Generating Game Map Layouts for Formally Specified Nonlinear Narratives

Saeed Amiri-Chimeh

Faculty of Computer Science and Engineering

Shahid Beheshti University

Tehran, Iran

s\_amirichimeh@sbu.ac.ir

Hassan Haghighi

Faculty of Computer Science and Engineering

Shahid Beheshti University

Tehran, Iran

h\_haghighi@sbu.ac.ir

Mojtaba Vahidi-Asl

Faculty of Computer Science and Engineering

Shahid Beheshti University

Tehran, Iran

mo\_vahidi@sbu.ac.ir

*Abstract*—Having a sophisticated and believable narrative that provides players with meaningful choices is an indicator of an enjoyable experience in many game genres. Authoring and debugging such narratives is a tedious task. Moreover, designing maps that support complex narratives is not a trivial process. Therefore, it is advantageous to provide authors with tools that prevent ambiguity, provide automatic validation, and facilitate procedural map generation. To this end, Ravi is a formal framework for authoring nonlinear narratives. This tool relies on formal narrative modeling and provides infrastructure for automatic assertion checking. However, the previous version of Ravi did not provide any features for map generation. In this paper, we extend Ravi to generate map layouts for formally specified narratives. We present a case study to demonstrate how the new features extend the capabilities of Ravi beyond an authoring tool and turn it into a procedural content generation one.

Keywords—nonlinear narratives; map generation; formal specification; procedural content generation

# Introduction

In many game genres, narrative is one of the pivotal aspects of the game. For example, it plays a substantial role in Role Playing Games (RPG), which have nonlinear and interactive narratives. The progression of such a narrative depends on player decisions, which might lead to different endings for the game [1]. Accordingly, providing players with different meaningful narrative choices increases the believability of their virtual experience and improves their immersion into the game.

Authoring a sophisticated nonlinear narrative with many branching storylines and endings is a difficult and complicated task. Therefore, usually, more than one author works on such big narratives, and ensuring consistency and integrity among different narrative branches requires extensive collaboration and care. Moreover, the characteristics of the game map are heavily influenced by the narrative when the game involves a sophisticated story. Thus, developers must collaborate closely with the authors to prevent misconceptions and resolve possible ambiguities to create a game map that supports a complex narrative. In this regard, even a minor misunderstanding or unresolved uncertainty might significantly surplus required time and budget and result in additional costs to the project.

Ravi [2] is a formal framework for authoring nonlinear narratives that prevents ambiguities by relying on rigorous, precise mathematical modeling. In addition, Ravi provides authors with an infrastructure for automatically checking and validating their narrative against user-defined formal assertions in order to ensure its integrity and consistency. Using this framework, which models the narrative as a multiple-entry finite automaton [3], the authors can define variables of the narrative state and unambiguously specify narrative interactions as formally defined choices.

However, the current version of Ravi does not provide any facility for generating map layouts that support the specified narrative. As mentioned before, the necessity of human communication between authors and map developers might become problematic when the narrative is complex; Therefore, this paper presents an extended version of Ravi, called SpaceR, which generates map layouts for specified narratives automatically.

Such a map layout should support the specified narrative. In other words, for every narrative event, there should be a proper location on the map, and any permitted player movement among these locations should be aligned with sequences of choices he/she would face. This layout can become the starting point for the developers and act as a referential artifact for tasks related to map generation. Furthermore, it can be used as the input of procedural content generation techniques that use algorithmic approaches to game map generation [4].

In the next section, we briefly review related works. Section 3 briefly explains key concepts of the Ravi framework, whereas Section 4 introduces SpaceR, our extension to Ravi for generating map layouts. Section 5 provides a simple case study to demonstrate the application of the introduced extension. Finally, we introduce some directions for future work and conclude this paper in Section 6.

# Related Works

This section overviewes tools and systems that bring formal specification to narrative authoring. It also mentions studies that generate supporting game maps for narratives.

In addition to Ravi, several other tools utilized the concept of formal specification to facilitate unambiguous and precise narrative authoring. For example, *Ceptre* [5] is a rule specification language intended to enable rapid prototyping for experimental game mechanics, especially in domains that depend on procedural generation and multi-agent simulation. Ceptre relies on a tradition of logical frameworks that use logical formulas to represent the rules of a system. Ceptre is based upon linear logic.

