ATL for Dynamic Gaming Environments

Marco Aruta[1], Aniello Murano[1], and Salvatore Romano[1]

University of Naples Federico II, Naples, Italy

Abstract. The integration of AI has become increasingly important in the gaming industry. However, it faces limitations when dealing with dynamic environments and unpredictable players. As a solution, in this paper we introduce a novel gaming framework that makes use of ATL, a powerful logic for formal strategic reasoning. ATL seamlessly aligns with Multi-Agent Systems, effectively mirroring the decision-making process of Non-Player Characters. With its key advantage of being built on Concurrent Game Structures and employing temporal operators, ATL efficiently models game state sequences. This paper introduces a groundbreaking in-engine, real-time AI method using ATL directly within Unreal Engine 5 blueprints, eliminating the need for external tools.

1 Introduction

Artificial intelligence (AI) has emerged as a transformative force in the video game industry, yet its potential remains largely untapped. Although AI techniques have evolved from traditional approaches to more sophisticated methods such as dynamic scripting [23], reinforcement learning [3], and behavioral cloning [19], the industry continues to grapple with the limitations of current AI implementations, particularly in adapting to dynamic environments and player actions. Many video games still rely on simplistic and predictable AI behaviors that do not provide a truly immersive and engaging experience. Non-player characters (NPCs) often exhibit repetitive patterns and struggle to adapt to changing circumstances. This can lead to frustration and boredom for players, ultimately detracting from the overall enjoyment of the game. In addition, commercial AI implementations often prioritize efficiency and cost-effectiveness over quality and realism. This can result in AI systems that are overly simplistic, lacking the depth and complexity required to create truly believable and intelligent NPCs. As a consequence, players may find themselves facing predictable and easily exploitable AI opponents.

In recent years, *formal methods* have emerged as a useful approach to designing, verifying, and analyzing AI systems. Logics for strategic reasoning, particularly when used in conjunction with Multi-Agent Systems (MAS)[18,26], provide powerful tools for representing knowledge and reasoning about complex interactions between autonomous agents. A prominent example in this area is *Alternating-time Temporal Logic* (ATL)[1], a notable framework within Symbolic AI. ATL enables the modeling and analysis of intricate agent interactions, allowing for the formal verification, by means of *model checking*[6], of both cooperative and competitive scenarios. Despite the natural applicability of ATL to dynamic environments, the potential of this logic (and the like) has never been explored in real-time video games.

Our Contribution In this paper we introduce a novel in-engine AI solution built by using an ATL model checking framework. What sets this approach apart is its unique combination of ATL model checking and in-engine implementation. While other AI solutions have never explored ATL for game development in practice, this paper presents a pioneering integration directly within the game engine "Unreal Engine 5". This eliminates the overhead of external tools and streamlines the development process, making it more accessible and efficient for game developers. The key lies in ATL's Concurrent Game Structure (CGS), which allows for natural modelling of video game environments and temporal reasoning. This translates into faster development and more complex AI behaviours thanks to Blueprints' visual scripting: enables the construction of intricate logic without traditional coding. Actions represent high-level goals rather than specific movement paths, further contributing to efficient processing. A pre-evaluated transition matrix facilitates rapid evaluation of state changes based on actions. This approach unlocks the potential for real-time adaptation of non-player characters (NPCs) to player actions and the environment, ensuring engaging and dynamic gameplay. The model checker allows the synthesis of optimal NPC strategies across various scenarios, leading to a more strategic and immersive experience. This solution is not limited to a single genre; it is suitable for real-time strategy games, first-person shooters, and various genres requiring intelligent NPC decision-making in which are distinguishable groups of goal states. This innovative ATL model checking solution empowers the creation of intelligent and adaptable AI agents, thereby enhancing the gameplay experience.

Related Work Formal logic applied to video games have been investigated in the literature, but the studies have primarily focused on external tools and offline verification processes [21,10,25]. In contrast, [4] and [15] investigate the use of formal methods in *real-time* applications by implementing a game engine through the representation and reasoning of MAS, but without directly employing formal logics. In [14], Lora et al. presented a novel implementation of Tetris, in which each piece is represented as an agent. This work has been further extended in more complex MAS scenarios such as Real-Time Strategy games employing potential fields [9]. Optimal control strategies have been achieved in various studies through the use of reinforcement learning for distributed control protocols in leader-follower MAS [17]. In [22] the authors have focused on persistent monitoring using a network of mobile agents, with the aim of eliminating uncertainty across all targets and ensuring comprehensive coverage of the mission domain. Extended work in this field has been also analyzed in [7,11,16,20,24]. The coordination of agents has also been a significant area of MAS research. Kapetanakis et al. showed that an agent trained in Q-learning can collaborate with an agent trained with FMQ to converge towards optimal joint action [12].

