FindMe: A Prototype Videogame AI based on CTL with an Optimized Synthesis Algorithm

Demonstration Track

Marco Aruta
University of Naples Federico II
Naples, Italy
marco.aruta@unina.it

Vadim Malvone
Télécom Paris
Paris, France
vadim.malvone@telecom-paris.fr

Aniello Murano
University of Naples Federico II
Naples, Italy
aniello.murano@unina.it

Vincenzo Pio Palma
University of Naples Federico II
Naples, Italy
vincenzop.palma@studenti.unina.it

Salvatore Romano
University of Naples Federico II
Naples, Italy
salvatore.romano2@unina.it

ABSTRACT

We present FindMe, a prototype tool that enhances AI-driven decision-making in Non-Player Characters (NPCs) within Unreal Engine 5.4. Our approach utilizes formal verification techniques, specifically Computation Tree Logic (CTL), to enable real-time adaptive behavior in NPCs. Additionally, we employ an innovative model optimization technique to enhance our verification process. Our in-engine implementation of CTL model checking outperforms existing tools like NuSMV, even in more complex scenarios.

KEYWORDS

Gaming; Model Checking; Reactive Synthesis; Strategic Reasoning

ACM Reference Format:
Marco Aruta, Vadim Malvone, Aniello Murano, Vincenzo Pio Palma, and Salvatore Romano. 2025. FindMe: A Prototype Videogame AI based on CTL with an Optimized Synthesis Algorithm: Demonstration Track. In *Proc. of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2025), Detroit, Michigan, USA, May 19 – 23, 2025*, IFAAMAS, 3 pages.

1 INTRODUCTION

Artificial Intelligence (AI) has revolutionized the video game industry but still faces challenges in adapting to dynamic environments and player actions. Despite advances like dynamic scripting [6], reinforcement learning [1], and behavioral cloning [4], many games use simple, repetitive AI that frustrates players: commercial AI systems rely on predefined rules, limiting adaptability and strategic gameplay. Formal methods offer a rigorous framework for verifying system properties, but real-time decision-making in game engines is still challenging. This paper introduces a novel formal verification solution for AI in video games, integrated within Unreal Engine 5. Using Computation Tree Logic (CTL), our method enables real-time decision-making and debugging, allowing adaptable NPC behaviors. This is the first practical integration of CTL in Unreal Engine, providing an in-engine solution that eliminates external tools and simplifies development: previous researches employed only external tools like NuSMV to analyze the correctness of game logic offline [3, 7] or completely simulated environments [5]. We utilize Kripke structures to model gaming environments and support temporal reasoning, enabling NPCs to dynamically react to player actions. Our optimized state evaluation through directed graph analysis, based on state reachability using the A* algorithm, fastens NPC responses. Our approach ensures optimal NPC tactics across various game genres and outperforms NuSMV in tests. This integration empowers developers to create intelligent, adaptable AI agents, providing enhanced strategic interactions and bridging the gap between static behaviors and dynamic gameplay.

Proc. of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2025), Y. Vorobeychik, S. Das, A. Nowé (eds.), May 19 – 23, 2025, Detroit, Michigan, USA. © 2025 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org).

2 THE TOOL

The tool, available on GitHub[1], features an innovative AI framework incorporated within Unreal Engine 5.4, utilizing Computation Tree Logic (CTL) for real-time NPC decision-making. This framework applies formal verification methods, typically reserved for thorough system analysis, to enable NPCs to adjust their behavior in response to the evolving game environment. Key Features: (1) *Adaptive AI*: NPCs intelligently make real-time decisions, adapting fluidly to environmental changes and player actions. (2) *Efficiency*: The tool employs optimized model-checking algorithms to concentrate on pertinent game states, enhancing performance and avoiding unnecessary data handling. (3) *Ease of Use*: From the player's perspective, there is no extra effort to interact with this AI. The tool is integrated into the game seamlessly and represents a significant advancement in the development of context-aware systems.

