Identifying plot holes in narrative stories by simulating events

ARON DAVIDS, University of Twente, The Netherlands

In the entertainment industry, a narrative writer's value is based on the quality of the stories they produce. Plot holes are generally seen as mistakes in the logic or consistency employed by the writer. This paper details the development of a program which simulates the events happening in a story, to validate the logical soundness and consistency of rules or facts stated in the story. To appeal to a wider range of writers, the program also has the ability to analyze stories that include a few adaptations of time-travel. A narrative writer can use this program during the building of their story to avoid some of the more common categories of plot holes, without any harm to the writer's creative freedom. The writers are free to ignore any warnings the program displays. This paper also proposes the use of the philosophical field of epistemic logic, which offers a way to model knowledge and beliefs. This topic can be used to expand the program's identifiable plot holes.

Additional Key Words and Phrases: narrative writers, plot holes, world simulation, epistemic logic, time-travel.

1 INTRODUCTION

In the entertainment industry, a creative writer's value is based on the quality of the stories they produce, as this makes whatever implements the story more popular and profitable. Naturally, tools have arisen that aid a writer in the different phases needed to get to a fully-written, high-quality story. For example, Creative Help [9] is a tool that generates suggestions for the next sentence during the writing of a story. To simplify proofreading a writer can use their text-editor's built-in spelling- and/or grammar checkers, or an external tool. [7] Additionally, Virtual Storyteller [11] is a system that fully automates plot creation, narration and presentation altogether. This paper introduces a program that aids in going from a concept to a logically-correct timeline, before the writing of a story starts. In literature, a distinction is made between the chronological ordering of events in a story, which is called the fabula, and the progression of these events in the narration of the story, the syuzhet. [13] The program focuses on supporting the creation of a fabula, such that writers are less prone to create inconsistencies in their narratives.

Such an inconsistency is called a plot hole, and can happen when a story builds a certain rule or fact in one part, but breaks it down or contradicts it in another. However, when looking at the majority of what consumers of these stories point out to be plot holes [3], a different definition of a plot hole can be found. Namely, a plot hole occurs when the consumer of the story finds an event or fact of the story to be logically wrong or unrealistic.

The **goal** of this paper is the development of a program that is able to detect both kinds of plot holes, as described above, in the timeline of a story. To support this goal, the following research questions will be answered:

- **RQ1:** Which plot holes can be generalized to abstract events?
- **RQ2:** Which attributes of elements or events in a timeline cause these plot holes?
- **RQ3:** Are there any limitations to the creative writing process caused by the rules implemented by this program?

Before starting to design and write the program, **RQ1** and **RQ2** need to have been answered. For **RQ1**, compilations of plot holes will be analyzed for suitable cases that can be generalized using certain similarities. For **RQ2**, a way to model the events leading up to the collected plot holes needs to be designed, and the conditions that cause the plot hole need to be singled out. After knowing the elements that need to be modeled and the conditions that cause the plot holes, a design will be made for the program, which then is implemented. The implemented program will show the restrictions that are put onto the timeline and these can be analysed for fairness to the writer's creative freedom, answering **RQ3**. To give the writer more creative freedom, a couple of science-fiction concepts will also be implemented into the model. These concepts will first be explored to understand the effects they have on a timeline. Next, the concepts will be taken into the design of the program, and restrictions are made to make the writer follow the concepts without creating more plot holes.

The first thing needed for this project, is a better understanding of the adaptations of time-travel the program will include. Section 2 will go into how these concepts from science-fiction effect a timeline and it's elements. Next, research into what the most common plot holes are, and how these could be corrected in a simulation, will be done in section 3. Afterwards, section 4 explores epistemic logic, a sub-field of epistemology in philosophy, which can be used to model knowledge and beliefs for the simulation. Section 5 brings together the elements to be considered for the simulation and creates a design for the program. Section 6 describes the format created for the input of the program. Furthermore, section 7 details a test plan for the program. Section 8 explores the results that have been achieved with this program within the time restraints of this project. Lastly, section 9 and section 10 conclude this paper with a general conclusion and possible further work.

