An Open-Source Visual Tool for Modeling, Validating, and Exporting Decision Trees for Game Development

Eduardo Miyake D. Martins, Rafael G. Barbosa, Maria Andréia F. Rodrigues

Programa de Pós-Graduação em Informática Aplicada (PPGIA)
Universidade de Fortaleza (Unifor)

{dudsmiyake, bgrafael, andreia.formico}@gmail.com

Abstract. *This work presents VisNeed, an open-source visual tool for modeling, validating, and exporting Decision Trees for interactive applications. Beyond simple visualization, it supports rapid prototyping of business logic, narrative flows, and interaction rules. Aimed at digital games, embedded systems, and educational platforms, it aligns with model-driven development, user-centered design, and agile workflows. Validated trees are exported in JSON format compatible with game engines and runtime systems, enabling seamless integration. VisNeed is designed to support collaboration between technical and non-technical stakeholders through clearer documentation, traceability, and shared understanding—making it a valuable asset in multidisciplinary projects.*

1. Introduction

Decision Trees are widely used in software engineering, business intelligence, game development, and machine learning to model structured decision-making [Quinlan 1986]. In applied software engineering, they represent conditional logic, business rules, and user interactions—-supporting the modeling, prototyping, and validation of dynamic system behaviors [Kotsiantis 2013]. In digital games, Decision Trees play a key role in branching narratives, NPC behavior modeling, and adaptive gameplay [Yannakakis and Togelius 2018]. Despite their versatility, existing tools often suffer from limitations in scalability, interactivity, and customization—particularly in multidisciplinary and interactive applications.

Game engines offer distinct approaches to authoring AI behavior, often creating a trade-off between integration and portability. Popular proprietary engines like Unity and Unreal exemplify this, providing toolkits for Decision Tree [Unity Asset Store 2016] and Behavior Trees [Epic Games, Inc. 2022], respectively. However, their closed-source, engine-specific nature inherently limits portability and extensibility. This challenge persists in the open-source realm. The prominent Godot engine, with its BeeHave addon for Behavior Trees [Bitbrain and Gerevini 2022], offers a seamless integrated workflow but at the cost of vendor lock-in, rendering its assets non-portable. Moreover, community-driven solutions like BeeHave face an additional hurdle: the lack of guaranteed long-term support and forward-compatibility found in native engine tools.

VisNeed fills this gap by providing a freely available, open-source platform for flexible and scalable Decision Tree development. It streamlines authoring workflows and supports correctness verification in gameplay logic design—enhancing productivity and traceability while aligning with core software engineering principles such as modularity, reusability, and collaboration. The platform supports software engineering practices—including specification, design, implementation, validation, and evolution—while promoting usability, interoperability, and traceability. Its adaptability makes it suitable for diverse domains such as education, embedded systems, and digital games. Furthermore, [Omitted name] facilitates empirical research by supporting structured documentation, experimental design, and comparative analyses, making it a valuable asset in both development and research contexts.

Building upon our previous work on a formal editor [Barbosa and Rodrigues 2025b], VisNeed addresses its predecessor's usability limitations through a graphical, interactive front-end fully compatible with the underlying model. Rather than replacing the original editor, it extends its reach by integrating modern UI frameworks and visualization libraries—bridging formal modeling with accessible tooling. This tight integration enables users such as software engineers, game designers, UX specialists, and domain experts to collaboratively develop, validate, and communicate decision logic in dynamic, interactive systems, reinforcing its role within the software engineering toolchain.

2. Iterative Design Process and Key Features of VisNeed

2.1. Iterative Design

VisNeed was developed through three incremental iterations, following a continuous cycle of prototyping and design refinement. Each iteration involved domain experts, whose contributions and feedback helped ensure the tool met both technical requirements and usability goals.

Iteration 1 – Requirements Definition and Technology Selection. The first iteration focused on analyzing the limitations of our previous formal editor [Barbosa and Rodrigues 2025b], identifying usability and interaction challenges. Brainstorming sessions with experts supported the initial formulation of functional and non-functional requirements, such as an intuitive and minimalist interface, support for complex Decision Trees, and JSON import/export capabilities. Concurrently, technology evaluations were conducted, comparing frameworks, as well as visualization libraries. Low-fidelity prototypes were created to explore interface alternatives, and expert analysis highlighted the need for improved structural and visual organization of the nodes.

