Eliminating the Impossible: A Procedurally Generated Murder Mystery

Henry Mohr
hmohr@haverford.edu

Haverford College
Haverford, PA, USA

Markus Eger and **Chris Martens**
meger@ncsu.edu, crmarten@ncsu.edu
Principles of Expressive Machines Lab
NC State University
Raleigh, NC, USA

Abstract

Games that involve solving puzzles or mysteries often suffer from low replayability once the solution is known. Procedural content generation has been used to aid with increasing variance in game play, leading to a higher replay value. However, games that feature information that must be obtained by the player present the additional challenge of tracking the player's knowledge and ensuring that it is possible for them to finish the game. In this paper we present a detective game with procedurally generated characters that pursue a range of goals, including one that commits a murder. The player's role is to solve the crime by collecting evidence from the actual game world as well as by interrogating the characters. Even though some of the characters might be inclined to hide evidence, our system guarantees that the player is always able to solve the murder case. To demonstrate the capabilities of our system we present a short example game that we have implemented using our techniques.

Introduction

Gregory: *"Is there any other point to which you would wish to draw my attention?"*
Sherlock Holmes: *"To the curious incident of the dog in the night-time."*
Gregory: *"The dog did nothing in the night-time."*
Sherlock Holmes: *"That was the curious incident."*
From: The Adventure of Silver Blaze

Detective stories in the tradition of Sir Arthur Conan Doyle's Sherlock Holmes are a cornerstone of today's entertainment culture, including TV Shows such as Sherlock or Monk, movies like Murder on the Orient Express or the long running Nancy Drew series of books. There are also video game adaption of several of these stories, as well as stand-alone detective video games, such as L.A. Noire. One limitation of these games, though, is that they always present the same story or a slight variation thereof, making it trivial for a player to solve the game after the first play-through. For example, in a faithful video game adaption of the Adventure of Silver Blaze, the curious incident of the dog in the night-time would be less curious for a player once they learned of its significance, namely that the dog recognized the culprit and was therefore silent.

One approach taken by several video games to improve variation and replayability is the procedural generation of content (Hendrikx et al. 2013). This can range from creating different maps the game is played on, variations of buildings shown in the game world, or procedurally generated characters with their own history (Adams 2017). The challenge in providing procedurally generated content for a game is to generate a wide enough range of outputs to be meaningfully different, while preventing undesirable output from occurring. For example, if a game generates a new maze for players to solve on every play-through, but the mazes are just minor variations on the same basic layout, they will not provide much of a challenge to the players. On the other hand, mazes that are literally unsolvable, because there is no path from the start to the goal, must also not be generated.

The genre of murder mystery stories lends itself nicely to procedural generation, because it follows a relatively fixed general structure, but allows for variety within that structure. In 1928 Willard Huntington Wright, using his pen name S.S. Van Dine, compiled a list of 20 rules that detective stories should follow (Van Dine 2015). Most of these rules are different aspects of the requirement that the murderer has to be found by logical deduction, rather than psychic means, trickery, or by accident. Owing to the time these rules were written in, the so-called Golden Age of Detective Fiction, they also recommend to avoid certain topics or techniques that were over-used at the time, such as having a dog not bark at night-time.

In this paper we present a detective game which utilizes a procedural generation approach to generate a new crime on every play through, and uses Dynamic Epistemic Logic to represent facts and clues found by the detective. Our approach provides a range of different experience, by generating a set of different characters with different motivations, one of which will commit the murder, but ensures that the resulting crime is always solvable by the player. We will first discuss some related work, including the underlying logical representation, then present our approach, followed by a brief discussion of an example domain which consists of a murder on a cruise ship. Finally, we will provide an outlook on how this work can be expanded upon in the future.

Related Work

Since the basis for detective games lies with detective stories, much of our work draws from prior work on story generation. It has been observed that plans and narratives share several properties, such as consisting of a temporal and causal ordering of actions, and planning can therefore be used to generate stories (Young 1999). In such an approach the author defines a goal for the story, and the planner will assign actions to characters to achieve this goal. However, Riedl and Young (2010) have noted that a pure planning approach does not account for the goals the individual characters have, and have presented an alternative planning approach where each character's actions have to serve a character motivation. Note that character motivations typically play an important role in detective stories as the motive of the murderer.

