Explanation and Clustering of Playtraces using Temporal Logics

Pablo Gutiérrez-Sánchez
pabgut02@ucm.es
Complutense University of Madrid
Madrid, Madrid, Spain

Diego Pérez-Liébana*
diego.perez@qmul.ac.uk
Queen Mary University of London
London, London, United Kingdom

Raluca D Gaina*
r.d.gaina@qmul.ac.uk
Queen Mary University of London
London, London, United Kingdom

Abstract

This paper addresses the challenge of explaining gameplay behaviours and traces in video games using methods based on linear temporal logics (LTL). Applications for this range from classifying a player's game-style to craft personalised user experiences, to exploring the most significant behaviour patterns within a set of trajectories, particularly in the context of data-driven design and quality control assisted by black-box algorithms. We divide the problem into two complementary tasks. First, to infer a temporal characterisation of a registered play-style by means of a predicate in LTL from a set of representative traces and potential counterexamples. Second, to classify a diverse set of traces into groups in order to identify behavioural patterns within the samples. The first problem focuses on recognising what makes a behaviour unique when compared to others, while the second problem seeks to detect meaningful patterns in groups of players. For the first task, we propose a series of heuristic search methods in the LTL predicate space, such as Monte Carlo Tree Search and Grammatical Evolution. For the second, we introduce a new algorithm that clusters traces based on predicates that split them into cohesive sets, demonstrating how the methods of the first problem can be extrapolated to the latter. Both approaches are evaluated with practical experiments on a 3D third-person stealth game developed in Unity 3D, showcasing how these techniques can be used for analysis. Preliminary results obtained with real player traces provide evidence that these methodologies can support a more comprehensive understanding of observed behaviours.

CCS Concepts

• **Theory of computation** → **Linear logic**; **Modal and temporal logics**; • **Computing methodologies** → **Game tree search**; **Discrete space search**; **Randomized search**; **Temporal reasoning**.

Keywords

Temporal Logics, Playtrace Analysis, Game Analytics, Monte Carlo Tree Search, Grammatical Evolution, Gameplay Clustering

ACM Reference Format:
Pablo Gutiérrez-Sánchez, Diego Pérez-Liébana, and Raluca D Gaina. 2025. Explanation and Clustering of Playtraces using Temporal Logics. In *International Conference on the Foundations of Digital Games (FDG '25), April 15–18, 2025, Graz, Austria.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3723498.3723719

*Both authors contributed equally to this research.

1 Introduction

The recent surge of complex black-box systems in artificial intelligence methods has led to a growing interest in devising simple, or at least human-understandable, explanations of the underlying logical processes behind these systems' abstractions. In fields such as healthcare, robot manipulation, or transportation, the design of models that are interpretable and directly tractable by humans has become a focal point for ensuring trust and adoption by real users [1, 24].

This applies to the video game industry as well, where there exists a steadily growing literature on the potential of deep learning for quality assurance (QA), testing, and even the design of bots, Non-Player Characters (NPCs), or artificial players for different areas of interest. It is in QA that these techniques have been most successfully adopted, as the resulting agents are largely proxies for human testers in order to explore environments, look for bugs in the game, or perform systematic regression tests to ensure that certain combinations of actions and interactions lead to the expected results as incremental changes are introduced to other parts of the product. These applications also suffer from a lack of explainability that reduces the trust of practitioners and hinders the clarity of the results gathered in the QA process.

This desire to clarify the rationale behind black box systems is essentially a dual form of another task of interest in the industry, namely the analysis of gameplay and play-styles. Arguably, this problem presents two distinct dimensions depending on the design objective to be pursued. For an already released game, or a game currently under development undergoing beta testing with real players, the first issue of interest is to understand what kinds of interactions are likely to be encountered in practice. This can take several shapes, among which the following can be highlighted:

- Classifying players on the basis of game profiles, which are not known in advance. For example, after recording the behaviour of dozens of users, we could separate them into those who solve a level in an aggressive way by engaging in hand-to-hand combat against their enemies, and those who prefer to follow a stealth strategy to avoid confrontation.
- Conducting a statistical balance analysis of which strategies work best or worst and under what circumstances. This might include observations about a card or character in a game being used more frequently by top players, or items that are not equipped as much as expected by design.
- Performing a systematic exploration of game map regions via techniques such as heat maps, to understand the physical distribution of events of interest such as deaths, item uses, or scouted areas.

The second dimension considers the problem of inferring what category or style a player belongs to in real time, either based on their closeness to some preconceived archetype (e.g., wary, explorer, fighter) or on their level of skill and proficiency within the game. This is particularly useful for guiding dynamic difficulty adjustment mechanisms or for customising the user experience to the individual's characteristics.

The former is generally an explorative problem where the focus is on detecting patterns in user behaviour, whereas the latter is more in line with a supervised regression or classification paradigm, although what this adjustment is performed on may be a vague notion (e.g., the skill level of a player, which can be complex to quantify objectively).

In this paper, we will focus mainly on this first exploratory question and, more specifically, on the inference of meaningful temporal patterns in game traces. For this purpose, and momentarily departing from the game domain, some of the most typically employed models include finite state machines and temporal logics, which not only possess a substantial set of theoretically desirable properties, but also come with a syntax and structure that are generally simple to understand, alongside human-friendly semantics. The latter turns them into mechanisms that are particularly well conditioned to construct interpretable models in the tasks described above.

Now, this problem, by its inherently exploratory nature, is ill-conditioned in that the notions of what is or is not interpretable, as well as what provides useful information and what does not, have no self-evident or theoretically straightforward numerical definition. As an example, a concise model that classifies a set of game traces correctly could be the trivial model that simply classifies them all as positive (a logical true statement), but this does not provide any useful information that explains what exactly is happening in the game. However, an overly explanatory model that details every step of the interaction could become overwhelming and hard to grasp; hence, the aim here is to strike a compromise between correctness and complexity within our explanations.

With this in mind, we begin the paper by summarising related work on similar problems in Section 2, after which we move on to describing the general concepts and establishing the notions and notations necessary to understand the rest of the article, with concise introductions in Section 3. We then elaborate on our first contribution, which involves formulating formal definitions of two exploratory problems of interest in video games. The first, described in Section 4, aims to characterise what differentiates a set of game traces from others through models that focus on temporal properties that are fulfilled in those examples but not in the provided counterexamples (e.g. one group might be distinguished by always defeating enemies and then opening a chest, while the other might try to avoid foes and go straight for the loot). The second, presented in Section 5, given a set of game traces from different potentially diverse play sets, addresses the problem of dividing them into well-structured groups that share similar behaviours. In turn, we differentiate them from each other, similar to the case of a clustering algorithm, but with a special focus on the temporal part.

For the first problem, we propose a set of predicate space search methods mostly based on heuristic techniques such as Grammatical Evolution or Monte Carlo Tree Search, in what is known as Linear Temporal Logic (LTL), a formal language for specifying properties of a system over time. For the second problem, we introduce a new algorithm that groups traces based on predicates that divide them into cohesive sets separated from each other. These methods are further detailed in the same sections where each problem is defined.

Subsequently, Section 6 reports on experiments conducted on a stealth game testing environment developed in Unity 3D, where the above algorithms are tested and contrasted on real player traces. Section 7 concludes the paper with discussions and future work.