Iteration 2 – High-Fidelity Prototyping and Preliminary Testing. The second iteration focused on building high-fidelity prototypes, incorporating core VisNeed actions (node creation, editing, and deletion), interactive tree rendering, and in-platform interaction. The visual design evolved to include a robust connection system between nodes and improved layout of interface components. Multiple approaches for presenting hierarchical information and navigating tree levels were explored.

Iteration 3 – Final Implementation and Refinement. In the final iteration, VisNeed's complete design was consolidated. A dictionary feature was added for managing reusable keywords, along with a scenario builder for constructing hierarchical structures based on dictionary content. The JSON import/export system was fully integrated, maintaining coherence with the platform's design principles. A unified visual language was applied across components to ensure a consistent and intuitive user experience.

2.2. Expert-Driven Design and Refinement

VisNeed's validation was iterative and feedback-driven, involving domain experts throughout development—from early design to testing. The process followed a cycle of design, technical experimentation, and functional refinement, supported by direct interaction with target users.

In the initial phase, the platform's architecture was shaped by evaluating deployment frameworks (e.g., *Electron*, *Next.js*, *Tauri*, *Flutter*) and Decision Tree visualization libraries (*Vis.js*, *React Flow*, *D3.js*, *Cytoscape.js*) [Zhao 2013, Rauch 2016, Google 2018, Christopher and Mahler 2018, Bostock et al. 2011a, Consortium 2016, Bostock et al. 2011b]. These technologies were assessed through usage simulations involving trees of varying complexity—including hierarchies exceeding 50 levels and multiple branches—to test performance, scalability, and interaction responsiveness. The evaluation considered trade-offs in usability, resource usage, and cross-platform compatibility to ensure the system could handle demanding, interactive decision modeling scenarios.

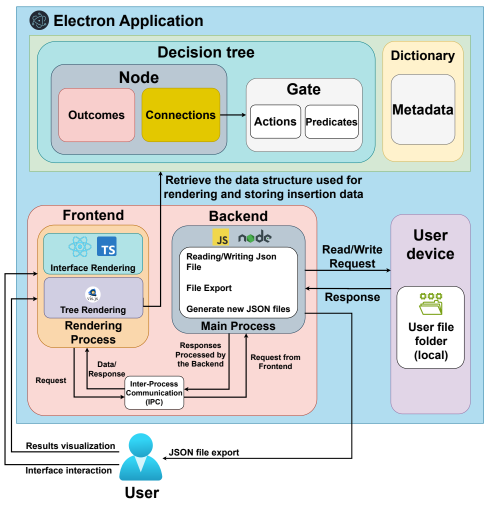
The next stage involved defining functional and non-functional requirements, emphasizing accessibility, flexibility, and ease of use. High-fidelity prototypes were developed to represent key application flows—such as tree creation and editing, dictionary management for reusable nodes, scenario construction, data export, and visual interface adaptation. These prototypes were reviewed in recurring meetings with domain experts, who provided detailed feedback on usability, interface organization, and functional alignment. Improvements were made accordingly, particularly in visual components, navigation flow, and clarity of interaction.

As implementation progressed, real-system validation was introduced. Experts engaged in hands-on testing of the working platform, assessing feature consistency, the behavior of interactive visualizations, and the fidelity of Decision Trees. This led to the identification of minor issues and usability misalignments, which were addressed through continuous refinements. The ongoing expert-driven validation process was essential to ensuring that VisNeed aligned with real-world needs. Their participation throughout—from feature ideation to functional verification—served as a key quality control mechanism. Based on these preliminary validation efforts, VisNeed shows strong potential to meet its goal of providing an interactive, scalable, and accessible platform for Decision Tree modeling and visualization for game development.