Another formal authoring tool is *Villanelle* [6]. This project is an approach to autonomous character authoring that integrates scripting with generativity. Villanelle uses a logic-based foundation and outlines its authors’ proposal for thinking about authoring languages rather than tools. In Villanelle, the authors examine interactive narrative authoring challenges around story characters and other aspects of the narrative environment external to the player.

Another related tool is *Boswell* [7], a web-based integrated development environment for writing, testing, playing, and sharing choice-based interactive narratives. Boswell models interactive narratives as context-free grammars. Accordingly, players implicitly generate syntax trees for those grammars when experiencing the game and interacting with its narrative.

1 https://github.com/SACHAM0RA/RaviFramework

There are very few studies that take narratives as inputs and generate supporting game maps. The most notable example is *GameForge* [8]. This system takes narrative content produced by a user, a computational story generation system, or other means, and builds a playable game map that supports the narrative. GameForge requires narratives to be provided as a list of plot points, high-level specifications of a period of time with a recognizable meaning. Also, plot points mention places where the narrative events are supposed to occur. Given a sequence of plot points, GameForge uses a genetic algorithm to generate supporting game map. It is worth mentioning that GameForge strictly concentrates on RPG games.

# Ravi Framework

Ravi1 is a Python package build upon a rigorous formal foundation for modeling and validating nonlinear narratives. In this section, we start by brieflyexplaining Ravi’s approach to narrative modeling. Then, we decribe Ravi’s key formal definitions and notations.

For any narrative, Ravi treats a narrative state as a valuation of the properties of a narrative context. A narrative context consists of several narrative entities, and each narrative entity encapsulates one or more properties. To better organize narrative entities, Ravi introduces the notion of entity classes. Every narrative entity is an instance of an entity class that declares the name and type of encapsulated properties. Moreover, Ravi allows classes to inherit from each other to increase their reusability.

Upon the narrative context, Ravi introduces concepts of filters and transforms. Filters determine if a state satisfies a specific condition, where transforms take a state and return a new one. Subsequently, Ravi defines a narrative choice as a pair of one filter and one transform. The filter of a narrative choice is called its precondition and determines the eligibility of a narrative choice in different states. The transform of a narrative choice is called its action and determines how the narrative state should change given that it satisfies the precondition.

Subsequently, to author a narrative with Ravi, one must declare the narrative entities for a context and specify choices to determine possible narrative interactions. Given these specifications, Ravi generates a narrative model that includes every possible sequence of choices and states. Moreover, it provides means to check the generated model against formally specified assertions automatically.

## Properties and Entity Classes

Ravi defines a **narrative property** as an ordered pair consisting of a symbol and a set. Given that is a property:

* p is named with the domain of .
* We write as it is a more convenient notation.

Subsequently, an **entity class** is defined as an n-tuple of properties. For any entity class , we define the domain of as .

## Narrative Entity and Narrative Context

In Ravi, a **narrative entity** is defined as the pair where is a symbol and is an entity class. Given that is a narrative entity:

* We say is an instance of named .
* For convenience, we write .
* Similar to properties, we say two narrative entities conflict if their names are the same.

Moreover, Ravi defines a ***narrative context*** as an n-tuple of narrative entities. Given a context in the form of :

* We define the narrative space of as

.

* We say is a narrative state of if and only if

.

Given any state of like , we reference by for any . Additionally, let us assume has properties. Consequently, we suppose . Hereafter, we reference by for any .

Also, given to be a member of , we define to be a clone of like where . We use this notation to create new states by substituting the value of an entity property in a given state.

## Narrative Filters, Transforms, and Choices

Given the narrative context , we refer to any function from to as a **narrative filter** for . Moreover, assuming be a filter for , Ravi defines . Furthermore, we refer to any function on as a **narrative transform** for .

Subsequently, a **narrative choice** in is the pair where is a filter, and is a transform for . We refer to as the set of all choices in .

