Università degli Studi di Milano

Facoltà di Scienze e Tecnologie

Corso di Laurea in Informatica

ARTIFICIAL INTELLIGENCE BASED NARRATION ENGINE TO IMPROVE ENGAGEMENT AND TO SUPPORT LONG TERM EXERGAMING.

Relatore: Prof. Alberto Borghese

Correlatori: Dr. Jacopo Essenziale, Dr. Renato Mainetti

Autore:
Riccardo Cantoni
Matricola: 901564

Anno Accademico 2017-2018

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Goals and Applications

Today, Narrative is produced and consumed through a variety of mediums, ranging from spoken words to videogames, from paper books to virtual reality. While it is usually considered to be a form of entertainment, it is, and has always been, employed to achieve other goals.

A long standing example is the field of Education: for centuries folktales have been used as a means to convey teachings from one generation to another [1]. More recently systematic studies of pedagogy allowed for more complex ways of employing Narrative in teaching [2]. The benefits given by the employment of Narrative in Education are multiple: stories can inherently increase appeal and engagement, while showing a certain cultural transcendence [3].

Also, even more recently, the concept of Serious Games draws a new connection between the narrative-rich world of video games and the field of education.

The field of Medicine is an other example of the use of narrative in other contexts: as a means to facilitate all the inherently necessary human interactions [4], as well as for more technical applications in psychological therapy [5].

This work focuses on the task of generating Narrative *procedurally*, via an algorithm. This would have numerous advantages, while also introducing new challenges.

Procedural Content Generation is the *algorithmic creation of content with limited or indirect user input* [6]. It is a concept often applied and originating from the field of games, but with implications spanning any other area in which content can be generated automatically.

The main advantages of generating contents procedurally include an increased efficiency of the content generation process (not requiring humans, or requiring less human work), an increased volume of content, as well as an increase in variety in the produced contents [6].

A context that would benefit strongly from a system capable of generating stories procedurally is one in which a large volume of material needs to be generated, while ensuring the adherence of every story produced to certain constraints. In cases like this, a single human author would not be capable of maintaining the production rates required, while a large group of human writers would be difficult to coordinate so that the output is always acceptable.

An automated procedure, however, would be able to produce material at a much faster rate, while guaranteeing output validity if designed correctly. Also it could be constructed in such a way that allows for the tailoring of each story in response to a set of specified parameters.

The quality of the material produced poses a difficult challenge: how does one capture the essence of a "beautiful" story in a way that allows its formalisation into a piece of code? this is an open problem which sees many attempted solutions, but no final answer (see section 1.2).

From the analysis of literature done within this work, two projects emerged, that offer the possibility of employing procedurally generated narrative in a very useful manner: enhancing and enriching systems aimed at improving the health, autonomy and quality of life of elderly people and other groups of patients.

In the next sections these two project will be briefly presented, in order to explain how procedural narrative can be integrated in their architecture and how it could be useful.

1.1.1 MoveCare Project

MoveCare [7] is a three-year EU-backed project, aiming to help elderly people to remain autonomous and continue living in their own homes, instead of needing to move to a nursery home.

The project is aimed at elderly people who are classifiable as Pre-Frail. Frailty is an ageing-related syndrome, common in geriatric patients, accompanied by a high risk of a disastrous decline in the overall health of a person [8]. Pre-frail people are vulnerable to becoming frail, if their cognitive and physical decline, boosted by ageing, continues.

The project tries to prevent or offset the transition from pre-frail to frail in elderly people, by promoting different kinds of activities in their day-to-day life and monitoring their status.

"MoveCare integrates an existing robotic platform with a domotic system, smart objects, a virtual community and an activity center, to provide, through artificial intelligence, assistance, activities and transparent monitoring to the elder at home. MoveCare can be tailored to each elder thanks to a modular design. It is completely unobtrusive as MoveCare does not require the elder to wear any particular device."

In the context of the project a hierarchical platform is being developed and tested, which integrates an array of different elements and functionalities:

- *Activity Center*: the component offers the elder a variety of activities, and supports him or her in their performing, designed to exercise physical, cognitive and social skills.
  Motivation, engagement and longevity of the experience are to be reinforced through the means of a gamification system and automatic narration techniques.

- *Virtual Community*: the elders are put in contact with each other and with caregivers, through this community system, in order to promote socialisation and avoid isolation, which is a major factor in the degradation of cognitive abilities.

- *Virtual Caregiver*: this component aids clinicians and caregivers in assisting the elder, while supporting and automating some features of the assistance activity. It collects and processes data using machine learning and artificial intelligence techniques, suggests activities through the Activity Center, while adjusting them to better suit the physiological and cognitive status of the elder, provides assistance in finding lost objects and in following a diet or a therapy.
  The Virtual Caregiver also provides caregivers and clinicians with data pertaining the status of the elder.

- Assistive Robot: the Giraff [9] robot is employed as an active companion to the elder, capable of autonomous navigation. It is used to provide companionship and feedback, while also being a valuable tool to render assistance in cases of need.

- Sensorised Objects: common items of everyday's life, ranging from pens and glasses to walking sticks are sensorised to gather data about the activities of the elder, while monitoring more clinical parameters such as gait stability and handling pressure. These data are combined together and are used by clinicians, caregivers and other MoveCare components, to monitor variations in the health status of elder, as well as to suggest specific beneficial activities.

- Domotics: domotic sensors, voice analysis and other data gathering systems provide measurements useful for the early detection of physical and cognitive decline.

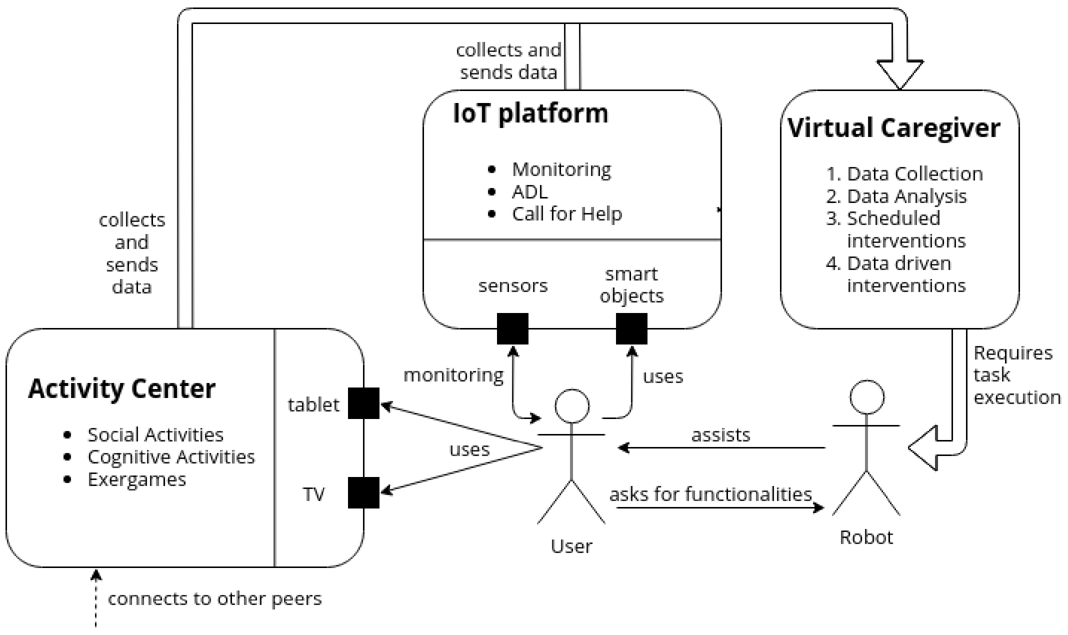


Figure 1.1: Architecture of MoveCare [10]

1.1.2 REWIRE Project

REWIRE [11] is a EU-financed project. It “develops, integrates and field tests an innovative virtual reality based rehabilitation platform system based on a multi-level rehabilitation platform.”.

Its aim is to provide patients that have already been discharged from hospitals with a means to continue the needed rehabilitation programme at home, with minimal effort and at minimal costs. Its architecture encompasses three main components:

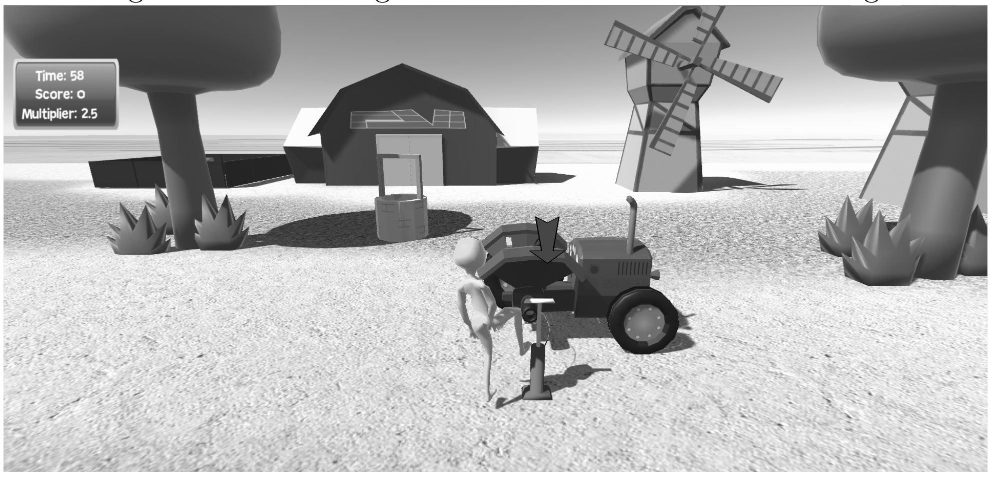
- *Hospital Station*: is used at hospitals by clinicians. It allows them to define, schedule and tune the rehabilitation programme to be followed by each patient. It also receives and aggregates data produced by the patient so that clinicians can monitor his or her status and better refine the specific therapy.

- *Patient Station*: is the means through which a patient can follow a rehabilitation programme, by playing a series of *Exergames* on a TV set. Patients are automatically guided by the system through these activities in order to prevent risks and maladaptation. Data pertaining to the motion of a patient are gathered during the activities.

- *Networking Station*: allows for collection, storing and analysis of data, making them available to clinicians and researchers for study.

Exergames are videogames that require physical exercise. They have been marketed for entertainment since the 1980’s but have enjoyed a wider diffusion only in the last decade, due to major advancements in motion tracking technology [12]. Other than in the field of entertainment, exergames have been employed successfully in the field of Medicine, especially as tools for data gathering and the measurement of health parameters [13], for rehabilitation [14] [15] and for therapy [16].

Figure 1.2: An Exergame for the rehabilitation of the legs



In the case of REWIRE, the patient carries out a therapeutic task in the game world through his or her avatar. Motion is recorded through devices such as Microsoft's *Kinect* and the balance board *TYMO* by Tyromotion, and is translated into movements of the avatar.

These games are the means through which patients carry out their rehabilitation activity. Each game is designed to respond to a specific therapeutic need, such as exercising specific muscles in a specific way, with a focus on whether the movement is carried out precisely in the correct way [15].

They are meant to make a repetitive, therapeutic task less abstract, by building an entertaining interface around it.

1.1.3 Gamification For Long-term Engagement

The matter of long-term motivation and engagement is crucial to projects of this kind: users, or patients, should keep interacting with these systems for prolonged periods of time. In the case of REWIRE, patients might have to follow a long rehabilitation period of several months, while as for MoveCare, elders should conceivably make use of the system for the entirety of their third age.

Since the effectiveness of these systems depends on the user's continuous participation in the activities, it is necessary to develop solutions to reinforce its interest in them. This lasting motivation becomes therefore a factor of major importance in the lasting effectiveness of the service offered.

The range and variety of activities available to these users is forcibly limited. Especially in the context of physical rehabilitation, each user is presumed to repeat the same limited set of exercises several times.

The task of maintaining long-term engagement is particularly challenging under these conditions, because of the inherently repetitive nature of some activities, together with the desired time span of their usage.

The ways in which motivation is increased have been subject of extensive studies in the field of social psychology. A distinction is made between *intrinsic*, or *internal*, and *extrinsic*, or *external* motivation.

The difference lies in where the motivation comes from: in the case of intrinsic motivation, the drive comes from within the activity itself, and the interest or satisfaction that derive from it [17].

In the case of extrinsic motivation, instead, external elements promote a behaviour indirectly, by giving additional goals that require said behaviour to be reached. This way, one sees the activities a means to an other end, rather than a driving force in itself [17].

For instance, a student could be motivated in studying with commitment because of an interest in the subject or the sense of pleasure and reward coming from the activity itself (intrinsic motivation). Or, the motivation could come from the desire of avoiding punishment, or obtaining some other reward, or from a sense of duty (extrinsic motivation).

Studies show that both types of motivation can be effective in reinforcing a desirable behaviour. However, extrinsic motivation can in some cases be have a detrimental effect, as individuals perceive it as externally regulated or alien to the activity [18] [19].

This limit is not shared by intrinsic motivation [19].

An approach that has been used to address the issue of engagement is the implementation of Gamification techniques. Gamification is defined as "using Game Design elements in non-gaming contexts" [20].

Gamification works by introducing elements reinforcing extrinsic motivation, especially to activities that struggle in generating intrinsic motivation, perhaps because of their repetitiveness or their lack of appeal [21].

Studies have shown how gamification can be effective in reinforcing desirable behaviours and the volume of user activity [22] in many fields of application, including:

- Learning: gaming elements can be added to pre-existing learning environments, to improve their effectiveness. Similarities between the process of learning a complex subject and playing a game facilitate this application. Students learn how to solve complex problems, unveiling more and more information as the game progresses towards a clearly defined goal [23].

  Also, elements that are typical of the field of gaming, such as the concept of *Flow* (A mental state of complete immersion and concentration upon a specific task) have proven to be successful in enhancing the effectiveness of learning systems[23]. Competition, progressive rewards, and tools for socialization have also been applied with positive results [24].

- Crowdsourcing: the practice of externalisation of parts of the processes of an enterprise is more and more common. Gamification methods are used to increase user engagement, support user activities and reinforce useful user behaviours [25]. Even monetary rewards have been introduced.

- Physical exercise: gamification is employed to increase adherence and user commitment [26].

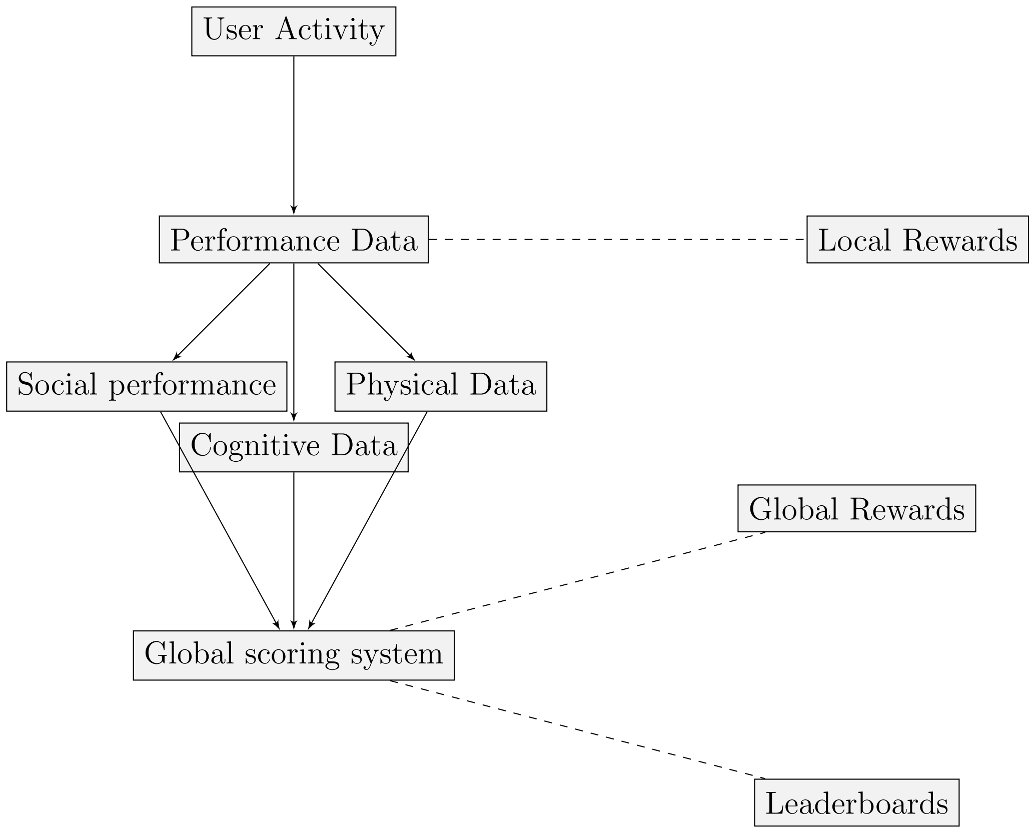
- User Engagement: studies show that implementing gamification techniques in various contexts increases the volume of user activity, even after a time span of years [27].

  Goals of this approach include increasing the volume of trade on e-commerce websites [28], and engaging the employees of an enterprise while simultaneously reinforcing positive behaviours and increasing their productivity [29].

Gamification is currently being designed to be integrated into MoveCare to respond to the specific issue of user engagement. The system is currently undergoing development.

However, a generic structure has been already laid down.

Figure 1.3: Gamification system in MoveCare



Performance of MoveCare users in the various activities are collected and used in a local system of rewards. Within this system, users who perform well unlock new features at the application level.

Performance data are then aggregated in three main areas: Social, Physical and Cognitive, following the three main focuses of MoveCare. This aggregated data is then further processed into a global scoring system.

Users who perform well at the global level access global rewards. Also, a leaderboard system allows them to compare their performance and score with that of other users, fostering competition.

1.1.4 Automatic Narration

While the employment of gamification can be effective in reinforcing engagement, other options are available.