2.3. System Architecture

On the left side of Figure 1 is shown the architecture of VisNeed, which is organized into four main modules: (1) **Graphical Interface (Frontend)**, (2) **Communication Layer**, (3) **Control Layer (Backend)**, and (4) **File Management**. Implementing this architecture involved a careful selection of key technologies. For the user interface, we chose *Electron*, valuing its cross-platform support, popularity, and ease of debugging. Its OS integration adds flexibility, despite higher resource usage. For the tree visualization component, *Vis.js* was favored for its high degree of customization and interactivity. Although less scalable than some alternatives, it perfectly meets the project's needs and offers a simpler learning curve. The application was built using the *Electron* framework, integrating *React* and *TypeScript* on the frontend, and *JavaScript* (running in a *Node.js* environment) on the backend.

The **Graphical Interface** is responsible for rendering all visual components, including the interactive elements and the Decision Tree visualization. It is implemented with *React* and *TypeScript* and uses the *Vis.js* library for graphical rendering. Users in-



```
{
  "dictionary": [
    { "key": "", "type": "" },
    ...
  ],
  "nodes": [
    {
      "id": "",
      "name": "",
      "visualization": "",
      "connections": [
        { "name": "",
          "targetId": "",
          "gate": {
            "predicates": [{ "key": "", "condition": "", "value": "", "logicalOperator": "" }],
            "actions": [{ "key": "", "operator": "", "value": "" }]
          }
        },
        ...
      ],
      "outcomes": [{ "key": "", "operator": "", "value": "" }]
    }
    ...
  ]
}
```

Figure 1. Left: Architecture of the VisNeed *Electron* application, showing the interaction between frontend (*React* + *TypeScript*), backend (*Node.js*), and the local file system via inter-process communication (IPC). Decision Trees include nodes, gates, and dictionary metadata, and are exported as structured JSON. Right: Excerpt of the JSON schema used to encode Decision Trees, including dictionary entries, nodes, connections, predicates, actions, and outcomes.

teract directly with this layer to add, edit, or delete tree nodes, triggering requests to the control layer. Due to *Electron*'s architecture, the **Main Process** is isolated from the **Renderer Processes**, which run the front-end inside a Chromium instance. This allows the use of modern web technologies in a desktop environment. To enable communication between these processes, VisNeed uses Inter-Process Communication (IPC), handled by the **Communication Layer**. This layer serves as a bridge between the front and back end, enabling message passing, function calls, and data exchange. The **Control Layer** (linked to the **Main Process**) handles the application's core logic. It manages reading and writing JSON files, processing data, and maintaining the internal structure of Decision Trees. It responds to front-end requests via the communication layer. Finally, the **File Management** module handles local storage of all JSON files created or modified by users. This ensures data persistence and accessibility, allowing users to resume their work at any time.

VisNeed builds on a formal decision modeling framework composed of two core modules: a **Dictionary**, which defines domain-specific variables and data types, and a **Decision Tree Module**, which models *nodes*, *connections*, *predicates*, *actions*, and *outcomes*. These foundations guide the internal structure and data schema of the tool. Through its interactive interface, VisNeed makes this formalism accessible by visually representing dictionary elements and decision structures, enabling users to build trees with reusable data and clearly defined logic flows. The system fully implements advanced features such as gate-based navigation, compound predicates (e.g., with *and/or* operators), and action scoping tied to either transitions or nodes. The exported decision structures conform to the original JSON schema, enabling seamless integration with engines such as Unity, Godot, and Unreal without additional transformation. In doing so, VisNeed combines formal modeling precision with visual authoring and practical deployment—positioning itself as a reliable, engine-ready solution for decision logic in interactive and cross-domain applications.

