University of Glasgow | School of Computing Science

Honours Individual Project Dissertation

Modelling and analysis of dice-based stochastic games

Lewis Dyer

Abstract

This project investigates the viability of using model checking through the probabilistic model checker PRISM in order to evaluate the game design of various games of chance involving dice. We consider three different case studies, including a hidden information game and a game with simultaneous action selection. We use model checking to identify flaws in the game presented in each case study, including the complexity of optimal strategies compared to simpler strategies with similar effectiveness, and the disproportionate influence of early actions in the overall result of a game.

Education Use Consent

Acknowledgements

I would like to express my gratitude to Dr. Gethin Norman, my project supervisor, for his continual support and guidance throughout the project. I would also like to thank William Kavanagh for offering several useful suggestions throughout the project. Finally, I would like to thank my parents for their encouragement and support throughout what has been a rather eventful year.

Contents

1 Introduction

In this section, we motivate the importance of game design, explain why model checking is a useful technique for determining game balance, and outline the structure the dissertation.

A key aspect of game design is ascertaining the balance of a game, particularly in reference to different possible strategies for playing the game. As players repeatedly play a game, their aim is to employ better strategies at winning over time, eventually approaching the optimal strategy for a game. However, in practice, the optimal strategy for a game should be complex enough that human players cannot employ this strategy in practice, since a simple optimal strategy leads to a game which is predictable and no longer interesting for players. For instance, the game of Noughts and Crosses has an extremely simple optimal strategy which human players learn almost immediately. Conversely, games such as Chess and Go have optimal strategies, but their complexity makes them infeasible to compute, leading to complex strategic decisions over thousands of years of gameplay.

While these games are entirely based on strategy, many games utilise random elements in order to encourage players to adapt their strategies as the game progresses, making games more interesting and enjoyable to play. One particularly common method for introducing random elements into a game is via the use of dice, which are cheap, satisfying to roll and well understood by the majority of players. Consequently, this dissertation focuses on games which use dice as a key mechanic in the game.

Currently, the most common method of evaluating game balance is via manual *playtesting*, where players repeatedly play early versions of games, providing feedback which designers can use in order to refine their games. However, this method is flawed for multiple reasons. Playtesting is resource intensive, requiring many people to spend significant amounts of time playing a game. In addition, for complex games with many possible actions, playtesting cannot consider all possible strategies. A more subtle issue is that playtesters are often experienced players, who may attempt to apply knowledge from other games in order to play effectively. This further limits the possible strategies that playtesters end up employing. These reasons motivate the use of an automated approach to analysing games, such as probabilistic model checking.

In many systems, simulation and testing are the two primary methods for analysing the behaviour of a system. While these methods are generally simple to implement, a key issue with these methods is that the system is not exhaustively analysed, and in particular statistical methods may fail to accurately consider the probability of rare events occurring in a game. By contrast, probabilistic model checking exhaustively considers all possible actions at each point of a game, leading to more precise observations about a game.

The remainder of the dissertation is outlined as follows. Chapter 2 discusses preliminary background information for the case studies presented in the dissertation, including an introduction to probabilistic model checking and the PRISM model checker. Chapters 3, 4 and 5 present the case studies, discussing specific examples of games in more detail, including extending models in order to allow for more complex types of games. Chapter 6 discusses several tools developed in order to improve the rigour of the experiments performed throughout the case studies, while Chapter 7 summarises the project and offers multiple suggestions for future work.

2 Background

In order to begin formal analysis of a game, we must first formally define an abstract mathematical model of a game, along with properties of this game we wish to consider.

2.1 Discrete-time Markov chains

We first introduce discrete-time Markov chains (or DTMCs), as described by Kwiatkowska et al. in [14]. This model is refined and adapted in each case study to develop different types of model suited to different types of games.

Definition 2.1. Given a set of atomic propositions, denoted AP, a discrete-time Markov chain (DTMC) is a tuple $\mathcal{D} = (S, \bar{s}, \mathrm{P}, L)$ such that:

- S is the finite set of states and $\bar{s}$ is the initial state;
- $\mathrm{P} : S \times S \to [0,1]$ is the *transition probability matrix*, where $P(s, s')$ is the probability of transitioning from state s to state s', such that the sum of probabilities of outgoing transitions from each state is 1.
- $L : S \to 2^{AP}$ is the *labelling function*, where for $s \in S$ a state, $L(s)$ denotes the subset of atomic propositions which hold at that state. In particular, we let $Sat(a) = \{s \in S \mid a \in L(s)\}$.

Playing a game is represented as an infinite *path*, denoted as a sequence $\omega = s_0 \to s_1 \to \cdots$ where $\mathrm{P}(s_k, s_{k+1}) > 0$ for all $k \geq 0$, or in other words where each transition is possible. In states where a game terminates, we typically define a self-loop in each of these states with probability 1 to ensure the path remains infinite.

DTMCs may also be augmented with *rewards* - numerical values representing various characteristics of a particular execution of a DTMC.

Definition 2.2. A *reward structure* for a DTMC $\mathcal{D}$ is a tuple (ρ, ι) where $\rho : S \to \mathbb{R}$ is a state reward function, associating each state with the value of a reward and $\iota : S \times S \to \mathbb{R}$ is a state transition function, associating each transition with the value of a reward.

These reward values can be utilised and examined in various different ways to evaluate a DTMC. More detail on this is provided in Section 2.2, but in particular the expected value of a reward value over all possible paths through a DTMC is frequently considered.

As a motivating example, we define a classic board game along with a reward using a DTMC.

Example 2.3. A game of chess can be defined as a DTMC from Definition 2.1 as follows:

- S is the set of states for the game, representing all possible board positions throughout a game of chess, along with the current player's turn;
- $\bar{s}$ is the initial board position of a standard game of chess, where white makes the first move.
- P assigns each valid move a probability of being chosen at a particular board position, where a transition is valid if the transition can be presented by one legal move according to the rules of chess by the player by the current player.

- L labels each state with a set of propositions which are true at that state. For instance, if we let $\mathtt{c_w}$ be the atomic proposition that the white player is in check in a given state, then for $s \in S$ a state, $\mathtt{c_w} \in L(s)$ means that the white player is in check in state s.

We may also define a reward structure $R_w = (\rho_w, \iota_w)$ where ρ_w returns 0 for all states, and ι_w returns 1 for all transitions where the white player captures a black piece, and returns −1 for all transitions where the black player captures a white piece. □

2.2 Property specification

When analysing probabilistic models, including DTMCs, we consider two main types of properties: *probabilistic reachability* properties and *reward-based* properties. We may define properties in terms of *Probabilistic Computation Tree Logic* (PCTL), as defined by Hansson and Jonsson in [9]. These properties are comprised of path quantifiers and temporal operators, though for the purposes of this dissertation we only focus on one temporal operator: the future operator, denoted F, considers whether a particular proposition *eventually* holds for some state on a given path.

For probabilistic reachability properties, we also include the P operator, which considers the probability of a particular property holding across all possible executions of the DTMC. For instance, using Example 2.3, if we define `white_win` as the proposition that white is considered to have won the game in the current state, the property $\mathrm{P}_{=?}$[F `white_wins`] represents the probability that white eventually wins the game of chess.

For reward-based properties, PRISM defines an extension to PCTL introducing the R operator [14], which allows for properties where the value of a reward is taken into account. For the purposes of this dissertation we only consider one type of reward-based property, namely the *reachability reward* property, referring to the expected cumulative value of a reward along a path, until a state satisfying a particular proposition is reached.

Example 2.4. Again building on Example 2.3 and using the reward structure presented in this example, the property $\mathrm{R}_{=?}$[F `white_win`] represents the expected value of the difference between the number of pieces the white and black player control in the cases where the white player eventually wins. □

These reward-based properties can provide useful information when balancing games. For instance, we may initially expect that the difference in pieces captured is a useful measure of how "close" a game of chess is, and this measure could be evaluated via model checking, where we may find it insufficient as a balance measure. For instance, this measure gives equal weighting to every piece, whereas some pieces such as queens are often considered more valuable than other pieces. This behaviour can be modified by changing the state transition function of the reward structure from Example 2.3.2, such as by defining ι_w such that it returns 1 when the white player captures a black pawn, and ι_w returns 2 when the white player captures a black piece that is not a pawn, along with the negation of these values when the black player captures a white pawn and a white non-pawn piece respectively.

2.3 Probabilistic model checking

Now that we have defined a DTMC, along with properties of that DTMC to consider, we now define probabilistic model checking - the class of methods for verifying the probability of some event occurring during the process of playing a game. Compared to non-probabilistic model checking, this presents some unique challenges, such as in the following example.

Example 2.5. Consider a game of chess, defined in Example 2.3, and let $\mathtt{m_w}$ and $\mathtt{m_b}$ be the atomic propositions that the white player and black player, respectively, are in checkmate in a particular position. In non-probabilistic model checking, we may consider the reachability of $\mathtt{m_w}$ - in other words, we consider whether we can always reach a state where the white player is in checkmate, regardless of which reachable board position we start in.

The result of such a query is binary - either we can always reach a state where $\mathtt{m_w}$ holds, or there exists a board position where white can never lose via checkmate. Indeed, in chess the latter holds - for instance, if the black player only has a king remaining, then there is no way for white to lose via checkmate, since black has insufficient material. Since our model is essentially a directed graph, we can employ standard widely-known algorithms such as a breadth first search in order to verify such reachability properties.

However, if we consider probabilistic model checking, then as well as qualitative properties, such as the above, we can also consider quantitative properties, such as the *probability* that the white player is in checkmate from a particular board position. In PCTL, this may be expressed as the property $P_{=?}[\mathrm{F}\ \mathtt{m_w}]$. This is a far more general and powerful property - for instance, an advanced chess player, who has found themselves in a state such that they are in disadvantage over the other player, may seek to reach states where the game is unlikely to end in checkmate, maximising the probability of a draw. However calculating this probability is more complex than for non-probabilistic reachability problems, since we must also consider every possible path that can be taken in order to reach a particular state. □

As we defined reachability and reward-based properties separately, we also define model checking reachability and reward-based properties separately, each using similar methods.

2.3.1 Model checking probabilistic reachability properties

Evaluating probabilistic reachability properties is a recursive process, since we need to consider the probabilistic reachability of each state that a given state can reach. Hence we need to define the "base cases" of this recursion, which are described below as in [14].

Definition 2.6. For a DTMC $\mathcal{D}$, with a set of states S and a given atomic proposition a, we can partition the state space into the following three subsets:

- S^{yes} is the set of all states where a eventually holds with probability 1;
- S^{no} is the set of all states where a never holds in any subsequent reachable state;
- $S^{?} = S \setminus (S_{yes} \cup S_{no})$ the set of all states where the probability of a holding eventually is in the interval $(0, 1)$.

The sets S^{yes} and S^{no} are fairly straightforward to compute using graph-based algorithms. The precise details of their construction are described further in [14], we will just state that S^{no} is constructed via a simple reverse traversal of the DTMC, starting from $Sat(a)$ and obtaining all states that can reach a state in $Sat(a)$, then removing these states from S. S^{yes} is constructed similarly, except this construction starts with S^{no} and traverses the DTMC to find all states where the probability of reaching a state in $Sat(a)$ is less than 1, then removing these states from S.

After this precomputation process, we can now calculate the probability of a probabilistic reachability property holding over the entire state space S, by extending the calculation to $S^{?}$.