2 Related Work

The problem of learning temporal properties from positive and negative examples has been widely discussed both in the field of robotic control and in more general contexts within various works in the literature. Some of the most relevant contributions in this field come from Grinchtein et al. [29] and Biermann et al. [3] on the synthesis of finite state machines from demonstrations, and again Grinchtein et al. [9] on the automatic learning of time-invariant properties of complex systems. More recently, Roy et al. [26] propose a symbolic approach supported by SAT Solvers for the task of learning predicates in Temporal Logic from positive examples only, which is known as the one class classification (OCC) problem, while Raha et al. [23] propose a set of techniques used to learn segments of predicates in linear temporal logic in a more general fashion.

On the other hand, the problem of automatically recognising play styles in video games enjoys an extensive literature base, both in archetypal style classification and in regression on skill level. Bontchev et al. [4] propose a model for recognising play styles using linear regression techniques on performance metrics in different subtasks in a game with adaptive difficulty. Valls-Vargas et al. [28] detail a new approach where it is suggested that it may be more effective to infer play styles as a dynamic and time-varying property, and suggest a player-modelling framework that exploits this characterisation. Ingram [14] makes use of unsupervised deep clustering strategies such as long short term memory (LSTMs) auto-encoders to convert game traces into points in a latent space of lower dimensionality on which to later apply clustering algorithms.

Similar deep learning techniques have also been applied to various unsupervised clustering problems of game trajectories and general systems, several of which also make use of temporal auto-encoders as a prior dimensionality reduction step [18, 30]. Deep learning has also been used in supervised problems in order to identify which player category a user belongs to by combining latent spaces with datasets of game trajectories labelled with their matching style [27]. While effective for building broad behavioural archetypes, these often lack the granularity to capture specific behavioural flows within categories. Our approach provides a complementary perspective by focusing on the general temporal characterization of groups of play-traces.

Lastly, a similar problem has been studied in the field of inverse reinforcement learning (IRL) to learn reward functions for RL agents based on positive demonstrations. Kasemberg et al. [15] learn LTL formulas from demonstrations of behaviour trajectories in Markov Decision Processes (MDPs), although they do so on the assumption that the underlying system's internal mechanisms are accessible. In turn, Hasanbeig at al. [13] introduce an algorithm

for inferring automata describing high-level goals for RL agents. It is worth mentioning that there are multiple additional works that take specifications expressed in temporal logics as a starting point to train agents by reinforcement, deriving learning mechanisms in which the customary process of reward engineering is avoided, in favour of more accessible and interpretable languages [11, 16, 31].

3 Preliminaries

In this section, we describe the general concepts and establish the notation to be used in the rest of the paper.

Game Traces. To formally represent the executions of a game, we will resort to the concept of traces defined over a non-empty set Σ of possible game states $s \in \Sigma$ to be considered in the analysis. A trace over Σ is a finite sequence $t = \{s_1, \ldots, s_n\}$, where $s_i \in \Sigma$, $1 \leq i \leq n\}$. In the context of this paper, a trace will correspond to an observation of the game states logged over the course of a play session of a given player in their interaction with a game. In what follows, we represent the length of a trace t by $|t|$. Additionally, we denote by Σ^* the set of all possible traces that can be generated from Σ. Here we speak of states in a deliberately ambiguous sense, and shall define what is understood as game state more precisely after introducing predicates and temporal logics.

Temporal Logics. By Temporal Logics (TLs) we refer to any system of rules and symbols that allows reasoning about propositions in terms of their evolution over time. TLs play an important role in the formal verification of requirements in both hardware and software [6], with Linear Temporal Logic (LTL) [17] being one of the most widely adopted examples of these logics. LTL formulas (which we shall also refer to as specifications in what follows) are defined based on a series of core logical and temporal operators and, more importantly, predicates of the form $f(s) > 0$, where $f : \Sigma \to \mathbb{R}$ represents a function applied to the system's state $s \in \Sigma$. The latter are the ones that are truly responsible for incorporating a certain knowledge base into our specifications, as well as conditions related to the world and context in which we operate, such as "being close to a goal" or "an enemy having a high awareness of the player." These conditions are highly domain-dependent and consequently, for each game we contemplate, we will inevitably encounter the need to define a specific set of predicates that will determine the expressiveness of our tasks. In what follows, we shall assume that system states are given directly through arrays of real-valued evaluations of the original game state, i.e. $\Sigma \subseteq \mathbb{R}^n$. An LTL specification ϕ adheres to the following syntax:

$$\begin{aligned} \phi := \top \mid f(s) > 0 \mid \neg\phi' \mid \phi'_A \wedge \phi'_B \mid \phi'_A \vee \phi'_B \mid \\ \phi'_A \Rightarrow \phi'_B \mid \mathcal{F}\phi' \mid \mathcal{G}\phi' \mid \phi'_A \,\mathcal{U}\, \phi'_B \mid \mathcal{X}\phi' \end{aligned} \tag{1}$$

Here, $\top$ is the True boolean constant, $f(s) > 0$ is a check on a domain function on the system state, $\neg$ (negation), $\wedge$ (conjunction), $\vee$ (disjunction) and $\Rightarrow$ (implication) are Boolean connectives, and $\mathcal{F}$ (eventually), $\mathcal{G}$ (always), $\mathcal{U}$ (until), and $\mathcal{X}$ (next) are temporal operators. The above grammar is complete in that it is capable of generating any LTL predicate.

Quantitative Semantics and Progress. All of our algorithms rely on the notion of progress of a trace with respect to a specification, which we will briefly introduce in this section. Intuitively, we are interested in defining a function that takes as inputs a trace and a predicate in LTL, and computes a real value indicating how well the trace has complied with the corresponding specification. When the predicate is fulfilled throughout a run $t \in \Sigma^*$ we shall say that the trace models the specification and write $t \models p$ or, alternatively, that t is an example of p. In the opposite case, we will say that t is a counterexample of p, and denote it by $t \not\models p$. Ideally, we would expect our fitness function to be positive if and only if the evaluated trace models the predicate, to be negative for counterexamples, and to report the appropriateness of this satisfaction or rejection by means of its magnitude; that is, the larger its absolute value, the greater the conformance or violation of the trace to the predicate under consideration, for the positive and negative cases, respectively. A function that fulfils these properties is commonly known as a robustness function or qualitative semantics [22].

In the case of the grammar from Equation 1, here we define the robustness function $\rho_s(\phi) := \rho(s, \phi)$ that outputs:

$$\begin{aligned} \rho_s(\top) &= 1 \\ \rho_s(\neg\phi) &= -\rho_s(\phi) \\ \rho_s(f(s) > 0) &= f(s) \\ \rho_s(\phi_A \wedge \phi_B) &= \min(\rho_s(\phi_A), \rho_s(\phi_B)) \\ \rho_s(\phi_A \vee \phi_B) &= \max(\rho_s(\phi_A), \rho_s(\phi_B)). \end{aligned} \tag{2}$$

The above expressions only consider predicates that do not include temporal operators or, in other words, that can only be evaluated over a single time instant and not over complete traces (hence the overloaded notation for state instead of trace). While a number of quantitative semantics exist in the literature that enable the evaluation of goodness-of-fit to predicates including temporal operators, in this paper we shall adopt the automaton construction from [11].

