Proceedings of the 2nd International Conference on Machine Learning and Automation
DOI: 10.54254/2755-2721/131/2024.20547

Boosting Cooperative NPC Effectiveness and Player Immersion through Behavior Tree Optimization in Gaming

Fangyu Zhu

Jiangsu University, Zhenjiang, Jiangsu, China

valhalla0130@gmail.com

Abstract. The purpose of this research is to improve the adaptability and intelligence of cooperative non-player characters (NPCs) in dynamic gaming situations. Although behavior trees (BTs) are commonly used to simulate non-player character (NPC) behaviors, their rigid hierarchical design restricts NPC adaptability in complex settings. This study employs a variety of optimization techniques, including evolutionary algorithms and hybrid strategies that combine Artificial Neural Networks (ANN) and Monte Carlo Tree Search (MCTS), to improve NPC adaptability. The findings reveal that these strategies enable flexible behavior transitions, with evolutionary algorithms strengthening BT's flexibility in decision-making scenarios and ANN and MCTS significantly increasing NPC intelligence and response capabilities. Furthermore, we highlight fundamental drawbacks in typical BTs, such as static node topologies, which may impede dynamic adaptation. These discoveries lay the groundwork for improving immersive gaming experiences, expanding the knowledge repository for NPC AI research, and providing vital insights into the development of more intelligent cooperative NPCs.

Keywords: NPC Effectiveness, Tree Optimization, Player Immersion.

1. Introduction

Enhancing the intelligence of game characters has consistently been a crucial focus for both industry professionals and academic researchers in game development and related disciplines. Over time, the application of various AI techniques in games has evolved significantly. Since the early 21st century, BTs have emerged as a pivotal tool for addressing the limitations of Finite State Machines (FSMs). BTs are still widely used in numerous industry applications and continue to influence future game development and academic research [1].

Despite the significant progress in game development, challenges such as unintelligent non-player characters (NPCs) continue to hinder immersion. Research indicates that these NPCs, whether they are part of the narrative or serve a functional role, often exhibit low intelligence and inflexible tactics, which negatively impact player experience. For instance, in games like ”Dark Souls” and ”God of War: Ragnar¨ok,” cooperative NPCs are intended to assist players, but their poor performance in complex scenarios often leads to player frustration [2]. We will examine key studies on cooperative NPCs, focusing on their design, implementation, and impact on player experience. We will particularly focus on the evolution of behavior trees in games and other methods that can improve NPC performance.

Standard behavior tree models, which rely on a hierarchical structure to represent NPC actions, are widely employed in game development. While effective for producing simple behavior based on given

DOI: 10.54254/2755-2721/131/2024.20547

situations, the fixed hierarchical structure generally has limited flexibility and capacity for adapting to changes within a gaming environment. To address these limitations, researchers have turned to optimizing behavior trees through various strategies, including evolutionary algorithms, graph-based methods, and hybrid approaches combining multiple optimization techniques.

Evolutionary algorithms simulate the natural evolution of creatures adapting to their environment to solve complex problems, such as optimizing NPC behaviors, which can be represented using behavior trees under dynamic conditions. Graph-based methods, while inclusive of trees, allow for greater flexibility by representing NPC behavior as a network of interconnected nodes, making it easier to model complex, non-hierarchical relationships and dynamic transitions between behaviors. Additionally, studies have shown that graph-based representations of NPC behavior are beneficial compared to modeling them entirely as trees. For example, Hossain demonstrated the advantages of using graph models to handle complex interactions and real-time adjustments in NPC behavior [3].

An example of combining AI techniques for game development can be seen in the work of Liu [4], who used Monte-Carlo Tree Search (MCTS) algorithms to create intelligent adaptive game opponents. They applied MCTS in a two-dimensional predator/prey game, demonstrating the algorithm's effectiveness in controlling NPCs. By training Artificial Neural Networks (ANN) with data from MCTS, they were able to validate the approach's efficiency. The study concluded that combining MCTS with ANN could create intelligent and adaptive game opponents, offering dynamic difficulty adjustment.

This paper aims to introduce several behavior tree optimization methods and explore directions for optimizing NPC behavior systems to offer more flexibility and dynamism. We will analyze behavior tree models, evolutionary algorithms, graph-based methods, and existing behavior tree optimization techniques. Our goal is to propose specific optimization directions for game developers and researchers and understand how technological advancements can further enhance game AI.