Definition 2.7. For a DTMC $\mathcal{D}$, with a set of states S, a partition of S as in Definition 2.6, and an atomic proposition a, we denote $Prob^{\mathcal{D}}(s, \mathrm{F}\ a)$ as the probability that an execution of $\mathcal{D}$ starting at state s eventually reaches a state where a holds. Furthermore, we have for any $s \in S^{?}$:

$$Prob^{\mathcal{D}}(s, \mathrm{F}\ \Phi) = \textstyle\sum_{s' \in S^{?}} \mathrm{P}(s, s') \cdot Prob^{\mathcal{D}}(s, \mathrm{F}\ \Phi)\ + \textstyle\sum_{s' \in S^{yes}} \mathrm{P}(s, s')$$

From Definition 2.7, calculating reachability probabilities requires solving a system of linear equations containing $|S^{?}|$ unknowns. There are many different methods for solving these equations, as discussed further in [5], but in practice iterative methods (such as the Gauss-Siedel method or power iteration) are preferred to direct methods (such as Gaussian elimination) - iterative methods generally do not give exact solutions, opting for values which eventually converge to an exact solution, but direct methods can require more time and space to compute an exact solution, and in many circumstances the performance improvements justify a slightly less precise result.

2.3.2 Model checking reward-based properties

The approach to model checking reward-based properties is similar in spirit to checking reachability properties, in that we recursively construct a series of linear equations, which may be solved using any standard method. However, a key remark is that, when evaluating the expected cumulative reward value until a state where some proposition a holds, we assume that a state where a holds is always eventually reached with probability 1. If this does not occur for some path in a DTMC, then a reward can accumulate infinitely often. Hence, the expected reward value in this case is considered to be infinite.

In order to evaluate reward-based properties, we first define a random variable representing the cumulative reward value of an arbitrary infinite path.

Definition 2.8. For a DTMC $\mathcal{D}$, let $Path(s)$ be the set of all paths of $\mathcal{D}$, starting in some stats s, and let T be the set of states in $\mathcal{D}$ where some proposition a holds, along with denoting ρ and ι as state and transition reward functions respectively. Then $X_{Reach(T)} : Path(s) \to \mathbb{R}$ is a random variable where, for any infinite path $\omega = s_0 \to s_1 \to s_2 \to \cdots$:

$$X_{Reach(T)}(\omega) = \begin{cases} 0 & s_0 \in T \\ \infty & \forall i \geq 0.\, s_i \notin T \\ \sum_{i=0}^{k_T-1} \rho(s_i) + \iota(s_i, s_{i+1}) & \text{otherwise} \end{cases}$$

where $k_T = min\{j \mid s_j \in T\}$, the first state in the path where the atomic proposition a holds.

From this, we then define $ExpReach(s, T) = \mathbb{E}(X_{Reach(T)})$, the expected value of the cumulative reward value over all possible infinite paths through $\mathcal{D}$, starting from state s.

Model checking reward-based properties consists precisely of calculating $ExpReach(s, T)$ for all $s \in S$, which can be expressed as the solution of a system of linear equations using the above definition:

$$ExpReach(s, T) = \begin{cases} \infty & s \notin S^{yes} \\ 0 & s \in T \\ \rho(s_i) + \sum_{s' \in S} \mathrm{P}(s, s') \cdot (\iota(s_i, s_{i+1}) + ExpReach(s', T)) & \text{otherwise} \end{cases}$$

Note that $ExpReach(s, T) = \infty$ precisely where there exists some path in $Path(s)$ such that a never eventually holds, or equivalently where T is never reached.

2.4 The PRISM model checker

PRISM [15] is a software tool which implements probabilistic model checking on a wide variety of models. In particular, PRISM defines two languages in order to facilitate model checking: the *PRISM language* for formally defining models, along with the *PRISM property specification language* for formally defining PCTL properties (more precisely, an extension of PCTL adding reward-based properties).

2.4.1 The PRISM modelling language

PRISM provides a modelling language in order to formally specify probabilistic models. In particular, many probabilistic models exhibit *emergent complexity*, where simple descriptions of behaviour can lead to complex models. Hence, a modelling language is essential to allow describing the behaviour of a model, rather than a more complicated model specification. Note that we do not provide a full description of the PRISM modelling language here, instead focusing on the key features used in the various case studies - more details are available in the PRISM manual [1], along with additional examples.

PRISM models are defined as a set of *modules*, where each module consists of a set of *variables* and a set of *commands*. Variables may be either bounded integers or boolean value. Variables can be accessed by any module, but modules can only alter their own variables.

The key structure of a PRISM model is a *command*, of the following form:

```
[] guard -> prob_1 : update_1 + ... + prob_n : update_n;
```

A *guard* represents a proposition, identifying the set of states where the command is applicable. The command here has n transitions, with a probability of each transition occurring such that the probabilities sum to 1. Each update is comprised of a set of variable updates. And each possible combination of values for each variable represents a state of the resulting model, so a set of variable updates represents a transition between two states. Guards may overlap, which allows for nondeterministic behaviour if supported by the type of model being constructed.

In some cases, there may be a desire for modules to synchronise commands - for instance, a player model making an action and a counter module updating a phase variable. This behaviour is supported via the use of *action labels*. As an example, consider the following PRISM command:

```
[cover_1] b1=0 -> (b1'=1);
```

Here, very little logic is given for when the variable `b1` is updated. However, the `cover_1` action label allows this command to synchronise with another command with the same action label in other modules. Action labels are frequently used in order to describe an execution of a model, which is particularly useful when debugging a model via manual simulation.

For convenience, PRISM models also define *formulas* and *labels*. Formulas act as shorthand for a particular expression, allowing for a complex expressed to be referred to on multiple occasions without needless code duplication. Labels act very similarly, representing a set of states where a particular boolean condition holds. A simple example of a formula is given below:

```
formula score = b1*1 + b2*2 + b3*3 + b4*4 + b5*5 + b6*6;
```

Finally, PRISM allows for specifying reward structures. Reward structures include a guard specifying when the reward is applicable, along with the value of a reward when this guard is satisfied. Note that PRISM supports two main types of rewards - instantaneous rewards, considering the value of a reward in a specific state, and cumulative rewards considering the total value of a reward throughout an execution of the model. The following example represents two rewards, which are intended to be cumulative and instantaneous respectively:

```
rewards "no_rolls"
    state=0: 1;
endrewards
```

```
rewards "cards_in_hand"
    true: card_count;
endrewards
```

Note that this distinction between cumulative and instantaneous rewards is purely contextual, and hence the correct reward type to use is based on the intended behaviour of the reward, as opposed to any inherent characteristics of the reward itself. Hence, this distinction is applied at the property specification stage, rather than model construction.

Action labels may also be utilised in order to represent transition rewards, such as the following example counting the number of times a player chooses to roll 2 dice:

```
rewards "2_rolls"
    [roll_2_dice] true : 1
endrewards
```

2.4.2 The PRISM property specification language

When specifying properties of a model to be evaluated, PRISM defines a property specification language which supports and extends a variety of probabilistic temporal logics, although for the purposes of this dissertation we only consider the extension of PCTL with reward-based properties, as described in 2.2. As a simple example, consider the following property in PRISM:

```
P=? [ F game_over&score=10 ]
```

The main difference between the PRISM modelling language and "pure" PCTL syntax is the definition of the atomic proposition. Here, we refer to labels and variables from within a PRISM module in order to define an atomic proposition, allowing us to easily define a set of states where this proposition holds, namely where the game is over and the current score is 10. Reward-based properties can also be defined in a similar manner, specifying the reward name being considered:

```
R{"total_boards"}=? [ F game_over ]
```

2.5 Related work

One of the first papers to discuss automatic game balancing was presented by Hom and Marks [10], who focused on combining and mutating the rules of various classic board games using genetic algorithms in order to define a new game, whose win rate was then analysed using a commercial game engine to determine the game's balance. A key feature of future work in automatic game balancing is the development of more sophisticated measures of balance. Hom and Marks primarily focused on two-player games using the win rate of each player, while Jaffe et al. [11] build on this foundation by developing a framework designed around answering a wider variety of balance questions, such as the importance of short-term and long-term strategy in decision making, via the use of agents with restricted behaviour, coupled with opponents who are capable of exploiting these restrictions.

These ideas are then applied by Milazzo et al. [20], using probabilistic and statistical model checking in multiple case studies to answer balance questions about games. In particular, the authors show that model checking can be applied to balance games on several different "levels" at once, both adjusting the parameters of various game mechanics and the inclusion of these mechanics in the first place (such as adjusting how much health players have, whether players can have multiple lives, and how these mechanics interact). This idea is further extended to develop Chained Strategy Generation [13], a technique for balancing games over the course of a *metagame*, where different selections of material choice in a game (such as players or teams in a football game) change in popularity over time. Kavanagh and Miller also consider an alternative formulation of player skill in [12], where actions have an associated *cost* based on comparison with the best possible action in a particular state, presenting another method for analysing strategies by comparing them to the optimal strategy for a game.

However, this literature review revealed a significant gap in current research on formal verification in game balancing. The games which have currently been considered are primarily turn-based, and more crucially they are all *perfect information* games, where the current state of the game is fully visible to all players. As a result, a key aim of this dissertation is to consider model checking on a wider variety of games, particularly in the second and third case studies considering hidden information games and concurrent games respectively, which we do not believe have been considered in previous literature.

3 Case Study 1: Shut the Box

In this case study, we introduce a game called Shut the Box. We first introduce a n extension of DTMCs, called Markov decision processes (MDPs), allowing for the modelling of decision making during a game. We then model Shut the Box using MDPs, analyse multiple variants of this game under different strategies, and use this analysis to evaluate the design of Shut the Box.

3.1 Shut the Box description

Shut the Box is a single player game, where the player aims to cover as many *boards* as possible through a series of dice rolls. The game starts with a series of boards, which are wooden panel on hinges in the physical version, as shown in Figure 3.1, each numbered sequentially starting from 1, with each board initially uncovered. In each round, a player rolls a set of dice values. The player must then cover a set of uncovered boards whose sum is equal to the sum of the dice. For instance, if the player rolls a 1 and a 2, then the player must either cover boards 1 and 2, or cover board 3. When a board is covered, it can no longer be considered when covering future boards. For instance, in the above example, if board 2 has already been covered, the player must cover board 3, provided that board 3 was still uncovered. The game proceeds in this manner until the player cannot cover a set of uncovered boards which match the sum of the dice values rolled. A player's *score* for a round of Shut the Box is then defined as the sum of all covered boards when the game ends, and a player's objective for Shut the Box is to maximise their score.

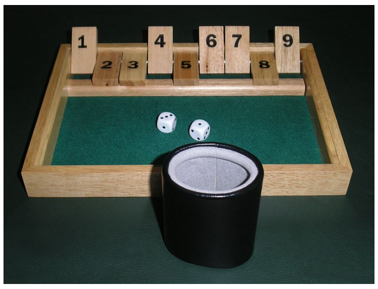

Figure 3.1: *A physical version of the 9-board variant of Shut the Box adapted from [23].*

Throughout the case study, unless otherwise stated, we consider the variant of Shut the Box where there are 12 boards, and the player rolls two six-sided dice in each round.

