A Probabilistic Analysis of Simplified Cluedo with Storm: The Birthday Cake Case*

Ezio Bartocci[1], Josée Desharnais[2], Peter Lindner[3], and Ana Sokolova[3]

[1] Technische Universität Wien, Austria
ezio.bartocci@tuwien.ac.at
[2] Université Laval, Canada
josee.desharnais@ift.ulaval.ca
[3] Paris Lodron University of Salzburg, Austria
peter.lindner@plus.ac.at,ana.sokolova@cs.uni-salzburg.at

Abstract. We present a family of probabilistic models of a simplified version of the Cluedo game. In this version of the game, instead of a murder happening, a birthday cake has mysteriously disappeared. The aim of the game is to guess, from the clues that each player will collect while playing, what happened to the cake. The winner is the player that first guesses who has eaten the cake and the room where this has happened. We implemented several probabilistic models of the game encoding different playing strategies as Markov Decision Processes in the Prism language. We investigate these strategies by comparing their effectiveness in winning the game using the Prism and Storm probabilistic model checkers. In particular, we use Prism for statistical results and Storm for exact computation. Since the generated state space is in general huge, we limit our models to only two players resulting in almost 15 billion states to check. We believe that this benchmark could serve to further improve the current state-of-the-art probabilistic model checking.

Keywords: Cluedo, Markov Decision Processes, Probabilistic Model Checking, Stochastic Games

1 Introduction

Prologue. *It is a stormy night in the Netherlands. Joost-Pieter is celebrating his birthday and has invited some of his dearest friends to arrive the night before the event. His friends brought a cake, a famous Limburg vlaai. In the morning, they all realize that a mysterious event happened: The cake has been eaten! After a short brainstorming, nobody recalls or admits eating the cake. They all decide on playing a game in order to find out who ate the cake and where it happened.*

The game is deductive and probabilistic. It is closely connected to the traditional board game Cluedo, originally designed by Anthony E. Pratt in United

* This work was partially supported by the Vienna Science and Technology Fund (WWTF) [10.47379/ICT19018] (ProbInG).

Kingdom in 1943, see e.g., [23]. Cluedo is known as Clue in North America. One difference between our version and the original game is that nothing terrible happens in our situation – we are dealing only with an eaten cake, not with a serious crime. Remarkably, after we decided on this change we realised that there is a version of Cluedo for children [14] that goes along these lines. We decided to model the game and analyse it with some of the most prominent model checking and simulation tools, PRISM [18] and STORM [15], to a great extent courtesy of Joost-Pieter Katoen's impressive work. We use the statistical model checking tool of PRISM for obtaining quick estimates, and the STORM model checker to check the properties on the obtained MDPs.

The game provides an interesting family of models, based on a "board", i.e., an imaginary plan of Joost-Pieter's house. It also shows the limits of what can be achieved, as the state space grows very rapidly. We simplify the board layout, and the game in some other respects, and create a family of models [4] of different complexity based on the "intelligence" of the players. The house layout in our model has four rooms and we have modelled four characters, thus simplifying the original game that has nine rooms and six characters. Initially, our model involved three players and one passive character. Due to the state space explosion, we were forced to limit ourselves to two players. The number of characters does not influence the size of the model and remains four in this case. We have created multiple versions of the model that are parametric in the level a player plays. We identify four levels starting from the simplest one, where a player just randomly guesses and uses no accumulated knowledge at all, to a level where a player uses the knowledge of players having a certain card (positive knowledge) and players not having a certain card (negative knowledge). In this first step, we have not modelled any sophisticated deduction mechanisms. We build the model using the PRISM modelling language. Despite the game being fairly simple, the model is huge and there were many interesting choices to make during the modelling process.

We believe that with this modest birthday present for Joost-Pieter, we have also contributed the following to the community of probabilistic analysis:

- We have created a new interesting case study for probabilistic model checking that involves probability, nondeterminism, different communication means like broadcast and handshaking, and can be scaled–by changing the size and layout of the board, by varying the number of characters and the number of players, etc. This work thus further reduces the lack of benchmarks for probabilistic analysis including both probability and nondeterminism, a lack that we have noticed in previous work [3], together with Joost-Pieter Katoen, and which has improved since then [12,5]. Our models are freely available [4] and will be added to the QComp and PRISM benchmark sets.
- We explore the limits of probabilistic model checking, and report the huge numbers of states obtained even for the simplified games. We then simplify even further in order to get a model that can be model-checked.

- We verify the intuitively made conjectures that the game is fair, i.e., the probabilities to win are more or less the same for all players, with slight advantage of the player whose turn is first.
- We investigate the expected number of turns needed to win the game. Moreover, we investigate "stupid" vs. "more intelligent" strategies of playing and show that players who play at a more advanced level have higher probability of winning and lower expected number of turns until they win, against players who play at a simpler level. Even a simple setup with no sophisticated deduction mechanisms makes a difference–in the original game (when played by humans) as well as in the simplified one that we model. We do not explore advanced epistemic deduction nor answer set programming approaches, although both could be done for more in-depth analysis.

The rest of the paper reports on: Related work in Section 2; The original game Cluedo and the rules, in Section 3; The representation of knowledge and the different strategies in Section 4; Our simplified version of the game, in Section 5; The PRISM model and the logic behind it, in Section 6; The PRISM and STORM results on the properties checked, in Section 7. We wrap up with conclusions and lessons learned in Section 8.

2 Related Work

In a *knowledge* game [21] such as Cluedo, winning requires from each player to gain knowledge about the ownership of the cards. A player can enrich their knowledge by moving to different rooms, by asking the right questions, and by reasoning about the answers of the other players. As such, this game is a natural case study in the field of epistemic reasoning [21,22,9,8,16]. In [21,22], van Ditmarsch was the first to provide a precise formalisation of the player's actions using dynamic epistemic logic for multi-agents with shared knowledge [19]. The use of epistemic logic allows to model the local knowledge of each player and to reason about the consequences of each player's action. Later, Dixon formalised in [9] the change of knowledge over time using temporal epistemic logic. Although these works provide a powerful framework to reason about the game dynamics, they do not consider the uncertainty due to rolling of the dice and they do not address the problem of assessing the effectiveness of different playing strategies.

Cluedo is also a suitable game environment to test artificial intelligent approaches [6,24,17,20] for learning and testing optimal strategies. Examples include expert systems [17], reinforcement learning [6], and entropy-reduction based strategy [24]. In [17], the author developed an *expert system* that can play Cluedo with five different strategies. These were tested on specific "deals" each one representing the "combination of 'guilty' cards". In these tests, all the players adopted the same strategy. Their strategies are partly similar to ours (the simplest - using no knowledge at all - being the same). However, in our work we also consider games where different players are using different strategies.