2.4. Core Features

VisNeed provides a set of core features designed to support the modeling, editing, and management of Decision Trees in interactive applications:

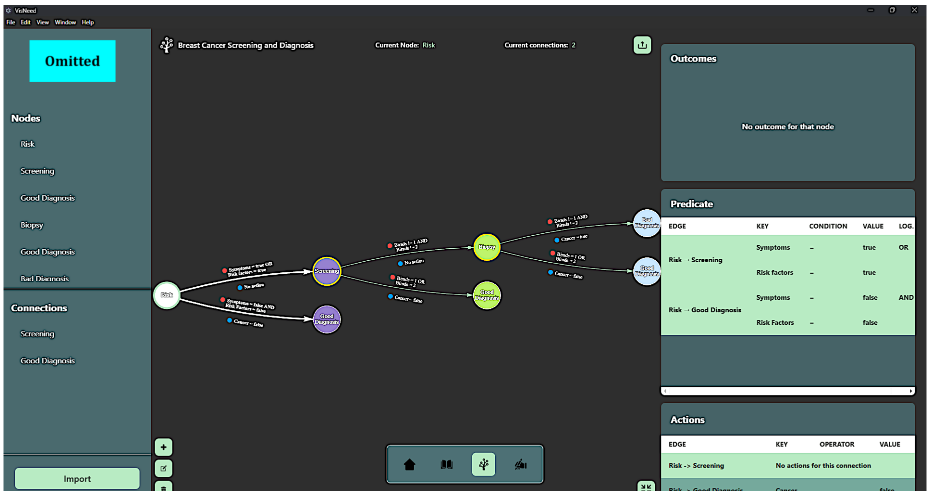
Project Management: Users can create and manage projects that organize multiple Decision Trees and enable access to all platform tools. Each project requires a name and can be deleted when no longer needed.
Element Creation: Users can add *nodes* by specifying attributes (*name*, *parent*, if any, *outcome*, *predicates*, and *actions*). *Predicates* may be added to a *connection* directly or via the source node's outgoing links. An internal *dictionary* lets users define reusable elements by *key* and *type* for building hierarchical *scenarios*.
Element Editing: Existing nodes and connections can be modified, allowing users to update properties such as *names*, *predicates*, or any previously defined attribute.
Element Deletion: When deleting a *node*, users can either promote a *child node* as the *new parent* or generate a new subtree from its children. *Dictionary* entries can also be removed as needed.
JSON Import and Export: Decision Trees can be imported/exported as JSON, enabling cross-device editing, version control, and collaboration. The right side of Figure 1 shows VisNeed's JSON schema for tree serialization and management.

3. Preliminary Validation Study

VisNeed was validated through iterative testing and internal analysis by the development team. Throughout its development, the platform was applied to real decision modeling tasks in interactive application contexts, with a focus on verifying feature completeness, interface consistency, and alignment between the visual interface and the underlying decision logic model. Several representative scenarios were developed to demonstrate the tool's cross-domain applicability. This paper presents two of them, selected due to space constraints. These scenarios were used to model and visualize Decision Trees using VisNeed, allowing the development team to assess the tool's modeling capabilities, structural flexibility, and JSON export fidelity. Although no external users participated at this stage, the process followed a structured methodology grounded in software engineering principles such as correctness, reusability, and traceability.

Case 1: Educational Decision Tree for Breast Cancer Diagnosis In this scenario, a Decision Tree was created from clinical protocols for breast cancer screening and diagnosis, supporting the development of an interactive educational tool (Figure 2). The model starts with initial symptoms and branches based on medical decisions, such as ordering a mammogram or biopsy. This case tested VisNeed's applicability beyond software engineering, highlighting its potential for modeling clinical workflows, enable adaptive storytelling, and visualizing diagnostic outcomes. The exported JSON was integrated into a Unity-based serious game [Barbosa and Rodrigues 2025a].

Case 2: Narrative Adventure RPG This scenario simulates the narrative logic of a role-playing game (RPG), where player choices affect the story's progression (Figure 3). The model includes decision points such as exploring different paths or interacting with key characters. VisNeed provided a clear overview of the entire narrative flow, supporting



