Modeling and analysis of the board game Mythic Battles: Pantheon through Markov Decision Processes

BRAM OTTE, University of Twente, The Netherlands

Supervisor: Milan Lopuhaä-Zwakenberg

Markov Decision Processes (MDP) have various use cases, but there is a limited amount of clear examples to test the tools used to model them on. We will be modeling and analysing the board game Mythic Battles: Pantheon. Board games Fit these models particularly well and Mythic Battles is particularly interesting due to its complexity. We model a small part of the game namely the normal attack using the probabilistic model checker PRISM.

In turn, we use PRISM to generate optimal strategies, that optimize for different objectives. Additionally, we will consider some naive strategies. Then we analyse those strategies by comparing their general performance and compare a few example cases where they differ.

Through the modeling we find that it's feasible to model a part of mythic battles and there is still room to expand the model before it becomes too big to handle. Although we do hit some limits in the tools and find workarounds to them. In the analysis we find the strategy for maximal expected damage is often equivalent to a naive strategy of re-throwing every dice.

Additional Key Words and Phrases: PRISM, PRISM-games, board games, Markov Decision Process, Mythic Battles: Pantheon

1 INTRODUCTION

MDPs are probabilistic models which also model the different choices that can be made. These models can be represented as a nondeterministic finite state machine where each state can have different actions associated with it each having transition with probabilities of going to a next state. In figure 1 we can see the states represented by S0 through S2 and the actions reprented by a0 and a1. MDPs are interesting because they can be used to model many things and have been used in a variety of fields from robotics to biology [4].

There has been limited research on modeling board games using MDP, further more there is a limited number of examples to test the tools for working with these models on. Board games fit these models nicely because they are turn-based and have very clear rules and probabilities which makes it easier to check whether the models are accurate.

To get useful insights out of MDPs we use so-called properties which define an objective and a metric to prompt the model.

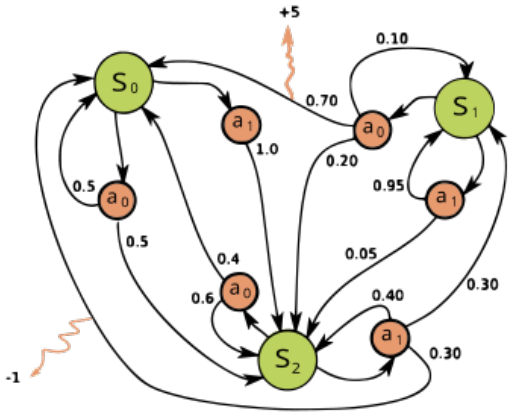
As these models become more complex they very quickly become infeasible to manually determine optimal strategies for, thus we use computer programs to evaluate the properties, one of such programs is PRISM [5]. PRISM also defines its own language for describing these models and properties. MDPs primarily grow based on the possible states of the model, since incorporating more aspects into a model tends to have a multiplicative impact on the number of states, these models tend to increase exponentially in size. For example if your model includes the location of a board piece, and you also add a health total to it these two things can change independently of each other thus resulting in the number of states being the number of positions times the number of possible health values. So the challenge in analysing these models is to have to analyse as little states as possible, for example by having fewer states in the first place or by combining states.

The game Mythic Battles: Pantheon is particularly interesting because of how many choices the player gets to make. Mythic battles is a complex game with many moving pieces, but some of its mechanics are similar to RISK, but its dice mechanic involves more choices as you can discard and or rethrow some dice. As Mythic Battles is such a complicated game we will focus on this dice mechanic when performing a normal attack.

To guide our research we have the following high level research goals:

(1) What can we learn about the game by modeling it?
   - What types of strategies are encouraged by the game design?
(2) How can we effectively model a part of Mythic Battles: Pantheon using PRISM?
(3) What can we learn about MDPs and the tools used to model them?