Outline The sequel of the paper is organized as follows. Section 2 recalls the main notions of ATL. Section 3 presents the implementation of our tool. Section 4 evaluates its performance by means of experiments. Finally, we conclude in Section 5.

2 Alternating-Time Temporal Logic

In this section we briefly recall the Alternating-Time Temporal Logic. An interested reader can refer to [1]. ATL formulas are evaluated within Concurrent Game Structures,

which are labeled graphs representing the agents' behaviors within the system. The subsequent section presents a rigorous definition.

Concurrent Game Structure A CGS consists of a tuple $S = \langle Agt, Q, \Pi, \pi, d, \delta \rangle$ where: (i) $Agt \geq 1$ is a natural number indicating the number of players. (ii) Q is a finite set of states. (iii) Π is finite set of propositions. (iv) π is a labelling function such that for each state q, $\pi(q) \subseteq \Pi$ is a set of true propositions at q. (v) $d_a(q) \geq 1$ indicates for each player a and each state q, the number of actions available in the state q for the player a. A *move vector* at q consist of a tuple $\langle j_1, \ldots, j_{Agt} \rangle$ such that $1 \leq j_a \leq d_a(q)$ for each player a. (vi) δ is a transition function that for each state q and each move vector, returns a state that results from q if each player a chooses the move in the vector. We now present the precise syntax of ATL.

ATL Alternating-Time Temporal Logic (ATL)[1,13] is a powerful logic specifically designed for reasoning about MAS. It considers situations where multiple agents, each with their own goals and capabilities, can take actions concurrently.

An ATL formula φ can be expressed using the syntax:

$$\varphi := p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \langle\langle A \rangle\rangle X\varphi \mid \langle\langle A \rangle\rangle G\varphi \mid \langle\langle A \rangle\rangle \varphi_1 U \varphi_2$$

where p is an atomic proposition, $\neg$ and $\wedge$ stand for the usual negation and conjunction, and X, G and U are temporal operators standing for "*next*", "*globally*" and "*until*", respectively. The $\langle\langle A \rangle\rangle$ operator in $\langle\langle A \rangle\rangle \varphi$ represents the capability of a set of agent A of enforcing states where φ holds true. Before defining Semantics we need to introduce some basic concepts, i.e. strategies, plays and outcomes.

Strategy In a game's context, *plays* encompass the strategic choices or moves that participants can make, fundamentally influencing the progression of the game's state. For a CGS S and its players, a player's strategy is a rule assigning a play to every finite prefix of a state's sequence, termed a *computation* of S, which is an unbounded series $\lambda = q_0, q_1, q_2, \ldots$ of states where each state q_{i+1} directly follows q_i. For a state $q \in Q$, a player group A, and a strategy set F_A including one strategy per player in A, the *outcomes* of F_A from q, denoted as $out(q, F_A)$, refers to the series of computations resultant from players in A applying the strategies in F_A.

Semantics ATL is an extension of Computational Tree Logic (CTL) by means of strategic operators. So, the semantics of temporal operators follows that of CTL, and to get the full semantics of ATL we just need to add the interpretation of the strategic operators. To illustrate this part, as an example, consider an ATL formula of the form $\langle\langle A \rangle\rangle X\varphi$ in a state q of S. To evaluate this formula we need to check for the existence of a set of strategies F_A such that for all computations λ in $out(q, F_A)$, we have $\lambda[1] \models \varphi$ with $\lambda[1]$ indicating the immediate successor state of q. For a detailed definition of the semantics of ATL, the reader can refer to [1].

3 Implementation

This part offers a comprehensive analysis of how the system is implemented using the Unreal Engine framework. It explores the system's interactions with the game environ-

ment, NPCs and player, thoroughly detailing the processes that govern these interactions as well as the development methodologies used.

Blueprint visual scripting BP empowers designers and artists to create game logic, animations, and interactions visually using connected nodes (actions, functions,...). This simplifies development, allowing rapid prototyping and complex logic without coding.

