Visualizing Narratives
Kalpesh Padia

Abstract—A narrative is an ordered sequence of connected events, usually involving multiple participants. Most existing visualization techniques represent narratives as a node-link graph where a sequence of links shows the evolution of causal and temporal relationships between characters in the narrative. These techniques make a number of simplifying assumptions about the narrative structure, however. They assume that all narratives progress linearly in time, with a well-defined beginning, middle, and end. They assume that all events occur in the narrative’s timeline, and that at least two participants interact at every event. Finally, they assume that every narrative is complete at the time it is visualized. Thus, while existing techniques are suitable for visualizing linear narratives, they are not well suited for visualizing evolving narratives, nor for narratives with multiple timelines. In this paper, we present an extension of StoryFlow, an optimization strategy for fast generation of narrative visualizations. Our technique can visualize narratives where only one participant exists within an event. We can filter events from the narrative using constraints based on characters and/or location. We present our approach using a novel visualization technique, Yarn, that represents narratives with multiple timelines. We propose ways to extend Yarn to create visualizations for both completed and evolving narratives, and to encode character properties like beliefs, sentiments, and goals to create a more informative visualization that describes a richer set of user experiences.

Index Terms—Narrative visualization, extension, diegetic narratives

1 INTRODUCTION

A story or a narrative is an ordered sequence of connected events in which one or more characters (or entities) participate [1]. The events in the narrative take place at various locations, and together with the entities, define the relationships that shape the course of the narrative. Understanding the evolution of these entity relationships is key to comprehending and analyzing how the narrative unfolds. To this end, storyline visualizations have been developed to represent a narrative based on the causal and temporal patterns of the entity relationships.

Existing storyline visualization techniques, whether automated [2, 3, 4] or hand-drawn [5], represent narratives as node-link graphs. As shown in Fig. 1, events in a narrative are laid out chronologically from left to right, with each entity represented as a line running from one event to another. Events are shown as nodes in the storyline, and a link between a pair of nodes represents an entity that participates chronologically in both events.

Initial visualization techniques produced an aesthetically pleasing visualization, but at the expense of time. Liu et al. [1] described an optimization strategy, called StoryFlow, for fast generation of storyline visualizations. StoryFlow creates a visualization using a four-stage pipeline that generates an initial layout, then performs ordering and alignment of nodes, and compaction of the overall layout to improve its appearance.

While these techniques can produce aesthetically pleasing and legible storylines, they do so by making simplifying assumptions about the structure of the narrative. Because of this, they may not support more complex, real-world storytelling and analysis tasks. First, existing techniques assume that at least two entities participate at every event in the timeline. However, many real-world narratives such as video game gameplay involve situations where the entities generate events without interacting with other players, or by interacting with players that are not present at the same location. This creates single-entity events in the narrative that are not supported by existing techniques. Second, many narratives involve character entities making choices. These choices directly influence the evolution of the causal relationships that shape the outcome of the narrative. For many real-world analysis tasks, it is important to not only visualize the narrative as it unfolds (reality timeline) but to also include alternative choices that can lead to alternate outcomes (diegetic timelines). Existing techniques visualize only reality timelines. They provide no support for diegetic narratives. Third, existing techniques assume that the narrative progresses linearly in time and has a well-defined beginning, middle, and end. While this makes them suitable for visualizing linear narratives such as a movie’s plot, they cannot visualize evolving or non-linear narratives.

In order to allow for single-entity events, we have extended the original StoryFlow algorithm to support single or multiple participant events at every stage in the pipeline. We have also implemented filtering of events based on entity and location
constraints. Our approach extends StoryFlow’s layout pipeline by introducing a new stage prior to the final stage in the pipeline for merging single-entity events with normal events, while satisfying any user-supplied constraints. It also reorders the entities and realigns the events to create a merged layout that is then supplied to the last stage in the StoryFlow pipeline, producing an aesthetically pleasing storyline visualization.

In addition to extending StoryFlow, we present a new storyline visualization technique, Yarn, that utilizes a hierarchical task network (HTN) representation of the narrative to create a storyline visualization to depict both the reality timeline and the possible diegetic timelines in a narrative. The HTN representation encodes information about all possible choices and their outcomes, identifying one of the outcomes as the reality timeline. This allows Yarn to compute layouts for each diegetic timeline in the narrative, which are merged to produce a final visualization.

Specifically, our work makes the following contributions.

1. An extension to the layout pipeline of StoryFlow to include single-entity events.
2. A modification to StoryFlow to filter narrative events based on location and entity constraints.