In [13], the authors analysed the trade-off between costs and benefits for a player adopting a "bluffing" strategy that consists in leading the other players to

infer the wrong knowledge from their actions. Their quantitative analysis is based on gathering statistical information over 20000 simulated games with random initial conditions. In contrast with the methods using simulation and testing, our approach based on exact model checking, is exhaustive. The game model we provide in this paper is the first PRISM model of a (simplified) variant of Cluedo. If we consider the state space, our model is the second Markov Decision Process in terms of size with respect to the ones presented in the quantitative verification competitions[1] [12,5] in the last years.

Last but not least, recent work focused on probabilistic analysis of other games using formal methods techniques. Such is, in particular, the mCRL2-based probabilistic analysis of the Game of the Goose [11] and Slot Machines [10]. We leave the investigation of how mCRL2 compares to Prism and Storm in the context of Cluedo and related games for future work.

3 The Original Game Cluedo

Our original motivation was to provide a Prism/Storm model for the game Cluedo [23], a popular logic-based, probabilistic board game. As expected, this turned out to be a very ambitious and computationally challenging task, so we simplified the game. Still, for the sake of completeness and reference to the original game we provide a brief, yet almost complete, description of the Cluedo game and rules here. Later in the paper, we describe our simplified version in more detail.

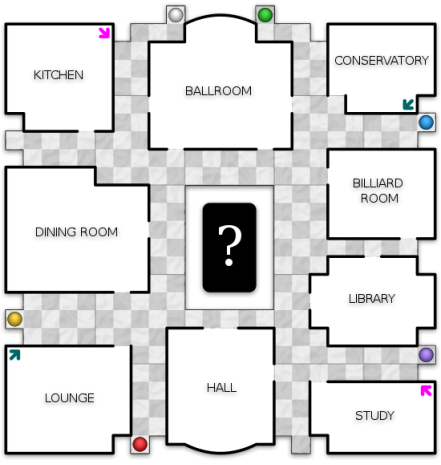


Fig. 1: The traditional board arrangement. Colored dots denote the initial positions of the characters. "The arrows in the four corner rooms denote secret passages to the opposite corner". Source: [23].

[1] `https://qcomp.org`

3.1 Brief Description

Cluedo is a board game that involves:

- A **board** with rooms and intermediate positions, that we sometimes refer to as fields, described by a grid, illustrated in Fig. 1.
- Six **character** pieces, each representing a possible suspect. that moves on the board. At any point in time a character is positioned on one of the fields of the board.
- **Weapon** pieces (because in the original game, the players also have to deduce what the murder weapon is).
- A set of **cards** consisting of: one card for each room, one card for each character, one card for each weapon.
- Up to six **players**, each playing a particular character. Thus some characters are not "played", but they are nonetheless potential suspects.
- A pair of **dice**, which are rolled by the active player at each turn in order to determine the next set of possible positions.

The rules for Cluedo are described next.

Initialisation

Prior to the start of the game, one room card, one character card, and one weapon card are selected by a random draw and stored in a safe place. These cards represent the committed **crime**: in which room it happened, who was the culprit, and which weapon was used. All other cards are evenly distributed among the players. As for character figures, they are positioned in their clearly marked positions on the board, according to their color. Weapons do not have a specific position.

Game Progress

The game proceeds in **rounds**. In each round, one of the players, the *active* player, has their[2] **turn**. The players take turns in a standard round-robin way. One round consists of the following steps:

1. The active player rolls the dice and chooses nondeterministically among a set of possible positions, that would bring the corresponding character in or closer to a room. For example, if the dice roll results in the number 10, the player can move their character in any direction for up to 10 steps, where moving to a neighbouring field on the board counts as one step. This is the standard way of moving on a grid in board games. If a room is reachable in, e.g., 7 steps, then the player can decide to move to this room and stay there. However, if a player starts a turn inside a room, they have to leave the room within this turn. We omit the rules pertaining to secret passages as they are not modelled in our version of the game.

[2] We use the gender-neutral personal pronoun form "they"/"their" throughout the paper in both singular and plural.

2. If no room was reached, the turn ends here.
3. If a room was reached, the player expresses a — potentially announced as final — **suspicion** which is the room the active player is in, a guess of a character, and a guess of a weapon. This suspicion is broadcasted, i.e., it is communicated to all players. The guessed character is moved to the room if it is not already there. If the suspicion is final, the active player is shown the hidden cards. If they match the suspicion made, this player wins and the game ends. If not, the game ends for this player only, the player loses the game. Final suspicions are not modelled in our version of the game since a simpler way of ending the game is possible.
4. Players respond to a stated non-final suspicion as follows:
   - If a player has one or more of the mentioned cards, they show one of these cards, in a secret (handshaking) way, to the active player.
   - If a player has none of the cards, they simply pass.
5. After the responses, if the game has not ended, everybody updates their knowledge according to the obtained information and the next player, in a round-robin fashion, becomes active and starts their turn.

4 Knowledge Representation, Updates, and Strategies

In this section we present our choices for representing and using the gathered knowledge and we identify and describe the different strategies, in form of different levels, for the players.

The dealing of the cards, which is the essence of the global state, is given by a function $\mathtt{c}$ that attributes cards to either a player or to the hidden pile (representing the crime):

$$\mathtt{c}\colon \text{Cards} \to \text{Players} \cup \{\mathtt{hidden}\},$$

with

$$\mathtt{c}(C) = \begin{cases} p & \text{if player } p \text{ has card } C, \\ \mathtt{hidden} & \text{if card } C \text{ is part of the crime.} \end{cases}$$

Note that the use of a capital letter for C is to contrast with the lower case c next to it and will be even more justified by further definitions.

The knowledge of players consists of an approximation of this function. We distinguish between positive and negative knowledge.

The **positive knowledge** of player p is a function

$$\text{knows}(p)\colon \text{Cards} \to \text{Players} \cup \{-1\}$$

with

$$\text{knows}(p)(C) = \begin{cases} p' & \text{if player } p \text{ knows that } p' \text{ has the card } C, \\ -1 & \text{otherwise.} \end{cases}$$

This local knowledge of player p is updated to a value different than -1 or p, when p is active and when, after having formulated a suspicion, they receive a card C from another player p'. Of course, if a player p has a card C, we have $\text{knows}(p)(C) = p$. Because we consider different levels of strategy, this update will not always be used by the players (in particular it is not used for levels below 2).

The *negative* knowledge is enriched when a player p' declares a "pass" on a guess from player p. This information is the same for all players, as it is broadcasted, and hence, we make it a global function instead of parametrizing it per player. The **global negative knowledge** is a function

$$\text{neg_know}\colon \text{Cards} \times \text{Players} \to \{\texttt{true}, \texttt{false}\}$$

with

$$\text{neg_know}(C, p) = \begin{cases} \texttt{true} & \text{if everybody knows } C \text{ is not in } p\text{'s hand,} \\ \texttt{false} & \text{otherwise.} \end{cases}$$

Clearly, both types of knowledge agree with the true dealing of cards:

$$\begin{aligned} \text{knows}(p)(C) = p' &\Rightarrow \texttt{c}(C) = p', \\ \text{neg_know}(C, p) = \texttt{true} &\Rightarrow \texttt{c}(C) \neq p. \end{aligned}$$

Note that, since the negative knowledge is global, we cannot infer negative knowledge on other players from the fact that $\text{knows}(p)(C) = p'$.

