# Probabilistic Model Checking for Activity Recognition in Medical Serious Games

Thibaud L'Yvonnet, Elisabetta de Maria, Sabine Moisan, Jean-Paul Rigault

# Probabilistic Model Checking for Activity Recognition in Medical Serious Games

Thibaud L'Yvonnet[1], Elisabetta De Maria[2], Sabine Moisan[1], and Jean-Paul Rigault[1]

[1] *STARS UCA-INRIA, Sophia Antipolis, France*
*{thibaud.lyvonnet,sabine.moisan,jean-paul.rigault}@inria.fr*
[2] *MDSC UCA-I3S, Sophia Antipolis, France*
*edemaria@i3s.unice.fr*

**Abstract**

Human activity recognition plays an important role especially in medical applications. This paper proposes a formal approach to model such activities, taking into account possible variations in human behavior. This approach is based on discrete-time Markov chains enriched with event occurrence probabilities. We use the PRISM and Storm frameworks and their model checking facilities to express and check interesting temporal logic properties concerning the dynamic evolution of activities. We illustrate our approach on two *serious games* used by clinicians to monitor Alzheimer patients. This paper focuses on the suitability of such a formal approach to model patients' behavior, to check behavioral properties of medical interest, and on the respective advantages of the PRISM and Storm frameworks. Our goal is to provide a new tool for doctors to evaluate patients.

***Index terms***— activity description, probabilistic model, model checking, serious games, bio-medicine

## 1 Introduction

Human behavior recognition has become a crucial research axis [1] in many contexts, such as visual surveillance, smart homes [2], or cars [3]. While much has been done on simple *action* recognition, especially in computer vision [4], we rather target complex *activities* that embed several combinations of actions. We consider an activity as a set of scenarios describing possible behavioral variants. The goal of recognition is to identify which scenario is played by analyzing input data coming from different sensors (in our case mostly video cameras but also binary or audio sensors).

Recently, serious games were introduced in the health domain to help evaluate the performances of patients with neuro-degenerative diseases [5]. Behavior

and performance displayed by patients during these games give indications on their disease level. In the line of [6], we expect that our approach to categorize the behavior of patients could provide doctors with significant information to complete their diagnosis. We chose to represent serious game activities as discrete-time Markov chains whose edges can be decorated with probabilities [7]. All the possible actions do not occur with the same probability, since some behaviors are most frequent than others. We propose to quantify the likelihood of these variations by associating probabilities with some actions. The recognition process itself remains deterministic since, at recognition time, only one scenario will be played and recognized.

To implement the Markov chains, we use existing tools, namely PRISM [8] and Storm [9], that both allow model-checking. Thus we can define properties of interest on a model and verify them. The advantage is that formal modeling and model checking techniques provide probabilities associated with *classes* of paths and allow to test universal properties on a model, contrary to simulation techniques which only deal with existential properties. However, a drawback of model-checking may be time explosion. Fortunately, Markov chains do not impose to associate a real duration with each action, contrarily to, e.g., timed automata. Thus we can manage the time in the activity models, restricting it only to the instants when significant events occur, hence reducing the duration of simulations or model checking.

We do not use formal modeling and model-checking in the classical engineering way, i.e. to develop and validate new software. Our perspective is different: we do not address serious game *design*, but we rather apply model checking to *existing* games that have already demonstrated their utility in medical practice to qualify the behavior of their players.

We modeled three serious games used by our medical partners to analyze the behavior of Alzheimer patients. These games target different brain functions and also raise different formal modeling issues. Due to space limitation, we only present two of them in this paper. We implemented the game activities as discrete-time Markov chains using the PRISM language [8], which is also accepted as input by Storm. We used temporal logic to encode some relevant properties and we applied model checking techniques to automatically validate the models with respect to these properties as well as to infer the probabilities of some interesting paths.

These models represent all the envisioned paths (possible sequences of actions from the patient or the environment), for both common and uncommon behaviors. Some actions are certain and need no probabilities, other ones depend on the disease stage of the patient and are associated with probabilities. The goal is to deduce how relevant to the disease stage the scenario played by a patient is. For example, if a patient known to be healthy plays a "mild cognitive impairment" (MCI) scenario, our system is able to detect this inconsistency.

Before performing clinical tests on patients, the focus of this paper is to validate our modeling approach, to explore its interest in medical applications, and to compare the respective advantages of PRISM and Storm concerning model-checking.

The paper is organized as follows. Section 2 details discrete-time Markov chains and their support in the PRISM and Storm model checkers. Section 3 illustrates our usage of model checking in the medical domain. Sections 4 and 6 respectively introduce the models of two games as discrete-time Markov chains. Sections 5 and 7 apply model checking to these models. Section 8 compares the modeling strategies for the games and the performances of PRISM and Storm. Section 9 concludes and opens future research directions.

# 2 PRISM and Storm Model Checkers

Several probabilistic model checkers exist (e.g., PRISM [8], Uppaal [10], Storm [9], or PAT [11]). We decided to rely on PRISM and Storm, which are tools for formal modeling and analysis of systems with random or probabilistic behavior. PRISM is well established in the literature and has already been used to describe human activity [12] while Storm is more recent. Both tools support several types of probabilistic models, discrete as well as continuous.