When playing Shut the Box, it soon becomes clear that some configurations of boards are highly desirable, while others are more challenging. For instance, consider the situation in Figure 3.2, where the 4 boards numbered 1, 2, 3 and 4 remain uncovered and suppose the sum of the dice values rolled equals 7. As shown in Figure 3.2, there are two possible coverings available. Consider the two possible coverings, we see that leaving boards 1 and 2 uncovered is more valuable than leaving board 3 uncovered - in particular, if the sum of the dice values rolled equals 3, then every board can be covered in both cases, but if the sum equals 2, then board 2 may be covered in the former case, while no valid covering exists in the latter case and the game would end. Hence

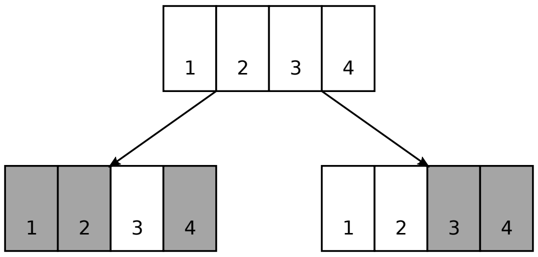

Figure 3.2: When boards 1 to 4 are all uncovered, and a 7 is rolled, there are two possible valid coverings - covering boards 1, 2 and 4 (on the left), and covering boards 3 and 4 (on the right).

covering boards 3 and 4 leads to a higher expected score at the end of the game than covering boards 1, 2 and 4.

Intuitively, this suggests that lower numbered boards are more valuable later in the game, since they increase the set of possible die rolls that a player can roll without ending the game. From this intuition, we develop two potential strategies for playing Shut the Box. The *high-board strategy* is the strategy where the player always elects to cover the highest numbered board in each round, while the *low-board strategy* is the converse, where the player always elects to cover the lowest numbered boards in each round.

To evaluate the effectiveness of these strategies, model checking is a suitable strategy. Since these strategies are deterministic, we may define DTMCs that model Shut the Box under these two strategies, define the score of the game as a reward structure, and evaluate the expected score when the game terminates. In addition to this analysis, we are also interested in the *optimal* strategy - that is, the strategy which maximises the expected score of Shut the Box. Trying to enumerate and consider every possible strategy would be challenging, both conceptually and computationally. Instead, we model the nondeterministic variant of the game, where the player makes a nondeterministic choice between all possible coverings after each dice roll, and derive an optimal strategy from here by choosing a series of actions in order to induce an optimal deterministic strategy. In order to achieve this, in the next section we introduce MDPs, a generalisation of DTMCs which support nondeterminism.

3.2 Background

We now introduce Markov decision processes (MDPs) as described in [6], which are generalisations of DTMCs allowing for actions to be taken at each state, each leading to different probabilistic transitions from that state. We then consider adversaries - resolutions of nondeterministic choice in MDPs - and briefly discuss how optimal adversaries are computed.

3.2.1 Markov decision processes

First, we define an MDP as an extension of a DTMC.

Definition 3.1. A Markov decision process (MDP) is a tuple $(S, \bar{s}, A, \delta, L)$, where S, $\bar{s}$ and L have the same meanings as in Definition 2.1, A is a finite set of *actions* on the MDP and $\delta : S \times A \rightarrow Dist(S)$ is a partial transition probability function, where pairs of states and actions are mapped to probability distributions denoting a transition to another state. In particular, we denote $Act(s)$ as the set of actions at s such that $\delta(s, a)$ is defined.

We must also modify our definition of reward structures, since transition rewards now also consider the action taken.

Definition 3.2. A reward structure for an MDP $\mathcal{M}$ is a tuple (ρ, ι) where $\rho : S \rightarrow \mathbb{N}$ is a state reward function assigning each state to a reward and $\iota : (S \times A) \rightarrow \mathbb{N}$ is a action reward function, associating each transition from s under action a with a reward.

For example, in a game of Shut the Box, A is the set of possible subsets of boards that the player can cover given the dice values rolled (such as the action of covering boards 3 and 5 simultaneously). Another possible action is rolling the dice, although there is no nondeterminism here. Hence, δ maps each state, including the current dice value and the current set of uncovered boards, to the set of all possible covering arrangements along with the associated state transition. If s is the state where boards 1, 2 and 3 are covered, and a 3 is rolled, then the two elements of $Act(s)$ are the covering of boards 1 and 2, along with the covering of board 3.

A slight extension to paths is required, in order to accommodate that paths also include the actions taken for each transition. This extension is required as when considering reward-based properties of MDPs, since the value of rewards, see Definition 3.2, may depend on the actions chosen even if the sequence of states reached remains the same.

Definition 3.3. An *infinite path* through an MDP is a sequence $s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \ldots$ such that $s_i \in S$, $a_i \in A(s_i)$ and $\delta(s_i, a_i)(s_{i+1}) > 0$ for all $i \in \mathbb{N}$. In other words, each transition under the given action must be possible. A finite path is a truncation of an infinite path.

In general, more than one action is available in a state. Indeed, this is where nondeterminism is introduced into the MDP. In order to resolve this nondeterminism, we introduce the concept of adversaries.

Definition 3.4. An adversary is a function σ which maps each finite path to an action available in the last state of the path.

A key remark on this definition is that, given a finite path $\omega = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \ldots \xrightarrow{a_{n-1}} s_n$, adversaries make decisions depending on the entire execution history up to and including the state s_n, not just the state s_n itself. However, we primarily consider *memoryless adversaries*, where the adversary always picks the same action in a given state. In particular, this adversary can be viewed as a map from states to available actions. For MDPs, these are sufficient to obtain optimal strategies for probabilistic and expected reachability properties. We also remark that optimal adversaries for MDPs are non-randomised, so given a particular state the same action is always chosen with probability 1.

When the behaviour of an MDP is considered under an adversary, the nondeterministic choices are resolved, and a DTMC is obtained. Hence, under a specific adversary, we can apply the model checking techniques introduced in Section 2.3 to evaluate properties of an MDP.

Note that adversaries are often referred to using other names, depending on context, such as schedulers or policies. Throughout this dissertation, by convention we refer to adversaries when discussing the concept over general MDPs, and refer to strategies in the context of a particular game (analogous to a human developing a strategy while playing a game).

We defer discussion on how optimal adversaries are generated until Section 3.2.3. For now, we discuss how optimal values of probabilistic reachability properties are computed, then modify this process in order to provide an adversary which obtains this optimal value.

3.2.2 Probabilistic reachability in MDPs

In a similar manner to Section 2.3.1, given an atomic proposition a, we define T to be the set of states in some MDP $\mathcal{M}$ where a holds. We only consider the minimum case here - the maximum case is analogous.

Since MDPs allow for nondeterminism, we must consider the minimum probability of reaching T from each state s of the MDP, over all possible resolutions of nondeterminism. More precisely, for some atomic proposition a, and state s (which may not necessarily be an initial state of the associated MDP) we denote this probability as $\mathrm{P}^{s}_{min=?}[\mathrm{F}\ a]$. In order to compute this probability, we introduce a method known as *value iteration*, as described in [4]. This method computes a sequence $(x^n_s)_{n\in\mathbb{N}}$, denoting the minimum probability of reaching T from a given state s within n steps. This sequence converges to the optimal value, yielding an approximation of the optimal value for large enough n.

First, we perform precomputation of several sets of states, very similarly to precomputation of S^{yes} and S^{no} in Definition 2.6. The details of these constructions are omitted and described in [6].

Definition 3.5. For a given MDP $\mathcal{M}$, with state space S and atomic proposition a:

- S^{yes}_{min} is the set of states where a eventually holds with probability 1, regardless of the resolution of the nondeterminism.
- S^{no}_{min} is the state of states where a never holds in any subsequent reachable state, for some resolution of nondeterminism.

We are now ready to define value iteration for calculating the minimum reachability probability for a set of target states T, via defining each element in the sequence $(x^n_s)_{n\in\mathbb{N}}$.

Definition 3.6. Given an MDP $\mathcal{M}$, with precomputed sets of states as described in Definition 3.5, we have that:

$$x^n_s = \begin{cases} 1 & s \in S^{yes}_{min} \\ 0 & s \in S^{no}_{min} \\ 0 & s \notin (S^{yes}_{min} \cup S^{no}_{min}) \text{ and } n = 0 \\ \min_{a\in A(s)} \left\{ \sum_{s'\in S} \delta(s,a)(s') \cdot x^{n-1}_{s'} \right\} & \text{otherwise.} \end{cases}$$

Using this method, optimal values can be obtained using an iterative process, and in particular the number of iterations required for a reasonable level of convergence can be determined during the algorithm's execution, such as terminating when the difference between two consecutive terms in the sequence is less than some pre-defined convergence criterion $\epsilon > 0$. However, this specific approach is somewhat flawed, particularly when probabilities are initially small, or the sequence converges slowly. Several pieces of recent work have proposed alternative notions of convergence which address these issues, including the interval iteration algorithm described in [7], which considers upper and lower bounds on value iteration and proceeds until these bounds are sufficiently close together.

We now have a method of approximating $\mathrm{P}^{s}_{min=?}[\mathrm{F}\ a]$, which we can then modify to produce not just an approximation of the optimal values themselves, but a method for producing the choices of actions which lead to these optimal values, known as adversary generation, which we discuss in the next section.

3.2.3 Adversary generation

In Definition 3.6, the inductive step is relatively simple: to find the minimum probability of reaching a state where a holds within n steps, we consider each possible action, consider each state it can transition to, then consider the probability of reaching a state where a holds from each of these states, and finally choose the action where this value is minimised.

Hence, to compute the optimal adversary itself, we simply need to select the *action* leading to the optimal value, rather than considering the value itself. Since our adversaries are memoryless and deterministic, and hence defined as a mapping from states to actions, corresponding closely with value iteration. An example of this derivation is given below for calculating a minimum strategy.

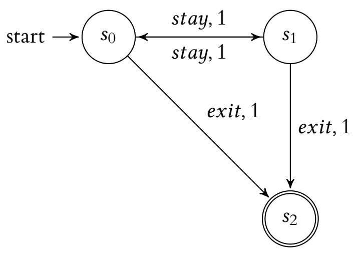

Figure 3.3: An MDP where the naive approach to adversary generation does not succeed.

Definition 3.7. The optimal adversary, obtaining the minimum probability of reaching a state where some atomic proposition a holds is given by

$$\sigma^{min}(s) = \arg\min_{a \in Act(s)} \left\{ \sum_{s' \in S} \delta(s,a)(s') \cdot \mathrm{P}^{s'}_{min=?}[\mathrm{F}\ a] \right\} .$$

A slight modification of this technique is required for the maximum reachability case to prioritise actions which arrive at the set of target states and prevent cycles which never reach the target state. To justify this, consider the following example.

Example 3.8. Consider the MDP presented in Figure 3.3. While in state s_0 or s_1, the $stay$ action will transition between s_0 and s_1, while the $exit$ action will transition to the goal state, s_2, and consider obtaining $\sigma^{max}(s_0)$ by replacing "min" with "max" in Definition 3.7. The $stay$ and $exit$ actions both have the same maximum value, so either may be selected, and similarly for $\sigma^{max}(s_1)$. But if $\sigma^{max}(s_0) = \sigma^{max}(s_1) = stay$, then the DTMC induced by σ^{max} never reaches s_2, even though this occurs with probability 1 by selecting the $exit$ action. □

Given a specific optimal adversary σ, computing the reachability probability of an MDP under the adversary is fairly simple, by considering the DTMC induced by σ and using the techniques introduced in Section 2.3.1. But the converse, that we have shown in this section, is potentially more surprising - given an optimal value for a reachability probability, an adversary can be constructed that obtains this optimal value, during the process of value iteration, with minimal additional computation.