There is more knowledge and deduction involved in playing the original Cluedo game. One could use more epistemic deduction and/or answer set programming methods to design a strategy, see e.g., [21,22,9,8]. The primary interest for us is to produce a probabilistic nondeterministic model, which could be model checked by existing tools. We therefore keep it simple and assume only this limited positive and negative knowledge.

We next identify several meaningful and clearly simplified **strategies** for a player. These are based on the notion of *a card being considered decided for a player*. Such cards will be ignored at the time of guessing. The idea is that a player will only suspect a room and/or character if the corresponding card is *undecided*, i.e., it is of interest to the player. We identify four different levels, described next. The difference between them is in the definition of a decided card.

- Level 0 – no knowledge used: A player just makes a random guess among the available options without using any knowledge, so no card is considered decided.
- Level 1 – own positive knowledge: player p makes a guess only on cards that they do not own, i.e., such that $\text{knows}(p)(C) \neq p$. Hence, a card is considered decided if the active player owns it.

- Level 2 – all positive knowledge used: player p makes a guess only on cards such that $\text{knows}(p)(C) = -1$. Hence, a card is considered decided for the active player if they know who owns this card.
- Level 3 – positive and negative knowledge used: Here, a card C is considered decided and hence will be ignored by player p if p knows who has it or p knows that no (other) player has it. Formally,

$$\text{knows}(p)(C) \neq -1 \vee \forall p' \neq p.\, \text{neg_know}(C, p') = \texttt{true}.$$

However, there are situations where players have to make a guess even if cards are decided, in order to prevent the game from deadlocking. This is, for example, the case when all character cards are decided, but the game has not ended yet. In such a case, we suspect the room where we are and randomly choose a character card, with probability $\frac{1}{|\text{Characters}|}$, and proceed. Note that this might seem not to be the best strategy for the active player, it might seem as if it would have been better to choose one of the character cards this player owns, if there are such, but it is necessary to prevent deadlock as we nowhere store the (local) knowledge of the hidden card and only make the winning guess more likely as the game proceeds. Hence, this is needed for the current version of the model.

Note that in a two-player game, playing at level 3 is in a sense (that we do not make precise here) equivalent to playing at level 2, as our experiments in Section 7 confirm.

These strategies could be seen as memoryless and a player could mix them nondeterministically at each turn. However, we do not expect that mixing would show meaningful results, and therefore investigate the behaviour of *consistent* players only, that is, players that stick to a level throughout the game. We reckon that there are many other possible strategies we could have considered, maybe some clever ones, but these offer simplicity as well as a satisfactory diversity.

We note that in our models the choice of strategy is hardcoded in the formula defining the predicate when a card is considered decided, except the code for level 0 which does not mention decided cards. The rest of the model is unaffected by this choice.

5 Simple Birthday Cluedo

We now briefly discuss the exact layout of the board as well as the choices made in our simple birthday version of Cluedo.

Our board consists of eleven positions, shown in Fig. 2. The corner positions of the 3×3 square represent the four rooms: the kitchen at position 0, the living room at position 2, the library at position 6, and the dining room at position 8. Hence, the rooms are near to each other and near to the initial positions of the players, at most a couple of steps away. Moreover, one can imagine the rooms having doors on both sides, e.g., the kitchen is reachable both from position 1 and from position 3 in one step. The additional positions 9 and 10 that stick out of the square are the initial positions of the two players. We originally planned four

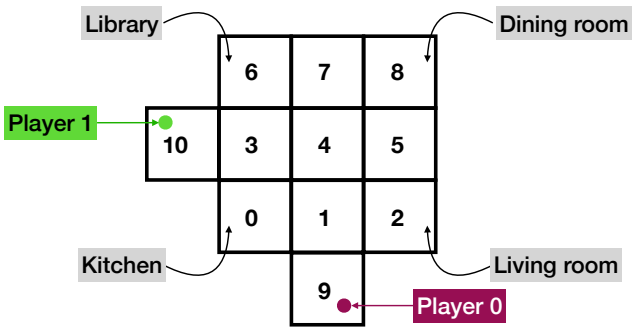


Fig. 2: The board of our simple birthday Cluedo game.

such position, in order to accommodate up to four players, but as mentioned we limit ourselves to two only. These positions are never visited later in the game, as a player needs to be in a room in order to make a guess and these positions are bringing players further away from the rooms.

There are no weapons (and hence no weapons cards) in the simplified birthday Cluedo. Clearly, this reduces the state space.

Moreover, we do not model the secret passages, as on the small board they would make all rooms easily accessible to the players.

While designing the model, we have noticed that it is of no importance at all where the inactive characters are positioned and hence we do not store and update their positions.

As already mentioned, originally we created a model with three players and one inactive character. The model was never fully built by Storm as we ran out of the approximately 8 TB memory that we had available at a point where the model had 4600 billion states. We therefore reduced the model further to two players and two inactive characters. However, we leave the logic of the game and the representation and use of knowledge as in the case of multiple players, that is, in the deduction we do not take advantage of the fact that there are (only) two players.

Finally, since our board is quite small, we only roll a single die in order to determine where to move next.

6 The Prism Model

In order to use the Storm model checker, we construct the model [4] using the Prism language. Reasoning about states is always through values of variables, transitions can be probabilistic, e.g., the dice roll example shown in Listing 1.1, or nondeterministic, if more transitions apply in a state. The language is very simple in its functionality, at times we wished for more expressive features, yet the simplicity also has its merits and the language is very clear and easy to use.

Transitions are guarded by Boolean-formula guards that may test all variables in the system. For more information on the PRISM language, please consult [1,18].

```
// Module for a simple coin flip
module coin
    // Face of the coin, 0 is tail and 1 is head
    face : [0..1] init 0;

    // Transition probabilities
    [flip] true -> 1/2 : (face'=0) + 1/2 : (face'=1);
endmodule
```

Listing 1.1: Prism Module of the Coin Example

Logically, on a high level, the model is a parallel composition of several processes:

- A *master* process that takes care of the initialization, the distribution of the cards, and gives the turn to the players in a round-robin fashion.
- The *board* with its movement restrictions and probabilistic information.
- A number of (here only two) *player processes*, one per player. These are different instances of the same PRISM module.