2 SCIENCE-FICTION

In science-fiction there exist certain concepts which change the form or logic of a story's timeline. Below is a list of events that can change a timeline and their effects. Only these concepts are considered when answering the research goal:

- **The story moves to an alternate universe:** Now all previously declared events do not necessarily have to be a fact anymore, as the story runs on a completely different timeline now.
- **A character travels through time:** In this case, different things can happen, as there are different implementations of time-travel

TScIT 37, July 8, 2022, Enschede, The Netherlands

possible. The majority of time-travel utilized in motion pictures can be generalized into three groups [1][2]:

- **Time dilation:** A character moving close to the speed of light experiences time faster than other characters, and thus moves over the same timeline at a higher speed.
- **Self-consistent time-travel:** If a character travels back in time and interacts with certain events, this version of the character was always at those events to begin with.
- **New-history time-travel:** The very act of a character travelling back in time will set the timeline onto a different route, because originally that version of the character did not exist at that point in time.

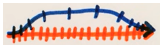


Fig. 1. Time-dilation

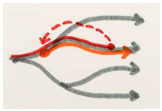


Fig. 2. Self-consistent time-travel

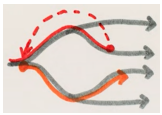


Fig. 3. New history time-travel

In these cases, stories can be expressed as having multiple overlapping, or completely different timelines. Complexity can be avoided by looking at time-travelling characters as being different characters as their previous selves.

New-history time-travel also involves moving to an alternate timeline. Therefore, when new-history time-travel is modeled, the ability to move to alternate timelines is automatically included.

3 DOMAIN ANALYSIS

To come to abstract ideas of plot holes, a lot of ideas have been taken from compilations and posts about plot holes in movies, found on Buzzfeed [3] or r/plotholes[3]. After each point, some examples of similar plot holes in movies are given. The category given to those plot holes is in **bold**, and mostly points towards what causes the plot hole. Below each point there is a paragraph detailing how plot holes like those could be modeled on an abstract level.

- In *Men In Black* (1997), the entirety of earth was in danger, yet they sent only two agents to fix it. In *Spider-Man 3* (2007), Peter takes a sample of the symbiote to his professor. The professor deduces it is alien, without freaking out about the fact that it is proof of alien life. These two plot holes are similar in that the event is given an illogical level of **significance**.

Theoretically, this could be modeled by having some algorithm or AI that is capable of deducing the logical amount of significance an event should have. The questions of whether this is possible, and how this can be realized in practice, will not be answered in this paper, as that could be a complete research topic on it's own.

- In *The Dark Knight Rises* (2012), a full team of police members was trapped underground for months, yet they all walk out clean-shaven and well-dressed. This plot hole is categorized as a wrong depiction of **biological processes**.

This does not seem like it could be generalized into one abstract event. It could be modeled by simulating the biological workings of each organism relevant to a story, but that will also not be possible in the limitations of this project. Another approach is to have a system that can logically deduce consequences of events. The abilities this system would need to have get close to common sense in humans. This will not be researched in this paper.

- In *Halloween* (1978), Michael Myers suddenly knows how to drive a car, even though he was incarcerated since the age of 10. In *Beauty and the Beast* (2017), Belle teaches the Beast how to read. However, the Beast was a highly educated prince. What is at fault in these movies is that there has not been enough attention to the **knowledge or ability** a character would have.

Thinking about how to model a character's knowledge prompts a look into epistemic logic. This will be done in section 4.

- In *Mamma Mia* (2008), Donna states her mother is dead, but in the sequel her mother is alive. This makes sense to be a plot hole, as **dead characters can not do anything**.

In a model, this would require updating the attributes of a character to signal it is dead, once it dies. Additionally, checking if the character is alive when the character attempts to do something later on in the story is necessary.

- In *Black Panther* (2018), the northern tribe is said to all be vegetarians. However, they do have fishermen in another point of the movie. This is a discrepancy in the **beliefs and preferences** of the characters.

Modelling belief is also part of epistemic logic, see section 4.

- In *Ant-Man* (2015), it is said that equal weight is retained when a person is shrunk. However, Dr. Pym keeps a shrunken tank on his key-chain. In *The Karate Kid* (1984), it was said that, in a fight, hitting an opponent in the head would lead to a disqualification from the championship. Yet, a fight is won with a kick to the head, without consequences. These movies are **inconsistent in their statements**.

If characters' knowledge and beliefs are adequately modeled, plot holes like these should also be uncovered. Because, if a general statement is made about the story's world, this statement can be stored in either a collective knowledge base or in the knowledge bases of the individuals that know about this statement. Then, if a contrary statement is made, this would be flagged as a conflict in the knowledge bases.