2 BACKGROUND

2.1 Narrative

Narrative theorists draw ideas from various fields, including literary theory, linguistics, cognitive science, folklore, and gender theory to define what constitutes a narrative and how it is different from other kinds of discourse, such as lyric poems, arguments, and descriptions [6, 7, 8, 9]. Researchers have studied narratives using numerous approaches such as rhetoric, pragmatic, and antemimetic [8, 9] to give multiple definitions of narrative. All definitions, however, agree that narrative is a way to organize spatial and temporal data into a cause-effect chain of events with a beginning, middle and end.

In recent years narrative theory has generated significant interest among computer scientists, especially in the field of artificial intelligence (AI), computational linguistics, and game design. Researchers have proposed numerous systems for automatic generation of narratives, such as Tale-Spin [10], Minskrel [11], Mexico [12], Virtual Storyteller [13], Fabulist [14], and Suspenser [15]. These systems identify a thematic pattern in a pre-existing corpus of fabula (elements of the story world and all causally related actions occurring within it) to create the narrative’s sjužhet (the story’s telling by selecting and ordering fabula events to create a linear progression of the story).

Significant research efforts have also been directed towards developing narrative engines which can automatically generate sjužhet (and thus, the narrative) based on the end goals specified in the fabula. These systems represent the fabula as a collection of state spaces that are searched to find a sequence that satisfies the end goal. Two approaches are commonly used. In the first, a STRIPS-like [16] formalism is adopted to represent the fabula as a collection of world models for all possible scenarios in the narrative. A world model is defined as a set of well-formed formulas using first-order predicate calculus. A well-formed formula is also used to state the goal condition. A set of operators enumerate possible actions and their effect on the world models. Various AI planning techniques are used to apply these operators to the world model collection to find a model that achieves the stated goal condition [17, 18, 19]. The sequence of events described in the solution model generates a narrative in which the goal condition is achieved.

In the second approach, the fabula is represented using a hierarchical task network (HTN) formalism. A hierarchical task network is a network representing ordered task decomposition. The root node in an HTN depicts the goal task. Each child node can be either a primitive action, or a sub-task that can be further decomposed into additional sub-tasks or primitive actions. Fig. 2 shows an HTN for the task “Build House”. Rectangular nodes represent primitive actions, and circular nodes represent sub-tasks in the decomposition. The final decomposition for the goal task contains all primitive actions, returned from a depth-first search of the HTN. A narrative’s fabula can be represented a collection of HTNs since narrative descriptions can be naturally represented as task decompositions [20, 21]. HTN planning is then employed to find a decomposition that satisfies a given goal condition [20, 22, 23, 24]. The sequence of primitive actions in the decomposition describe a narrative in which the goal condition is achieved.

2.2 Temporal Visualizations

The temporal axis is a key component for a variety of datasets. Considerable research has been directed towards developing visualization techniques to examine temporal changes in data with the help of visual metaphors. ThemeRiver [25] uses a stacked area chart to represent a river of themes or events whose flow can ebb or increase over time to indicate a change in frequency or importance (Fig. 3). EventRiver [26] visualizes temporally related topic clusters in a document corpus as a stream of event bubbles where each bubble represents a cluster of documents that belong to the same topic and are adjacent in time. Users can browse the bubbles for significant events, and track event evolution by examining a bubble’s stream. NameVoyager [27] visualizes historical trends in popularity of baby names.
as an area chart, allowing users to compare the popularity of names over time.

While these techniques facilitate easier understanding of temporal changes in data, they do not visualize entity relationships, which are important for many analytic tasks. Thus, a number of additional techniques have been presented to visualize relationships along with the progression of event sequences, often with the help of directed edges. SemTime [28] uses distinct types of directed edges and time independent stacking of multiple timelines to show relationships between events. tmViewer [29] connects related entities along the timeline using simple directed edges. Continuum [30] enables hierarchical relationships in temporal data to be represented and explored by dynamically reordering the event space.

Although the above techniques are sufficient for visualizing a small number of entity-event relationships, they are not well suited for visualizing large numbers of relationships, such as narratives, due to the amount of visual clutter created by line crossings. Thus, alternate techniques are required to visualize narratives.

### 2.3 Narrative Visualizations

Recently, a new visualization technique called storyline has emerged as a way to visualize narratives. In this technique, inspired by Munroe’s movie narrative charts [5], events in the narrative are laid out chronologically from left to right, and each entity is shown as a line running from one event to another. Events are represented as nodes in the storyline. A link between a pair of nodes represents an entity that participates chronologically in both events.