The Dealing

The first step is the assignment of all the cards, that is, who has which card and which cards are hidden as the solution. This information is given by the function $\mathtt{c}$ and is stored using a variable $\mathtt{c}C$ for each card C. These card variables take values from -1 to n, where n is the number of players, as follows (and assigned in this order):

$$\mathtt{c}C = \begin{cases} -1 & \text{if card } C \text{ has not been assigned to a player yet (initial value),} \\ n & \text{if card } C \text{ is hidden (one of the two cards of the solution),} \\ i & \text{if card } C \text{ is in the hand of player } i \in \{0, 1, \ldots, n-1\}. \end{cases}$$

These variables are written as, e.g., `cKitchen` for the kitchen card, and `cChar3` for the card representing character 3.

The Rounds

In each round of the game we have an active player, which we often consider to be player 0 in the sequel. The round consists of the following steps, to be executed sequentially:

1. Dice roll and choice within the set of available rooms;
2. Stating a suspicion;
3. Collecting responses;
4. Updating;

5. End of game or re-initialization for the next round.

In order to ensure sequentiality, and distinguish between the different phases (as well as eliminate needless states), we introduce global Boolean flag variables. These are: `dice`, `guess`, `response`, `update0`, `update1` as well as some others that guard the corresponding sequential phases in the behaviour of the active player. For example, `dice` is only true in the first phase (dice roll) when it is time to roll the dice and move across the board accordingly, as described next. We will discuss why we need multiple variables in the, e.g., update phase when we detail the corresponding phase below.

In the rules expressed in the sequel we skip writing unnecessary guards. For example, all the mentioned phases happen after the initialisation phase, and hence include a guard that checks that initialisation has been completed, but we do not write that in the rules to keep them a bit shorter and easier to read.

The Suspicion

We use global variables, `roomSusp` and `charSusp`, to express the two suspicions. They take a value between 0 and 7, following the encoding in Table 1. The same type is used for variables `show0` and `show1` which record the card that a player shows upon a guess from another player.

Room card	Encoding room card	Character card	Encoding character card
Kitchen	0	Character 0	4
Living room	1	Character 1	5
Library	2	Character 2	6
Dining room	3	Character 3	7

Table 1: Encoding of the cards

The Knowledge

For representing the knowledge functions, we also use global and local variables. For the local positive knowledge of player $p \in \{0, 1, \ldots, n-1\}$ we use variables of the form

$$\texttt{c}Cp = \begin{cases} i & \text{if } p \text{ knows that card } C \text{ is owned by player } i \in \{0, 1, \ldots, n-1\}, \\ -1 & \text{otherwise (initial value).} \end{cases}$$

For example, $\texttt{cKitchen0} = \texttt{1}$ denotes that player 0 knows that the kitchen card is owned by player 1; This is close to the notation of Section 4, where it is knows(0)(`cKitchen`) = `1`.

The negative knowledge is a global information as in Section 4. Which card is *not* owned by which player is stored in a set of Boolean variables of the shape `nc`C`Player`p where C is a card, and p a player. For example,

$$\texttt{ncKitchenPlayer0} = \texttt{true}$$

represents the common knowledge that player 0 does not have the kitchen card.

Dice Roll and Move to the Next Position

Given that the active player is at a position of the board, we precompute the probabilities of moving to a set of new positions, see Table 2. This is the main source of probabilistic information in the model.

$0 \to \frac{1}{6}\colon (1 \vee 3) + \frac{1}{6}\colon (2 \vee 4 \vee 6) + \frac{1}{6}\colon (2 \vee 5 \vee 6 \vee 7) + \frac{1}{2}\colon (2 \vee 6 \vee 8)$
$1 \to \frac{1}{6}\colon (0 \vee 2 \vee 4) + \frac{1}{6}\colon (0 \vee 2 \vee 3 \vee 5 \vee 7) + \frac{2}{3}\colon (0 \vee 2 \vee 6 \vee 8)$
$2 \to \frac{1}{6}\colon (1 \vee 5) + \frac{1}{6}\colon (0 \vee 4 \vee 8) + \frac{1}{6}\colon (0 \vee 3 \vee 7 \vee 8) + \frac{1}{2}\colon (0 \vee 6 \vee 8)$
$3 \to \frac{1}{6}\colon (0 \vee 4 \vee 6) + \frac{1}{6}\colon (0 \vee 1 \vee 5 \vee 6 \vee 7) + \frac{2}{3}\colon (0 \vee 2 \vee 6 \vee 8)$
$4 \to \frac{1}{6}\colon (1 \vee 3 \vee 5 \vee 7) + \frac{5}{6}\colon (0 \vee 2 \vee 6 \vee 8)$
$5 \to \frac{1}{6}\colon (2 \vee 4 \vee 8) + \frac{1}{6}\colon (1 \vee 2 \vee 3 \vee 7 \vee 8) + \frac{2}{3}\colon (0 \vee 2 \vee 6 \vee 8)$
$6 \to \frac{1}{6}\colon (3 \vee 7) + \frac{1}{6}\colon (0 \vee 4 \vee 8) + \frac{1}{6}\colon (0 \vee 1 \vee 5 \vee 8) + \frac{1}{2}\colon (0 \vee 2 \vee 8)$
$7 \to \frac{1}{6}\colon (4 \vee 6 \vee 8) + \frac{1}{6}\colon (1 \vee 3 \vee 5 \vee 6 \vee 8) + \frac{2}{3}\colon (0 \vee 2 \vee 6 \vee 8)$
$8 \to \frac{1}{6}\colon (5 \vee 7) + \frac{1}{6}\colon (2 \vee 4 \vee 6) + \frac{1}{6}\colon (1 \vee 2 \vee 3 \vee 6) + \frac{1}{2}\colon (0 \vee 2 \vee 6)$
$9 \to \frac{1}{6}\colon (1) + \frac{1}{6}\colon (0 \vee 2 \vee 4) + \frac{1}{6}\colon (0 \vee 2 \vee 3 \vee 5 \vee 7) + \frac{1}{2}\colon (0 \vee 2 \vee 6 \vee 8)$
$10 \to \frac{1}{6}\colon (3) + \frac{1}{6}\colon (0 \vee 4 \vee 6) + \frac{1}{6}\colon (0 \vee 1 \vee 5 \vee 6 \vee 7) + \frac{1}{2}\colon (0 \vee 2 \vee 6 \vee 8)$

Table 2: Probabilistic moves from any position to a set of reachable positions after rolling the dice.

For example, the line for position 0 states that with probability $\frac{1}{6}$, which is the probability of outcome 1 after rolling the dice, the player can move to position 1 or 3, as these are one step away from the position 0. In two steps, with probability $\frac{1}{6}$ the player can reach positions 2, 4, or 6. In three steps, again with probability $\frac{1}{6}$ it could reach 2, 5, 6, or 7. Here, 2 is still there, as the player may decide to go to this room (living room) next, no matter that it could go further away. Moreover positions 5 and 7 are also legal, as they bring the player closer to a room that is not reachable yet. Finally, with the remaining probability of $\frac{1}{2}$ corresponding to the dice outcome being 4, 5, or 6, the player can reach all other rooms and has therefore the choice between positions 2, 6, and 8.