4 EPISTEMIC LOGIC

Epistemic logic [8], a sub-field of epistemology, concerns itself with approaching knowledge and belief with logic. In epistemic logic K and B are operators representing knowledge and belief, respectively. And $K_a\varphi$ and $B_a\varphi$ are read as "agent A knows that φ" and "agent A believes that phi", respectively. Epistemic logicians use epistemic logic to attempt to characterize real knowledge and belief, the logical relations between different conceptions of knowledge and belief,

[1]This classification was personally created after watching the video which the figures are taken from.
[2]Figures 1, 2 and 3 are taken from "Time Travel in Fiction Rundown", Youtube video, posted by "minutephysics", October 26th, 2017. Timestamps 0:24 and 1:31.
[3]https://www.reddit.com/r/plotholes/

and the epistemic features of groups of agents. Basic logic rules are used to evaluate what an agent knows from the base truths that the agent starts out with. $K_a p \wedge p \rightarrow q$ can be evaluated to $K_a q$, as agent A knows p, and p implies q, so agent A knows q.

When the knowledge and beliefs of characters on a timeline are modeled, epistemic logic will allow the identification of contradictions by evaluating formulas. A contradiction in this logic will mean the timeline's story is not logically sound, and thus includes a plot hole.

4.1 Logical omniscience

A complaint to be made against the use of epistemic logic, is the problem of logical omniscience. [5] This problem states that the actual human reasoning needed to realize complex epistemic models, is close to superhuman. In a model where q follows logically from p, and agent A knows p to be true, the agent would also know q to be true. However, in real life a person can overlook the fact that q follows from p, and thus not come to know that q is true. This is more plausible the more complex statements q and p become.

To avert the problem of logical omniscience, awareness logic [2, 10, 12] can be used. By adding awareness as an operator, and redefining $K_a\varphi$ to mean "agent A *implicitly* knows that φ", *explicit* knowledge of φ can be modeled to result from *implicit* knowledge of φ *and* awareness of φ. This way, actual human reasoning can be modeled by tracking real conclusions they have made with awareness for those conclusions.

4.2 Higher-order attitudes

When exploring epistemic logic, one might stumble upon nested formulas such as $K_a K_a \varphi$. This would read as "agent A knows that agent A knows that φ." When this is written in the formula $K_a\varphi \rightarrow K_a K_a \varphi$ it now entails that agent A has knowledge over *all* that it knows. This is the principle called '4', a popular principle and one that can be used to build a logic set. Similar to this is the principle called 'KB2': $B_a\varphi \rightarrow K_a B_a \varphi$.

4.3 Principles

Other main principles of epistemic logic include those in table 1 [1]. These principles can also be used for belief, instead of knowledge.

Additionally, from literature [4, 6], three principles emanate regarding the relations between knowledge and believe, illustrated in table 2.

An epistemic logic can be built upon a combination of any base principles that can be used to model knowledge and belief. The relevance to this paper is the ability to track knowledge of characters throughout a timeline. The best way epistemic logic can be applied to the program developed in this paper is by building a logic set with relevant base principles that allows for the modelling of characters knowledge and beliefs, whilst also using the concept of awareness logic so the reasoning of the characters is not displayed as superhuman.

5 THE SIMULATION

The following elements are to be considered for the simulation according to section 3:

	Formula	Reads as
K	$K_a(\varphi \rightarrow \psi) \rightarrow (K_a\varphi \rightarrow K_a\psi)$	If you know φ implies ψ, then, if you know φ, you must know ψ.
T	$K_a\varphi \rightarrow \varphi$	If you know φ, then φ must be true.
D	$K_a\varphi \rightarrow \neg K_a \neg\varphi$	If you know φ, then you do not know the negative of φ.
4	$K_a\varphi \rightarrow K_a K_a \varphi$	If you know φ, then you must know that you know φ.
B	$\varphi \rightarrow K_a \neg K_a \neg\varphi$	If φ is true, then you must know that you do not know the negative of φ.
5	$\neg K_a\varphi \rightarrow K_a \neg K_a \varphi$	If you do not know φ, then you must know that you do not know φ.

Table 1. Epistemic principles and their implications.