2. Background

2.1. Behaviour tree evolution

Behavior Trees (BTs) make it an intuitive and effective way to organize and manage the behaviors and decisions of non-player characters (NPCs) through a tree structure. Behavior trees are hierarchical decision-making models that represent different behaviors and decision paths through a hierarchy of nodes. Each node can be an action node, a condition node, or a composite node. The execution of a behavior tree starts at the root node and traverses and evaluates each node in turn until an executable behavior is found.

In the structure of a behavior tree, the root node is the starting point for decision-making. Conditional nodes are used to check whether specific conditions are satisfied, action nodes are used to perform specific actions, and composite nodes combine other nodes to form more complex behavioral patterns. The behavior tree allows designers to modularly define and manage the behavior of NPCs so that they can flexibly respond to various dynamic changes and complex scenes in the game [5].

Behavior trees are especially widely used in video games. They not only simplify the design and implementation of complex behaviors but also improve the readability and maintainability of behavior logic. Designers can quickly modify and optimize the behavior of NPCs by adjusting and expanding the nodes in the behavior tree to enhance the interactivity and player experience of the game.

For example, in an adventure game, NPCs may need to react to player actions and environmental changes. The behavior tree can define a series of conditions, such as detecting the distance of the player, the state of the surrounding environment, and the current health of the NPC. When these conditions are evaluated, the behavior tree will decide whether the NPC will attack, run away, or seek cover based on predefined logic.(Like fig:1)

DOI: 10.54254/2755-2721/131/2024.20547

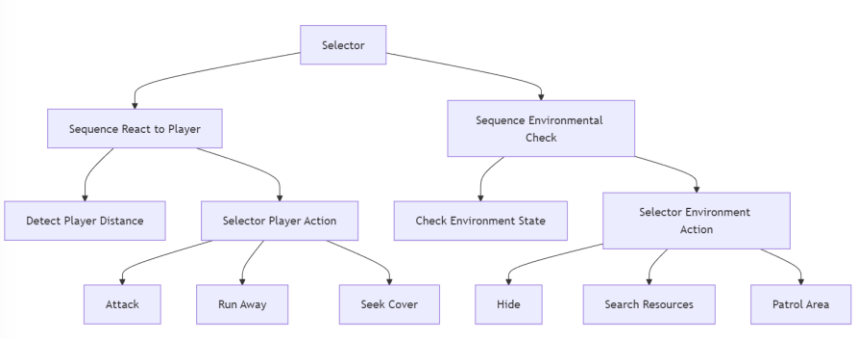

Figure 1. Traditional behavior tree

In addition, the behavior tree supports parallel and sequential execution of behaviors, which allows complex behavioral patterns to be realized by simple node combinations. For example, an enemy NPC can detect both the presence of a player and the obstacles around it, and when it detects a player, it first determines whether there is an obstacle in the way, and if there is, it bypasses the obstacle before pursuing the player.

The Game Developer article provides an in-depth explanation of how behavior trees improve game AI design by simplifying complex behavior patterns and making NPC actions more modular and responsive to player inputs. As described in the article, behavior trees not only enhance the AI's decision-making but also streamline the development process, making NPC behavior more engaging and realistic for players [6].

2.2. GE-evolved Behavior Trees

Grammatical Evolution (GE) is a method that uses genetic algorithms to evolve programs, generating strings that conform to specific rules defined by a grammar, which are then interpreted as program code. In the research by Miguel Nicolau [7], GE is used to automate the evolution of behavior tree structures. This method not only improves the adaptability of behavior trees but also generates efficient solutions specific to the problem. Through this approach, GE-evolved Behavior Trees effectively handle dynamic decision-making problems such as those in video games, showing superior performance compared to traditional behavior trees.

Behavior Trees (BTs) are hierarchical decision-making structures frequently used in game AI (Fig. 2). When evolved using Grammatical Evolution (GE), BTs can automatically adapt to specific challenges by optimizing their structure, allowing for more flexible and efficient decision-making processes in complex environments.