Any predicate in LTL can be transformed into a Finite State Predicate Automaton (FSPA) by means of various different procedures [2]. A FSPA is a finite state automaton that processes a system trace step by step and advances through its different states until it decides whether to accept or reject the input, with the particularity that its edges are equipped with propositional logic predicates and a quantitative semantics. At each step of the execution, the predicates associated with the outgoing edges from the current state of the automaton are evaluated over the next state in the trace, and the most robust positive transition is chosen according to the rules of Equation 2. Whether or not an FSPA constructed from a predicate accepts or rejects a trace (i.e., whether or not it ends up in an acceptance state) is equivalent to deciding if the trace does or does not model the predicate at hand. It is worth noting that an FSPA can be based on different types of automata, as it is simply an abstract construction, with the most common ones being deterministic Rabin and Büchi automata.

An advantage of using an auxiliary automaton to analyse traces is that it makes it possible to calculate the progress of a trace in the specification, measured as the distance advanced in the structure towards the nearest acceptance condition. This is generally simpler to compute than other quantitative semantics of temporal predicates in LTL that do not rely on such constructs, and also easier to interpret, as it always results in a value between 0 and 1 that intuitively denotes what proportion of the specification has been successfully satisfied.

Let us express this notion formally. Let $t = \{s_0, \ldots, s_n\}$, $s_i \in \Sigma$ be a system trace, ϕ the predicate in LTL containing our specification, s^t_{FSPA} the state of the corresponding automaton reached after processing t sequentially, and D_{FSPA} the maximum distance between two nodes within the automaton. By now terming the minimum distance between the final state of the automaton and an acceptance state as $d_{\text{FSPA}}(s^t_{\text{FSPA}})$, we define the progress attained in the trace $\rho_{\text{FSPA}}(t)$ as:

$$\rho_{\text{FSPA}}(t) = \frac{D_{\text{FSPA}} - d_{\text{FSPA}}(s^t_{\text{FSPA}})}{D_{\text{FSPA}}} + \frac{1 + \max_{e \in impEdg}(\rho_{s_n}(\phi_e))}{D_{\text{FSPA}}}. \tag{3}$$

The second term in the expression represents the distance progressed in the current state to advance to any adjacent state of the automaton that is closer to an acceptance condition (here, $e \in impEdg$ are edges that lead to states improving the progress metric in the specification).

As an example, if we define a predicate $\mathcal{F}(\texttt{enemyDefeated} \wedge \mathcal{X}(\mathcal{F}(\texttt{itemCollected})))$, the resulting automaton would have three states, one for "enemy not yet defeated", one for "enemy defeated but item not yet collected", and a last one for having completed both goals, with $D_{\text{FSPA}} = 2$. If the enemy has just been defeated and the player is still far from the item, we would likely observe that $\rho \approx 0.5$, as the current distance to the next acceptance state is 1, which is halfway through the automaton size.

4 Inferring Temporal Characterisations of Play-Traces

In this section, we present our algorithms for learning temporal predicates to characterise play styles through a set of examples and counterexamples. We begin by formally defining the problem at hand.

Let $\mathcal{P}_g$ be the (possibly infinite) set of LTL predicates that can be generated from an LTL grammar g. We define a trace evaluator σ as a function that takes a predicate $p \in \mathcal{P}_g$ and a real-valued trace of length $n \in \mathbb{N}$, $t = \{s_1, \ldots, s_n\}$, in a given trace space Σ^* and outputs an array of fitness values representing how well the predicate models the trace according to the function's criteria in a series of dimensions. More formally, if $m \in \mathbb{N}$ is the number of fitness dimensions (or alternatively, the number of metrics that are computed for each trace), this is a function with signature $\sigma : \Sigma^* \times \mathcal{P}_g \to \mathbb{R}^m$:

$$\sigma(t, p) = (\sigma_1(t, p), \ldots, \sigma_m(t, p)),$$

where $\sigma_i(t, p) \in \mathbb{R}$, $0 \leq i \leq m$, is the i-th fitness value of the trace t with respect to the predicate p.

This allows us to convert a set of traces into a real vector representation that can be used to compare traces and predicates with each other according to a set of user-defined heuristics. On top of this, we can define additional functions to aggregate the results of each evaluator considered in the analysis into a single goodness value accounting for evaluators' outputs across sample sets (usually a combination of a positive examples set and a negative one). A typical aggregation function has the signature $\gamma_\sigma : (\Sigma^*)^* \times (\Sigma^*)^* \times \mathcal{P}_g \to \mathbb{R}$.

With these notions in mind, we are ready to formally state this section's problem.

Problem 1. Given a set of positive traces or examples P, a second set of negative traces or counterexamples N for a game, a list of trace evaluators σ_i, $i = 1, \ldots, m$, and a metrics aggregator γ_σ, learn an LTL predicate ϕ such that: (1) $t \models \phi, \forall t \in P$; (2) $t \not\models \phi, \forall t \in N$; and (3) $\gamma_\sigma(P, N, p) >= \gamma_\sigma(P, N, p'), \forall p' \in \mathcal{P}_g$.

The first two conditions indicate that we aim to find predicates that fit the considered examples and counterexamples, while the third one states that we are also interested in the chosen predicates being optimal with respect to the established metric aggregator. In practice, we will use approximate strategies to address this problem, so the above conditions will not always be met exactly.

4.1 Brute Force Tree Expansion

The first search method we will consider to approximate Problem 1 can be seen in Algorithm 1, and is a brute-force approach in which all possible solutions within certain constraints are explored in search of the optimal one in terms of fitness.

First of all, it is to be noted that, from this point onwards, whenever we speak of a grammar, we will assume that we are working with one in Backus-Naur form (commonly BNF), a meta-syntax notation for context-free grammars, often used to describe the structure of language specifications. We will also assume when referencing clauses representing functions over the state of the system ($f(s) > 0$ in Equation 1) that the grammar has been extended to represent all variables of interest in the context where the language is used. This implies that the $f(s) > 0$ symbol can be expanded to any of the possible literals recorded in a game as an implicit rule.

That stated, the brute-force algorithm starts from the initial node of the grammar considered, and builds a production tree $\mathcal{T}_g$, in which each node is expanded according to all the possible derivations allowed from it. Since this tree is usually infinitely deep (one need only consider the recursive rules that reference a predicate within another), here we opt to prevent this construction from exceeding a maximum depth specified by the parameter $D \in \mathbb{N}$. Once the pruned tree is available, it is possible to perform any traversal of its leaves, collecting all the predicates that have been completely expanded within the imposed limit. For each of the predicates gathered in this way, it is enough to apply the chosen metrics aggregation function, and keep the individual with the best value according to this heuristic.

Algorithm 1 Brute Force Tree Expansion with Fixed Depth D.

Generate a production tree $\mathcal{T}_g$ from input grammar g, allowing a maximum depth of D.
For each leaf node in the tree that results in a fully expanded LTL predicate p, compute $\gamma_\sigma(P, N, p)$.
return $p \mid \gamma_\sigma(P, N, p)$ is maximal among terminal nodes.

