Procedural constrained story generation based on Propp's and Fabula models

Riccardo Cantoni, Jacopo Essenziale, Manuel Pezzera, Renato Mainetti, N. Alberto Borghese

Applied Intelligent Systems Laboratory AIS-Lab, Computer Science Department
Università degli Studi di Milano, Milan, Italy Telephone: +39 02-50316328
Email: {riccardo.cantoni1}@studenti.unimi.it, {jacopo.essenziale, manuel.pezzera, renato.mainetti, alberto.borghese}@unimi.it

***Abstract*—Using narrative to increase engagement in serious games where the user is required to repeatedly interact with the same game for a long period of time, as in rehabilitative exer-games, could help in reducing the boredom inherent to the task. However, producing enough narrative content to cover long rehabilitation periods could require much effort from the authorial point of view. Having a tool that, given a wide enough knowledge base, can virtually produce an infinite set of stories, could greatly help in addressing this issue. For this reason, we explored the field of automatic story generation, designed and developed a multi-layered tool capable of automating the creation process of stories. The tool allows the user to define constraints at setup-time to ensure that a given set of events will appear in the generated story plot. These events may be mapped to specific serious games, making the resulting stories suitable, for instance, for integration in exer-games based rehabilitation platforms.**

***Index Terms*—narrative generation, procedural content generation, exer-games, user engagement, serious games**

I. Introduction

Thanks to its engagement capabilities, narrative has always been used with different purposes, from pure entertainment to teaching; the pedagogical aspects of narrative, in fact, have been widely discussed [1][2] in literature. The importance of narrative for mankind has been highlighted by the fact that narration itself seems to be the most important mechanism through which we analyse, interpret, and communicate our experiences [3].

Also the media used to present narrative content evolved over time: from spoken words, to books, music, movies, video games and virtual reality. For video games, in particular, the application of artificial intelligence techniques to procedurally generate stories or quests to enhance games longevity and replayability has been discussed in recent years [4][5]. Some technical solutions have been proposed using, for instance, autonomous agents [6] or authorial planning [7] and applied to different specific fields such as education, training, entertainment, or art [8], however, a robust framework or set of algorithms known to reliably produce interesting and engaging stories are still missing.

Video games are being increasingly used in "serious" contexts, as a way to provide health services directly at patients' homes. Tele-rehabilitation platforms [9][10], for example, allow patients who need rehabilitation to exercise at their homes autonomously, without the direct help of a therapist [10]. Tele-rehabilitation platforms often use exer-games to make the rehabilitation process, which may require intensive and prolonged exercise to be performed almost every day, more enjoyable for patients[1]. Examples of exer-gaming rehabilitation platforms are [12][13][14].

Unfortunately, even though video games may help to relieve users from the boredom of repetitively performing the same exercise in a short time, exer-games are usually implemented as very simple games and can become monotonous after some time, once the player has been able to master the mechanics, causing a loss of motivation [15].

According to Ryan et al. [16], motivation can be of two kinds: intrinsic and extrinsic. In the case of intrinsic motivation, the drive comes from the activity itself and the interest that derives from it, while in the case of extrinsic motivation the satisfaction comes from external reinforcement, like goals and achievement. Several studies demonstrated that pushing too much on extrinsic motivation can be counterproductive and decrease the patient's overall motivation. This problem does not seem to affect intrinsic motivation [17].

This is where automatic narration comes into play: integrating activities within a narrative structure could increase intrinsic motivation [18] by making the user experience more various and interesting [19].

For these reasons, we started exploring automatic narration by developing a two-layer tool to automatize the creation process of story plots. The system borrows elements from both the Propp [20] and Fabula [21] models to formalise story components into data structures, it then generates a plot as a believable path on a set of story events graphs.

When a path is produced, the system maps each node to a list of meta-strings describing the story events. The plot, however, is generated independently from the way the story itself will be presented to the final user, giving us the possibility to visualize the event using different media: a simple text, an image, or even an entire exer-game. This

[1]Exer-game is a term used to define a video-game that is also a form of exercise [11].

978-1-7281-9042-6/20/$31.00 ©2020 IEEE

approach makes it easy to temporarily pause the narration and requires users to play a game or perform some kind of exercise without completely disrupting the narrative flow.

The article is structured as follows: in section II we discuss the implemented system, analysing its multi-layered architecture, and explaining how the plot structure is first generated and then enriched with semantic meaning. In section III we try to highlight what are the advantages and disadvantages introduced by the use of the developed tool, presenting some ideas for further developments. We finally summarise its main features in section IV.