In this work we use discrete-time Markov chains (DTMC), which are transition systems augmented with probabilities. Their set of states represents the possible configurations of the system being modeled, and the transitions between states represent the evolution of the system, which occurs in discrete-time steps. Probabilities to transit between states are given by discrete probability distributions. Markov chains are memoryless, that is, their current state contains all the information needed to compute the future states. More precisely:

**Definition** A Discrete-Time Markov Chain over a set of atomic propositions $AP$ is a tuple $(S, S_{init}, P, L)$ where $S$ is a set of states (state space), $S_{init} \subseteq S$ is the set of initial states, $P : S \times S \to [0, 1]$ is the transition probability function (where $\sum_{s' \in S} P(s, s') = 1$ for all $s \in S$), and $L : S \to 2^{AP}$ is a function labeling states with atomic propositions over $AP$.

## 2.1 Modeling Language

PRISM provides a state-based modeling language that Storm accepts as input [13]. A model is composed of a set of *modules* which interact with each other. The state of a module is given by the values of its *local variables* and the global state of the whole model is determined by the union of the local states of all its modules. The dynamics of each module is described by a set of commands of the form: $[\,]guard \to prob_1 : update_1 + ... + prob_n : update_n$; where $guard$ is a predicate over all the variables of the model, corresponding to a condition to be verified in order to execute the command and each $update$ indicates a possible transition of the model, achieved by giving new values to variables. Each $update$ is assigned a probability and, for each command, the sum of probabilities must be 1. The square brackets at the beginning of a command can either be empty or contain labels representing *actions*. These actions can be used to force two or more modules to transit simultaneously.

Let us consider a simple DTMC with a unique integer variable $y$ which ranges over $\{0, 1\}$. Whenever $y$ equals 0, the value of $y$ remains at 0 with probability 0.5 or switches to 1 with probability 0.5. This can be expressed by the PRISM command: $[\ ]y = 0 \rightarrow 0.5 : (y' = 0) + 0.5 : (y' = 1)$. PRISM models can be extended with *rewards* [14], associating (integer) values with model states or transitions. For instance, in the previous example we can associate a reward with $y$: each time $y = 1$, the reward is incremented. This is expressed in PRISM with the following annotation: $y = 1 : 1$;.

## 2.2 Probabilistic Temporal Logic

The dynamics of DTMCs can be specified thanks to the PCTL* (Probabilistic Computation Tree Logic) temporal logic. PCTL* extends the CTL* logic (Computation Tree Logic) with probabilities. The following state quantifiers are available in PCTL*: **X** (next time), **F** (sometimes in the future), **G** (always in the future), and **U** (until). Note that the classical path quantifiers **A** (forall) and **E** (exist) of CTL* are replaced by probabilities. Thus, instead of saying that some property holds for all paths or for some paths, we say that a property holds for a certain fraction of the paths.

The most important operator in PCTL* is **P**, which allows to reason about the probability of event occurrences. A property `P bound [prop]` is true in a state $s$ if the probability that the property *prop* holds in all the paths from $s$ satisfies the bound *bound* (where a bound is a comparison operator followed by a probability value). As an example, the PCTL* property `P= 0.5 [X (y = 1)]` holds in a state if the probability that $y = 1$ is true in the next state equals 0.5. Furthermore, the **P** PRISM operator can be used as `P=?[prop]`; this computes the probability for *prop* to occur. For instance, the property `P=?[G(y = 0)]` expresses the probability that `y` always equals 0.

PRISM and Storm also support the notion of integer user-defined "rewards" which can be seen as counters. They do not impact the number of states and transitions of the model nor its behavior. The **R** operator allows to retrieve reward values. PRISM provides model checking algorithms to automatically validate DTMCs over PCTL* properties and reward-based ones. In particular, it deals with the PCTL* fragments PCTL (Probabilistic Branching Time Logic) and PLTL (Probabilistic Linear Logic). Storm can only deal with PCTL properties.

PRISM and Storm provide different model checking engines. Among them, we use PRISM *hybrid* and *explicit* engines as well as Storm *sparse* engine. PRISM hybrid engine uses symbolic data structures, such as binary decision diagrams (BDDs) and explicit data structures, such as sparse matrices, which makes it a partly symbolic engine. PRISM explicit engine and Storm sparse engine both use only explicit-state data structures. Since they do not use symbolic data structures for model construction, they can perform model checking in cases where other engines fail.

In addition to classical model checking tools, PRISM offers the possibility to run *experiments*, which is a "way of automating multiple instances of model

checking", according to the PRISM authors. This feature allows users to obtain curves displaying the results of a property with respect to one or several variables. Besides, PRISM also proposes *statistical model-checking*, a way to test properties through simulations.

# 3 Model Checking in the Medical Domain

Real-time activity recognition is particularly relevant for serious games: the expected behavior is well identified and it is possible to rely on different sensors while playing the game. In the health domain, serious game can be used to train medical staff [15] or to help diagnose and treat patients [16, 17]. When formally modeling a patient playing a serious game, one can associate probabilities with actions to characterize a healthy or a pathological behavior. These probabilities are initially defined according to physicians' past experience. Properties can then be written to extract relevant data to be compared, first, with experimental results in order to refine the model and, ultimately, with real patients results.