4.2 Grammatical Evolution Search

Grammatical Evolution (GE) [21] is a type of genetic programming (GP; [19]) that evolves solutions by generating computer programs or expressions, guided by a predefined grammar. A common tool used in GE is precisely the Backus-Naur Form grammar, which provides a formal way to specify the syntactical structure of the

programs or expressions to be evolved. GE encodes solutions as sequences of integers, which are then mapped to syntactically correct programs using the BNF grammar.

The adaptation of our problem to a grammatical evolution model is straightforward by setting up the search grammar we are interested in alongside the heuristic function given by the corresponding metric aggregator, and is relatively simple to implement with genetic programming libraries such as the MOEA Framework [12].

4.3 Grammar-Based Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is a highly-selective best-first search method that is often used for decision-making. This algorithm balances between exploitation and exploration to balance the search tree growth towards the most promising parts of the search space. This iterative method combines a tree policy (e.g. UCB1 [8]) with Monte Carlo simulations or *rollouts*, which are trajectories sampled at random in the decision space. For more information about this algorithm and its variants, please refer to [5].

In our particular case, we model the search as a sequential decision problem on which production of the reference grammar to apply at each step of a derivation. That is, starting from a first state given by the initial symbol of the grammar, in each run of the MCTS we are interested in first deciding which of the symbols of our current state we wish to expand and, secondly, which of its production rules to use to perform this expansion. As MCTS heuristics, we make use of the metric aggregation function common to all the algorithms in this section, assuming a value of 0 for predicates that have not yet been fully expanded. This means that only rollouts that result in final states can yield useful information.

Algorithm 2 Grammar-Based MCTS.

Initialise base predicate s_p state with the base symbol from the chosen search grammar.
while s_p not fully expanded **do**
  Apply MCTS to s_p with allocated budget to select the seemingly best symbol to expand from the set of unexpanded symbols in s_p and production rule from the given grammar.
  Update s_p after applying the suggested expansion.
end while
return s_p.

5 Tree Clustering of Play-Traces

The second problem to be undertaken in this paper is to propose a classification or clustering approach to distil general behavioural patterns within samples from a potentially diverse set of traces. This is a purely exploratory analysis problem, and as such more complex to define than the task in the previous section. In this case, we shall frame it in terms of looking for predicates that are capable of "separating" traces as much as possible in the sense that the sets of samples that do and do not satisfy the predicate are cohesive and at the same time distant from each other.

More specifically, starting from the previously defined trace evaluators and some form of standard clustering metric, our problem is to construct a predicate that seeks to optimise this metric for the partition given by the sets of traces that it accepts and rejects, respectively.

More formally, given a predicate $p \in \mathcal{P}_g$ and a set of traces T, we define the sets $C^p_{\models}(T) = \{t \in T, t \models p\}$ and $C^p_{\not\models}(T) = \{t \in T, t \not\models p\}$ of traces accepted and rejected by that predicate. By now applying a trace evaluator σ on each member of these clusters we obtain two new sets of individuals $\sigma(C^p_{\models}(T)), \sigma(C^p_{\not\models}(T)) \subset \mathbb{R}^m$, on which it is now possible to apply a clustering metric to heuristically evaluate how good the partition of the samples into these two sets is. With this, we can formally formulate our second problem.

Problem 2. Given a set of unclassified traces or examples T, a list of trace evaluators $\sigma_i, i = 1, \ldots, m$, a clustering quality score function q, and a quality threshold $\alpha > 0$, learn a sequence of predicates $(p_j)_{j=0}^n$ such that:

$$P_j = C^{p_{j-1}}_{\not\models}(P_{j-1}), N_j = N_{j-1} \cup C^{p_{j-1}}_{\models}(P_{j-1}), P_0 = T, N_0 = \emptyset, \quad (4)$$

$$q(\sigma(C^{p_j}_{\models}(P_j)), \sigma(C^{p_j}_{\not\models}(P_j)), \sigma(N_j)) >= \alpha, j = 0, \ldots, n-1, \quad (5)$$

$$q(\sigma(C^{p_n}_{\models}(P_n)), \sigma(C^{p_n}_{\not\models}(P_n)), \sigma(N_n)) < \alpha. \quad (6)$$

Intuitively, we are describing an iterative task. We start from a set of traces T to be partitioned based on predicates p_j that explain a subset of samples with sufficient confidence $\alpha > 0$. In the first iteration, we are simply trying to find a predicate p_0 that splits T into two cohesive sets that are sufficiently separated from one another according to our clustering criterion in the real space induced by our selected metrics. N_1 will then become the set of traces that were modelled by p_0 and that as such do not need further classification, while P_1 becomes populated with the traces discarded in the first round. The next iterations follow this pattern, with the subtlety that we keep expanding N_j with the traces modelled by the sequence of splitting predicates, and we then try to find subsequent predicates in such a way that the groups $\sigma(C^{p_j}_{\models}(P_j)), \sigma(C^{p_j}_{\not\models}(P_j))$ and $\sigma(N_j)$ maintain a good clustering score. Therefore, at each iteration we ask ourselves the question "which trace groups can we find that appear to behave differently to all previous explanations?". In other words, we look for predicates that can explain unique and meaningful behavioural subsets, and progressively more specific ones. These iterations continue until it becomes infeasible to find predicates that induce a reasonable partitioning according to the quality threshold.

Algorithm 3 specifies a procedure by which to heuristically and approximately construct a sequence of predicates with the above conditions. Leveraging the constructs developed in the previous section for Problem 1, the tree clustering algorithm aims to model the trace separation steps as predicate searches that optimise the result of the metric aggregator given by:

$$\gamma_\sigma(P, N, p) = q(\sigma(C^p_{\models}(P)), \sigma(C^p_{\not\models}(P)), \sigma(N)). \quad (7)$$

That is, at each step we look for a predicate that induces the best possible separation between the traces we have already explained through previous predicates, the traces that become explained by the new predicate, and the traces that remain to be modelled.

Algorithm 3 Tree Clustering of Trace Set T with Threshold α.

Choose a list of trace evaluators $\{\sigma_i\}_{i=1}^{m}$.
Choose a clustering quality score q.
Choose a search algorithm S_{alg} from Problem 1, selecting $\gamma_\sigma(P,N,p) = q(\sigma(C^p_{\models}(P)), \sigma(C^p_{\not\models}(P)), \sigma(N))$ as metrics aggregator.
$P_0 = T$.
$N_0 = \emptyset$.
$j = 0$.
$p_0 \leftarrow S_{\text{alg}}(P_0, N_0)$.
while $\gamma_\sigma(P_j, N_j, p_j) >= \alpha$ **do**
  $j = j + 1$.
  $P_j = C^{p_{j-1}}_{\not\models}(P_{j-1})$.
  $N_j = N_{j-1} \cup C^{p_{j-1}}_{\models}(P_{j-1})$.
  $p_j \leftarrow S_{\text{alg}}(P_j, N_j)$.
end while
return $\{p_j\}$, sequence of splitting predicates.

6 Experiments

We start by defining the remaining trace evaluators and the aggregation function that we will use for the first set of experiments, which approach problem 1 by means of different methods.

