FindMe Reforged: Temporal Logic AI for Richer Videogame Scenarios

Vadim Malvone[1][a], Aniello Murano[2][b], Vincenzo Pio Palma[2][c] and Salvatore Romano[2][d]
[1]*Télécom Paris, Paris, France*
[2]*University of Naples Federico II, Naples, Italy*
vadim.malvone@telecom-paris.fr, {aniello.murano,salvatore.romano2}@unina.it, vincenzop.palma@studenti.unina.it

Keywords: Gaming, Model Checking, Reactive Synthesis, Strategic Reasoning.

Abstract: This work presents an enhanced version of *FindMe*, a framework for adaptive and formally verified decision-making in Non-Player Characters (NPCs) within Unreal Engine 5.4. Using Computation Tree Logic (CTL), the system not only verifies temporal properties but also synthesizes action policies in real time, addressing a key limitation of traditional model checkers restricted to validation. To improve scalability and responsiveness, heuristic search through A* is combined with systematic exploration via Breadth-First Search (BFS), balancing efficiency with formal guarantees. By reducing computational overhead while preserving strategic depth, the tool enables NPCs to adapt to unpredictable player behavior and complex scenarios. This contribution shifts the focus from static verification to real-time policy synthesis, offering a practical approach for embedding goal-driven and formally grounded AI in modern games.

1 INTRODUCTION

Video game development faces a persistent tension between immersion and computational feasibility: players expect Non-Player Characters (NPCs) to display strategic depth and adaptive behaviors, yet commercial implementations often rely on static scripts or predefined rules (Duarte et al., 2020). These techniques provide immediate responses but struggle to maintain coherence as scenarios become more complex or player actions diverge from expected patterns, making NPCs appear predictable and reducing long-term engagement (Saini et al., 2021). Achieving dynamic reasoning requires balancing expressiveness, efficiency, and verifiability; while machine learning supports adaptivity, it rarely ensures consistency or safety, whereas traditional formal verification offers guarantees but is typically limited to offline analysis external to the game engine (Rezin et al., 2018). This gap constrains real-time responsiveness and adaptation to unexpected player strategies, both crucial for maintaining challenge andF immersion in modern, increasingly complex games.

Recent advances have begun to bridge this divide by showing that temporal logic-based policy synthesis can be executed directly within modern engines (Aruta et al., 2025), enabling action strategies that are both verified and adaptive at runtime. Building on this foundation, the present work extends the approach through heuristic planning techniques and lightweight player models, improving responsiveness while preserving the rigor of formal reasoning. These enhancements establish the basis for a detailed exploration of the mechanisms introduced here: by integrating real-time temporal reasoning, player modeling, and heuristic planning, the framework addresses prior limitations and enables NPCs to exhibit more nuanced and strategically responsive behaviors. The following sections outline the formal foundations, implementation strategies, and experimental evaluation that demonstrate the effectiveness and scalability of this approach across diverse game scenarios.

Our Contribution. This work presents an upgraded version of the previously introduced *FindMe* framework, extending its formal verification core from concurrent settings to support turn-based gameplay, a natural specialization of concurrent interaction. While this setting formally simplifies the interaction model, the experimental scenario introduces a higher level of complexity by allowing both players and NPCs to choose among up to four possible actions at each turn. This significantly enlarges the branching factor of the underlying model and makes the search space degen-

[a] https://orcid.org/0000-0001-6138-4229
[b] https://orcid.org/0000-0003-4876-3448
[c] https://orcid.org/0009-0009-7251-1452
[d] https://orcid.org/0009-0007-4640-8705

erate more rapidly with depth, thereby amplifying the computational challenge. The contribution lies in enhancing the original integration of CTL-based reasoning within Unreal Engine (UE) by combining it with search techniques tailored to turn-based contexts. Specifically, Breadth-First Search (BFS) addresses long-term strategies like survival over successive turns, whereas A* is applied to optimize short-term objectives, such as enhancing damage dealt to the opponent, balancing strategic planning with tactical efficiency. Furthermore, a lightweight player-modeling mechanism refines NPC adaptability by tracking the frequency of selected player actions and anticipating potential strategies without requiring external machine learning components. By building on the verified decision-making architecture of the previous framework, this version broadens its applicability to more complex scenarios, ensuring intelligent and context-sensitive NPC behavior.