If we assume be a member of and be a choice in , we can write if and only if:

(1)

## Narrative Model

Assuming be a narrative context, we say is a **narrative model** for if and only if is a multiple-entry finite automaton that can be represented by the 4-tuple , where:

* is a finite nonempty set of states.
* is a finite nonempty set of choices that act as the alphabet of the automaton.
* is a transition function that drives the automaton’s state alternation.
* is a finite nonempty set of termination states.

In summary, states of such automaton represent narrative states, and transitions resemble narrative choices.

Given that is a member of and is a narrative model, we write (read it as “ is reachable from in ”) if and only if is equal to or there is a sequence like in where:

(2)

Ravi considers to be consistent if and only if . Moreover, we say is terminable if and only if .

## Model Generation

Ravi provides an algorithm that automatically generates consistent narrative models. This algorithm takes three inputs:

* A set of initial states
* A set of choices
* A set of terminations filters

It starts from every initial state and checks if they satisfy the precondition of each choice. The algorithm applies the choice’s action on the current state to generate a new state for every eligible choice. This process is repeated for every new state that does not satisfy any termination filter.

# Map Layout Generation

This section introduces SpaceR, an extension to Ravi that includes new formal definitions and algorithms. These new features provide SpaceR with means to generate map layouts for narratives that are modeled by Ravi.

We present the map layout as an undirected graph, where vertices correspond to locations, and edges represent connectivity among locations. A critical property of a usable layout graph is its planarity. A planar graph is a graph that could be drawn on a plane without having any crossing edges [9]. Such a graph is the dual of neighboring areas on the plane (e.g., countries map). Therefore, the planarity of a layout graph ensures its usefulness in other procedural techniques that focus on generating fully realized virtual worlds.

We start by providing some definitions that extend the formal foundations of Ravi. Then, we propose a method for evaluating the importance of any possible connection between locations in order to generate map layouts that are better aligned with feasible sequences of player choices. Finally, we introduce two different approaches for generating layout graphs.

## Definitions

The following definitions add spatial features to the Ravi’s formalism.

### Location Sets and Location Constraints

We define a **location set** as a collection of symbols, each representing where the player may reside during the progression of the narrative. These locations constitute the set of vertices of the final layout graph. Assuming be a location set, we define **possible connections** of as . Members of this set resemble every possible edge of the final layout graph.

Now, let us assume that is a narrative model. We define a **location constraint** of as a total function from to . Such function maps every choice of the model to a location. By declaring such a mapping, game designers can determine *where* the players should access a specific choice.

In other words, assuming be a location constraint, players can access choice only if they are at location .

### Spatial Dependency Graphs

Given the location constraint for the narrative model , we consider the **Spatial Dependency Graph (SDG)** of and as the tuple where:

* is a total function from to such that

To simplify, an SDG is a directed graph created by converting states of a narrative model into vertices, turning its transitions into edges, and mapping its choices into location labels. Fig. 1 depicts an example model with eleven narrative states and seven choices, along with a location constraint and the resulting SDG.

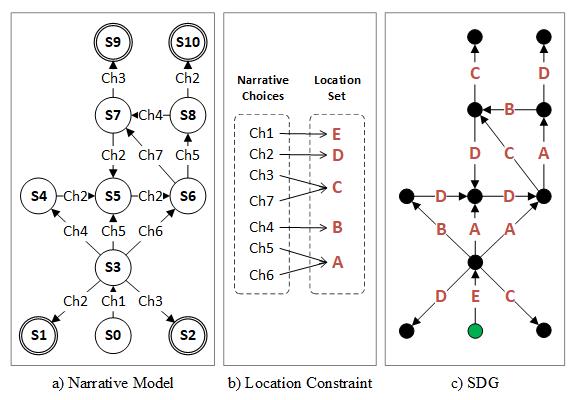


Fig 1: A narrative model, a location constraint, and their resulting SDG

## Evaluating the importance of possible edges

There are possible connections between members of a location set of size . For big values of , we might face an explosion in the number of possible connections. Thus, it is essential to evaluate the importance of any connection between any two locations. Such an evaluation can guide us in choosing pairs of vertices in a layout graph that are more eligible for having an edge.