In particular, gamification works by adding extrinsic motivation to an activity. This, as stated in the last section, is not always effective, and can be even detrimental in some cases.

Automatically generated narrative can be employed as a tool to strengthen and reinforce the engagement and commitment of users. Integrating activities in a narrative structure would be a means to increase intrinsic motivation [30], by making the participation of a user more interesting and rewarding [31].

The goal of this work is to produce the means to employ automatically generated narrative to reinforce the engagement and commitment of users, in the context of the aforementioned projects MoveCare and REWIRE.

This involves experimenting with Story Generation, in order to design a system capable of encasing different activities into a narrative superstructure.

The concept of an exergame, especially, shows great potential in becoming an element within a story. Exergames are, in essence, simple scenes in which the user is the main character who is requested to perform a certain simple task, through the medium of a digital avatar. A careful design of the setting of an exergame, coupled with the appropriate narrative superstructure, allows it to become a section in a sequence of scenes constituting a narrative.

The task becomes then to generate a coherent, and possibly interesting, story from which individual scenes or sequences can then be mapped on exergames.

On the subject of coherence, a coherent story is really just a story that adheres to a set of predefined rules. Whatever architecture one decides to use, it must be defined in a way that only produces valid stories: a story in which the Prince *eats* the Princess is, under normal circumstances, not valid, while one in which the Prince *marries* the Princess is acceptable.

Clearly, the more constrains are required to define the specifics of desired stories, the more rules will need to be in place. This will impact negatively on the possible variety of outputs, by taking away elements from the output space of all possible stories. The issue of ensuring sufficient variety needs careful consideration.

Clearly, a complete story is made of a variety of highly differentiated scenes. The degree of variety in the set of exergames scheduled for a user to perform is much more limited: in fact, the number of physical exercises designed for the rehabilitation of a certain patient, or the activities made available in a health care system, is forcibly reduced. Moreover, it is not possible to map every conceivable story scene into a game or activity.

Also, the developing of new games and activities is a costly endeavour which is outside the scope of this work. Any procedure for the generation of narrative will therefore have to face severe limitations in the amount of story elements that can be mapped into user interactions.

One obvious path would be to restrict the set of scenes that can be used to compose a story so that it coincides with the set of available activities. This is, however, highly impractical: there are only so many ways in which to organise a small set of sequences until the overall narrative degrades into meaningless repetition.

A more sensible approach is to accept that not every story element will be translated into an activity: some scenes and events can be in fact delivered to the user through other mediums, being initially written text, and later perhaps images or even animations.

The final story would therefore appear to the user as a sequence of scenes in which activities with him or her as the protagonist are interleaved with text, images and such.

Still, the problem of variety remains. Given that activities constitute fixed scenes that are always the same, whatever story is going to be generated shall be strictly constrained by the necessity of incorporating these rigid elements. A way to address this issue is to build exergames in such a way that allows for the replacement of assets in the game scene at runtime. This way, the same activity could be dressed in different ways to increase variety, without incurring in the higher costs of the development of a new one.

For example, the same activity of moving one's body to avoid obstacles that are in the path of the avatar, can once become a scene in which the avatar is moving on a horse through a forest, and once one in which it is crossing a body of water on a boat. The mechanics would remain, and the activity itself would have the same effect on the health of the user. While the comparatively simple operation of replacing the assets would allow for the scene to be recycled in various narrative contexts.

This would in turn increase the size of the output space. The generator, under the same circumstances, would not be forced to only generate stories in which the hero travels through a forest, but could also generate some in which he travels on the ocean, or through a city, or a desert, and so on. This would allow for more varied and interesting stories.

1.1.5 Requirements

Java was chosen as programming language to be used in this work. This ensures that it remains reasonably compatible with the systems explained above, as well as being easy to deploy and maintain.

The objective is to outline an architecture capable of generating narrative. The resulting stories should adhere to a set of constrains, defined to ease their applicability as means to improve engagement. They should:

- **Be coherent**: Coherence is a requisite for acceptance. Valid stories "make sense". They exhibit consistency and reliability, while character behaviour and events are consistent with the setting, without extreme deviations.

- **Contain struggle**: moments of struggle are central in the dramatic structure of any narrative. They are necessary to provide the audience with a focus for their interest. Moreover, moments of struggle, seen as problems a character needs to overcome, represent a prime opportunity for the introduction of user activities, in which the user himself takes part in the resolution of the problem.

- **Focus on a single character**: The applications of this system are in single-user contexts. Generating stories that have a single character as protagonist, around whom the entire plot revolves, makes it easier to associate this single user to the main character, so that him or her becomes directly the acting subject of any activity.

Given the desired properties that apply to the output, there are necessities pertaining to the system itself. Features of the generation architecture include:

- **Expandability**: while the purpose of this work is to outline the architecture of a Narration Engine, the prototype will have to be designed in a way that allows for its further development into a more complete piece of software, following the Software Engineering principle of ”Design-For-Change” [32].

- **Customisability and Modularity**: the software must be built in a modular way, that allows for quick redefining and replacement of some of its functional parts in order to alter its behaviour. Requirements about the output can be changed or simply be defined more finely.

  A monolithic architecture would prove extremely expensive to adapt to the evolution of requirements, as well as to tailor to the specific needs of any real-world application.

- **Programmability**: the system must allow for the specification of parameters for the generation process. The most prominent of these parameters is the definition of items that have to be part of the output. Since some story elements will be associated to real-world activities that carry a therapeutic meaning, a way is needed to specify whether some story elements must be present in the final story.

1.2 Structure of the Thesis

Here is outlined the structure of this work:

- In chapter 2, the state of the art is reviewed: how human and machine can cooperate in the production of narrative, how the different levels of automation are classified, what level of automation is going to be achieved with the approach proposed.

  Also, two sources will be explain in greater depth, being the theoretical bases on which the algorithmic structure proposed in this work is built upon.

- Chapter 3 describes a first attempt at a narrative generation structure. The approach is explained, as well as the problems and obstacle that arose and eventually caused the abandonment of it.

- The general architecture of the generation system is described in chapter four. What modules have been implemented, a recount of their function and how they interact with each other.

- Chapters 5 and 6 describe how the system works, and offer some insight into how it was implemented. The first focuses on how the plot is generated, while the second on how said plot is transformed into readable text.

- Chapter 7 is the conclusion: how the proposed approach is different from those present in literature, what are its strengths and limits. Also the last steps necessary in order to produce a deployable prototype are outlines, as well as suggestions on future works of refinement and improvement.

Chapter 2

State Of The Art

2.1 The Mixed Initiative Approach

Computers are often used as a means to design, produce, share, visualize and consume narratives of various description. The degree of involvement of the machine in the life cycle of a narrative product can vary substantially.

There is a contribution, albeit minimal, even when a text editor is used to write a physical book. At the other end of the spectrum some forms of narrative entertainment, such as videogames, animation or films, require a more direct participation of the machine, starting from the generation process all the way to the moment of its consumption by the public.

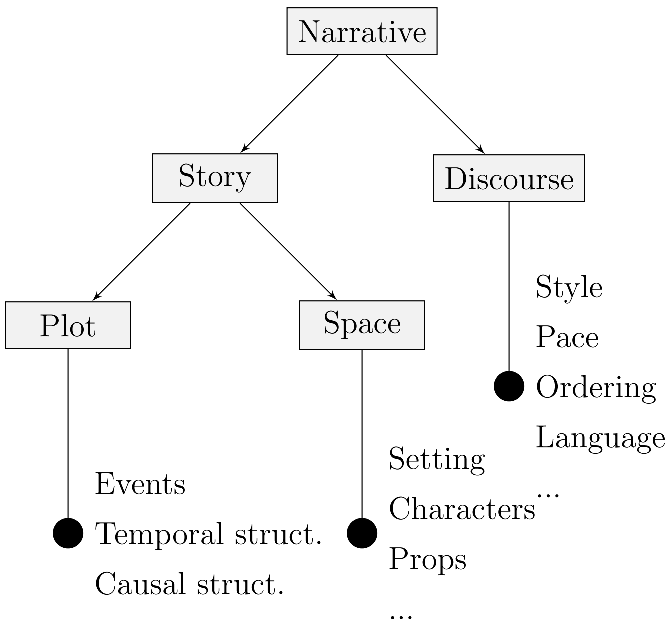
The subject of this work has to do with the generation part of the process.

A computer cannot be the sole author of a creative work, but rather it needs the human factor to produce meaningful narrative [33]. Any procedural generation algorithm aiming to produce narrative needs inputs that can only come from humans and their endeavours. This necessary cooperation and interdependence between human and machine in a creative process is usually referred to as the *Mixed Initiative* approach [34], in which both agents participate varyingly.

Before discussing the different degrees of automation that can be applied to the components of Narrative, it is necessary to define them precisely. A formal definition of the structure of a narrative work is given by Kybartas and Bidarra [35]:

- *Narrative* is defined by *Story* and *Discourse* [36].
- *Story* is the content of the narrative: the events that constitute it, the *Plot* and the setting in which they happen, the *Space.*
- *Plot* is the collection of all the events that take place in the story [37]. These are organised in a temporal ordering, and are linked by causal relations.
- *Space* includes the physical setting, the characters, objects, props, and anything present in or affected by the events described by the *Plot* [38]. Characters, objects and locations in which the story unfolds are usually grouped under the term *Existents* [39].
- *Discourse* is the specific way in which the story is recounted: the style of the narration, the ordering of the events, the pace, the linguistic choices.

Figure 2.1: Structure of Narrative Components



The generation of Discourse falls outside of the scope of this work. It is a complex and challenging task, for which different approaches have been proposed. Generation based on a formal decomposition of Discourse [40], systems focusing on cinematic discourse [41], and on style [42], are among the most prominent.

The focus of this work is the generation of Plot and Space. Depending on the level of automation desired in the generation of these elements, different problems and challenges have to be undertaken, and the properties of the final result change.

As for the generation of Plot and Space, five degrees of automation are identifiable[35].

Plot Automation:

1. *Manual:* the contribution of the computer is non existent or minimal. The human author produces the entirety of the Plot.

2. *Structure:* the computer generates the structure of the Plot, while the human authors events and ordering. The human is basically given a set of generated constraints within which he operates in autonomy.

3. *Template:* the computer generates the Plot, complete with events and ordering. The human participant provides the linkage to the Space, by instantiating setting and existents.

4. *Constrained:* The entirety of the plot is generated procedurally, and the linkage to the Space is also provided by the machine. However, the generated Plot follows structural constrains authored by a human.

5. *Automated:* The human contribution is minimal.

Space Automation:

1. *Manual:* the structure and content of the Space is entirely human-made.
2. *Modification:* a human authors the contents of the space entirely, but the machine can change part of it automatically, trying to improve the quality of the final result.
3. *Simulation:* interactions between elements of the Space can be simulated, and become the initial state of the story.
4. *Constrained:* the contents of the space can be generated by the computer, following a set of given rules and constraints authored by humans. these can include certain specific existents that must appear in the story.
5. *Automated:* The human contribution is minimal.

2.2 Constrained Plot, Constrained Space

In order to define how this work is going to proceed, it is mandatory to identify the specific level of automation required in both Plot and Space generation.

Constrained Plot: The system will have to be able to generate the entire plot procedurally. However, it will be forced to follow a pre-defined structure, a set of constraints given by the human author, as well as allow for the adjustment of parameters that influence the generation process.

Constrained Space: A human will generically define the possible contents of the space, in a way that enforces certain rules, such as the role of existents. The narrative engine will then compose the actual story space, and link it appropriately to the Plot.

In literature, most attempts at automated story generation focus on the generation of the Plot, while the Space is static. However, there are a few examples in which systems have been developed which generate constrained Plot and Space.

Crowdsourcing has been employed as a method to support the generation of narrative: Swanson and Gordon's *Say Anything* [43] system uses a large corpus of stories collected from internet blogs to give a knowledge base to the computer. A human and a machine then take turns, writing one sentence each, in a collaborative process that generates the story. The sentences returned by the computer are picked from the story corpus through a complex system of semantic matching. This approach clearly can not be adapted to the purpose of this work, since it requires a human to be present during the generation process. Also, the generated stories show a lack of consistency that is not compatible with the requirements in place.

Scheherazade [44] is an other system that relies on crowdsourcing to generate narrative: a group of anonymous writers are asked to write a short story on a given subject. The resulting set of narratives is manually translated into a graph-like structure, which is then employed as the structure for an interactive game in which the player is presented with multiple choice decisions which determine the development of the story.

An attempt was made to follow a similar approach within the scope of this work, without success (see chapter 3).

An other example is the *Thespian* [45] system, designed as a tool to support the writing of dramatic literature. The human user defines goals and motives of characters, and the system simulates character interactions, yielding control to the human when the resulting behaviours seem problematic. This tool serves more as a support for the production of literary material rather than a system that can produce many instances of narrative. It also requires a human in the production cycle, which is something that is not available.

Of all the scientific literature on the subject, two items have been used as the main theoretical base of this work. The relevant material is described in the next two sections.

2.3 Swartjes and Theune's Fabula Model

Ivo Swartjes and Mariet Theune's work, *A Fabula Model for Emergent Narrative* [46], presents a model to describe the Fabula of a narrative, proposing its employment in the generation of a story emerging from on the simulation of an environment. Before describing the subject, it is necessary to introduce the terminology in use. These definitions are borrowed from the field of narratology [47]:

- *Fabula:* "is an account of what happens in the story world and why". It encompasses all the events that take place, as well as the causal relations between them. It can be seen as a network of events and facts, linked by causation.
- *Plot:* is a part of the Fabula. It comprises a subset of the events of the Fabula, while still constituting a coherent whole. Consequently, the same Fabula can lay the base for multiple plots, each of them encompassing a different coherent subset of events.
- *Presentation:* "is the information needed for the actual presentation of the plot in a certain medium". Being the events of a Plot, the relevant causal links, an appropriate ordering and so on.

A definition of Plot had already been given in section 1.2 as, in short, the ordered set of events that constitute a story. The definitions are similar but not equivalent. To avoid ambiguity, the last definition given will only apply in this section, while in the rest of this text the first one will.

The article gives a formal ontological structure of a Fabula that is based on a previous work by Tom Trabasso et al. [48], describing a *General Transition Network* (GTN) model. This model defines a network of elements grouped in six classes, connected by links that express causal relations between them.

The Fabula model groups story elements in six classes as well, similarly to the GTN model. However, the classes are defined differently. Here follow the definition of classes given in the context of the Fabula Model.

- *Goal* (G). Goals are driving factors behind the actions of any character, they represent the desire of a character to attain or avoid a certain condition.
- *Action* (A). Any change to the story world, intentionally induced by a character in the pursuit of its goals.
- *Outcome* (O). Every Goal has an associated Outcome. When a character realises (or believes) to have fulfilled its goal, we have a positive Outcome, otherwise a negative one.
- *Event* (E). Any change to the story world, not resulting from an intentional action of any character.
- *Perception* (P). Anything that a character perceives and / or acknowledges.
- *Internal Elements* (IE). Anything that happens within a character: emotions, beliefs, feelings.

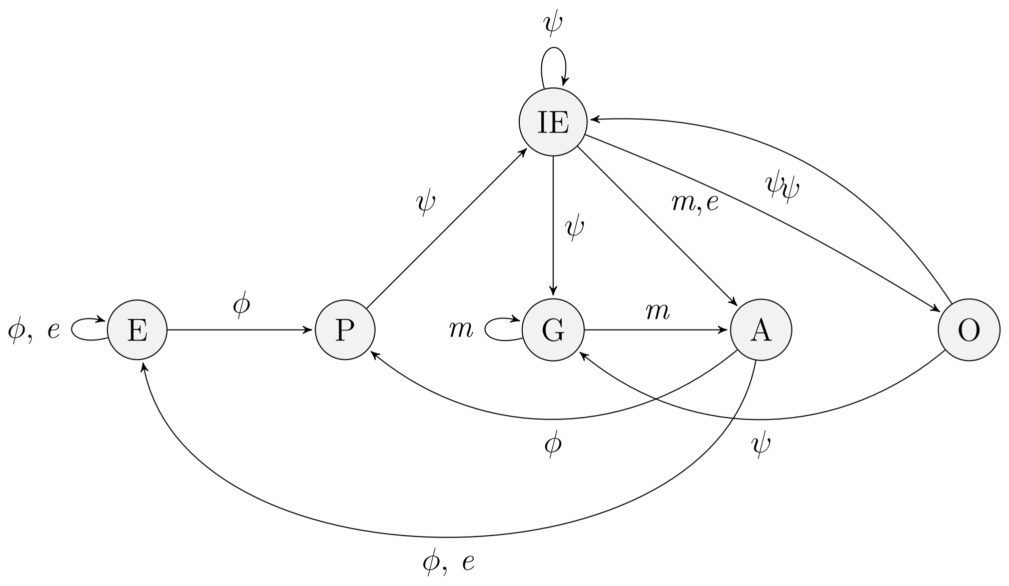
Given the six classes of story elements that take part in the fabula network, the causal links are also grouped in four classes:

- *Physical* (ϕ). Events and Actions can physically cause something else to happen. An intentional Action can cause an Event, an Event can cause an other Event, such as a storm causing the sinking of a ship. Both Events and Action can cause Perceptions, when a character perceives them or their results.
- *Motivation* (m). This is a type of causality that originates from within a character. A Goal can motivate an other Goal (also called *subgoal*). A Goal can clearly motivate an Action, as well as an Internal Element can motivate an Action.
- *Phsychological causality* (ψ). Similarly to the previous case, this kind of causality stems from the mind of a character. The difference between psychological causality and motivation is the fact that the latter is intentional, while the first isn't. An example of this kind of causality is a character being enraged by the fact that an other character, friend to it, has been murdered.

- *Enablement* (e). The broadest kind of causality, it occurs when a story element satisfies the preconditions to an other element.