	Formula	Reads as
KB1	$K_a\varphi \rightarrow B_a\varphi$	If you know φ, then you must believe φ.
KB2	$B_a\varphi \rightarrow K_a B_a \varphi$	If you believe φ, then you must know you believe φ.
KB3	$B_a\varphi \rightarrow B_a K_a \varphi$	If you believe φ, then you must believe you know φ.

Table 2. Principles regarding relations between knowledge and belief.

- Characters and their:
  - knowledge.
  - abilities.
  - beliefs.
  - preferences.
  - state of being, e.g. dead or alive.
- Common- or world-wide knowledge.

In order to incorporate these elements into a bigger picture of the world, and also to prevent other contradictory events, first a base simulation will be set up with the following elements:

- Objects.
- Which character holds which object.
- Places.
- Which objects and/or characters are at what places.
- Characters' ages.
- Objects' ages.

For the purpose of integrating the adaptations of time-travel explored in section 2, the simulation must work with a universe that has multiple worlds or timelines in it. Furthermore, the story must be able to be imported into the program. This will be done by importing the story as a file in a specific data format, which stores the timelines and elements in a way that is useful for this

program. The conversion of the actual narrative story to this format is not included in this project. Next, the file can be converted to representative objects in the program. Then, these objects can be traversed step by step to update the simulation accordingly.

Keeping everything reported in this section in mind, the class diagram shown in figure 4 has been made.

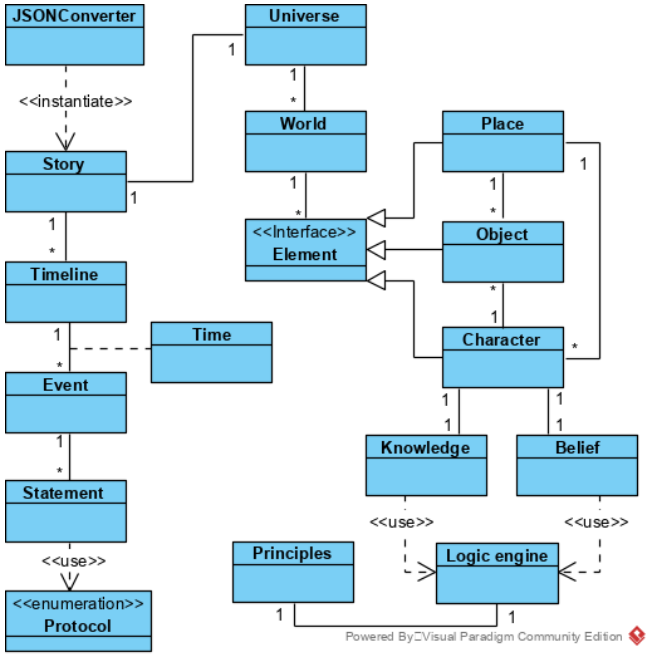


Fig. 4. Class diagram

With this design, errors can be raised when the following events happen:

- An element is used in a statement, but this element was never initialized before.
- An element is initialized, but an element with this name already exists.
- An element's age does not match up with the age it should have for the story.
- A character acts in the story, while this character is dead.
- An object is transferred from character A to character B, but character A does not actually have the object.
- A character obtains an object, but this object is actually held by another character. A difference in the terms transfer and obtain is important here.
- Different versions of time-travel are used in the same story, or the way time-travel is used is not logical.
- An element suddenly changes it's attributes or behaves like a different kind of element after time-travelling.

6 DATA FORMAT

The format needs to be able to represent a variety of events that can happen in a story. Interactions between the elements and the traversing of elements between timelines have to be represented by statements. The following list shows the different levels and elements in the format:

- Title (string)
- Time-travel type (string)
- Timelines (array of blocks):
  - Timeline title (string)
  - Timeline identifier (integer)
  - Events (array of blocks):
    - Date (string)
    - Description (string)
    - Statements (array of strings)

This format has been chosen as it incorporates exactly the elements that the program will use, in a way that they are easily retrieved and stored in their respective objects in the program.

