Conclusion

This report examined four core AI architectures alongside two possible additive layers in order to analyze and compare the different results of NPC AI behavior obtained from each of these architectures. Each architecture was analyzed through its implementation mechanics, computational characteristics and complexities, and practical game examples to establish a foundation for existing works and their observed natures for comparative evaluations.

The comparative analysis revealed a clear progression in capability across architectures when evaluated against the criteria of behavioral realism, computational complexity, predictability, and long term development maintainability. Rule-based systems, while computationally inexpensive at the start, resulted in low realism, heavy computational costs, and difficult maintainability over large and complex systems. Finite state machines offered structure to previously unstructured implementations but still kept low modularity and difficulty in maintaining and updating. The behavioral results were also limited in transitions between actions for believable realism. Behavior trees address the scalability and modularity shortcomings of FSMs through their hierarchical design and priority-based execution, making them the dominant architecture in modern commercial game development, though they remain deterministic in their basic form and carry higher initial development overhead. Utility-based AI ranked highest overall, offering the most balanced combination of behavioral realism, extensibility, and non-deterministic capability through noise-based scoring, at the cost of higher design complexity.

Genre-specific analysis further demonstrated that no single architecture is universally optimal across all game types. Rule-based systems and FSMs remain appropriate for constrained genres such as turn-based and puzzle games, while behavior trees suit action and shooter genres, and utility-based AI is best deployed in simulation and open-world RPG contexts where emergent and varied NPC behavior is essential to player immersion.

REFERENCES

[1] T. B. Požar, T. Skala, M. Maričević, and P. Srdanović, *Evaluation of NPC Behaviour in a Simulated Video Game Environment Through Player Perception*, Acta Graphica, vol. 34, no. 1, pp. 29–38, Mar. 2026. Available: https://www.actagraphica.hr/actagraphica/article/view/41. [Accessed: Apr. 13, 2026].

[2] J. Huizinga, *Homo Ludens*, 1955. Available: https://merton.bellarmine.edu/files/original/b0899cfad820ab8ad7033952b7a022ba1d7cab9d.pdf. [Accessed: Apr. 13, 2026].

[3] I. Millington, *Artificial Intelligence for Games*. Boca Raton, FL: CRC Press, Taylor & Francis Group, 2019. Available: https://theswissbay.ch/pdf/Gentoomen\%20Library/Game\%20Development/Programming/Artificial\%20Intelligence\%20for\%20Games.pdf. [Accessed: Apr. 13, 2026].

[4] G. N. Yannakakis and J. Togelius, *Artificial Intelligence and Games*, 2nd ed. Springer Nature, 2025. Available: https://gameaibook.org.

[5] Spirit of the Law, "Defensive Stance, Guard, and Follow: What's the difference?" YouTube, Sep. 22, 2022. Available: https://www.youtube.com/watch?v=YSyD3n_Wk6A. [Accessed: Apr. 13, 2026].

[6] J. Orkin, "Three States and a Plan: The A.I. of F.E.A.R.," Semantic Scholar, 2006. Available: https://www.semanticscholar.org/paper/Three-States-and-a-Plan%3A-The-A.I.-of-F.E.A.R.-Orkin/012ef03d0f951092b8645b69aebdbce900ac03e4. [Accessed: Apr. 13, 2026].

[7] J. Orkin, "Three States and a Plan: The AI of F.E.A.R.," GDC Vault, 2006. Available: https://gdcvault.com/play/1013282/Three-States-and-a-Plan. [Accessed: Apr. 13, 2026].

[8] Y. A. Sekhavat,"Behavior Trees for Computer Games," *International Journal on Artificial Intelligence Tools*, vol. 26, no. 2, p. 1730001, Jan. 2017. Available: https://doi.org/10.1142/S0218213017300010. DOI: https://doi.org/10.1142/S0218213017300010.