2 RELATED WORKS

Multi-Agent Systems in Video Games Previous research has investigated the use of formal logic both in video games and in AI more broadly (Aruta et al., 2024), mainly through external tools and offline verification (Rezin et al., 2018) applied to game mechanics (Igawa et al., 2020; Wayama et al., 2022). When combined with Multi-Agent Systems (MAS) (Parasumanna Gokulan and Srinivasan, 2010; Wooldridge, 2009; Cermák et al., 2018; Murano et al., ; Jamroga et al., 2019; Jamroga and van der Hoek, 2004), formal logic provides a robust framework for modeling the decision-making architectures of NPCs. Several studies have explored the integration of MAS into game engines to assess their expressiveness, functionality, and computational overhead. For instance, MAS-based architectures within game engines were investigated in (Chover et al., 2020) and (Marín-Lora et al., 2020). Lora et al. proposed an innovative implementation of *Tetris* in which each game piece operates as an independent agent (Marín-Lora et al., 2022). Related agent-based approaches had previously been explored in Real-Time Strategy games through the use of potential fields (Hagelbäck et al., 2009). Reinforcement learning techniques have also been employed to derive optimal control strategies in distributed leader–follower MAS settings (Moghadam and Modares, 2018). Moreover, persistent monitoring problems using mobile agents have been studied with the goal of reducing uncertainty and ensuring mission coverage (Song et al., 2014), with further developments in (Mazouchi et al., 2017; Qiu et al., 2023).

Learning and In-Engine AI In two-player games, studies have explored cryptographic techniques and strategic communication mechanisms. Urbano et al. developed a protocol for reaching a correlated equilibrium in two-person games with rational payoffs (Urbano and Vila, 2002). Farina et al. addressed the computation of Extensive-Form Perfect Equilibrium (EFPE) even during execution uncertainty (Farina and Gatti, 2017). Jia et al. analyzed Q-learning for approximating Nash equilibrium in two-player stochastic games (Jia et al., 2019). Agent coordination within MAS has also been extensively researched: Kapetanakis et al. showed that Q-learning can enable effective collaboration with an FMQ-trained agent for optimal joint actions (Kapetanakis and Kudenko, 2003). Yu et al. (Yu et al., 2022) demonstrated strong performance in cooperative MAS scenarios using Proximal Policy Optimization algorithms. Individual AI agents in games have also been a significant focus. Evolutionary behavior tree algorithms were explored in previous work (Zhang and Xu, 2022). Various game genres have been investigated using deep reinforcement learning, including first-person shooters (Tastan et al., 2012; Lample and Chaplot, 2016), real-time strategy games (Barriga et al., 2019), and fighting games (Kim et al., 2020). The potential of case-based reasoning systems in real-time strategy games was analyzed (Ontanón and Ram, 2011; Jaidee et al., 2011), with the k-nearest neighbor algorithm applied to counteract adversaries (Yamamoto et al., 2014). This study builds upon the in-engine AI framework presented in (Aruta et al., 2025), which employed CTL model checking for adaptive NPC decision-making within UE.

3 BACKGROUND

In this section we recall CTL. An interested reader can refer to (Clarke et al., 1986; Jamroga and Murano, 2014; Urban et al., 2018). CTL formulas are evaluated within Kripke Structures, which are labeled transition systems representing the possible states and transitions of a system. The next section presents a rigorous definition.

3.1 Kripke Transition Model

To define CTL, we first need to introduce the Kripke Transition Model, a mathematical structure that represents the states and transitions within a system. It provides the foundation upon which the semantics of CTL are built, allowing us to evaluate how properties evolve over time across different possible execution

paths. Formally, a Kripke Transition Model M is defined as a tuple:

$$M = (S, \mathcal{R}, L)$$

where:

- S is the set of possible states. Each state represents a configuration of the system in a given moment.
- $\mathcal{R}$ is the transition relation. It is a binary relation defined as $\mathcal{R} \subseteq S \times S$. For any states $s, s' \in S$, the pair $(s, s') \in \mathcal{R}$ indicates that there is a transition from state s to state s'.
- L is the labeling function. It is defined as $L : S \to \mathcal{P}(AP)$, where $\mathcal{P}(AP)$ is the power set of the set of atomic propositions AP. It maps each state $s \in S$ with the set of atomic propositions that are satisfied in that state.