Model-checking techniques have been used in the medical domain to verify safety critical software or artifacts and it becomes a crucial step in medical tools engineering [18]. In these applications, the model of the patient mainly relies on physiological data (blood pressure, pulse rate, respiratory rhythm...) that may influence the modeled device operation or the ongoing medical process. Contrarily to this kind of utilization, we do not address model checking of serious games software for design purposes, but we model the actions and behavior of patients playing these games in order to help physicians refine their diagnosis. Our goal is to observe and record data on the patient's interaction with the game. The type of interaction and its differences with a "normal" game session will help physicians diagnose a possible disease.

This usage of model checking is related to human activity recognition. It is close to [19] where model checking is used to detect differences between actual medical actions and ideal ones as described in a formal model of clinical guidelines. Similarly, in [20] the authors apply temporal logic and model checking to support automated real-time recognition of activities of daily living. These approaches apply model checking as we do, both to verify properties of a predefined formal model of activity and to confront it with real data coming from patients or doctors activities. However, they do not rely on probabilities, which is essential in our case to accurately differentiate the possible patients' behaviors.

We model the behavior of a patient in serious games with discrete-time Markov chains (DTMCs). To the best of our knowledge, DTMC models are barely used for the description of human behavior; yet we can cite [21] that uses DTMC to describe pedestrian trajectories. In computer vision, Hidden Markov Models (HMMs) are a popular approach for the description of activities [22, 23]. However, neither PRISM nor Storm (nor most probabilistic model checkers) allow to check temporal logic properties over HMMs.

As use cases, we consider serious games currently used by our medical part-

ners to analyze the behavior of Alzheimer patients. These games were designed to provide fun training tools for patients with mild cognitive impairment, but each one targets a different brain function. Thus they constitute a good panel of tests that can complement the set of validated neuro-psychological tests usually conducted by clinicians. In this paper we describe the modeling of two of them: the *Code game* [5] and the *Inhibitory Control game* [24].

As we started modeling with PRISM modeling language, we had to cope with one of its limitations: in PRISM Markov chains, it is not possible to put a limit on the number of times the model can loop over a state. Even with a low probability on the loop transition, there is still a risk for a simulation to never quit the loop. To avoid this issue, all the possible states must be explicitly represented in the model, leading to an "unwound" model. But this unwinding turns out to be an advantage to implement the effect of patients' fatigue. Indeed as the game proceeds, patients are likely to become tired, thus the probabilities of some actions should vary accordingly. We included in all game models a simplified yet realistic fatigue profile based on *pre-computed* probabilities at each time step of the game in the unwound model. As PRISM modeling language can be used as input for the Storm model checker, all presented models are verified with both PRISM and Storm.

The games, their models, and some interesting properties on these models are detailed in the next sections.

# 4 Code Game Model

The Code game [5] targets selective and sustained visual attention functions. It runs on a touch-pad. Patients are asked to match a random picture displayed in the center of the touch-pad with the corresponding element in a list of pictures at the bottom of the screen (see figure 1). If the patient chooses the right picture, a happy smiley is displayed and a new picture is proposed. Otherwise a sad smiley is displayed and the patient is asked to try again. If the patient does not interact quickly enough with the touch-pad (more than 10 seconds of inactivity), the game prompts her to choose a picture. Whenever the patient exits the game zone, the game is aborted.

A game session lasts at most five minutes (three-hundred seconds), thus we know that there will be a finite number of states in the Markov chain. Hence, in the PRISM model, we made the assumption that a patient needs at least three seconds to select a picture (minimum time needed to think of which picture to choose and to touch the screen to select it).

## 4.1 Model Design

With the previous assumption, we translated the time constraint of three-hundred seconds into a maximum number of actions (or events) that can happen in a scenario. If the patient keeps on selecting pictures, a smiley (happy or sad) is displayed. This event is called *selection* and it cannot happen more than a


Figure 1: Display of the Code game.

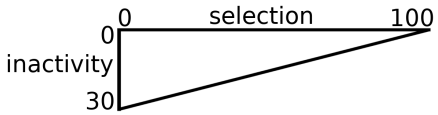


Figure 2: Combinations of events triangle.

hundred times in a row ($300/3 = 100$). On the other hand, if the patient does not interact with the game for ten seconds, the system displays a message. This event is called *inactivity* and it cannot happen more than thirty times in a row ($300/10 = 30$).

To represent all the combinations of these two events, we picture a right-angle triangle (figure 2). The edge of length one hundred (representing the scenario of a succession of *selections*) and the edge of length thirty (representing the scenario of a succession of *inactivities*) form the perpendicular sides of the triangle. Each state of this triangle, except those on the hypotenuse, has three exclusive possible transitions: either increment *selection* and move on the *selection* axis, increment *inactivity* and move on the *inactivity* axis, or leave the game. To represent the action of the patient leaving the game before the end of the five minutes we use a Boolean variable called *quit_game*. If it is true, the state previously reached in the triangle is considered as the final state of the game session.

All states on the hypotenuse represent the end of the five minutes of the game. The only possible transition from them is equivalent to *quit_game*.