The network of classes and causal relations can be described with a graph-like structure.

Figure 2.2: Graph of the Fabula Model



The usefulness of this ontology is that it provides a single, simple, unifying language with which to describe a Fabula and, given that a Plot is a subset of a Fabula, a Plot. It already suggests a digital representation as a graph-like data structure, and enriches the way of looking at a Plot as a whole, not only as a sequence of events, but rather as a sequence of *causally related* events.

Also, the classification of story elements provided can be a first step towards their differentiation necessary to associate them with different methods of delivery to the audience (see section 1.1.3). The causal links can, instead, be useful for the generation of Discourse, by providing connectives between the description of related events.

The ways in which this model has been incorporated in the designing and construction of the Narration Engine will be described in the next chapters.

2.4 Vladimir Propp's *Morphology of the Folktale*

Vladimir Propp's seminal work *Morphology of the Folktale* [49] dates back to 1928. However it was largely ignored in the West until it was translated decades later. It is considered to be one of the most important works in the study of Folklore, as well as in the field of Morphology.

Propp relies heavily on a large collection of folktales from traditional Russian folklore, *Russian Fairy Tales*, composed by Aleksandr Afanas'ev and originally published in Russia in 1855. It is a collection of nearly 600 tales, which Propp uses as a dataset to base his analysis on.

Morphology of the Folktale describes the results of a broad, systematic, scientific analysis of the structure of Russian folk tales. It focuses on the narrative structure of tales, and the classification of characters induced by their role in the story. Also it proposes an analysis of the Theme of folktales, they way in which they are combined together in tradition, and their intended meaning.

The work is broad and rather lengthy. Here, only the contents relevant to this work will be highlighted.

Propp notices how different characters in different tales often perform the same actions, to the point where in some case they might even be interchangeable without changing the general flow of the tale. This highlights the existence of a fixed structure within different tales, which would therefore contain recurring, constant elements. These elements are called *Functions*. As a consequence of his analysis, Propp states that:

1. "Functions of characters serve as stable, constant elements in a tale, independent of how and by whom they are fulfilled. They constitute the fundamental components of a tale."
2. "The number of functions known to the fairy tale is limited."
3. "The sequence of functions is always identical. "

4. “All fairy tales are of one type in regard to their structure.”

This sequence of Functions then constitutes a constant skeleton, upon which every tale is constructed. They are defined as:

0. Initial situation: not properly classified as a function, it is still part of every tale. In it the setting is presented, and some of the characters are briefly introduced.
1. *Absentation*: One of the members of a family, or group, absents himself from home.
2. *Interdiction*: The Hero is presented with an interdiction.
3. *Violation*: The interdiction is violated.
4. *Reconnaissance*: The Villain attempts to gather information. Information about the hero, his family, the location of an object and so on.
5. *Delivery*: The Villain receives information
6. *Trickery*: The villain attempts to deceive a victim. He or she tries to gain possession of the victim directly or of some of his/her belongings via trickery.
7. *Complicity*: The victim unwillingly helps the Villain.
8. *Villainy*: The Villain harms someone belonging to the community.
9. *Lack*: One member of the community desires or lacks something.
10. *Mediation*: The lack or misfortune is communicated to the Hero. He is commanded, requested or allowed to go, or simply presented with the problem.
11. *Counteraction*: The Hero accepts or decides to act.
12. *Departure*: the Hero departs.
13. *First Function of the Donor*: The Hero is tested, potentially allowing him to receive help in the form of a magical agent or a helping character. The test can take many forms, such as an attack, an interrogation, a request for help and so on.

14. *Hero's Reaction*: The Hero reacts to the challenge posed to him by the Donor's test. He either fails or succeeds.
15. *Receipt of a Magical Agent*: The Hero acquires the use of a magical agent. This can be an object a helpful creature.
16. *Guidance*: The Hero is transported or led to the object of his search. Thanks to a variety of means.
17. *Struggle*: Hero and Villain fight each other.
18. *Branding*: The Hero is branded.
19. *Victory*: The villain is defeated by the Hero.
20. *Liquidation*: The original misfortune or lack is liquidated.
21. *Return*: The Hero returns to his place of origin.
22. *Pursued*: A pursuer follows the Hero while he is returning.
23. *Rescue*: The Hero escapes his pursuer.
24. *Unrecognised arrival*: The Hero returns home, or reaches an other land, without being recognised.
25. *Unfounded Claims*: A false hero poses as the hero to gain a prize in his place.
26. *Difficult task*: The hero is presented with some kind of ordeal, or test.
27. *Solution*: The test is resolved.
28. *Recognition*: The hero's identity is confirmed.
29. *Exposure*: The identity of the False Hero is confirmed. He is exposed as an impostor.
30. *Transfiguration*: The Hero changes appearance. In some instances, the lifestyle of the Hero is changed instead.
31. *Punisment*: The Villain is punished.
32. *Marriage*: The Hero marries and / or ascends to the throne.

Given the definition of these Functions, the author observes how they can be grouped in *spheres of action*, groups in which certain characters, or *Dramatis personae*, how they are commonly defined in narratology, is always performing the action that is being recounted. Each Function has therefore a strong focus on one single specific *dramatis persona*.

Seven spheres are identified, leading to the definition of seven distinct dramatis personae:

- *Hero*: The main character of the tale, he departs on a quest to liquidate some lack, meets other characters, is tested, fights Villains and is often rewarded for defeating them.
- *Villain*: He who causes harm to someone, and is then sought after, fought and possibly defeated and punished.
- *Donor*: The Donor is the character who provides the hero with a magical agent.
- *Helper*: He transfers the Hero to its destination, helps him in the passing of a test, rescues him from pursuit or directly liquidates a lack.
- *Princess*: The person who is being sought for, often associated to her father or a similar figure, can assign tests, brand the hero, recognise him, punish villains and dispense rewards.
- *Dispatcher*: He who announces the lack or misfortune to the Hero, and dispatches him.
- *False Hero*: He claims the reward that should be justly given to the Hero, and is sometimes confronted, exposed and punished.

The strong claim presented here is that this classification of characters, and the character-function mapping is fixed and constant among all tales.

This allows for a very simple way of defining the skeleton of a story, by defining the functions that constitute it, and mapping the relevant characters to each of them.

An other interesting result described by Propp is that, not only characters are on the scene during the same group of functions in every tale, but also are introduced to the read in the same ways, and in the same moments.

- Hero, Princess and Dispatcher are always presented in the introductory scene.
- The Donor is met accidentally by the Hero, while he is in his quest.
- The Helper appears as a gift or reward to the Hero, after the passing of a test.
- The Villain appears suddenly from outside, committing his villainy. He then disappears until the Hero has reached him.
- The False Hero is sometimes introduced at the beginning of the tale, together with Hero, Princess and Dispatcher. Alternatively he is only made known when he claims the Hero's reward in his stead.

These are only a few of the results presented by Propp in his book, but are the ones that are more relevant to this work. They highlight a simple structure to a tale, almost rigid in its definition, which constitutes a very strong and useful base upon which to build a generation algorithm.

It is also important to note that these results come from the analysis of a dataset containing only tales from the traditional folklore of Russia. Russian folktales show particular promise in being used as a base for a Narration Engine, because they are particularly hero-centric, and therefore adapt well to a context in which a single user is at the center of the story.

The ways in which these concepts have been used in the construction of the Engine are explained in the next chapters.

Chapter 3

A First Attempt

An attempt was made at designing a generation algorithm based on the Fabula Model described in section 2.3. This path proved to be unfruitful.

The general procedure was to start from real-world tales, as the Fabula Model provides a way in which to deconstruct each of them into a network of story elements belonging to the ontology described by the model.

Once a sufficient number of these networks was obtained, merging them together would have created a larger and more complex graph structure. Stories would have been generating by traversing this graph.

In this chapter are reported the details of this unsuccessful approach.

3.1 Tale Deconstruction

The first step was the deconstruction process. This procedure was to be performed manually.

Each story event in the story was to be translated into a *Story Node* in the network, classified appropriately as an element of the ontology. Clearly, the classification of elements into six groups (Goals, Actions, Outcomes, Perceptions, Events and Internal elements) is not sufficient to capture the complexity of a story. A hierarchy of classes was devised to retain as much information as possible in the decomposition process, without needing an exceedingly large amount of them.

For example, Goal elements were further divided into three groups:

- *AttainGoal*: the goal is to make a desired property true.
- *AvoidGoal*: the goal is to make a desired property false.
- *ObtainGoal*: the goal is to possess a certain item, or to hold a certain character captive.

Similar classification systems were used to further subdivide the other classes of elements in meaningful groups.

This required the design of a representation of a story State, so that properties of the current situation could be properly modeled. A system similar to that used within the STRIPS standard [50] was employed, representing a state as a collection of logic predicates, each of them expressing one property of the environment. Since this system is also part of the final architecture of the Narration Engine, it is explained in more detail in chapter 6.

This allowed for a more informative labeling of elements in the Fabula Network. Each node in the network carries a minimal description of an event, being its classification, as well as further specification in the form of one or more STRIPS predicates.

A few examples of this translation are given here.

Table 3.1: Decomposition of Story Events

Story Event	Node Classification	Specification Predicates
Little Red Riding Hood wanted to visit her gran...	Internal: Desire Goal: AttainGoal	location(girl, grans_house) location(girl, grans_house)
Suddenly, the wolf appeared beside her.	Event	location(wolf, forest)
Poor Granny did not have time to say another word, before the wolf gobbled her up!	Action: HarmAction	captivity(wolf, gran), harm(wolf, gran)
A few minutes later, Little Red Riding Hood knocked on the door.	Outcome: AttainPositive	location(girl, grans_house)
She realized that the person in the bed was not her Grandmother, but a hungry wolf.	Perception	location(wolf, grans_house)

Given a few decomposed stories, the resulting networks were observed, considering the structure of the associated graphs. It was noted that:

- The resulting graph is akin to a simple, linear, path.

  This can be explained by observing the process which led to it: the Fabula associated to the tale can be seen as a connected graph, and a subset of its nodes is taken to define the Plot, as described in the Fabula Model (section 2.3). Then, the causality links force the order in which nodes have to be recounted, inducing a fixed ordering, as stated by Propp (section 2.4). Clearly, if links are induced by causality, and causality only propagates forward in time, there can be no loop.

Then the Plot is narrated, producing the text that was used for the decomposition. The decomposition task is performed following the order of the narrated story, and therefore replicates it in the resulting graph.

As a consequence, the final result is a connected graph describing a loopless sequence of events, following an ordering induced by causality.

- There can be no branches that don't merge back into the main branch.

  There can be branches merging into the main path of the story, but there can not be paths that branch out from it. Any outwards branching would necessarily have to merge back into the main one, or it would result in a loose end in the story: a chain of events that leads to nothing.

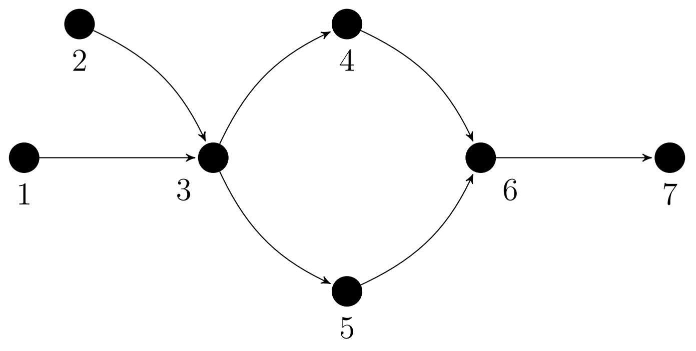
These properties are not true in general for any kind of narrative: in literature there are numerous examples in which events are not narrated following their causal ordering, and there can be loose ends to a story. There are even cases in which time is not considered to be entirely linear, possibly causing causality loops.

However, in the context of folktales generally adhering to Propp's model, the relative simplicity of the narrative structure makes these properties hold. The simplest way of recounting a story is in an ordered sequence of events, and this reflects on the shape of the resulting graph.

Below, the branching structure of a deconstructed tale is shown. The tale is Little Red Riding Hood, sourced from the *Multilingual Folk Tale Database* [51].

Each dot does not represent a single story node, but rather a linear sequence of nodes, constituting a branch in the full network.

Figure 3.1: Branching structure of Little Red Riding Hood



Each branch therefore represents a part of the whole story, as follows:

1. The main character, Little Red Riding Hood, is introduced, as well as her intentions to visit her grandmother. Her mother is also presented. Then, the girl departs from her home.
2. The Wolf is introduced.
3. The Wolf sees Little Red Riding Hood and approaches her, he asks about her, finds out what she is doing and leaves.
4. The Wolf runs to the grandmother's house through a shortcut, tricks her into opening the door, eats her and wears her clothes as disguise.
5. Little Red Riding Hood slowly reaches her grandmother's house, picking flowers along the way.
6. Little Red Riding Hood sees the disguised Wolf and is suspicious about his appearance.
7. The Wolf attempts to eat her and she screams for help. A passing hunter intervenes, saving the girl and her grandmother, and killing The Wolf.

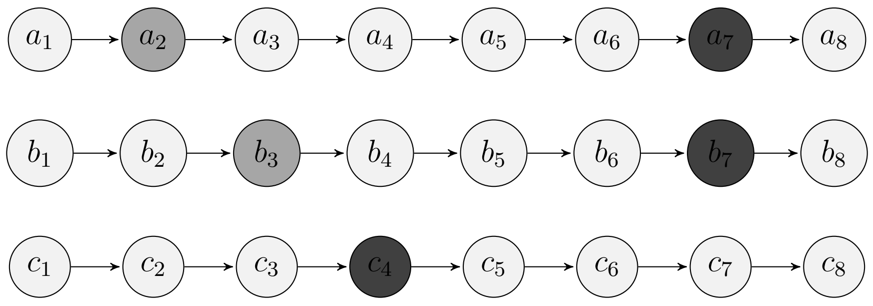
3.2 Graph Merging and Issues

Once a few stories were decomposed into sequence-like graphs, the next step was to merge them into a larger one. This was to be achieved by superimposing individual nodes respecting a certain definition of similarity.

Given two graphs of story nodes, and given a definition of similarity between two nodes, it is certainly possible to identify pairs, or sets, of similar nodes and merge the two graphs, merging those nodes together. Repeating this for several stories, hence several sequences, would yield a large interconnected graph of story nodes.

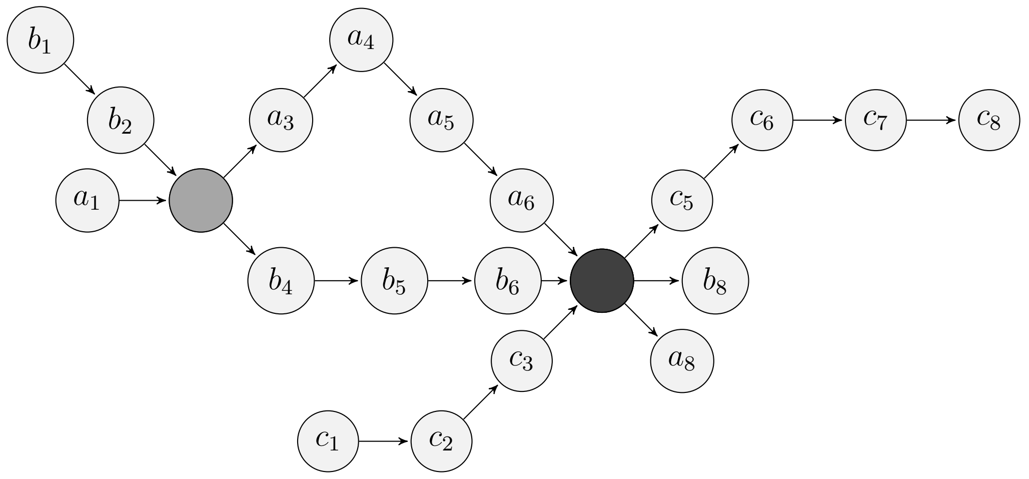
An example of such procedure is shown here. Three stories, labelled A, B and C, have been decomposed into the three sequences $a_1...a_8$, $b_1...b_8$ and $c_1...c_8$. Two sets of nodes have been identified as similar: $\{a_2, b_3\}$ and $\{a_7, b_7, c_4\}$.

Figure 3.2: Plot sequences with node similarities



The merging together of similar nodes would produce the following graph structure.

Figure 3.3: Merged Plot Graph



A graph like this could be explored in multiple ways, starting from the beginning of any story sequence and ending at any story end.

Nodes that have been merged would act as choice points between the following of different story sequences, while the exploration policy would determine which stories are included in the final path. An example of such exploration path, on the graph shown above, would be described by the sequence $b_1 ... b_3, a_3 ... a_7, c_5 ... c_8$.

The output of an exploration process would be a sequence of story nodes, constituting a new story. This new story would be a mix of real-world ones intertwined together.

The adherence to constraints could be observed by pre-processing story sequences prior to the merging, extracting the necessary information and storing them into an appropriate data structure which would be, in turn, used during the exploration process to determine which story is to be followed next.

The exploration of this graph, and the subsequent output of the corresponding exploration path, would have resulted in a new sequence of story nodes and therefore in a new story, as mix of pre-existing real world ones.

This system was not fully developed nor fully tested. A few stories were decomposed into story node sequences, but then such problems emerged that it became clear that this approach would have proved unsuccessful. These problems have to do with two phases of the process: the phase in which real-world tales are decomposed, and the phase in which the resulting structures are merged together.