Another important aspect in detective stories is that of character beliefs. In the beginning of the story, the detective, like the reader, is unaware of the murderer's identity, but the murderer often knows that the detective is looking for them. Likewise, other characters may actually have observed some aspect of the crime, but may have something else to hide themselves. Ryan et al. 2015 describe virtual characters that build mental models of the world in a mutable way, i.e. they may misremember or forget parts of what they know, or they may (intentionally) lie to other agents. However, these lies are generated randomly and not in service of some goal the character may have.

Because of the relatively formulaic structure of murder mysteries, the idea of generating them has inspired a number of previous efforts. Green et al. (**?**) describe a system that generated murder mystery adventure games from Wikipedia articles about historical people. Stockdale (2003) presented ClueGen, a procedurally generated murder mystery game in which the player acts as a detective in a murder case. In this game, the focus lies on characters' relationships with each other, which informs how they share or hide information with the detective. Due to the uncertainty the player has about information provided by the non-player characters, the game provides the player with audio clues when a character is lying to enable solvability. In contrast, our approach focuses on generating physical clues about the murder, with some of the information related to them truthfully provided by non-player characters, guaranteeing the case to be solvable. As Eger and Martens (2017a) have described, Dynamic Epistemic Logic, which we will briefly discuss in the next section, can be used to keep track of character knowledge in stories, and what they have deduced about the story world. In our game, we utilize this approach to track which options the player has eliminated and to determine whether or not they have enough evidence to arrest the murderer.

Dynamic Epistemic Logic

To formalize the deduction process of the detective, we use Dynamic Epistemic Logic, the logic of changing knowledge and beliefs (Van Ditmarsch, van Der Hoek, and Kooi 2007). In Epistemic Logic, beliefs are usually represented as sets worlds that the different agents consider possible, called the *appearance* of the actual world to each agent. Dynamic Epistemic Logic adds an action language that describes how agents' beliefs change over time by introducing new worlds that they consider possible or eliminating those that they no longer consider possible. There are several differen flavors of Dynamic Epistemic Logic, of which we use one developed by Baltag (2002). It follows what Van Ditmarsch et al. call the dominant approach, in which actions are represented in a similar way to states, where each action also has an appearance to the different agents. In this particular flavor it is also possible to have actions that agents only suspect to happen, without any action actually occurring. Eger and Martens also described a macro system and runtime environment, Ostari, that greatly simplifies writing epistemic actions in Baltag's logic and is available freely (Eger and Martens 2017b). Our game is built on top of this system as described in the next section.

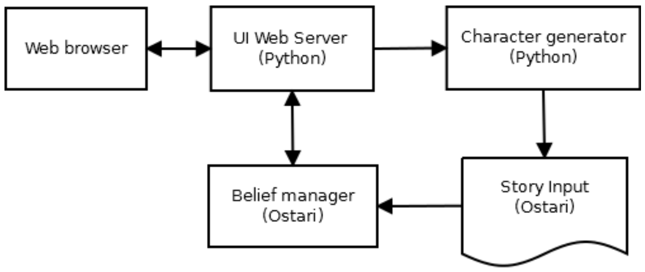


Figure 1: A diagram of the system architecture

Approach

We have built a system that generates playable murder mystery stories based on properties of the different characters and how the detective can obtain clues to narrow down their search. These clues can be found either by examining the environment, or by questioning the characters themselves. However, some characters may be hiding a secret that they have committed a different transgression, and will not share information that would reveal that transgression to the detective, unless the detective already knows about it. We track the knowledge of the detective by using Dynamic Epistemic Logic in Ostari, which allows us to determine when the player has actually collected enough evidence to have a case in court.

The game consists of several parts, shown in Figure 1. Players interact with the game using a web browser that connects to a Python web server that manages the game. Upon initiating a session, the characters will be generated by a Python script that generates an Ostari input file as described below. This file is then executed by Ostari, resulting in a murder and the corresponding clues, either in physical form in the game world, or as observed by the characters. During game play, Ostari manages the player's beliefs and is responsible for updating them with the information obtained by the investigative actions taken by the player.