## 4.2 PRISM Implementation

The model is composed of a single module which includes three integer variables representing the location of the patient and the events of selecting a picture or of being inactive for more than 10 seconds (called respectively *location*, *selection*, and *inactivity*). For instance, *location* ranges from 0 to 2 (in the room, in the gaming area or outside it) and *selection* ranges from 0 to 100, where the value $i$ represents the fact that the patient has had $i$ interaction(s) with the game so far.

To take advantage of PRISM rewards, we use Boolean variables and associated rewards to represent the other events of the game. The event "a happy (resp., sad) smiley is displayed" for a good (resp., bad) answer is represented by the variable *happy_ smiley*, with reward *Happy_ smiley_ reward* (resp., *sad_ smiley*, with reward *Sad_ smiley_ reward*). The event "the patient leaves the game area before the end of the five minutes" is represented by *quit_ game*, with reward *Leave_ game_ reward*. The event "the console displays a message after ten seconds of inactivity" is represented by *non_ interaction*, with reward *Non_ interaction_ reward*.

Only one of these Boolean variables at a time can be *true*, which means that the event it represents happened, thus the associated reward is incremented. The amount of time spent in the game by the patient is represented by *Gaming_ time*. Its reward is more complex than the others because it increases by three units for each good or bad answer and by ten units for each inactivity message displayed by the console. A global Boolean variable *time_ Is_ Over* is also defined to ease the readability of the module. It determines whether the maximum number of actions that a patient can perform has been reached.

Depending on the values of all the previous variables, the state of the patient can go through different transitions. Some of them have attached probabilities. The transitions taken along a path describe the patient's behavior in a specific scenario. These probabilities come from discussions with medical staff and represent a typical patient with mild cognitive impairment. For instance, the probability to give a good (resp. bad) answer at the beginning of the game (when the patient is not tired yet) is around 0.5 (resp. around 0.25).

In this model, for 5 minutes of game, PRISM generates $7,903$ states and $21,466$ transitions, which is easily tractable.

# 5 Temporal Logic Properties for Code Game

In the previous model we encoded and tested several properties in PCTL, both to verify the model and to provide interesting medical indications. This section presents two properties of the last category.

The first property concerns interaction of the patient with the game. Among all properties of this model, this one is the longest to compute both in PRISM and in Storm.

**Property 1. PRISM.** What is the probability for a patient to choose the correct picture exactly once and to never choose a good one again until the end of the game?

$$P =?[(F\ happy\_smiley)\ \&\ (G\ (happy\_smiley\ =>\ (X\ G\ !happy\_smiley\ \&\ !quit\_game)))]$$

The probability to verify this property is $1.6475\times10^{-9}$ in PRISM which is rather low. The complexity of this property comes from the nesting of $G$ operators that makes this formula impossible to verify with Storm because it does not support

this type of nesting (specific to PLTL formulas). Thus we wrote another formula that checks the same paths for Storm.

**Property 1.** **Storm.** What is the probability for a patient to get only one good answer reward until the end of the game?

$$P =?[true\ U\hat{}\{rew\{"Happy\_smiley\_reward"\} <= 1\ ,$$
$$rew\{"Happy\_smiley\_reward"\} >= 1\ ,$$
$$rew\{"Leave\_game\_reward"\} <= 0\}\ (location = 2)]$$

This formula relies on probabilities to gather rewards, which is not supported by PRISM. The probability to verify this property is exactly the same as in PRISM, $1.6475 \times 10^{-9}$.

Concerning medical interpretation, in the case of a cohort selection based on this model, MCI patients are not expected to choose only one right answer and then stay until the end of the game without exiting. If a cohort differs too much in the frequency of this behavior, practitioners must discard or deeply change it. Otherwise, the risk to perform a clinical test on the wrong sample of population is too high.

The second property is relative to the quality of the actions (correct or not) that can be performed by a patient. It provides an average "score" for the model.

**Property 2.** Is the average amount of good responses given by a patient greater than or equal to, e.g., 30?

$$R\{"Happy\_smiley\_reward"\} >= 30[F\ (location = 2)]$$

This property is *true* both in PRISM and Storm for our model.

Still in the case of a cohort pre-selection, the group of patients should obtain an average "score" similar to the one given in this property. If the score differs too much from this result, the cohort must be rejected.

# 6 Inhibitory Control Game Model

The Inhibitory Control game is part of the TAE Web platform [24]. It targets the inhibitory control function whose role is to inhibit some reflexes and which is affected in several neuro-degenerative diseases.

Figure 3: Display of the Inhibitory Control game signals. The coin is the target and the warning sign is the decoy.

In this game, the patient must click exactly once, as fast as possible, on a target (figure 3 left) that appears on the screen for a short period of time. Knowing that human's visual reaction time for a healthy adult performing a "go/no-go" task is around 259 ms [25], the game considers a possible anticipation from the patients and refuses to award a point if they clicked in less than 259 ms after the occurrence of the target. As long as the patient does not click and as long as a new signal does not appear, the patient can click and score a point. A second signal, a "decoy" (fig. 3 right), tests the inhibitory control function. It is a warning sign that may appear instead of the target. In this case the patient must *not* click on it, despite a color similar to the target. The difficulty relies on both the brevity of the occurrence of signals and on their common color. A patient who clicks on every decoy may have an inhibitory control problem. The delay between signal occurrences is random within a given range, defined at game configuration. There is a short training phase at the beginning of each session: three targets and a decoy are presented to the patient, but the game does not register the patient's results. Then the "real" game starts and the results are stored.

