Towards Story-Based Content Generation: From Plot-Points to Maps

Josep Valls-Vargas
Drexel University
Philadelphia, PA, USA
Email: josep@valls.name

Santiago Ontañón
Drexel University
Philadelphia, PA, USA
Email: santi@cs.drexel.edu

Jichen Zhu
Drexel University
Philadelphia, PA, USA
Email: jichen@drexel.edu

***Abstract*—Some computer game genres require meaningful stories and complex worlds in order to successfully engage players. In this paper we look at a procedural approach to story-based map generation focusing on the tight relationship between stories and the virtual worlds where those stories will unfold. Our long term goal is to develop procedural content generation techniques that can produce maps supporting multiple stories. We present an approach that takes, as input, the specification of a story space as a collection of *plot points*. Causal relations between these plot points and spatial relationships between locations define different story and spatial structures. Our system generates multiple configurations of a map, determines the stories that are actually supported in each map, and evaluates their quality, in order to find maps that support high quality stories from a storytelling perspective.**

I. Introduction

Procedural content generation (PCG) refers to the creation of content automatically through algorithmic means [1]. Some computer game genres require meaningful stories and complex worlds in order to successfully engage players. However, most procedural map generation techniques typically neglect the role of the story in the construction of the map. In this paper, we present a novel PCG approach for generating maps intended to be played in a computer game context that uses stories to guide the generation process.

Our approach is motivated by the following observations: First, procedural content generation has the potential to increase the variability and replayability of games. This in turn can lead to an increase in player interest in these games, as it will take longer for the player to see or complete everything in the game [2]. Second, game maps impose constraints to the set of possible stories that may happen in a game. For example, the order of narrative events should not be at odds with the spatial configuration of the map. For example, we may not want a player of a murder mystery game to leave the current room before the next critical narrative event can be triggered.

In this paper, we intend to explore the correlation between stories, represented as sequences of plot points, and the spatial configuration of the environment where these stories unfold. Our goal is to procedurally generate story-based game maps. We propose a framework which, given the specification of a *story space*, represented as a collection of *plot points* and their dependencies, can generate maps that support one or more stories from that story space. Our system searches in the space of possible spatial configurations of the map, determining the set of stories that can unfold in each of those configurations. Then, using automatic story evaluation techniques, it determines the narrative quality of each possible story, the combination of which determines the overall quality of the map. Finally, the map is *realized* into a graphical representation.

The key technical challenges that we address here are: 1) how to automatically generate game maps from a collection of plot points, 2) how to evaluate the quality of a given map based on the different stories that it supports, and 3) how to generate a graphical realization of the map.

The remainder of this paper is organized as follows: Section II presents the problem of procedural map and story generation, including a brief overview of existing work on the area. In Section III we describe our approach of story-based map generation and we give details of the different steps involved in the process. In Section IV we describe our experimental evaluation for the proposed system. In Section V the paper closes with conclusions and future work.

II. Procedural Map Generation

Designing a spatial environment for action, adventure or role-playing computer games is a complex process. As argued above, it requires the consideration of spatial requirements and narrative requirements as well as their interconnection. Existing work has been done in interactive narrative and procedural map generation separately from one anther in most cases. On the one hand, there has been extensive work towards procedurally generating different types of spatial structures, focusing mostly on terrain and urban environments [3], [4]. On the other hand, there has also been a significant amount of work on automatic story generation and interactive storytelling [5], [6]. Both bodies of work have been applied to computer games, the former in several forms, especially maps for dungeon crawlers and sandbox simulations (e.g. Rogue, Minecraft) [7]; and the later through the concepts of experience and drama management (e.g. PaSSAGE, Left 4 Dead) [8], [9]. Doran and Parberry [2] describe a system capable of procedurally generating stories or quests for computer role-playing games. Tomai [10] presents another system that can adapt quests for a predefined virtual world. Recent work has started to integrate narrative components into map generation. Game Forge, by Hartsook et al. [11], is a system capable of generating a map given a linear story represented by a sequence of plot points. Their system uses a genetic algorithm approach to infer the spatial relationships between the locations annotated in the given story and generates a map genotype as a space tree. In a second step, the system maps

978-1-4673-5311-3/13/$31.00 ©2013 IEEE

the genotype to a phenotype where the space tree has been embedded on a grid. Then, the virtual world is graphically realized as a 2D map by instantiating predefined image tiles and handed to a game engine so that it can be navigated by the player's avatar. Dormans and Bakkes [12] introduce the idea of stories and spatial environments (referred to *missions* and *spaces* in their work) as separate structures; the first holding the logical causal relationship between the sequence of events and the second one the spatial description of the playable map where the story will unfold (or where the player will execute their *mission*). They describe the use of generative grammars to generate *missions* as a sequence of *tasks*, and then they break the generated mission linearity by using graph grammars and the idea of *locks and keys* [13]. Their system uses a rubber band-based algorithm to layout the graph and a shape grammar to realize it into a playable game map. In CHARBITAT, Nitsche et al. [14] propose a system that uses the *lock and key* model and drama management techniques to generate a game map on the fly. Their system places pre-generated elements into a randomly generated map to unfold a predefined story. Even though some of the more prominent approaches to procedural map generation depended on randomness, Hullett and Mateas explored alternatives that would yield a higher level of internal consistency and believability for training simulations and games [15]. Constraint solvers and hierarchical planners have also been used successfully for procedurally generating spatial environments tailored to videogames [16] but those methods seem to deviate from our proposition to use narrative tools as basis for map generation.