As a final remark, adversary generation for reward-based properties is analogous to adversary generation for reachability properties, using almost identical applications of value iteration and precomputation to compute minimum and maximum expected values until reaching a state where a given atomic proposition holds. For this reason, precise details of constructing these adversaries are omitted.

3.3 Results and Analysis

In Section 3.1, the high-board and low-board strategies were defined. Our main aim in this study is to analyse these strategies, compare them to the optimal and suboptimal strategies for Shut the Box across different variants, and evaluate the design of Shut the Box based on this information.

3.3.1 The standard variant

Table 3.1 shows the expected score for the high-board and low-board strategies defined in Section 3.1, along with the optimal values generated by PRISM. We note that the expected score for the pre-defined high-board strategy is close to the maximum possible expected score, and similarly with the low-board strategy and the minimum possible expected score. Further information, in particular the probability distribution of the final score under each strategy,

Strategy	Expected score
Minimum expected score	23.9742
Low-board	24.3019
High-board	42.8088
Maximum expected score	42.9186

Table 3.1: *The expected score of the standard variant of Shut the Box under different strategies.*

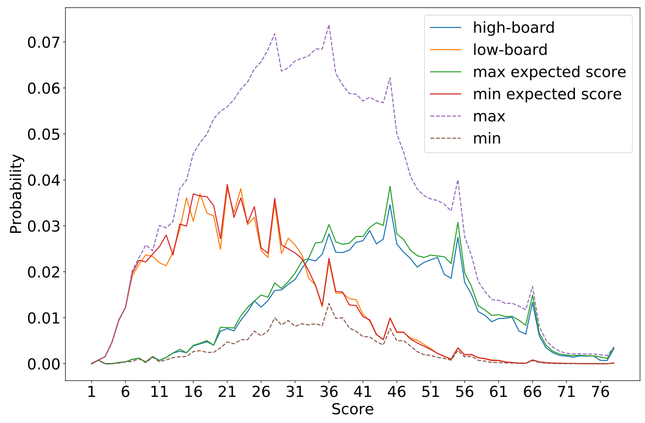

Figure 3.4: *Probability of each score in the standard variant of Shut the Box under different strategies.*

is included in Figure 3.4 with dotted lines representing the minimum and maximum possible probabilities for attaining each score in Shut the Box. In addition, Figure 3.4 demonstrates that the high-board and low-board strategies are very similar to the maximum and minimum expected value strategies respectively.

In a more abstract sense, the difference between the minimum and maximum strategies shown in Figure 3.4 represents the impact that strategy can have on the game. For instance, several peaks on the probabilities for different strategies occur at triangular numbers, which are the sum of the first n natural numbers for some n, since these present opportunities for bad strategy to significantly impair future rolls. To take a basic example, when no boards are covered, rolling a 10 in the standard variant means that the high-board strategy will cover board 10, while the low-board strategy will cover boards 1, 2, 3 and 4. The former case still allows the game to continue regardless of the value of the next roll, whereas the low-board strategy means that any future rolls between 2 and 4 will always cause the game to end. Therefore, while players can be unlucky regardless of their strategy (for instance, rolling a 2 twice in a row will always lead to the game ending), sound strategy can leave more options open in later rounds.

This initial analysis suggests that the high-board strategy is effective, and even close to optimal for obtaining high scores. We investigate this further in two main directions. We first consider the relative importance of high boards compared to low boards, then compare the choices of the optimal strategy with the high board strategy to ascertain their differences.

3.3.2 Quantifying the value of high boards

Intuitively, we expect high boards to be more important than low boards, for several reasons. Firstly, they clearly contribute more to the final score of the game. But more importantly, due to their higher value, there are fewer possible die rolls which can cover high boards, which suggests that an opportunity to cover a high board should be prioritised over covering lower value boards.

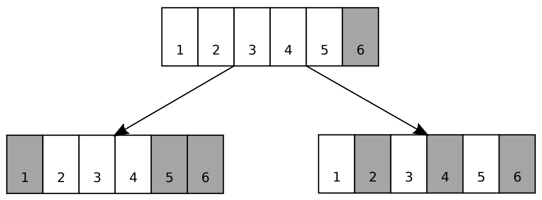

Figure 3.5: The optimal strategy covers boards 4 and 2, while the high-board strategy covers board 5 and 1. Covering boards 1, 2 and 3 is also possible, though neither strategy chooses this covering.

In essence, this is the key idea behind the high-board strategy. This correlation is exemplified further in Appendix A.

This idea can also be considered from a more combinatorial point of view. Given a dice roll of value n, each possible covering is a partition of n into distinct parts, also known as a *strict partition* of n. For instance, there are 27 possible coverings which cover board 1, but only 2 coverings which cover board 11, further exemplifying that opportunities to cover high boards are rare. This difference is exacerbated when multiple dice are used, including the standard variant, since especially high dice rolls are rarer to obtain, compared to using one dice with a uniform probability of rolling each possible value.

3.3.3 Comparing the optimal and high-board strategies

Along with generating an optimal strategy, PRISM also provides support for exporting a graphical output of this optimal strategy, allowing potential comparisons between strategies. However, due to visualisation constraints, we only consider one variant of Shut the Box - the variant where 6 boards are used, and a single 6-sided die is rolled. While this variant is relatively small and simple, comparing the high-board strategy with the optimal strategy still leads to interesting behaviour. To exemplify this, consider the following example.

Example 3.9. Consider the strategy which maximises the expected score of Shut the Box, which is 21.0 for a 6-board variant. Now consider the situation presented in Figure 3.5, where boards 2, 4 and 6 are uncovered while a 6 is rolled. The optimal strategy chooses to cover boards 4 and 2, while the high-board strategy chooses to cover board 5 and 1. This is because the high-board strategy aims to cover the highest board possible after any roll, and since 6 is already covered then 5 must be covered, and as a result 1 must also be covered.

Two modified versions of the PRISM model of the 6-board variant of Shut the Box were created, with the nondeterministic model and the model under the high-board strategies modified to represent the decision made by each strategy respectively.

From this position, the expected score of the high-board strategy is 16.4167, while the expected score of the optimal strategy is 16.4230. This is unsurprising, since the optimal strategy is specifically generated to attain the maximum possible expected score.

However, the probability of attaining a perfect score is approximately 0.1389 underthe high-board strategy, while maximum possible probability of attaining a perfect score under the optimal strategy after covering boards 4 and 2 is approximately 0.0746. Hence the high board strategy is more likely to attain a perfect score, compared to the optimal strategy for attaining the maximum expected score. □

This presents an interesting trade-off between strategies - while the optimal strategy aims to maximise the expected score of Shut the Box (by design), the high-board strategy performs better when considering the probability of attaining the maximum possible score as opposed to the maximum expected score.

Inherently, this suggests the impact of risk in comparing different strategies. A trade-off exists between attaining a higher average score and attaining the maximum possible score, and this trade-off can be exploited in the design of Shut the Box.

For instance, suppose two players play a round of Shut the Box sequentially, aiming to get a higher score than the other player. The first player's aim is to get the highest score possible, but the second player's objective is simply to earn more points than the first player, regardless of the margin of victory. If the first player scores 20 points on the 6-board variant, the second player should play the aforementioned high-board strategy, since this has a higher probability of gaining 21 points, the only possible score which can win outright. On the other hand, if the first player scores very few points, then a more consistent strategy, such as the optimal expected score strategy, may be preferred, in order to maximise the probability of winning. Alternatively, an additional incentive could be provided to encourage players to attain a perfect score, such as a "jackpot" if Shut the Box is played as a gambling game.

3.4 Evaluating the design of Shut the Box

Overall, while we have shown the high-board strategy is successful (and conversely that the low-board strategy is unsuccessful), the results of Table 3.1 indicates the main flaw with Shut the Box. When designing games, optimal strategies should be sufficiently complex so that human players cannot feasibly recreate this strategy. Optimal strategies may be seen as less enjoyable for players to play with, since decisions at various points throughout a game are effectively defined beforehand, rather than the player making decisions and adapting during the game.

While the optimal strategy for Shut the Box is somewhat complex, Section 3.3.3 shows that a simple strategy is very close to optimal. This shows that Shut the Box is flawed, because the additional complexity involved in learning a more optimal strategy does not lead to a significant improvement in the overall outcome of the game. While the trade-off between risk and reward is potentially interesting, in most cases the difference is fairly small, so the impact of this trade-off is minimal. We have also suggested several possible variants of Shut the Box to alter the balance between risk and reward, such as the addition of a "jackpot" to emphasise riskier strategies, given the lack of differentiation between strategies these variants are unlikely to lead to meaningful changes in strategy.

Throughout this case study, we have analysed a perfect information game and compared manually created strategies to optimal strategies in order to evaluate the design of that game. Now we consider a different game, where the full state of the game is unknown to each player.

4 Case Study 2 (Liar's Dice)

In this case study, we introduce Liar's Dice, a dice game utilising hidden information. We introduce partially observable MDPs (known as POMDPs) to represent this hidden information, model a small version of the game using a POMDP, and analyse the susceptibility of Liar's Dice to the snowball effect.

4.1 Liar's Dice description

Liar's Dice is a dice game for multiple players, where each player must be able to bluff and detect opponent's bluffs in order to win. Liar's Dice takes place over a series of rounds, where each player rolls their dice, keeping the values on the dice hidden from other players. A player then makes a *bid*, which is comprised of a face value on a die, and the number of dice that show that value. Players then rotate in turn, choosing to either make a higher bid (with either a higher face value, a higher number of dice, or both), or challenge the previous bid. If a challenge is made, all dice are revealed. If the bid was correct, the challenging player loses a die. If the bid was incorrect, the bidding player loses a die. The player who lost starts the next round, and play continues until only one player has any remaining dice.

In Liar's Dice, every player starts with the same amount of information, since every player starts with the same number of dice. However, as the game progresses, some players will have more dice than others, changing bidding behaviour, as demonstrated by the following example.

Example 4.1. Consider the situation presented in Figure 4.1, where the first player has rolled two 2s while the second player has rolled a 5. The first player can confidently make a bid that there are two 2s, but the second player cannot see enough dice to determine the correctness of this bid. Hence, the second player has three options:

- If the second player challenges the bid, they will lose a dice, and therefore lose the game;
- If the second player increases the face value of the bid, then the first player can immediately challenge the bid and win the game. Since the first player knows the values of all dice except for one, they know that there cannot be a pair of dice in the game with face value higher than 2.
- If the second player increases the quantity of the bid, to bidding that there are three 2s, the first player can challenge and win with probability 1, although the first player's belief is that they will win with probability $\frac{5}{6}$, since they do not know whether or not player 2 has a 2 on their dice.

As a result, the first player is able to use their increased access to information in order to increase their chances of success. □

This example represents a potential issue with the design of Liar's Dice. Initially we expect that the probability of either player losing a round is even, depending primarily on the strategies the players employ. However, as the game progresses, players with fewer dice are more likely to lose subsequent rounds, making it harder and harder for players to win from behind, a phenomenon known as the *snowball effect* in game design, as discussed further in [2].