Our evaluation strategy relies on the following two assumptions:

* If two choices are more likely to be made consecutively, the connection between their corresponding locations is more important.
* If the minimum number of steps for encountering a choice is less in comparison to another choice, then the corresponding location of the former is more important.

The first assumption is based on the intuition that the connectivity of the layout graph is better to be aligned with possible sequences of player choices. On the other hand, the second assumption prioritizes connections that become relevant early in the game. The motive behind this assumption is that players are rarely asked to travel far distances at the initial stages of the game.

According to these two assumptions, we define the following metrics for scoring a possible edge between two vertices like and . Please not that these layout vertices are also SDG labels by definition.

* **Frequency:** The number of SDG vertices that have a head edge labeled and a tail edge labeled , or vice versa.
* **Depth:** The minimum number of edges from a given set of initial vertices to an SDG vertex that has a head edge labeled and a tail edge labeled , or vice versa.

We prefer edges that are more frequent and less deep. To combine these two metrics into a single criterion, we define **edge importance** as follows.

* An edge is more important than another if it has a higher frequency
* Given two edges with equal frequencies, we say the one with the lowest depth is more important.
* Two edges are equally important if and only if their frequencies as well as their depths are equal.
* To automatically calculate these metrics for every possible edge of the layout graph, we give a recursive procedure that traverses nodes of a given SDG and updates the frequency and depth of every possible connection between locations. Algorithm 1 () presents the pseudocode of the recursive part of this procedure. In this algorithm, we process a single node of SDG, if not processed before (lines 1 and 2), to update the frequency and depth of its related layout edges (lines 3 to 9). Then, the procedure is recursively called for every adjacent node of the recently processed one (lines 11 to 17).

Subsequently, Algorithm 2 () indicates the pseudocode that initiates the recursive process. This algorithm takes a set of initial nodes from SDG and runs the procedure on them. Finally, it returns frequencies and depths of every possible edge of the layout graph.

1. processNode

|  |
| --- |
| **Inputs:**   * *:* a reference to the set of processed nodes * : |
| **Algorithm:**   1. **if** **then** 2. **return** 3. **for** every such that 4. **for** every such that   8. **if** **then** 10. **for** every such that 11. ( 12. ) |

Table 1 presents the result of running Algorithm 2 on the SDG depicted in Fig. 1 with the green node as the initial one. In this table, edges are ordered by their importance.

Table I. Frequency and depth of possible edges of the layout graph, ordered by importance

| Edge | Frequency | **Depth** |
| --- | --- | --- |
| *E, A* | 2 | 1 |
| *A, D* | 2 | 2 |
| *B, D* | 2 | 2 |
| *E, D* | 1 | 1 |
| *E, C* | 1 | 1 |
| *E, B* | 1 | 1 |
| *A, C* | 1 | 2 |
| *A, B* | 1 | 3 |
| *C, D* | 1 | 3 |
| *B, C* | 1 | 3 |

## Graph Generation

Here, we present two approaches for building the map layout graph. Both of these approaches require an ordered list of all possible edges. We obtain this ordered list by sorting all possible edges based on their importance.

The first approach prefers as much connectivity as possible. In this approach, we start from a complete graph where all pairs of locations are connected. Such a graph is probably not planar; hence, we remove the least important edges one by one until the result is planar. To keep the resulting graphs connected, we skip edges that destroy the connectivity of the graph.

1. GetEdgeMetrics

|  |
| --- |
| **Inputs:** |
| **Algorithm:**   1. for every 2. **for** every 4. ) 5. **return** |

Fig. 2 shows how this approach generates a planar and connected layout graph, considering the SDG mentioned in Fig. 1 and the importance of edges implied from Table 1.