We now present the precise syntax of CTL.

3.2 CTL Syntax

It is a formalism used to describe and verify the temporal aspects of computational systems. It is specifically designed to handle systems where the future is not a single linear sequence but rather a branching tree of possible future states. This branching nature is central to how CTL operates and is crucial for reasoning about complex systems. It employs *path quantifiers* to reason about different possible future executions of a system, providing a framework for expressing how certain conditions can evolve over time, which is particularly relevant in the environments of video games. By applying CTL in game AI, it is possible to ensure that agents and systems behave consistently and predictably, even in complex and unpredictable game worlds. The formulas in CTL are defined in two classes: *state formulas* and *path formulas*.

$$\phi ::= a \mid \neg\phi \mid \phi \wedge \phi \mid A\varphi \mid E\varphi \quad \text{(state formulas)}$$

$$\varphi ::= X\phi \mid G\phi \mid \phi U \phi \quad \text{(path formulas)}$$

- **State formulas** (ϕ) are evaluated at individual states. Atomic propositions a belong to this class.
- **Path formulas** (φ) describe properties of infinite paths starting from a given state.
- Temporal operators; X (*Next*, meaning the property holds in the next state), G (*Globally*, meaning the property holds in every state along a path) and U (*Until*, meaning one property must hold until another does) apply only to path formulas.
- Path quantifiers; A (universal quantifier, *for all paths*) and E (existential quantifier, *there exists a path*).

For example, $\mathbf{AG}\phi$ means that ϕ holds in all future states of all paths, and $\mathbf{EX}\phi$ means that there exists a path where ϕ holds in the next state[1].

3.3 CTL Semantics

The semantics of CTL formulas is established through a satisfaction relation $\models$, which connects program states with the formulas. For a program state s, we write $s \models \phi$ if and only if the formula ϕ holds in the program state s. The semantics for CTL state formulas is formally given as follows:

$$\begin{aligned}
s \models a &\iff a \in L(s) \\
s \models \neg\phi &\iff s \not\models \phi \\
s \models \phi_1 \wedge \phi_2 &\iff s \models \phi_1 \text{ and } s \models \phi_2 \\
s \models \phi_1 \vee \phi_2 &\iff s \models \phi_1 \text{ or } s \models \phi_2 \\
s \models \mathbf{A}\varphi &\iff \forall \text{ paths } \sigma \text{ starting from } s : \\
&\qquad \text{such that } \sigma \models \varphi \\
s \models \mathbf{E}\varphi &\iff \exists \text{ path } \sigma \text{ starting from } s : \\
&\qquad \sigma \models \varphi
\end{aligned}$$

The semantics of path formulas φ is defined as follows:

$$\begin{aligned}
\sigma \models \mathbf{X}\phi &\iff \sigma[1] \models \phi \\
\sigma \models \mathbf{G}\phi &\iff \forall i \geq 0, \sigma[i] \models \phi \\
\sigma \models \phi_1 \,\mathbf{U}\, \phi_2 &\iff \exists i \geq 0, \sigma[i] \models \phi_2 \text{ and} \\
&\qquad \forall 0 \leq j < i, \sigma[j] \models \phi_1
\end{aligned}$$

Whereas $\sigma = (s_0, s_1, s_2, \dots)$ is called *path*, an infinite sequence of program states such that each consecutive pair (s_i, s_{i+1}) is related by the transition relation of the underlying system. The notation $\sigma[i]$ denotes the state at position i along the path. This formal definition captures the meaning of the CTL operators across different branches of possible future executions.

3.4 Model Checking

Model checking is a formal verification technique that provides a systematic approach to ensuring that a system operates in accordance with its specifications. By evaluating whether a model satisfies given properties or constraints, this method offers a rigorous framework to verify correctness and reliability. In the context of game AI, where decision making is inherently complex and often constrained by strategic objectives, model checking can be used to assess whether the behavior of the AI complies with the intended rules and goals across different scenarios.

[1]For a more in depth analysis of CTL syntax and semantics, please refer to (Clarke et al., 1986).

Let $M = (S, \mathcal{R}, L)$ be a Kripke transition model, a state q in M, and a temporal logic formula ψ. Algorithmically, model checking can be defined in two main ways:

- **Local Model Checking**: applied to a single state q of the model M, it evaluates whether the formula ψ holds in that state, returning a Boolean result.
- **Global Model Checking**: applied to the entire model M, it evaluates ψ across all states and returns the set of states in which the property holds.