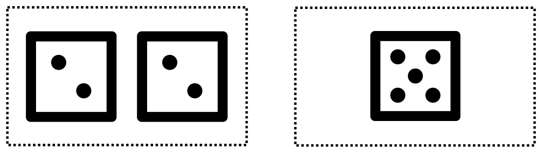

Figure 4.1: *A game of Liar's Dice, where the first player has rolled two 2s and the second a 5.*

In particular, the snowball effect means that the overall results of games may be decided fairly early on, even if the games are long. This is frustrating for players — the players are still required to play several rounds of a game where the result is already a foregone conclusion. Moreover, with multiple players, this presents an opportunity for one player to be eliminated very early, since the game continues without their involvement.

Our aim when analysing Liar's Dice will be to examine to what extent Liar's Dice exhibits the snowball effect. Firstly, we introduce a variant of MDPs which allows for partial observability, in order to model the hidden information present in Liar's Dice.

4.2 Background

A key aspect of Liar's Dice is partial observability, which we now introduce in order to augment MDPs, as described in [21].

Definition 4.2. A partially observable Markov decision process (POMDP) is a tuple $\mathcal{P} = (S, \bar{s}, A, \delta, L, \mathfrak{O}, obs)$ such that:

- $(S, \bar{s}, A, \delta, L)$ is an MDP, as in Definition 3.1.
- $\mathfrak{O}$ is a finite set of *observations.*
- $obs : S \to 2^{\mathfrak{O}}$ labels each state with a subset of observations.
- For any states $s, s' \in S$, if $obs(s) = obs(s')$ then the available actions at s and s' must be identical. When this occurs, we say that states s and s' are *observationally equivalent.*

The last point in this definition indicates how POMDPs can represent hidden information. Rather than being able to directly access every state, decisions can only be made based on observations. For instance, in many card games, some cards will be visible to a particular player, while others are hidden. In order for two states to be observationally equivalent in a model of this game, we only require that the visible cards are the same in both states, while the hidden cards can range across any permutation of valid cards. If a player could view the full state of the game, they would be able to view the value of hidden cards, which would allow for strategies to be generated for models which break the rules of the games they are intended to represent. For Liar's Dice, a player can observe their own dice, the current bid and whether a challenge has been made, but not any opponent's dice.

Since POMDPs are an extension of an MDP, adversaries for POMDPs are defined in terms of adversaries for the corresponding MDP. However, in order to reflect the added constraints that observations provide, we require that adversaries on observationally equivalent paths are equivalent:

Definition 4.3. For a POMDP P, an adversary σ of P is an adversary of the underlying MDP such that for any paths:

$$\pi = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \ldots s_n \quad \text{and} \quad \pi = s'_0 \xrightarrow{a'_0} s'_1 \xrightarrow{a'_1} \ldots s'_n$$

if $obs(s_i) = obs(s'_i)$ and $a_i = a'_i$ for all $i \in \mathbb{N}$, then $\sigma(\pi) = \sigma(\pi')$. In other words, σ makes the same decisions after observationally equivalent paths.

This definition references a key difference in optimal adversary generation between MDPs and POMDPs. In MDPs, optimal adversaries are deterministic and memoryless, so adversaries map states to actions. This allows for aforementioned methods such as value iteration to be applied, allowing for efficient computation of optimal values and adversaries attaining these optimal values. By contrast, Madani et al. show in [19] that obtaining optimal values and adversaries in POMDPs, both for reachability properties and reward-based properties, is undecidable. We instead focus on approximate solutions presented in the following section.

4.2.1 Belief MDPs

First, we introduce the *belief MDP* of a POMDP. Given a POMDP, we may construct an equivalent MDP, which consists of beliefs, which are probability distributions representing which state the player believes they are currently in.

Definition 4.4. Given a POMDP $\mathcal{P} = (S, \bar{s}, A, \delta, L, \mathcal{O}, obs)$, as in Definition 4.2, the *belief MDP* of $\mathcal{P}$ is given by $\mathcal{B}(\mathcal{P}) = (Dist(S), \delta_{\bar{s}}, A, \delta^{\mathcal{B}}, L)$. The states of this MDP represent *beliefs* about the state of the player in the corresponding POMDP, and the transition function becomes:

$$\delta^{\mathcal{B}}(b,a)(b') = \textstyle\sum_{s\in S} b(s) \cdot \left(\sum_{o\in\mathcal{O}\wedge b^{a,o}=b'} \left(\sum_{s'\in S\wedge obs(s')=o} \delta(s,a)(s')\right)\right) .$$

In particular, $b^{a,o}$ represents the belief reached after performing action a with observation o in belief b, calculated as:

$$b^{a,o}(s') = \begin{cases} \dfrac{\sum_{s\in S}\delta(s,a)(s') \cdot b(s)}{\sum_{s\in S} b(s) \cdot \left(\sum_{s''\in S\wedge obs(s'')=o} \delta(s,a)(s'')\right)} & obs(s') = o \\ 0 & \text{otherwise.} \end{cases}$$

We also need to modify how state and transition rewards are calculated for a belief MDP.

Definition 4.5. Let ρ be a state reward function and ι be an action reward function for some POMDP $\mathcal{P}$. For the corresponding belief MDP, we set:

$$\rho^{\mathcal{B}}(b) = \textstyle\sum_{s\in S}\rho(s) \cdot b(s) \quad \text{and} \quad \iota^{\mathcal{B}}(b,a) = \sum_{s\in S}\iota(s,a) \cdot b(s) .$$

In particular, this modification of rewards ensures that reachability and reward-based properties coincide between a POMDP and its corresponding belief MDP as shown in [21].

One key remark about this belief MDP is that its state space is now continuous, since the states of the belief MDP represent probability distributions of states in the POMDP. This means that the number of states in the belief MDP is infinite (indeed, uncountably infinite), so the techniques from Section 3.2.2 are no longer feasible. Instead, we use a finite set of grid points in order to obtain an upper bound of the property in question, then directly generate a strategy on the POMDP itself to obtain a lower bound.

4.2.2 Property verification in POMDPs

For simplicity, we only describe maximum reachability probabilities - the other cases of minimum probabilities and expected rewards are similar.

First, we obtain an upper bound for the property in question. The key idea, as presented by Bertsekas and Yu in [3], is to consider a finite set of states in the belief MDP, known as *representative beliefs*, which form a convex hull of $Dist(S)$. Multiple methods are possible for choosing these representative beliefs, with the simplest method producing a uniform grid G_M as follows:

$$G_M = \left\{ \frac{1}{M}v \mid v \in \mathbb{N}^{|S|} \wedge \textstyle\sum_{i=1}^{|S|} v_i = M \right\}$$

The resolution of the grid is denoted by M. The number of grid points in G_M is the number of non-negative integer solutions to the equation $v_1 + \cdots + v_{|S|} = M$, which is equal to $\binom{M+|S|-1}{M}$. Hence, the grid size is exponential in M.

When this grid has been defined, value iteration is applied to the belief MDP, in the same manner as Section 3.6. However, for beliefs which are not in G_M, interpolation is applied, such as via Freudenthal triangulation as discussed further in [18], in order to avoid directly computing values for all beliefs in the belief MDP.

During value iteration, in a very similar manner to Section 3.2.3, an adversary σ^* can be obtained, and the DTMC induced under σ^* can be solved in order to obtain an approximation for the desired property. However, this adversary is a finite-memory adversary, and as previously discussed this will not always suffice for determining an optimal adversary for a POMDP. As a result, the value obtained under this strategy is a lower bound for the optimal value.

Finally, the resolution of the grid M can be increased in order to try and achieve tighter bounds on the optimal value, with the caveat that, since grid size is exponential on M, care must be taken to ensure these bounds can still be computed in reasonable time. However, due to the undecidability of property verification of POMDPs, increasing M may make bounds worse, since lower resolution grids are in general not subsets of higher resolution grids.

We have now defined an extension to MDPs which allow us to model games which utilise hidden information. Now, we model and analyse Liar's Dice to show an example of such a game.

4.3 Results and analysis

In this case study, our main aim is to analyse Liar's Dice in order to determine its susceptibility to the snowball effect, as discussed in Section 4.1. We also discuss the how to limit the state-space of the POMDPs we consider.

4.3.1 State space reduction

As discussed in Sections 4.2.1 and 4.2.2, verifying properties of POMDPs involves constructing an approximation of the belief MDP, and in particular the size of the grid presented in Section 4.2.2 increases exponentially with the number of states in the POMDP. Hence, in order to allow for efficient analysis, reducing the number of states in the POMDP is crucial.

For Liar's Dice, the key method employed to reduce model size is to model each *round*, as opposed to each *game*. There are four main reasons why this construction is preferred.

- Rounds in Liar's Dice are mostly independent - the main parameters of each game are the number of dice for each player, along with the starting player, which is always the loser of the previous round, or the first player in the first round of the game. Hence, transitioning between rounds is straightforward.
- As a result of the previous point, some configurations may appear multiple times throughout a game. For instance, suppose each player starts with 3 dice. If player 1 loses the first round while player 2 loses the following two rounds, then player 1 has 2 dice, player 2 has 1 die, and player 2 starts the 4th round. This also occurs when player 2 loses the first and third rounds, while player 1 loses the second round. Hence, we would like to store the results of this round, in order to avoid computing this round on multiple occasions. In essence, this is a form of dynamic programming for model checking.
- As an extension of the previous point, some nodes are symmetrical, such as the left and right children of the root node in Figure 4.2. This means that separate computation of each of these nodes is unnecessary, so symmetry can be exploited in order to further reduce computation.

- If user-defined strategies are employed, then the strategy used by each player can be freely exchanged between rounds. This is especially important for Liar's Dice, since we would expect that a player who has more dice than their opponents would play differently to a player with only one die.

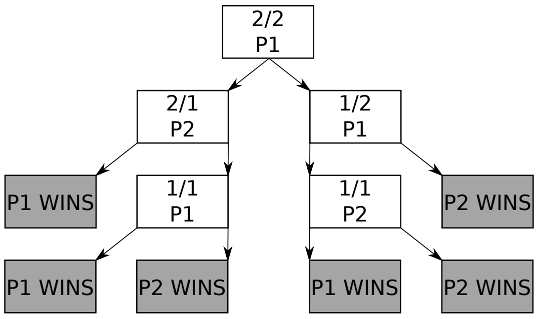

Figure 4.2: *A game tree of Liar's Dice. Each player starts with 2 dice, and each node represents the number of dice player 1 and player 2 have respectively, along with the starting player for that round.*

This construction allows us to define a *game tree*, as in Figure 4.2, where each node represents one round of the game. This construction allows for simpler computation of the probability of a player winning a game. Rather than check a property for one large model, we check the win rate for each smaller model, then traverse the tree, obtaining a weighted sum based on the probability of winning each round.

Another key implication of using game trees is that sub-games can be considered very easily. For instance, in the game presented in Figure 4.2, a handicap can easily be considered, since the right child of the root node represents handicapping player 1 by removing one of their dice at the start of the game.

4.3.2 Model constraints

Due to the requirement to reduce model size, several assumptions and limitations were imposed on the model. In particular, rather than being able to make any bids, players are assumed to only either increase the face value by 1 or increase then quantity of the bid by 1. This limits the number of possible actions per bid, since there are now only two types of bids, plus the option of challenging the other player. In particular we note that increasing the bid value by at most one each turn is potentially problematic since the player may need to make several incorrect bids in order to reach a correct bid, but this limitation is required in order to allow the game to be modelled feasibly. Another key limitation is that we only consider variants of Liar's Dice with two players, and at most 2 dice per player.