In this approach, the XML syntax of a behavior tree is defined as a collection containing conditions, actions, subtrees, and filters. Although GE provides the flexibility to combine these elements, early experiments have found that unstructured guidelines can lead to confusing tree structures that are difficult to read and inefficient to implement.

To resolve the issues caused by unstructured designs, the syntax of the behavior tree is restricted by grammar. The design of the behavior tree follows an "and-or" tree structure, similar to a binary decision diagram, which is considered to be an efficient way to build a behavior tree for game AI.

Specifically, the behavior tree consists of:

–Root node: acts as a selector and contains multiple Behavior Block (BB) subtrees, each encoding a specific sub-behavior.

–Behavior Block (BB): each behavior block contains a series of conditions, followed by a series of actions or subtrees.

–End Behavior Block: this is the final, unconditional behavior block, which may be a series of actions and subtrees or the default navigation behavior when using the A* algorithm.

DOI: 10.54254/2755-2721/131/2024.20547

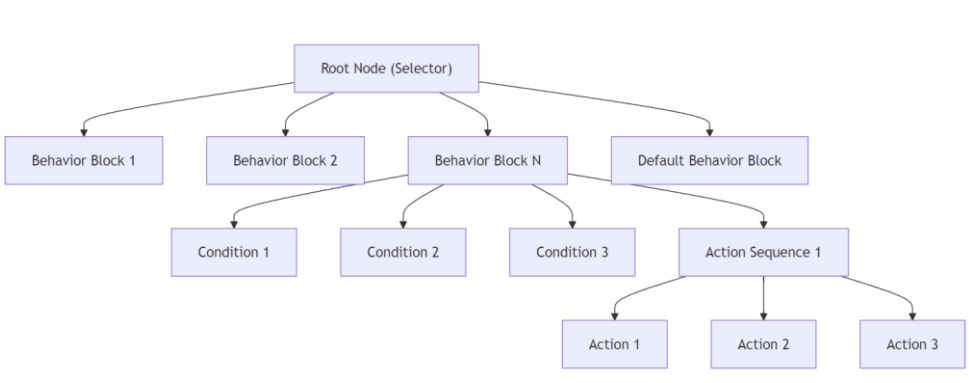

Figure 2. Structure of evolved BTs

At the start of behavior tree execution, the root selector first activates the leftmost behavior block. If the relevant conditions are not satisfied, the execution sequence continues in left-to-right priority order. The number of conditions per behavior block is typically limited to one or two to maintain efficiency and prevent over-complicating the decision-making process. By limiting conditions, the execution remains streamlined, ensuring that behavior trees can operate effectively in real-time environments. Meanwhile, the number of actions and subtrees is unlimited, allowing for greater flexibility in the behavior execution once conditions are met.

The default behavior block is positioned at the end of the sequence, making it the lowest-priority block in the tree. This block is only executed if all other behavior blocks fail to meet their conditions. Depending on the implementation, this block may either contain simple fallback actions or, in cases involving navigation, it may utilize pathfinding algorithms like A* to determine the next course of action.

In Nicolau's research [7], double-point intersections were employed to allow variable numbers of behavior blocks to be exchanged between individuals. This technique, similar to subtree exchanges in genetic programming, provides greater flexibility in evolving behavior trees by allowing more dynamic reconfigurations during the evolutionary process.

2.3. Monte Carlo Tree (MCT)

Monte Carlo Tree Search (MCTS) is a powerful algorithm widely used in artificial intelligence for decision-making, particularly in games. MCTS operates by exploring potential moves in a decision tree through random simulations. Each simulation involves a random sequence of actions from the root node to a leaf node, allowing the algorithm to estimate the expected values of different actions based on their outcomes.

MCTS has proven effective across various game genres, including classic board games like Chess and Go, as well as modern video games. It facilitates dynamic learning and adaptation by enabling game AI to refine its strategies based on player behavior continuously. For instance, in the paper "Monte-Carlo Tree Search: A New Framework for Game AI" by [8], the authors highlight how MCTS can efficiently generate challenging AI behaviors without requiring extensive domain knowledge. Another study discusses how MCTS can adapt to both perfect and imperfect information games, providing a flexible framework for enhancing game AI [9].