- **Progress**. We shall consider progress $\rho(t,p)$, already defined in Section 3, as our first metric. Note how progress is 1 if and only if $t \models p$.
- **Exploitation**. Let i be the instant where the FSPA associated with the predicate accepts or rejects the trace t; we define exploitation $e(t,p)$ as $e(t,p) = 1 - \frac{|t|-i}{|t|}$, or the proportion of trace steps that are processed before the underlying FSPA reaches an acceptance decision. Intuitively, we are interested in a predicate showing high values of exploitation in positive examples, where this metric can often be interpreted as implying that the predicate contains relevant information about the reference trace.
- **Reward**. We define the reward $r(t,p)$ as the sum of variations in progress for each of the automaton's execution steps, $r(t,p) = \sum_{i=1}^{|t|-1}(\rho_{\text{FSPA}_p}(t^{i+1}) - \rho_{\text{FSPA}_p}(t^i))$. This value allows us to identify segments in which a player is actively striving to achieve a sub-goal through local variations in the robustness of certain state variables.

Let us look at a concrete example of the above metrics. For instance, if we have that $p = \text{SSEQ}[g_1, g_2, g_3]$, where g_i corresponds to the predicate of reaching a certain location in the level, and in a sample trace the player sequentially visits these locations and then immediately terminates the log (note how this is not necessarily the same as finishing the game; instead, this refers to ending the current play trace), we will observe that $u(t,p) \approx \gamma(t,p) = 1$, since the vast majority of the states in the trace have been relevant to the FSPA, and if the player has additionally managed to steadily approach their goals without straying, we will further notice $r(t,p) \approx 1$. This denotes that the trace successfully models the predicate.

Alternatively, if the player continues to perform other tasks on the trace after having completed the visits to the target points from the predicate, we would perceive a reduction of both the exploitation and the reward of the trace, due to the automaton reaching an early state of acceptance, ignoring all subsequent events, with a progress of 1. This is a symptom that, although the trace models the predicate, the latter is probably not sufficiently informative. In one last scenario, if the player only visits the first goal and then finishes the game, then the exploitation would be 1, since the entire trace is processed by the automaton. However, both the progress and the reward would be fairly low as no FSPA acceptance states would have been encountered. This suggests that the predicate is specifying a more extensive behaviour than what is actually being experienced in the game.

In order to evaluate the quality of a predicate for which the sets of examples and counterexamples in the game have been fixed, we define the following aggregation function:

$$\gamma(P,N,p) = \frac{\sum_{t\in P}(e(t,p) + r(t,p) + \rho(t,p))}{|P|} + \frac{|\{t \in P \mid t \models p\}|}{|P|} + \frac{|\{t \in N \mid t \not\models p\}|}{|N|}. \tag{8}$$

In doing so, we aim to maximise the above metrics for examples, as well as the proportion of examples accepted and counterexamples rejected by the predicate under consideration.

6.1 Game Environment and Recorded States

Liquid Snake [10] is a prototype third-person stealth game developed in the Unity3D engine, in which players control the main character to navigate various levels while collecting valuable items and avoiding detection by robotic guards, security cameras and other surveillance mechanisms. Guards follow pre-set patrol routes, but, if a guard detects the player, they will visit the spot where they last saw them and look around for a short time. As long as the guard keeps noticing the player, or senses them in their vicinity, a suspicion meter will gradually build up. If the suspicion meter reaches a maximum level, the guard will enter an aggressive mode and chase the protagonist until they are captured. The player can use the environment to hide, crouch behind mid-height objects to avoid detection by vision cones, and run to quickly dash through sections of the level before the enemies' suspicion gauges can grow too high. Certain levels also require the player to momentarily allow themselves to be spotted by the robot guards and draw their attention to a secluded spot, forcing them to inspect that location while the protagonist advances through the areas the guards have left unprotected in the process.

In this environment, we register a set of variables as robustness functions on the state of the system every 0.1 seconds of a game in progress. In several of them we make use of the notion of proximity of one object to another within a bounded range, defined as:

$$\delta(o_1,o_2,b,B) = \begin{cases} 1 - \frac{||\vec{o_1 o_2}||}{b}, & 0 \le ||\vec{o_1 o_2}|| < b \\ \frac{||\vec{o_1 o_2}||}{b-B}, & b \le ||\vec{o_1 o_2}|| \le B \\ -1, & B < ||\vec{o_1 o_2}|| \end{cases} \tag{9}$$

where $||\vec{o_1 o_2}||$ represents the Euclidean distance between the points of interest o_1 and o_2. In practice, we take $b = 1, B =$ level size. The variables considered are:

- `objectCollected`$_i$: with a value of 1 if the i-th object has been picked up by the player and $\delta(\text{player}, \text{object}_i, b, B)$ otherwise.
- `goalReached`: same as for `objectCollected`, but only taking on a value of 1 when the player reaches the level exit.
- `safeFromEnemy`$_i$: with a value of -1 if the player is visible within the detection cone of the i-th enemy or camera and 1 otherwise.
- `enemySuspectingPlayer`$_i$: with a value of -1 + awareness, with awareness $\in [0, 1]$ being the proportion of suspicion accumulated by the enemy, and 1 when awareness = 1.
- `playerCrouching`: with a value of 1 if the player is crouching and -1 otherwise.

Within the game, we performed trace recordings of several groups of players interacting with a reference level. This environment, depicted in Figure 1, requires the player to collect all yellow items in order to open the door (blocking the exit to the left of the image) and escape. The lower area of the environment is guarded by a security camera that slowly rotates to cover the room in which it is placed, while the upper area houses a guard who patrols around its central structure with several watch-points that he inspects with greater caution. In this level, we recorded 5 different behaviour patterns defined as follows, with 3 traces per play-style, by having a human tester playing the game according to the instructions given bellow:

- **Style A.** Traces in which the player collects the first two objects, does not crouch to hide, and is then detected by a camera.
- **Style B.** Traces in which the player collects the first two objects, attempts to crouch to hide, but is still detected by a camera.
- **Style C.** Traces in which the player collects the first object, crouches to hide, and then proceeds to collect all other objects in order, while avoiding detection by following the patrolling robot from behind.
- **Style D.** Traces in which the player collects the first object, crouches to hide, and then proceeds to collect all other objects. Follows an order inverse to that of the patrolling robot and is forced to hide behind walls. Sometimes detected, but gets away safely.
- **Style E.** Traces in which the player collects the first object, crouches to hide, and then proceeds to collect all other objects. Follows inverse robot order and is eventually spotted and captured before reaching the exit.

6.2 Temporal Characterisations

With these traces and the variables described above, we apply the algorithms from Section 5 to Problem 1 to try to infer predicates that characterise each of the play styles relative to the rest of the styles. That is, if the styles are given by $S = [A, B, C, D, E]$, then we are considering the problems that take $P = S_i$, $N = \bigcup_{j \neq i} S_j$, $i = 1, \ldots, 5$. In all of them we use the metrics aggregator from Equation 8 and the grammar from Equation 1.