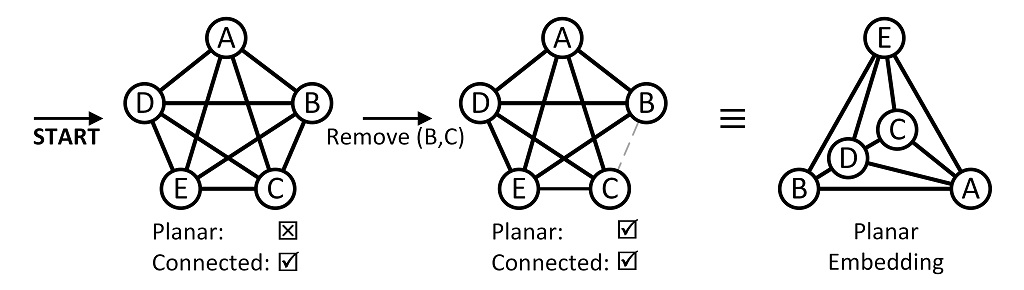


Fig 2: Generating layout graphs by removing least important edges

Conversely, the second approach prefers sparse layout graphs. In this approach, we start from a graph with no edges. Then, we gradually add the most important edges until the resulting graph is fully connected.

Fig. 3 shows how this approach generates sparse, planar, and connected layout graphs, considering the SDG mentioned in Fig. 1 and the importance of edges implied from Table 1.

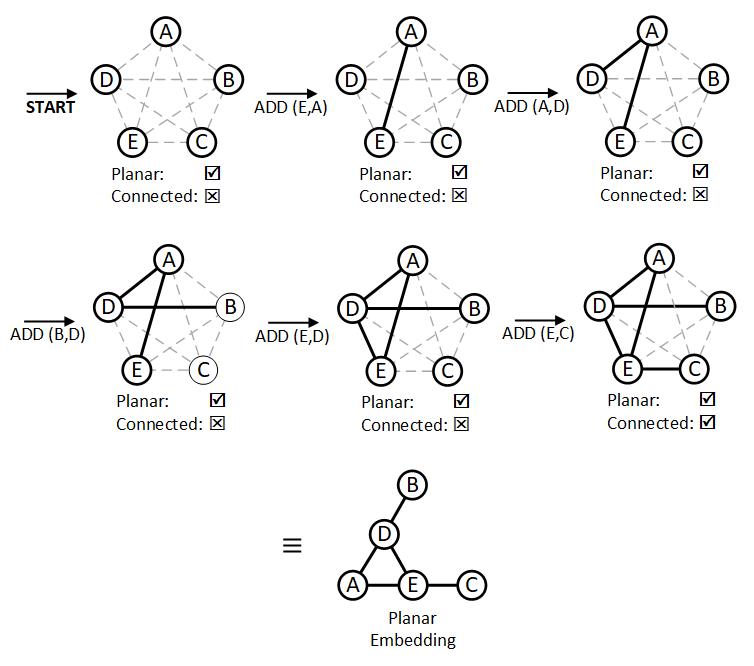


Fig 3: Generating layout graphs by adding most important edges

# Case Study

Throughout this section, we present a case study to demonstrate how our approach to layout generation works for the following interactive narrative.

*Initially, the player must choose to join one of two factions. The first one is called the* ***Army of Virtue****, led by the* ***General*** *and located at the* ***Mountains****. The other one is called* ***Clan if Chaos****, led by the* ***Preceptor*** *and located in the* ***Jungle****. If players ally with the Army, they are presented with two options. First, they can fight with the Preceptor at* ***Riverside*** *and defeat the Clan. Second, they can betray the Army and institute a new faction called* ***Order of Doubt*** *near the* ***Sea Shores****. If they choose the latter, they can fight one or both of the two other factions.*

*Similarly, if players join the Clan, they are presented with two options. They can fight with the General in the* ***Desert*** *and defeat the Army; or, they can betray the Clan and institute the Order of Doubt. If they choose the latter, they can fight one or both of the two other factions.*

*There are three possible endings for this narrative. If players defeat the General, they can ignite anarchy by burning the* ***City****. Moreover, if players defeat the Preceptor, they can end all wars at the expense of establishing a tyranny in the city. If both factions are defeated, they can give people the authority to govern their destiny by declaring an statement near the Shores.*

First, we have to formally specify the mentioned narrative using Ravi’s formal definitions. We start by introducing three entity classes of *character*, *player*, and *world*.

(3)

Where,

(4)

Now, we define the context of the mentioned narrative based on these classes.

(5)

As an example, is a state of , which means:

Considering this context, we define the following narrative filters.