- **Decomposition Phase**: in order to function effectively, the system needs to be primed with a sufficient number of decomposed tales. The more tales are available, the more varied the output will be, since the graph on which the exploration is performed would grow accordingly. Too small a number of available tales would cause the output to be extremely uniform.

  the decomposition of a tale into a sequence of story nodes proved to be an extremely intricate task: it required a sequence of complex operations to be performed on each sentence in the original tale:

  1. Each sentence has to be carefully analysed to determine its full meaning.
  2. Such meaning has to be then decomposed into individual nodes.
  3. Each one of these nodes has to be classified following a complex hierarchical system, in order to preserve the relative information.
  4. The type of causal link between each pair of nodes has to be determined. In some cases there is only one possible causality type, as from the original Fabula Model, while in other cases a choice must be taken between a few alternatives.
  5. The whole result has to be fed into a computer so that it can be processed.

  It is immediate that this process is extremely time consuming, and single person would require an exceedingly long time to decomposed a suitable number of stories. Involving multiple people to collaborate in this process was not possible in the context of this work, and it would have introduced new problems in the training of these collaborators, and the necessity of maintaining a certain uniformity in the classification, since personal interpretation is part of the task.

In literature, a few attempts have been made at automating the decomposition of narrative, one of which, based on a mix of Hidden Markov Models and Latent Semantic Analysis, was examined, since it was specifically designed to work on Russian folktales in the context of Propp's model [52]. However, the system did not meet the required level of accuracy in the decomposition, since it only focused on recognising Propp Functions.

An other issue with the decomposition process was the fine tuning of the level of accuracy of the node classification. As new stories were incorporated, lacks in the existing classification system were made evident, requiring the addition of new classes or new predicate types to correctly represent the new information. This in turn caused an explosion in the amount of classes to be used, which made the classification task even more demanding.

- **Merging Phase**: the merging of story sequences consisted in the merging of some of their story nodes. Every time two nodes are merged together, a new choice point is induced in the general graph.

  In order to have an effective system, it is necessary to generate a suitable number of these points: unless enough choices are available, the exploration process will necessarily proceed along the path of one of the original stories, without generating anything different.

  The merging, or lack thereof, of story nodes depends upon a definition of the property of similarity between nodes, based on their representation.

  Each node, as shown in table 2.1, is represented as a pair: $< Classification, Specification >$, where $Classification$ consists in a node type and optionally a node subtype, while $Specification$ in a set of logic predicates.

  Different definitions of the property were attempted: the loosest of which being a simple confrontation of type and subtype. Stricter definitions applied different logic unification models to the Specification predicates, even involving a classification of characters following Propps' model.

No definition proved to be broad enough to produce a sufficient number of choice points, while also being strict enough to preserve coherence. It is possible that, given a sufficient number of stories, it would have been easier to design an effective similarity function but, as explained above, the amount of available stories was severely limited.

An explanation of such difficulty in finding similar story nodes can be searched in the process that brought these centuries-old pieces of narrative to us.

These tales emerged from a long process of oral transmission from person to person. During this process, multiple variants of each tale emerged, and different tales were sometimes mixed together, re-elaborated, truncated. This would seem to imply the coexistence of many similar stories, caused by this chaotic process of "collaborative writing" among multiple authors and multiple generations. This would in turn imply the existence of many similar nodes belonging to similar tales.

However, between the 19th and the early 20th century, the main systemic collection works of these, previously orally transmitted, stories were published. Notable examples include *Kinder- und Hausmärchen* by the Grimm brothers (1812), and the already cited work by Aleksandr Afanas'ev (1855).

In these large collections, sets of similar tales were "collapsed" together into a single version [49], thereby filtering out most of the variants to a single theme. These collections form the basis of most folktale datasets, including those used in this work.

This might explain why it was difficult to find stories with a high degree of similarity: because most of them didn't survive the filtering process necessary in the gathering of a large amount of them in large collections.

Once this approach proved to be ineffective, an other was attempted, which instead led to the design of a working, albeit simple, system. This will be the subject of the next chapters.

Chapter 4

General Architecture

This chapter will be an overview of the general architecture of the system. Structure and function of the various elements involved will be explained more in depth in the next chapters.

This system, once developed further, is to be used to reinforce the engagement of users in a set of activities that will be proposed to them, by encasing a group of them in a narrative structure.

This requires it to produce a meaningful story and associate some of its elements to activities, specified prior to the generation.

The final output will be in the form of a .json file, carrying a mapping between pieces of texts and activities to be proposed to the user, where the activity is null for all those text pieces that are not associated to anything.

Before any generation can take place, the system has to be correctly initialised and configured. The operation of providing each module with the appropriate parameters will be explained along with the corresponding module.

In particular, one of the parameters required is the set of activities that are to be part of the output. The story output will necessarily contain them.

Once the system is ready and all parameters are set, a Plot Generation layer (chapter 4) defines a Plot, as a sequence of Story Nodes similar to those described in the last chapter.

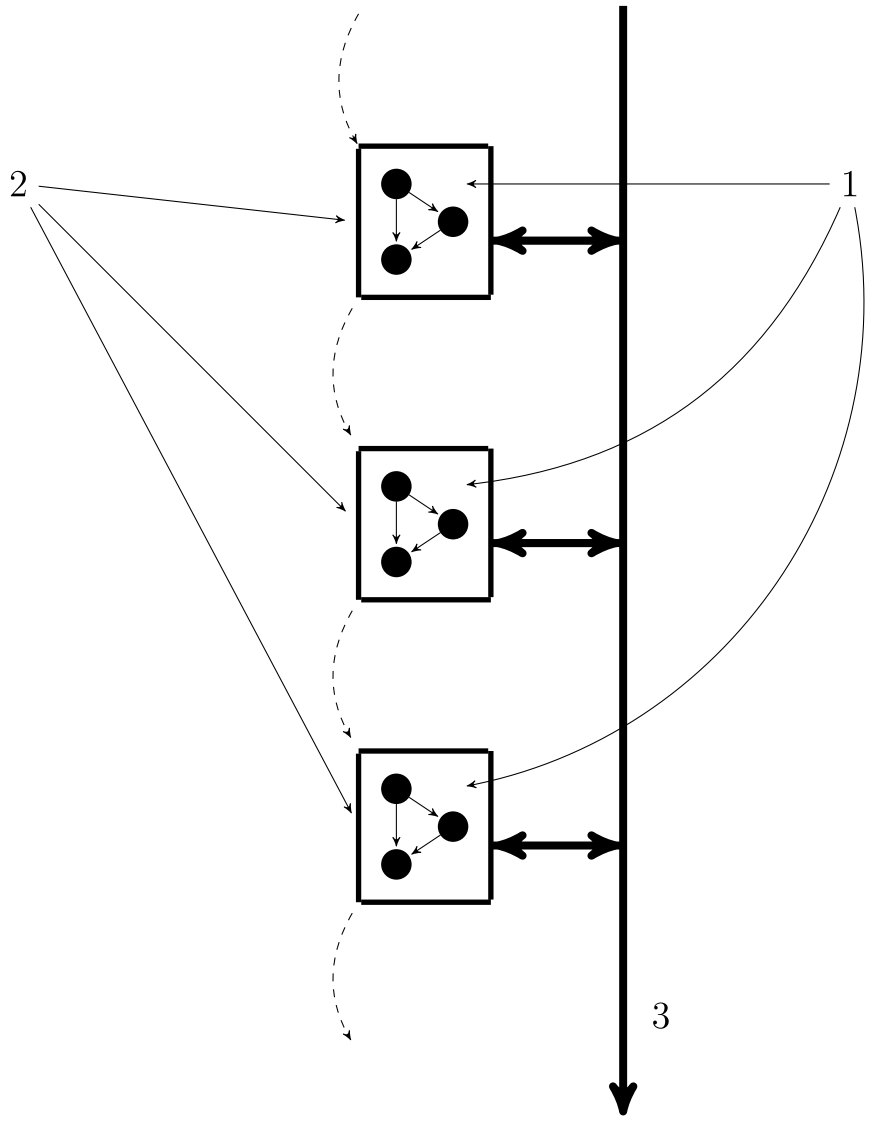
The inner workings of this layer borrow elements from both Propp's model and the Fabula Model. A sequence of Propp Functions is composed, based on an algorithm that follows a set of pre composed Plots, while recursively introducing new subplots to overcome obstacles in case they arise.

In parallel, within each Function, the ways in which that scene can develop are defined through a probabilistic graph structure. Nodes within these structures are defined according to the Fabula Mode and represent atomic story events.

A graph search algorithm then traverses the graph outputting a sequence of nodes that constitute the specific chain of events that make up the corresponding scene.

At both these levels of abstraction, the adherence to generation constraints and coherence rules are ensured through the interaction with a structure describing the State of the generation. This module is implemented employing the STRIPS standard, and it will explained further in section 5.2.1.

Figure 4.1: Architecture of the Plot Generation Layer



1. Probabilistic graphs serve as model of the possible sequences of events that constitute a scene/function. Traversing one of these graphs gives out a sequence of nodes that describes the developing of the scene.
2. Functions are chained together following recursive rules based on plots and subplots. The concatenation of the corresponding node sequences is what constitutes the final plot.

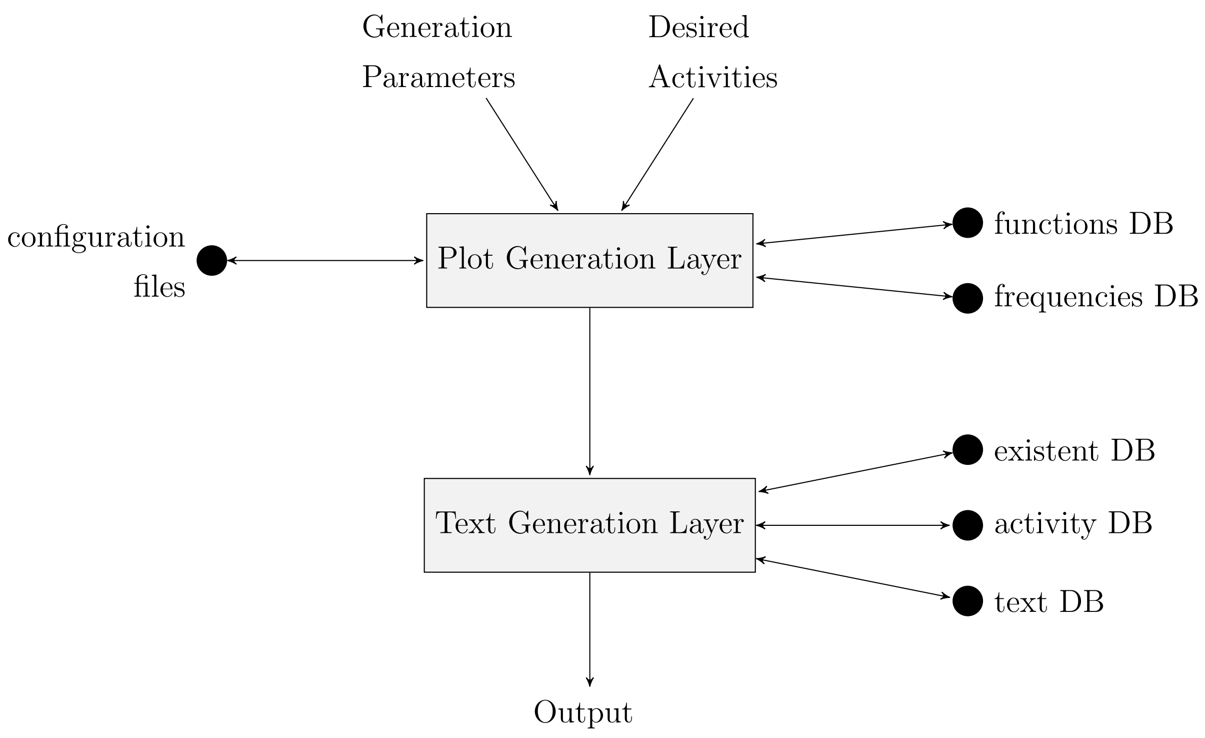
3. Both components interact with a State that propagates forward during the generation process, used to ensure adherence to constraints.

At the end of this process, a Plot is given in output, as a sequence of story nodes. This is going to be input for the second layer, that focuses on translating this plot into readable text.

This component iteratively parses the plot, associating nodes to variables, and variables to text and activities, using a set of dictionaries to process variables into appropriate, consistent, ground expressions. This process will be the subject of chapter 6.

Here is shown a schematic of the general architecture.

Figure 4.2: General Architecture



A set of database files is used by both layers of the architecture: they constitute a knowledge base, containing data necessary to the process, as well as containing information regarding past activities of the system. Their function will be described in the next chapters, along with the inner workings of the relative generation layer.

The final output is a .json file: a sequence of triplets, each containing: a node identifier, e small piece of readable text and, when such association is possible, an identifier to an activity to be proposed to the user. This structure can then be used to compose a story interweaved with user activities, delivered by whatever media are available.

Chapter 5

Plot Generation

The subject of this chapter is the first layer in the generation architecture. This component generates a Plot, given a set of user-defined parameters and the proper knowledge base.

5.1 Scenes as Propp Functions

The Plot Generation process is based on the concept of Function, as defined by Propp (section 2.4). Each function can be seen as a scene in the story: each one of these scenes can develop in different ways. Propp already gives, for each function, the different ways in which that function has been carried out in the dataset he used.

5.1.1 Modelling a Function

Each Function/scene is modeled as a probabilistic graph, in which different paths correspond to different ways in which the scene can progress. Nodes in these graphs are defined following the Fabula Model explained in section 2.3, similarly to the nodes used in the deconstruction of tales explained above.

The difference between these nodes and those explained before lies in the fact that these follow a simplified ontology. In the failed attempt at story generation explained in the previous chapters nodes were used to capture the semantic structure of a given tale.

In the original work in which the model was proposed they were being used for character simulation. In this case, however, nodes are not required to carry any semantic information, which will be provided later by an association with text.

As a consequence the nodes used contain simpler information, being the pair $< label, classification >$. The $classification$ variable identifies a class in a simplified version of the Fabula Network Model. The original model defined six classes: Actions, Goals, Perceptions, Outcomes, Internal Elements and Events. The classes Perceptions and Internal Elements have been grouped together in the Perception-Internal class. This was possible because the distinction between the two was aimed at providing a simulation step in the internal process of reasoning of a character. Given that this is no longer being done, the distinction became redundant.

The resulting data structure borrows from both Propp's model and the Fabula Model: a graph of nodes based on the ontology given within the latter constitutes one of the elements (a Function) described in the first.

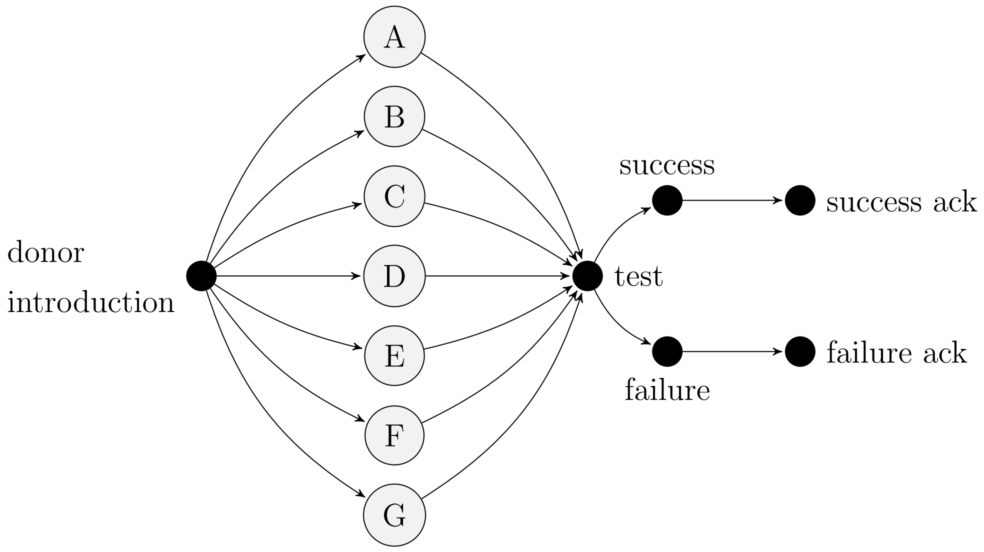
Following this model, the 31 Functions described by Propp have been grouped together into 17 graph structures, each one describing a scene. An example is given here in the form of the *FirstFunction-Reaction* Function, which groups together two of Propp's: *First Function of the Donor*, and *Hero's Reaction.* In this, the Hero meets the Donor and performs some kind of test.

In the figure, black dots represent a single node with its label, while 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, as follows:

- **A**: the test consists in a difficult task to be performed by the Hero.
- **B**: a test of knowledge. The Donor interrogates the Hero.
- **C**: the Hero finds the Donor to be prisoner of someone else. The testing lies in whether he frees him or not.
- **D**: the Hero captures the Donor. He asks for mercy.

- **E**: the Donor attacks the hero without provocation. The hero "passes" the test if he emerges victorious.
- **F**: the Donor offers the Hero an exchange, or a trade.
- **G**: the Donor is in need of help, the Hero may or may not intervene.

Figure 5.1: Structure of a Propp Function



An example of a complete path on this graph, passing through the sequence *C* is given below. For each node a brief interpretation is given for readability. This is not part of the real data structure.

Table 5.1: Node in a Scene Path

Label	Classification	Interpretation
donor_introduciton	Event	the Donor character is introduced
prisoner_ack	Perception/Internal	the Donor perceives the Hero
desire_freedom	Goal	the Donor wants to be free
freedom_request	Action	the Donor asks the Hero for help
request_ack	Perception/Internal	the Hero receives the request
test	Action	the Hero is subjected to the test (in this case freeing the Donor)
success	Outcome	the Hero is successful
success_ack	Perception/Internal	the Donor acknoweledges the result

5.1.2 Graph Traversal and Constraints

Given a properly defined graph structure, some nodes can be defined as "initial" nodes if they have an indegree of 0. Specularly, nodes with outdegree 0 are considered "final".

The generation of a scene sequence involves the finding of a path, connecting an initial node to a final one.