When solving a real-world problem, storyline visualizations are generally simplified by imposing application-specific constraints, thereby achieving success at the cost of loss of generality [1]. More recent research efforts have focused on developing a generic storyline visualization tool. Tanahashi and Ma [2] presented a storyline layout technique based on a genetic algorithm. While their approach creates an aesthetically appealing and legible storyline visualization, it takes considerable time to create the layout. Liu et al. [1] presented a strategy called StoryFlow that formulates the difficult problem of creating an effective storyline layout as an optimization problem, creating an aesthetically appealing visualization significantly more quickly than Tanahashi and Ma.

These techniques, however, make simplifying assumptions about the structure of the narrative. This means they may not support more complex, real-world storytelling and analysis tasks. The techniques assume that at least two entities participate at every event in the timeline, so they are not suited for visualizing real world narratives that contain single-entity events. Further, these techniques provide no support for diegetic narratives, nor can they visualize evolving narratives.

Our approach extends the original StoryFlow algorithm to visualize narratives with single-entity events. We also implement filtering of events based on entity and location constraints. In addition, we present Yarn, a new storyline visualization technique to visualize diegetic narratives using their corresponding HTN representations.

### 3 EXTENDED STORYFLOW

#### 3.1 Overview

Given a session table and a location tree as inputs, StoryFlow uses a four stage layout pipeline to create a storyline visualization (Fig. 4). Interaction between multiple entities during two adjacent time frames is termed a “session” and constitutes an event in the narrative. The session table serves as a lookup table to determine which session an entity belongs to during a given time frame. The location tree lists all event locations in the narrative in the form of a hierarchical tree.

Initially, the hierarchy generation stage of the layout pipeline uses the session table and the location tree to generate a hierarchical relationship tree for every time frame in the narrative. Each node in the tree represents a location and includes all sessions occurring at that location. The set of all relationship trees represents all events and their locations in the narrative. This stage is followed by three layout optimization stages: ordering, alignment, and compaction.

In the ordering stage, the entity lines are sorted vertically to minimize the number of line crossings. Producing such an ordering is an NP hard problem [31] and is divided into two smaller problems: sorting the location nodes, and ordering the sessions and entities within each location node. First, a greedy algorithm is used to recursively sort the location nodes from top to bottom, at every time frame. The sessions and entities under each location node are then ordered using a modified directed acyclic graph sweeping algorithm [32, 33] to produce a layout that minimizes the number of line crossings.

In the alignment stage the entity lines between successive
time frames are aligned in order to minimize the number of wiggles and maximize the number of straight lines in the layout. Finally, the compaction stage applies a quadratic optimization algorithm \cite{1,34} to minimize the white space between various entity lines and create a final layout for the storyline visualization.

StoryFlow defines a session as the interaction between two or more entities during a given time frame. Many real world narratives, however, involve situations where entities generate events without interacting with other entities present at the same location. These single-entity events play an important role in shaping the overall narrative, and reveal interesting character and plot properties. For example, when a character in a movie narrative uncovers critical information, the event contributes to the plot’s advancement by shaping the narrative arc in a desired form. Similarly, the frequency and nature of character death and respawn events in a narrative representing video game gameplay define its type (role playing, first person shooter, etc.), playable difficulty, and the overall skill of the player. Visualizing single-entity events highlights interesting features of a narrative.

StoryFlow and other storyline visualization techniques were primarily developed to visualize movie narratives, which are largely linear and have a few tens of events. However, real world narratives are often constituted from a much larger number of events. For example, a narrative representing how the attention of political analysts (entities) shifts from one topic to another during an election season may be built from a few thousand hashtags (events). This is also true for narratives representing literary works that are of epic proportions such as the Bible or the Mahabharata—each epic is a narration of a large number of events depicting interactions between a number of characters. Storyline visualizations for large narratives tend to be squeezed into a limited screen space, increasing the number of wiggles and line crossings in the visualization. In such cases it would be useful to filter events based on certain constraints in order to limit the number of visualized events, thereby reducing visual clutter. Additionally, filtering based on constraints can be used to visualize events in a narrative with varying degrees of granularity.

In this work, we extend StoryFlow to support single-entity events in the narrative by introducing a new penultimate stage, called merge events, in the layout pipeline (Fig. 5). In this stage, we first create sessions for single-entity events while considering any user-supplied constraints. These sessions are then merged with the layout from the alignment stage to generate the input layout for the compaction stage in the pipeline. We also modify the ordering stage to support filtering of multi-entity events based on the same user-supplied constraints.

Defense of the Ancients 2 (DOTA 2) is an online cooperative multiplayer video game that requires each player to control a single character in one of two teams. The objective of the game is to destroy the main structure of the opposing team. Starting with a fixed objective, each team forms its own strategy, and engages in conflicts with the other team until one team emerges victorious. Each gameplay forms a unique narrative of how players co-operate with each other as a team to achieve a common goal. We use the gameplay for a single session of DOTA 2 as an example narrative to illustrate extended StoryFlow and compare its layout with the original StoryFlow visualization.