4.3.3 Pre-defined strategies

As with our first case study, we define some initial strategies in order to make comparisons to the optimal strategy. We now introduce two main types of player, exemplifying the different possibilities for each of these types of decisions. The *risky player* makes a random bid at the start of the game, even where this bid may be initially incorrect. When bidding, they always increase the quantity of the bid where possible before increasing face value, and they always challenge when the sum of the face value and quantity is at least half the maximum possible sum. The *safe player* makes a random bid at the start of the game, but only a bid which is guaranteed to appear in their hand. When bidding, they always prioritise increasing the face value of the bid rather than the quantity, and always challenge when the sum of the face value and quantity is at least 75% of the maximum possible sum.

It is important to note that these strategies are deliberately very simple, and unlikely to be effective. This is by design - here we are more interested in comparing various rounds of the game, as opposed to different strategies. These strategies are designed to represent the extremes of possible behaviour in Liar's Dice, rather than representing effective strategies for the game, so adversary generation is unlikely to add additional insight on the design of Liar's Dice. For this reason, we do not consider optimal values and strategies, and instead focus on comparing the above two simpler strategies in order to analyse the overall structure of Liar's Dice.

4.3.4 The two-dice variant

In order to analyse Liar's Dice, we consider the game tree of the two-dice version presented in Figure 4.2. To investigate the snowball effect, we consider the probability of player 1 winning at the root of the game tree, then consider the probability of player 1 winning at each child of the game tree. We consider this for each permutation of the safe and risky strategies defined in Section 4.3.3, and the results are shown in Table 4.1. In this table, a probability of below 0.5 indicates an advantage for player 1, while a probability of above 0.5 indicates an advantage for player 2.

Round	P1 loss probability (safe vs. safe)	P1 loss probability (risky vs. risky)	P1 loss probability (safe vs. risky)	P1 loss probability (risky vs. safe)
2/2, P1	0.1451	0.3403	0.5000	0.6972
2/1, P2	0.6250	0.5780	0.3611	0.1806
1/2, P1	0.3750	0.4210	0.5000	0.6389
1/1, P1	0.5278	0.3611	0.6667	0.5718
1/1, P2	0.4722	0.6389	0.4282	0.2500

Table 4.1: *The loss probability for player 1 in Liar's Dice in the 2-dice variant with each possible permutation of the safe and risky strategies.*

This table exhibits some interesting results. In particular, exchanging each player's set of dice, and changing the starting player, swaps the loss probability of each player. This is as expected for players with the same strategy, but this also means that symmetry could be exploited in these cases to reduce computation time, as suggested in Section 4.3.1. However, this does not occur when both players have different strategies, since the safe and risky strategies determine their bids differently.

When player 1 utilises a risky strategy, their loss probability is considerably lower when they make the first move, compared to going second. This can be explained by the risky player's strategy for initial bidding, which uniformly chooses any possible bid which is feasibly possible, regardless of whether the bid is correct. For instance, given 4 dice, the probability that at least three dice have the same value is $\frac{7}{72}$, or approximately 9.72%, while the probability of making a bid with quantity at least 3 is 50%. Hence, many initial bids will be incorrect to begin with. In addition, both players use a threshold-based challenge strategy, where a challenge is always made if the sum of the quantity and face value exceeds a given threshold. The thresholds used in the safe and risky strategies are 75% and 50% of the maximum possible sum respectively, and again since the initial bid is decided uniformly these thresholds may be crossed immediately following an initial bid.

Another interesting point of comparison arises between cases where both players employ the same strategy, and cases where players employ different strategies. In particular, we note that when both players use the same strategy, a player is more likely to win immediately after losing a die, since they gain the ability to make the first move next round. For instance, when both players employ the safe strategy, player 1 has two dice, player 2 has one die, and player 2 makes the initial bid, player 1 will lose the round with probability 0.6250. If player 1 loses, then they will lose the following round (and hence lose the game) with probability 0.5278. This suggests that being able to choose the initial bid is more beneficial than the additional information gained by having more dice than the opponent.

In addition, multiple cases arise when player 1 adopts the safe strategy while player 2 adopts the risky strategy where the probability of player 1 losing is 0.5, so neither player has an advantage. In these cases, after player 1's initial bid, each player alternates increasing the face value and quantity of the bid respectively, until the challenge threshold of 75% is reached. The number of steps required to each this threshold has a uniform probability of being even or odd, giving the probability of 0.5.

The results of Table 4.1 can be combined with the game tree in Figure 4.2 in order to determine the probability of the first player winning the game. In general, for a given round r, the probability of player 1 winning the game starting at round r is the weighted sum of the probability of winning each possible subsequent round multiplied by the probability of reaching that round. In this case, we suppose that the starting player chooses to play either the safe or risky strategy, then the other player may choose their strategy based on this, although we note that in practice the exact strategy chosen will not necessarily be known. In this case, player 1 aims to minimise their loss probability while player 2 aims to maximise it. For instance, in the starting round, player 1 should not choose the risky strategy, since player 2 can then use the safe strategy to attain a loss probability of 0.6972, the highest across each permutation of these two strategies. Hence player 1 chooses the safe strategy, and player 2 chooses the risky strategy. In this manner, the probability of player 1 winning is obtained as:

$$(0.5000 \cdot 0.6389) + (0.500 \cdot 0.3611 \cdot 0.4282) + (0.500 \cdot 0.500 \cdot 0.5718) \approx 0.5397$$

In a very similar manner, winning probabilities can be obtained for every starting round, as shown in Table 4.2.

Round	Probability of P1 winning game
2/2, P1	0.5397
2/1, P2	0.7935
1/2, P1	0.2859
1/1, P1	0.4281
1/1, P2	0.5718

Table 4.2: *The probability of player 1 winning eventually from each round in the 2-dice variant of Liar's Dice, with optimal selection between the safe and risky strategies.*

The results of Table 4.2 demonstrates a clear issue with Liar's Dice. If the starting player loses the first round, then their probability of winning is approximately 29%, while if they win the first round their probability of winning is approximately 79%. This is despite the overall probability of each player winning being fairly even at the outset of the game, showing the impact of the snowball effect on Liar's Dice. Another interesting paradox occurs when each player has exactly 1 die left. In this case, player 1 is actually more likely to win if they go second compared to going first. This is a consequence of each player only knowing two strategies, and having to commit to a strategy on their first move. In practice, players will adapt their strategy over time based on their opponent's actions. For instance, a player making a high initial bid may elicit a different strategy, compared to starting with a low bid and repeatedly increasing the bid over time.

4.4 Evaluating the design of Liar's Dice

Overall, our analysis of a small variant of Liar's Dice suggests that while the starting player has a small advantage overall, the snowball effect is present in Liar's Dice, meaning the game is too dependent on the results of the first round compared to subsequent rounds. While Liar's Dice attempts to mitigate this via allowing the player who just lost a dice to make the next initial bid, the added information by having more dice than the opponent outweighs this potential advantage.

This case study has also presented several limitations of modelling hidden information games using POMDPs. In particular, the key issue is that observable variables cannot be defined individually for each player, meaning that only one player can employ belief-based strategies, which dominate strategies which are not belief-based. An extension of POMDPs, known as partially observable stochastic, described further in [8], allows for different observations for each player, but in practice these models are even harder to solve than POMDPs. As a result of this, POMDPs are best suited to single-player hidden information games. For instance, POMDPs could be an effective tool in order to model the game of blackjack, since in blackjack the player only competes against the dealer, rather than other players. In addition, the dealer employs a simple strategy that is known to all players, so the dealer cannot employ a belief-based strategy in the first place.

As discussed further in Section 4.2.2, the size of the abstraction of the associated belief MDP means that analysis of larger variants of Liar's Dice, such as those which include multiple players or those which start with multiple dice, were not viable to consider for this case study. However, the principle of a game tree presented in Section 4.3.1 can still be employed for larger variants of Liar's Dice to help reduce the state space of each model, and in particular this technique could also be applied to many other applications of model checking, most notably those that occur over multiple "rounds" where the size of the model decreases each round, and each round can be parameterised by a small set of variables.

5 | Case Study 3 (26.2)

In this section we introduce 26.2, a dice game where players make decisions simultaneously. A new model type is introduced in order to facilitate this behaviour, and analyse of 26.2.

5.1 Game description

The game of 26.2 takes place over a series of rounds, where players choose how many dice they wish to roll, up to a maximum of ten dice. These dice are all rolled simultaneously, and their values are examined. If three or more of a player's dice have the same face value, or two pairs of two dice have the same face value, the player is said to have *gone bust*, and does not move in this round. Otherwise, the player moves forward a number of spaces equal to the sum of each of their dice. When a player reaches space 262 or greater (representing 26.2 miles, the length of a marathon), the game ends, and the winner is the player who has travelled the furthest. If this results in a tie, a winner is randomly selected from the players who have tied. In general, players are motivated to roll more dice per turn, in order to move further and reach the end of the game before their opponent, but this also increases a player's risk, since they may end up going bust and not moving at all in a given turn.

One key aspect of this game is that players simultaneously choose how many dice they roll. This allows for some interesting decision making, as discussed in the following example.

Example 5.1. In a game of 26.2, suppose that player 1 and player 2 are both 5 spaces away from the goal. Broadly speaking, both players have two options: they can either roll a small number of dice, giving a low probability of going bust, or roll a large number dice, leading to a higher probability of going bust, but also moving more spaces. If 26.2 was not a concurrent game, then player 1 would choose how many dice to roll first, followed by player 2. If player 1 choose to roll a small number of dice, then player 2 could roll a slightly higher number of dice, since this would increase the probability of rolling more spaces than player 1, without substantially increasing the probability of going bust. However, if player 1 instead rolled a high number of dice, then player 2 could roll a small number of dice, anticipating that player 1 has a high probability of going bust. □

In order to allow for concurrent decision making to be modelled, we need to introduce a new type of model.

5.2 Background

In this section, we introduce concurrent stochastic games, where property verification uses methods from game theory.

5.2.1 Concurrent stochastic games

As discussed in Example 5.1, the addition of concurrency in 26.2 can alter the strategies players adopt. In order to account for this behaviour, we introduce *concurrent stochastic games*, as described in [16].

Definition 5.2. A concurrent stochastic game (CSG) is a tuple $\mathscr{G}=(N, S, \bar{s}, A, Act, \delta, AP, L)$ where:

- $N = \{1, \ldots, n\}$ is a finite set of players;
- S is a finite set of states and $\bar{s}$ is the initial state;
- $A = (A_1 \cup \{\bot\}) \times \cdots \times (A_n \cup \{\bot\})$ is the set of actions, where A_i is a finite set of actions available to some player i, and $\bot$ represents an idle action which is disjoint from the actions of all players;
- $Act : S \to 2^{\cup_{i=1}^{n} A_i}$ is an action assignment function, where $Act(s)$ denotes the available set of actions in state s;
- $\delta : S \times A \to Dist(S)$ is a partial transition function;
- AP is a set of atomic propositions and $L : S \to 2^{AP}$ labels each state with a subset of atomic propositions holding at that state.

This definition has a lot in common with Definition 3.1 — the main difference is that players simultaneously choose actions, and the transition between states depends on the combination of actions chosen by all players. Indeed, an MDP can be considered as a CSG with only one player.

And as in the previous case studies, adversaries are defined very similarly, though we must consider adversaries for all players. We also remark that these strategies may be probabilistic, such as in the game of rock-paper-scissors, where the optimal strategy is to choose each symbol with probability $\frac{1}{3}$.