The idea of performing a search in separate spaces has already been described in the literature. For example, the approach of Dormans and Bakkes [12] uses two search spaces (*missions* and *spaces*), and in the work of Hartsook et al.[11] the story is an input to the system, and they search only in the map space. These approaches isolate the story and map spaces and limit the story space into a linear or a tree-based structure. In the model of Dormans and Bakkes [12], for example, a mission (or, more generally, a story) is generated. Then when the map is being generated, there are rules in place to avoid cycles in order to prevent short circuiting the mission by accidentally connecting the final room to a room near the entrance.

In this paper we propose a system that uses storytelling techniques to generate and evaluate both stories and spaces. Once a satisfying pair has been found it proceeds to the game map realization. Our approach enables us to generate complex game maps, including cycles and allowing non-linear or multiple story lines.

In summary, a player's narrative experience in action, adventure or role-playing computer games is related to how the space can be navigated. Due to this tight relationship between the story and spatial structures, a holistic approach to procedural map generation should take into consideration the causal and temporal relationships in the story structure and preserve them in the generated spatial structure.

III. STORY-BASED MAP GENERATION

In this paper, we focus on the problem of story-based map generation. In order to address this problem, we present a system that combines story-telling and procedural map generation techniques. Specifically, our system takes as input a collection of *plot points*, which are events that are key to the game story (for example: "the player discovers the existence of a hidden door"). This collection of plot points defines the *story space* (e.g. the space of all the potential stories we could generate). The goal of our map generation system is to generate game maps that support a subset of stories from the story space which are of high-quality based on our given storytelling criteria.

Our plot point representation is based on those used in planning-based story generation systems, such as ASD [8]. It is extended in order to allow the use of story quality evaluation functions, such as those developed by Weyhrauch for MOE [17]. Additionally, we take as input a set of author goals that stories need to accomplish [8]. Specifically, the input of our system is a tuple $\langle S, L, player, I, G, P\rangle$, where:

- S is a list of symbols (e.g. *tree*, *sword*, *cave*, *warrior*, *use*, *path*, etc.).
- $L \subseteq S$ is the subset of those symbols that represent location names (e.g. *cave*).
- $player \in S$ is the symbol that identifies the player.
- I is the initial state, consisting of a list of positive literals of the form $o(s_1, ..., s_n)$, where $o \in S$, and $\forall_{i=1...n} s_i \in S$. The initial state can include information about NPCs, objects, and locations that are referenced by the plot points (e.g. $at(path, warrior)$, $has(warrior, sword)$...).
- G is the set of goals that need to be achieved to complete the game, represented as a list of literals.
- P is a set of plot points. A plot point is defined as a tuple $p = \langle \kappa, \alpha, \delta, A\rangle$ (in a very similar way to how planning operators are defined), where:
  - κ represents the preconditions of the plot point, as a list of (potentially negated) literals.
  - α (or the *add* list) are the literals that will become true after this plot point occurs.
  - δ (or the *delete* list) are the literals that will become false after this plot point occurs.
  - A is a list of additional annotations we might need for this plot point. In our particular implementation, we annotated each plot point with the set of features required to use a story evaluation function inspired by the work of Weyhrauch [17] (which we elaborate later in Section III-B).

Given the input, our system generates, as output, a graphically realized map based on a rectangular grid. Each cell of the grid is annotated with a location symbol in L, the objects or characters that should initially be in that location, and which of the four neighboring cells the current cell is connected to.

Internally, our system uses an abstract intermediate representation of maps before they are graphically realized. We call this representation a *spatial configuration* graph. Formally, a spatial configuration is defined as a graph $E = \langle L, K\rangle$, where the nodes are the locations L and $K \subseteq L \times L$ are the edges

between those nodes. An edge connecting two nodes represents that the locations are accessible from each other. Later, we transform the spatial configuration into a planar, grid-based, orthogonal graph (we call this the *graph embedding* step). Once embedded, the graph can be realized as a playable game map (we call this the *graphical realization* step).

The main steps of the process can be summarized as follows: First, our system generates spatial configurations. Then, the system generates all the possible stories that can unfold in a given spatial configuration, based on the set of input plot points. After that, each spatial configuration is evaluated by assessing the narrative quality of all possible stories that could unfold in it, and, finally the system graphically realizes one of the spatial configurations.

The entire process takes into account three different spaces: 1) the *spatial configuration space* (which locations are connected to each other), 2) the *story space* (sequences of plot points), and 3) the *graph embedding space* (planar graph embeddings which will become graphically realized game maps, into rectangular grids in our experiments).

The input of the system defines a set of symbols L that represent locations. However, their spatial relationships (from which ones characters can reach others) are not specified in the input of our system. The first step described in Section III-A assumes as input a spatial configuration graph $E = \langle L, K \rangle$, describing which locations are accessible from one another. The graph may be initialized randomly. Section III-F describes the ϵ-greedy algorithm that uses the evaluation of the graph E from Section III-C to refine the construction of E for the next iteration.