Game Introduction The testbed video game, available for exploration and experimentation within the linked GitHub [1] repository, takes place in a labyrinth with two randomized escape doors. Player, in the "Chasers" team, cooperates with an NPC teammate to close both doors before an enemy NPC - the "Escapist" - escapes. AI agents on both sides have complete knowledge of the environment (layout, door/player locations) while player does not. This asymmetry creates a strategic challenge for the player who rely on real-time recommendations from its NPC teammate, guided by the internal model checking algorithm to choose its decisions and those of the player.

Game Environment Formalization In order to perform ATL model checking, the game environment has been abstracted and simplified by focusing on its most essential aspects through the use of state representation. Each state is costituted by 8 boolean variables. Three variables describe the Chaser Team, indicating the closest entity to the door and a flag of proximity for both doors. Three describe the Escapist Team, indicating the target door, the enemy proximity and its escape status, while one variable is used for each door for its open or closed status. This abstraction resulted in 160 distinct states. The game employed 9 different actions in two categories: Movement (Idle, GoToDoor, CloserToDoor) which are used to navigate towards a door, and Interaction (CloseDoor, Escape) to perform door related actions. Each door is associated with its own action (GoToDoor1/2), doubling the total number of actions. To efficiently evaluate the transition function, which determines how states change based on actions, a weighted graph was implemented. To handle the matrix complexity the model was divided into four submatrices, each comprising 40 states, that were categorized based on the entity (player or agent) closest to the target door and whether the target door was the first or second one. This resulted in a total of 14,400 transitions. Dividing the model into submatrices, omits certain transitions, but it has a minimal impact on the model's accuracy, given that there is no particular action to be taken specifically when the destination door changes for both the player and the agent. Each distinct combination of agent actions was encoded as an integer value. This approach reduced the required number of cells to 6,400, without altering the matrix size regardless of the number of agents.

Fig. 1: Labyrinth Map with two randomly spawned Doors.

Witnessed Model Checking To expedite the model checking process, propositional formulas are initially parsed into binary trees with internal nodes - with logical and temporal connectives - and leaves - incorporating atomic propositions - with each node

[1] https://github.com/Salvthor/LabyrinthATL

storing states where the proposition is true, optimizing runtime checks. The algorithm leverages this tree structure to identify states that satisfy the entire formula along with the corresponding strategies leading to those states. The algorithm traverses the tree in a left-to-right manner; upon reaching a leaf node, it retrieves the connective within its parent node and performs operations based on its type inserting the satisfying states within the current node. This process is iterated for all nodes in the tree, culminating with the set of states satisfying the entire formula being stored within the root node.

Formula Analysis Consider the following ATL formula example: $\langle\langle A \rangle\rangle\ G\ Door1Closed$ This formula expresses the goal of the *coalition A* to ensure that *Door 1* is and remains closed in all future states of the game starting from a current state s. To achieve this goal, the algorithm must find a sequence of actions, or a *strategy*, that will successfully guide the game state to a state where the formula is satisfied.

AI Integration The AI controller of NPCs integrates the model checker for the making of decisions in real-time decision-making. To optimize performance, the system first checks if the formula has already been parsed into a binary tree. For new formulas, the algorithm follows the previously described steps and stores both strategies and satisfying states within a cache structure, that is consulted whenever the formula results are needed during gameplay iterations. To prevent a lack of strategy during the game loop, the system employs a Prioritized Formula Queue. This queue holds propositional formulas representing desirable states for the team, ordered by their importance. The most critical formulas are evaluated first: if a high-priority formula cannot be satisfied, meaning there is no strategy from the current state to enforce it, the system moves on to the next formula in the queue. The first satisfied formula determines the chosen strategy and actions for each agent, following the principles of best-effort strategies synthesis [2]. This iterative evaluation process is crucial because player choices can dynamically change the game state. This might make a previously unsatisfied critical formula achievable, triggering a new strategy selection based on the updated formula priority and the current game state.

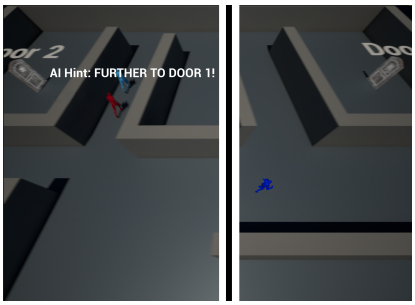

Fig. 2: On the left the NPC (cyan) reach the Door2 before the opponent (red) escapes, while giving hint to Player in blue, on the right.