Character Generation

Each game session in our game starts with newly generated characters produced by a Python module. This module uses a template file, which contains the Ostari code with the actions for the particular game domain, but also has several placeholder variables that will be filled in with the character information. A game domain defines character properties, each of which has a list of possible values. For example, a property `haircolor` may have values `blonde`, `black` and `brown`. For each character, we randomly draw from these lists of possible property values until we obtain a unique assignment of property values. This uniqueness constraint ensures that the murder is solvable, if each property value can be observed and the detective can determine what value the properties of the murderer have. Once the values for the properties of the characters are determined, their assignment is converted to Ostari code and the appropriate placeholder in the template file replaced with that code.

Generating the Murder and Clues

The generation of the murder itself, and therefore the clues which the player finds, is handled by Ostari. The generated characters act as players within an Ostari game, with access to domain dependent actions for moving, waiting, murder, witnessing events, and potential other crimes. These actions are constrained by their `motivation` property, so NPC agents will not take significant actions such as murder or theft without an in-story reason. Character motivations provide goals for the NPC agents. Currently, agents take actions randomly until they have achieved their goals, but because actions other than movement are constrained by their motivation, they can only take actions that will eventually lead to their goals being achieved. Actions taken by NPC agents can generate evidence, either in physical form, by e.g. leaving a hair behind to indicate their hair color (which sets the `hairclue` property of their current location to be their `haircolor`), or by being witnessed by another character. For example, if an NPC wanders around the game world and observes the murder or parts thereof, they will have evidence for the player that will help them determine the murderer. However, not all NPCs will be forthcoming with the information they have, as we will describe in the next section. The game makes sure that the player can always have enough evidence to solve the murder by initially creating evidence in the location of the murder for every property of the murderer, then deleting pieces of evidence if a character witnesses the corresponding properties. This can be seen in the code for the `commitmurder` and `witnesshair` actions, as shown in Figures 3 and 4. This serves the goal of creating a satisfying story, by reducing the chance that the evidence will be trivially gathered into one place while maintaining the solvability of the murder.

Solving the Murder

To solve the murder, the player investigates the locations in the game and interviews the suspects to determine who could have committed the crime. The player can search the location they are currently at, question or observe the physical properties of a suspect in the same location as them, or move between locations. The player can also accuse any suspect who has not been ruled out as the murderer, although the accusation will fail unless all other suspects have been ruled out, i.e. the detective has enough evidence to bring the case to court. Searching a room will always find any clues in the room, whether they relate to the murder or to other crimes. Suspects who know properties of the murderer will tell the inspector upon being questioned, unless they have the `hidesecret` motivation, which in-story signifies that they a secret, such as being involved in another crime. However, if the inspector learns their secret (which is represented in the mechanics by believing that the suspect's `secret` property has a specific value), and questions them after doing so, they are willing to admit to their actions and will share any information they might have. For example, if a character stole something and saw the murderer in the process, they will not be forthcoming during questioning because they want to hide their own crime. However, when the detective searches the thief's room they might find the stolen property and be able to confront the thief, who will now cooperate. In epistemic terms, the thief will cooperate once the detective has knowledge of their crime, which is what is tracked by Ostari. As the properties of the murderer and of the suspects are revealed, the game uses Ostari's belief system to track the player's knowledge of which suspects have which properties, and who could or could not have committed the murder. The player may accuse a suspect at any point during the game, but only if they have gathered enough evidence to eliminate all other suspects, the accusation will be successful, and the player wins. If the player accuses the wrong suspect, or even if they accuse the actual perpetrator of the murder but do not have enough evidence to uniquely identify them as the murderer, the game ends with a loss for the player.