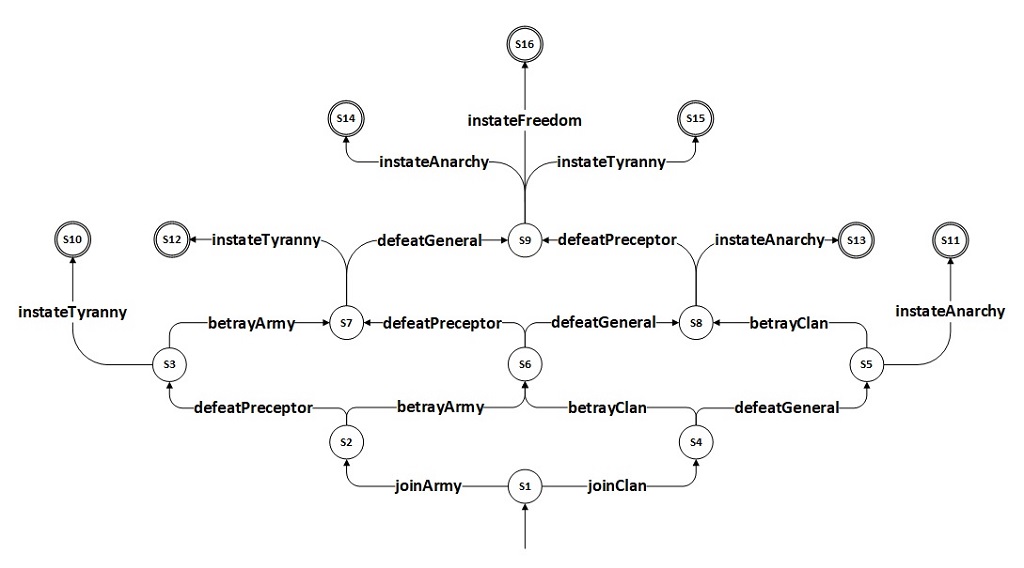
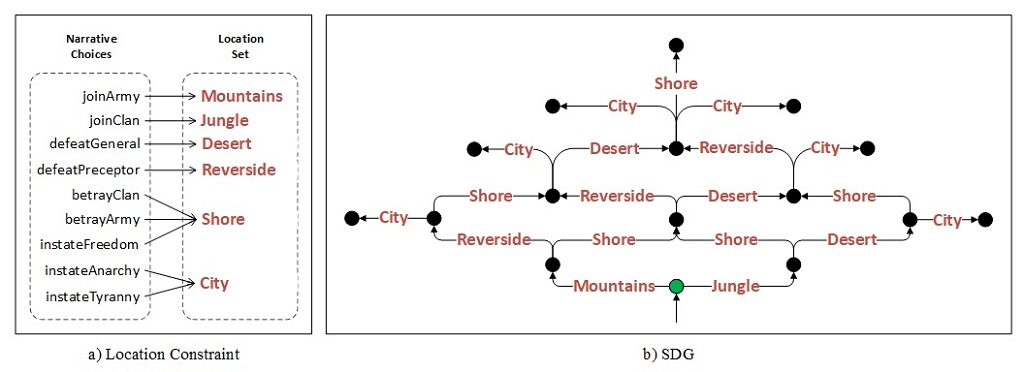
Also, we define the following narrative transforms.

Fig 5: The location constraint and the SDG of the narrative in our case study

Fig 4: The model of the narrative in our case study

(7)

Considering the presented filters and transforms, we define the following narrative choices.

(8)

As the example narrative implies, the initial state should be . Moreover, we can consider as the termination condition of this interactive narrative. Given the presented narrative choices, the mentioned initial state, and this termination condition, Ravi generates the narrative model presented in Fig. 4.

Now, we have to define a location constraint for this model based on the given narrative. Fig. 5 presents such location constraint and its resulting SDG. Accordingly, Table 2 presents the frequency and depth of every possible neighborhood between mentioned locations in the given narrative, ordered by their importance.