4 Experiments

We have performed extensive tests for the proposed game framework on a computer running Windows 11 with 16GB of RAM, an Intel Core i5 9600k CPU, and an Nvidia 2060 Super GPU with 8GB of GDDR6 VRAM. Unreal Engine 5.3 served as the development platform, utilizing its Blueprint visual scripting tool (based on C++). A Python function was used to generate complex scenarios where any state can transition to any other, unlike real video games with invalid transitions limiting the matrix analysis. The testing focused on two main areas: scalability and impact of agent capabilities. Three agents (two allies and one enemy) and transition matrices of varying sizes (10, 100

and 250 states) were employed. The experiments investigated how different agent capabilities, defined by varying the number of actions per agent (2, 3, and 5), influenced the model checker performances while maintaining constant formula complexity (10 propositions), since prior studies indicated that formula length did not impact model checker performance. To ensure consistency, five randomly generated models per transition matrix complexity level underwent 500 test runs. The results in Figure 3 show the efficiency of the algorithm in handling larger transition matrices even with more complex formulas. This emphasis on scalability reflects the critical role of performance in video games, where efficient algorithms are essential for smooth gameplay. Although with larger models with a transition graph consisting of 312,500 elements, the model checker experienced a performance bottleneck, this issue is not particularly problematic.

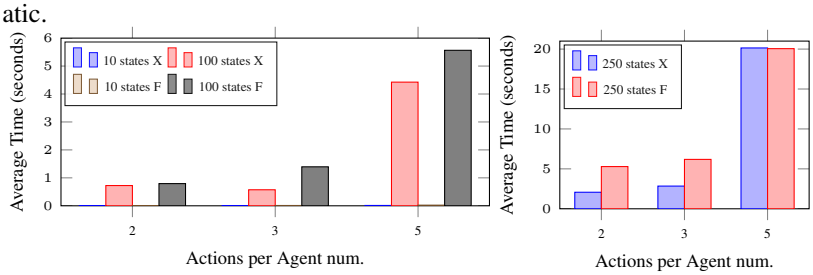

Fig. 3: Model checking response times with: 3 agents, formula with 10 atomic proposition with varying agents capabilities.

In facts, real-time game model analysis, facilitated by swift execution times, is undoubtedly an accomplishment, however, such instantaneous feedback is not essential for this application. Model checking can be performed offline with the outcomes stored for later use in real-time scenarios. Thus, the simplicity of the model does not undermine the reliability of the model checker because its verification can occur during the game loading phase. The specific game model can then be bypassed, allowing tests to be run on randomly generated, highly complex models. The main goal was to demonstrate the practical feasibility of ATL model checking using a simplified model, a significant contribution given the limited number of ATL real-world in-game implementations.

5 Conclusions

This work demonstrates the feasibility and scalability of applying ATL model checking in video games, particularly for NPCs that require high responsiveness. The model checker provides an effective method for adapting to the player's gameplay style, ensuring that NPCs can intelligently complete tasks while continuously adapting to the environment, leading to an engaging and responsive experience. Our tool can be applied to real-time strategy games and macro-decisions in first-person shooters such as deciding what action to take between shoot, go to cover, throw a grenade or heal, etc. In future work, reinforcement learning could automate transition matrix population, simplifying invalid transition identification and potentially increasing commercial adoption of the approach through the addition of other temporal or strategic logics that more closely represent the game model. Additionally, developing extensions for existing model checkers like VITAMIN[8] and NuSMV[5] could enable performance comparisons to identify the most efficient tool.