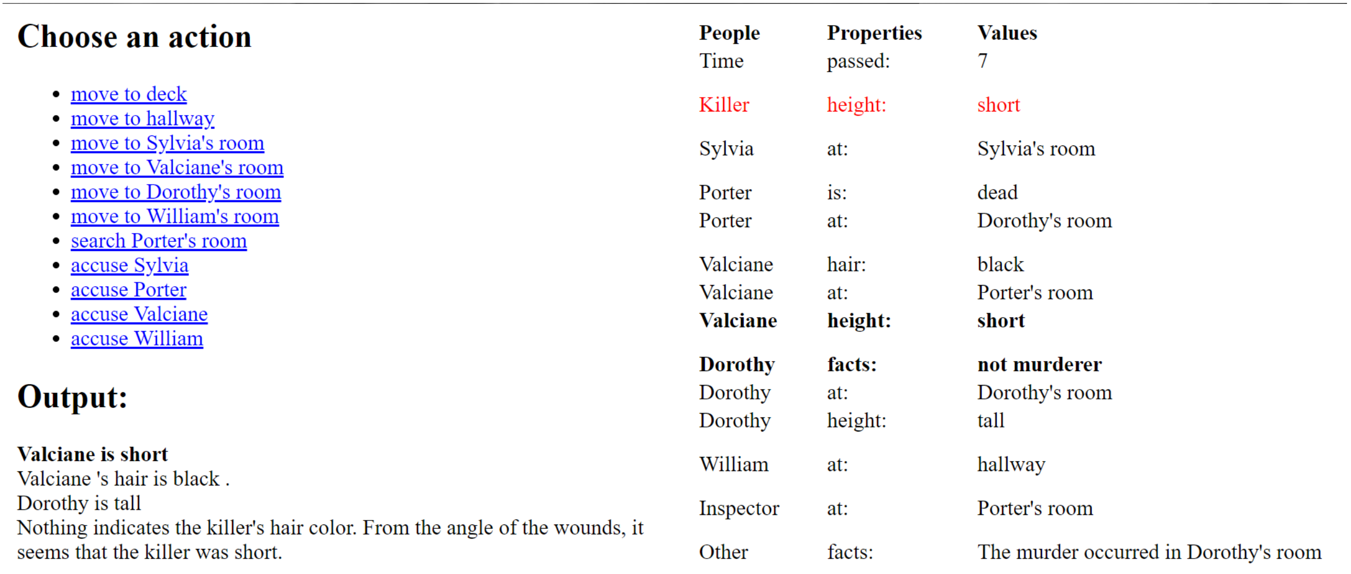
To prevent the game from devolving into a matter of simply gathering as much evidence as possible, there is a limit on the number of actions the player can perform. Once the player runs out of time, the game also ends with a loss for the player. Players therefore have to carefully plan where to go and which information to gather.

User Interface

Our game is presented in the form of a web page. The user is first presented with a short back story, and is then redirected to the user interface shown in Figure 2. There they can choose which action to perform by clicking the appropriate link. Each action has a corresponding effect, such as a move action moving the player character to a different location, or an observational action providing the player with information. This effect is reported to the player, but to simplify the process of eliminating the impossible, our user interface also provides a summary table of the information they have obtained about the various suspects.

Results

In this section we will provide a short discussion of an example scenario we have developed to demonstrate the capabilities of our approach. A scenario consists of a setting,



Choose an action

- move to deck
- move to hallway
- move to Sylvia's room
- move to Valciane's room
- move to Dorothy's room
- move to William's room
- search Porter's room
- accuse Sylvia
- accuse Porter
- accuse Valciane
- accuse William

Output:

Valciane is short
Valciane 's hair is black .
Dorothy is tall
Nothing indicates the killer's hair color. From the angle of the wounds, it seems that the killer was short.

People	Properties	Values
Time	passed:	7
Killer	height:	short
Sylvia	at:	Sylvia's room
Porter	is:	dead
Porter	at:	Dorothy's room
Valciane	hair:	black
Valciane	at:	Porter's room
Valciane	**height:**	**short**
Dorothy	**facts:**	**not murderer**
Dorothy	at:	Dorothy's room
Dorothy	height:	tall
William	at:	hallway
Inspector	at:	Porter's room
Other	facts:	The murder occurred in Dorothy's room

Figure 2: A screenshot of the browser-based interface provided by our game. The player can choose between the different actions on the left, and will be told about the outcomes of these actions below. On the right side the user is presented with a summary of the facts they have uncovered about the suspects as well as what they know about the murder and murderer.

which is presented in the back story at the beginning of the game, character properties and motivations, potential goals, and the corresponding actions the characters can perform. Currently it is necessary that each property can be observed in some way by the detective. Additionally, properties that can be observed by NPCs can serve as evidence obtained from questioning them.

Setting and Characters

For our example game we chose a cruise ship, the Enchanted Princess of the Horizon, as the setting. As the back story informs the player, one of the passengers on the ship has been murdered while the ship is at sea and it is now headed for the next harbor. The detective only has limited time (or actions, in game terms) to determine the murderer, until the passengers are allowed to disembark making it significantly harder for the authorities to apprehend the culprit. The game is limited to a small area of the ship, consisting of a deck, a hallway, and the rooms of a small cast of NPCs, which are the suspects. The NPCs are Sylvia, Valciane, Porter, Dorothy, and William. Each character has two physical properties, which are hair color (which can be blonde, black, or brown) and height (which can be tall or short). Additionally, one character has a valuable gem.