We chose the following settings for the Markov chain model: 300 ms of display time for each signal, $2000 \pm 1500$ ms separating two signals (not constant), 10 occurrences of target and 5 of decoy, 3 training occurrences of target and 1 of decoy. These parameters are the same for all players.

## 6.1 Model Design

To model this game we consider it as a succession of events, each one corresponding to the occurrence of a signal. Since the model only transits on signal occurrences, to determine the current scenario we need to know the previous action of the patient. Thus, for each event, we take into account the action of the patient since the last event.

We consider six potential actions: no click, one click but with anticipation, one fast click, one slow click, one double click, and several clicks. Given the rather short duration of the game (up to 72.2 s), we do not consider the possibility to prematurely leave the game. The initial probabilities associated with the different actions depend on two factors: the nature of the previous signal and the time elapsed before the occurrence of the new signal. To represent this time in a Markov chain we consider that the time elapsed between two signals can be short, medium, or long. This information is reflected in the label names of the transitions between states. The nature of the signal is reflected by both the label and the state variables. The probabilities of the different actions also depend on the time spent in the game to take into account the fatigue factor.

## 6.2 PRISM Implementation

Contrary to the previous game, the patient behavior strongly depends on the game one. Thus we used two separate modules, one for the game and one for the patient to separate the two concerns. We also added an observer module to

facilitate the computation of probabilities. The implementation thus contains three PRISM modules.

The **game module** is composed of seven Boolean and two integer variables. The integer variables count the number of targets (resp. decoys) that appeared during the game session. The main Boolean variables determine if the next signal will appear in a short, medium, or long time span, if the next transition will lead to a target or to a decoy, or if the game will reach its end. Both the nature of the next signal and its speed are decided using probabilities. For each signal, the probabilities for it to be fast, medium, or slow are identical ($\frac{1}{3}$). The probability for the next signal to be a target or a decoy depends on the number of remaining signals.

The implementation contains 24 transition labels to represent all possible combinations of features related to patient actions and signals (such as occurrence speed, type of signal, etc.). The most important label is *transiting*, which is associated with transitions updating some of the previous Boolean variables. Other labels are associated with transitions leading either to a new signal or to the end of the game.

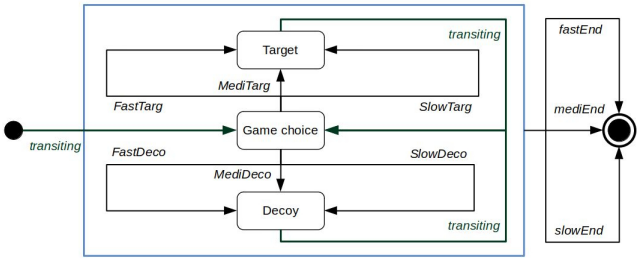


Figure 4: Simplified automaton of the Inhibitory Control game.

Figure 4 depicts the main transitions of the model. The label names are a combination of keywords indicating a fast, medium, or slow occurrence of a target or a decoy, either in the training phase or in the regular game phase. All the labels of this module are used to synchronize with the patient module.

The **patient module** is composed of fifteen Boolean variables that mainly (i) describe actions of the patient between two consecutive signals, (ii) memorize the previous signal to determine whether the patient displayed the correct behavior. To manage every transition of the model, we spell them out for each possible combination of the following elements: nature of the previous signal, nature of the new signal, and occurrence speed of the new signal. When the end of the game is about to be reached, we consider the combination of the nature of the previous signal and the speed at which the end screen will appear. Depending on the transition, the probabilities associated with the actions vary.

The probability for the *anticipate* action is computed taking into account the time lap preceding the previous signal. The longer it is, the higher the probability, inducing a global decrease of all other action probabilities. The probabilities for *unique click* (fast or slow), *several click*, *double-click*, and *no*

*click* actions mainly depend on the nature of the signal and on the time elapsed since the beginning of the game. These probabilities are computed with different functions depending on the phase of the game (training or regular phase). These functions refer to variables belonging to the observer module.

The **observer module** is mainly composed of integer variables that take into account the speed of signal appearance, the delay between signals, and the number of occurrences of targets or decoys. These variables facilitate the expression of interesting behavioral properties in the next section.

In this model, for a total of 13 targets and 6 decoys, PRISM generates 485, 347 states and 2, 640, 864 transitions, which is still tractable with a standard modern computer.

# 7 Temporal Logic Properties for Inhibitory Control Game

Table 1 summarizes the model variables used in this section. We present first a reward-based property that gives an idea of the overall performance of a patient during the game.

| Variable | Representation |
|---|---|
| *prev_none* | Boolean true before the first signal. |
| *prev_targ*<br>*prev_deco* | Boolean indicating if the previous signal was a target or a decoy. |
| *transiting* | Boolean indicating if the model is in a transition state. |
| *not_click*<br>*click_fast*<br>*click_slow* | Boolean indicating the type of patient's action. |
| *num_action* | Integer counting the number of actions done by the patient. |
| *next_end* | Boolean indicating the end of the game. |
| *game_on* | Boolean indicating if the game is on or off |

Table 1: Excerpt of Inhibitory Control Game model variables.