II. Materials and Methods

In this work, we propose a method for the constrained generation of narrative material, to be used as a narrative superstructure to a set of exercises/activities. This implies some domain-specific main requirements regarding the narrative material produced:

- It has to be coherent and believable, otherwise the engagement is lost.
- It should be focused on a single character, in order to encourage the immersion of a single user. This excluded techniques widely represented in literature, based on character simulation [21][22], where the action is not focused on a specific character.
- It must be customizable: the set of Activities to be served to a user is pre-determined, and can not be modified by the generation process. This means that the system needs to allow for the specification of a set of activities, mapped to story events, which are guaranteed to appear in the generated narrative content.
- It can not require any human input during the generation process, or between successive generation processes. Any human input shall be during the initialisation phase, after which the system should work autonomously.

Generating narrative contents procedurally is always a *Mixed Initiative* [23][24], in which human authors and a machine work together in the generation process, achieving varying degrees of automation [25]. In the case of the method we propose here, the human authorial work takes the form of defining a knowledge base to be used by the system in the generation process.

The system proposed adopts a two-layer architecture (Figure 1). A first layer called the *Plot Generation Layer*, processes input parameters, and generates a data structure representing the Plot of the story. This plot is a sequence of events carrying minimal semantic information, which will act as the backbone of the narrative [26]. A *Text Generation Layer*, then, takes the generated plot as input and outputs readable text describing the story.

Both layers rely on data from the human-defined Knowledge Base as space on which the story is constructed.

The final output is a file containing the texts describing each event in the story, as well as, for those events that have been mapped to an activity/exer-game, the identifier of said activity, so that it can be proposed to the user once the corresponding event is to be narrated.

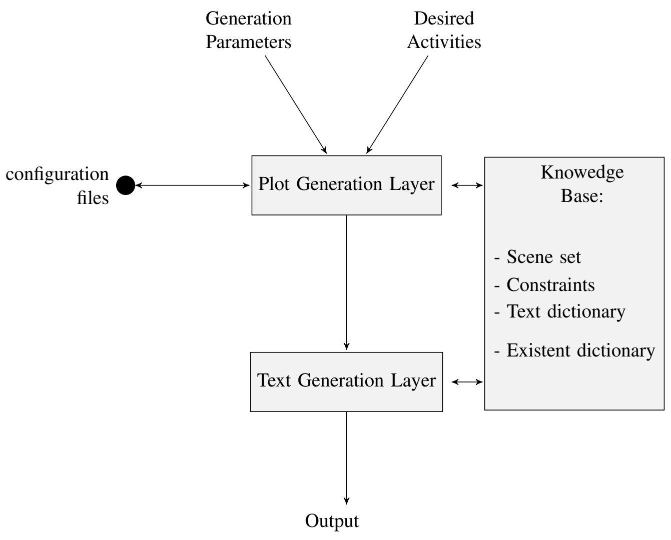

Fig. 1. General Architecture

A. Plot Generation Layer: Overview and Plot State

The Plot Generation Layer works on two different levels of abstraction: at the lowest level, the plot is a series of causally related events, as from the Fabula Model [21]. At a higher level of abstraction, events are grouped into scenes that can be repeated across different stories, as stated in Propp's formalism [21].

The system works on the two levels alternatively, relying on a *Plot State* to propagate information between them and forward along the generation process.

This data structure describes properties of the current situation the plot generation is in. Its form follows that used in the *STRIPS* [27] standard commonly used in classic AI: it is a collection of logic predicates, each of which describes a property of the state that is currently true. Adding predicates to this structure corresponds to adding properties to the State, while removing one corresponds to acknowledging that a property no longer holds.

Logic predicates usually follow the form $predicate(values)$, where $predicate$ denotes what property is being described, and one or more $values$ specify its characteristics. A typical example would be something like $isAt(Bob, park)$, encoding the fact that, in the current situation, Bob is at the park.