A. Planning and Story Generation

From a given spatial configuration graph E and a given input $\langle S, L, \ player, I, G, P \rangle$, we want to know what stories can unfold in E. We perform an exhaustive search to generate all possible stories. We are interested in identifying spatial configurations that feature low quality stories (e.g. short circuited or without any challenges) or no stories at all (e.g. no way to reach any goal) so that can be discarded.

Our system uses a forward chaining planner in order to generate stories. The given plot points are used as the planning operators. A story consists of the list of plot points generated by the planner for reaching the goals G from the initial state. The initial state used by our planner to generate stories consists of the initial state I expanded with one literal $path(l_1, l_2)$, for each edge (l_1, l_2) that exists in E.

Moreover, we are interested in obtaining the set of *all* possible stories that can unfold in a given spatial configuration. In order to obtain such set, our planner performs breadth-first-search. When a state is reached where all the goals in G have been satisfied, the operator history is saved in the set of solution candidates $\mathbb{S}_E$ for the current spatial configuration graph. Then the planning process continues with the next branch. If it is not possible to find operators that add new literals to the state, the search along a branch will be terminated even if there are goals remaining, therefore preventing loops that may contribute to solutions of infinite length. Moreover, our planner gives a special treatment to movement plot points

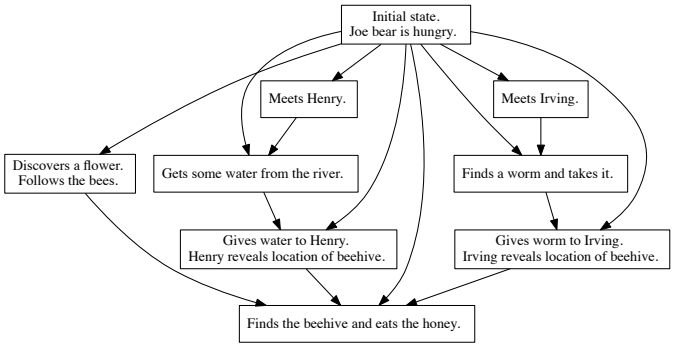

Fig. 1. Example story causal link graph inspired by the plots produced by TALE-SPIN. Boxes represent plot points and arrows precondition/postcondition dependencies.

and adds a literal ($been(character, l)$) after each movement action in order to allow different locations to be explored and revisited but preventing duplicated search states (e.g. visiting a bird before and after having acquired bird food). Also, some of the literals in the state are dealt with in a special way by some modules in our system:

- $at(object, l)$: Specifies object or character locations in the spatial environment.
- $path(l_1, l_2)$: Specifies whether a path exists from a location to another.
- $locked(character, l_1, l_2)$: Specifies whether a path exists but is currently locked for a character.

The output of the planning module is a set of plans $\mathbb{S}_E$ representing all the stories found in the current spatial configuration. Each story in $\mathbb{S}_E$ consists of an ordered list of pairs $[\langle s_1, p_1 \rangle, ...\langle s_m, p_m \rangle]$ with the states s_i and the executed plot points $p_i \in P$. More complex planners can be used, but our current breadth-first forward-checking planner suffices for the purposes of our experiments. Also, for large story spaces, it might not be feasible to generate the complete set of all stories that can unfold in a given spatial configuration, and we might need to sample it. Sampling should suffice as long as we can identify the lowest quality stories, which in our experiments are correlated with the shorter stories (e.g. when short circuiting the initial plot point with some goal). This idea is part of our future work.

Figure 1 shows an example plot point dependency graph. The story is inspired by plots produced by TALE-SPIN [18]. Some of the preconditions have been removed for display. Figure 2 shows a specific story for the plot point dependency graph shown in Figure 1. The main character in this story is *Joe Bear*, who starts at the location *cave*. The numbers in the edges in Figure 2 represent the order in which the plot points occur.

B. Story Evaluation

An important component of our system is the story evaluation which has the goal of computing a numeric value representing the quality of a story according to some predefined criteria to a given story $[\langle s_1, p_1 \rangle, ...\langle s_m, p_m \rangle]$.

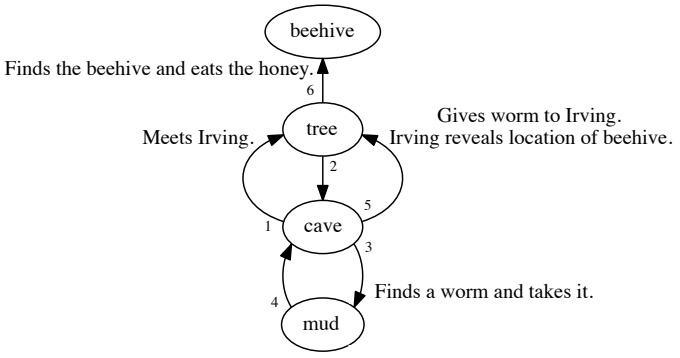

Fig. 2. Example story planned from the story space described by Figure 1. Nodes represent locations, edges represent movement between locations and are annotated with the ordered plot points describing a story.