3 INTEGRATION

Our model checking system is implemented in C++ and integrated in Unreal Engine 5.4 structures and classes. Our testbed game is a strategic asymmetric survival-stealth game featuring two opposing entities: a human-controlled Hunter and two AI-controlled Runners (a demonstration video can be found on Youtube[2]). The Hunter's goal is to eliminate both Runners using their firearm, while the Runners must find one of four potential exits. It takes place on a

[1]https://github.com/VincenzoPalma/FindMe
[2]https://youtu.be/Ujv0Gx6oUaA

labyrinthine map characterized by loops, dead ends, and multiple pathways. This design challenges players to devise and adapt strategies for successful navigation. The Hunter starts in the center of the map, while the Runners begin in a corner. Two to three exits are randomly selected for each game, preventing the Hunter from focusing solely on a single point and adding to the strategic depth of the game. To enhance the efficiency of our model checking system, instead of loading the entire model from a file, we load only the states reachable from the current game state. This ignores irrelevant states, reducing parsing time and ignoring many other states (e.g., NPC1 alive when currently dead) that could be irrelevant. Further efficiency is achieved by a modified A* algorithm prioritizing states most likely to satisfy the formula. The optimization approach includes: (1) *State Exploration*: States are prioritized using a heuristic (h) and path cost (g) combined into f = g + h. The heuristic h reflects how many sub-formulas (representing sub-objectives) of φ the state fails to meet, while g counts how many states are traversed starting from s. (2) *Priority Queue*: States are managed in a priority queue, with those having the lowest f-scores explored first. Visited states are tracked to prevent redundant checks. (3) *Target State Identification*: If a state satisfies φ, the algorithm outputs the path to it. If no solution exists, the queue becomes empty. (4) *Dynamic Loading*: When unexplored states are needed, the model expands to include them, recalculating scores as necessary. This approach accelerates model checking by focusing only on relevant states and leveraging heuristic-driven exploration to find a solution efficiently. (5) *Caching*: A singleton class caches the starting state and formulas to minimize file access and JSON parsing overhead.

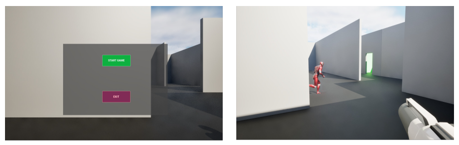

Figure 1: The game into which the tool is integrated.

4 EXPERIMENTS

Testing was conducted on a system with Windows 11, 32GB of DDR4 RAM, an AMD Ryzen 7 5800X octa-core CPU, and an NVIDIA RTX 3070 GPU with 8GB of GDDR6 VRAM. The in-engine model checking system was tested in Unreal Engine 5.4.3, with data collected via Unreal Insights, while NuSMV testing used Windows PowerShell and the Measure-Command command. Each test, repeated 1000 times per formula, ensured that every CTL operator was tested, examples of formula employed are: (1) AG(¬NPC1InDanger ∨ EF(¬NPC1InDanger)); (2) EX(¬NPC1InDanger); (3) EX(¬NPC1InDanger ∧ NPC1NearExit). Tests used a total of three models (with 10, 100, and 250 states obtained by either increasing or decreasing the abstraction of our base game state space), examining the impact of model size and transitions on evaluation time. Formulas were evaluated starting from intermediate states to highlight the optimized system's performance gains. For both in-engine (pre-optimization model, post-optimization model) and NuSMV models were tested formulas with and without nested CTL temporal operators up to one nesting level: the complexity of the formulas does not exceed this level, as CTL formulas in real scenarios typically do not become more complex. The tests focused on response times, which include both model parsing and formula evaluation: parsing, involving file reading and state graph construction, dominated the total time due to single-formula evaluations. NuSMV performs consistently with smaller models, but its performance degrades significantly as the number of states increases because it fully loads and evaluates the entire model. In contrast, the in-engine system, particularly after optimization, scales more effectively by only loading reachable states from the starting point, substantially reducing parsing and evaluation times. The optimized in-engine system outperforms its pre-optimization version, achieving up to 50% faster execution times. Fully connected models with a large number of transitions further highlight the optimized system's advantages, as it focuses only on the necessary parts of the model. Additionally, simpler formulas with fewer subformulas or operators generally result in faster evaluation times compared to complex, nested formulas. However, the optimized system maintains relatively stable execution times due to its efficient design.