Still, not all paths from an initial node to a final one are necessarily valid: depending on how the graph was defined, there might be cases in which integrity constraints make a path not valid. For instance, in the *Struggle - Branding* Function, in which the Hero fights the Villain, one of the possible outcomes is that the Hero acquires an item previously in possession of the Villain. Any path in which the hero is defeated cannot therefore contain the node (or nodes) in which the Hero acquires said item.

A system is in place to support the introduction of constraints concerning the "validity" of paths: each node can optionally be associated with logic preconditions that define whether it can be included in the scene path or not. Since this system is more intensely used in the concatenation of different scenes it will be explained in the next sections.

The generation of a valid path is performed via a standard graph exploration algorithm. The graph is traversed Depth-First: at each step the preconditions of all successors to the current node are evaluated, and only those for which they are satisfied are considered in the exploration.

This effectively equals to performing the same traversal algorithm on a graph from which invalid nodes are removed completely. The output is then guaranteed not to contain any of them, therefore only producing valid paths.

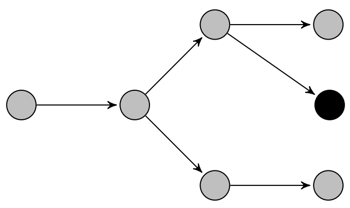
The disabling of nodes based on logic conditions is not the only available mechanic to impose constraints on the generation of a scene. A system is in place that allows for the specification of nodes that must be part of the final path.

This set of nodes must be specified before the exploration process begins. The operation of making the presence of a node mandatory is given the name of *Injection*.

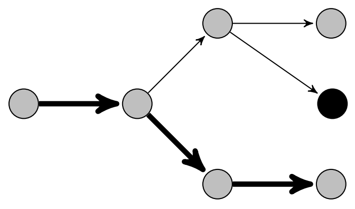
Injections are implemented by exploiting a property of the Depth-First search algorithm used to traverse the graph: if a "dead end" is reached, the process backtracks to the last choice point available.

Therefore, whenever the exploration process reaches a final node, instead of it terminating immediately, the current reverse path to the initial node is evaluated. If it is found to contain all the injected nodes, the process terminates. Otherwise it backtracks and continues.

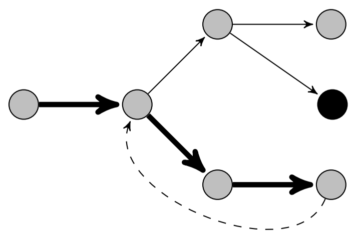
Figure 5.2: Backtracking in DFS search



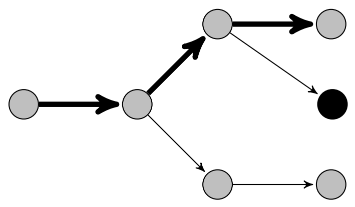
This is the graph associated to a Function. A node has been injected (in black) and must necessarily appear in the final path.



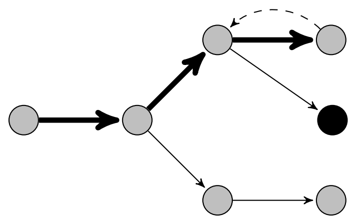
The Depth-First Search algorithm finds a possible path (highlighted) to one of the final nodes.



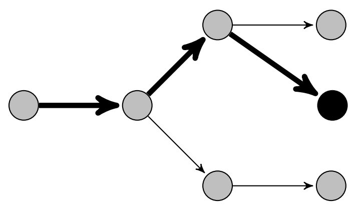
The full path does not contain the injected node.
The algorithm backtracks to the last open choice point.



DFS resumes its running, until again it reaches a final node.



Once again, the path is not valid. The algorithm backtracks.



The search finds an other complete path. This time the path is acceptable and the process terminates.

These two mechanics allow for the definition of two types of contraints: logical constraints that enable/disable nodes, and injections that force the presence of a node in the final output.

The first is used to ensure semantic consistency in the construction of a scene.

The latter is particularly useful to ensure that the generated story actually contains the set of activities that are to be presented to the final user. This is done by injecting the corresponding nodes so that they become part of the final result. This will be explained in more depth in the next sections.

The behaviour of the graph traversal algorithm can be altered and tuned by changing the policy used to select the next node to be visited. All graph traversal algorithms use a data structure called *Fringe*, or *Frontier*, which holds all the potential "next" nodes to be visited. At each step one of these is selected and the traversal proceeds from it.

Four different selection functions have been implemented and are available:

- **Pure DFS**: the next node to be explored is the furthest away from the starting node. If multiple nodes share the same distance from the origin, one of them is selected at random.
- **Pick First**: This policy is mostly for debugging purposes: a predictable version of DFS in which nodes are picked in order.

  The ordering on the nodes is defined as a side effect of how the graph structure is stored, and reflects the order in which nodes were added to the graph.
- **Random Search**: the next node is picked at random.
- **Inverse Frequency**: this is intended to be the primary selection policy to be used. The Plot Generation Layer keeps memory of how many times each node was used: both in the current story generation process (local frequency) and in all the stories generated by the system in the past (global frequency). Using this policy, a "roulette wheel" selection is used, in which the probability assigned to each node is inversely proportional to their frequency.

  The rationale between this system is that, this way, generated plots will be as variable and differentiated as possible, without being predictable.

The setting of the desired node selection policy can be done by appropriately from the "configuration.cfg" file.

Summarising, this module uses a pre-defined graph structure to generate a sequence of nodes describing a scene, belonging to a classification of scenes/functions close to that defined by Propp.

A variant of a DFS search is performed on the graph in a way that ensures adherence to constraints about consistency and content, while trying to minimise repetitivity and predictability.

5.2 Linear Plot Generation

The module explained in the previous section generates a single scene, from the corresponding probabilistic graph. In order to generate an entire Plot, it is necessary to chain together multiple scenes appropriately.

A *Linear Plot* is the simplest form of plot: a pre-defined sequence of scenes. A specific module takes such a sequence as input, and outputs the corresponding sequence of events.

This operation is carried out iteratively: each scene is processed into a node sequence describing it. At each step, the output is chained with the node sequences produced previously. Then, the next scene is processed, until the list of scenes terminates.

The output is a chain containing the nodes describing each scene, in order.

5.2.1 STRIPS State Encoding And Dependencies

What happens within a scene has repercussions on what happens in others, further down the line.

For instance, if in the scene mentioned above, corresponding to the *First Function of the Donor* function, the Hero acquires the servitude of the Donor, this allows the hero to be guided to its destination by the Donor in the *Guidance* scene.

Conversely, if the Hero had failed in acquiring the help of the Donor, the same chain of events describing how the Hero is guided by him could not be followed.

This means that it is necessary to propagate some information from within the generation of a scene to other scenes downstream. This is accomplished through the definition of a State component that receives, stores and provides access to this kind of information.

The State is modelled in a way that loosely follows the STRIPS standard [50]. STRIPS is a framework for the definition of agents for the resolution of AI problems through planning. It included the definition of a "State of the problem" structure, which was implemented exploiting mechanics typical of logic / functional programming.

Since this work was developed in Java, and since the level of flexibility required was not as high as for a generic planning agent, a simplified version of the STRIPS encoding was used to define the state of the Plot generation.

The state is a collection of objects resembling logic predicates. While logic predicates are flexible in their structure, the predicates in use here are fixed as a $< String, String >$ pair. The first parameter defines what type of information is being represented, while the second represent its content.

For example, the predicate $< test\_outcome, success >$ states that the test performed by the Hero in order to gain the help of the Donor had a positive outcome.

Nodes belonging to any function can be associated with pre and postconditions regarding the state: in order for a node to be valid and usable, the preconditions must be respected. If a node is actually used in the scene, its postconditions are applied to the state.

Pre and postconditions take the form of logic expressions on predicates. An ad-hoc group of classes has been implemented to imitate basic mechanics of a logic programming language, including predicate matching, full first-order logic expressions on the matching of predicates, and "universal unification": a value which matches any other value. By using these functionalities, one can easily specify complex dependencies between nodes, even when they belong to different functions.

As a minimal example in the following table is presented the set of conditions necessary to specify two dependencies: the Hero can be guided to his destination by the Donor only if he successfully acquired his servitude by passing a test. At the same time the Hero can fail in finding his way only if he isn't currently being helped by the Donor.

Table 5.2: Dependency Specification

Label	Interpretation	Cond. Type	Condition
success	the hero passes the test devised by the Donor	POST	$< test\_outcome, success >$
guidance	the hero is guided by the Donor	PRE	$< test\_outcome, success >$
guidance_failure	the hero fails in finding the way	PRE	$NOT < test\_outcome, success >$

As explained in section 5.1.2, during the traversal of a graph defining a function / scene, the preconditions of all nodes are evaluated, ensuring that the final sequence of nodes produced in output contains no invalid nodes. Once a node is added to the path its postconditions are put in effect, affecting the state of the generation.

5.3 Non-linear Plot Generation

A Linear Plot is clearly too limited and uninteresting: the same sequence of scenes repeats itself unchanged in each generated story.

To address this, a set of mechanics are in place to introduce a higher degree of variability in the story generation process, based on the concept of *Subplots*.

A Subplot is a new linear plot introduced within an other, to respond to a specific issue. During the generation of a Plot, some story events would prevent the storyline from continuing: for instance, if the Hero fails in finding his way to the Villain, the part of the story in which the two engage in a fight, and eventually the villain is defeated can not take place.

These events are given the name of *Impasses*. They are part of the graphs describing functions, and so they can become part of the story. Once they are, they constitute an obstacle to the linear flow of the storyline.

A subplot must be introduced in which the obstacle is removed. Continuing with the example given above, if the Hero fails in finding his way to the Villain, a subplot is required in which he resumes his travelling and eventually tries again.

5.3.1 Impasses

A specific component, the *ImpasseScanner*, can be set with two different kinds of impasse definitions. The component scans a scene path after it has been produced and before generating the next one.

- **Node labels**: if certain label is found in the path, declare an impasse.
- **State properties**: if a certain condition regarding the state is true, declare an impasse.

For instance, in order to define the impasse caused by the Hero failing at finding his way to the Villain's location, one could specify the label of the corresponding node, $guidance\_failure$, or or a logic condition which is true whenever the post-conditions of that node are true.

Once an impasse is detected by the ImpasseScanner, the generation process is paused. An other component, the *ImpasseHandler*, receives information describing the impasse and introduces a subplot in order to remove the obstacle.

5.3.2 Impasse Handling and Subplots

The role of the ImpasseHandler is to receive information about an impasse, and launch the generation of a subplot appropriate to resolving it.

This requires a set of subplots to be defined by an author, as well as a mapping between impasses and subplots. Subplots are defined as sequences of scenes / functions, while the mapping is defined directly within the ImpasseHandler component.

Returning to the same example, the *Guidance* function has been generated. The resulting sequence of nodes contains one in which the Hero fails in finding his way to the Villain. The ImpasseScanner detects this and raises an Impasse, passing it to the ImpasseHandler.

Injections can be used to force scenes to develop in a direction that is appropriate to the subplot

The handler accesses the mapping between impasses and subplots, and picks the appropriate subplot object: *RETRY_GUIDANCE*, defined as follows.

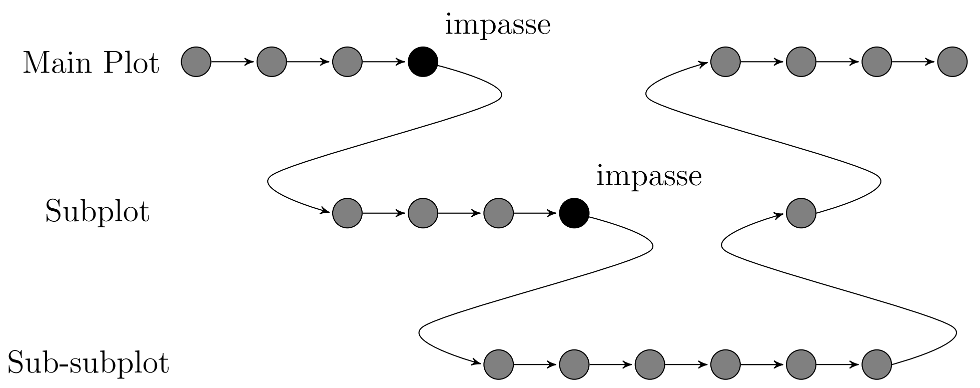
Table 5.3: The RETRY_GUIDANCE Subplot

Function	Possible Developments	Injections
Filler	Filler scene. Introduces a new subplot.	*hero_retry**
FirstFunctionReaction	The Hero meets a Donor character, is tested, and may or may not pass said test.	
Acquisition	Consequences of the test: the Donor may or may not help the Hero.	
Guidance	The Hero tries to find a way to the Villain. He can be helped by the Donor or by an item given him by the Donor.	

* this forces the graph traversal algorithm to output a scene path containing the node *hero_retry*, in which the Hero decides to make a new attempt at the task he was pursuing earlier.

It is possible that one or more impasses occur in a subplot. The whole mechanic is recursive: a subplot can be interrupted by an impasse, and an other subplot is introduced to clear it.

Figure 5.3: Recursive Subplots



One problem that is immediately apparent is that there is no guarantee that the recursion will terminate. In theory it would be possible that each successive subplot keeps running into an impasse and introducing an other and so on, in an infinitely deep nesting.

In practice this is resolved by the node selection policy used in the graph traversal procedure. As explained in 4.1.2, the default policy selects the next available node according to a probability distribution in which nodes that have already been used in the story are less likely to be picked. The probability decreases each time the node is used.

Impasses depend on the usage of specific nodes, and since every time an other level of recursion is called one of these nodes is used, the probability of impasses occurring decreases with the depth of the recursion and with the length of the story.

As a consequence, impasse-causing nodes are unlikely to be picked deep in the recursion nesting, as well as down along the story. This means that every impasse will eventually be cleared and the generation algorithm will be allowed to terminate.

Once the generation of main plot terminates, the entire Plot Generation process terminates, producing in output a sequence of nodes as the concatenation of all the function/scene paths generated, in order.

Chapter 6

Text Generation

The previous chapter explains how the Plot Generation Layer of the system generates a sequence of nodes, each of which describing a story event, linked to one another.

This sequence is the input to the Text Generation Layer, which will then process it into readable text.

The main obstacle to be overcome during this step derives from the lack of semantics in the input. This is because the nodes that compose it lack any semantic meaning. This in turn derives from the process from which they originated.

These nodes come from the graph structures described in the previous chapter. A human author has designed these graphs. Each one of them models the different ways in which a scene can unfold. This in turn means that these graphs have been designed with a semantic meaning.

However, this semantic information is not carried within the nodes, but rather in its interpretation in the mind of the human author. This interpretation is not encoded in the structure of nodes and is therefore lost as soon as the graph is defined.

The main role of this Text Generation Layer is to add again semantic value to these events, resulting in a readable, and meaningful, output.

The option of encoding the semantic meaning of each node within its structure was considered and discarded. It was discarded because of two reasons:

- It would make the association between node and semantic static. A dynamic definition makes it possible to change of the semantic information associated to nodes on demand. even at runtime.
- The definition of graph structure and the nodes they contain is already rather delicate. Requiring the definition of semantics within it would make it even more complex.

As a consequence, the definition of the semantic meaning of a node takes place after the plot sequence has been generated by the first layer.

This is achieved through the definition of *Text Dictionaries*, that associate nodes with short pieces of symbolic text. These texts are not in the final form of the output, and need to be processed further, but manage to provide semantic information to nodes. For each node, the dictionary associates its label to an array of pieces of text.

Definition and properties of these text dictionaries, and how they are used by the Text Generation module are explained in the next section.

Moreover, this layer provides linking to the Story Space (see chapter 2), by introducing existents appropriately. Ths mechanic will be the subject of section 6.1.2.

6.1 From Nodes to Text

The algorithm that processes the node sequence into readable text proceeds as follows:

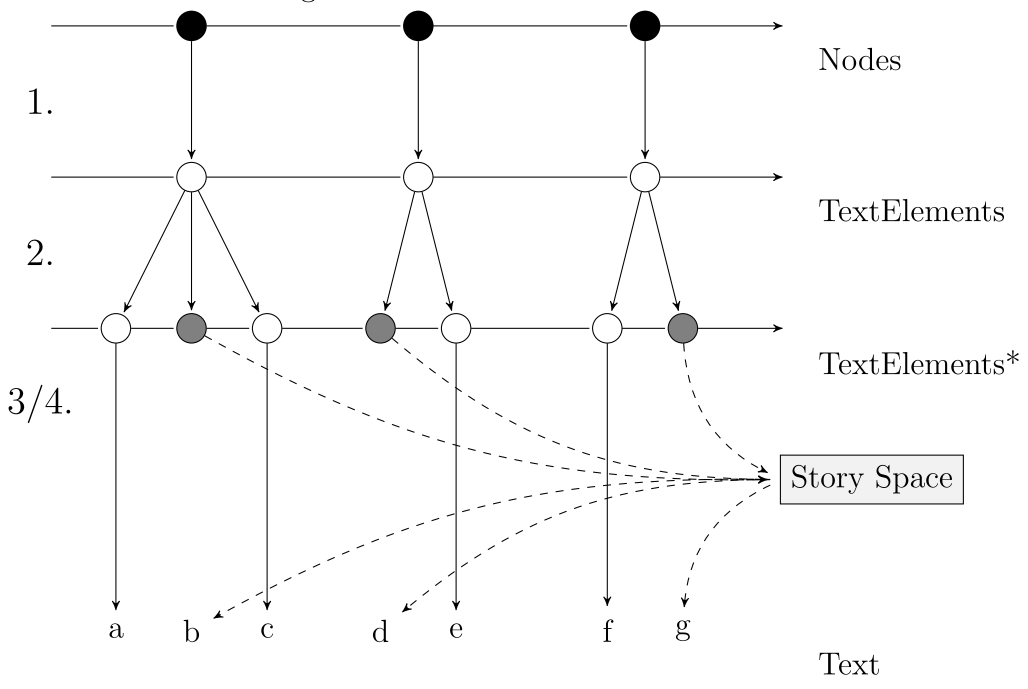
1. As a first step, the sequence is parsed and each node is processed into a *TextElement.* Different subclasses of TextElement carry out specific different roles in the writing of the final text.
2. The sequence is filtered, removing all those TextElements coming from nodes that were a by-product of the generation process, but that are not to be transformed into text.