Definition 5.3. For a CSG $\mathscr{G}$, a strategy profile σ is a tuple $(\sigma_1, \ldots, \sigma_n)$ of strategies for each player. A strategy σ_i for some player i maps each finite path π to $Dist(A_i)$, such that if $\sigma_i(\pi)(a_i) > 0$, then a_i is an available action for player i at the last state of π.

We can also augment CSGs with reward structures as with previous model types, with action rewards and state rewards.

We now discuss computing reachability probabilities for CSGs (the case for reward properties follows similarly). These methods are somewhat similar in spirit to those for DTMCs and MDPs, but the computation of specific values requires a new class of techniques, which also allow players to form coalitions, and collaborate or compete with different players.

5.2.2 Matrix games

We first discuss some preliminaries from game theory, specifically the idea of matrix games. We can represent a simplified game as a matrix, as in the following definition.

Definition 5.4. A *matrix game* Z is the $l \times m$ matrix such that $A_1 = a_1, \ldots, a_l$ and $A_2 = b_1, \ldots, b_m$ where $Z_{i,j}$ represents the *utility* of the game for player 1 if player 1 performs action a_i and player 2 performs action b_j.

This game is simpler than the games we have considered in these case studies, since only one action is taken before the end of the game by each player. In addition we only consider two-player zero sum games, which are games where the sum of utilities for each player sum to 0. This allows us to model games with one winner and one loser, by setting a player's utility to -1 if they lose and 1 if they win.

We are interested in obtaining the *value* of a matrix game, which represents the maximum utility player 1 can ensure it obtains when playing the game and conversely is the minimum utility player 2 can ensure. More specifically, we say that for some matrix game Z, $val(Z) = v^*$ if player 1 has an optimal strategy such that the utility of the game is always at least v^*, and conversely if player 2 has an optimal strategy such that the utility of the game is at least $-v^*$. Every matrix game has a value, a consequence of the minimax theorem proved by von Neumann in [22]. This value can be obtained as the solution to a linear programming problem, specifically maximising v subject to the following constraints:

$$\begin{aligned} v \leq p_1 \cdot Z_{1,j} + \cdots + p_l \cdot Z_{l,j} \quad & \text{for } 1 \leq j \leq m \\ p_i \geq 0 \quad & \text{for } 1 \leq i \leq l \\ p_1 + \cdots + p_l = 1 \quad & \end{aligned}$$

The solution $(p_1, \ldots, p_l)$ yields an optimal strategy for player 1, and the analogous problem for player 2, which involves minimising v, yields an optimal strategy for player 2.

5.2.3 Property verification for CSGs

Computing reachability properties and reward-based properties for CSGs uses a similar approach to Section 3.2.2, where a sequence is defined based on the value of the required property after n steps, then approximating the actual result with large enough n. This process is described further in [16]. However, there are two key differences compared to property verification for MDPs. Firstly, rather than taking the minimum or maximum choice of each possible action, the value of an associated matrix game must be considered. Secondly, the properties have slightly different definitions, as shown below from [17]:

Definition 5.5. Given a coalition of players C in a CSG $\mathscr{G}$, with Σ^1 and Σ^2 representing the set of strategies for players in and out of the coalition respectively, for a given set of target states T, the maximum probability coalition C can ensure of reaching T is given by:

$$\mathrm{P}^{C}_{\mathscr{G}}(T) = \sup_{\sigma_1 \in \Sigma^1} \inf_{\sigma_2 \in \Sigma^2} \mathrm{P}^{\sigma_1, \sigma_2}_{\mathscr{G}}(T),$$

where $\mathrm{P}^{\sigma_1,\sigma_2}_{\mathscr{G}}(T)$ is the probability of reaching T under strategies σ_1 and σ_2.

We note that the players in the coalition aim to maximise the result while the players outside of the coalition aim to minimise the result. This allows for collaboration to be modelled using CSGs, although for the purposes of this case study we only consider cases where the sets of coalition and non-coalition players both have just one player.

We now focus on 26.2 in more detail, comparing pre-defined strategies to more complex optimal strategies.

5.3 Results and analysis

To begin with, we discuss a few constraints required to ensure the size of the model for 26.2 remains feasible to perform model checking. First, rather than allowing up to 10 ten-sided dice per player, each player can roll at most four 4-sided dice. The length of the board is also reduced from 262 spaces to 50 spaces. Even with these adjustments, a computing cluster with 600 gigabytes of RAM was required in order to perform probabilistic model checking. As a result of the size of this model, generating and analysing adversaries during this case study was infeasible.

The simplest possible strategy for 26.2 is to always roll the same number of dice. However, we remark that going bust when rolling 2 dice is impossible, hence there is no reason to roll only one dice. As a result, we only consider the strategies where 2, 3 and 4 dice are always rolled. In addition, we also consider a "hybrid" strategy, where the player rolls 3 dice if they are in the lead, or currently tied for the lead, and rolls 4 dice if they are behind. The rationale behind this approach is that rolling 3 dice while in the lead helps to consolidate a lead while minimising the probability of going bust and not moving for a turn, whereas rolling 4 dice from behind allows the player to catch up to the leader, albeit with the higher risk of going bust and staying even further behind.

First, we verify that, when both players apply the same strategy, the probability of either player winning is 0.5, up to numerical convergence. This is important for providing additional assurance that the model behaves as expected, and moreover that permuting players 1 and 2 makes no difference to the overall result of the game.

As a short aside, the probability of going bust after rolling a given number of dice can be easily calculated. Rolling 1 or 2 dice clearly means that the player cannot go bust, while rolling 3 dice means going bust with probability $\frac{1}{16} = 6.25\%$ since all three dice must have the same face value. The full derivation of the bust probability for 4 dice is omitted for space reasons, but the cases with 3 dice of the same value and 2 pairs of dice with the same value must be considered separately, giving a bust probability of $\frac{22}{64} = 34.375\%$.

Then, various combinations of strategies were compared, and their win rates are summarised in Table 5.1. From this table, a number of key insights about the game can be ascertained.

P1 strategy	P2 strategy	Probability of P1 winning game
4-dice	2-dice	0.8427
3-dice	4-dice	0.5630
hybrid	3-dice	0.5536
hybrid	4-dice	0.5685
optimal against 3-dice	3-dice	0.5608

Table 5.1: *The winrate of player 1 with various combinations of pre-defined and optimal strategies in the small version of 26.2.*

The first row in the table shows that the 4-dice strategy beats the 2-dice strategy handily. This is expected, since while the 4-dice strategy goes bust in around 34% of rolls, the expected value per roll is clearly double that of the 2-dice strategy. However the 4-dice strategy fares poorly against the 3-dice strategy, which wins with probability of approximately 0.56. This is in line with the theoretical bust probabilities which were computed earlier — the increase in bust probability outweighs the additional movement provided by rolling an extra die.

The third and fourth rows of the table show the effectiveness of the hybrid strategy against both fixed strategies. Against the 3-dice case, we expect both players to mostly stick close together during the game, since they are both rolling the same number of dice. However the variance of dice rolls means either the hybrid player could maintain a small lead, which the 3-dice player struggles to over come, or the hybrid place falls behind, where they then roll 4 dice in order to make up the difference and retake the lead. Against the 4-dice case, again for most of the game we expect both players to roll the same number of dice, but if the 4-dice player goes bust, the hybrid player can capitalise on this lead to start rolling 3 dice and maintain a consistent lead.

The most interesting result in this table is when considering the optimal strategy against the 3-dice strategy. Given an opponent who always rolls 3 dice, the optimal strategy generated by PRISM against this opponent wins with probability 0.5608, compared to the hybrid strategy which wins with probability 0.5536, a fairly small difference. Moreover, against the 4-die strategy, the 3-dice and hybrid strategies both perform very similarly. These raise a similar issue to Shut the Box, in that complex strategies are not sufficiently differentiable to simple strategies. The time spent to reason about and to derive a more optimal strategy is not justified by the performance improvement given as a result.

5.4 Evaluating the design of 26.2

Our analysis of a small variant 26.2 shows that the game contains similar flaws to Shut the Box, in that optimal strategies are too similar to simple strategies to justify the time required to learn those optimal strategies. In particular, without model checking, determining whether a strategy is optimal is infeasible for human players, so they may try many complex strategies, only to become frustrated and stop playing the game when they realise how similar their outcomes are to a very simple strategy.

There are a number of ways to resolve this issue in 26.2. Increasing the number of sides per dice would reduce the rate of increase of the bust probability, allowing for more dice to be rolled at once. This would lead to a greater variety of strategies, meaning the difference between effective and ineffective strategies should become larger. Also, while this was not included in the case study, the original version of 26.2 allows the player to collect *water tokens* when rolling a 1, which can be spent in order to perform various actions to mitigate the impact of luck, such as rerolling dice or reducing the value of a dice. Including these actions in the game would create an interesting comeback mechanic, since players who fall behind due to rolling low numbers will be able to make riskier choices in subsequent turns.

Along with showing the flaws of 26.2, this case study also shows a particular challenge for modelling board games. For most of the game, the number of spaces each player has moved is unimportant - the distance between each player is far more important. This allows human players to simplify their understanding of the game, and allows for the length of the board to change substantially without adding conceptual difficulty to players.

On the other hand, every possible combination of possible spaces for each player must be modelled as a separate state, meaning that the number of states in the game of 26.2 is $O(n^p)$, where n is the number of spaces on the board and p is the number of players. This makes larger variants of 26.2 infeasible to model. Hence, model checking is less suited for games which are long, but where behaviour is mostly invariant as the game progresses.

6 Evaluation

In this chapter, we briefly discuss multiple key tools which were developed during the process of conducting the case studies in project, in order to ensure these case studies were rigourously conducted.

During the project, multiple PRISM models were constructed of each case study. However, a key challenge when creating PRISM models is that PRISM offers little functionality for automatically generating PRISM model code. PRISM does offer module renaming, which allows additional modules to be defined via renaming variables, but not commands within modules.

In order to solve this problem, a preprocessor was implemented. This preprocessor allows for annotated PRISM models, containing a mixture of PRISM model code and preprocessor calls, surrounded by @ symbols. These calls provide parameters to the preprocessor, which allow for model code to be generated using other programming languages, in this case Python. This preprocessor has several key benefits when generating PRISM models.

- More complex operations, such as obtaining the probability of each possible total after rolling multiple dice, can be calculated using a more familiar programming language, rather than manually defining PRISM models which is prone to error.
- Parameterising models, such as altering the number of dice or the number of sides per dice, is far simpler to perform. PRISM allows for some parameterisation, such as varying the initial value of states, but the preprocessor allows for parameterisation that impacts the overall structure of the model.
- The annotated PRISM model provides a clear picture of the high-level structure of a PRISM model, rather than focusing on many individual lines of PRISM code.

In order to ensure that experiments would be repeatable, a series of Jupyter notebooks were created in order to run experiments. These notebooks contain commands for constructing PRISM models using the preprocessor, performing model checking via the command line to obtain experiment results, storing the ensuring data logs, and creating visualisations using these logs. Beyond ensuring the replicability of experiments, this also allows for performance improvements, since the results of model checking can be computed once, then the resulting data can be used to produce a number of visualisations. This is in contrast to the graphical user interface provided with PRISM, which requires an experiment to be performed every time a new visualisation is generated.

Alongside these tools to improve rigour, the process of probabilistic model checking is inherently rigorous. Other methods for analysing stochastic games, such as statistical model checking or Monte Carlo methods, generate numerous random samples in order to provide an estimate, or at best a probabilistic guarantee, of the required result. This can be unreliable, especially for rare properties, and requires quantifying the uncertainty of the result. By contrast, probabilistic model checking directly considers the probability of each transition, and operates an a deterministic manner. Hence, repeating probabilistic model checking will lead to the same result.