Global model checking plays a central role in the design and validation of game AI strategies, as it enables the systematic identification of all states in which specific properties or objectives are satisfied (Rezin et al., 2018; Wayama et al., 2022). This comprehensive evaluation is particularly valuable for decision making, since it allows the development of strategies that remain consistent across all possible scenarios. However, a fully exhaustive analysis can be computationally demanding, especially in highly dynamic environments such as video games. To address this challenge, an effective compromise can be obtained through depth-limited model checking. By restricting the exploration to a bounded depth of the state space, this approach balances between the completeness of global model checking and the efficiency of local verification. In practice, it provides sufficient insight into the relevant behaviors of the system while maintaining computational tractability, making it well-suited for real-time decision making in interactive and resource-constrained settings.

4 IMPLEMENTATION

This section describes the implementation of the framework within the UE environment and illustrates how temporal reasoning, search, and player modeling are combined to produce verified, adaptive NPC decisions in a turn-based decision cycle. The implementation is organized around an in-engine CTL model checker, a dual search strategy for policy synthesis, and a compact player model used to bias successor generation; these components interact at each turn so that action sequences satisfying temporal objectives are synthesized and evaluated within the constrained horizon of the current decision step.

4.1 The Use of CTL

CTL is adopted as the temporal formalism because it naturally expresses existential and universal properties over branching futures, which are intrinsic to turn-based decision-making. In this context, existential quantification captures the availability of favorable action combinations, while universal quantification supports reasoning about guarantees that must hold against all possible opponent responses within a turn. Rather than evaluating the global model of the game, CTL reasoning is applied online at each turn through depth-limited exploration starting from the current state, allowing policy synthesis to remain both expressive and computationally feasible despite the combinatorial growth induced by joint actions.

4.2 Integration

The framework has been embedded into Unreal Engine 5.4 as a native C++ module and is publicly available on GitHub[2]. Integration is designed to preserve the engine responsiveness while executing a bounded forward exploration at every decision point of the turn cycle. To validate the pipeline and measure runtime behavior a dedicated turn-based game named *LogicFighters* was implemented. In the game two agents select one of three classes (Warrior, Ranger, or Mage) and, at each turn, both choose simultaneously one among up to four move types: a basic attack that upgrades to a special attack when the special-meter is full, a defensive action, a counter attack, or a self-buff. To make online reasoning tractable, nine Kripke transition structures, one per class pairing, are generated offline by a Python utility and loaded at startup; at runtime the model checker always treats the current game state as the root and explores only a bounded subtree of depth d, synthesizing plans that the NPC can commit to or re-evaluate at subsequent turns. Policy synthesis alternates between two complementary search modalities according to the objective under consideration. When objectives demand immediate achievement, as when maximizing short-term inflicted damage, an A* search is employed using an evaluation function $f(s) = g(s) + h(s)$, where g denotes the depth from the current root and h estimates the remaining unsatisfied CTL sub-formulas; states are managed in a priority queue to favor promising short-horizon solutions. Conversely, when objectives benefit from postponement, for example when survivability across several turns is prioritized, breadth-first search is preferred because it naturally surfaces more distant satisfying states and therefore supports plans that defer risky moves. This hybrid arrangement reconciles tactical immediacy and strategic depth while keeping exploration bounded and predictable within the per-turn decision process.

[2]https://github.com/VincenzoPalma/FindMe

4.3 Player Modeling and Plan Validation

To bridge abstract reasoning and concrete opponent behavior, a lightweight player-modeling component is updated during each encounter and reinitialized at the start of a new match. This module records recent player actions and assigns higher likelihood to more frequent or recent moves, using these likelihoods to bias successor generation during the bounded search and strategy planning; the goal is to anticipate the most probable opponent responses without incurring the cost of heavy machine learning pipelines. At the beginning of each turn the NPC synthesizes a short action sequence compatible with the current model and the selected CTL objective; prior to executing the next action the system verifies that the remaining plan still matches the player behavior observed so far and that the CTL goal remains satisfiable within the bounded horizon. If a mismatch or unsatisfiability occurs, synthesis is re-invoked from the updated state and the NPC replans according to the updated model.