3. Monte-Carlo Tree with Artificial Neural Network

Monte Carlo methods are used to estimate value functions by simulating many possible outcomes through random sampling. This approach is well-suited to scenarios where deterministic calculations are either impractical or computationally expensive. MCTS applies Monte Carlo principles to explore a game tree efficiently, balancing exploration and exploitation using strategies such as Upper Confidence Bounds for Trees (UCT). This method evaluates different moves by progressively building a search tree and simulating future actions [10].

DOI: 10.54254/2755-2721/131/2024.20547

3.1. Improving Opponent AI Adaptability with Monte Carlo Algorithms

In games like **"Settlers of Catan"**, Monte Carlo Tree Search (MCTS) is used to enhance the AI's adaptability by simulating various strategies to explore resource management and player interactions. The AI makes decisions on moves such as building settlements or trading resources by selecting legal action from the available options. The possible actions include building, trading, or holding resources, each impacting the long-term strategy.

At each turn, MCTS constructs a search tree with potential moves, ensuring that no strategic options are overlooked. After running multiple simulations, MCTS selects the branch with the highest success probability. The AI then executes the move represented by the winning branch, adjusting dynamically to the evolving game state [11].

3.2. Performance Comparison of Neural Networks and Monte Carlo Algorithms

The computational time consumption of the Monte Carlo method can be controlled by setting computational limits, but its performance cannot be determined in the same way. Due to its randomness and time consumption, it is not easy to adjust its performance. However, it can generate "good" data to defeat unknown strategies. In contrast, ANN performs excellently in efficiency and less time consumption. Even in the most challenging environments, ANN shows absolute advantages in time consumption. In experiments conducted by Liu [4], 1000 threads were constructed to run game boards on a computer. ANN performance remained stable without significant decline. However, the Monte Carlo method could not adapt to this extreme environment.

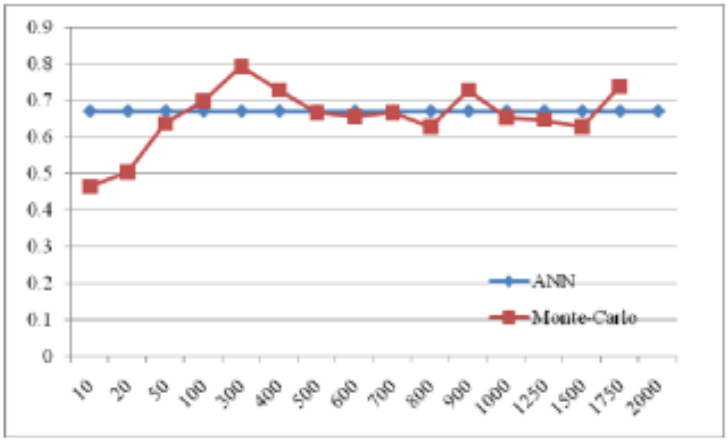

Figure 3. Performance Comparison between MonteCarlo and ANN fighting with same Strategy [4]

His study shows a comparison of the performance of the Monte Carlo method and the ANN under the same strategy. (As shown in Figure 3.) Essentially, the computational time consumption of the Monte Carlo method can be manually controlled, but its performance cannot be determined in the same way. Due to its stochastic nature and time-consuming nature, the programmer cannot easily adjust its performance. However, the Monte Carlo method can generate "good" data to beat the unknown strategy. In contrast, ANN excels in terms of efficiency and time-consumption.

3.3. Combining Monte Carlo Methods and Artificial Neural Networks

By combining Monte Carlo methods and ANN, intelligent adaptive game opponents can be created. First, use the Monte Carlo method to find ways to defeat the player, then train ANN to play against known players, while the Monte Carlo method can use idle resources to handle unknown players. The Monte Carlo method can find optimized solutions for strategies in game development and help find solutions to deal with unknown strategies. Artificial Neural Networks can be trained with data collected from the Monte Carlo method and handle known strategies. It has good correspondence to specific strategies.

DOI: 10.54254/2755-2721/131/2024.20547

In summary, the combination of Monte Carlo Tree Search and Artificial Neural Networks is a powerful and efficient technique that can significantly enhance the intelligence and adaptability of game AI. By continuously optimizing and expanding these two technologies, their advantages can be utilized in a broader range of application scenarios, providing more intelligent and realistic gaming experiences.