Fig. 1. Markov Decision Processes - By waldoalvarez - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=59364518



2 RELATED WORK

A variety of board games have already been modeled using similar methods, some early examples are Monopoly [1], RISK [8] and blackjack [13]. Some more recent examples are Snakes and ladders [10], Hobbit Adventure and Incan Gold [7]. Notably Hobbit Adventure and RISK have some similarities with Mythic Battles: Pantheon. In these games you roll a set of dice repeatedly in order to defeat

TScIT 42, January 31, 2025, Enschede, The Netherlands

enemy units. In Tan's analysis of RISK [12] he answers for different initial army sizes what the probability is of capturing a territory and also what the expected losses are. In Mocanu's analysis of Hobbit Adventure [7] he answers given optimal strategy what the maximal probability is to inflict different minimum damage values, Additionally he determines the maximum expected value and based on the optimal strategy what dice the player is encouraged to re-roll. His analysis of Incan Gold is interestingm because when he tries to add more players to the model it became infeasible for him to analyse. This is because the number of states explodes when adding too many things into the model. In order to evaluate bigger models we either need exponentially better hardware, have less resource intensive tools or discard, combine or approximate large portions of the state of the model since these models mostly grow with the number of states.

There are several tools for working with these models such as PRISM [5] and Storm [2]. Both of these tools are very capable and are in active development, there are also more tools but these generally perform worse or don't support the same features. Storm is generally a lot faster than PRISM but its relatively new tool and does not directly support SMG's which PRISM does support via PRISM-games [6].

These probabilistic model checking tools have a wide range of applications such as in energy grid management [3], security protocols, robotics and even biology [4]. Another interesting example is the use of these models to generate Digital circuits from specification like Kwiatkowska et al.'s use of PRISM to generate transistor logic [11]. This is in no way a complete list, in general applications where uncertainty and decision-making needs to be formally modelled these tools can be very helpful.

3 PRISM

PRISM is a probabilistic model checking tool, which allows us to create models such as MDPs in the PRISM modeling language and analyse this model by prompting prism with queries also called properties. Additionally, PRISM also allows us to generate strategies that optimize for maximizing or minimizing these queries.

The PRISM modeling language allows us to define the behavior of the model over many states using conditional guards instead of having to enumerate all states, actions and transitions of the MDP. This becomes very important as models quickly gain a lot of states when more things are added and it also makes it easier to understand the model.

A model represented in the PRISM modeling language consists of a set of modules, variables, constants and rewards. Each module can contain its own variables and commands. Figure 2 shows a small example of the different parts that make a module. These commands represent the behavior of the model while the variables represent the state. Each command consists of a guard, which is a condition determining when the action should be executed, followed by a set of actions with their associated probability, each action consists of updates to the variables these updates can be arbitrary expressions.

Modules by default represent the possibility of updates happening in arbitrary orders, by adding action labels. Commands with the same label are synced up and will update simultaneously. For the purpose of our research we will always label the guards since the steps of the game need to happen in the same order every time. Modules can be used to model sepperate espects of a model sepperately instead of having to make sure that each guard handles all the espects. Additionally, a module can be used as a template for another model allowing to easily repeat logic over multiple entities.

If there are multiple actions that can be taken from a state this is simply represented by having overlapping guards within the same module, the presence of such choices is also called nondeterminism. Guards are overlapping if they are true for the same state for example the guards $x > 10$ and $x > 11$ overlap when $x > 11$.

A strategy is a specific resolution of the nondeterminism, in case of prism this results in picking one of the actions when the quards overlap. A strategy can be created based on different metrics or queries, these can be defined via PRISM properties.