5 THE MODEL

This section describes the model underlying the framework: to formalize the decision-making problem, we define a compact state-action representation that captures the core mechanics of the combat scenario. It specifies how resources such as hit points, special power, and temporary effects evolve through player-NPC interactions, providing the structural basis for reasoning within the engine.

5.1 State Representation

Each game state is represented as a pair of feature vectors, one for the player and one for the NPC:

$$\langle \mathbf{s}_P \,;\, \mathbf{s}_N \rangle, \qquad \mathbf{s}_i = (\mathrm{HP}_i, \mathrm{SP}_i, \mathrm{Def}_i, \mathrm{AntiDef}_i, \mathrm{Buff}_i)$$

where HP and SP denote hit points and special power (integers), Def indicates the availability of a defensive action, AntiDef the availability of an action specifically effective against an opponent's defense (Boolean), and Buff encodes the remaining duration of temporary effects (integer), for $i \in \{P, N\}$.

5.2 Action Encoding

Actions are represented internally by small integer codes for efficiency while textual labels are used for exposition:

```
0 : attack
1 : defense
2 : counter_defense
3 : buff
4 : special_attack
```

5.3 Transition Example

To illustrate the semantics of state transitions and how paired actions affect the branching structure, consider a compact example in which Player and NPC start with identical resources:

⟨60-0-true-true-0 — 60-0-true-true-0⟩

Step 1 - First level transitions. From this state, two illustrative outcomes could be:

- `[Player: attack, NPC: defense]` → The Player's attack is mitigated by the NPC's defense, reducing nominal damage from eight to four and producing NPC HP = 56.
- `[Player: buff, NPC: attack]` → The NPC deals eight damage (Player HP = 52), while the Player activates a buff that subtracts four points from subsequent damage for the next two turns.

Step 2 - Second level transitions. From the second outcome further transitions include:

- `[Player: defense, NPC: attack]` → The buff reduces incoming damage to four and the defensive move halves the remainder, so the Player sustains two HP of loss.
- `[Player: attack, NPC: counter_defense]` → The Player inflicts eight damage while the NPC's counter effect, which would normally reflect a reduced amount, is absorbed by the Player's active buff.

These examples show how paired actions drive per-turn state evolution and generate the bounded branching structure that the in-engine model checker explores during policy synthesis.

6 EXPERIMENTS

This section presents the experimental evaluation designed to assess multiple complementary aspects of the framework: the effectiveness of lightweight player modeling when matches provide only a brief interaction window, the impact of depth-restricted model checking on runtime performance, and a direct comparison between our CTL-based approach and a traditional Behavior Tree (BT) baseline.

Experimental Setup All experiments were conducted in the *LogicFighters* prototype; each match lasted on average twelve turns per player, and player models were reinitialized at the start of every encounter, so any adaptation obtained from modeling was strictly intra-match. Specifically, the experimental protocol combines player modeling strategies with a systematic sweep over depth limits and corresponding CTL thresholds, under the hypothesis that model-checking depth and quantitative goal parameters must be considered jointly to obtain meaningful outcomes. Concretely, three lightweight modeling strategies were evaluated: frequency-based profiling, short-term memory with exponential decay, and heuristic clustering into coarse behavioral labels. These models are updated online during a match and are used solely to bias successor generation in the policy synthesis phase. For the model-checking component, the depth limit d of the forward exploration was varied across the representative set $d \in \{3,5,6\}$. Each depth value was paired with one or more thresholds T in the CTL objectives whenever those objectives were quantitative. For instance, a formula such as:

$$\mathrm{EF}_{\leq d}\ (\Delta hp \geq T)$$

expresses that within d steps there exists a path leading to a state in which the NPC's health delta (remaining health) is at most T. The interaction between depth and threshold is critical: if the depth increases while the threshold remains fixed, the formula may already be satisfied at the minimal sufficient depth, leading to redundant exploration with no new outcome. Conversely, if the threshold is relaxed without increasing the depth, the objective can become trivially satisfied within the already explored horizon. Only by adjusting both parameters jointly can the evaluation preserve a non-trivial balance between feasibility and computational effort. A second line of experimentation investigates the comparison between CTL-based planning and a BT AI to evaluate whether the symbolic reasoning of our approach can produce tangible advantages over standard hierarchical structures. In this setting, the goal is not limited to raw performance but extends to qualitative differences in strategy that emerge across matches. BTs are well known for execution speed and simplicity, yet they rely on predefined scripts with limited adaptability. By contrast, CTL reasoning explores possible futures in relation to explicitly encoded objectives, enabling more tactical behavior and potentially more balanced match dynamics. Quantitative measures include the average planning time per turn and memory usage during exploration, which together provide a picture of runtime overhead. In parallel, qualitative assessment focused on the observable differences in NPC decision-making induced by the alternative player models and by the CTL versus BT distinction, in order to verify whether the reasoning and modeling assumptions translate into distinct strategic choices.