Table III. Frequency and depth of possible neighborhoods between locations of the narrative

| Edge | Frequency | **Depth** |
| --- | --- | --- |
| *Shore, Riverside* | 4 | 2 |
| *Shore, Desert* | 4 | 2 |
| *Riverside, City* | 4 | 2 |
| *Desert, City* | 4 | 2 |
| *Riverside, Desert* | 2 | 3 |
| *Mountain, Riverside* | 1 | 1 |
| *Mountain, Shore* | 1 | 1 |
| *Jungle, Desert* | 1 | 1 |
| *Jungle, Shore* | 1 | 1 |
| *Mountain, Jungle* | 0 |  |
| *Mountain, Desert* | 0 |  |
| *Mountain, City* | 0 |  |
| *Jungle, City* | 0 |  |
| *Jungle, Riverside* | 0 |  |
| *Shore, City* | 0 |  |

Based on these values, Fig. 6 presents two layout graphs for the given narrative. One of these layouts results from the connectivity-preferring approach, whereas the other is obtained from the sparse-preferring approach.

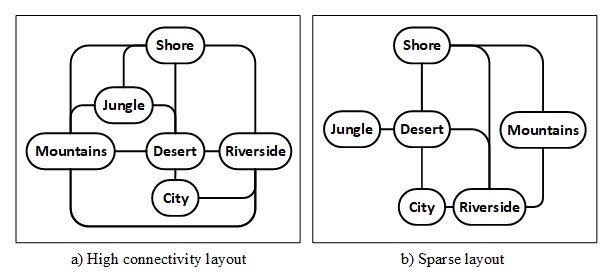


Fig 6: Resulting map layouts for the narrative

# Conclusion

Authoring nonlinear narratives and developing supporting game maps are complex tasks and susceptible to ambiguity and misunderstanding. Therefore, it is worth providing authoring tools that prevent uncertainties and generate abstract map structures for the authored narratives. To this end, we presented an extension to Ravi, a framework for authoring and validating nonlinear narratives, which relies on rigorous formal foundations. This extension stretches the capabilities of Ravi from a narrative authoring system to a procedural content generation system and enables it to generate abstract map layouts that support formally specified narratives. We demonstrated the application of this new feature through a case study.

Our approach to map layout generation is currently based on particular assumptions regarding the characteristics of a good map layout for a given narratives. Specifically, it is assumed that a connection between two locations is more important than others if the locations are related to more consecutive narrative interactions. Also, we assumed that early player interactions increase the importance of related location connections. Although intuitive, these assumptions are not based on careful investigations of game maps and their dependency on the underlying narrative. In the future, we plan to comprehensively study the relationship between narrative interactions and location adjacencies to refine the assumptions mentioned above and improve the usability of the presented methods.

##### References

1. S.L. Bowman, “Immersion and shared imagination in role-playing games.” In Role-Playing Game Studies, pp. 379-394. Routledge, 2018.
2. S. Amiri-Chimeh, H. Haghighi, and M. Vahidi-Asl, “Ravi: A Formal Framework for Authoring Interactive Narratives.” unpublished.
3. A. Gill and L.T. Kou, “Multiple-entry finite automata.” Journal of Computer and System Sciences 9, no. 1 (1974): 1-19.
4. J. Togelius, G.N. Yannakakis, K.O. Stanley, and C. Browne, “Search-based procedural content generation: A taxonomy and survey.” IEEE Transactions on Computational Intelligence and AI in Games 3, no. 3 (2011): 172-186.
5. C. Martens, “Ceptre: A language for modeling generative interactive systems.” In Eleventh artificial intelligence and interactive digital entertainment conference. 2015.
6. C. Martens, O. Iqbal, S. Azad, M. Ingling, A. Mosolf, E. McCamey, and J. Timmer, “Villanelle: Towards Authorable Autonomous Characters in Interactive Narrative.” In INT/WICED@ AIIDE. 2018.
7. I. Holmes, “A web-based editor for multiplayer choice games.” In FDG. 2014.
8. K. Hartsook, A. Zook, S. Das, and M.O. Riedl, “Toward supporting stories with procedurally generated game worlds.” In 2011 IEEE Conference on Computational Intelligence and Games (CIG’11), pp. 297-304. IEEE, 2011.
9. R.J. Trudeau, Introduction to Graph Theory, Dover publications, 1993.