3.2 Single-Entity Events

A narrative arc is generally made up of five stages: Exposition: a series of events that complicate matters for the protagonist and raise the narrative’s suspense or tension; Climax: the point of greatest tension in the narrative serving as a turning point in the narrative arc; Falling Action: a series of events that release the tension; and Resolution: a narrative’s conclusion \cite{35}. While most narratives use multi-entity events to establish the various stages of the narrative arc, certain narrative genres such as suspense often employ single-entity events to establish climax in the form of the protagonist discovering crucial information that leads towards falling action and resolution of the plot. We term events that are responsible for shaping the narrative arc plot events.

Narratives also contain other single-entity events that are helpful in defining traits of various characters. We term such events character events. Although character events most often appear during exposition, they may also appear during other stages of the narrative arc. Simple character events manifest themselves in the form of birth and death events, while complex character events such as stealing or committing suicide may also be present in the narrative. Apart from contributing towards the shape of the narrative arc, these events highlight a character’s traits, and may be used to justify or reason about other events in which the character participates.

Since StoryFlow does not support single-entity events, it ignores all plot and character events in the narrative. This leads to two issues. First, ignoring plot events creates a visualization that is an incomplete depiction of the narrative. If the ignored plot events occur in the middle of the narrative, the narrative perceived from the visualization may still be a good approximation of the actual narrative. However, missing plot events at the beginning or the end alter the perceived narrative significantly. Second, certain character events such as death and rebirth require entities to be represented using discontinuous character lines in the visualization. Ignoring these events creates a visualization that represents all entities using continuous character lines, producing an inaccurate depiction of the narrative.

To support single-entity events in a narrative we extend StoryFlow’s layout pipeline by introducing a new penultimate stage called Merge Events (Fig. 5). This stage is responsible for merging single-entity events with normal events to generate a merged layout which is then sent to the compaction stage as before.

We begin by creating a new session table containing entries for all single-entity events in the narrative. We then filter the ta-
ble using user-supplied constraints (discussed in next section). This prunes the session table to contain only those events which satisfy the supplied constraints. The filtered events are categorized as fatal and non-fatal depending on whether the event results in the death of the entity.

Next, we merge all non-fatal events with those already present in the storyline layout generated from the alignment stage of the pipeline. For each non-fatal event we identify the entity name and time frame for that event. We then retrieve the entity’s region table, which stores all conflict regions for the entity sorted by time frames, and add the time frame for the event to this table.

For each fatal event we identify the name of the entity and truncate his corresponding line at the time frame specified in the event. We then scan the entity’s region table to find the time frame of the first event that occurs after this fatal event. If the entity dies permanently, the region table will not have any matching entries and we do not need to restart the entity line. On the other hand, if the entity is respawned in the narrative, the corresponding event would have been classified as a non-fatal event and would have already been added at the appropriate position in the region table. We can therefore retrieve the time frame of the respawn event, and connect it with the next event in the region table, restarting the entity line. The merged layout is sent to the last stage of the pipeline to create a final compact layout of the storyline visualization.

Fig. 6: Comparison of the visualization of a DOTA 2 gameplay session using: (a) StoryFlow, (b) Extended StoryFlow.
In our example narrative, players from two teams—Dire and Radiant—compete with each other to destroy the opponent’s main structure. During the game, players find various items and level up, creating single-entity events. Players of opposing teams come together at a location for conflict or battle, creating multi-entity events. If a player dies in the battle, he automatically respawns after a few time frames. Occasionally players from the same team may choose to collaborate together on a common goal.

Fig. 6(a) shows the storyline visualization for our example narrative, created using StoryFlow. Each colored line represents a player from one of the two competing teams, Dire and Radiant. The legend at the bottom right corner groups all players according to their teams, and lists their names and colors. StoryFlow ignores all single-entity events in the narrative, including all death and respawn events. This produces a narrative that appears straightforward with a handful of conflicts between opposing teams, none of which resulted in any casualties.

Fig. 6(b) shows the storyline visualization for our example narrative, created using Extended StoryFlow. No user-defined constraints have been applied for this visualization. The visualization depicts both single-entity and multi-entity events in the narrative. The single-entity events are helpful for analyzing how quickly each player leveled up, and may provide important insights into their strategies. Additionally, the inclusion of death and respawn events now helps to highlight conflicts between the two teams, identify nemeses, and analyze which of the two teams is stronger.