Search Space Growth From a computational standpoint, the branching factor of the search space explains why depth values greater than six were impractical. At each turn, each player can select up to four possible actions, so in a two-player setting the joint branching factor is bounded by $4 \times 4 = 16$ possible action combinations per turn. Consequently, every node in the search tree may have up to 16 successors, and this process compounds at each level of exploration. Already at depth $d = 3$ the worst-case search space reaches $16^3 = 4096$ nodes, at depth $d = 5$ it explodes to $16^5 \approx 1.05 \times 10^6$ nodes, and at depth $d = 6$ it grows further to $16^6 \approx 1.68 \times 10^7$ nodes. This exponential growth rapidly degenerates into intractability, causing the exploration to stall at larger depths, which justifies the decision to cap the evaluation at $d = 6$.

7 RESULTS

The experimental evaluation produced a set of quantitative and qualitative findings that illustrate both the potential and the limitations of our approach. Overall, the results highlight how the interaction between depth and threshold strongly influences the feasibility of the synthesis process, while the comparison with the BT baseline provides insights into the strategic differences induced by symbolic reasoning. Regarding survival objectives, empirical runs confirm that differences in outcome become visible only under specific depth–threshold alignments. With shallow horizons ($d = 3$), survival goals were typically achieved with health differences between 0 and 12 hit points, while extending the horizon to $d = 5$ allowed survival with deltas between 12 and 24 hit points. At $d = 6$, survival margins could reach 24 to 36 hit points, but the computational overhead became increasingly severe. In fact, while runs with $d = 3$ averaged around one hundred milliseconds and $d = 5$ remained under one second, at $d = 6$ exploration already required several seconds and frequently exhibited stuttering, making execution unstable. Beyond this depth, experiments were not feasible, as the exponential growth of the search space rendered computation intractable. These outcomes confirm that the growth of the branching factor observed in the design phase translated into practical limitations in runtime execution. Table 1 report the average planning times per configuration, with ranger policies exhibiting con-

sistently higher costs compared to warrior and mage due to their broader action patterns.

Table 1: Depth sensitivity with BFS (furthest-goal policy). Times are average planning time per turn [ms].

Depth d	Avg time (Warrior, Mage)	Avg time (Ranger)	Remarks
3	110	420	baseline, stable execution
5	740	970	noticeable slowdowns
6	7 500	8 400	stuttering, practical limit

Beyond raw timing, the evaluation also revealed the practical role of lightweight player modeling. Since it is reset at each new encounter, adaptation remained bounded to the short horizon of a single match and therefore did not accumulate across sessions. Under these conditions, the influence of modeling was modest but consistent: NPCs exhibited a tendency toward greater coherence in action selection, avoiding erratic shifts and producing more reasonable patterns. While this did not substantially alter high-level outcomes, it provided qualitative improvements in the form of smoother match dynamics. Lastly, the direct comparison between CTL-based planning and BT execution confirmed the trade-offs expected from the two paradigms. BTs maintained their well-known advantage in terms of responsiveness and negligible overhead, while CTL reasoning introduced additional runtime costs but compensated with richer tactical exploration and more varied strategies. Over repeated matches (500 runs), CTL-based agents displayed a broader range of situationally adapted behaviors, whereas BT agents tended to reproduce a limited repertoire of responses. The results thus substantiate the claim that symbolic reasoning, even when bounded by shallow depths, can generate perceptible strategic diversity without destabilizing the gameplay experience. As shown in Table 2, win counts remain closely balanced between BT and CTL, confirming that our approach does not sacrifice competitiveness. At the same time, the qualitative analysis indicates that CTL strategies diversify decision patterns, especially in the case of Warrior and Mage classes, which exploit their tactical range more effectively under symbolic reasoning than under BT scripting.