In the current implementation of the system proposed here, however, predicates are not restricted to this form, being implemented as collections of atoms. The first item in the collection is assumed to correspond to the $predicate$. The same information carried by $isAt(Bob, park)$, would be therefore encoded as something similar to $[isAt, Bob, Park]$. This

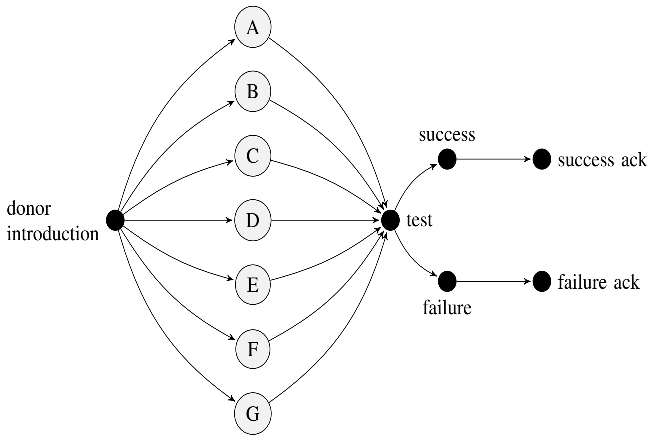

Fig. 2. Structure of a Propp Function

encoding, while retaining the properties of STRIPS, allows for greater flexibility in the definition of predicates.

B. Plot Generation Layer: The Process

Before any generation process can take place, a human author is required to input the Knowledge Base. Rather than describing all its components here, each of its elements will be described as it takes part in the process.

1) Scene modeling: The first main item that is required is a set of scenes, each accompanied by a probabilistic graph describing how it can develop. In this graph, nodes correspond to story events and edges correspond to causal relations, as from the Fabula Model [21].

While developing this system, the set of scenes used was borrowed from the formal description of Russian folktales given by Propp. In his original work, what we call *scenes* here, are given the name of *functions*. The same setting provided a classification of characters that fits well with the selection of scenes. More details on how characters become part of the story will be given in Subsection C.

As an example of one of these scenes, as it is modelled in the Knowledge Base as a probabilistic graph, we describe here the *FirstFunction-Reaction* scene, which groups together two of the functions described by Propp: the *First Function of the Donor*, and *Hero's Reaction*. In these, the Hero meets another character and is subjected to some kind of test, resulting in either a success or a failure.

In Figure 2 black dots represent a single node. The labels suggest the event associated with each node. Nodes A to G represent longer, linear paths comprising multiple nodes. Each of these corresponds to one of the possible developments of the scene as defined by Propp. For the sake of brevity, we only describe two of them here:

- The test consists of a difficult task to be performed by the Hero.
- The Hero is attacked without provocation. He "passes" the test if he emerges victorious.

While any set of scenes can be defined and used, the one defined by Propp has some well-known properties:

- They are focused on a *Hero* character. This makes it easier to ensure that the user will always have a main character to identify with.
- They contain several moments of struggle, that are easier to associate to exer-games or activities that require user interaction as they involve more action.
- They always follow each other in the same order. Clearly some variability is allowed, otherwise, every story would be exceedingly similar, but one of the points made by Propp is that many folktales share the same scene structure, and that it is, therefore, possible to obtain interesting and varied stories without major changes to this underlying sequence of scenes.

This main scene structure is therefore taken as a starting point upon which each actual story is built. It corresponds to a story in which villainy is committed, the hero learns of it and departs to avenge or repair the misdeed. He then meets a character that tests him and eventually becomes a helper. He then moves to meet the villain and defeats him. The tale always ends well with the hero returning and being rewarded, usually getting married in the process. Propp mapped this to 31 scenes. This set was simplified to 17 for the purpose of this work.

2) Scene Plot Generation: The generation algorithm follows the same steps for each scene, generating its Plot before moving onto the next, starting from the first one.

First, the *Scene Plot*, as the sequence of events that make up a scene is generated, via a form of graph traversal on the probabilistic graph associated with it. The traversal algorithm uses a probabilistic policy that favours nodes that have been used less in the current story, as well as in all the other stories generated before. This is in order to ensure variability within a story and across multiple ones.

The traversal algorithm is also what ensures that the final story will include the set of events specified in the input, being the set of events that will be substituted by activities to be presented to the user. During the generation of each scene, the algorithm ensures that the scene will contain all the nodes specified in input that are part of the graph structure associated with it. This is achieved by using backtracking: every time a Scene Plot is produced, the algorithm determines whether the corresponding path includes the nodes that it should. If it does not, it backtracks to the last choice point and tries to generate a new one. Since each scene is modelled on a connected graph, and since this is performed only on those scenes that include the node in question somewhere, the algorithm is guaranteed to terminate, producing a valid Scene Plot.