   These nodes were introduced during plot generation to prevent some errors from occurring, for debugging purposes and to allow useful logging of the procedure. Since they are no longer needed, these TextElements are screened out and discarded.

   Any other node, whose role is yet to be determined, is of the *StringElement* type.
3. For each String Element, a first expansion step is performed:
   (a) The node label is used to retrieve one of the pieces of text defined in the text dictionary mentioned above, at random. This adds the semantic meaning carried by a dictionary entry into the system again.
   (b) Each word in this text is processed back into subclasses of TextElement: words that are all in upper case are made into ExistentElements, which will later be processed into ground existent instances. All other words are made into PlainElements, simple wrappers for a string.
   (c) The resulting subsequence of TextElements substitutes the single TextElement from which it originated, in the main sequence.
4. The expanded sequence is parsed again, this time processing each TextElement into an actual word. PlainElements are resolved to their original string value, while ExistentElements are resolved to the name of an existent.

The process through which ExistentElements are given their definitive form is subject of the next section.

Figure 6.1: The Text Generation Process



(*)some of which are ExistantElements, in grey;

As an example, here is given an example of the series of steps that transform a node label into a short piece of text.

Table 6.1: Processing a Node into Text

Data Type	Value
Label	”fight”
StringElement(SE)	SE(fight)
string	”HERO and VILLAIN engage in combat”
ExistentElement(EE)/ PlainElement(PE)	EE(HERO),PE(and),EE(VILLAIN),PE(engage), PE(in),PE(combat)
string	”The knight and the evil witch engage in combat.”

The final story text is the concatenation of all the texts resulting from the node sequence.

6.1.1 Space and Existents

The pieces of text associated to plot nodes contain a certain degree of indetermination. Existents are not defined, and are only referenced to as variables.

The text given above, *"HERO and VILLAIN engage in combat"*, is an example of this: *HERO*, and *VILLAIN* are keywords that identify a class of existent. In the final text, these keywords will be substituted by ground instances of this class.

The instantiation of existents belong to the Space of the story. Two are the steps that need implementation: first it is necessary to define what specific existent will be the ground instance of the class and second, consistency must be ensured, even considering the fact that there can be multiple instances of the same class.

There can be multiple Donor characters, for instance, interacting with the hero in different moments fo the story.

A system is implemented to provide these two steps in a modular and customisable way.

First of all, a simple classification of existents was given, based on the classification given by Propp.

- *Hero, Villain, Donor, Dispatcher, False Hero*: these are borrowed directly from Propp's ontology, as explained in chapter 1.
- *Victim*: this class was introduced to group together all those characters that can be victim to the Villain's misdeed.
- *Location, Item.* Propp's classification only considered characters. Places and physical objects are also existents and have to be part of the Space.

Then, a dictionary file is populated with a set of possible instances for each class. Just to give a few examples, instances of the class *HERO* can be something like *The Knight, the Prince, Boris, Ivàn*, and so on.

The definition of these instances in a separate file allows for a quick change of setting. If, instead of generating stories with a classic folktale setting, one was to generate narrative set in science-fiction space setting, this would simply require to swap out one dictionary for an other, defining the new setting.

The first step of the process, in which the ground instance of the existent class is determined, becomes really simple: an instance is selected at random from the dictionary.

Then it is necessary to ensure consistency. Once it is determined that the Hero will be *Boris*, this must hold until the end of the story, rather than randomising it again every time the variable *HERO* is encountered.

Some variables should only be randomised once. There is only one hero, and only one villain, for instance.

Others can, and should, be randomised multiple times. Multiple items will appear during the course of the narration, multiple locations and multiple donors.

This is achieved by keeping track of the state of the Text generation process, by the means of the same system used during Plot generation (section 5.2.1).

Predicates are used to encode information about the instantiation of existents. This information is generated when the randomisation takes place, and propagates forward during the parsing process.

When an existent is randomised, a predicate in the form $< Class, Instance >$, such as $< HERO, Boris >$ is added to the State. From this point downstream, every time an existent variable is encountered for the class *HERO*, its value is known and it can be substituted without randomisation.

Still, this is not sufficient, because some existents do need to be randomised again multiple times.

A solution to this issue comes from the recursive structure of the plot generation algorithm: the existents that need to be randomised again are those introduced in subplots. For instance, if the hero fails in passing a test devised by a donor, a subplot will be introduced in which he will meet a new one.

In order to follow the recursive structure of the plot, a stack of state objects is used instead of a single one. Whenever the parsing encounters the beginning of a subplot, a new state is added to the stack, containing the predicates of the parent state, filtered to remove those describing existents that need to be randomised again in this subplot.

When the subplot terminates, the sub-state is merged back into that of the parent plot and removed from the stack.

This mechanic may seem an overcomplicated way to meet the requirements: a simpler way to achieve the same result would have been to introduce a numbering in the existents, and distinguish an instance from the other already at generation time.

For instance, the first item to be part of the story would have been *ITEM1*, then in the next subplot *ITEM2* and so on. This way, consistency between occurrence of the same instance would be maintained, while different instances would have been easily detected and re-randomised.

However, the system currently in place is much easier to expand, in the context of any future development of this work.

The hooks it provides in correspondence to the introduction and resolution of a new subplot allow for a stricter and deeper interaction with the state of the generation process, to be used in any more advanced text generation algorithm.

6.2 Final Output

The final output is a .json file. It contains an array of Json Objects, each of which has four fields:

- *label*: the label of a story node, from the plot sequence.
- *text*: the line of text produced from said node, in its ground, resolved, form.
- *node_type*: the class under which the node falls, following the simplified classification explained in 4.1.
- *activity*: the identifier of an activity, or *null* if no activity is associated to this node. Only the activities specified in input will appear in the output.

The length of the array varies, according to the variability inherent to the plot generation algorithm. With the current, "proof of concept", settings it varies from around 75 to around 150 elements.

Chapter 7

Results

In this chapter are shown two examples of generated stories: *"Prince Boris and the Black Knight"*, and *"The Youth and the Evil King"*.

The plot generation layer does not provide any title to its stories. Titles have been given to them in order to provide some reference and context.

The titles follow a pattern that is common in many folktales: *Peter and the Wolf*, *Little Red Riding Hood and the Wolf*, *Morozko and Father Frost*, *Princess Vasilisa and the Fire Bird*. In all these cases the title presents the two main characters, being the hero and the villain.

In order to replicate this, once the Plot Generator Layer has generated the sequence of story nodes defining the plot, an additional node is added at the top of it before submitting it to the Text Generation Layer. Since it is added after the plot generation process is complete, this node will not influence the development of the story.

This additional node is associated to the text string *"HERO and VILLAIN"*.

Following the text generation process outlined in chapter 6, this node is processed along with the other nodes, and the keywords *"HERO"* and *"VILLAIN"* are resolved to ground existents picked from the appropriate dictionaries, like *"Prince Boris"*, as the hero, or *"the Black Knight"*, as the villain.

This way *"HERO and VILLAIN"* is translated into *"Prince Boris and the*

Black Knight", while the text generation state mechanic explained in section 6.1 ensures the consistency in the choice of characters, so that those appearing in the title are the same ones that take part in the story itself.

In the output, the title will appear as the first line.

For the generation of these two stories the existent dictionaries, containing locations, characters and items, have been populated with items from the classic setting of Russian folktales.

As explained in section 6.1.1, the dictionary contains 8 classes of existents: *Hero* (populated with 10 entries), *Villain* (8 entries), *Donor* (8 entries), *Dispatcher* (3 entries), *False Hero* (5 entries), *Victim* (5 entries), *Item* (8 entries), *Location* (6 entries).

Any story generally follows a main sequence of scenes, which is fixed and static, as defined in Propp's formalism. This sequence of scenes is described in chapter 2.

In short it develops as follows: a villainy is committed, the hero embarks in a quest to liquidate it, obtains help in the form of a helper character or an inanimate object, goes to where the villain is, fights him, is victorious and finally returns to be rewarded.

Clearly, any number of things can go wrong in this sequence of events and when this happens the story derails: other characters are introduced, the hero attempts to overcome the same obstacles multiple times, he is maybe defeated a few times, but eventually always triumphs.

This means that the generated story can be longer or shorter depending on the amount of subplots that need to be introduced in order to resolve the various impasses.

Each line is the result of the text generation process described in chapter 6. Each node given in output by the plot generation layer is translated into its final textual form over multiple steps, one of which is the transformation of keywords in actual instances of the corresponding existent.

These two stories have been selected as examples because they highlight different features of the generation process.

Along with their text the processes that led to their definition are explained, in order to show how the generation process works.

7.1 Prince Boris and the Black Knight

This generated story has been chosen as a first example because it develops nearly linearly, most obstacles being overcome at the first attempt. This way the narrative structure is more evident, as only a minimal amount of subplots is needed.

The story begins as follows:

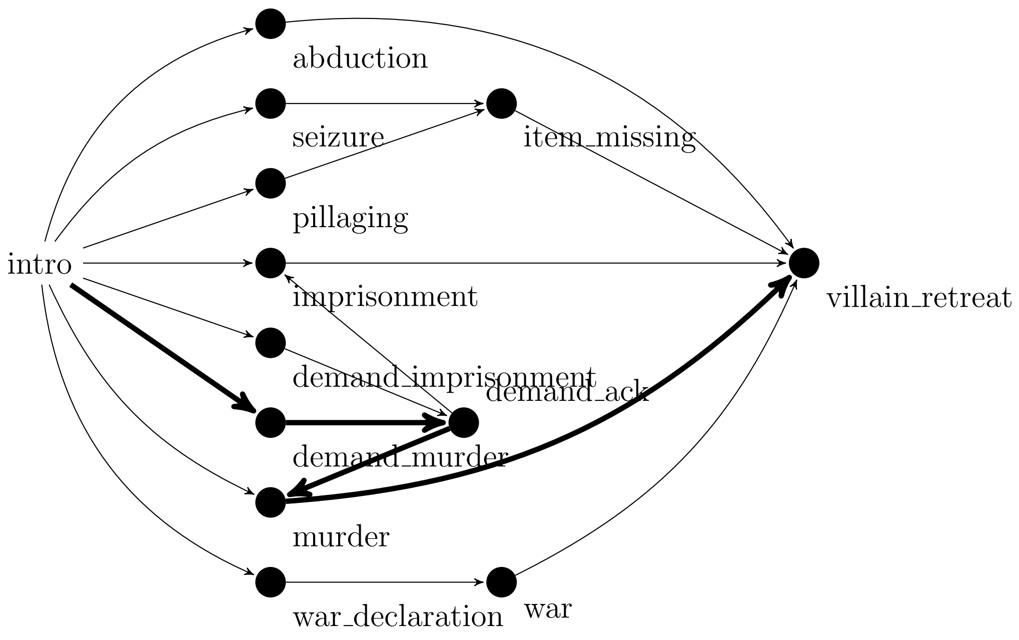
```
PRINCE BORIS AND THE BLACK KNIGHT

prince Boris lives at the village
the Black Knight demands the old man to be murdered
the demand is accepted
the old man is murdered
the Black Knight goes in hiding at the tower
```

It opens with the *Villainy* scene. In it, the villain commits some form of misdeed towards an innocent character, and then leaves the scene.

The graph structure associated to the scene is shown here. For ease of comprehension, node labels do not represent the actual ID of the nodes, but instead reflect the semantic meaning that will be given to the nodes during the text generation step.

Figure 7.1: Structure of the *Villainy* scene



The graph traversal algorithm described in 5.1.2 traverses the graph until a final node is reached. It then outputs the sequence of nodes it encountered along the resulting path (highlighted in the figure). The sequence contains the nodes: *intro*, *demand_murder*, *demand_ack*, *murder*, *villain_retreat*.

As explained in 5.1.2, for each node in the graph defining a scene, the *local frequency*, as the number of times the node has been used in the current generation process, and the *global frequency*, as the number of times it has been used since the first generation was launched, are recorded.

After the graph traversal step, both frequencies are updated: each node in the path has its frequencies increased by one.

Although several are available, the system is designed to use the traversal policy *inverse frequency* (see 5.1.2). This policy uses information about the depth of the node in the exploration tree and about the frequencies of the node to select what node is to be explored next.

As a consequence, the more a node has been used, the less likely it is to be used again, both within the same story and in any future ones.

In the later step of text generation, explained in detail in chapter 6, each node is associated to a piece of text, while existents are picked from the appropriate dictionary. The first node, *intro*, is takes here as example.

1. The system uses the ID of the node to retrieve a string from a set of short pieces of text associated to it in the text dictionary.

   In this case the set contains strings like:

   - *HERO lives at LOCATION*
   - *at LOCATION lives HERO*
   - *once upon a time, at LOCATION lived HERO*
   - *VICTIM lives at LOCATION*
   - ...

2. The dictionary returns a random string from the set. In this case it returned *HERO lives at LOCATION*

3. The string is parsed. Words that are not in *All Caps* will result as they are in the final text, while words that are are recognised as keywords and will be subjected to another processing step. In this case, the keywords are *HERO* and *LOCATION.*

4. Each keyword is resolved to a ground instance of an existent. For each of them, the module records in a stack of *Text Generation State* objects (section 6.1) information about the previous instantiation of the same existent.

   Focusing on the *HERO* keyword, since this is the first line of the story it has never been encountered and instantiated before. In other circumstances, the logic predicate $< HERO, value >$ would be present in the state data structure and the keyword would be substituted with *value.*

   Since no previous instantiation of *HERO* is found, the system needs to retrieve a new random value for the keyword from the dictionary of existents.

   The dictionary contains several examples of heroes:

- *Ivàn*
- *The Youth*
- *the woodcutter*
- *prince Boris*
- *the knight*
- ...

In this case, *Prince Boris* was picked. The predicate $< HERO, PrinceBoris >$ is added to the current state so that in later text generation the hero remains consistent.

In the same way, the keyword *LOCATION* is given the value of a location from the existent dictionary.

5. Each keyword is substituted with its final value. in this case *HERO* was given the value *prince Boris*, while *LOCATION* was given *the village*.

Following this procedure the node *intro* was translated into *Prince Boris lives at the village*, *demand_murder* into *the Black Knight demands the old man to be murdered*, and so on.

The resulting text is that shown above.

The story continues.

```
vengeance is required
the king sends prince Boris on the mission
prince Boris prepares
the quest begins
prince Boris departs
prince Boris reaches the mountain
```

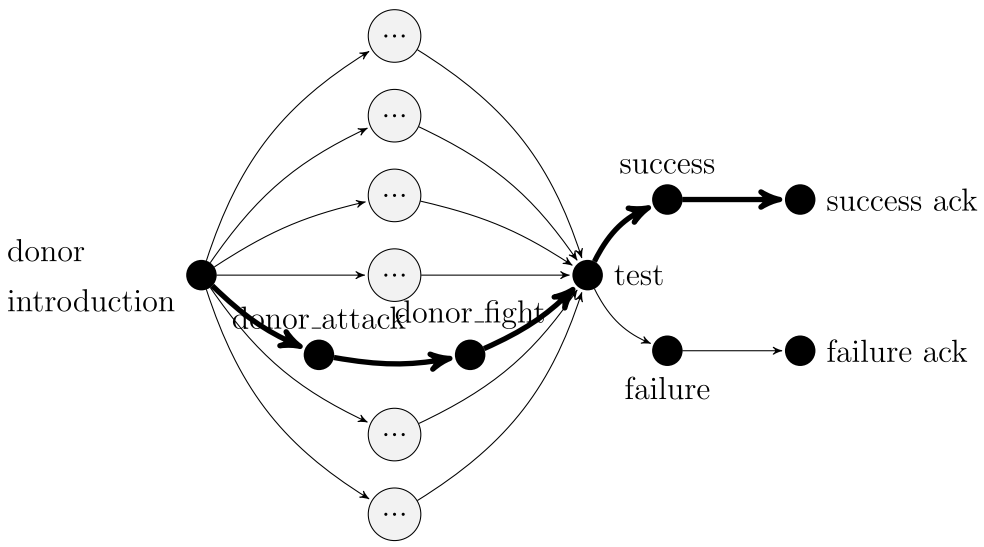
The generation proceeds with the next scenes in sequence: in this group of scenes, in this case, a dispatcher character (the king) orders the hero to prepare and leave to avenge the villainy.

```
at the mountain lives the hermit
the hermit attacks prince Boris
the hermit and prince Boris fight each other
prince Boris begins the test
prince Boris has passed the test
```

This scene is generated by the *FirstFunctionReaction* function. It encompasses multiple of the original functions defined by Propp, in which a donor character is introduced, the hero is tested, and the result of the test is made known.

The graph associated to this function is shown below. In the figure, some nodes not involved in the path have been grouped together for ease of readability. A more complete documentation of the structure of this graph is in section 5.1.1.

Figure 7.2: Graph traversal, *FirstFunctionReaction* function



The graph traversal algorithm outputs the sequence of nodes *donor_introduction*, *donor_fight*, *donor_test*, *success*, *success_ack*.

Each one of these nodes is then translated into a line of the above by the Text Generation Layer.

In the current implementation the text resulting from the testing of the hero is merely a placeholder to indicate that a test is taking place. With more authorial work more detail about these tests could be made part of the graph structure and appear in the story.