The evaluation function used in our experiments, draws from the fields of experience and drama management, specifically from the work of Weyhrauch [17]. Our evaluation function is a weighted sum of a set of simpler normalized feature values. The selected features try to evaluate certain logical and aesthetic properties that characterize story quality from a storytelling point of view:

- **Thought flow**: Measures whether one event in the player's experience relates logically to the next. Each plot point may be annotated with a *thought tag* grouping logically related events (e.g. visiting the bird and acquiring bird food share one tag whereas seeing a flower and following some bees may have different tags). We implemented this feature value counting the total number of distinct thought groups in a given story normalized by the number of transitions between distinct thought groups.
- **Activity flow**: Measures boredom of the player caused by moving without unfolding new events of the story. We measure the activity flow by the total number of distinct visited locations normalized by the total number of transitions between locations.
- **Manipulation**: Measures how manipulated the player feels by the limit of meaningful choices available. We measure the manipulation by computing the average of the number of available operators before executing each plot point in the story normalizing by the total number of operators.
- **Intensity**: Measures the player's excitement built as the story unfolds. Each plot point may be annotated with a numeric *intensity value* describing the authors intended intensity for an event. We measure the difference between a prototype dramatic arc and the dramatic arc represented by the sequence of intensities annotated in the current story. Any particular desired dramatic arc shape could be specified. In our experiments we approximate an Aristotelian arc by $f(x) = \sin(\pi x^2)$ where x is in the interval $[0, 1]$.
- **Action consistency**: Measures the player perceived logic in a sequence of actions by penalizing redundant or complementary occurrences. Each plot point may be annotated with one or more *complementary thought tags* describing logically incompatible events to the current plot point. We measure the number of violations normalized by the total number of distinct thoughts.
- **Length**: Measures how much the story length deviates from a desired story length. We implemented this feature value by comparing the current story length to a predefined value (part of the input annotations). We return a value interpolated linearly from 1.0 when the lengths match exactly to 0.0 when the current story length is $\pm 75\%$ or more.

Finally the individual values are added together using a weighted sum. In our implementation we defined some arbitrary weights representing our desired aesthetic values. The values used in our experiments are $0.1, 0.05, 0.05, 0.1, 0.1$ and 0.6 respectively, for each of the individual features defined previously.

C. Spatial Configuration Evaluation

Given the set of stories $\mathbb{S}_E$ that can unfold in a spatial configuration E (computed by our planner), and the evaluation of each of the stories in $\mathbb{S}_E$, we are interested in aggregating the evaluations and compute a numeric value representing the quality of the spatial configuration E.

In order to give a numerical value to the list of individual story evaluations we experimented with several aggregation operators with different properties and we selected the following:

- Average: We average the list of individual evaluations.
- Minimum: We compute the minimum of the list of individual evaluations (which identifies the worst story).

D. Graph Embedding

Once a spatial configuration E has been selected (we describe how the previous processes are combined to select a spatial configuration in Section III-F), we need to turn the graph E into a planar graph (a graph that can be drawn in a plane without having intersecting edges), so that a final map can be realized. We call this output a *graph embedding* $\mathbb{Y}_E$.

There is an infinite number of drawings or plane embeddings for a graph. When embedding a spatial configuration E, we would like to take into account a variety of factors. First, allowing two edges to cross, will result in the creation of additional paths, not in E, when the spatial configuration is realized, allowing the player to access locations that should not be readily accessible. Even though those can be tackled with multi-level environments or the use of certain elements, those may not make sense in some story domains (like teleports in a historical setting) or scales (like multi-level continental maps). Thus, in order to provide a story-agnostic graph handling we decided to enforce planar embeddings.

When embedding E into a planar graph, we consider edges to be undirected since we are assuming that locations will be accessible from each other when connected.

Several hierarchical and force-driven (e.g. rubber bands) algorithms have been discovered for finding planar embeddings,

but those may not be suitable for our purpose since we impose several aesthetic and playability constrains. Thus, we decided to use a rectangular grid layout for our graph embedding step (although our algorithm would work for arbitrary reticules, such as hexagonal patterns). We employ a technique inspired by Goldschmidt and Takvorian [19].

Before embedding the graph, two preprocessing steps are executed in order to improve the chances of finding a planar embedding:

1) Inspired by the idea of the *Steiner points* in the *Steiner tree problem* [20], the graph is searched for cliques of size $>= 4$. If any is found, the edges between the nodes are removed and replaced by vertices to a new intermediate node.
2) In order to enable orthogonalization on a rectangular grid, nodes with a vertex degree > 4 (vertices with more than four edges) are split into several nodes with vertex degree of at most 4 following the *Giotto* approach presented by Tamassia [21].

Then, the graph is processed as follows:

1) Select a subpath from the graph, lay down the nodes consecutively on the grid and mark the nodes as processed. The selected subpath is initially the longest of all the shortest paths in the graph.
2) Select an unprocessed node (if any) that has only one neighbor already marked processed, lay it in the grid adjunct to its neighbor and mark it as processed. The neighbor position is selected from the first available position searching clockwise.
3) Select an unprocessed node (if any) and search for paths between all the neighbors already marked as processed. Select the shortest path and split the node as many items as necessary to cover all the steps in the path and lay the items in the grid, marking the node as processed. The paths are searched using A* and Manhattan distance.
4) Iterate until there are no more nodes to process or the remaining nodes cannot be processed.