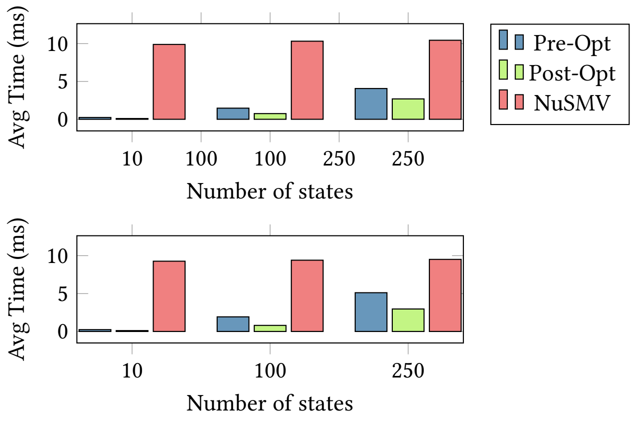

Figure 2: Response times for CTL formulas (In-Engine vs NuSMV) utilizing both EG (1° graph) AG operator (2° graph).

5 CONCLUSIONS

We presented *FindMe*, a tool integrating Computation Tree Logic (CTL) model checking within a real-time game engine to enhance dynamic decision making for NPCs. By leveraging formal verification, FindMe enables adaptive NPC behaviors that dynamically respond in real-time to player actions and environmental changes. Our integration within Unreal Engine 5.4, bypassing external tools like NuSMV, marks a significant advancement in game AI development. However, scalability in large environments remains a challenge, impacting real-time performance. Future work will explore scalability for larger environments and refine optimization techniques, such as state abstraction [2].

ACKNOWLEDGMENTS

This research has been supported by the PRIN project RIPER (No. 20203FFYLK), the PNRR MUR project PE0000013-FAIR, the InDAM 2023 project "Strategic Reasoning in Mechanism Design" and the PNRR MUSA project INFANT.

REFERENCES

[1] Nicolas A. Barriga, Marius Stanescu, Felipe Besoain, and Michael Buro. 2019. Improving RTS Game AI by Supervised Policy Learning, Tactical Search, and Deep Reinforcement Learning. *IEEE Computational Intelligence Magazine* 14, 3 (2019), 8–18. https://doi.org/10.1109/MCI.2019.2919363

[2] Francesco Belardinelli, Angelo Ferrando, Wojciech Jamroga, Vadim Malvone, and Aniello Murano. 2023. Scalable Verification of Strategy Logic through Three-valued Abstraction. arXiv:2310.17219 [cs.MA] https://arxiv.org/abs/2310.17219

[3] Nao IGAWA, Tomoyuki YOKOGAWA, Mami TAKAHASHI, and Kazutami ARIMOTO. 2020. Model Checking of Visual Scripts Created by UE4 Blueprints. In *2020 9th International Congress on Advanced Applied Informatics (IIAI-AAI)*. 512–515. https://doi.org/10.1109/IIAI-AAI50415.2020.00107

[4] Tim Pearce and Jun Zhu. 2021. Counter-Strike Deathmatch with Large-Scale Behavioural Cloning. arXiv:2104.04258 [cs.AI]

[5] Ruslan Rezin, Ilya Afanasyev, Manuel Mazzara, and Victor Rivera. 2018. Model Checking in Multiplayer Games Development. In *2018 IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA)*. 826–833. https://doi.org/10.1109/AINA.2018.00122

[6] Pieter Spronck, Marc Ponsen, Ida Sprinkhuizen-Kuyper, and Eric Postma. 2006. Adaptive game AI with dynamic scripting. *Machine Learning* 63 (2006), 217–248.

[7] Kazuki Wayama, Tomoyuki Yokogawa, Sousuke Amasaki, Hirohisa Aman, and Kazutami Arimoto. 2023. Verifying Game Logic in Unreal Engine 5 Blueprint Visual Scripting System Using Model Checking. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering (ASE '22)*. Association for Computing Machinery, New York, NY, USA, Article 213, 8 pages. https://doi.org/10.1145/3551349.3560505