Table 2: Match outcomes over 500 games: BT vs CTL.

Outcome	Warrior	Mage	Ranger
BT Wins	170	160	150
CTL Wins	175	165	160
Draws	155	175	190

8 CONCLUSIONS

We presented an enhanced version of *FindMe*, a framework integrating CTL model checking within a real-time game engine to support adaptive and formally grounded NPC decision making. Building on previous work, the current improvements enable NPCs to reason about possible futures more efficiently, producing behaviors that are more flexible, tactically varied and responsive to player actions. Scalability in large or complex environments remains a challenge, affecting real-time performance; future work will investigate state-space abstraction, selective action pruning, heuristic-guided exploration to optimize CTL reasoning and richer player modeling to support longer-term adaptive behaviors. Overall, the enhanced *FindMe* further demonstrates that temporal logic reasoning can be embedded directly in commercial-grade engines highlighting promising directions for integrating formal methods in interactive entertainment (Belardinelli et al., 2023).

ACKNOWLEDGEMENTS

This research has been partially supported by the PRIN project RIPER and the PNRR projects INFANT, APLAND, and FAIR.

REFERENCES

Aruta, M., Malvone, V., Murano, A., Pio Palma, V., and Romano, S. (2025). Findme: A prototype videogame ai based on ctl with an optimized synthesis algorithm. In *Proc. of the 24th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS'25)*, pages 2997–2999.

Aruta, M., Murano, A., and Romano, S. (2024). Atl for dynamic gaming environments. In *European Conference on Multi-Agent Systems*, pages 129–137. Springer.

Barriga, N. A., Stanescu, M., Besoain, F., and Buro, M. (2019). Improving rts game ai by supervised policy learning, tactical search, and deep reinforcement learning. *IEEE Comp. Intel. Magazine*, 14(3):8–18.

Belardinelli, F., Ferrando, A., Jamroga, W., Malvone, V., and Murano, A. (2023). Scalable verification of strategy logic through three-valued abstraction.

Cermák, P., Lomuscio, A., Mogavero, F., and Murano, A. (2018). Practical verification of multi-agent systems against slk specifications. *Inf. Comput.*, 261:588–614.

Chover, M., Marín, C., Rebollo, C., and Remolar, I. (2020). A game engine designed to simplify 2d video game development. *Multimedia Tools and Applications*, 79:12307–12328.

Clarke, E. M., Emerson, E. A., and Sistla, A. P. (1986). Automatic verification of finite-state concurrent sys-

tems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 8(2):244–263.

Duarte, F. F., Lau, N., Pereira, A., and Reis, L. P. (2020). A survey of planning and learning in games. *Applied Sciences*.

Farina, G. and Gatti, N. (2017). Extensive-form perfect equilibrium computation in two-player games. In *Proc. of the AAAI Conf. on Artificial Intelligence*, volume 31.

Hagelbäck, J., Johansson, S. J., et al. (2009). A multiagent potential field-based bot for real-time strategy games. *Int. Journal of Computer Games Technology*, 2009.

Igawa, N., Yokogawa, T., Takahashi, M., and Arimoto, K. (2020). Model checking of visual scripts created by ue4 blueprints. In *9th Int. Congress on Advanced Applied Informatics (IIAI-AAI)*, pages 512–515.

Jaidee, U., Muñoz-Avila, H., and Aha, D. (2011). Case-based learning in goal-driven autonomy agents for real-time strategy combat tasks.

Jamroga, W., Malvone, V., and Murano, A. (2019). Natural strategic ability. *Artif. Intell.*, 277.

Jamroga, W. and Murano, A. (2014). On module checking and strategies. In *Proc. of Int. Conf. on Autonomous agents and multi-agent systems (AAMAS'14)*, pages 701–708.

Jamroga, W. and van der Hoek, W. (2004). Agents that know how to play. *Fundam. Informaticae*, 63(2-3):185–219.

Jia, Z., Yang, L. F., and Wang, M. (2019). Feature-based q-learning for two-player stochastic games.

Kapetanakis, S. and Kudenko, D. (2003). Reinforcement learning of coordination in heterogeneous cooperative multi-agent systems. In *Symposium on Adaptive Agents and Multi-agent Systems*, pages 119–131.