Clearly, the data from the table can be pre-computed in an automated way for larger boards. Note that just like in the original Cluedo game, we do not allow the active player to remain at their old position.

We originally modelled the move to a room as a nondeterministic choice, indicated in the table by using the disjunction symbol. However, this gave the model-checker the possibility to use information unavailable to players, as we will discuss in the next section. Moreover, the model was huge and impossible to check, see Section 7. Hence, we convinced ourselves that it makes sense to choose randomly within the rooms of interest after ignoring decided ones (according to the level played) – and this is exactly what we model now. In our current models,

we now have a probabilistic choice, influenced by the level of the active player, i.e., by cards being decided.

For example, if the active player 0 is at position 0 and the cards for library and dining room are decided, then the player can move to the living room (encoded by 1) and suspect it or to the other available positions and end their turn without making a suspicion, according to the following probabilistic transition rule:

```
dice & (turn = 0) & (pos0 = 0)
     & (!cKitchenDecided0)
     & (!cLivingRoomDecided0)
     & (cLibraryDecided0)
     & (cDiningRoomDecided0) ->
           1/12:  (pos0' = 1) & (dice' = false)
                   & (endTurn' = true)                  +
           23/36: (pos0' = 2) & (dice' = false)
                   & (guess' = true) & (roomSusp' = 1) +
           1/12:  (pos0' = 3) & (dice' = false)
                   & (endTurn' = true)                  +
           1/12:  (pos0' = 4) & (dice' = false)
                   & (endTurn' = true)                  +
           1/18:  (pos0' = 5) & (dice' = false)
                   & (endTurn' = true)                  +
           1/18:  (pos0' = 7) & (dice' = false)
                   & (endTurn' = true).
```

The probabilities are calculated by replacing the nondeterministic disjunction with a uniform probability distribution in each parenthesis, after ignoring the rooms of the decided cards.

It suffices to have 8 rules (testing whether the dining room, the living room, and the library cards are decided) of this kind for position 0. Note that we need not test the kitchen card at position 0 as the player is in the kitchen and has to change their position. At positions that are not rooms, we need to have 16 rules per line in the table, testing which room cards are decided. We precompute the rules using a Python script. This is one of the main sources of complexity in the model. However, our current Python script actually creates 16 rules for each position, hence in the model we are also (unnecessarily) testing for the kitchen card in the above situation.

The Guess

As we saw in the rule above, changing the position upon throwing a dice immediately determines the guess for the "suspected room". It is now time to make the guess for the "suspected character" in the current turn.

This is again a point where the level information is used in order to express a guess. However, thanks to the notion of a decided card, the rules are the same in all levels. We have 16 rules, each testing which of the 4 character cards have

been decided and normalizing the probabilities accordingly. If none of the cards is decided, then the rule (for player 0 being the active player) is:

```
guess & (turn = 0)
      & (!cChar0Decided0)
      & (!cChar1Decided0)
      & (!cChar2Decided0)
      & (!cChar3Decided0) ->
            1/4: (charSusp' = 4) & (guess' = false)
                  & (response' = true) & (update0' = true) +
            1/4: (charSusp' = 5) & (guess' = false)
                  & (response' = true) & (update0' = true) +
            1/4: (charSusp' = 6) & (guess' = false)
                  & (response' = true) & (update0' = true) +
            1/4: (charSusp' = 7) & (guess' = false)
                  & (response' = true) & (update0' = true).
```

The reason why we set the update-flag for player 0 to true will be discussed below in the update subsection.

Here, again, we decided for a uniform distribution instead of nondeterminism. This is not only to make the model smaller, but also to comply with reality. Nondeterminism is not the right modelling choice for this situation in the game, as it could lead the model to use global information that a single player does not have. If the choice was nondeterministic here, the maximum probability would be obtained by guessing the right character from the start, which is a piece of global information stored in the model that a single player does not have.

The rule when all cards are decided has exactly the same conclusion, due to our choice to then select randomly between any of the four cards, as mentioned above in the discussion of the levels.

Another example, where we do not have all possible options is expressed by the following rule:

```
guess & (turn = 0)
      & (cChar0Decided0)
      & (!cChar1Decided0)
      & (cChar2Decided0)
      & (!cChar3Decided0) ->
            1/2: (charSusp' = 5) & (guess' = false)
                  & (response' = true) & (update0' = true) +
            1/2: (charSusp' = 7) & (guess' = false)
                  & (response' = true) & (update0' = true).
```

The Response

The response of the players in the current round is recorded in the variables `show0, show1` which can contain one card—the number encoding the card, or

−1 for "pass", or for the case when the player is the active player who does not show anything.

When a player, say player 1, shows a card, we change the value of the equally numbered variable `update1` to true. This is sort of a way to count how many of the players are done with showing cards. Only when all are done, the update step can start. This is also the reason why the active player, player 0, sets `update0` to true immediately after making the guess. [3] The rules for the response are, for example, as follows:

```
response & (turn != 1) & (update1 = false)
         & (roomSusp = 2) & (charSusp = 7)
         & (cLibrary = 1) & (cChar3 = 1) ->
              (show1' = 7) & (update1' = true).
```

This rule refers to an action of player 1, who is not the active one - not their turn now, but who owns both of the suspected cards. Note that with the encoding, the suspect room is the library and the suspect character is character 3. In this case, player 1 decides nondeterministically to show the card character 3. The other (nondeterministic) option for this player is:

```
response & (turn != 1) & (update1 = false)
         & (roomSusp = 2) & (charSusp = 7)
         & (cLibrary = 1) & (cChar3 = 1) ->
              (show1' = 2) & (update1' = true).
```

In our view, the nondeterministic response is adequate in the model and it reflects the nature of the game, as there is little to use in the global state. Moreover, it makes the model amenable to the addition of more sophisticated strategies in the future.

If player 1 (who is not the active player) has only one of the two cards, then this is the card they have to show. The corresponding rules are then as follows:

```
response & (turn != 1) & (update1 = false)
         & (roomSusp = 2) & (charSusp = 7)
         & (cLibrary = 1) & (cChar3 != 1) ->
              (show1' = 2) & (update1' = true).


response & (turn != 1) & (update1 = false)
         & (roomSusp = 2) & (charSusp = 7)
         & (cLibrary != 1) & (cChar3 = 1) ->
              (show1' = 7) & (update1' = true).
```

[3] Note that having a show-flag and an update-flag per player is not necessary in the case of two players only, but we have designed the model to work for multiple players too, and only reduced to two due to the limits of computational resources.

as well as the last possibility, when the player has none of the cards and broadcasts "pass":

```
response & (turn != 1) & (update1 = false)
         & (roomSusp = 2) & (charSusp = 7)
         & (cLibrary != 1) & (cChar3 != 1) ->
              (show1' = -1) & (update1' = true).
```

We could have saved two of the four rules that refer to showing a card that a player has. We keep the presentation here faithful to the model, which cannot easily be changed due to the need to rerun experiments that take many hours. This only influences the number of lines of code in the model.