**Property 1.** What is the average accumulation of good answers on targets at the end of the game?

$$R\{\text{"}good\_on\_target\text{"}\} =? \ [F \ (!game\_on \ \& \ next\_end)]$$

The result for this property is 5.55 for both PRISM and Storm. This result is consistent with a modeled patient who has issues playing this game and probably needs further examination regarding the inhibitory control function.

We also used the "run experiment" tool of PRISM (which does not exist in Storm) to obtain diagrams that summarize the evolution of the patient's good

or bad actions in time. The following property explores the evolution of the probabilities during the game, it provides a good understanding on how the fatigue influences the patient's behavior.

**Property 2.** What is the probability to click only when required for the game signal number $i$?

$$P =? [\ F\ (num\_action = i\ \&\ ((prev\_none\ \&\ not\_click) \\ |\ (prev\_targ\ \&\ (click\_fast\ |\ click\_slow)) \\ |\ (prev\_deco\ \&\ (not\_click)))\ \&\ !transiting)]$$

This property evaluates the probability to perform a good action in the game and a similar property can be written to check the probability to perform a bad action. Figure 5 shows the results for these two properties. It gives a global idea of the patient's behavior and clearly shows the good effect of the training phase between action 2 and 6 but, starting from action 10, the patient does more bad actions than good ones.

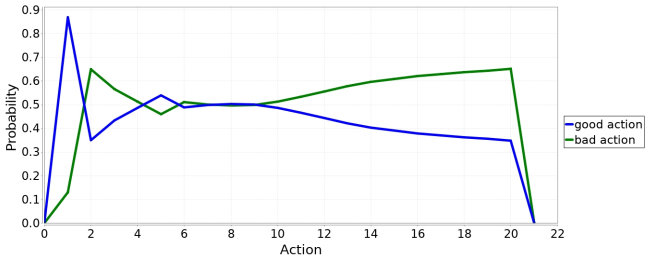


Figure 5: Probability to perform a good or a bad action for each instant when an action is expected from the patient. Here, action number 1 on the horizontal axis is the one recorded before the occurrence of the first signal.

# 8 Experiment Feedback

Specifying an activity formal model suitable for model checking is not straightforward and there is no unique modeling approach. This process highly depends on the modeler's personal experience. This paper described our modeling experience for two different serious games. However, we are aware that this approach is not the only possible one.

A game may necessitate more than one model, mainly one for the patient's behavior and one for the program implementing the game. The patient model should represent all possible behaviors, including the ones that deviate from the rules. The game model should represent the intrinsic game strategy and also possible reactions to patients' actions. These two models must communicate and exchange various kinds of information. To this end, we used the PRISM synchronization mechanism on transition labels which corresponds to a *rendez-vous* communication: the first module which reaches a transition waits for the

other ones to reach the same transition. On the formal side, all the models must satisfy the model checking requirements, in particular a bounded time and a finite (possibly huge) state space.

When modeling our case study games, we implemented different modeling strategies corresponding to the specific features and requirements of each game. We identified three different factors that impact the difficulty to model a serious game: time, past event dependency, and action randomness. These factors, even more when they are combined, increase both game modeling difficulty and model complexity. However, cognitive serious games have usually a rather short duration and few possibilities of answers. Hence, we do not face huge scalability issues.

## 8.1 Game Models

In the Code game, the answer for each picture is right or wrong and does not depend on the picture itself nor on past events. This game was rather easy to model since its only factor of difficulty is its timed aspect. In this kind of games, only the patient's model is needed: it is a simple state machine which is the same for all the game steps. Of course time must be discretized.

The Inhibitory Control game requires two models, one representing the game sequence of events, the other the patient's behavior. Its game model is rather complex because it must incorporate the random selections of both the signal to display and the delay to display it. This random behavior greatly increases the state space. This game was the most complicated one to model because it combines the three factors of difficulty. Verifying this model with PRISM is more time-consuming than the other game (up to 1 or 2 seconds compared to a few one hundredths of seconds). There are even cases when the PRISM computation seems to never stop, whereas Storm terminates in a reasonable time.

The PRISM code of the games can be found at `https://gitlab.com/ThibLY/activity-recognition-modeling`.

## 8.2 PRISM and Storm Comparison

We used both PRISM and Storm model checkers (the hybrid and explicit engines of PRISM and the sparse engine of Storm) to verify all properties on the two games. The computations times are rather different (usually less for Storm), as shown in Table 2 for the properties presented in this paper, except for property 2 of Inhibitory Control game which is specific to PRISM "run experiment" tool.

For all properties in table 2, Storm is faster than PRISM, especially for properties 1 of both games. These properties require a thorough exploration of the model to find the specified states. Our intuition is that, at least for the Code game, since states and paths cannot be easily compacted, the decision diagrams used to build models in the (default) hybrid engine of PRISM cannot be fully

| Property | PRISM | Storm |
|---|---|---|
| Code Game | | |
| *Property 1* | 1.92 | 0.28 |
| *Property 2* | 0.016 | 0.005 |
| Inhibitory Control Game | | |
| *Property 1* | 2.428 | 0.534 |

Table 2: Comparison PRISM/Storm computation times in seconds.

exploited. The explicit data structures used to build models by the (default) sparse engine of Storm seem more suitable for our case studies.