While traversing the graph, the Plot State data structure is used to ensure the enforcement of any consistency rule defined in the Knowledge base. This is achieved by allowing a set of preconditions and post-conditions to be associated with any node of any probabilistic graph. Before incorporating a node in any Scene Plot, its preconditions are evaluated against the Plot State. If they evaluate to false, the node cannot be used. Also, as soon as a node becomes part of a Scene Plot, its post-conditions are applied to the Plot State.

This means that at each step in the traversal algorithm, the State is updated, and those nodes that are affected by this update are enabled or disabled depending on their preconditions. In turn, this ensures that no consistency rule/constraint is ever violated. The fact that the Plot State is updated after each step means that consistency rules can take into consideration events that belong to the same scene, as well as events from different scenes.

An example of a consistency rule like these is the following: in some stories, the Hero is wounded in combat. This can cause a permanent scar or a bodily mark of some description. Another event that can happen (in the context of the current Knowledge Base) is that a False Hero can pose as the Hero and claim his rightful reward. The Hero can then try to prove his identity thanks to his bodily mark.

A consistency rule can ensure that the hero cannot prove his identity this way unless he actually has the scar in question. This would be achieved by giving a post-condition to the node corresponding to the event in which the hero is scarred, such as the predicate $[hero\_mark, true]$, while also using the same predicate as the precondition to the event in which he is recognised thanks to said scar. This would only allow the “recognition by scar” event if the “scarring” event has happened previously.

Once the graph traversal algorithm reaches a node with no successors, it terminates, yielding the exploration path as result. This sequence of nodes, corresponding to a sequence of events, becomes the Scene Plot of the current scene.

3) Impasses and Recursive Subplots: Once a Scene Plot is generated, it is necessary to determine which scene is to follow next. This can be the next scene in line from the sequence the algorithm was previously working on, or it can be a new scene, from a new sequence.

This is determined by scanning the generated Scene Plot, searching for one or more *Impasses*. Impasses are conditions that would prevent the current narrative flow to continue any further. An example of this would be a case in which the Hero, while trying to reach some location, is unable to find his way to it. This procedure of searching for Impasses is performed by a specialized component, the *ImpasseScanner*.

The definition of impasses used by the ImpasseScanner is part of the Knowledge Base. They can be defined simply as story events, meaning “if this event occurs, then declare an Impasse”, or they can be defined as logic conditions on the Plot State. If the conditions evaluate to true, then an Impasse is declared.

If no Impasse is detected, it means that the narration can continue along its previous heading, and so the algorithm steps forward in the current scene sequence, generating another scene.

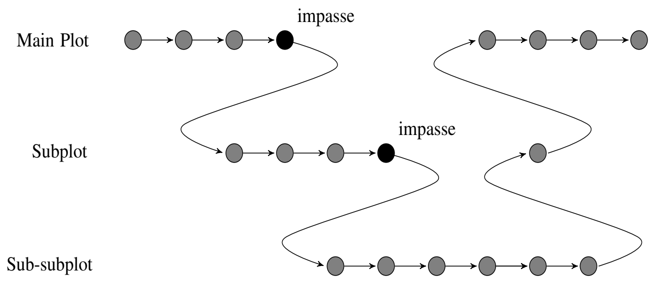

Fig. 3. Recursive Subplots

If instead, the ImpasseScanner has detected an impasse, it relays the relevant information to another component, the *ImpasseHandler*. This module determines a new sequence of scenes to be followed by the algorithm, with the final goal of removing the obstacle represented by the Impasse, and allowing the resumption of the generation of the original sequence. These sequences, introduced to overcome obstacles to the narration that emerge during the generation process, are called *Subplots*.

Going back to the previous example of a character not finding his way to the desired location, a suitable sequence of scenes that can remove the problem would be one in which he meets another character that can potentially help him, and then try again.

Clearly, it is possible that the second attempt at overcoming the obstacle leads to another failure, or even that during the development of the subplot another Impasse arises. In these cases, the ImpasseHandler will recursively introduce other “sub-subplots” as necessary (Figure 3).

When a subplot is generated in its entirety without any Impasse, the algorithm resumes from the previous sequence of scenes. When the main sequence terminates, the Plot Generation process is complete. The plot returned is the concatenation of the Scene Plot of all the scenes generated.