### 3.3 Constraint-Based Filtering

Real world narratives often contain a large number of events. As screen space is limited, these events tend to squeeze together when the narrative is visualized. This increases the number of wiggles and line crossings in the visualization, which reduces its comprehensibility. If the size of the visualization is made independent of the dimensions of the screen, we can extend it beyond the available screen space. Narrative events can then be rendered without crowding. However, we would require the use of pan and zoom controls to view the entire visualization.

Another technique is to visualize only those events that lie within a chosen range of time frames in the narrative timeline. However, a new range must be selected to view events that occur within a different range of frames. Although these techniques utilize the available screen space in a manner that avoids overcrowding of events, they show only a portion of the narrative at a time. This causes a temporal separation between the events that are currently visualized, and the events that occur before and after the selected range of time frames.

We propose a new technique that uses constraint-based event filtering to visualize large narratives without any temporal separation between events. In our technique we first inspect the narrative’s fabula to identify narrative elements, such as event location, props, and number of participants in an event, that can serve as criteria to group events in the narrative. By applying constraints on the possible values of these criteria, we can filter the list of events to include only those that satisfy the supplied constraints. This allows us to visualize events that occur under certain narrative conditions while ignoring the rest.

In this work, we modify the hierarchy generation stage of StoryFlow’s layout pipeline to implement constraint-based filtering of narrative events. We first check if each event in the session table fulfills the chosen criteria and the supplied constraints. All events that do not fulfill the constraints, except birth, death and respawn events, are removed from the session table. We then use the pruned table to generate a set of event-location relationship trees which are passed to the subsequent stages of the pipeline to create a visualization.

Our technique depicts only those events that fulfill the supplied constraints. This reduces visual clutter, and allows us to visualize the entire narrative within the available screen space more clearly than previous techniques. Since a storyline visualization represents the sjužhet form of the narrative this technique also allows us to generate alternate sjužhetes, each representing a “what-if” scenario modeled using different criteria and their corresponding constraints. Multiple scenarios can thus be visualized and compared with each other to perform an exploratory analysis of the narrative.

In a co-operative multiplayer game, such as our example narrative, understanding the gameplay of each team member and identifying strong team players is crucial for creating an effective strategy. A player who appears often with other team members in various events is a better team player than one who seeks opponents alone. Similarly, a player who actively seeks conflicts with players of the other team and survives them is an asset while playing an offensive rather than a defensive game. To identify effective team players in such a narrative we can use the number of participants in an event as a criterion. We can then perform an exploratory analysis of the narrative by applying constraints on the number of participants from each team and the total number of participants in the event.

For our example narrative, we represent the numeric constraints on the number of participants in an event as the tuple \(<\text{min}_{\text{Overall}}, \text{min}_{\text{TeamDire}}, \text{min}_{\text{TeamRadiant}}>. \text{min}_{\text{TeamDire}}\) indicates the minimum number of team Dire participants required for the event to fulfill the constraint. \text{min}_{\text{TeamRadiant}} indicates the minimum number of team Radiant participants required for the event to fulfill the constraint. \text{min}_{\text{Overall}} indicates the minimum number of total participants required for the event to fulfill the constraint. The participants may be from different teams, or may all be from the same team to satisfy this constraint. These constraints are used together to filter the events using a set intersection operation. If, for example, the constraints supplied are \(<2, 1, 1>\), then the event must have at least two participants, one from each team. Similarly, if the constraints supplied are \(<3, 1, 2>\), then the event must have at least three participants, with at least one participant from team Dire, and at least two participants from team Radiant. If no constraints are supplied, or if the supplied constraints are one of \(<0, 0, 0>\) and \(<1, 0, 0>\), then all events are represented in the visualization.

Fig. 7 shows the storyline visualization for our example narrative with the constraints \(<2, 0, 0>\). With these constraints, at least two players must participate in an event for it appear in the visualization. The players may either belong to the same team, or to separate teams. All single entity events, with the exception of birth, death and respawn events, are absent from the
Fig. 7: A session of DOTA 2 gameplay visualized using Extended StoryFlow with constraints based filtering. Constraints $<2, 0, 0>$ are applied to the chosen criterion of “number of event participants” with constraints $<\min_{\text{Overall}}, \min_{\text{TeamDire}}, \min_{\text{TeamRadiant}}>$. At least two participants must participate in an event for it appear in the visualization.