Fig. 2. PRISM command

```
module Example

variable: [min_value..max_value] = initial_value;

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

An extension to MDPs are Turn-base Stochastic Multiplayer Games (SMG) which allow for multiple players to be modeled that might have opposing goals.

4 MYTHIC BATTLES: PANTHEON

First we will introduce the game rules with an example of a normal attack then we will formulate research questions followed by a brief overview of our approach and explanation of the model, finally we will analyse the model and answer the research questions.

4.1 Game Rules

Mythic battles: Pantheon is a miniatures war-game, themed after Greek mythology. In this game each player chooses a divinity and units for its army, the goal is to defeat all enemy armies (there is a second win condition, but we will not cover this). The game is played with 1 to 4 players. At the start of the game, players have

to choose a board and the units for their army. A player's army comprises one divinity and any number of other units depending on how they spend their recruitment points. These units have different stats (offense, defense, range, speed and available powers) which can change depending on their vitality. For example most units' stats will generally decrease, and a berserker unit might gain offense when damaged. The stats of a unit are tracked on their dashboard which has a slider showing the current stats.

Each round the player can preform a limited number of actions such as moving around their piece or attacking. Initially we will be modeling the normal attack, which has the following 4 faces:

(1) Effective value calculation:
The effective offense of the attacker and defense of the defender are determined.
(2) First assault:
The attacker throws a number of dice equal to their effective offense, each die is a 6 sided die with value 0 through 5 and each die that lands on a 0 is immediately discarded.
The attacker can choose to discard more dice, for each die they discard they can increase the value of another (not yet discarded) die by one, we will also refer to this process as boosting a die. Each die that matches or exceeds the defense of the defender is placed on the attacker's dashboard and will inflict a wound.
(3) Second assault:
For each die that has a value of 5, the attacker can choose to re-roll it. If the re-rolled die lands on a value of 0 it, including its previous value, is discarded. Otherwise, the value the die landed on will be added on-top of its previous value.
Again the attacker can choose to discard more dice, for each die they discard they can increase the value of another (not yet discarded) die. Each die that matches or exceeds the defense of the defender is placed on the attacker's dashboard and will inflict a wound.
(4) Wounds:
For each wound inflicted the defender will lose 1 vitality point, this will generally decrease their stats and when a unit's vitality reaches 0 the unit is destroyed.

If the attack was not ranged, the attack was not a retaliation, the defender survived, has not yet retaliated in this turn and discards one activation card, they can retaliate. When a unit retaliates they can preform a normal attack on their attacker as a response.

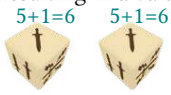
4.1.1 Example. For example: Hoplites attacks Leonidas, Hoplites' has an effective offense of 4 and Leonidas an effective defense of 7 and a vitality of 6.

In their first assault the attacker throws 4 dice: a 5, 4, 2 and 0.

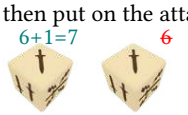
The 0 die is discarded leaving them with a 3, 4 and 5.

They choose to discard the 3 to bring the 4 up to a 5.

For their second assault they choose to re-roll 2 dice, both land on 1 resulting in a value of 6.



In order to inflict a wound they need dice with a value of at least 7, so they choose to discard one of the sixes to make a seven which is then put on the attacker's dashboard and will inflict a wound.



One wound is inflicted on Leonidas decreasing their vitality from 6 to 5.

4.2 Research Questions

Before we go into modeling the game we devise the following set of research questions:

(1) What strategies maximize the expected damage?
(2) For different offense and defense values what is that expected damage?
(3) What is the strategy that maximizes the probability of reaching a minimum amount of damage?
(4) For different offense, defense and minimum damage what is the probability of achieving at least a minimum amount damage?
(5) What can we take away from these strategies while playing the game?
  (a) When is the player encouraged to discard a die?
  (b) When is the player encouraged to take a die into second assault?

4.3 Model

[1] The model consists of 2 modules, a Time module responsible for ensuring the different steps of the model are executed in the right order and a Dice module responsible for preforming the steps and calculating the results of the dice. The model will preform the attack in different steps corresponding to the labels: first assault, first count, first boost, first discard, second assault, second count and second boost.

[1] For full source code refer to our GitHub [9].

Fig. 3. Used module structure