The recursion introduced by this algorithm is not guaranteed, per se, to terminate. In theory, the story generation process could encounter Impasse after Impasse requiring an infinite amount of subplots. However, such an event would require the same events to repeat infinite times, in order to generate the same impasses over and over again. This is prevented by the graph traversal policy explained above, as it makes the event that already occurred less and less likely the more they occur. As a consequence, the generation process is guaranteed to terminate in a finite time. Fine-tuning the exploration policy can result in plots that are more or less linear, with a greater or lesser “depth” of subplots.

C. Text Generation Layer

Once the Plot of the story is generated, it is passed to the *Text Generation Layer* (Figure 4). This component transforms the plot into readable text, enriching the semantic information associated with events through the use of a set of dictionaries. It also provides linking to the *Story Space*, the setting in which

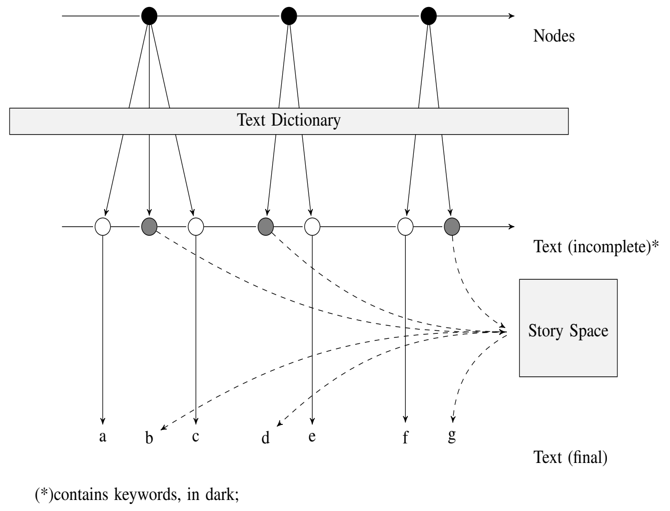

Fig. 4. The Text Generation Process

the narrated events take place [28]. This includes who the characters are, where they are, and what are the objects they interact with. In narratology, characters, items, and locations are collectively known as *Existents* [29].

The option of encoding semantic information within the structure of a node, or in the probabilistic graphs mentioned above was discarded, for two different reasons:

- Using external dictionaries makes the association dynamic, allowing for changes in the semantic definition of events at runtime.
- The definition of graph structures to be used in Plot Generation is already a demanding task, even ignoring the semantic significance of nodes.

As a consequence, the association of semantic meaning to events is done entirely during Text Generation. The input of this procedure is a Plot, as produced by the Plot Generation Layer. This data structure is an ordered sequence graph nodes, each of which represents an event.

The Text Generation algorithm translates each event into a short piece of text, over a series of successive steps. Aside from implementation details, the two main logical steps are the following:

1) In the first main step each event is substituted by a piece of *incomplete* text, picked at random from a dictionary in which each event is associated with a set of these texts. As an example, in the Knowledge Based used for the development of this system, the event node labeled "fight" is associated to "HERO and VILLAIN engage in combat", "HERO attacks VILLAIN", "VILLAIN attacks HERO", "VILLAIN and HERO fight each other"...
   These texts contain keywords (in *All Caps*) that mark where an existent should be, while its precise identity is still undefined. Existents have been grouped into categories, some of which are inspired by Propp's classification of characters. These categories are: *Hero*, *Villain*, *Donor*, *Dispatcher*, *False hero* (borrowed directly from Propp's work), *Victim* (all those characters who can be hurt by the Villain), *Location*, *Item* (not considered by Propp due to not being characters, are nonetheless necessary when classifying existents).
2) In a successive step, keywords that refer to an existent generically are substituted by actual existents. The keyword "HERO" can, for instance, become a specific heroic character, such as "the white knight". This decision has to be taken in a consistent manner: when a keyword is substituted, sometimes a new existent is to be introduced, while in other cases it should be substituted with an existent introduced earlier.

Consistency in the choice of existent is ensured by a data structure similar to the Plot State introduced above: the *Text Generation State* (TGS, in short). The structure itself is the same, but its usage is different: rather than storing information regarding properties of the current state it stores information about past choices of existents, to ensure that a new existent for the same role does not get picked again. Every time a keyword is encountered, the system checks in this data structure whether that existent has already been encountered and picked from the dictionaries. If it has, it uses it to substitute the keyword. If it has not, it selects a new one. Once an existent is chosen, a predicate <*existentClass, existentName*> encodes this decision in the data structure.