The Update

Given our setup, in particular the negative knowledge being global, most of the updates are made by the active player, player 0.

This player needs to update their local positive knowledge based on the cards shown to them and to update the global negative knowledge based on players not showing cards.

Due to the design of the language and the nondeterminism involved, we need a way to record that all this has been done. We achieve this with two more Boolean flags, `updateShow0` and `updateShow1`, one per player. Initially, their value is false, and it becomes true if the knowledge gained from `show0`, `show1` has been updated, respectively. An example rule is:

```
(turn = 0) & (update0 = true) & (update1 = true)
           & (updateShow1 = false) & (show1 = 0) ->
                (cKitchen' = 1) & (updateShow1' = true).
```

Here, `show1` $= 0$ means that the card shown by player 1 was the kitchen card, as 0 encodes this card. We have such a rule for each possible card (room and character). Furthermore, we need all of these rules also checking `show0` and `updateShow0`. Finally, we need checks for a "pass" that updates the global knowledge:

```
(turn = 0) & (update0 = true) & (update1 = true)
           & (updateShow1 = false) & (show1 = -1)
           & (roomSusp = 0) & (charSusp = 4) ->
                (ncKitchenPlayer1' = true)
                 & (ncChar0Player1' = true)
                 & (updateShow1' = true).
```

Such a rule is needed for each possible combination of room and character cards, as suspects, to update the global knowledge (for player 1 here, as player 1 responds with a "pass").

Now, it only remains to move the suspect into the suspected room. Note that we do not store the position of characters that are passive, and hence we only move the suspect character if it is a player. This is given by the rule (here we also assume the affected player is player 0):

```
(turn = 0) & (update0 = true) & (update1 = true)
           & (updateShow0 = true) & (updateShow1 = true)
           & (charSusp = 4) & (roomSusp = 0) ->
                (pos0' = 0) & (response' = false)
                 & (checkWinner' = true).
```

This completes the update of the current round. We need such a rule for every possible value of `roomSusp`. Note that `checkWinner` is another flag that takes us to the last phase of the round, described next.

The Final Step before a New Round, or Game Over

At this point, we may have reached the end of the game. This is the case if none of the players has shown anything, and the active player 0 has none of the cards they asked for. We encode the end of the game as value n (here 2) of the global variable `turn` which normally takes a value i with $0 \leq i \leq n-1$ if player i is supposed to take the next turn, respectively. One can see this as the master getting the last turn. At the same time we also declare the winner, by assigning the identity of the active player to the global variable `winner`. To achieve this, we have rules of the kind:

```
(turn = 0) & (checkWinner = true) & (show1 = -1)
           & (update0 = true) & (update1 = true)
           & (updateShow0 = true) & (updateShow1 = true)
           & (roomSusp = 0) & (charSusp = 4)
           & (cKitchen != 0) & (cChar0 != 0) ->
                (winner' = 0) & (turn' = n).
```

We have such a rule for every possible room and character suspect, and every player in place of the active player.

In order to proceed in case there is no winner yet, we add a set of rules like the following rule:

```
(turn = 0) & (checkWinner = true)
           & (update0 = true) & (update1 = true)
           & (updateShow0 = true) & (updateShow1 = true)
           & ((show1 != -1) | (cKitchen = 0) | (cChar0 = 0))
           & (roomSusp = 0) & (charSusp = 4) ->
                (checkWinner' = false) & (endTurn' = true),
```

where | denotes disjunction, as in the Prism language and `endTurn` is another Boolean flag used to signal that a new turn should start. One such rule per

suspect pair and non-active player (testing that the player passed) is added to the model.

This completes the turn. We can then give the turn to the next player by:

```
endTurn & (checkWinner = false) ->
              (turn' = (turn + 1) mod n)
               & (dice' = true) & (endTurn' = false)
               & (update0' = false) & (update1' = false)
               & (show0' = -1) & (show1' = -1)
               & (updateShow0' = false)
               & (updateShow1' = false)
               & (guess' = false) & (response' = false).
```

7 Verification with PRISM and STORM

For our experiments, we first used the statistical model checking feature of PRISM `4.8.1` to obtain some early results on our model. Afterwards, we performed precise model checking using a development build of STORM `1.8.2`. We present the results of the runs in the subsequent sections.

In a nutshell, our results confirm that the game is well-designed, i.e., the individual players have almost the same winning probabilities. Moreover, the more advanced the strategy the players used, the shorter the game.

7.1 Hardware and Technical Setup

We ran PRISM on an Intel-based consumer-grade machine running Ubuntu 22.04.3 LTS, equipped with an Intel i7-13700H processor featuring 20 cores at 3.7-5.0 GHz (24 MB L3 cache) with 32 GB of memory. The results are briefly discussed in the following subsection.

Afterwards, we performed precise model checking using a development build of STORM `1.8.2` on the Mach-2, which "is the largest shared-memory machine in Austria" [2]. The Mach-2 is operating on SuSE Linux Enterprise Server 12 with a total amount of 1728 processor cores and approximately 20 TB of global shared memory. These 1728 cores are allocated across 72 blades, each holding "two 12-core processors of type Intel Xeon E5-4650V3 operating at 2.1-2.8 GHz with 30 MB L3 cache" [2]. To execute STORM, we had access to 18 of these blades, totalling 432 cores, and approximately 8 TB memory. Due to the size of our model, we enabled the `TRACE` logs to gain more insight into running computations and set the precision of the solvers used by STORM to 0.01. This allowed us to obtain results within an acceptable timeframe, at the cost of slightly reduced precision.

Note that using Mach-2 was not an overkill: we previously tried (and failed) using the Intel-based consumer-grade machine mentioned above, as well as two local servers (bigiron8 and bigiron9) of which the larger (bigiron9) has 64 processors of type AMD Opteron 6376 at 2.3-3.2GHz (12 MB L3 cache), and 528354580 kiB, i.e., approximately 504 GiB of memory.

7.2 Statistical Model Checking using Prism

Table 3 shows the results of statistical model checking with Prism, which we used for obtaining quick results and estimates. We used the CI (confidence interval) simulation method, unknown width, 0.01 confidence, 50000 simulation samples, and maximal path length of 10000. Increasing the number of simulation samples resulted in only insignificant changes.