In some cases, as for property 1 of Code Game, it was not possible to encode the same formula in PRISM and Storm because they do not support the same fragments of PCTL$^*$. In particular, Storm does not accept PLTL properties (where several state quantifiers can be nested), but some of them can be encoded in Storm using its reward-bounded feature.

In a few other cases we did not get exactly the same result in PRISM and Storm. As matter of fact, this is a general problem in probabilistic model checking, where it is not usually possible to have a good guarantee of the accuracy of the result. This is due to the representation of numbers and to choices made in the resolution methods.

# 9 Conclusions and Future Work

This paper targets complex human activity recognition, which remains a challenging topic [26]. We propose a formal approach based on discrete-time Markov chains to model human activities. Important properties of such models can be automatically verified thanks to model checking. The proposed approach complements the main existing ones in the field of activity recognition. We use probabilities to explore paths associated with different behaviors.

We applied this approach to patients playing serious games in the medical field. The goal is to differentiate their level of damages among different cognitive functions. We identified three prospective applications of our approach. First, to evaluate a new patient before the first diagnosis of doctors, we can compare her game performance to a reference model representing a "healthy" behavior. Second, to monitor known patients, a customized model can be created according to their first results, and, over time, their health improvement or deterioration can be monitored. Finally, to pre-select a cohort of patients, a reference model can determine, in a fast way, whether a new group of patients belongs to a specific category.

We selected several serious games for MCI patients, only two of them being presented in this paper. With the help of clinicians, we encoded them as DTMCs in PRISM, and we tested meaningful PCTL properties thanks to the PRISM and Storm model checkers. Modeling these games allowed us to validate our approach and to test its performance for different applications. To evaluate the

scalability of the models, we tested them with different (realistic) parameters (e.g., more signals in the Inhibitory Control Game and a longer duration for the Code Game). In both cases, the size of the model remains tractable. It also demonstrated that the apparent complexity of a game does not impact the difficulty of the modeling task. The results encourage us to pursue our research on behavior modeling for patient analysis.

We used several model-checker engines from PRISM, but also from Storm to verify medically oriented properties. We chose Storm because its model checker accepts several modeling languages as input, including the PRISM one.

The contribution of this paper is a mandatory first step before demanding clinical studies with patients playing the games. The probabilities in our models are initially given by medical practitioners and need to be updated according to real clinical experimentation results, in order to obtain more realistic models and to provide more accurate predictions. To this end, we set up different reference profiles (such as mild, moderate or severe Alzheimer) with the help of clinicians and we proposed a medical protocol that was deployed in November 2020 and which is underway. Over a total period of ten months, two groups of people are playing these games: a control group with no known cognition deficit and a patient group with an identified medium cognition deficit. All game data (scores, answers, and response times) as well as video recordings (focused only on the hands of the participants) will be recorded and anonymized. At the end of the experimentation, the final results will be used to adjust the models in order to obtain a better representation of the behavior variants and to make our models effective. Our ultimate goal is to integrate the model checking approach proposed in this paper into a medical monitoring system.

**Acknowledgements** We thank the French Provence-Alpes-Côte d'Azur region for financing Thibaud L'Yvonnet Ph.D. We also thank our medical partners from the CoBTeK laboratory of Université Côte d'Azur for our fruitful interactions, and Tim QuatMann of the Storm team for his help.

# References

[1] Michalis Vrigkas, Christophoros Nikou, and Ioannis A. Kakadiaris. A Review of Human Activity Recognition Methods. *Frontiers in Robotics and AI*, 2015.

[2] S. Weerachai and M. Mizukawa. Human behavior recognition via top-view vision for intelligent space. In *Int. Conf. on Control, Automation and Systems (ICCAS)*, pages 1687–1690, 2010.

[3] Ujjwal Ujjwal, Aziz Dziri, Bertrand Leroy, and Francois Bremond. Late Fusion of Multiple Convolutional Layers for Pedestrian Detection. In *15th IEEE Int. Conf. on Advanced Video and Signal-based Surveillance*, pages 1–6, 2018.

[4] Hong-Bo Zhang, Yi-Xiang Zhang, Bineng Zhong, Qing Lei, Lijie Yang, Ji-Xiang Du, and Duan-Sheng Chen. A comprehensive survey of vision-based human action recognition methods. *Sensors*, 19, 2019.

[5] Minh Khue Phan Tran, François Brémond, and Philippe Robert. Assistance for Older Adults in Serious Game Using an Interactive System. In *4th Int. Conf. on Games and Learning Alliance (GALA)*, pages 286–291, 2015.

[6] Alexandra König, Carlos Fernando Crispim Junior, Alexandre Derreumaux, Gregory Bensadoun, Pierre-David Petit, François Bremond, Renaud David, Frans Verhey, Pauline Aalten, and Philippe Robert. Validation of an automatic video monitoring system for the detection of instrumental activities of daily living in dementia patients. *Journal of Alzheimer's disease : JAD*, 44, 2015.