To give an example, let us assume that the first event of the plot gets associated (during step 1) to the text "HERO lived in LOCATION". The system then searches in the TGS structure, which is empty at this point, for a predicate in the form <*HERO, _*>. Not finding it, it selects a new hero from the appropriate dictionary, let us say "Ivan the hunter". The algorithm then stores this decision in the TGS, as the predicate <*HERO, Ivan the hunter*>, a similar process would happen for the "Location" keyword, resulting in the line "Ivan the hunter lived at the forest". From this point on, every time the system encounters the "HERO" keyword, it will substitute it with the appropriate character.

While every story has only one hero, it can have multiple locations: this is why the system does not use a single TGS structure, but rather a stack of them: every time the story moves into a subplot, a new TGS object is pushed onto the stack, initialized with a subset of the predicates contained in the previous top of the stack. This subset only includes those existents that will remain consistent throughout the story such as heroes and villains. Other existents, such as items or locations, are filtered out and do not pass through to subplots, so that new instances of their classes can be introduced. As soon as the subplot is resolved, the two TGS at the top of the stack are merged together, while predicates introduced in the most recent one overwrite the older ones. The resulting TGS remains at the top of the stack for the continuation of the current plot/subplot.

Going back to the example, when a subplot is introduced the <*HERO, Ivan the hunter*> predicate will pass through while <*LOCATION, the forest*> will not. If a new location is encountered during the subplot, it will then overwrite the old one as the subplot terminates, becoming the new location at which the action takes place. If no new locations appear in the subplot, the previous choice will remain as soon as the subplot terminates, since it was kept in the second TGS in the stack.

III. Discussion

Our approach to automatic story generation presents both advantages and disadvantages; in this section, we are going to briefly discuss what are the main strengths and weaknesses of the methodology presented and propose some ideas for further improvements.

A. Advantages and Disadvantages

One of the main characteristics of our tool is that all the human authorial work necessary to build the knowledge base required by the system to work, is moved at setup time, before the generation starts. After that, the system is able to generate several stories without any further human interaction as in the case of other approaches to automated story generation [25][30][31].

To the best of the authors' knowledge, this approach distinguishes itself for being one of the few in which multiple levels of abstraction are considered, especially among those trying to achieve a similar degree of automation [30][31][32]; this characteristic together with a high degree of modularity, makes the system flexible, customizable, and expandable.

Considering the serious gaming domain, the possibility to specify, at configuration time, a set of events that must happen in the story, or even the amount of repetitions of the same event makes the proposed system especially suitable for the considered field of application. For instance, in rehabilitative exer-gaming sessions in which some exer-games must be eventually played by the patient as specified by the therapists, mapping exercises to these fixed events, will make the system try to build a believable story around them, integrating them in the plot, therefore possibly boosting user immersion.

Even though the human-interaction part has been moved at setup time, the most evident limitation of the presented system is still the amount of work required prior to the generation process.

Indeed a human author needs to define:

- A set of scenes composing the main plot and a set of scene sequences describing subplots. The system can work with any group of scenes, currently, the set defined by Propp is being used.
- A probabilistic graph structure for each scene defining the events that can take place. Again, in the current implementation, these are based on the definition of scenes given by Propp.
- Proper logic rules to ensure consistency within a scene and between scenes.
- Impasses, and the subplots to be introduced to solve them.
- A Text Dictionary, containing the semantic information to be associated with node labels.
- Dictionaries of existents, to be used in the linking of the plot with the Story Space.

Of these, the design and input of graph structures are the most demanding tasks.

This step requires to define how a scene can develop, encode this information into a graph with the appropriate language, and define the opportune rules to assure consistency in the results.

In order to obtain a high-quality output a lot of effort is required in the authorial phase. The system can generate stories only inside the space given in the knowledge base, so the higher is the quality of the input, the higher will be quality of the generated stories.

B. Further improvements

Before being able to use the system in a real-world application few improvements are needed, especially in the Text Generation phase. Currently, the system is able to add semantic information to a story structure, but it could hardly generate any text that could be considered engaging by the final user, as fluent narrative text generation still appears to be an open research problem [33][34].

A workaround to this issue that could be applied to the generation of stories for video-game based applications, could be to mix up simple texts with visual media, for example procedurally generated images, composed by fetching assets from a database of sprites and animated in a way that correctly portraits the event.