The initial motivations available in our scenario are having a grudge against another NPC, and wanting to steal the gem. Each of these two intentions is assigned to one of the characters, who will then set out to perform the murder and steal the game, respectively. At the conclusion of their crimes, both of the characters obtain a new motivation of hiding their secret, which makes them uncooperative during questioning. However, the thief will cooperate if their theft is uncovered by the player.

Action Encoding

As mentioned above, actions are written in Ostari. Figure 3 shows the code for the `commitmurder` action. The parameter `p1` represents the murderer, whichis only known to `p1` themselves, i.e. all other characters will know that a murder occurred, but not who committed it. The action ensures that the murderer has a grudge against the victim, prevents edge cases such as the murderer killing themselves, or the inspector committing the crime with appropriate preconditions, performs the actual murder and places the evidence. Additionally, a placeholder character called `killer` is set up to have the same properties as the murderer. The `public(killer, p1)` indicates that this information is public *to these characters only*, but hidden from everyone else (by default, properties are not known to any character).

As noted above, each murder places evidence for the murderer's height and hair color, but if another character is able to observe one of these attributes, and the detective is therefore able to obtain this information by interrogating them, this evidence is removed from the crime scene. Figure 4 shows the action of witnessing a character's hair color. It has preconditions that ensure that there was a murder, the character is alive/active and at the location of the murder and that they or someone else don't already know the hair color of the murderer. It then removes the evidence indicating the hair color from the murder location, by setting it to `unknown` and updates the players belief with the `telltruth` statement.

```
commitmurder(p1(p1): Suspects, victim: Suspects)
{
    precondition (grudge(p1) == victim) or (grudge(p1) == unknown);
    precondition eq(p1) != victim;
    precondition eq(p1) != inspector;
    precondition Forall p in Suspects: murderer(p) == False;
    precondition (location(p1) == location(victim) or
        (location(p1) == unknown) or (location(victim) == unknown));
    precondition active(p1) == True;
    precondition active(victim) == True;
    murderer(p1) = True;
    active(victim) = False;
    print("$*$", victim, ":is:dead^$^");
    public (killer, p1) motivation(killer) = motivation(p1);
    public (killer, p1) haircolor(killer) = haircolor(p1);
    public (killer, p1) tall(killer) = tall(p1);
    public (killer, p1) grudge(killer) = grudge(p1);
    hairclue(location(p1)) = haircolor(p1);
    heightclue(location(p1)) = tall(p1);
    murderlocation(game) = location(p1);
    motivation(p1) = hidesecret;
    secret(p1) = murder;
    ...
}
```

Figure 3: The code for the `commitmurder` action. The ommitted code only contains output directives for the back story.

```
witnesshair(p1: Players, c: Colors)
{
    precondition active(p1) == True;
    precondition location(p1) == murderlocation(game);
    precondition Exists p in Suspects: active(p) == False;
    precondition murderer(p1) == False;
    precondition Forall x in Colors: (eq(x) == unknown) or
        (not B (p1): haircolor(killer) == x);
    precondition Forall p in Suspects: ((murderer(p) == True) or
        (Forall x in Colors: (eq(x) == unknown) or
        (not B (p): haircolor(killer) == x)));
    hairclue(murderlocation(game)) = unknown;
    telltruth (p1): haircolor(killer) == c
}
```

Figure 4: The code for the `witnesshair` action.

Example Game

For an example play-through, the system generated the following characters and motivations:

- Sylvia is tall, has black hair, and has a grudge against Porter.
- William is tall, has brown hair, and has the gem.
- Valciane is short, has black hair, and wants to steal William's gem.
- Porter is short, and has blonde hair.
- Dorothy is short, and has brown hair.

Note that the assignment of properties to characters is unique, i.e. no two characters have the same values for all properties. This means, if the detective finds evidence for the murderer's size and hair color, they can uniquely identify the culprit.

After the characters have been generated, they perform actions according to their motivations:

- Valciane goes to William's room and steals his gem, and her motivation changes to hiding the fact that she stole the gem.
- Sylvia runs into Porter in the hallway, and murders him. Her motivation changes to hiding the fact that she murdered Porter. Valciane is in the hallway, and witnesses the fact that Sylvia has black hair (but not her height!).