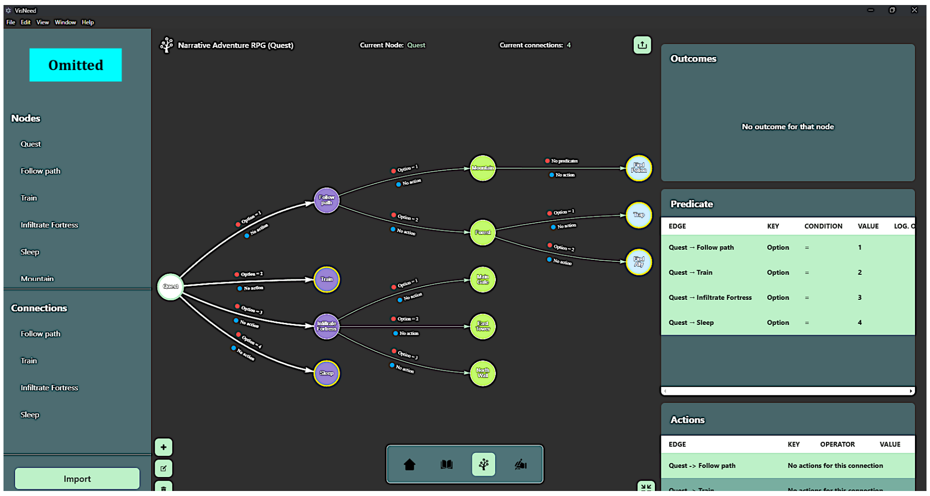
No outcome for that node

EDGE	KEY	CONDITION	VALUE	LOG.
Risk → Screening	Symptoms	=	true	OR
	Risk factors	=	true	
Risk → Good Diagnosis	Symptoms	=	false	AND
	Risk Factors	=	false	

EDGE	KEY	OPERATOR	VALUE
Risk -> Screening	No actions for this connection		

Figure 2. Decision Tree for breast cancer screening, modeled in VisNeed to illustrate clinical workflows and outcomes interactively.

planning, balancing, and iterative adjustments. Its interface allowed easy identification and correction of logical inconsistencies.



No outcome for that node

EDGE	KEY	CONDITION	VALUE	LOG. O
Quest → Follow path	Option	=	1	
Quest → Train	Option	=	2	
Quest → Infiltrate Fortress	Option	=	3	
Quest → Sleep	Option	=	4	

EDGE	KEY	OPERATOR	VALUE
Quest -> Follow path	No actions for this connection		

Figure 3. Narrative logic model for an RPG built with VisNeed.

To complement each case description, we conducted a qualitative assessment of VisNeed's behavior across the two scenarios. Table 1 summarizes how the tool supported relevant technical aspects, such as visualization clarity, modeling ease, complex branching, integration capacity, among others. These criteria reflect typical concerns in early-stage modeling tool validation. While Case 1 benefited from structured predicate hierarchies based on domain dictionaries, Case 2 involved open-ended player choices with less hierarchical, rule-based logic. Thus, relationship modeling from the *dictionary* was only partially applicable in the RPG scenario.

In addition, Table 2 maps the functional requirements addressed by VisNeed across the two case studies. This checklist highlights the tool's consistency in supporting core operations (e.g., node insertion, export) while also revealing areas for future improvement, such as real-time collaboration support, more intuitive node linking (drag-to-connect), and cloud-based saving instead of local-only storage. The selected cases helped validate the tool's versatility and modeling robustness. Together, these structured analyses provide preliminary evidence that VisNeed is flexible and applicable to different domains. Although further usability testing with end users is required, these findings support its relevance as a prototyping and documentation tool in software engineering, education, and game design contexts.

Table 1. Technical Assessment

Criterion	Case 1	Case 2
Visualization clarity	✓	✓
Ease of modeling	✓	✓
Support for complex branching	✓	✓
Integration with external systems	✓	✓
Loop/feedback representation	✓	✓
Dictionary utility in modeling	✓	✓
Relationship modeling from dictionary	✓	Partial

Table 2. Requirement Coverage

Functional Requirement	Case 1	Case 2
Node and transition insertion	✓	✓
Multi-branch grouping	✓	✓
Structured export (e.g., JSON)	✓	✓
Reuse in external systems	✓	✓
Auto keyword addition	✓	✓
Saved keyword reuse	✓	✓