[7] Elisabetta De Maria, Thibaud L'Yvonnet, Sabine Moisan, and Jean-Paul Rigault. Probabilistic Activity Recognition for Serious Games with Applications in Medicine. In Osman Hasan and Frédéric Mallet, editors, *Formal Techniques for Safety-Critical Systems*, pages 106–124, Cham, 2020. Springer International Publishing.

[8] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of Probabilistic Real-time Systems. In *Proc. 23rd Int. Conf. on Computer Aided Verification (CAV'11)*, pages 585–591, 2011.

[9] Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk. A storm is coming: A modern probabilistic model checker. In *International Conference on Computer Aided Verification*, pages 592–600. Springer, 2017.

[10] Gerd Behrmann, Alexandre David, and Kim G. Larsen. A Tutorial on Uppaal. In *Formal Methods for the Design of Real-Time Systems: 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM-RT 2004*, pages 200–236, 2004.

[11] Jun Sun, Yang Liu, Jin Song Dong, and Jun Pang. PAT: Towards Flexible Verification under Fairness. In *International Conference on Computer Aided Verification*, pages 709–714, 2009.

[12] Dorsa Sadigh, Katherine Driggs-Campbell, Alberto Puggelli, Wenchao Li, Victor Shia, Ruzena Bajcsy, Alberto L. Sangiovanni-Vincentelli, S. Shankar Sastry, and Sanjit A. Seshia. Data-Driven Probabilistic Modeling and Verification of Human Driver Behavior. In *Formal Verification and Modeling in Human-Machine Systems, AAAI Spring Symposium (FVHMS)*, pages 1–6, 2014.

[13] Ernst Moritz Hahn, Arnd Hartmanns, Christian Hensel, Michaela Klauck, Joachim Klein, Jan Křetínskỳ, David Parker, Tim Quatmann, Enno Ruijters, and Marcel Steinmetz. The 2019 comparison of tools for the analysis

of quantitative formal models. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 69–92. Springer, 2019.

[14] Marta Kwiatkowska, Gethin Norman, and David Parker. Stochastic model checking. In *Int. School on Formal Methods for the Design of Computer, Communication and Software Systems*, pages 220–270, 2007.

[15] Fabio Buttussi, Tommaso Pellis, Alberto Cabas-Vidani, Daniele Pausler, Elio Carchietti, and Luca Chittaro. Evaluation of a 3D serious game for advanced life support retraining. *Int. Journal of Medical Informatics*, 2013.

[16] S. D. Atkinson and V. L. Narasimhan. Design of an introductory medical gaming environment for diagnosis and management of Parkinson's disease. In *Trends in Information Sciences Computing (TISC)*, pages 94–102, 2010.

[17] Theresa M. Fleming, Lynda Bavin, Karolina Stasiak, Eve Hermansson-Webb, Sally N. Merry, Colleen Cheek, Mathijs Lucassen, Ho Ming Lau, Britta Pollmuller, and Sarah Hetrick. Serious Games and Gamification for Mental Health: Current Status and Promising Directions. *Frontiers in Psychiatry*, 2017.

[18] Silvia Bonfanti, Angelo Gargantini, and Atif Mashkoor. A systematic literature review of the use of formal methods in medical software systems. *Journal of Software: Evolution and Process*, 30(5), 2018.

[19] P. Groot, A. Hommersom, P. Lucas, R.-J. Merk, A. ten Teije, F. van Harmelen, and R. Serban. Using model checking for critiquing based on clinical guidelines. *AI in Medicine*, 46(1):19–36, 2008.

[20] Tommaso Magherini, Guido Parente, Christopher Nugent, Mark Donnelly, Enrico Vicario, Frederico Cruciani, and Cristiano Paggetti. Temporal logic bounded model-checking for recognition of activities of daily living. In *Proc. 10th IEEE Int. Conf. on Information Technology & Applications in Biomedicine*, pages 1–4, 2010.

[21] M. Hassan. A performance model of pedestrian dead reckoning with activity-based location updates. In *2012 18th IEEE Int. Conf. on Networks (ICON)*, pages 64–69, 2012.

[22] A. S. Ahouandjinou, C. Motamed, and E. C. Ezin. A temporal belief-based hidden Markov model for human action recognition in medical videos. *Pattern Recognition and Image Analysis*, 2015.

[23] Ahmad Jalal, Shaharyar Kamal, and Daijin Kim. A Depth Video-based Human Detection and Activity Recognition using Multi-features and Embedded Hidden Markov Models for Health Care Monitoring Systems. *Int. Journal of Interactive Multimedia & Artificial Intelligence*, 2017.

[24] Tâches Attentionnelles Exécutives. `https://cmrr-nice.fr/lab/tae/`, 2018. CoBTeK lab, Université Côte d'Azur.

[25] James T Eckner, James K Richardson, Hogene Kim, David B Lipps, and James A Ashton-Miller. A novel clinical test of recognition reaction time in healthy adults. *Psychological assessment*, 24(1):249, 2012.

[26] Eunju Kim, Sumi Helal, and Diane Cook. Human activity recognition and pattern discovery. *IEEE pervasive computing*, 2009.