This is where the player character comes in to investigate Porter's murder. To solve the case, they first observe the body, and deduce that the murderer must be tall from the angle of the wound. They then observe the suspects and can already rule out Valciane and Dorothy, both of whom are short. However, questioning the remaining two suspects does not yield any information, because William does not know anything about the murder and Sylvia wants to hide her involvement. When questioning the exonerated suspects the player is told that Valciane is hiding something, goes to search her room and finds the stolen gem. When the player questions Valciane again, she will tell them what she saw, since her theft was uncovered and she no longer has anything to hide. With this new information, the player is able to determine that Sylvia is the murderer and bring her to justice.

Note that in another play-through of the game, the thief may observe both properties of the murderer, or none at all, or have another character, who has nothing to hide and is thus more willing to cooperate, observe the murderer, requiring the player to employ different approaches in how to solve the murder. Additionally, while this sample scenario is relatively simple, it is possible to add more properties and motivations for the characters, resulting in more opportunities to hide and find information related to the murder.

Conclusion and Future Work

We have presented our approach to procedurally generated murder mystery games, which works by generating characters that are guaranteed to be unique, and the necessary evidence to identify a murderer. In addition to physical evidence our system allows other characters to observe the murder and either share that information or intentionally hide it to protect another secret they have. This means that solving the game once does not trivialize the game in the future. Player knowledge is tracked using Dynamic Epistemic Logic as implemented by Ostari system to determine when they have collected enough evidence. We have also shown an example game with a small number of properties, that results in a moderate number of meaningfully distinct detective stories. For future work, we want to build upon this foundation to create a wider variety of scenarios, by adding additional character properties and motivations.

While our characters only follow very basic motivations, a possible future extension could give them more complex goals, which would result in more variation. For example, rather than having a grudge as the underlying motivation for a murder, wanting the gem could also serve as the impetus. A character with such a motivation would break into the room in which the gem is held, and if the owner is present would necessarily have to murder them to obtain the gem. A challenge that needs to be addressed in this case is to ensure that exactly one murder happens when multiple characters have motivations that might result in one. Such an extension might also incorporate Ostari's AI planning system more deeply. Additionally, characters in our game might conceal what they know, but they will never outright lie to the player. However, lying of suspects is a staple of detective fiction, and we are looking at ways to incorporate that into our game. Rather than outright eliminating possibilities, as the belief system does in the version of Ostari used by the game, the detective would have to reason about the probabilities of different scenarios. Determining when the detective has successfully solved a murder in such cases remains a problem for future work.

References

Adams, T. 2017. Secret identities in dwarf fortress. Working Notes of the AIIDE 2017 Workshop on Experimental AI in Games.

Baltag, A. 2002. A logic for suspicious players: Epistemic actions and belief–updates in games. *Bulletin of Economic Research* 54(1):1–45.

Eger, M., and Martens, C. 2017a. Character beliefs in story generation. Working Notes of the AIIDE 2017 Workshop on Intelligent Narrative Technologies.

Eger, M., and Martens, C. 2017b. Practical specification of belief manipulation in games. Proceedings of the 13th AAAI International Conference on Artificial Intelligence and Interactive Digital Entertainment.

Hendrikx, M.; Meijer, S.; Van Der Velden, J.; and Iosup, A. 2013. Procedural content generation for games: A survey. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* 9(1):1.

Riedl, M. O., and Young, R. M. 2010. Narrative planning: balancing plot and character. *Journal of Artificial Intelligence Research* 39(1):217–268.

Ryan, J. O.; Summerville, A.; Mateas, M.; and Wardrip-Fruin, N. 2015. Toward characters who observe, tell, misremember, and lie. *Proc. Experimental AI in Games* 2.

Stockdale, A. 2003. Cluegen: An exploration of procedural storytelling in the format of murder mystery games. In *Proceedings of the AIIDE workshop on Experimental AI in Games*, volume 2.

Van Dine, S. 2015. *Twenty rules for writing detective stories*. Booklassic.

Van Ditmarsch, H.; van Der Hoek, W.; and Kooi, B. 2007. *Dynamic epistemic logic*, volume 337. Springer Science & Business Media.

Young, R. M. 1999. Notes on the use of plan structures in the creation of interactive plot. In *AAAI Fall Symposium on Narrative Intelligence*, 164–167.