Recall that the higher the level, the better a player in this level is supposed to perform. Indeed, this is confirmed by our results. In particular, we see that if player 0 plays at level 3, their probability of winning against a dumb adversary (with level 0) is very high, around 0.86. If both players follow the dumb strategy and play at level 0, then the expected number of rounds for the game is the highest, at more than 21. Moreover, also as expected, the chances of winning for the player who starts the game, player 0 in our experiments, are slightly higher than for the other player independent of the configuration, i.e., their levels.

Levels		Properties		
Player0	Player1	P(Player0 wins)	P(Player1 wins)	R(rounds)
0	0	0.5126	0.4874	21.26188
0	1	0.27598	0.72402	11.64354
0	2	0.18804	0.81196	8.1258
0	3	0.19398	0.80602	8.09902
1	0	0.7668	0.2332	11.12688
1	1	0.54422	0.45578	7.90266
1	2	0.446	0.554	6.48912
1	3	0.44302	0.55698	6.48658
2	0	0.86494	0.13506	6.94006
2	1	0.6704	0.3296	5.89656
2	2	0.56972	0.43028	5.52136
2	3	0.56478	0.43522	5.54222
3	0	0.86226	0.13774	6.95394
3	1	0.67554	0.32446	5.92966
3	2	0.56586	0.43414	5.49614
3	3	0.56682	0.43318	5.51918

Table 3: Results of statistical model checking using Prism, with properties: P(Player0 wins) = `P=? [F(turn=2)&(winner=0)]`, P(Player1 wins) = `P=? [F(turn=2)&(winner=1)]`, R(rounds) = `R{"rounds"}=? [F(turn=2)]`.

We note that the Prism statistical engine resolves nondeterministic choices by first uniformly sampling one of the enabled nondeterministic transitions, and then choosing the next state according to the probability distribution of the chosen transition. Hence, it considers only one randomised scheduler, turning an MDP into an MC, in contrast to Storm that computes the min/max probability of a property over all possible schedulers.

7.3 Probabilistic Model Checking using Storm

Using Storm, we checked for the minimum expected number of rounds ($R_{\min}$) for the game to end. It would also be interesting to investigate the minimum and maximum probabilities of winning in the future, as the model still contains a small amount of nondeterminism. It is worth mentioning that we initially started with a model that had a high degree of nondeterminism. Specifically, the choice of a position on the board after the dice roll was nondeterministic, and in addition, we had three players. Even Mach-2, with approximately 8 TB of RAM, ran out of memory while building that model. The machine entered an endless loop of garbage collection by the Sylvan [7] library, which is used to represent the model symbolically. The number of states in the model was 4,628,706,795,793, hence 4600 billion, at the time Storm stopped making progress.

We also experimented with different Storm engines. In the beginning, we tried the Sparse engine on the initial large model, but later switched to the symbolic DD engine, which provides the results reported here. We also attempted to use the abstraction-refinement engine, but it does not support checking expected rewards, which was our focus. It is worth noting that prior to reducing the model to the stage described here, we experimented with intermediate models, particularly those with high nondeterminism. For these models, we used the abstraction-refinement engine to check the minimum and maximum probability for a certain player to win. However, the results were trivial: The minimal probability was 0 and the maximum was 1. As we mentioned earlier, this outcome is due to the "trivial schedulers" that just guess the right answer immediately for one player or the other.

In Table 4, we present the results of our experiments with Storm on the family of models described in the paper. In the discussion of the results, we refer to the experiment runs using two-digit numbers, the first one representing the strategy level of player 0 and the second one the level of player 1. So 00 stands for: both players follow the strategy of level 0. When Storm checks the expected number of rounds, that we report in the table, it chooses a scheduler in a way that the cards the players show each other will be optimal, in order to minimize the expected number of rounds, i.e., turns. We implemented this by defining a reward `rounds` inside our model, which just counts the number of turns. The results that we obtain for the expected number of rounds are similar, slightly smaller, to the ones computed with Prism and statistical model checking.

We monitored the memory consumption of each configuration using the GNU `time` command, which gives statistics about a specified program. These statistics also include the maximum resident set size which we report on in Table 4. The most memory intensive task, as expected, concerns the game in which both players play at level 0. For this configuration and all others marked with a star, garbage collection was initiated.

Somewhat surprising, the runtimes of our experiments in different configurations vary significantly: the span is between approximately 6 and 26 hours per configuration. In particular, we sometimes get very different runtimes even for configuration models that are of comparable size, see e.g., the results for the 20

Levels		Results				
Player0	Player1	$R_{\min}$ (rounds)	Execution time	Memory (Peak)	States	Transitions
0	0	20.81023631	12:05:53	7.591 TiB*	14.780 B	47.271 B
0	1	11.60163129	17:04:42	7.452 TiB*	637 M	1.952 B
0	2	7.451742675	41:11:13	7.451 TiB*	620 M	2.429 B
0	3	7.451742675	24:57:51	7.454 TiB*	620 M	2.429 B
1	0	11.11053715	26:36:12	7.459 TiB*	641 M	1.964 B
1	1	7.905685058	22:46:12	7.138 TiB	23 M	66.200 M
1	2	6.064613559	15:32:07	6.806 TiB	23 M	85.421 M
1	3	6.064613559	08:38:40	6.806 TiB	23 M	85.421 M
2	0	6.730102023	06:01:02	6.603 TiB	594 M	1.784 B
2	1	5.764917719	08:31:56	6.149 TiB	20 M	56.541 M
2	2	5.33706423	07:34:15	5.844 TiB	18 M	47.481 M
2	3	5.33706423	05:31:12	5.844 TiB	18 M	47.481 M
3	0	6.730102023	22:28:10	6.603 TiB	594 M	1.784 B
3	1	5.764917719	10:03:21	6.149 TiB	20 M	56.541 M
3	2	5.33706423	06:42:59	5.844 TiB	18 M	47.481 M
3	3	5.33706423	06:49:06	5.844 TiB	18 M	47.481 M

Table 4: Results of probabilistic model checking using Storm where M abbreviates million, and B abbreviates billion.

and 30 runs, where player 0 plays at level 2 and 3, respectively, and player 1 plays at level 0. Another curious fact about the runs 20 and 30 is that the obtained results are identical. This simply demonstrates that for two players only, the use of the global negative knowledge makes no difference. This observation is strengthened by all runs where one player plays at level 2 and at level 3, for which we obtained the same expected number of rounds. The complete list of experiments is also available via [4].

8 Conclusions

We investigated a family of probabilistic models implemented in the Prism language to reason about different playing strategies in a simplified version of the Cluedo game. In our variant, we considered two players on a board with four rooms located at the four different corners of a 3x3 grid where one of four possible characters is suspected to have eaten the birthday cake. We formalised the global knowledge of the ownership of each card and the local knowledge of each player encoding it also in our Prism models.