If any node remains after processing, our embedding algorithm returns a failure token, otherwise it returns a graph embedding defined as the tuple $\mathbb{Y}_E = \langle E, coords \rangle$ where $coords$ are $\langle x, y \rangle$ coordinate pairs for each node in E.

Figure 3 shows an example spatial configuration graph. Figure 4 shows an example graph embedding for that spatial configuration graph. We used letters rather than the full names of the locations for space reasons (e.g. *a* for *log*, *b* for *beehive*...). It can be observed how a node (*cave*) has been split multiple times (first into *u* and *v*, and later *v* has been split again to form a path).

The algorithm described in this section is not guaranteed to find a planar embedding for a graph, even with the preprocessing steps defined previously. If no embedding is found, the process will backtrack to the first step and will select the subsequent longest of all the shortest paths. If no graph embedding is found a failure token will be returned that will prompt the ϵ-greedy search to continue to another spatial configuration graph.

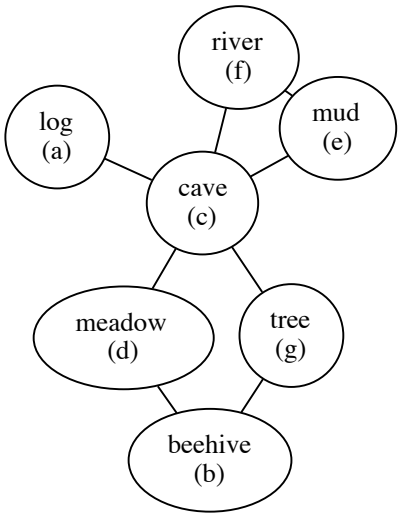

Fig. 3. Example environment configuration for an environment inspired by the plots produced by TALE-SPIN (input A in our experiments).

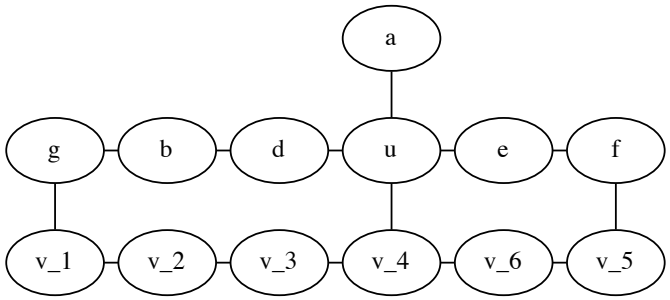

Fig. 4. Example environment configuration embedding for the environment in Figure 3.

E. Graphical realization

Given a graph embedding $\mathbb{Y}_E$, we want to compute a graphical realization for the spatial configuration E. Once a grid layout has been found, we proceed to the graphical realization by instantiating grid cells at the coordinates given by $\mathbb{Y}_E$ with a bitmap representation for rooms, doors for connected rooms and areas for split nodes. Figure 5 shows our graphical realization output for the graph embedding shown on Figure 4.

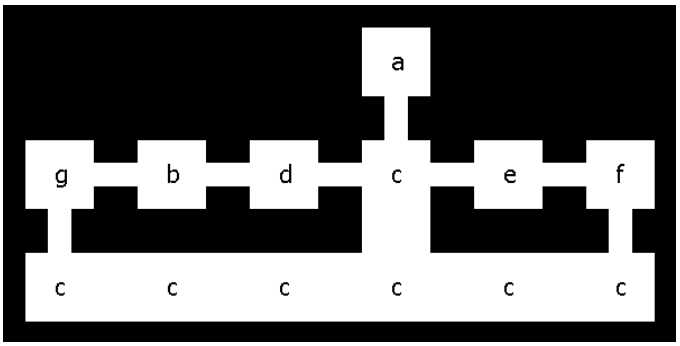

Fig. 5. Example graphical realization for the graph embedding in Figure 4.

F. Algorithm construction

This section describes the combination of the steps presented so far into an unified algorithm to generate a fully realized map, given the input to the system, $\langle S, L, player, I, G, P\rangle$.

Our algorithm works by performing two main tasks: first, decide on a *spatial configuration graph*, specifying which locations are connected with each other; then, embed and realize the given spatial configuration. One of the main difficulties on choosing a spatial configuration graph, is that there exist $2^{\binom{|L|}{2}}$ different spatial configurations for a given story world specification. Thus, we propose an ϵ-greedy strategy to search for a good candidate in the spatial configuration space. Spatial configuration graphs candidates with different features will be built and evaluated. Based on the features selected and the evaluation obtained, subsequent graphs will be built. Once a satisfactory candidate is identified, the algorithm will proceed with embedding and realization steps.