Assessing the quality of generated stories, in terms of users engagement and appreciation would be a fundamental step in validating this tool, and crowdsourcing could be used for that for instance by:

1) Selecting a set of real-world tales, decomposed them by hand into sequences of the same texts defined in the text dictionary. In this way, they will be indistinguishable from generated stories in terms of language. Whatever differences in structure and existents remain.
2) Using the generator, generate a set of stories of the same size.
3) Composing a set of existents, containing some from the real-world stories and others that did not appear in them. Ensuring that enough foreign existents are part of the set will make real stories less recognisable.
4) Scrambling existents in the two groups of stories.
5) Presenting a sample containing both groups, shuffled, to a crowd of sufficient size. Ask the subjects to estimate how much each story seems "real" or "fake", or "interesting" and why.

This process could help in finding possible differences between the structure of the real stories and the generated ones, while differences in language and existents should have been already removed by the translation and scrambling steps.

To remove these differences it would be possible to iteratively repeat the process, and refine the generator at the end of each step until an acceptable level of difference is reached.

Another potential further improvement to the system could consist in finding a way to reduce the manual work required in the definition of the graph structure that describes scenes, by trying to automate the process.

For example, the system could automatically learn to adapt itself to a new dataset of stories without requiring human interaction. Unfortunately, in this case, the main complication would be the lack of suitable datasets, this could be overcome by exploiting crowdsourcing.

IV. Conclusion

In this paper, we presented a new approach for Constrained Story Generation, based on a two-layered architecture.

The Plot Generation layer works on two levels of abstraction in parallel: human-authored probabilistic graphs model scenes, while a recursive algorithm determines the sequence of scenes to be generated, introducing new subplots to resolve impasses in the generation process.

An encoding of the State of the generation, based on principles borrowed from classic AI (STRIPS) and logic programming, is used to ensure the adherence to constraints and specifications regarding the output.

The output of this layer is used by a Text Generation layer, which re-introduces semantic meaning using a human authored dictionary and a stack-like structure to model the State of the text generation, in order to properly substitute keywords with the name of existent.

From preliminary results, the narrative generated, and the system that generates it, show the following properties:

The narrative is coherent. This is ensured by the mechanics in place that allow for the specification of coherence and consistency rules.

It contains Struggle, as the system generates narrative from a knowledge base given as input, it is sufficient to define such knowledge base in a way that produces struggle in the output. The knowledge base currently in use is based on Propp's formalisation of Russian folktales, so that the struggle between hero and villain, and the series of obstacles that the hero has to overcome, that is one of their defining factors reflects in the output.

It is focused on a single character. Again, a property of the knowledge base chosen, as most real-world folktales focus on a Hero character.

The system is expandable: it is currently a proof-of-concept, but it has been designed in a highly modular way that allows for easier enhancement, employing a system of content-providing dictionaries that can easily be interchanged or developed further.

The system is programmable: the knowledge base of the system is completely modular and can be changed at will: changing graph structures will change how the same scenes develop, changing the sequences of scenes used will change the general narrative structure, changing the dictionaries allows for the changing of text, language, and setting. Most importantly, the system allows for a set of story events to be specified before the generation of the story, and ensures that these events will appear in the final output.

Acknowledgments

This work is partially supported by the European H2020 project Movecare grant ICT-26-2016b – GA 732158.

References

[1] Sigrun Gudmundsdottir. "The narrative nature of pedagogical content knowledge." In: (1991).

[2] Marsha Rossiter. "Narrative and Stories in Adult Teaching and Learning. ERIC Digest." In: (2002).

[3] Roy F Baumeister and Leonard S Newman. "How stories make sense of personal experiences: Motives that shape autobiographical narratives". In: *Personality and Social Psychology Bulletin* 20.6 (1994), pp. 676–690.

[4] Noor Shaker, Julian Togelius, and Mark J Nelson. *Procedural content generation in games*. Springer, 2016.

[5] Mark Hendrikx et al. "Procedural content generation for games: A survey". In: *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* 9.1 (2013), p. 1.

[6] James Richard Meehan. *The metanovel: writing stories by computer.* Tech. rep. YALE UNIV NEW HAVEN CONN DEPT OF COMPUTER SCIENCE, 1976.

[7] Michael Lebowitz. "Story-telling as planning and learning". In: *Poetics* 14.6 (1985), pp. 483–502.