Fig. 8: Constraints $<3, 0, 0>$ applied to a session of DOTA2 gameplay. The visualization is made up of straight lines since the narrative does not have any event with three participants. Instead, the visualization focuses on events that have at least two participants. This allows us to clearly identify co-operative events (where all participants belong to the same team), and conflict events (where at least one participant belongs to an opposing team). We observe that the player “Wisp” from team Radiant does not participate in any events, while the most active player is “Crystal Maiden” from team Dire, participating in three conflicts and three co-operative events. We also observe that the players from team Radiant rarely interact with each other, while team Dire shows a greater degree of co-operation among its players.

Fig. 8 shows the storyline visualization for our example narrative with the constraints $<3, 0, 0>$. These constraints filter out all events that have less than three participants, with the exception of birth, death and respawn events. As there are no events in the narrative with three or more participants, the resulting visualization contains only character lines running from left to right for the duration of the visualization. Broken lines indicate birth, death and respawn event. However, due to the absence of multi-entity events, we do not see any line crossings in the visualization.

Fig. 9 shows the storyline visualization for our example narrative with the constraints $<2, 0, 1>$. The visualization depicts only those events that have at least two participants with at least one of them from team Radiant. Since our narrative does not have any event with three participants, these constraints require that at most one player from team Dire participates in an event. This filters out all co-operative events among team Dire players.
and instead depicts only the conflict events for team Radiant. Once again we observe the lack of co-operation within team Radiant.

3.4 Summary

We have presented an extension to StoryFlow that allows us to generate a storyline layout for narratives with both single-entity and multi-entity events. The visualizations created using extended StoryFlow are more accurate descriptions of the narrative due to the inclusion of single-entity events that depict various character and plot properties. We have also introduced a new technique for visualizing narratives with larger numbers of entities. Our technique prunes the session table by filtering narrative events using user supplied constraints. The resulting visualization includes only those events that satisfy the constraints, and contains fewer wiggles and line crossings. Different criteria and their corresponding constraints can be used to define different scenarios for the narrative. When visualized, these scenarios can be compared with each other to identify character and plot properties, and to perform an exploratory analysis of the narrative.

4 Yarn

4.1 Overview

As discussed earlier, significant research has been directed towards developing visualization techniques for narratives. However these techniques apply several simplifying assumptions to the narrative, which makes them suited only for linear, well-defined and complete narratives. Many real-world narratives, on the other hand, are incomplete, and often involve situations where the character entities must make choices. Each choice unfolds the narrative along its reality timeline, while also creating diegetic timelines involving the alternate choices. Visualizing how a narrative progresses over time can reveal fascinating details about its evolution, and aid in understanding how it might have changed had it followed an alternative path.

In this work, we present Yarn, a new technique for visualizing narratives with multiple alternate timelines. Our system represents the narrative as a hierarchical task network (HTN), and utilizes HTN based planning to generate a timeline of events in the narrative. Given an HTN formulation of the narrative, Yarn generates an interactive storyline visualization that shows its current progression. For complete narratives, Yarn creates a visualization that depicts events in the reality timeline of the narrative. For incomplete narratives, the visualization depicts events in the currently evolving timeline. Additionally, if a narrative has more than one timeline, the alternate timelines can be revealed and visualized.

4.2 Hierarchical Task Networks

Hierarchical Task Networks (HTNs) are networks that represent ordered task decomposition. HTNs are based on the idea that many tasks in real life have a built-in hierarchical structure. The top-level task in an HTN is typically also the main goal. Each task can be decomposed into sub-tasks, which can be further decomposed into smaller tasks until all tasks are represented as primitive actions. HTNs are commonly built using AND/OR graphs. When a task can have several possible decompositions, it is represented using an OR node. Each sub-task is a perfectly valid decomposition for the parent task. When a task has several decompositions that can be ordered in some fashion to complete the task, it is represented as an AND node. Thus, an HTN can be seen as an implicit representation for the set of possible solutions for a task [36].

Narrative descriptions can be naturally represented as task decompositions, and hence are well suited for representation using HTN planning [20, 21]. Further, unlike STRIPS-like planning, HTN planning allows compound goals that are structured as a collection of multiple goals or primitive tasks. Fig. 10 shows an HTN for one of the characters in our example narrative. Goals are represented using circles and actions are rep-
represented using rectangles. AND and OR nodes are connected to their decomposition using blue and red links, respectively. The main goal, represented by the root node, is achieved by decomposing the sub-goals and performing actions in order.

While traditional HTN planning first generates a decomposition for the current goal and then executes it, interactive storytelling requires interleaving of plan and execution [37]. Cavazza et al. [20] have described a technique that assumes total order in the HTN graph and sub-task independence to produce a suitable plan. Their technique searches the HTN left to right depth-first, and executes any primitive action it encounters in the process. If any primitive actions fail, the algorithm backtracks to find a suitable alternative. In our work, we use this technique to generate a timeline of events by finding a decomposition for the various character goal states to generate a timeline of events.