7 Conclusion

The results presented demonstrate that model checking can be an effective tool for answering balance questions about stochastic games. The games presented in these case studies are simple, but even these simple games provide interesting examples of various issues in game design. Shut the Box and 26.2 are both flawed games because optimal strategies are not sufficiently differentiable from simple strategies, while Liar's Dice is flawed because the early rounds of the game have a disproportionate influence on the overall outcome of the game, leading to games which feel artificially long without the later rounds of the game feeling meaningful. Of particular note is the variety of games presented here, with Liar's Dice including hidden information and 26.2 employing simultaneous action selection. This shows the potential of probabilistic model checking for analysing games beyond turn-based perfect information games, which current research primarily focuses on.

While this dissertation has considered numerous different possibilities for analysing games using model checking, a number of further questions are still to be considered.

- When generating optimal strategies for games, an important factor in the viability of a strategy is its complexity. As discussed in Section 3.4, players are unlikely to employ complex strategies if a simpler strategy is similarly effective. However, in general the complexity of a strategy is poorly quantified. Future work in model checking could involve defining a measure for the complexity of an adversary, then penalise strategies which are too complex when computing optimal adversaries. While this will likely lead to less effective strategies, the strategies generated in this manner will be more understandable for human players.
- The key bottleneck for further analysis of games is the model size, making full computation infeasible for many realistic games, and improvements in this area would lead to a substantial increase in the viability of model checking for determining game balance. For instance, the game tree construction presented in Section 4.3.1 could potentially be automated, allowing for subgames to be generated and considered for many types of models.
- Another potential avenue for future research is improving the visualisation of generated strategies. In particular, comparing two strategies is challenging, since their visualisations are large and differences in actions are unclear. Improved tool support in this area would be especially useful for designers who are less familiar with formal verification, allowing for optimal strategies to be more easily considered and evaluated.
- As briefly mentioned in Section 5.4, extending probabilistic model checking to partially observable stochastic games would allow for similar techniques to be applied to games where all players have different sets of hidden information, rather than just one player having hidden information, which would allow the techniques presented throughout this dissertation to be employed in a much wider class of games.

A | Conditional probability of board coverings in Shut the Box

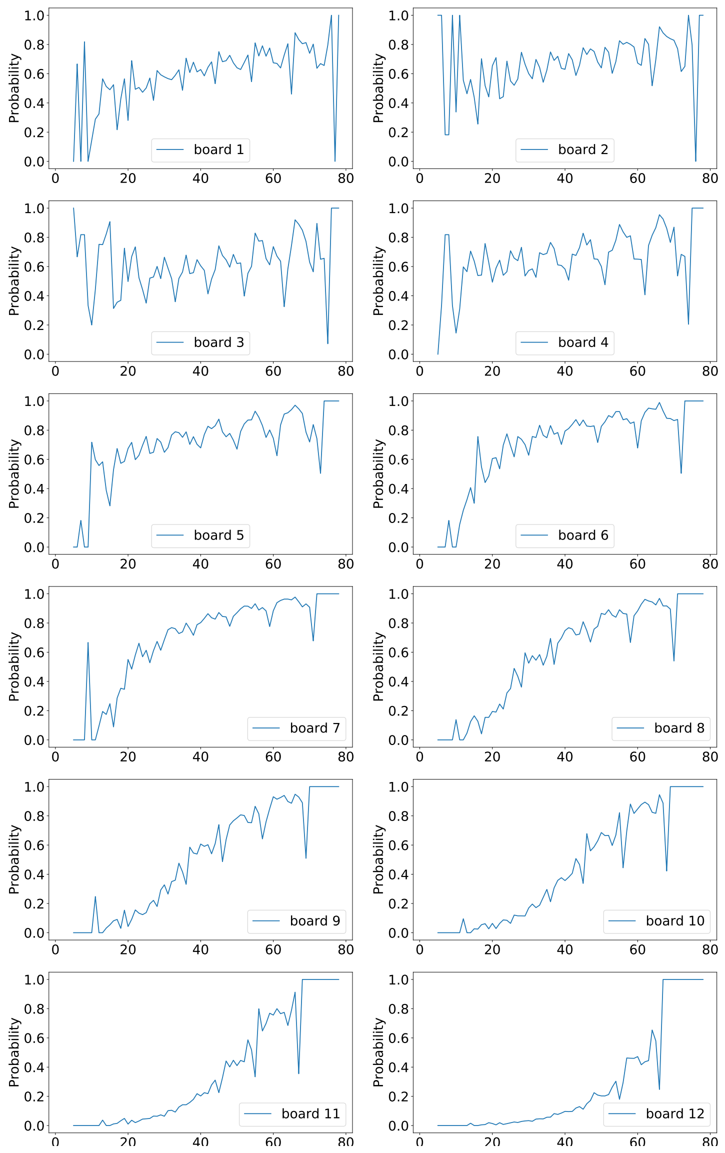

Figure A.1: *The conditional probability of each board being covered in Shut the Box given the final score, in the standard variant under the high-board strategy.*

Figure A.1 shows the probability of each board being covered in Shut the Box, given a particular final score. Every board has a positive correlation with the final score – this is as expected, since higher scores typically cover more boards. However, this positive correlation is much stronger for higher value boards, suggesting that high value boards are strong indicators for high scores in Shut the Box.

7 Bibliography

[1] PRISM Manual | Main / Welcome, . URL http://www.prismmodelchecker.org/manual/.

[2] The Snowball Effect (And How to Avoid It) in Game Design, . URL https://gamedevelopment.tutsplus.com/articles/the-snowball-effect-and-how-to-avoid-it-in-game-design--cms-21892.

[3] D. Bertsekas and H. Yu. Approximate solution methods for partially observable markov and semi-markov decision processes. *undefined*, 2006. URL /paper/Approximate-solution-methods-for-partially-markov-Bertsekas-Yu/12ecec6e8443da78e922390b552b343406dc3079.

[4] K. Chatterjee and T. A. Henzinger. Value Iteration. In O. Grumberg and H. Veith, editors, *25 Years of Model Checking: History, Achievements, Perspectives*, Lecture Notes in Computer Science, pages 107–138. Springer, Berlin, Heidelberg, 2008. ISBN 978-3-540-69850-0. doi: 10.1007/978-3-540-69850-0_7. URL https://doi.org/10.1007/978-3-540-69850-0_7.

[5] J. W. Demmel. *Applied Numerical Linear Algebra*. Other Titles in Applied Mathematics. Society for Industrial and Applied Mathematics, Jan. 1997. ISBN 978-0-89871-389-3. doi: 10.1137/1.9781611971446. URL https://epubs.siam.org/doi/book/10.1137/1.9781611971446.

[6] V. Forejt, M. Kwiatkowska, G. Norman, and D. Parker. Automated verification techniques for probabilistic systems. In *International school on formal methods for the design of computer, communication and software systems*, pages 53–113. Springer, 2011.

[7] S. Haddad and B. Monmege. Interval Iteration Algorithm for MDPs and IMDPs. *Theoretical Computer Science*, 735:111 – 131, July 2018. doi: 10.1016/j.tcs.2016.12.003. URL https://hal.archives-ouvertes.fr/hal-01809094. Publisher: Elsevier.

[8] E. A. Hansen, D. S. Bernstein, and S. Zilberstein. Dynamic Programming for Partially Observable Stochastic Games. page 7.

[9] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, Sept. 1994. ISSN 1433-299X. doi: 10.1007/BF01211866. URL https://doi.org/10.1007/BF01211866.

[10] V. Hom and J. Marks. Automatic design of balanced board games. In *Proceedings of the Third AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, AIIDE'07, pages 25–30, Stanford, California, June 2007. AAAI Press.

[11] A. Jaffe, A. Miller, E. Andersen, Y.-E. Liu, A. Karlin, and Z. Popović. Evaluating competitive game balance with restricted play. In *Proceedings of the Eighth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, AIIDE'12, pages 26–31, Stanford, California, USA, Oct. 2012. AAAI Press.

[12] W. Kavanagh and A. Miller. Gameplay Analysis of Multiplayer Games with Verified Action-Costs. *The Computer Games Journal*, pages 1–22, 2020. Publisher: Springer.

[13] W. Kavanagh, A. Miller, G. Norman, and O. Andrei. Balancing Turn-Based Games with Chained Strategy Generation. 2019. doi: 10.1109/tg.2019.2943227.

[14] M. Kwiatkowska, G. Norman, and D. Parker. Stochastic model checking. In *International School on Formal Methods for the Design of Computer, Communication and Software Systems*, pages 220–270. Springer, 2007.

[15] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *International conference on computer aided verification*, pages 585–591. Springer, 2011.

[16] M. Kwiatkowska, G. Norman, D. Parker, and G. Santos. Automated verification of concurrent stochastic games. In *International Conference on Quantitative Evaluation of Systems*, pages 223–239. Springer, 2018.

[17] M. Kwiatkowska, G. Norman, and D. Parker. Verification and Control of Turn-Based Probabilistic Real-Time Games. In M. S. Alvim, K. Chatzikokolakis, C. Olarte, and F. Valencia, editors, *The Art of Modelling Computational Systems: A Journey from Logic and Concurrency to Security and Privacy: Essays Dedicated to Catuscia Palamidessi on the Occasion of Her 60th Birthday*, Lecture Notes in Computer Science, pages 379–396. Springer International Publishing, Cham, 2019. ISBN 978-3-030-31175-9. doi: 10.1007/978-3-030-31175-9_22. URL `https://doi.org/10.1007/978-3-030-31175-9_22`.

[18] W. S. Lovejoy. Computationally Feasible Bounds for Partially Observed Markov Decision Processes. *Operations Research*, 39(1):162–175, 1991. ISSN 0030-364X. URL `https://www.jstor.org/stable/171496`. Publisher: INFORMS.

[19] O. Madani, S. Hanks, and A. Condon. On the undecidability of probabilistic planning and related stochastic optimization problems. *Artificial Intelligence*, 147(1):5–34, July 2003. ISSN 0004-3702. doi: 10.1016/S0004-3702(02)00378-8. URL `https://www.sciencedirect.com/science/article/pii/S0004370202003788`.

[20] P. Milazzo, G. Pardini, D. Sestini, and P. Bove. Case Studies of Application of Probabilistic and Statistical Model Checking in Game Design. In *2015 IEEE/ACM 4th International Workshop on Games and Software Engineering*, pages 29–35, May 2015. doi: 10.1109/GAS.2015.13.

[21] G. Norman, D. Parker, and X. Zou. Verification and control of partially observable probabilistic systems. *Real-Time Systems*, 53(3):354–402, May 2017. ISSN 1573-1383. doi: 10.1007/s11241-017-9269-4. URL `https://doi.org/10.1007/s11241-017-9269-4`.

[22] J. v. Neumann. Zur Theorie der Gesellschaftsspiele. *Mathematische Annalen*, 100(1):295–320, Dec. 1928. ISSN 1432-1807. doi: 10.1007/BF01448847. URL `https://doi.org/10.1007/BF01448847`.

[23] R. S. . R. S. a. G. Wikipedia. Deutsch: Shut the Box, July 2006. URL `https://commons.wikimedia.org/wiki/File:Shut_the_box.jpg`.