Using the processes defined in the previous sections, our system executes the following steps:

1) Initialize two tables, M and N:
   a) M has $|L|-1$ rows, one entry for each of the possible number of edges in E. All entries are initialized to zero.
   b) N has $\frac{|L|(|L|-1)}{2}$ rows, one entry for each possible edge in E. All entries are initialized to zero.
2) Create a candidate spatial configuration E by:
   a) Using an ϵ-greedy sampling strategy, select a number of edges m form the table M (e.g. select the maximum entry with probability $1-\epsilon$, and one at random with probability ϵ). In our experiments $\epsilon = 0.3$.
   b) Select m edges by using an ϵ-greedy sampling strategy in table N (e.g. we use the ϵ-greedy strategy m times, to get m edges, without replication).
3) Using our planner, generate all possible sequences of plot points (e.g. stories) that could unfold in the given spatial configuration E.
4) Evaluate each individual story using our evaluation function.
5) Aggregate the individual story evaluations to obtain an evaluation for the spatial configuration E.
6) If the evaluation of E is below a threshold, update the tables M and N with the evaluation of E (each entry in M stores the average evaluation of the spatial configurations that have a given number of edges, and each entry in N stores the average evaluation of a particular edge in the spatial configurations), and go back to step 2. Otherwise, go to step 7.
7) Generate a planar embedding $\mathbb{Y}_E$. If none found, go to step 2.
8) Realize the planar embedding $\mathbb{Y}_E$ into a graphical representation (a grid in our experiments).

IV. Experimental Evaluation

In order to evaluate our system, we designed three experiments reported in the following subsections. The first experiment aims at evaluating our planner used for story generation. A second experiment tests our ϵ-greedy search strategy for generating spatial configurations along with the story and configuration evaluation functions. Our third experiment tests the competence of our graph embedding algorithm.

A. Planning and Story Generation

Our first experiment aims at evaluating the behavior of our story planner for inputs with different properties. We experimented with three different inputs to our system, which we altered in order to demonstrate the effects of adding and removing goals, plot points, locations and paths between locations:

- Input A is based on the plots produced by TALE-SPIN, specifically the *Joe Bear,* Irving Bird and *Henry Ant stories. It is composed of 8 plot points and 7 locations.*
- *Input B features a key and lock puzzle that requires revisiting some locations in order to accomplish the given goal. It is composed of 8 plot points and 6 locations.*
- *Input C is a task-based story inspired in the Tolkien universe. The player is presented with a set of 14 tasks represented as goals that must be satisfied. It is composed of 26 plot points and 11 locations.*

As we expected, the spatial configuration parameters have great impact on the number of stories found. Table I shows the results from this experiment. Each row shows one different experiment using a specific variation of one of the three inputs described above, paired with a specific spatial configuration E. The first five columns show the given base input, the number of plot points, locations, the number of edges in E, and the number of goals in each experiment. The last column ($|\mathbb{S}_E|$), show the total number of stories that our planner could find in each of the experiments.

The first three rows show results for input A when using a spatial configuration with a predefined number of edges (6), with no edges (0) and with all the possible edges (21). As expected, with no edges, no stories could be found. Rows 4 and 5 show two variations of input B, where in the second one, we added an extra location and two additional edges to E. The result is that the number of stories that can unfold grows significantly (from 2 to 21). Notice that we are not measuring story quality yet. Finally, the last three rows show three variations of input C, where we varied the number of goals (from 4 to 14), and also varied the number of locations and edges. As we can see, adding goals constraints the planner to trying to satisfy each goal and therefore reducing the number of different stories found. The results also show the exponential relationship between the number of plot points and the number of stories found.

B. Spatial Configuration

In order to validate our spatial configuration ϵ-greedy search process, we compared the configuration evaluations obtained against those obtained by some manually authored and and some random generated spatial configurations. In the experiments we use the input *A* described in the previous experiment. Results are shown in Table II.

TABLE I. PLANNING RESULTS FOR DIFFERENT EXPERIMENTS. $|P|$ IS THE NUMBER OF PLOT POINTS, $|L|$ IS THE NUMBER OF LOCATIONS, $|K|$ IS THE NUMBER OF PATHS OR EDGES IN THE SPATIAL CONFIGURATION E, G IS THE NUMBER OF GOALS AND $|\mathbb{S}_E|$ IS THE NUMBER OF STORIES FOUND.

	$\|P\|$	$\|L\|$	$\|K\|$	$\|G\|$	$\|\mathbb{S}_E\|$
A	8	7	6	1	10
A	8	7	0	1	0
A	8	7	21	1	100
B	6	5	4	1	2
B	8	6	6	1	21
C	19	4	3	14	8
C	19	4	3	4	192
C	26	11	55	4	384

TABLE II. NUMBER OF STORIES FOUND ($|\mathbb{S}_E|$) IN A SERIES OF SPATIAL CONFIGURATIONS, AND THEIR EVALUATION USING BOTH A *average* AND A *minimum* AGGREGATION OPERATORS.

	$\|\mathbb{S}_E\|$	Avg.	Min.
A_6	10	0.862	0.771
A_8	64	0.851	0.709
A_3	1	0.746	0.746
K_7	100	0.858	0.709
R_{Avg}	64.2	0.701	0.596
R_4	0	0	0
R_5	3	0.919	0.915
R_{11}	85	0.865	0.737
R_{17}	96	0.871	0.747
R_{19}	99	0.855	0.711
G_7^{Avg}	3	0.92	0.915
G_7^{Min}	3	0.916	0.914

The first four rows show the results for input A, using three authored spatial configurations with 6, 8 and 3 edges respectively and the complete K_7 (with 21 edges). The spatial configurations were manually crafted by ourselves using our subjective aesthetic preferences. We can observe how the number of paths affects positively the number of stories found $|\mathbb{S}_E|$ but has a negative impact on the *average* and a *minimum* evaluation aggregates. We also tested the input A with 89 randomly generated spatial configuration graphs with different number of edges. The fifth row shows the averages for the 89 random graphs followed by 5 samples, R_i, where the subindex i indicates the number of edges in the randomly generated graph. Finally, rows 11 and 12 show results from our ϵ-greedy search after running for 500 iterations. G_7^{Avg} was obtained trying to optimize the *average* evaluation aggregation operator and G_7^{Min} trying to optimize *minimum* instead. The subindex 7 indicates that the spatial configuration found contains seven edges.