In this case, the plot generation layer outputs a sequence of events in which the aggressor is defeated immediately.

```
prince Boris will now be helped by the hermit
prince Boris finds the magical sword
prince Boris desires the magical sword
prince Boris acquires the magical sword
prince Boris discovers a way to the tower
prince Boris follows the path to the tower
prince Boris arrives to the tower
```

The plot generation layer has generated an atypical sequence of events that is particularly kind to the hero: he immediately finds a helping character, he immediately finds a magical item, he immediately finds a way to reach the tower where the villain is.

None of these events is guaranteed to happen, especially not at the first attempt. In a more typical story more impasses arises demanding for a subplot to be introduced. In this case that is not necessary yet.

```
prince Boris fights the Black Knight
prince Boris is victorious
prince Boris is scarred
the Black Knight is defeated
the Black Knight is imprisoned
```

This series of events covers the *Struggle* and *Liquidation* scenes. The output of the plot generation layer is yet an other victory for the hero, who defeats the villain immediately. Other possibilities would have been a defeat, requiring the hero to run away, or a victory obtained by employing the help of a helper character or a magical item.

In this case the hero could have needed the help of the hermit he met earlier, or could have needed the magical sword he has found.

Although victorious, the hero is scarred during the struggle. this event opens up new possibilities later in the story, since other events that are dependent from this one can now happen.

This is made possible by the dependency specification system, based on the STRIPS [50] standard explained in section 5.2.1: the node corresponding to the event E, in which the hero was scarred, has a postcondition *Post*, in this case being the logic predicate $< hero\_mark, true >$. This predicate is added to the state structure.

From this point on, any event E_1, with precondition *Pre* for which it holds that $Post \Rightarrow Pre$ is enabled and can be part of the plot.

An example of this behaviour is shown later within this story, when the hero manages to be recognised because of his scar.

```
prince Boris returns
prince Boris is returning
the servant overhears prince Boris talking about his success
the servant learns of what prince Boris has done
the servant presents his claims to the king
the king acknowledges the claims presented by the servant
the servant is rewarded
```

Not everything goes according to plans: while the hero is returning, he unintentionally informs an other character, a false hero, of what happened.

This false hero claims the rewards that would justly be given to the hero. The dispatcher character believes his claims and rewards him.

This is an impasse: the hero *must* be rewarded at the end of the story, and if the rewards are claimed by someone else, the story cannot proceed.

The *Impasse Scanner* module (see section 5.3) detects the node causing the impasse, and the *Impasse Handler* introduces the appropriate subplot. In this subplot the hero presents his own claims to the dispatcher trying to obtain his just reward. If no other impasses occur, he eventually succeeds.

```
prince Boris presents his claims to the king
the king listens to prince Boris presenting his claims
the claims are not accepted
```

The king is not convinced. The story generation is paused again and, once again through *Impasse Scanner* and *Impasse Handler*, a new subplot is introduced in which the hero tries again.

```
prince Boris confronts the servant
prince Boris is recognised by the mark he is carrying
prince Boris is finally recognised
```

This time, rather than a scene in which he hero appeals to the dispatcher, the plot generation layer produces one in which the hero directly confronts the false hero.

The hero is here recognised by a bodily mark. This is clearly possible only if he was previously marked in some way.

As explained earlier, the node associated to the event in which the hero is scarred during the fight with the villain had a poscondition *Post* which caused the logic predicate $< hero\_mark, true >$ to be added to the state of the generation.

The node associated to the event in which the hero is recognised by his bodily mark has a precondition *Pre* that is true if *Post* is true. In this case *Pre* is simply the same predicate $< hero\_mark, true >$.

The graph traversal algorithm that defines the develoment of this scene, while traversing the graph, performs a logic analysis of the state, which is an object encoding the logic expression $predicate_1 \wedge predicate_2 \wedge predicate_3$....

If it evaluates that $state \Rightarrow Pre$, so that the precondition to the node is true in the context of the current state of the generation, then the node is enabled.

In this case this is true, since one of the predicates part of the state is $Post$.

Then, the traversal has produced a path that includes the node in question, which becomes part of the plot.

Since the hero has now succeeded in being recognised, no more impasses occur, and the subplot can continue.

```
the servant is exposed
the reward to the servant is withdrawn
the servant is punished
the servant is killed
```

The subplot in which the hero solves the problem of his reward being unjustly given to a false hero ends with the false hero being exposed. In this case the scene ends with the servant being killed.

```
prince Boris is rewarded with a magical crown
```

The story draws to a close and the hero is rewarded. The reward can take many forms, in this case he receives a valuable item.

7.2 The Youth and The Evil King

This is an other example of generated story. It was chosen because it highlights the workings of the recursive subplot system, without being overly long.

In it, the hero fails multiple times in overcoming a certain obstacle, and multiple subplots are introduced as a consequence.

```
THE YOUTH AND THE EVIL KING

the Evil King hears the Youth from afar
the Youth unwillingly reveals information,
        upon which the Evil King hatches a plan
```

This is an instance of the *Reconnaissance* function. In it, the villain attempts to obtain information about the hero or an item, or an other character. Once this information is obtained, the villain then commits its misdeed.

While being part of the main sequence that is used in every story, this function did not appear in the previous example. This is because this chain of events can or can not happen, and sometimes the graph exploration algorithm generates an empty sequence.

```
the Evil King seizes the magical bow
the Evil King runs away to the forest
```

In this case the villainy is not a murder, but rather the theft of an object.

```
the magical bow is needed
the king announces the misfortune to the Youth
the Youth acknowledges the problem
the quest begins
the Youth departs
the Youth reaches the forest
```

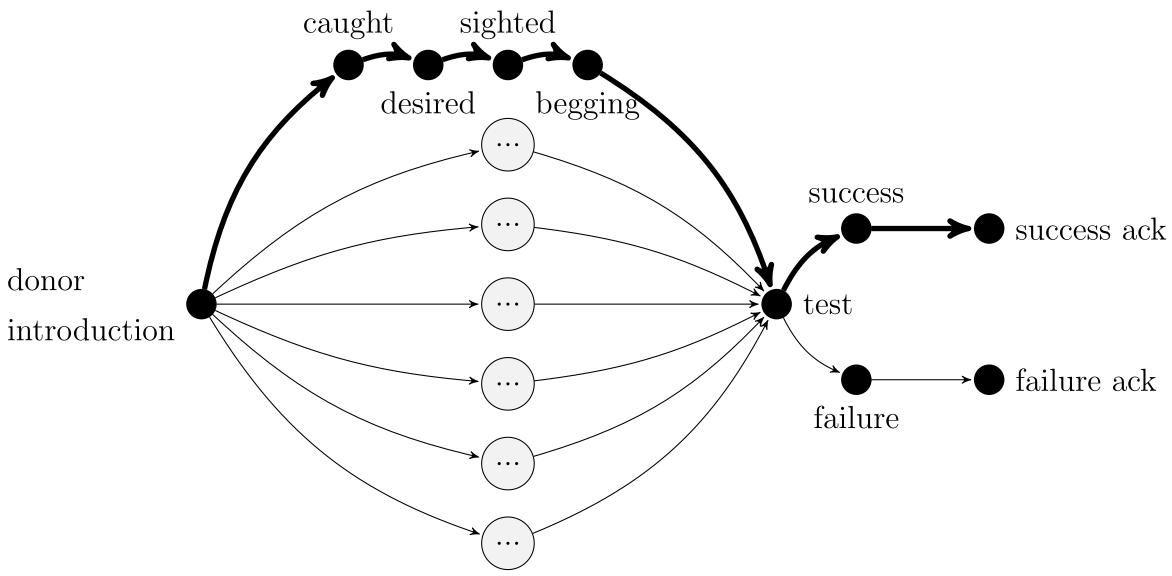
This time the plot generation layer has produced a different development of the *Departure* function.

Rather than avenging the villainy, the hero will try to take back the missing object. In this case He wasn't issued an order by an authoritative character, but rather decides autonomously to depart.

```
the Pixie is at the forest
the Youth sees the Pixie
the Youth wants to catch the Pixie
the Youth catches the Pixie
the Pixie begs the Youth
the Youth begins the test
the Youth has passed the test
the Youth will now be helped by the Pixie
the Pixie gives the magical sword to the Youth
the Youth acquires the magical sword
```

This scene was generated based on the *FirstFunctionReaction* function, just as in the example given in the previous story. The same graph structure can lead to different developments of the scene, depending on how the traversal algorithm traverses it.

Figure 7.3: Traversal of the graph associated to the *FirstFunctionReaction* function



Similarly to what happened in the same scene from the previous story, the hero meets a character, is tested, passes the test and obtains a magical item as reward.

In this case, however, the test was not a fight, but rather a test of kindness: the hero captures the Pixie which begs him for freedom. He frees it and obtains a sword in exchange.

The fact that, in both cases, the hero has been rewarded with a magical sword is not a consequence of any feature of the algorithm but instead is merely coincidence: the text generation layer has, in two different stories, picked the same item from the dictionary of existents.

```
the Youth finds a path to the forest
the Youth follows the path to the forest
the Youth arrives to the forest
```

In this *Guidance* scene, the hero autonomously finds a way to the location at which the villain is.

This sometimes fails: the hero can be unable to proceed to that location. In these cases a subplot is introduced in which he seeks for the help of an other character, who can in turn show him the way or transport him directly.

```
the Evil King and the Youth fight
the Youth is unable to effectively fight the Evil King
the Evil King is victorious
```

The hero is defeated. This is clearly a major obstacle in the subsequent development of the story, and therefore an impasse is raised.

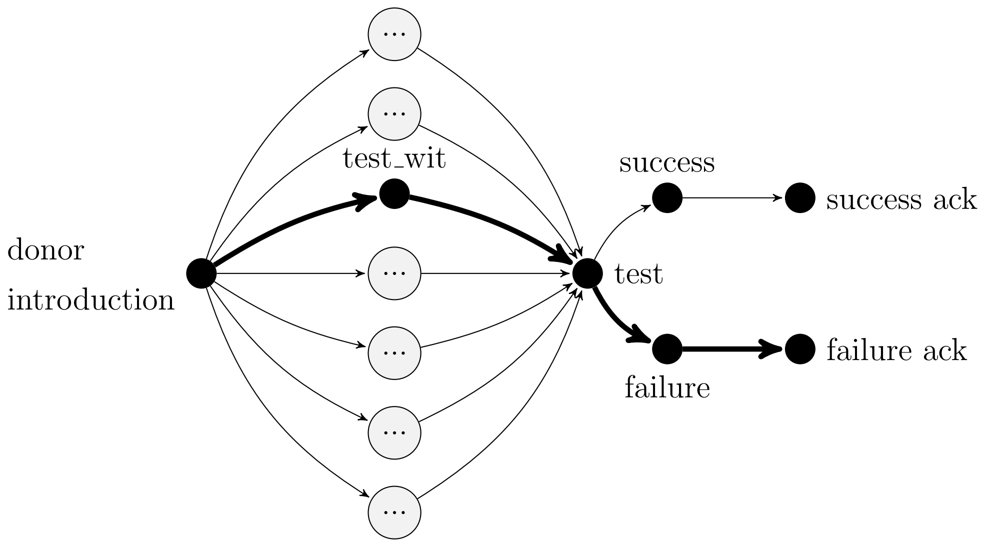
The relevant modules introduce the new subplot, in which the hero runs away to safety, tries to obtain help in the form of a helping character or an item, and eventually tries again.

```
the Youth barely escapes with his life
the Youth resumes his travelling
the Youth is now at the castle
the hermit is at the castle
the wit of the Youth is tested by the hermit
the Youth begins the test
the Youth has failed the test
the Youth understands he has failed the test
```

After resuming his traveling, the hero meets an other donor character, in a new instance of the same *FirstFunctionReaction* scene, in which he meets the donor, is tested, and reacts to the test.

The same graph structure has been traversed in a different way and this has produced a different scene: a test of wit rather than a test of mercy, resulting in a failure instead of a success.

Figure 7.4: Traversal of the graph associated to the *FirstFunctionReaction* function



The fact that different traversals of the same graph structure yield different scenes in beneficial, and is reinforced by the *inverse frequency* policy used by the algorithm.

In this scene the hero has met a new donor character.

This character is different from the first donor the hero encountered. The first one was a pixie, while this one is a hermit. The text generation layer correctly picked a new existent to fulfill the same role, due to the mechanics explained in section 6.1.

In this case the hero is subjected to a test of intelligence and wit, but fails. Again, this generates a new impasse, and a subplot to this subplot is introduced recursively. This subplot will be resolved when the hero manages to pass a test and obtain help.

```
the Youth resumes his travelling
he Youth arrives at the tower
the warrior is held in captivity. He sees the Youth approaching
the warrior wants to be free
the warrior asks the Youth for freedom
the Youth begins the test
the Youth has failed the test
```

The hero is tested again and fails again. An other subplot is required.

```
the Youth resumes his travelling
at the mountain lives the lady of the forest
the lady of the forest is displaying the flying shoes
the lady of the forest sees the Youth
the lady of the forest sees the Youth
the lady of the forest offers a trade
the Youth begins the test
the Youth has passed the test
```

The hero meets yet an other character, is tested again, and this time succeeds.

This results in a known pattern that is common in folktales: the triple repetition of an event. However, this is not coded in any way, but rather it emerges randomly from the recursive nature of the subplot system.

The subplot is resolved and, because of the recursive structure in use, so are the subplots from which it was introduced.

The generation process can continue from the subplot in which the hero was trying to fight the villain again.

```
the lady of the forest decides to help the Youth
the Youth is magically transported to the mountain
        by the lady of the forest
the Youth is finally at the mountain
```

The recently met donor helps the hero by transporting him to the desired location. He can now attempt to defeat the villain again.

```
the Youth and the Evil King engage in a fight
the magical sword helps the Youth in the fight
the Youth defeats the Evil King
the Youth obtains the magical bow
the Youth seizes the magical bow
```

The two fight and the hero is victorious, thanks to the item he had obtained previously. The State of the generation contains information about the villainy committed, and this enables a path in the graph structure of the *Liquidation* scene (in which the villainy is liquidated) in which the hero obtains the item he was searching for: the magical bow seized by the villain in the *Villainy* scene.

```
the Youth returns
the Youth has arrived back
the Youth marries
```

The hero returns, since the node in which information about the hero's deeds are divulged to a malevolent false hero, no obstacle to the hero being rewarded arises and no subplot in which he struggles to prove his claims is needed.

He simply goes back and marries.

These two stories show how the mechanics described in the previous chapters concur in generating the final narrative.

The same two stories are available in appendix A in a more readable, uninterrupted, form.

More general considerations about this architecture in its entirety will be the subject of the next chapter.

Chapter 8

Discussion

This approach has a few characteristics that set it apart from others already present in literature:

- At the best of our knowledge, it is one of the very few attempts at generating Constrained Plot and Constrained Space (see 2.2), the others being the systems *Say Anything* [43], *Scheherazade* [44] and *Thespian* [45].

  Among the other examples of this level of automation, it is unique because:

  - Differently from *Thespian*, it is not based on character simulation. This allows for the generated stories to be always centered on a single character (the hero), making them better suited to serve as narrative superstructure to activities in which the user is always the protagonist, upon which the narration is focused.
  - It doesn't require a human to be present during the generation process, as it is in the case of *Say Anything* or *Scheherazade*.

    While a certain amount of authorial work is still required to provide the system with the necessary knowledge base so that it can operate, this work takes place before the generation starts and only once.

    Many stories can be generated thereafter by the system in autonomy without the need for any further interaction.

- While others have a narrower scope, this is the only approach in which multiple levels of abstraction are considered: this system is based on the cooperation between a high level algorithm, generating the scene structure of the story based on Propp's formalism, and one at a lower level, based on the Fabula Network Model, that generates the specific chain of events that constitute each scene.
- The high degree of modularity allows for customisation, flexibility and expansion of the system.

  In particular the mechanics in place allow for the specification of a set of events that will necessarily happen in the story, or even a certain amount of repetitions of the same event.

8.1 A potential application

In order to reach the final goal of providing context, meaning and an interesting narrative superstructure to a set of activities, generating a story is not enough, as it has to be integrated with the system proposing these activities and delivered to the user.

The actual integration of the story generator into a project such as MoveCare or REWIRE is outside the scope of this work, mainly because the system that processes the story and delivers it to the user is yet to be fully defined.

In short it is proposed to deliver stories to the user partially in the form of text, and partially in the form of animations. This is discussed in more detail in section 9.1: *Further Developments and Improvements*.

However, here is outlined a way in which a story can potentially be integrated and delivered. Two of the three activities listed below are merely an example and do not represent real-world implemented ones.

The story used as example is *Prince Boris and the Black Knight*. It already served a similar purpose in the previous chapter.

Here only extracts from it are reported, it can be found in its entirety in appendix A.

Let us assume that a user is to be proposed a set of three activities:

1. An exergame for physical rehabilitation: the user sits or stands alternatively. This translates into a movement of his or her avatar who is riding a horse. The correct movements of the user allow the avatar to dodge various obstacles that would otherwise hit it while the horse moves along a path. This is an example of a real-world exergame, called *Horse Rider* that has been developed in REWIRE.
2. A puzzle game designed as a cognitive exercise: the user has to correctly compose an image of a landscape. When the image is complete a path is visible on it.

3. An other physical exergame in which movements of the user translate into movements of an avatar who fights an other character, perhaps in dodging the blows of the opponent, or in striking in some way.

These activities could be mapped to story events as follows:

1. The horse riding exergame is mapped to events in which the hero reaches a new location.
2. The puzzle game can represent moments in which the hero is trying to reach a new location
3. The fighting exergame fits well with the moment of struggle between hero and villain.

A .json file containing an $< ActivityID, StoryNodeID >$ pair for each activity to be proposed is passed as parameter to the generator. This will force the generator to produce a story containing at least one instance of the nodes identified by $StoryNodeID$, and to associate it to the appropriate $ActivityID$ in the output.