[8] Jonathan P Rowe et al. "STORYEVAL: An Empirical Evaluation Framework for Narrative Generation." In: *AAAI Spring Symposium: Intelligent Narrative Technologies II*. 2009, pp. 103–110.

[9] Alessandro Peretti et al. "Telerehabilitation: Review of the State-of-the-Art and Areas of Application". In: *JMIR rehabilitation and assistive technologies* 4.2 (2017).

[10] Michela Agostini et al. "Telerehabilitation and recovery of motor function: a systematic review and meta-analysis". In: *Journal of telemedicine and telecare* 21.4 (2015), pp. 202–213.

[11] Yoonsin Oh and Stephen Yang. "Defining exergames & exergaming". In: *Proceedings of Meaningful Play* (2010), pp. 1–17.

[12] Jose-Antonio Lozano-Quilis et al. "Virtual rehabilitation for multiple sclerosis using a kinect-based system: randomized controlled trial". In: *JMIR serious games* 2.2 (2014).

[13] Nunzio Alberto Borghese et al. "An integrated low-cost system for at-home rehabilitation". In: *Virtual Systems and Multimedia (VSMM), 2012 18th International Conference on*. IEEE. 2012, pp. 553–556.

[14] Johanna Jonsdottir et al. “Serious games for arm rehabilitation of persons with multiple sclerosis. A randomized controlled pilot study”. In: *Multiple sclerosis and related disorders* 19 (2018), pp. 25–29.

[15] Raph Koster. *Theory of fun for game design*. " O’Reilly Media, Inc.", 2013.

[16] Richard M Ryan and Edward L Deci. “Intrinsic and extrinsic motivations: Classic definitions and new directions”. In: *Contemporary educational psychology* 25.1 (2000), pp. 54–67.

[17] Richard M Ryan and Edward L Deci. “Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being.” In: *American psychologist* 55.1 (2000), p. 68.

[18] Melanie C Green, Timothy C Brock, and Geoff F Kaufman. “Understanding media enjoyment: The role of transportation into narrative worlds”. In: *Communication Theory* 14.4 (2004), pp. 311–327.

[19] Penelope Sweetser and Peta Wyeth. “GameFlow: a model for evaluating player enjoyment in games”. In: *Computers in Entertainment (CIE)* 3.3 (2005), pp. 3–3.

[20] V Ja Propp. *Morfologija skazki [Morphology of the Tale]. Leningrad, ACADEMIA, 1928. 152 p.*

[21] Ivo Swartjes and Mariët Theune. “A fabula model for emergent narrative”. In: *International Conference on Technologies for Interactive Digital Storytelling and Entertainment*. Springer. 2006, pp. 49–60.

[22] Mei Si, Stacy C Marsella, and David V Pynadath. “THESPIAN: An Architecture for Interactive Pedagogical Drama.” In: (2005).

[23] T. Lubart. “How can computers be partners in the creative process: Classification and commentary on the Special Issue”. In: (2005).

[24] N. Negroponte. *Soft Architecture Machines*. MIT Press, 1975.

[25] B. Kybartas and R. Bidarra. “A Survey on Story Generation Techniques for Authoring Computational Narratives”. In: (2017).

[26] K. Kukkonen. “”Plot”, in The Living Handbook of Narratology”. In: (2014).

[27] R. Fikes and N. Nilsson. “STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving”. In: (1971).

[28] M. L. Ryan. “”Space”, in The Living Handbook of Narratology”. In: (2014).

[29] S. Chatman. “Story and Discourse”. In: (1980).

[30] Reid Swanson and Andrew S Gordon. “Say anything: A massively collaborative open domain story writing companion”. In: *Joint International Conference on Interactive Digital Storytelling*. Springer. 2008, pp. 32–40.

[31] Boyang Li, Stephen Lee-Urban, and Mark Riedl. “Crowdsourcing interactive fiction games.” In: *FDG*. Citeseer. 2013, pp. 431–432.

[32] Mei Si, Stacy C Marsella, and David V Pynadath. “THESPIAN: An Architecture for Interactive Pedagogical Drama.” In: *AIED*. 2005, pp. 595–602.

[33] Charles B Callaway and James C Lester. “Narrative prose generation”. In: *Artificial Intelligence* 139.2 (2002), pp. 213–252.

[34] Albert Gatt and Emiel Krahmer. “Survey of the State of the Art in Natural Language Generation: Core tasks, applications and evaluation”. In: *Journal of Artificial Intelligence Research* 61 (2018), pp. 65–170.