The possible forms of a statement and their meanings are as follows (parts between { and } are variable):

- "{name} > born"
  - A character is born.
- "{name} > dies"
  - A character dies.
- "{name} > age > {number}"
  - When a character does something that can only be done at a certain age, this should be added to the event as a manual check.
- "{name} > knows > {statement}"
  - A character gains knowledge about a statement.
- "{name} > beliefs > {statement}"
  - A character starts believing in a statement.
- "{name} > founded"
  - A place is founded.
- "{name} > moves > {name}"
  - An element with the first name moves to a place with the second name.
- "{name} > created"
  - An object is created.
- "{name} > obtained > {name}"
  - An object with the first name is obtained by a character with the second name.
- "{name} > transferred > {name}-{name}"
  - An object with the first name is transferred from a character with the second name to a character with the third name.
- "{name} > lost > {name}"
  - An object with the first name is lost by a character with the second name.
- "[{name}, {name}, ...] > timetravel > {date}"
  - A list of elements timetravel to another date.
- "[{name}, {name}, ...] > arrive > {date}"
  - A list of elements arrive from another date.
- "[{name}, {name}, ...] > timedilation > {time}"
  - A list of elements experience time-dilation. During this time-dilation they experience an amount of time equal to {time}.
- "[{name}, {name}, ...] > stopdilation"
  - A list of elements exit time-dilation. The time that has actually passed is the difference between the dates of the events where the time-dilation started and stopped.

These statements are designed to be easily parsed by the program, as the 'command' is always given after the first '>', and the first and optional third parts of the statements are always variable.

Because there needs to be a distinction between a character and their time-travelling counterpart, '.2' has to be appended to their name once a character has travelled back in time to a timeline that also has their original character still in it. Then the program can recognize the young and old versions of the character and continue the simulation.

7 TESTING

To test the program for correct functionality, unit and system tests have been made. Unit tests have been written to test the functionalities of individual methods using assertions. System tests have been written in the form of example stories that can be run through the program to test the complete functionality. The results of such system tests need to be manually inspected for unintended behaviour.

7.1 Unit testing

Individual methods of the most important classes have been tested by code that manually creates environments in the simulation and asserts that the effect of the methods on that environment are as intended. The tests also include cases where the detection of plot holes are possible and asserts that in environments where those plot holes are present, they are correctly caught.

```
def test_character_born(world):
    """
    Tests ._character_born()
    """
    world._World__character_born("test_name")
    assert world._World__search(world.characters, "test_name").name == "test_name"
    try:
        world._World__character_born("test_name")
    except DuplicateError as error:
        print(error.message)
    else:
        raise AssertionError
```

Fig. 5. A test for a single method from a class.

7.2 System testing

In order to test functionality in larger parts of the program, stories that trigger the intended functionalities were designed. If the output of the program raised errors or warnings because of these stories, it meant the functionality was not as intended.

8 RESULTS

Figure 7 shows the class diagram of the current working version of the program at the time this paper is written. The program is capable of detecting a set of plot holes in a story, as specified by section 5. Some of these are rarely seen in practice, but are logical consequences of the way the simulation is designed.

Figures 8 and 10 are screenshots of files in the format explained in section 6. Figures 9 and 11 are the outputs of the program when these files are given as input.

The research into epistemic logic has shown confidence that it is possible to model the knowledge and beliefs of characters. Once a logic engine suited for epistemic logic is incorporated into the program, the detection of plot holes regarding knowledge, abilities, beliefs, preferences, or inconsistencies in statements made in the story would be detected by the program. Warnings would be raised if the logic engine notices two statements that are contradictory. These statements would be added if a character learns something new, or gains a new belief.


```
"title": "Self-consistent test",
"time-travel type": "Self-consistent",
"timelines": [
    {
        "title": "Timeline 1: Original",
        "id": 0,
        "events": [
            {
                "date": "2001-04-25 ?",
                "description": "John is born and two infinite timers are created.",
                "statements": ["John > born", "Timer > created", "Timer2 > created"]
            },
            {
                "date": "2017-??-? ?",
                "description": "The timers are gifted to John.",
                "statements": ["Timer > obtained > John", "Timer2 > obtained > John"]
            },
            {
                "date": "2020-03-07 ?",
                "description": "Old John arrives in 2020.",
                "statements": ["[John.2, Timesuit.2, Timer.2, Timer2.2] > arrive > 2025-6-23 ?"]
            },
            {
                "date": "2021-02-01 ?",
                "description": "Old John loses the first timer.",
                "statements": ["Timer.2 > lost > John.2"]
            },
            {
                "date": "2021-03-01 ?",
                "description": "Old John creates a timesuit for Young John.",
                "statements": ["Timesuit > created", "Timesuit > obtained > John.2", "Timesuit > transferred > John.2-John"]
            },
            {
                "date": "2023-11-07 ?",
                "description": "Young John learns how to make a timesuit. And travels back to 2020.",
                "statements": ["[John, Timesuit, Timer, Timer2] > timetravel > 2020-03-07 ?"]
            },
            {
                "date": "2023-12-07 ?",
                "description": "Old John loses the second timer.",
                "statements": ["Timer2.2 > lost > John.2"]
            }
        ]
    }
]
```