```
mdp

const int max_attack = 10;
const int attack;
const int defense;

module Dice
// Buckets of dice with the same value 1 through 10
d1: [0..max_attack] init 0;
...
d10: [0..max_attack] init 0;
...
endmodule

module Time
// The current step of the process
r: [0..8] init 0;
// How many times to repeat an operation dice for a step
// For example how many dice to throw
i: [0..max_attack] init attack;
...
endmodule
```

One of the more important things to keep in mind when modeling is how you represent its state. There are multiple ways to represent equivalent states, like we represent the values of the dice as buckets but you could also model each die as its own module making their state multiply, while with the buckets the order the dice are thrown in is discarded from the state. The buckets result in having $\binom{5+N}{5}$ states while having each die modeled individually results in 6^N states for N dice. For 8 dice this is 1287 states for the buckets and 1 679 616 states which is significantly more, and the difference will only become larger as more dice are modelled. However, certain things will not affect the number of states as PRISM will only process states that are reachable from the specified initial state. For example immediately handling wounds in the assaults or first putting them in a dice bucket does not increase the number of states eventhough it adds an extra variable to the state these states are corrilated such that no remaining state is left over. It is however important to be careful not to have too much state that does not contribute to the model.

Fig. 4. Multiple Dice modules

```
module Dice
d1: [0..6] init 0;
...
endmodule

module Dice2 = Dice1 [d1 = d2] endmodule
...
```

In the first assault 5 a number of dice with the value 0 through 5 are rolled and are put into buckets corresponding to their value. We group the dice together in buckets in order to more easily get the lowest die or highest die. By doing this we do not only make the following steps easier, but we also greatly reduce the state space by not taking into account the order that the dice are in.

Fig. 5. First assault

```
module Dice
...
[first_assault] r=0 & d1+d2+d3+d4+d5+wounds < max_attack
  -> 1/6: true
  +  1/6: (d1' = d1 + 1)
  +  ...
  +  1/6: (d5' = d5 + 1);
...
endmodule

module Time
r: [0..8] init 0;
i: [0..max_attack] init attack;

[first_assault] r=0 & i >  1 -> (i'=i-1);
[first_assault] r=0 & i <= 1 -> (i'=attack) & (r'=1);
...
endmodule
```

Then in first count the number of non-zero dice are counted in order to know in the next step how many dice we can discard to boost into the second assault. Then in first boost the player gets to choose how many dice to take into the second assault put these aside into their own bucket and count the number of dice that need to be discarded ensuring we never discard more die than we have left. Here we assume we always want to take the dice with the highest value into the second assault.

Fig. 6. First boost

```
[first_count] r=1
  -> (points' = min(max_attack, d1+d2+d3+d4+d5));

// Choose to take highest die into second assault
[first_boost] r=2 & defense > 5 & points >= 5+1-5
  & d5 > 0
    -> (d5'=d5-1)
    & (points' = points - (5+1-5))
    & (reroll'=min(max_attack, reroll+1));
...

// Choose to not take die into second assault
[first_boost] r=2 -> true;
```

Then in first discard the appropriate number of dice are discarded to make this possible Here we assume we always want to discard the dice with the lowest values first.

Fig. 7. Discard

```
[first_discard] r=3 & discard>0
  & d1>0
    -> (d1'=d1-1) & (discard'=discard-1);
...