4. Conclusion

The integration of Monte Carlo Tree Search (MCTS) with Artificial Neural Networks (ANN) significantly boosts game AI by improving the adaptability and intelligence of non-player characters (NPCs). MCTS, with its robust decision-making through random simulations, is ideal for dynamic environments like real-time strategy games. However, its high computational demands challenge its real-time application. Incorporating ANN, which excels in pattern recognition and efficient processing, mitigates these limitations. Liu's study highlights this hybrid approach's success, where MCTS data trains ANNs to adapt to strategies and manage real-time decisions effectively. This combination maximizes both algorithms' strengths, providing rapid processing power and comprehensive decision-making capabilities, ensuring superior performance in demanding gaming environments. This not only enhances NPC behavior but also deepens player immersion, advancing game AI development.

Additionally, Grammatical Evolution (GE) in optimizing Behavior Trees (BTs) enhances game AI's flexibility and efficiency. GE applies genetic algorithms to evolve BT structures, creating adaptable solutions for dynamic decision-making challenges. This method enables NPCs to adjust to game changes, surpassing traditional BTs automatically. By setting the XML syntax of BTs, GE automates the creation of complex and adaptable behavior patterns. When integrated with MCTS and ANN, GE-evolved BTs equip developers with robust tools to develop intelligent, adaptive NPCs. This approach simplifies game AI development and maintenance and enriches the gaming experience by rendering NPCs more lifelike. As game AI progresses, these sophisticated techniques will be pivotal in the future of interactive entertainment.

References

[1] Yoones A Sekhavat. "Behavior trees for computer games". In: International Journal on Artificial Intelligence Tools 26.02 (2017), p. 1730001.

[2] Dmitrii Iarovoi, Richard Hebblewhite, and Phoey Lee Teh. "AI's Influence on Non-Player Character Dialogue and Gameplay Experience". In: Science and Information Conference. Springer. 2024, pp. 76–92.

[3] Md Yousuf Hossain and Loutfouz Zaman. "NCCollab: collaborative behavior tree authoring in game development". In: Multimedia Tools and Applications 82.3 (2023), pp. 4671–4708.

[4] Xiao Liu et al. "To create intelligent adaptive game opponent by using monte-carlo for the game of Pacman." In: 2009 Fifth International Conference on Natural Computation. Vol. 5. IEEE. 2009, pp. 598–602.

[5] Marek Kopel and Tomasz Hajas. "Implementing AI for non-player characters in 3D video games". In: Intelligent Information and Database Systems: 10th Asian Conference, ACIIDS 2018, Dong Hoi City, Vietnam, March 19-21, 2018, Proceedings, Part I 10. Springer. 2018, pp. 610–619.

[6] Game Developer. Behavior trees for AI: How they work. Accessed: September 24, 2024. 2024. URL: https: / /www.gamedeveloper.com/programming/behavior-treesfor-ai-how-they-work.

[7] Miguel Nicolau et al. "Evolutionary behavior tree approaches for navigating platform games". In: IEEE Transactions on Computational Intelligence and AI in Games 9.3 (2016), pp. 227–238.

[8] Gertjan M. Chaslot et al. "Monte-Carlo Tree Search: A New Framework for Game AI". In: Proceedings of the National Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE). AAAI Press, 2008, pp. 216–221. URL: https://ojs.aaai.org/index.php/AIIDE/article/view/18700.

DOI: 10.54254/2755-2721/131/2024.20547

[9] H. L. Cheng, R. K. G. V. Q. Teixeira, and E. M. F. P. S. M. V. A. Oliveira. “Monte Carlo Tree Search in Imperfect Information Games”. In: arXiv preprint arXiv:2103.04931 (2021). URL: https://ar5iv.org/html/2103.04931.

[10] Author Name(s). “Beyond games: a systematic review of neural Monte Carlo tree search applications”. In: Applied Intelligence 51.5 (2021), pp. 3587–3605. DOI: 10. 1007 / s10489 - 021 - 02288 - w. URL: https: //link.springer.com/article/10.1007/s10489-021-02288-w.

[11] Author 2 Author 1. “Monte Carlo Tree Search: A Review of Recent Modifications and Applications”. In: ar5iv.org (2021). URL: https://ar5iv.labs.arxiv.org/abs/2103.04931.