Kim, D.-W., Park, S., and Yang, S.-i. (2020). Mastering fighting game using deep reinforcement learning with self-play. In *IEEE Conf. on Games (CoG'20)*, pages 576–583.

Lample, G. and Chaplot, D. S. (2016). Playing FPS games with deep reinforcement learning. *CoRR*, abs/1609.05521.

Marín-Lora, C., Chover, M., and Sotoca, J. M. (2022). A multi-agent specification for the tetris game. In *Distributed Computing and Artificial Intelligence, Volume 1: 18th Int. Conference 18*, pages 169–178. Springer.

Marín-Lora, C., Cercós, A., Chover, M., and Sotoca, J. (2020). *A First Step to Specify Arcade Games as Multi-agent Systems*, pages 369–379.

Mazouchi, M., Naghibi-Sistani, M. B., and Sani, S. K. H. (2017). A novel distributed optimal adaptive control algorithm for nonlinear multi-agent differential graphical games. *IEEE/CAA Journal of Automatica Sinica*, 5(1):331–341.

Moghadam, R. and Modares, H. (2018). Resilient adaptive optimal control of distributed multi-agent systems using reinforcement learning. *IET Control Theory & Applications*, 12(16):2165–2174.

Murano, A., Perelli, G., and Rubin, S. Multi-agent path planning in known dynamic environments. In *18th Int. Conf. PRIMA 2015*, LNCS 9387, pages 218–231.

Ontanón, S. and Ram, A. (2011). Case-based reasoning and user-generated artificial intelligence for real-time strategy games. In *Artificial Intelligence for Computer Games*, pages 103–124. Springer.

Parasumanna Gokulan, B. and Srinivasan, D. (2010). *An Introduction to Multi-Agent Systems*, volume 310, pages 1–27.

Qiu, S., Li, Z., Pang, Z., Li, Z., and Tao, Y. (2023). Multi-agent optimal control for central chiller plants using reinforcement learning and game theory. *Systems*, 11(3):136.

Rezin, R., Afanasyev, I., Mazzara, M., and Rivera, V. (2018). Model checking in multiplayer games development. In *32nd IEEE Int. Conf. on Advanced Information Networking and Applications (AINA'18)*, pages 826–833.

Saini, F., Sharma, T., and Madan, S. (2021). A comparative analysis of expert opinions on artificial intelligence: Evolution, applications, and its future. *Advanced Journal of Graduate Research*.

Song, C., Liu, L., Feng, G., and Xu, S. (2014). Optimal control for multi-agent persistent monitoring. *Automatica*, 50(6):1663–1668.

Tastan, B., Chang, Y., and Sukthankar, G. R. (2012). Learning to intercept opponents in first person shooter games. *IEEE Conference on Computational Intelligence and Games (CIG'12)*, pages 100–107.

Urban, C., Ueltschi, S., and Müller, P. (2018). Abstract interpretation of ctl properties. In *Static Analysis: 25th Int. Symposium, SAS 2018, Freiburg, Germany, August 29–31, 2018, Proc. 25*, pages 402–422. Springer.

Urbano, A. and Vila, J. E. (2002). Computational complexity and communication: Coordination in two–player games. *Econometrica*, 70(5):1893–1927.

Wayama, K., Yokogawa, T., Amasaki, S., Aman, H., and Arimoto, K. (2022). Verifying game logic in unreal engine 5 blueprint visual scripting system using model checking. In *Proc. of the 37th IEEE/ACM Int. Conference on Automated Software Engineering*, pages 1–8.

Wooldridge, M. (2009). *An introduction to multiagent systems*. John wiley & sons.

Yamamoto, K., Mizuno, S., Chu, C. Y., and Thawonmas, R. (2014). Deduction of fighting-game countermeasures using the k-nearest neighbor algorithm and a game simulator. In *IEEE Conf. on Computational Intelligence and Games*, pages 1–5.

Yu, C., Velu, A., Vinitsky, E., Gao, J., Wang, Y., Bayen, A., and Wu, Y. (2022). The surprising effectiveness of ppo in cooperative multi-agent games. *Advances in Neural Information Processing Systems*, 35:24611–24624.

Zhang, K. and Xu, C. (2022). A evolutionary behavior tree ai for neural mmo challenge. In *4th Int. Conference on Artificial Intelligence and Advanced Manufacturing (AIAM'22)*, pages 327–332.