[9] M. Colledanchise and P. Ögren, *Behavior Trees in Robotics and AI: An Introduction*. Springer, 2018.Available: https://arxiv.org/abs/1709.00084. DOI: https://doi.org/10.1201/9780429489105.

[10] D. Isla, "Managing Complexity in the Halo 2 AI System," GDC 2005. Available: https://www.gdcvault.com/play/1020270/Managing-Complexity-in-the-Halo. [Accessed: Apr. 13, 2026].

[11] D. Isla, "Managing Complexity in the Halo 2 AI System," GDC 2005, uploaded to YouTube by Bungie Halo Archive, Jun. 04, 2021. Available: https://www.youtube.com/watch?v=m9W-hpxuApk. [Accessed: Apr. 13, 2026].

[12] D. Mark, "Architecture Tricks: Managing Behaviors in Time, Space, and Depth," GDC Vault, 2013. Available: https://www.gdcvault.com/play/1018040/Architecture-Tricks-Managing-Behaviors-in. [Accessed: Apr. 13, 2026].

[13] J. Doornbos, "Those Darned Sims: What Makes Them Tick?" GDC Vault, 2001. Available: https://www.gdcvault.com/play/1013969/Those-Darned-Sims-What-Makes. [Accessed: Apr. 13, 2026].

[14] M. Jacob, S. Devlin, and K. Hofmann, "'It's Unwieldy and It Takes a Lot of Time' — Challenges and Opportunities for Creating Agents in Commercial Games," *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 16, no. 1, pp. 88–94, Oct. 2020. doi: https://doi.org/10.1609/aiide.v16i1.7415.

[15] The AlphaStar team, "AlphaStar: Mastering the real-time strategy game StarCraft II," Google DeepMind, Jan. 24, 2019. Available: https://deepmind.google/blog/alphastar-mastering-the-real-time-strategy-game-starcraft-ii/. [Accessed: Apr. 13, 2026].

[16] Mr Morton (YouTube channel), "Development of Left 4 Dead by Mike Booth GDC Lecture," YouTube, Jan. 29, 2023. Available: https://www.youtube.com/watch?v=PJNQl3K58CQ. [Accessed: Oct. 19, 2025].

[17] M. Booth, *The AI Systems of Left 4 Dead*, 2009. Available: https://steamcdn-a.akamaihd.net/apps/valve/2009/ai_systems_of_l4d_mike_booth.pdf. [Accessed: Apr. 13, 2026].

[18] Spirit of the Law, “How the AoE2 AI Thinks (ft. Promi),” YouTube, Jan. 12, 2019. Available: https://www.youtube.com/watch?v=-S1CkfzEHSU. [Accessed: Apr. 13, 2026].

[19] D. G. Giovanni and S. D. Facchini, *A Comparative Study of Rule-Based and Data-Driven Approaches in Industrial Monitoring*, arXiv preprint arXiv:2509.15848, 2025. Available: https://arxiv.org/abs/2509.15848. [Accessed: Apr. 13, 2026].

[20] D. M. Bourg and G. Seemann, *AI for Game Developers*, Beijing; Cambridge: O’Reilly, 2004.

[21] P. Rohlfshagen, J. Liu, D. Perez-Liebana, and S. M. Lucas, *Pac-Man Conquers Academia: Two Decades of Research Using a Classic Arcade Game*, IEEE Transactions on Games, vol. 10, no. 3, pp. 233–256, Sep. 2018. doi: https://doi.org/10.1109/tg.2017.2737145.

[22] M. Buckland,*AI Game Programming by Example*,Plano, TX: Wordware; Lancaster, 2004.

[23] J. A. Bagnell *et al.*, *An Integrated System for Autonomous Robotics Manipulation*, 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, Oct. 2012. doi: https://doi.org/10.1109/iros.2012.6385888.

[24] M. Dawe, S. Gargolinski, L. Dicken, T. Humphreys, and D. Mark,*Behavior Selection Algorithms*,Game AI Pro 360, Sep. 2019.doi: https://doi.org/10.1201/9780429055058-1.