As Table II shows, randomly generated maps have a low evaluation, and, in some instances, don't even support any story (e.g. R_4). The two spatial configurations generated by our algorithm, however obtain very high evaluation (always above 0.9), while supporting 3 stories. Comparing the authored spatial configurations against the algorithm results we can observe how the later obtained a significantly higher evaluation. However, the human authored spatial configurations take into account some interesting features, like allowing for a larger variety of stories or a more coherent distribution of the space. Those features are not taken into account by our evaluation function but might be desirable in some situations. As part of our future work, we would like to incorporate some of those features in our evaluation function.

Finally, Figure 6 shows the evolution of the evaluation

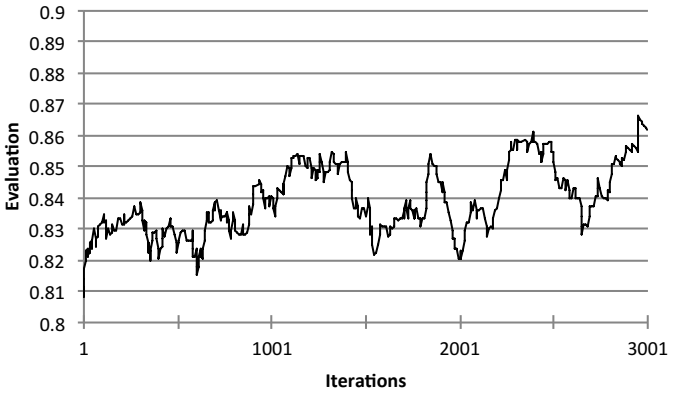

Fig. 6. ϵ-greedy evaluation evolution over 3000 iterations, smoothed with a sliding window of size 300.

scored for the spatial configurations during 3000 iterations. In order to account for the variability and a significant number of spatial configurations with no stories found, we show a smoothed chart with a sliding window of size $W = 300$. The plot shows that, as the search process progresses it yields spatial configurations with higher evaluation functions.

C. Embedding and Graphical Realization

The graph embedding algorithm processes a graph and yields a layout $\mathbb{Y}_E$ to be used by the graph realization process. We tested the algorithm by hand crafting a set of 25 planar spatial configuration graphs to test different features. The graphs range from 6 to 16 nodes and from 9 to 27 edges. Our algorithm yields a satisfying layout for 18 out of the 25 graphs.

We also tested the algorithm against 82 of the randomly generated graphs defined in the previous experiment. The graphs are not guaranteed to be planar and feature between 2 to 15 nodes and between 1 to 34 vertices. Our algorithm yields a satisfying layout for 62 out of the 82 graphs.

Figure 7 gives an example of the final graphical realization of one of the graphs that the algorithm can handle. Note how a new node *X* has been added replacing the K_4 subgraph formed by the nodes a, b, c, d. Also, notice how several locations, such as f, g or h, have been expanded to occupy more than one grid cell.

V. CONCLUSIONS AND FUTURE WORK

In this paper we have presented a novel approach to procedurally generating game maps using storytelling techniques. We have described a system that can generate stories and then design a map supporting those stories. The system evaluates the stories in order to maximize the quality of the output. We use an intermediate graph structure to describe the spatial relationships between locations in the map, which we then use to graphically realize the map. Our experimental evaluation showed promising results, although there is a lot of room for improvement.

As our experiments show, our system can effectively explore the space of spatial configurations, searching for those

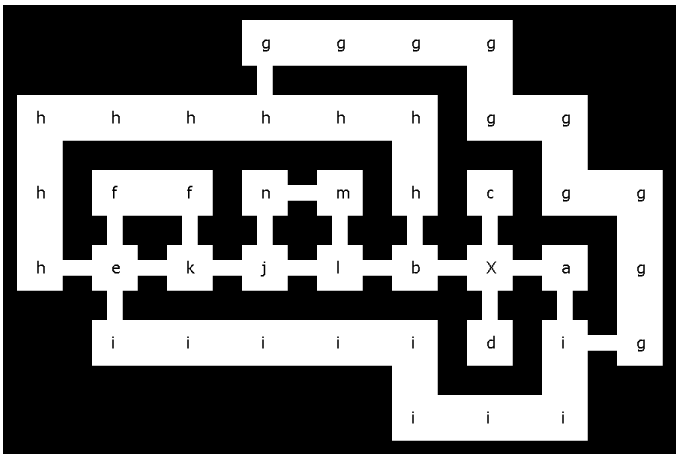

Fig. 7. Example graphically realized graph.

that support high quality stories from the story space. Especially interesting is the capability of our system to generate game maps that support multiple stories or that contain cycles.