[first_discard] r=3 & discard<=0 -> true;
```

Then in second assault the dice selected during first boost are rerolled and put into their corresponding value bucket.

Fig. 8. Second assault

```
[second_assault] r=4 & reroll > 0
  ->1/6: (reroll' = reroll-1)
  + ...
  + 1/6: (reroll' = reroll-1) & (d10' = d10+1);

[second_count] r=5 -> (points' = min(max_attack,
  d1+d2+d3+d4+d5+d6+d7+d8+d9+d10));

[second_boost] r=6 & wounds < max_attack
  & d9 > 0
  & points >= 1+max(0, defense-9)
    -> (d9' = d9 -1)
    & (points'=points-(1+max(0, defense-9)))
    & (wounds'=wounds+1);
```

Finally, in the second boost the number of inflicted wounds is calculated. Here we assume we always want to boost dice with the highest value first and discard dice with the lowest value.

Additionally, we define two rewards one that counts the number of wounds (at the end end when r=7) one that count the number of dice we take into the second assault (after boosting r=3). We then make the following properties in order to answer our research questions:

Fig. 9. Rewards

```
rewards "wounds"
r = 7: wounds;
endrewards

rewards "reroll"
r=3 & die_i=1: reroll;
endrewards
```

Fig. 10. Properties

```
R{"reroll"}max=? [F r=8];
R{"reroll"}min=? [F r=8];
R{"wounds"}max=? [F r=8];
R{"wounds"}min=? [F r=8];
Pmax=? [F r=8 & wounds>=1];
...
Pmax=? [F r=8 & wounds>=10];
```

4.4 Results

For our first two research questions: "What strategies maximize the expected damage?" and "For different offense and defense values what is that expected damage?", we generate strategies based on the max wounds property 10. This results in a very large file containing the action the strategy will take for each state. Their sheer size makes it difficult to analyse. To address this instead of looking at all the actions of the strategy we look at how it preforms compared to some naive strategies. In figure 11 we plot the optimal strategy (rmax wounds) along with a strategy that takes all possible dice into the second assault (rmax reroll) and one which takes none (rmin reroll). Additionally, we also plot the worst possible strategy. From this we can see that on average taking all possible dice into the second assault preforms very similarly to the optimal strategy. Similarly, not taking any dice into the second assault is close to the worst strategy.

However, the strategies do differ in some key ways, specially there are situations where re-throwing every die is not optimal and where not re-throwing anything is not the worst. This also goes some ways into answering our fifth research question of what actions are encouraged by the game and will thus be used by an optimimal strategy. In order to generate examples of this we created a program which goes through every state after the first assault and compares the different strategies and how many dice are rethrown. One example that covers both cases is when the defender has 6 defense and 10 dice are thrown: 5 ones and 5 fives

The optimal number of dice to rethrow is 5 (expected damage (ED) 5.17) whereas the maximum is 6 (ED 5) and the worst is 4 (ED 4.33). Not rethrowing results in an expected damage of 5 since we can get a guarantied hit on all the fives by simply boosting them with one of the twos. Each dice we rethrow initially trades one guarantied wound and for a 5/6 chance of a wound and a left over 2. If we rethrow 5 dice we are left with enough twos to get a guarantied wound, so this will give us an extra wound compensating for the 5/6 chance and gives the highest expected damange. If we rethrow an extra dice we trade a guarantied wound for a 5/6 chance of a wound making rethrowing all dice not the optimimal strategy. Rethrowing fewer dice is more encouraged when a minimum amount of damage has to be reached as apposed to a maximal average damage. When the defender only has a limited number of vitality left this is most relivant.