The results obtained can be found in Tables 1, 2 and 3. Each of these tables includes, for every style used as a positive example set: the best predicate found according to the metrics, its fitness

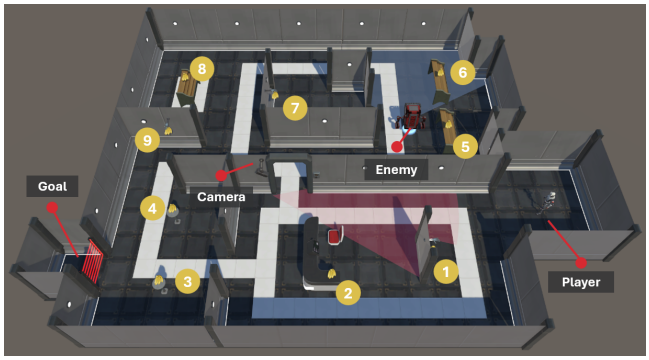

Figure 1: Top View of Reference Environment.

measure normalised to the interval $[0, 1]$ by dividing the value of the metric aggregator by 5, the time taken to reach the solution, and the partial results of each of the terms that make up the search heuristic. Here, $\rho(P)$ refers to progress reached in the predicate by traces in the positive set, $r(P)$ and $e(P)$ are shortened notation for reward and exploitation, respectively, and acc_P, rej_N denote the proportion of traces in the positive and negative examples sets that are accepted or rejected. All of these experiments were run on a PC with 16GB of RAM and an Intel Core i7-9750H, 2.60GHz processor.

In the case of the brute-force algorithm, we choose a maximum expansion depth $D = 6$ to try to strike a balance between expressiveness and computational time. With the given grammar, this leads to process a total of 428,750 terminal nodes in each of the searches; the best one with respect to the chosen fitness metric is reported. As can be noted in the results from Table 1, the time spent is very similar in all cases, at around 6 minutes of computing time. Since this approach systematically explores all possible expansions within the established limit, and these do not vary from one case to another, this result is to be expected and makes it the most stable algorithm in terms of time, although it is important to mention that this would grow exponentially according to the branching factor of the grammar used. The predicates proposed as solutions are fairly modest in this approach given the constraint imposed on the number of derivations of the grammar, and should therefore be considered as broad features of the modelled traces rather than detailed behavioural explanations. Thus, for example, the best individual for the first style is `G(!playerCrouching)`, which corresponds to the property that registered players did not crouch at any time to hide, but does not address the level progress made. Moreover, this very inability to produce complex or highly sequentialised predicates means that the reward values tend to be low: assertions of the type 'globally' result in automata in which it is impossible to increase progress once the state of acceptance has been reached, and eventuality assertions limited to a goal only generate reward in the segment prior to its attainment. In short, this approach produces general explanations that, while optimal in terms of depth, do not yield very fine-grained explanations of the observed behaviour.

In the case of the grammatical evolution algorithm (see Table 2), we choose an initial population size of 100 individuals with individual size of 15, and a total of 100 generations to evolve from this first sample. The time required in this case is markedly shorter than that

Table 1: Brute-force Algorithm Results

Style	Best Individual	Fitness	Time	$\rho(P)$	$r(P)$	$e(P)$	acc_P	rej_N
A	G(!(playerCrouching))	0.81	5m 52s	1	0.03	1	1	1
B	G(!(objectCollected03))	0.76	5m 58s	1	0.03	1	1	0.77
C	F((goalReached)&(playerSafeFromCamera))	0.85	6m 0s	1	0.32	0.99	1	0.92
D	F((objectCollected07)&(goalReached))	0.84	5m 58s	1	0.50	0.99	1	0.69
E	G(F(objectCollected08))	0.76	5m 49s	1	0.34	1	1	0.46

Table 2: Grammatical Evolution Results

Style	Best Individual	Fitness	Time	$\rho(P)$	$r(P)$	$e(P)$	acc_P	rej_N
A	G(!((objectCollected04)|(playerCrouching)))	0.81	13s	1	0.03	1	1	1
B	!(F(objectCollected05))	0.76	1m 18s	1	0.03	1	1	0.77
C	F(objectCollected06)	0.67	3m 45s	1	0.24	0.63	1	0.50
D	F(goalReached)	0.80	31s	1	0.33	0.99	1	0.69
E	F(objectCollected06)	0.68	22s	1	0.24	0.71	1	0.46

of the brute-force algorithm, largely due to the fact that a stagnation state is typically reached quite rapidly when the individuals start to look very similar to each other. Except for time, however, the results obtained with this procedure are generally strictly worse or equal to those of the other algorithms, without this resulting in more interesting predicates from a qualitative point of view.

The MCTS algorithm positions itself as the most successful overall (albeit modestly so), yielding the individuals with the highest fitness in all styles, and with the best metrics in most cases (see Table 3). Here, we take exploration constant $K = \sqrt{2}$, rollout length = 15 and a budget of 300 iterations per expansion. The search capabilities of the algorithm to explore the derivation tree in depth result in predicates that are generally more elaborate than in the other two cases, but at the same time more informative. For instance, in the case of style E, the proposed explanation starts by specifying that the level is not completed, followed by a characteristic notion of order: object 3 is collected first, and then at some point object 8 is collected. This corresponds to the observation that this type of player traverses the top of the level in reverse order to the order in which the guard traverses. In the case of style B, on the other hand, it is indicated that there always comes a point in these traces where the player becomes detected by the camera until the end of the game, which again is a representative explanation of the reason for the defeat of this group. Note how this algorithm generally features significantly higher values for the reward metric, which we intuitively link to the ability of the predicate to capture consistently positive variations in progress across traces, and is a desirable property in individuals exhibiting a good degree of detail.

Something noteworthy here is the time required to run style C, which amounts to about 28 minutes of computation time. Since the budget in this case is specified for each node expansion decision in the derivation tree, and the depth is not initially bounded, this means that it is sometimes necessary to explore a large number of predicates of potentially large size if the rollout leads to a state deep in the structure. Furthermore, it is known that the cost of translating LTL to deterministic automata can grow significantly quickly in practice in both time and size as predicate complexity increases [7], which means that solutions computed at deep nodes may take an unforeseen amount of time. This can be solved by imposing more restrictive conditions on the search, such as limiting the maximum depth of the structure; for the moment, we leave this as future work.

6.3 Tree Clustering

With these same traces, the next step is to apply Algorithm 3 to try to establish a tree structure that explains the most prominent behavioural patterns of the recorded players. In this case, instead of using the grammar from Equation 1, we will introduce a new grammar useful for representing time-ordered sequences of events to demonstrate that our system allows searching over different subsets of predicates driven by more specific questions by simply modifying the reference grammar:

$$\begin{aligned} \phi &:= \texttt{SSEQ(lit-seq)} \\ \texttt{lit-seq} &:= \texttt{lit} \mid \texttt{lit, lit-seq} \\ \texttt{lit} &:= f(s) > 0 \mid \neg(f(s) > 0), \\ \texttt{SSEQ}(l) &= \mathcal{F}(l), \\ \texttt{SSEQ}(l_1, \ldots, l_n) &= \mathcal{F}(l_1 \wedge \mathcal{X}(\texttt{SSEQ}(l_2, \ldots, l_n))) \wedge !l_2 \mathcal{U} l_1. \end{aligned} \tag{10}$$

As can be seen, this new grammar makes use of the SSEQ operator to represent sequences of literals that occur in a strict order. Based on this grammar, we can then use the clustering algorithm to explore what types of behavioural sequences define the traces of T. The result of the analysis can be found in Figure 2. In this case, we have chosen the Silhouette Score [25] as the clustering quality metric, the MCTS-based search algorithm, and $\alpha = 0.51$.