Acknowledgements

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

References

1. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. J. ACM **49**(5), 672–713 (sep 2002). https://doi.org/10.1145/585265.585270, https://doi.org/10.1145/585265.585270
2. Aminof, B., De Giacomo, G., Lomuscio, A., Murano, A., Rubin, S.: Synthesizing Best-effort Strategies under Multiple Environment Specifications. In: Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning. pp. 42–51 (11 2021). https://doi.org/10.24963/kr.2021/5, https://doi.org/10.24963/kr.2021/5
3. Barriga, N.A., Stanescu, M., Besoain, F., Buro, M.: Improving rts game ai by supervised policy learning, tactical search, and deep reinforcement learning. IEEE Computational Intelligence Magazine **14**(3), 8–18 (2019). https://doi.org/10.1109/MCI.2019.2919363
4. Chover, M., Marín, C., Rebollo, C., Remolar, I.: A game engine designed to simplify 2d video game development. Multimedia Tools and Applications **79**, 12307–12328 (2020)
5. Cimatti, A., Clarke, E., Giunchiglia, F., Roveri, M.: Nusmv: a new symbolic model checker. STTT **2**, 410–425 (03 2000). https://doi.org/10.1007/s100090050046
6. Clarke, E.M.: Model checking. In: Foundations of Software Technology and Theoretical Computer Science: 17th Conference Kharagpur, India, December 18–20, 1997 Proceedings 17. pp. 54–56. Springer (1997)
7. Farina, G., Gatti, N.: Extensive-form perfect equilibrium computation in two-player games. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 31 (2017)
8. Ferrando, A., Malvone, V.: Vitamin: A compositional framework for model checking of multi-agent systems (2024)
9. Hagelbäck, J., Johansson, S.J., et al.: A multiagent potential field-based bot for real-time strategy games. International Journal of Computer Games Technology **2009** (2009)
10. IGAWA, N., YOKOGAWA, T., TAKAHASHI, M., ARIMOTO, K.: Model checking of visual scripts created by ue4 blueprints. In: 2020 9th International Congress on Advanced Applied Informatics (IIAI-AAI). pp. 512–515 (2020). https://doi.org/10.1109/IIAI-AAI50415.2020.00107
11. Jia, Z., Yang, L.F., Wang, M.: Feature-based q-learning for two-player stochastic games (2019), https://arxiv.org/abs/1906.00423
12. Kapetanakis, S., Kudenko, D.: Reinforcement learning of coordination in heterogeneous cooperative multi-agent systems. In: Symposium on Adaptive Agents and Multi-agent Systems. pp. 119–131. Springer (2003)
13. Laroussinie, F., Markey, N., Oreiby, G.: On the expressiveness and complexity of ATL. CoRR **abs/0804.2435** (2008), http://arxiv.org/abs/0804.2435
14. Marín-Lora, C., Chover, M., Sotoca, J.M.: A multi-agent specification for the tetris game. In: Distributed Computing and Artificial Intelligence, Volume 1: 18th International Conference 18. pp. 169–178. Springer (2022)
15. Marín-Lora, C., Cercós, A., Chover, M., Sotoca, J.: A First Step to Specify Arcade Games as Multi-agent Systems, pp. 369–379 (05 2020). https://doi.org/10.1007/978-3-030-45688-7_38

16. Mazouchi, M., Naghibi-Sistani, M.B., Sani, S.K.H.: A novel distributed optimal adaptive control algorithm for nonlinear multi-agent differential graphical games. IEEE/CAA Journal of Automatica Sinica **5**(1), 331–341 (2017)
17. Moghadam, R., Modares, H.: Resilient adaptive optimal control of distributed multi-agent systems using reinforcement learning. IET Control Theory & Applications **12**(16), 2165–2174 (2018)
18. Parasumanna Gokulan, B., Srinivasan, D.: An Introduction to Multi-Agent Systems, vol. 310, pp. 1–27 (07 2010). https://doi.org/10.1007/978-3-642-14435-6_1
19. Pearce, T., Zhu, J.: Counter-strike deathmatch with large-scale behavioural cloning (2021)
20. Qiu, S., Li, Z., Pang, Z., Li, Z., Tao, Y.: Multi-agent optimal control for central chiller plants using reinforcement learning and game theory. Systems **11**(3), 136 (2023)
21. Rezin, R., Afanasyev, I., Mazzara, M., Rivera, V.: Model checking in multiplayer games development. In: 2018 IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA). pp. 826–833 (2018). https://doi.org/10.1109/AINA.2018.00122
22. Song, C., Liu, L., Feng, G., Xu, S.: Optimal control for multi-agent persistent monitoring. Automatica **50**(6), 1663–1668 (2014)
23. Spronck, P., Ponsen, M., Sprinkhuizen-Kuyper, I., Postma, E.: Adaptive game ai with dynamic scripting. Machine Learning **63**, 217–248 (2006)
24. Urbano, A., Vila, J.E.: Computational complexity and communication: Coordination in two–player games. Econometrica **70**(5), 1893–1927 (2002)
25. Wayama, K., Yokogawa, T., Amasaki, S., Aman, H., Arimoto, K.: 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 (2023). https://doi.org/10.1145/3551349.3560505, https://doi.org/10.1145/3551349.3560505
26. Wooldridge, M.: An introduction to multiagent systems. John wiley & sons (2009)