As part of our future work, we would like to explore the advantages of using more advanced story planners. The authored spatial configurations exhibit some properties that have not been encoded in the numerical interpretation of our evaluation function and are subjectively of higher quality. We plan on improving our evaluation function to account for additional features. We also would like to incorporate drama-management concepts and use a player model along with annotations in our input game world specification to generate maps tailored to specific player preferences or player archetypes [22]. We would like to tighten the relationship between the planning and the graph embedding processes and incorporate elements from partial order planning into the environment creation process in order create maps more relevant to the story and improve the scalability of the system. Additionally, the graph embedding process has shown to be one of the most challenging parts of the system. In order to reduce the running time to a reasonable bound for our experiments we implemented several heuristics and limited backtracking which may prevent us from finding a planar embedding for graphs that actually are planar. We would like to improve our preprocessing steps for better handling subgraphs homeomorphic to K_5 and $K_{3,3}$. Finally, we would like to connect our map generation system with a game engine in order to have a fully playable game to perform user studies and validate our results. We would also like to look into on the fly map generation incorporating player modeling into our evaluation function and drama management techniques for further improving the player experience.

REFERENCES

[1] J. Togelius, G. Yannakakis, K. Stanley, and C. Browne, "Search-based procedural content generation," *Applications of Evolutionary Computation*, pp. 141–150, 2010.

[2] J. Doran and I. Parberry, "A prototype quest generator based on a structural analysis of quests from four MMORPGs," *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games - PCGames '11*, pp. 1–8, 2011.

[3] R. Smelik, K. Jan de Kraker, S. Groenewegen, T. Tutenel, and R. Bidarra, "A survey of procedural methods for terrain modelling," *Proceedings of the CASA Workshop on 3D Advanced Media In Gaming And Simulation (3AMIGAS)*, 2009.

[4] G. Kelly and H. McCabe, "A survey of procedural techniques for city generation," *Institute of Technology Blanchardstown Journal*, 2006.

[5] M. Riedl and V. Bulitko, "Interactive Narrative: An Intelligent Systems Approach," *AI Magazine*, vol. 34(1), pp. 1–13, 2013.

[6] P. Gervás, "Computational Approaches to Storytelling and Creativity," *AI Magazine*, vol. 30, no. 3, pp. 63–70, 2009.

[7] J. Togelius, E. Kastbjerg, D. Schedl, and G. N. Yannakakis, "What is procedural content generation?: Mario on the borderline," in *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games*. ACM, 2011, p. 3.

[8] M. Riedl, D. Thue, and V. Bulitko, "Game AI as storytelling," *Artificial Intelligence for Computer Games*, no. 125, 2011.

[9] D. Thue and V. Bulitko, "Procedural game adaptation: Framing experience management as changing an mdp," in *Eighth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2012.

[10] E. Tomai, "Towards Adaptive Quest Narrative in Shared, Persistent Virtual Worlds," *Eighth Artificial Intelligence and Interactive Digital Entertainment (AIIDE 2012)*, pp. 51–56, 2012.

[11] K. Hartsook, A. Zook, S. Das, and M. O. Riedl, "Toward supporting stories with procedurally generated game worlds," *2011 IEEE Conference on Computational Intelligence and Games (CIG'11)*, pp. 297–304, Aug. 2011.

[12] J. Dormans and S. Bakkes, "Generating Missions and Spaces for Adaptable Play Experiences," *Computational Intelligence and AI in Games, IEEE Transactions*, vol. 3, no. 3, pp. 216–228, 2011.

[13] C. Ashmore and M. Nitsche, "The quest in a generated world," *Proc. 2007 Digital Games Research Assoc. (DiGRA) Conference: Situated Play*, pp. 503–509, 2007.

[14] M. Nitsche, "Designing Procedural Game Spaces: A Case Study," *Proceedings of FuturePlay*, 2006.

[15] K. Hullett and M. Mateas, "Scenario generation for emergency rescue training games," in *Proceedings of the 4th International Conference on Foundations of Digital Games*. ACM, 2009, pp. 99–106.

[16] R. Lopes, T. Tutenel, R. M. Smelik, K. J. de Kraker, and R. Bidarra, "A constrained growth method for procedural floor plan generation," in *Proc. 11th Int. Conf. Intell. Games Simul*, 2010, pp. 13–20.

[17] P. Weyhrauch, "Guiding interactive drama," Ph.D. dissertation, 1997.

[18] J. Meehan, "Tale-spin," *Inside computer understanding: Five programs plus miniatures*, pp. 197–226, 1981.

[19] O. Goldschmidt and A. Takvorian, "An efficient graph planarization two-phase heuristic," *Networks*, vol. 24, no. 2, pp. 69–73, 1994.

[20] F. K. Hwang, D. S. Richards, and P. Winter, *The Steiner tree problem*. North Holland, 1992, vol. 53.

[21] R. Tamassia, "On Embedding a Graph in the Grid with the Minimum Number of Bends," *SIAM Journal on Computing*, vol. 16, no. 3, pp. 421–445, 1987.

[22] A. Ramirez and V. Bulitko, "Telling Interactive Player-Specific Stories and Planning for It: ASD+ PaSSAGE= PAST," *Eighth Artificial Intelligence and Interactive Digital Entertainment (AIIDE 2012)*, pp. 173–178, 2012.