4. Background and Related Work

Data-driven architectures are common in software engineering to decouple behavior from hardcoded logic. In these systems, control flow and decision-making processes are driven by structured data (e.g., JSON, XML), allowing runtime adaptability and easier customization by non-programmers [Blasch et al. 2018]. This is especially useful in interactive systems requiring flexible behavior modeling, like educational platforms, embedded apps, and digital games, where scripting or declarative configuration enables behavioral extensions. Digital games illustrate the importance of modifiability and extensibility. Many expose internal logic through scripts or visual modding tools, enabling users to redefine interactions and create custom content [Letzter 2015]. Platforms like *Nexus Mods* reflect the widespread use of data-driven customization [Black Tree Gaming Ltd. 2021], though toolkits often rely on low-level editing or limited XML configurations.

Unlike previous tools, VisNeed is open-source and stands out by supporting node-level metadata, JSON export, and branching validation. In contrast to game-focused editors, it offers a generic structure tailored to software processes and educational workflows. This design promotes high interoperability; its engine-agnostic JSON output allows the decision models to be integrated into any game engine, application, or platform equipped with a standard JSON parser.

5. Final Remarks

This work presented VisNeed, an open-source visual tool designed to support the modeling, validation, and export of Decision Trees for game development and other interactive applications. VisNeed addresses a practical gap by combining formal structure with an intuitive interface. Tested across diverse domains, it enabled the design and integration of decision logic with promising results in terms of usability, flexibility, interoperability, and reusability. Future work includes usability evaluations with domain experts and multidisciplinary users, as well as extending features for collaborative modeling and large-scale deployment.

Artifact Availability

All artifacts from this research are available under the LGPLv3 license, and the source code is available at `https://zenodo.org/records/15459896`.

Acknowledgements

Maria Andréia F. Rodrigues thanks CNPq (Grant 314776/2023-0).

References

Barbosa, R. G. and Rodrigues, M. A. F. (2025a). Breast cancer diagnosis through serious gaming: Clinical reasoning, AI-driven character morphing, and emotional engagement. *Entertainment Computing*, 52:100863. https://doi.org/10.1016/j.entcom.2024.100863.

Barbosa, R. G. and Rodrigues, M. A. F. (2025b). An interchangeable editor to create generic and adaptable decision trees for versatile applications and game development scenarios. *Entertainment Computing*, 52(100864). https://doi.org/10.1016/j.entcom.2024.100864.

Bitbrain and Gerevini, V. (2022). Beehave: Behavior tree AI for Godot Engine.

Black Tree Gaming Ltd. (2021). Nexus mods.

Blasch, E., Ravela, S., and Aved, A. (2018). *Handbook of Dynamic Data Driven Applications Systems*. Springer.

Bostock, M., Heer, J., and Ogievetsky, V. (2011a). D3.js: The JavaScript library for bespoke data visualization. Last visited: 19/09/2024.

Bostock, M., Ogievetsky, V., and Heer, J. (2011b). D3: Data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309.

Christopher, G. and Mahler, M. (2018). ReactFlow: A customizable React component for building node-based editors and interactive diagrams. Last visited: 18/11/2024.

Consortium, C. (2016). Cytoscape.js: Graph theory (network) library for visualisation and analysis. Last visited: 20/11/2024.

Epic Games, Inc. (2022). Unreal Engine - Behavior Trees.

Google (2018). Flutter: An open source framework for building beautiful, natively compiled, multi-platform applications from a single codebase. Last visited: 18/11/2024.

Kotsiantis, S. B. (2013). Decision trees: a recent overview. *Artificial Intelligence Review*, 39:261–283.

Letzter, R. (2015). Online communities are changing video games to make them better, weirder, and much more wonderful.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1(1):81–106.

Rauch, G. (2016). Next.js: The React Framework for the Web. Last visited: 05/11/2024.

Unity Asset Store (2016). Decision Tree Toolkit, by MorbidCamel.

Yannakakis, G. N. and Togelius, J. (2018). *Artificial Intelligence and Games*. Springer.

Zhao, C. (2013). Electron: Framework for cross-platform desktop applications using web technologies. Last visited: 25/11/2024.