Let us qualitatively analyse the information provided by this trace clustering diagram. From the 15 starting traces, originally introduced without any kind of labelling in the algorithm, the process

Table 3: MCTS Results

Style	Best Individual	Fitness	Time	Steps	$\rho(P)$	$r(P)$	$e(P)$	acc_P	rej_N
A	G(!objectCollected09) & F(objectCollected02 & X(playerSafeFromCamera))	0.89	1m 1s	34	1	0.60	1	1	0.85
B	F(G(!playerSafeFromCamera)) & G(!objectCollected06)	0.88	0m 42s	23	1	0.62	1	1	0.77
C	F(objectCollected09 & objectCollected02 & objectCollected07 & objectCollected04 & X(playerSafeFromCamera) & goalReached & objectCollected06)	0.93	28m 34s	50	1	0.73	1	1	0.92
D	F(objectCollected05 & objectCollected09 & goalReached & objectCollected06 & objectCollected08)	0.87	1m 8s	26	1	0.67	0.99	1	0.69
E	G(!goalReached) & F(objectCollected03 & X(F(objectCollected08)))	0.87	3m 55s	31	1	0.37	1	1	1

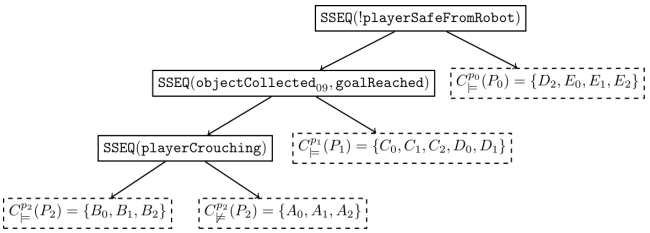

Figure 2: Clustering Diagram for Player Traces.

ends up generating 4 distinct groups based on a total of 3 separation predicates with the newly described grammar. Interestingly, at a preliminary level, the groups established in this way, represented as dashed boxes in the figure, bear a close resemblance to the traces belonging to each of the original play styles, with the only exception being style D, whose traces are distributed between the groups with the E and C styles. Intuitively, this a desirable result, as it suggests that the algorithm is identifying significant patterns that agree with what we consider to be relevant. The new clusters can then be interpreted as:

- Traces in which the player was detected by the robot.
- Traces in which the player collected object 9 and then reached the goal, while not being detected by the robot.
- Traces in which the player crouched at some point, but without collecting object 9 and then reaching the goal, nor being detected by the robot.
- Traces in which player never crouched, nor was detected by the robot, nor collected object 9, and then reached the goal.

As can be seen, each group is defined on the basis of a negation of all previous properties (this follows by construction: had a trace been modelled by a predicate, it would have been discarded in subsequent iterations from that point onwards), plus an additional constraint that refines its characterisation when contrasted with the remaining traces.

7 Discussion and Future Work

The algorithms and experiments described in this paper are not intended as anything beyond initial pilots and proofs of concept to explore what kind of feedback these techniques can generate in quality control and user analysis tasks, as well as what the practical challenges are for introducing them into the design and development loop of a game.

Technically, there remain a number of unresolved issues to be considered for future iterations of the study:

- All the algorithms for Problem 1 are markedly redundant in the searches, for a number of reasons. The first of these is the semantic symmetry of the grammars used, so that, for example, one has that $a \wedge b \equiv b \wedge a$, and the same applies to the disjunction operator. The second, linked to the first, is that at no point are we talking about minimal or simplified predicates according to some system of LTL reduction rules, and so we wind up treating predicates like $\mathcal{F}(\mathcal{F}(p)) \equiv \mathcal{F}(p)$ or $a \vee \neg a \equiv \top$ separately. The same is true for the absence of equivalence rules that would aid in not having to handle pairs of predicates such as $\mathcal{G}(\neg p) \equiv \mathcal{F}(p)$ separately.
- This redundancy significantly affects the time and space cost of the algorithms described here: since the process of translating a predicate in LTL into an automaton can potentially be expensive in both of these ways, calculating automata for equivalent predicates is something that should be avoided as much as possible in the future.
- Moreover, the above algorithms may attempt to evaluate the same predicate more than once due to randomness or construction. The clearest example is in grammatical evolution, where it is possible to generate identical individuals that are then evaluated separately. Similarly, MCTS rollouts can lead to the same terminal nodes. Solution caching is possible to some extent and can alleviate some of the time cost, but in return a potentially significant memory cost may be incurred when the number of cached elements becomes high.

On the other hand, the grammars used for the searches play an integral role in conditioning both the search space, the efficiency of the algorithms and the form of the solutions produced.

- First, a grammar acts as a query in a certain sense, in that by selecting a grammar that generates only predicates with a very specific structure, we are effectively restricting the problem to a more focused and therefore manageable one.

We provide an example of this with Grammar 1 in order to search for predicates that refer to sequences of events across time; while this grammar is not complete in that it does not output every predicate in LTL, it can become even more valuable from a design point of view, by yielding predicates that are simple to interpret and that meet a well-delimited practical question. The same could be done with grammars conditioned on global properties (i.e. events that are true at all times), on logical consequences (events that happen whenever others happen before them), and so on.

- Second, the shape of the grammar can have a significant effect on the quality of the algorithms that employ it, usually by inducing some kind of bias on the probability distribution of the different types of solutions. While beyond the scope of this paper, some early studies on the effect of these properties in the case of grammatical evolution can be found in [20].

Another important point to consider is the metrics used and their effect on the types of solutions obtained. Since we are dealing with unsupervised problems of an exploratory nature, it is complex to define quantitative quality measures that guarantee that the predicates found are truly helpful and insightful; the exploitation, progress and reward metrics are first approximations to understanding which properties might be of interest, but other factors, such as the length or complexity of the predicate, or interactions between different metrics, might be worthwhile depending on how this ill-conditioned notion of fitness is interpreted within this problem.

Acknowledgments

This work was supported by the Spanish Ministry of Science and Innovationnder Grant No.: PID2021-123368OB-I00.

References

[1] [n. d.]. Explainable AI | Royal Society. https://royalsociety.org/news-resources/projects/explainable-ai/

[2] [n. d.]. *Explicit or Symbolic Translation of Linear Temporal Logic to Automata.* Thesis. https://scholarship.rice.edu/handle/1911/71687

[3] A. W. Biermann and J. A. Feldman. 1972. On the Synthesis of Finite-State Machines from Samples of Their Behavior. *IEEE Trans. Comput.* C-21, 6 (June 1972), 592–597. https://doi.org/10.1109/TC.1972.5009015

[4] Boyan Bontchev and Olga Georgieva. 2018. Playing style recognition through an adaptive video game. *Computers in Human Behavior* 82 (May 2018), 136–147. https://doi.org/10.1016/j.chb.2017.12.040

[5] Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. 2012. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games* 4, 1 (2012), 1–43. https://doi.org/10.1109/TCIAIG.2012.2186810

[6] Igor Buzhinsky. 2019. Formalization of natural language requirements into temporal logics: a survey. In *2019 IEEE 17th International Conference on Industrial Informatics (INDIN)*, Vol. 1. 400–406. https://doi.org/10.1109/INDIN41052.2019.8972130