After completing the generation process, the system returns an other .json file containing the text of the story and the association between story events and activities. For more details about the format of the output file, see section 6.1.

The story opens with the presentation of the main character. This can be conveyed to the user via a short piece of text.

Then, the Black Knight order a murder to be committed:

```
the old man is murdered
```

This could be made into a short, simple animation showing the murder in a simplified way.

This would require a set of dictionaries of assets representing the various existents and characters. This issue i discussed further in section 9.1.

After the murder, the hero is informed of what happened by the King, and then departs to seek vengeance.

This can become a piece of text, perhaps interleaved with an animation showing the king talking to prince Boris.

```
prince Boris reaches the mountain
```

The hero reaches a new location. The .json returned by the generator associates this event with the horse riding exergame. The users plays until the game is completed.

Once the game finishes, the story continues: prince Boris is assaulted by a hermit who lives in the forest and wins. Prince Boris then finds a magical sword.

This part can again be translated into a mix of text and animations: perhaps an animation can show the fight, or an other can show prince Boris wielding the newly found magical sword.

The hero is now attempting to reach the location at which the villain has retreated after committing his misdeed, so that he can be fought and vanquished.

```
prince Boris discovers a way to the tower
```

This event is associated to the second activity: the cognitive puzzle game.

The user plays and completes the game.

The story continues with the hero reaching the tower where the villain is and engaging him.

```
prince Boris fights the Black Knight
```

Here the third activity is presented to the user: the fighting exergame.

Once that has been completed as well, the story continues: although the hero was victorious, he was injured in the fight and remains scarred.

while the hero is returning home he inadvertently informs a servant of his deed.

The servant then pretends to be the one who vanquished the villain and claims prince Boris' just reward in his stead.

An animation could show the servant talking to the king, or the king rewarding the servant.

The hero then tries to disprove the servant's lies but he fails, he then confronts the servant and finally prevails, proving his deed thanks to the scar he received from the villain. The servant is exposed and punished.

```
prince Boris is rewarded with a magical crown
```

The story comes to a close: an animation shows the hero being justly rewarded by the king.

This is an example of how a relatively simple story could provide context to a set of three, unrelated, activities, in an way that is more interesting and engaging to the user.

Chapter 9

Conclusions

In conclusion, this work presents a new approach for Constrained Story Generation (see 2.1: *The Mixed Initiative approach*), based on a two-layered architecture.

The Plot Generation layer works on two levels of abstraction, in parallel: human-authored probabilistic graphs model scenes (4.1), while a recursive algorithm determines the sequence of scenes to be generated, introducing new subplots to resolve impasses in the generation process (4.2 to 4.3).

An encoding of the State of the generation, based on principles borrowed from classic AI (see 5.2.1: *STRIPS State Encoding*) 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 (5.1) and a stack-like structure to model the State of the text generation, in order to properly substitute keywords with the name of existents (5.1.1).

Referring to the requirements given in 1.1.5, results show how the system adheres to the specification, in particular, 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 (see sections 5.1 and 5.2).

- **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 that is one of their defining factor 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.
- **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.

These properties ensure that this approach is in fact applicable in the context of those healthcare-related projects described in section 1.1, as well as in other similar contexts.

A hypothetical example of application has been given in the previous chapter.

At the same time, the system shows a clear limitation, being the amount of authorial work that is required prior to the actual generation process.

In particular, in order to initiate any actual generation process, a human author needs to define the following:

- A set of scenes composing the main plot, and a number of scene sequences describing subplots. Currently the set of Functions defined by Propp is being used, but the system can work with any group of scenes.
- A probabilistic graph structure for each scene, defining the different events that can happen. 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, carrying the semantic information to be associated to node labels.

Especially the design and input of graph structures are particularly demanding tasks.

This step requires the author to decide in what ways a scene can develop, encode this information into a graph with the appropriate language, and define the appropriate rules to ensure consistency in the results.

Given that the system cannot generate anything outside the space given to it as knowledge base, the quality and variety of generated stories depends on the quality and volume of the inputs. For this reason, quality in the output requires a substantial investment in the authorial phase.

The ways in which these limits can be improved upon or overcome are part of the next section.

9.1 Further Developments And Improvements

The result of this work is not a complete product. Some developments are needed in order to make it usable in the context of projects akin to MoveCare or REWIRE (see chapter 1). Moreover, it could benefit from a few improvements that would make it more effective in fulfilling its role.

9.1.1 Necessary Steps

In order to make this architecture usable in a real-world application, it is first of all necessary to further develop the Text Generation step. The current system (see Appendix A for examples) is sufficient to show that semantic information can be re-introduced in the story structure, but is too primitive to generate text that can be considered interesting by a user.

The generation of complex text, or the *Discourse* (see chapter 2) of a story, is a vast subject, with many open questions, difficult to undertake.

As explained in the *Discussion* chapter, the proposed way to develop the system in order to overcome this obstacle is to convey the output to the user partly in the form of text, and partly in the form of animated images.

It is certainly possible to compose simple, 2-dimensional, animations based on the short texts used to define the semantic meaning of node labels.

For instance, starting from a database of sprites and animations, a piece of text like *the knight and the witch engage in a fight* can be used to generate and show a short animated clip of a knight and a witch fighting. The same can apply to many more events of the story.

At the same time, some text is still required. In some cases the events narrated don't translate well into an animation: *the prince reflected upon the problem* would probably translate into an uninteresting animation, for instance.

Also, having images or animations accompanied by text would be the preferred way of presenting a story and, while requiring less text than a text-only presentation, would still require a refinement of the text generation procedure.

Enhancing the text generation process would therefore have a strong impact on the final effectiveness of the whole system.

A first, relatively inexpensive step would be the expansion and partial rewriting of the text dictionary. Adding more descriptive, and perhaps more interesting, pieces of text. This task is beside the scientific goals of this work, and therefore has not being tried at the moment of writing.

Whether it be for the purpose of a more advanced text generation, or as a step in generating images / animations, the preliminary step should be the definition of an ontology representing (and coding) the different classes of existents, their properties, and the relationship between them.

This structure should be usable at runtime to associate together characters, locations, items and animations in way that ensures a more interesting characterisation of any existent.

For instance, a certain character could be associated to a set of items that he owns and that can be displayed in an image or in the text introducing it. It could be associated to a location where he lives and where he interacts with the hero. Different existents could have properties defined for them that describe how they interact with other existent, how they can be used in the story, how they can be animated, and so on.

An other student is currently working on this specific task. Since he started very recently it was not possible to include any account of his work in this thesis.

9.1.2 Suggestions on Further Research

In order to test the effectiveness of this system in promoting the engagement of users in a set of activities with the goal of improving their health, it is required to develop a complete prototype and to deploy it in the context of a project such as the ones described in chapter 1. Following the timings and schedules of a clinical trial.

However, this is not necessary for the testing of the system as a pure story generation architecture.

Here is proposed a process to determine whether the system is successful in generating acceptable stories, through crowdsourcing.

1. The generator is used to produce a set of stories.
2. A set of real-world tales is selected.
3. These stories are decomposed by hand into sequences of the same texts defined in the text dictionary. This way they will be indistinguishable from generated stories in terms of language. Whatever differences in structure and existents remain.
4. A set of stories of the same size is generated.
5. A set of existents is composed, 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.
6. Existents in the two groups of stories are scrambled.
7. A sample containing both groups, shuffled, is presented to a crowd", of sufficient size. Subjects are asked to estimate how much each story seems "real" or "fake", and why.

This process should highlight any differences in the structure of stories, between real and generated ones. Differences in language and existents should have been removed by the translation and scrambling steps.

If this quality evaluation proves to be effective in highlighting these differences, it could be employed to refine the system iteratively, thought multiple iteration of testing and a refinement steps.

The procedure is just outlined here: some steps are not entirely defined. In particular the process of decomposition of a real tale into the language used by the generator requires once again authorial work by a human, who would need to face part of the obstacles emerged during the failed attempt at story generation explained in chapter 3.

This step is however necessary. Without the development of a *perfect*, Discourse generation system, indistinguishable from a human, generated stories and real one would always be recognisable by the differences in their Discourse: the language used, the ordering of the events, the way in which they are recounted, the rhythm of the narration and such.

An other potential area of exploration that is opened by the outlining of this architecture is the automation of the authorial task of defining the graph structure describing scenes.

Fruitful work in this direction would greatly improve the effectiveness of this approach, by significantly reducing the amount of authorial work required for the initialisation of the system. In turn this would make the system a great deal more flexible, because it could automatically adapt to a new dataset of stories, or a more sizeable one.

The first obstacle to any research on the subject would be the lack of datasets with annotations of sufficient size. Possibly crowdsourcing can offer an answer in this sense as well.

Appendix A

Two examples of generated narrative

Below are reported the two generated stories discussed in chapters 7 and 8, in a more readable form, without commentary.

A.1 Prince Boris and the Black Knight

```
PRINCE BORIS AND THE BLACK KNIGHT

prince Boris lives at the village
the Black Knight demands the old man to be murdered
the demand is accepted
the old man is murdered
the Black Knight goes in hiding at the tower
vengeance is required
the king sends prince Boris on the mission
prince Boris prepares
the quest begins
prince Boris departs
prince Boris reaches the mountain
at the mountain lives the hermit
the hermit attacks prince Boris
the hermit and prince Boris fight each other
prince Boris begins the test
prince Boris has passed the test
prince Boris will now be helped by the hermit
prince Boris finds the magical sword
prince Boris desires the magical sword
prince Boris acquires the magical sword
prince Boris discovers a way to the tower
prince Boris follows the path to the tower
prince Boris arrives to the tower
prince Boris fights the Black Knight
prince Boris is victorious
prince Boris is scarred
the Black Knight is defeated
the Black Knight is imprisoned
prince Boris returns
prince Boris is returning
the servant overhears prince Boris talking about his success
the servant learns of what prince Boris has done
the servant presents his claims to the king
the king acknowledges the claims presented by the servant
the servant is rewarded
prince Boris presents his claims to the king
the king listens to prince Boris presenting his claims
the claims are not accepted
prince Boris confronts the servant
prince Boris is recognised by the mark he is carrying
prince Boris is finally recognised
```

```
the servant is exposed
the reward to the servant is withdrawn
the servant is punished
the servant is killed
prince Boris is rewarded with a magical crown
```

A.2 The Youth and The Evil King

```
THE YOUTH AND THE EVIL KING


the Evil King hears the Youth from afar
the Youth unwillingly reveals information,
        upon which the Evil King hatches a plan
the Evil King seizes the magical bow
the Evil King runs away to the forest
the magical bow is needed
the king announces the misfortune to the Youth
the Youth acknowledges the problem
the quest begins
the Youth departs
the Youth reaches the forest
the Pixie is at the forest
the Youth sees the Pixie
the Youth wants to catch the Pixie
the Youth catches the Pixie
the Pixie begs the Youth
the Youth begins the test
the Youth has passed the test
the Youth will now be helped by the Pixie
the Pixie gives the magical sword to the Youth
the Youth acquires the magical sword
the Youth finds a path to the forest
the Youth follows the path to the forest
the Youth arrives to the forest
the Evil King and the Youth fight
the Youth is unable to effectively fight the Evil King
the Evil King is victorious
the Youth barely escapes with his life
the Youth resumes his travelling
the Youth is now at the castle
the hermit is at the castle
the wit of the Youth is tested by the hermit
the Youth begins the test
the Youth has failed the test
the Youth understands he has failed the test
the Youth resumes his travelling
he Youth arrives at the tower
he Youth arrives at the tower
the warrior is held in captivity. He sees the Youth approaching
the warrior wants to be free
the warrior asks the Youth for freedom
the Youth begins the test
the Youth has failed the test
the Youth resumes his travelling
the Youth resumes his travelling
at the mountain lives the lady of the forest
the lady of the forest is displaying the flying shoes
the lady of the forest sees the Youth
the lady of the forest sees the Youth
the lady of the forest offers a trade
the Youth begins the test
the Youth has passed the test
the lady of the forest decides to help the Youth
the Youth is magically transported to the mountain
        by the lady of the forest
the Youth is finally at the mountain
the Youth and the Evil King engage in a fight
the magical sword helps the Youth in the fight
the Youth defeats the Evil King
the Youth obtains the magical bow
the Youth seizes the magical bow
the Youth returns
the Youth has arrived back
the Youth marries
```

Bibliography

[1] R. B. Bottenheimer. “Grimms‘ Bad Girls and Bold Boys: The Moral and Social Vision of the Tales”. In: (1989).

[2] S. Gudmundsdottir. “The Narrative Nature of Pedagogical Content Knowledge”. In: (1991).

[3] M. Rossiter. “Narrative and Stories in Adult Teaching and Learning”. In: (2002).

[4] R. Charon. “Narrative Medicine: A Model for Empathy, Reflection, Profession, and Trust”. In: (2001).

[5] G. Monk et al. “Narrative therapy in practice: The archaeology of hope”. In: (1997).

[6] J. Togelius et al. “What is Procedural Content Generation? Mario on the borderline”. In: (2011).

[7] *MoveCare: Multiple-actors Virtual Empathic Caregiver for the elder.* URL: `http://www.movecare-project.eu`.

[8] QL Xhue. “The Frailty Syndrome: Definition and Natural History”. In: (2012).

[9] *Giraff.* URL: `http://www.giraff.org`.

[10] Matteo Luperto et al. “A Multi-Actor Framework Centered around an Assistive Mobile Robot for Elderly People Living Alone”. In: (2018).

[11] *REWIRE: Rehabilitative Wayout in Responsive home Environments.* URL: `https://sites.google.com/site/projectrewire/`.

[12] K. Tanaka. “A Comparison of Exergaming Interfaces for Use in Rehabilitation Programs and Research”. In: (2012).

[13] A. Staiano and S. Calvert. “The promise of exergames as tools to measure physical health”. In: (2011).

[14] A. Staiano and S. Calvert. “The role of exergaming in Parkinson’s disease rehabilitation: a systematic review of the evidence”. In: (2014).

[15] Michele Pirovano et al. “Exergaming and rehabilitation: A methodology for the design of effective and safe therapeutic exergames”. In: (2016).

[16] JW Hung et al. “Randomized Comparison Trial of Balance Training by Using Exergaming and Conventional Weight-Shift Therapy in Patients With Chronic Stroke”. In: (2014).

[17] Robert J Vallerand et al. “The Academic Motivation Scale: A measure of intrinsic, extrinsic, and amotivation in education”. In: (1992).

[18] R DeCharms. “Personal causation. 1968”. In: ().

[19] Richard M Ryan and Edward L Deci. “Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being.” In: (2000).

[20] S. Detering. “Gamification: Using Game Design Elements in Non-Gaming Contexts”. In: (2011).

[21] Katie Seaborn and Deborah I Fels. “Gamification in theory and action: A survey”. In: (2015).

[22] Juho Hamari, Jonna Koivisto, and Harri Sarsa. “Does gamification work?– a literature review of empirical studies on gamification”. In: 47th Hawaii international conference on system sciences (HICSS), 2014.

[23] J. Hamari et al. “Challenging games help students learn: An empirical study on engagement, flow and immersion in game-based learning”. In: (2014).

[24] B. Monterràt and Elise Lavouè. “Toward Personalised Gamification for Learning Environments”. In: (2013).

[25] J. Hamari, B. Morschheuser, and J. Koivisto. "Gamification in Crowdsourcing: A Review". In: (2014).

[26] J. Hamari and J. Koivisto. "Social Motivations To Use Gamification: An Empirical Study Of Gamifying Exercise". In: (2015).

[27] J. Hamari. "Do badges increase user activity? A field experiment on effects of gamification". In: (2017).

[28] J. Hamari. "Transforming homo economicus into homo ludens: A field experiment on gamification in a utilitarian peer-to-peer trading service". In: (2017).

[29] O. Ruhi. "Level Up Your Strategy: Towards a Descriptive Framework for Meaningful Enterprise Gamification". In: (2015).

[30] Melanie C Green, Timothy C Brock, and Geoff F Kaufman. "Understanding media enjoyment: The role of transportation into narrative worlds". In: (2004).

[31] Penelope Sweetser and Peta Wyeth. "GameFlow: a model for evaluating player enjoyment in games". In: (2005).

[32] D. L. Parnas. "Designing Software for Ease of Extension and Contraction". In: (1979).

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

[34] N. Negroponte. "Soft Architecture Machines". In: (1975).

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

[36] H. P. Abbott. "Cambridge Introduction to Narrative". In: (2008).

[37] K. Kukkonen. ""Plot", in The Living Handbook of Narratology". In: (2014).

[38] M. L. Ryan. ""Space", in The Living Handbook of Narratology". In: (2014).

[39] S. Chatman. "Story and Discourse". In: (1980).

[40] M. Monfort. "Curveship's automatic narrative style". In: (2011).

[41] A. Jhala M. Young. “Cinematic Visual Discourse: Representation, Generation, and Evaluation”. In: (2010).

[42] J. Goguen D. Fox Harrell. “Style: A Computational and Conceptual Blending-Based Approach”. In: (2010).

[43] R. Swanson and A. Gordon. “Say Anything: A Massively Collaborative Open Domain Story Writing Companion”. In: (2008).

[44] B. Li, S. Lee-Urban, and M. Riedl. “Crowdsourcing Interactive Fiction Games”. In: (2011).

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

[46] I. Swartje and M. Theune. “A Fabula Model for Emergent Narrative”. In: (2006).

[47] M. Bal. “Narratology: Introduction to the
of Narrative”. In: (1985).

[48] T. Trabasso, PP. van den Broek, and S. Y. Suh. “Logical necessity and transitivity of causal relations in stories”. In: (1989).

[49] V. Propp. “Morphology of the Tale”. In: (1928).

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

[51] *Multilingual Folk Tale Database.* URL: `www.mdb.org`.

[52] S. Malec. “Autopropp: Toward the automatic markup, classification, and annotation of russian magic tales”. In: (2010).