In figure 12 we can see the expected wounds for every combination of offense and defense values.

For third and fourth research questions we want to know what the optimal strategy for achieving a minimum amount of damage and what the resulting probabilities are for achieving said minimum damage. We plot the probability of achieving a minimum number of wounds given optimal strategy for a fixed defense value and different attack values in figure 13. From this we can see that the probability first slowly decreases and then quickly decreases before leveling off again as it approaches zero. The higher the attack the more wounds can be inflicted before this probability drops off.

For the fifth research question we want to know when a player is encouraged to discard dice and when they are encouraged to boost a die. Which dice to discard and boost first can simply be answered with reasoning, as the player will always want to discard the lowest dice first to boost the highest dice since boosting a lower die doesn't give any additional benefit as any die that is lower than the defense won't inflict a wound. For this reason we also have the model assume this. If we led this up to player choice the results should be the same if we look at the optimal strategy, but this assumption does affect the other strategies as they now won't

Fig. 11. Strategy comparison naive and optimimal

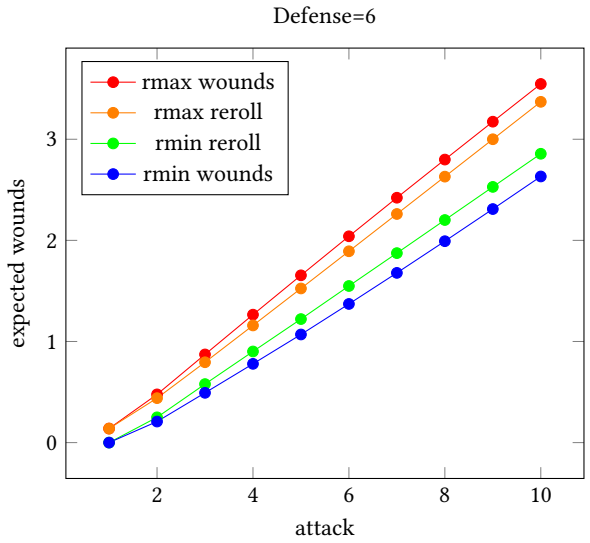


Fig. 13. Probability mininum wounds

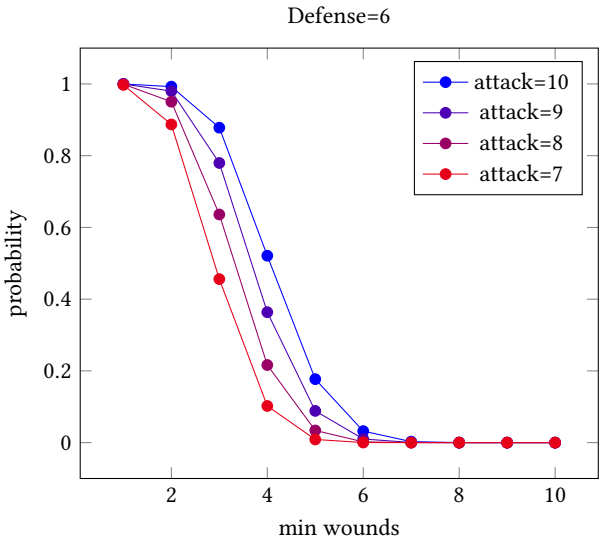


Fig. 12. Maximized Expected wounds for offense and defense 0 through 10

def → / off ↓	1	2	3	4	5	6	7	8	9	10
1	0.8	0.7	0.5	0.3	0.2	0.1	0.1	0.1	0.1	0.0
2	1.7	1.4	1.1	0.8	0.5	0.5	0.4	0.3	0.2	0.1
3	2.5	2.1	1.7	1.3	1.0	0.9	0.7	0.6	0.4	0.3
4	3.3	2.8	2.3	1.9	1.4	1.3	1.1	0.9	0.7	0.5
5	4.2	3.5	3.0	2.4	1.8	1.7	1.4	1.2	1.0	0.7
6	5.0	4.3	3.6	3.0	2.3	2.0	1.8	1.5	1.2	0.9
7	5.8	5.0	4.3	3.5	2.7	2.4	2.1	1.8	1.5	1.2
8	6.7	5.8	5.0	4.1	3.2	2.8	2.5	2.1	1.7	1.4
9	7.5	6.5	5.6	4.6	3.6	3.2	2.8	2.4	2.0	1.6
10	8.3	7.3	6.3	5.2	4.0	3.5	3.1	2.7	2.3	1.8

be able to do this part suboptimally. This means that the decision the player makes comes down only to how many dice to rethrow. As discussed in the beginning of the results section, assuming the player want to deal the most possible damage on average the player is almost always encouraged to throw a lot of their dice but in cases can hold back some dice to get a guarantied wound in and achieve a higher expected damage. We will not cover all cases, but they only result in a slight advantage on average but the biggest advantage for defense 6 is $\frac{5}{6}$ of a wound, there are also some extreme cases with higher defense where you can gain 0.74 wounds over just 2.97 wounds when defense is 9. How common these cases, where the optimimal strategy is better than the naive one, differs quite a bit for each defense level, from 10% of the states for defense 7 to 73% of states for defense 10, assuming attack 10.

4.5 Modeling Experience

One problem we encountered due to size of the model, is that modeling using PRISM is quite error-prone. Namely, by default PRISM does not tell if guards cover all relevant states but instead assumes the state does not change and inserts transitions accordingly. This might result in steps being skipped or in the model getting stuck at intermediate state. This can be addressed passing PRISM the -nofxdl flag which prevents it from inserting these transitions and will will give an error with the states which are not covered by the model allowing us to add the necessary guards.