![HTN for a character in our example narrative. Blue links connect AND nodes and their children, and red links connect OR nodes and their children.](image)

Fig. 10: HTN for a character in our example narrative. Blue links connect AND nodes and their children, and red links connect OR nodes and their children.

### 4.3 Implementation

Yarn is a web based narrative visualizer that visualizes both complete and evolving narratives. It analyses the narrative structure of the input to identify different timelines embedded in the narrative, and presents them as a set of node-link graphs.

The input to Yarn is a set of JSON files, one for each character in the narrative. A character’s JSON file stores the HTN representation of his goals, and all the events that he has participated in. We classify each node in the HTN as either a parent node or a leaf node, and represent it using a JavaScript object. Any node in the HTN that is not a leaf node is considered a parent node in our technique, and has the following key-value pairs:

- **title**: A descriptive title for this node/state in the HTN.
- **type**: A value “and” or “or” to signify whether the node is an AND node or an OR node.
- **children**: A JSON array of child nodes. A child node may be a parent node or a leaf node.

Leaf nodes signify events in the narrative, and have the following key-value pairs:

- **id**: A unique id for this leaf node (event).
- **title**: A descriptive title for this node/state in the HTN.
- **location**: A named location where this event takes place.
- **completed**: An optional key with value “yes” to indicate that the event has taken place.
- **pre_req**: An optional key whose value is an array of events that must take place before this event can occur.

By analyzing the HTN for every character in the narrative, Yarn can generate a plan that represents the sequence of events in the narrative. First, the JSON files are parsed to create an in-memory HTN for each character. Next, we create a set of all event nodes in the narrative. We call this the NodeSet, containing nodes for all events, both complete and incomplete. We apply the HTN planning algorithm described by Cavazza et al. [20, 22] to obtain a decomposition of the goal state for each character, represented as an ordered sequence of events. Interleaving each character’s event sequence creates the timeline of events in the narrative. We call this the reality timeline. By building a set of links between the nodes in the timeline, we create a LinkSet and mark each link as “taken”. We next create links for alternate paths that do not exist in the reality timeline. We mark each link as “alternate” and add it to the LinkSet. The completed LinkSet represents a node-link graph that connects the various events in the narrative.

To visualize the LinkSet, we must first create a layout that positions the event nodes in a correct temporal order while minimizing line crossings between nodes. This layout problem is similar to the ordering stage of StoryFlow’s layout pipeline. We use the same ordering algorithm to create our final layout. The layout is then rendered using JavaScript, SVG, and D3.

### 4.4 Evaluation

We use a narrative adapted from the video game Witcher 2 to evaluate Yarn and compare it to StoryFlow and Extend StoryFlow.

“Triss Merigold and Cedric are investigating Sile de Tanserville for her involvement with the Lodge of Sorceresses. The investigation leads them to her room in an inn in Flotsam. A short while later, both Triss and Cedric disappear. When Geralt of Rivia hears about their mysterious disappearance, he decides to investigate and rescue his friends. He first goes to their last known location, the inn in Flotsam. At the inn he meets Margot who tells him that she’s been secretly spying on Sile, and might know where Triss and Cedric are (First Choice Point). Depending on how Geralt responds to Margot’s confession about spying on Sile, she may or may not provide him with information about their whereabouts. Without Margot’s help Geralt can’t proceed and the narrative ends. On the other hand, if she chooses to tell him that Cedric vanished into the forest, Geralt travels to the nearby forest where he meets a dying Cedric. Before dying Cedric tells Geralt that he must look for the Trolls who know where Triss is. Geralt continues on his journey into the forest where he first meets the He-Troll. The troll asks Geralt for his help in apologizing to his wife (Second Choice Point). If Geralt decides to help him, he meets the She-Troll deeper in the forest. Otherwise, he needs to challenge both

1[^http://www.d3js.org/]:

[^http://www.d3js.org/]:

Troll deeper in the forest. Otherwise, he needs to challenge both...
trolls to a duel, and the narrative ends. Once Geralt meets the She-Troll he gives her husband’s apology to her, and asks her to forgive him (Third Choice Point). If she accepts the apology, the She-Troll gives Geralt a horn as a gesture of goodwill, and tells him where to find Triss. Geralt continues his journey into the forest to find Triss in an unconscious state, and the narrative ends. On the other hand, if the She-Troll doesn’t accept the apology, the couple duel with Geralt, and the narrative ends.