Fig. 6. A story, in the format, made to test the functionality of self-consistent time-travel in the simulation.

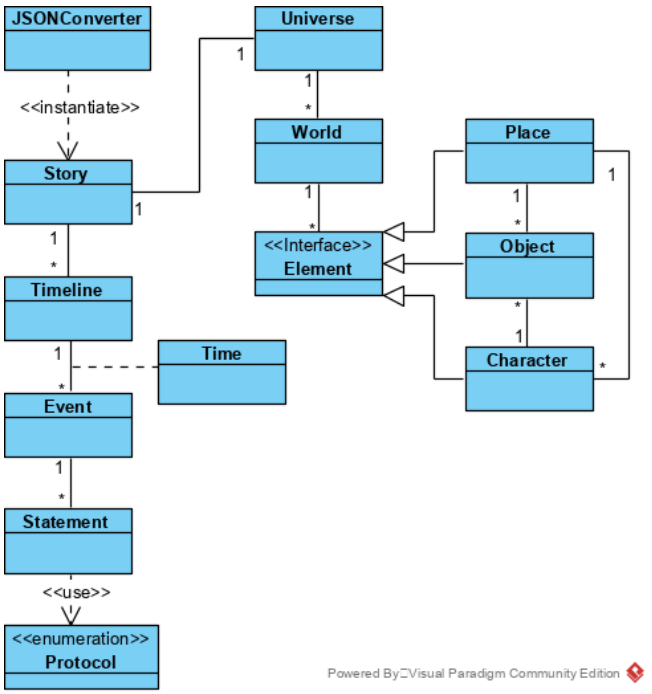


Fig. 7. The class diagram of the current working version of the program.

```
{
    "title": "Error test",
    "time-travel type": "None",
    "timelines": [
        {
            "title": "Timeline 1: Original",
            "id": 0,
            "events": [
                {
                    "date": "1000-01-01 ?",
                    "description": "Jack does not exist.",
                    "statements": ["Jack > dies"]
                },
                {
                    "date": "1001-01-01 ?",
                    "description": "Jack is born.",
                    "statements": ["Jack > born"]
                },
                {
                    "date": "1002-01-01 ?",
                    "description": "Jack is born again.",
                    "statements": ["Jack > born"]
                },
                {
                    "date": "1003-01-01 ?",
                    "description": "Jack is not 10.",
                    "statements": ["Jack > age > 10"]
                },
                {
                    "date": "1004-01-01 ?",
                    "description": "Jack dies.",
                    "statements": ["Jack > dies"]
                },
                {
                    "date": "1005-01-01 ?",
                    "description": "Jack dies again.",
                    "statements": ["Jack > dies"]
                },
                {
                    "date": "1006-01-01 ?",
                    "description": "Ball is created by new-born John.",
                    "statements": ["John > born", "Ball > created", "Ball > obtained > John"]
                },
                {
                    "date": "1007-01-01 ?",
                    "description": "Jack gives John the ball.",
                    "statements": ["Ball > transferred > Jack-John"]
                },
                {
                    "date": "1008-01-01 ?",
                    "description": "Jack obtains the ball.",
                    "statements": ["Ball > obtained > Jack"]
                },
                {
                    "date": "1009-01-01 ?",
                    "description": "John time-travels.",
                    "statements": ["John > timetravel > 2000-01-01 ?"]
                }
            ]
```

Fig. 8. A story, in the data format, containing a lot of errors.

```
In [1]: runfile('C:/Users/arond/eclipse-workspace/timeline_verifier/tests/system_tests/wrongstory.py')
ELEMENT 'Jack' is not found in the current world.
'Jack' is already used as a name in the current world. Duplicates are not allowed.
Element 'Jack' has been stated to be 10 years old, but in reality is between 1 and 2 years old.
Character 'Jack' can't die, because 'Jack' is dead.
Character 'Jack' is not the current holder of object 'Ball'.
Object 'Ball' is already held by character 'John', so the object can't be obtained again.
Can't use time-travel when time_travel_type is NONE.
```