Secondly PRISM by default does not tell where nondeterminism occurs. Having unintended nondeterminism might result in strategies differing on steps that should be deterministic, for example when discarding dice the number of dice is determined in the previous step, if there exist unintended nondeterminism a strategy might not discard all the dice it is supposed to. This issue can be resolved by turning the model into an SMG as this allows us to specify which guards belong to which player and will give an error when nondeterminism occurs on a guard that is not assigned to a player. However, a disadvantage of using an SMG is that PRISM evaluates properties on it significantly slower, an order of magnitude slower in some cases.

Lastly we created an alternative implementation of the model in the form of a simulation and check if they roughly match. The simulation takes many samples of the first assault and then for each of these samples takes many samples for the different amounts of dice to take into the second assault and pick the highest average wounds and finally averages out all these samples. When creating such a simulation one has to be careful to not introduce sampling bias. For example if when picking the action we have a small sample count we will skew the results to a higher average wounds, this can be easily demonstrated if we have a single sample for each action its like trying the second assault multiple times and picking the best result. In the end the simulation and the model mostly matched but for larger number of dice they start to diverge more and more. This might be due to one of the implementations being incorrect, the

simulation having a sampling bias or the state space becoming too large. Looking at examples the simulations seems to have some odd behaviors while the PRISM model always seemed to give results we expected, and thus we will assume the PRISM model is the correct implementation.

5 CONCLUSION

In this paper we sought out to model and analyse a part of Mythic Battles: Pantheon via MDPs using PRISM. We did so in order to satisfy our research goals of: learning about the strategies of the game, how to effectively model MDPs using PRISM and what the limitations are of PRISM. For this we first looked at what other boardgames have been modeled and what kind of questions are interesting to ask. Then we took a part of Mythic Battles namely the normal attack and modeled it using the Prism modeling language. We then used PRISM to generate strategies optimizing for different goals. And then we analysed these strategies through PRISM which allowed us to analyse the performance of these strategies but slowed down significantly as we tried larger models and was difficult to say much in particular about how the strategies operated. Through our own program we were able to more quickly iterate on our queries and also investigate the operation of the strategies by finding examples of where the ste strategies differ. Finally, we reflected on our modeling experience.

In the end we were able to determine optimal strategies and that the strategy for maximising damage is very similar to a naive strategy of rerolling every possible dice but is superior in a select percentage of cases which might make a difference in some games. Along the way we learned about PRISM and the multiplicative nature of the state space of the models we can create with it, as well as things to look out for to minimize the state space. Such as representing the dice as buckets, so the order of the dice do not contribute to the state space.

Future work could explore adding multiple players to the model. Our preliminary research explored two units attacking each other back and forth, this did find that it results in a manageable model and that whether the second player can retaliate has a large influence on their chances of winning, but we did not have time to analyse this model further.