Our example narrative involves one favorable outcome, and two unfavorable outcomes from Geralt’s perspective. In the most favorable outcome Margot chooses to co-operate with Geralt, Geralt decides to help He-Troll, and She-Troll accepts his apology. Fig. 11 shows the visualization for this narrative created using Yarn. Circular nodes represent events in the narrative, and are laid out chronologically from left to right. Each colored line represents a character’s movement from one event to another. Solid lines depict a character’s movement along the reality timeline, while dashed lines depict a character’s movement along a diegetic timeline. By default, Yarn shows only the reality timeline. The diegetic timelines can be revealed by clicking on the solid nodes which depict events with multiple outcomes.

Fig. 12 shows the visualizations of the favorable outcome for our example narrative, generated by StoryFlow (a), Extended StoryFlow (b), and Yarn (c). Both StoryFlow (SF) and Extended StoryFlow (eSF) show the events in the reality timeline of the narrative. Additionally, eSF also depicts the single-entity events in the narrative. As noted earlier, Yarn hides the diegetic timelines by default, creating a visualization similar to SF and eSF. The alternate outcomes can be revealed by clicking on the nodes with a solid fill (Fig. 11). This allows a user to explore the visualization on demand without overwhelming him with all possible outcomes.

Fig. 13 shows the visualization for an evolving narrative generated by the three techniques. In this example we consider that Geralt has just finished talking to Cedric and has not yet met the trolls. While both SF and eSF show only the completed portion of the narrative, Yarn visualizes the entire narrative. It recognizes that the end-goal for one or more characters isn’t met, and the narrative is still evolving. It visualizes the completed portions of the narrative with a solid line, and reveals all possible future outcomes with a dashed lines. While it hides the alternate outcomes for past events, leaving them to be uncovered manually, it expects every outcome for the future events to be equally likely, thus revealing all possible future outcomes.

In order to visualize evolving narratives and narratives with multiple timelines, Yarn makes the following simplifications. First, it assumes that all outcomes are equally likely. Therefore, in the case of evolving narratives it can not predict which of the multiple timelines is more likely to convert into the reality timeline. Second, when a user clicks on a solid node, it reveals alternate outcomes for that node alone. As HTN planning is employed only during the link set generation phase, Yarn does not automatically expand all alternate outcomes that may be connected to each other. A user must therefore perform exploratory visual analysis to identify a possible diegetic timeline that connects such alternate outcomes.

4.5 Summary

We have presented a new visualization technique, Yarn, that allows us to generate a storyline layout for both complete and evolving narratives. Given an HTN representation for a narrative’s fabula, Yarn executes planning to identify and visualize a sequence of events that lead to the goal state. For complete narratives only the reality timeline is initially visible. Alternate outcomes can be later revealed by interacting with the visualization. On the other hand, for evolving narratives the events on the reality timeline are shown along with all future outcomes. This gives the user an opportunity to analyze the different ways in which the narrative could evolve.
(a) Visualization of our example narrative using StoryFlow. Note the absence of single-entity events.

(b) Visualization of a completed narrative using Extended StoryFlow, showing both single-entity and multi-entity events.

(c) Visualization of our example narrative using Yarn, depicting the reality timeline of the narrative. Clicking on nodes with a solid fill reveal alternate outcomes.

Fig. 12: Visualization of a completed narrative using StoryFlow, Extended StoryFlow, and Yarn.
(a) Visualization of our example narrative using StoryFlow. Note the absence of all events that have not yet transpired.

(b) Visualization of our example narrative using Extended StoryFlow. Note the absence of all events that have not yet transpired.

(c) Visualization of our example narrative using Yarn. Any events that have not yet transpired belong to the diegetic timeline, and are depicted using a dashed line.

Fig. 13: Visualization of an evolving narrative using StoryFlow, Extended StoryFlow, and Yarn.
5 Future Work
As our future work we propose extending Yarn in various ways to predict and visualize the most favorable outcome in an evolving narrative, and to provide a richer user experience. First, we propose to encode various character properties—beliefs, sentiments, goals—in the visualization. This would help a viewer to better understand the decisions taken by the character at a choice point, and the motivation embedded in following a particular narrative arc. Second, we propose to apply HTN planning in real-time to automatically expand all alternate outcomes that may be connected to each other. This would allow for an easier comparison of individual timelines. Finally, we intend to attach probabilities to the outcomes of various choices (and thus resultant diegetic timelines) to automatically predict and visualize the most favorable outcome for a narrative. This would be very helpful in the visualization and analysis of an evolving narrative by suggesting the most likely outcome and allowing the user to explore and analyze the most probable scenario given current conditions.

6 Acknowledgements
The author would like to thank Dr. Michael Young for the data files containing DOTA2 gameplay, and Dr. Jeffrey Ligon for the UI shell to render the visualization.

References