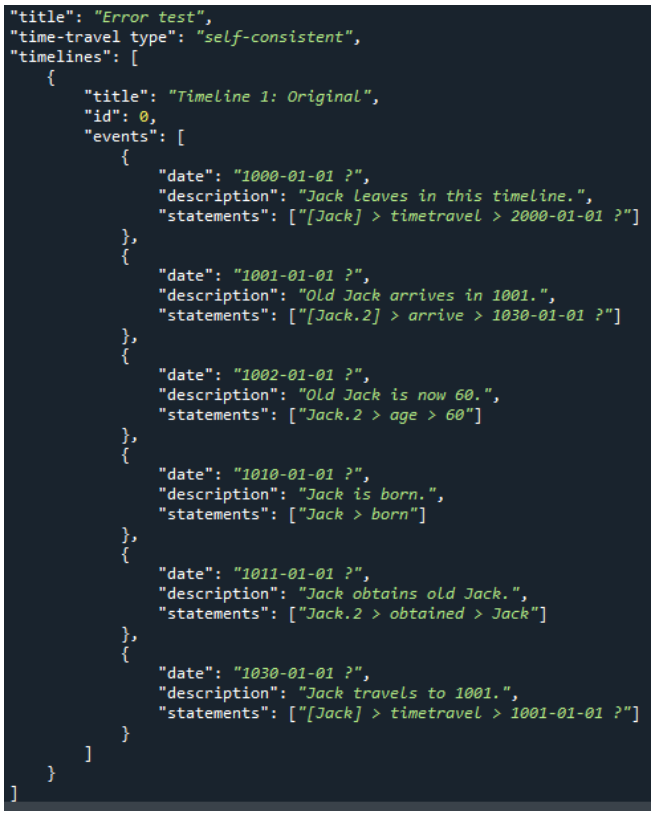
Fig. 9. The output of the program when the file of figure 8 is given as input.

As a narrative writer can simply ignore warnings of certain types of plot holes if they wish to keep the events of their story as they are, any limitations on the creative freedom of the writer can only be caused by the fact that the program does not yet support a type of element, attribute, interaction, or time-travel that the writer would like to have. Nonetheless, a writer can also choose to only check the parts of their story that are supported by this program, and so still have the freedom to do what they want with the story.

9 CONCLUSION

In general, there is a lot that can still be added and improved on this research. This is not an especially hard task, but it will take more time than this project has allowed.

The research goal of this paper has been reached, as the program has been developed and is able to detect plot holes. The research



```
"title": "Error test",
"time-travel type": "self-consistent",
"timelines": [
    {
        "title": "Timeline 1: Original",
        "id": 0,
        "events": [
            {
                "date": "1000-01-01 ?",
                "description": "Jack leaves in this timeline.",
                "statements": ["[Jack] > timetravel > 2000-01-01 ?"]
            },
            {
                "date": "1001-01-01 ?",
                "description": "Old Jack arrives in 1001.",
                "statements": ["[Jack.2] > arrive > 1030-01-01 ?"]
            },
            {
                "date": "1002-01-01 ?",
                "description": "Old Jack is now 60.",
                "statements": ["Jack.2 > age > 60"]
            },
            {
                "date": "1010-01-01 ?",
                "description": "Jack is born.",
                "statements": ["Jack > born"]
            },
            {
                "date": "1011-01-01 ?",
                "description": "Jack obtains old Jack.",
                "statements": ["Jack.2 > obtained > Jack"]
            },
            {
                "date": "1030-01-01 ?",
                "description": "Jack travels to 1001.",
                "statements": ["[Jack] > timetravel > 1001-01-01 ?"]
            }
        ]
    }
]
```

Fig. 10. A story, in the data format, containing errors about time-travel.

```
In [1]: runfile('C:/Users/arond/eclipse-workspace/timeline_verifier/tests/system_tests/wrongtimetravel.py')
While using self-consistent timetravel, elements need to have arrived before they can leave for the past.
Element type conflict: 'Jack' acts as both Elements.CHARACTER and Elements.OBJECT after time-travelling.
Jack's age has suddenly changed after time-travelling.
```