This model could be further analysed by comparing different strategies in this more complex model. The model could also be expanded further to include more of the game, for example include more units or player or include more mechanics, like the movement of pieces around the board. This would make the model larger and come closer to reaching the limits of PRISM especially when adding more players. This more complex model would be more interesting to optimize and experiment with. Another way of pushing PRISM would be to simply make the existing model bigger by including more dice.

In order to manage these bigger models we can look into different tools such as Storm, being a more recent effort written in cpp, it also supports the PRISM modeling language, but it is typically several times faster as Christian Hensel et al. [2], show in their paper with the QComp benchmarks. Additionally, we could look into improving the PRISM itself for example: when PRISM is used to generate strategies it will output every single state thus is not able to make use of more efficient solving techniques, having some more efficient way of generating and representing strategies could allow prism to generate and analyse strategies for bigger models.

ACKNOWLEDGMENTS

Special thanks to my supervisor Milan Lopuhaä-Zwakenberg, track chair Peter Lammich, Molly Waite and the people at the UT Writing Center, my study-advisor Rianne de Jonge and finally thanks to my family.

REFERENCES

[1] Robert B. Ash and Richard L. Bishop. 1972. Monopoly as a Markov Process. *Mathematics Magazine* (1972). https://doi.org/10.1080/0025570x.1972.11976187
[2] Christian Hensel, Sebastian Junges, Joost-Pieter Katoen, Tim Quatmann, and Matthias Volk. 2020. The Probabilistic Model Checker Storm. *arXiv: Software Engineering* (2020). https://doi.org/10.1007/s10009-021-00633-z
[3] Hanno Hildmann and Fabrice Saffre. 2011. Influence of variable supply and load flexibility on Demand-Side Management. *2011 8th International Conference on the European Energy Market (EEM)* (2011), 63–68. https://api.semanticscholar.org/CorpusID:24068628
[4] Joost-Pieter Katoen. 2016. The Probabilistic Model Checking Landscape. *Logic in Computer Science* (2016). https://doi.org/10.1145/2933575.2934574
[5] Marta Kwiatkowska, Gethin Norman, and David Parker. 2011. PRISM 4.0: verification of probabilistic real-time systems. *International Conference on Computer Aided Verification* (2011). https://doi.org/10.1007/978-3-642-22110-1_47
[6] Marta Kwiatkowska, David Parker, and Clemens Wiltsche. 2018. PRISM-games: verification and strategy synthesis for stochastic multi-player games with multiple objectives. *International Journal on Software Tools for Technology Transfer* 20 (2018). Issue 2. https://doi.org/10.1007/s10009-017-0476-z
[7] Daniel Mocanu. 2023. Modeling and Analyzing Board Games through Markov Decision Processes. https://drive.google.com/file/d/1XA_yJpUKsgMtFZWI0SdqG5m5z2AANec7/view
[8] Jason A. Osborne. 2003. Markov Chains for the RISK Board Game Revisited. *Mathematics Magazine* (2003). https://doi.org/10.1080/0025570x.2003.11953165
[9] Bram Otte. 2025. *Modeling Mythic Battles Pantheon, source code.* https://github.com/bramotte/prism-battles
[10] Vishva Sundarapandian Raani. 2021. Modeling and Analysis of Board Games. https://drive.google.com/file/d/1skXLTmwyCuteMh1fHWsjHP5HQpfppWCK
[11] Michael Raitza, Steffen Märcker, Jens Trommer, Andre Heinzig, Sascha Klüppelholz, Christel Baier, and Akash Kumar. 2020. Quantitative Characterization of Reconfigurable Transistor Logic Gates. *IEEE Access* 8 (2020), 112598–112614. https://api.semanticscholar.org/CorpusID:219719427
[12] Barış Tan. 1997. Markov chains and the RISK board game. *Mathematics Magazine* 70 (12 1997). https://doi.org/10.2307/2691171
[13] Michael B. Wakin and Christopher J. Rozell. 2004. A Markov Chain Analysis of Blackjack Strategy. https://api.semanticscholar.org/CorpusID:14857861