We considered different scenarios where each player uses different levels of knowledge and compared the effectiveness of different combinations of these levels for both players. We then employed both statistical and exact model checking to check different properties on all the models: the probability for each player to win (in the statistical case) and the expected minimum number of turns to

win the game. Encoding these models in the PRISM language and verifying such properties using exact model checking tools such as STORM was interesting and challenging. Each model has almost a thousand of lines of code. We believe that this effort could be relieved with new probabilistic programming language primitives that could support the definition of more complex model patterns at a higher level. Another option would be to define an independent higher-level probabilistic language that compiles to PRISM code. We are interested in working on this, possibly in collaboration with Joost-Pieter, in the future.

Recall that the largest generated state space of almost 15 billion states required about 8 TB of memory, while each verification task required multiple hours. To the best of our knowledge, compared with the quantitative verification competition reports [12,5], this is the second largest MDP benchmark for STORM. We hope that the proposed benchmark will stimulate further improvement of the STORM model checker, but first of all we wish you a happy birthday, Joost-Pieter!

References

1. The probabilistic model checker PRISM (2024), https://www.prismmodelchecker.org/
2. The supercomputer MACH-2 (2024), https://www3.risc.jku.at/projects/mach2/
3. Arming, S., Bartocci, E., Chatterjee, K., Katoen, J., Sokolova, A.: Parameter-independent strategies for pMDPs via POMDPs. In: Proc. QEST 2018. pp. 53–70. LNCS 11034 (2018). https://doi.org/10.1007/978-3-319-99154-2_4
4. Bartocci, E., Desharnais, J., Lindner, P., Sokolova, A.: Birthday cake Cluedo: Models and additional data (2024), https://github.com/plindnercs/birthday-cluedo
5. Budde, C.E., Hartmanns, A., Klauck, M., Kretínský, J., Parker, D., Quatmann, T., Turrini, A., Zhang, Z.: On correctness, precision, and performance in quantitative verification - QComp 2020 competition report. In: Margaria, T., Steffen, B. (eds.) Proc. ISoLA 2020. pp. 216–241. LNCS 12479 (2020). https://doi.org/10.1007/978-3-030-83723-5_15
6. Cai, C., Ferrari, S.: A Q-learning approach to developing an automated neural computer player for the board game of CLUE®. In: Proc. IJCNN 2008. pp. 2346–2352. IEEE (2008). https://doi.org/10.1109/IJCNN.2008.4634123
7. van Dijk, T., van de Pol, J.: Sylvan: Multi-core decision diagrams. In: Proc. TACAS 2015. pp. 677–691. LNCS 9035 (2015). https://doi.org/10.1007/978-3-662-46681-0_60
8. van Ditmarsch, H.P., van der Hoek, W., Kooi, B.P.: Concurrent dynamic epistemic logic for MAS. In: Proc. AAMAS 2003. pp. 201–208. ACM (2003). https://doi.org/10.1145/860575.860608
9. Dixon, C.: Using temporal logics of knowledge for specification and verification - a case study. J. Appl. Log. **4**(1), 50–78 (2006). https://doi.org/10.1016/J.JAL.2005.08.003
10. Groote, J.F., van Heesch, S., Volk, M.: Formal modelling and analysis of slot machines (2024), https://arxiv.org/abs/2407.06809, arXiv:2407.06809 [cs.LO]

11. Groote, J.F., Wiedijk, F., Zantema, H.: A probabilistic analysis of the game of the goose. SIAM Rev. **58**(1), 143–155 (2016). https://doi.org/10.1137/140983781, https://doi.org/10.1137/140983781
12. Hahn, E.M., Hartmanns, A., Hensel, C., Klauck, M., Klein, J., Kretínský, J., Parker, D., Quatmann, T., Ruijters, E., Steinmetz, M.: The 2019 comparison of tools for the analysis of quantitative formal models - (QComp 2019 competition report). In: Proc. TACAS 2019: TOOLympics. pp. 69–92. LNCS 11429 (2019). https://doi.org/10.1007/978-3-030-17502-3_5
13. Hansen, D.M., Hansen, K.D.: Clues about bluffing in Clue: Is conventional wisdom wise? IEEE Transactions on Games **13**(3), 310–314 (2021). https://doi.org/10.1109/TG.2019.2957353
14. Hasbro: Clue Jr, The Case of the Missing Chocolate Cake (2003), http://www.hasbro.com/common/instruct/Clue_Jr.,The_Case_of_the_Missing_Chocolate_Cake_(2003).pdf
15. Hensel, C., Junges, S., Katoen, J., Quatmann, T., Volk, M.: The probabilistic model checker Storm. Int. J. Softw. Tools Technol. Transf. **24**(4), 589–610 (2022). https://doi.org/10.1007/S10009-021-00633-Z
16. Iepsma, R.: Preposterior Decision Analysis in the game Cluedo. Bachelor's thesis, University of Amsterdam (2012), https://staff.fnwi.uva.nl/b.bredeweg/pdf/BSc/20112012/Iepsma.pdf, Faculty of Science
17. Kingston, J.: Comparing question answering strategies for Cluedo. In: Proc. AISB 2017: Workshop on AI and Games. pp. 332–335. Society for Study of Artificial Intelligence and the Simulation of Behaviour (2017), https://research.brighton.ac.uk/files/452820/conference-edition-proceedings.pdf
18. Kwiatkowska, M.Z., Norman, G., Parker, D.: Prism 4.0: Verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) Proc. CAV 2011. pp. 585–591. LNCS 6806 (2011). https://doi.org/10.1007/978-3-642-22110-1_47
19. Meyer, J.C., van der Hoek, W.: Epistemic logic for AI and computer science, Cambridge tracts in theoretical computer science, vol. 41. Cambridge University Press (1995). https://doi.org/10.1017/CBO9780511569852
20. Mitchell, K.: Investigating Artificial Intelligence for Playing 'Speed Clue', a Stochastic Game involving Logical Inference. Bachelor's thesis, University of Leeds (2020), https://info.bb-ai.net/student_projects/project_reports/Games/Katie-Mitchel-Cluedo-project-report.pdf, school of Computing
21. van Ditmarsch, H.: Killing Cluedo. Natuur en Techniek **11**, 32 – 40 (2001)
22. van Ditmarsch, H.: The description of game actions in Cluedo. In: Game theory and Applications. vol. 8, pp. 1–28 (2002), https://personal.us.es/hvd/oldpubs/gta8.pdf
23. Wikipedia: Cluedo (2024), https://en.wikipedia.org/wiki/Cluedo, https://en.wikipedia.org/wiki/Cluedo. Fig. 1 by cmglee, Google, Twitter, emojione contributors, Construct, Zeus, Patrick Yavitz, La Mula Francis - This file was derived from: Cluedo board.svg, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=127937647
24. Xu, J.: Playing Clue: An Entropy-based Computer AI for the Classic Board Game. Bachelor's thesis, Princeton University Senior Theses (2013), http://arks.princeton.edu/ark:/88435/dsp01hx11xf36z, Computer Science