Fig. 11. The output of the program when the file of figure 10 is given as input.

questions have proven effective in the support of this development. **RQ1**, *"Which plot holes can be generalized to abstract events?"*, was answered in section 3, and a list of common generalized events was made. **RQ2**, *"Which attributes of elements or events in a timeline cause these plot holes?*, was also partly answered in section 3, and summarized in section 5. The elements that are responsible for these plot holes are the ones that have been included in the simulation. **RQ3**, *"Are there any limitations to the creative writing process caused by the rules implemented by this program?"*, is answered in section 8. If a writer wants the entirety of their story to be checked by the program, their creative freedom is limited to the elements, attributes, interactions, and types of time-travel that are currently implemented in the program.

10 FURTHER WORK

One major part that is missing for this project to be useful to narrative writers, is the conversion to the data format. A GUI could be developed to allow a writer to easily create the different elements and interactions of a story. The combination of such a GUI and this

program would allow the writer to immediately see warnings if there are plot holes present in the current configuration of their story.

Additionally, the functionality of the program can be expanded by increasing the complexity of the simulation, or by adding more kinds of elements and interactions. For example, as is stated in section 3, biological processes can be tracked, and the significance of events can be computed.

REFERENCES

[1] Guillaume Aucher. 2014. Principles of Knowledge, Belief and Conditional Belief. In *Interdisciplinary Works in Logic, Epistemology, Psychology and Linguistics: Dialogue, Rationality, and Formalism*, Manuel Rebuschi, Martine Batt, Gerhard Heinzmann, Franck Lihoreau, Michel Musiol, and Alain Trognon (Eds.). Springer, Cham, 97–134. https://doi.org/10.1007/978-3-319-03044-9_5

[2] Ronald Fagin and Joseph Y. Halpern. 1987. Belief, awareness, and limited reasoning. *Artificial Intelligence* 34, 1 (1987), 39–76. https://doi.org/10.1016/0004-3702(87)90003-8

[3] Allie Hayes. 2022. 60 plot holes people can't overlook in otherwise good movies. https://www.buzzfeed.com/alliehayes/bad-plot-holes-in-good-movies-reddit

[4] Kaarlo Jaakko Juhani Hintikka. 1962. *Knowledge and belief: An introduction to the logic of the two notions.* Cornell University Press, Ithaca, NY, USA.

[5] Mark Jago. 2007. Hintikka and Cresswell on logical omniscience. *Logic and Logical Philosophy* 15, 4 (Mar. 2007), 325–354. https://doi.org/10.12775/LLP.2006.019

[6] Wolfgang Lenzen. 1978. Recent work in epistemic logic. *Acta Philosophica Fennica* 30 (1978), 1–219.

[7] Marcin Miłkowski. 2010. Developing an open-source, rule-based proofreading tool. *Software: Practice and Experience* 40, 7 (2010), 543–566. https://doi.org/10.1002/spe.971

[8] Rasmus Rendsvig and John Symons. 2021. Epistemic Logic. In *The Stanford Encyclopedia of Philosophy* (Summer 2021 ed.), Edward N. Zalta (Ed.). Metaphysics Research Lab, Stanford University, Stanford, California, USA.

[9] Melissa Roemmele and Andrew S. Gordon. 2015. Creative Help: A Story Writing Assistant. In *Interactive Storytelling*, Henrik Schoenau-Fog, Luis E. Bruni, Sandy Louchart, and Sarune Baceviciute (Eds.). Springer International Publishing, Cham, 81–92.

[10] Burkhard C Schipper. 2014. Awareness. (2014).

[11] Mariët Theune, Sander Faas, Anton Nijholt, and Dirk Heylen. 2003. The virtual storyteller: Story creation by intelligent agents. In *Proceedings of the Technologies for Interactive Digital Storytelling and Entertainment (TIDSE) Conference*, Vol. 204215. 116.

[12] Fernando Velazquez-Quesada. 2011. *Small Steps in Dynamics of Information.* Institute for Logic, Language and Computation, Amsterdam, The Netherlands.

[13] Wikipedia contributors. 2022. Fabula and syuzhet — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Fabula_and_syuzhet&oldid=1083326717 [Online; accessed 20-June-2022].