[7] Javier Esparza, Jan Křetínský, Jean-François Raskin, and Salomon Sickert. 2022. From linear temporal logic and limit-deterministic Büchi automata to deterministic parity automata. *International Journal on Software Tools for Technology Transfer* 24, 4 (Aug. 2022), 635–659. https://doi.org/10.1007/s10009-022-00663-1

[8] Sylvain Gelly and Yizao Wang. 2006. Exploration exploitation in go: UCT for Monte-Carlo go. In *NIPS: Neural Information Processing Systems Conference Online trading of Exploration and Exploitation Workshop.*

[9] Olga Grinchtein, Martin Leucker, and Nir Piterman. 2006. Inferring Network Invariants Automatically. In *Automated Reasoning*, Ulrich Furbach and Natarajan Shankar (Eds.). Springer, Berlin, Heidelberg, 483–497. https://doi.org/10.1007/11814771_40

[10] Pablo Gutiérrez-Sánchez, Marco Antonio Gómez-Martín, Pedro A. González-Calero, and Pedro Pablo Gómez-Martín. 2022. Liquid Snake: a test environment for video game testing agents. In *Actas del I Congreso Español de Videojuegos, Madrid, Spain, December 1-2, 2022 (CEUR Workshop Proceedings, Vol. 3305)*, Raúl Lara-Cabrera and Antonio José Fernández Leiva (Eds.). CEUR-WS.org. https://ceur-ws.org/Vol-3305/paper7.pdf

[11] Pablo Gutiérrez-Sánchez, Marco Gómez-Martín, Pedro Gonzalez-Calero, and Pedro Gómez-Martín. 2024. A Progress-Based Algorithm for Interpretable Reinforcement Learning in Regression Testing. *IEEE Transactions on Games* PP (01 2024), 1–10. https://doi.org/10.1109/TG.2024.3426601

[12] D. Hadka. [n. d.]. MOEA Framework: A Free and Open Source Java Framework for Multiobjective Optimization (Version 4.5) [Computer Software]. https://github.com/MOEAFramework/MOEAFramework. [Accessed 24-10-2024].

[13] Mohammadhosein Hasanbeig, Natasha Yogananda Jeppu, Alessandro Abate, Tom Melham, and Daniel Kroening. 2021. DeepSynth: Automata Synthesis for Automatic Task Segmentation in Deep Reinforcement Learning. https://doi.org/10.48550/arXiv.1911.10244 arXiv:1911.10244 [cs, stat].

[14] Branden Ingram, Clint van Alten, Richard Klein, and Benjamin Rosman. 2023. Generating Interpretable Play-Style Descriptions Through Deep Unsupervised Clustering of Trajectories. *IEEE Transactions on Games* 15, 4 (Dec. 2023), 507–516. https://doi.org/10.1109/TG.2023.3299074

[15] Daniel Kasenberg and Matthias Scheutz. 2017. Interpretable Apprenticeship Learning with Temporal Logic Specifications. https://doi.org/10.48550/arXiv.1710.10532 arXiv:1710.10532 [cs].

[16] Xiao Li, Zachary Serlin, Guang Yang, and Calin Belta. 2019. A formal methods approach to interpretable reinforcement learning for robotic planning. *Science Robotics* 4, 37 (Dec. 2019), eaay6276. https://doi.org/10.1126/scirobotics.aay6276

[17] Xiao Li, Cristian-Ioan Vasile, and Calin Belta. 2017. Reinforcement learning with temporal logic rewards. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Vancouver, BC, 3834–3839. https://doi.org/10.1109/IROS.2017.8206234

[18] Naveen Sai Madiraju, Seid M. Sadat, Dimitry Fisher, and Homa Karimabadi. 2018. Deep Temporal Clustering : Fully Unsupervised Learning of Time-Domain Features. https://doi.org/10.48550/arXiv.1802.01059 arXiv:1802.01059 [cs, stat].

[19] Robert I McKay, Nguyen Xuan Hoai, Peter Alexander Whigham, Yin Shan, and Michael O'neill. 2010. Grammar-based genetic programming: a survey. *Genetic Programming and Evolvable Machines* 11 (2010), 365–396.

[20] Miguel Nicolau and Alexandros Agapitos. 2018. *Understanding grammatical evolution: Grammar design.* 23–53. https://doi.org/10.1007/978-3-319-78717-6_2

[21] Michael O'Neill and Conor Ryan. 2003. Grammatical Evolution: Evolutionary Automatic Programming in a Arbitrary Language, volume 4 of Genetic programming.

[22] Yash Vardhan Pant, Houssam Abbas, and Rahul Mangharam. 2017. Smooth operator: Control using the smooth robustness of temporal logic. In *2017 IEEE Conference on Control Technology and Applications (CCTA)*. IEEE, 1235–1240.

[23] Ritam Raha, Rajarshi Roy, Nathanaël Fijalkow, and Daniel Neider. 2022. Scalable Anytime Algorithms for Learning Fragments of Linear Temporal Logic. In *Tools and Algorithms for the Construction and Analysis of Systems*, Dana Fisman and Grigore Rosu (Eds.). Springer International Publishing, Cham, 263–280. https://doi.org/10.1007/978-3-030-99524-9_14

[24] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. Model-Agnostic Interpretability of Machine Learning. https://doi.org/10.48550/arXiv.1606.05386 arXiv:1606.05386 [cs, stat].

[25] Peter J. Rousseeuw. 1987. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *J. Comput. Appl. Math.* 20 (1987), 53–65. https://doi.org/10.1016/0377-0427(87)90125-7

[26] Rajarshi Roy, Jean-Raphaël Gaglione, Nasim Baharisangari, Daniel Neider, Zhe Xu, and Ufuk Topcu. 2023. Learning Interpretable Temporal Properties from Positive Examples Only. *Proceedings of the AAAI Conference on Artificial Intelligence* 37, 5 (June 2023), 6507–6515. https://doi.org/10.1609/aaai.v37i5.25800

[27] Rukma Talwadker, Surajit Chakrabarty, Aditya Pareek, Tridib Mukherjee, and Deepak Saini. 2022. CognitionNet: A Collaborative Neural Network for Play Style Discovery in Online Skill Gaming Platform. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22)*. Association for Computing Machinery, New York, NY, USA, 3961–3969. https://doi.org/10.1145/3534678.3539179

[28] Josep Valls-Vargas, Santiago Ontañón, and Jichen Zhu. 2015. Exploring Player Trace Segmentation for Dynamic Play Style Prediction. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 11, 1 (2015), 93–99. https://doi.org/10.1609/aiide.v11i1.12782

[29] Marcell Vazquez-Chanlatte, Susmit Jha, Ashish Tiwari, Mark K. Ho, and Sanjit A. Seshia. 2018. Learning Task Specifications from Demonstrations. https://doi.org/10.48550/arXiv.1710.03875 arXiv:1710.03875 [cs].

[30] Junyuan Xie, Ross Girshick, and Ali Farhadi. 2016. Unsupervised Deep Embedding for Clustering Analysis. https://doi.org/10.48550/arXiv.1511.06335 arXiv:1511.06335 [cs].

[31] Xuan Zhao and Marcos Campos. 2021. Reinforcement Learning Agent Training with Goals for Real World Tasks. *arXiv:2107.10390 [cs]* (July 2021). http://arxiv.org/abs/2107.10390 arXiv: 2107.10390.