# Novella 2.0: A Hypertextual Architecture for Interactive Narrative in Games

Daniel Green
Creative Technology
Bournemouth University, UK
dgreen@bournemouth.ac.uk

Charlie Hargood
Creative Technology
Bournemouth University, UK
chargood@bournemouth.ac.uk

Fred Charles
Creative Technology
Bournemouth University, UK
fcharles@bournemouth.ac.uk

## ABSTRACT

The hypertext community has a history of research in Interactive Digital Narrative (IDN), including experimental works [27] and systems to support authoring [6]. Arguably the most prevalent contemporary form of IDN is within the world of computer games where a mixture of large-scale commercial works and smaller indie experimental pieces continue to develop new forms of interactive storytelling. We can explore these pieces through the lens of hypertextual theory and support them with hypertextual architectures, but there are unique challenges within modern game-based storytelling that these frameworks sometimes struggle to capture on a content level, leaving us in some cases with insufficient models and vocabulary. In this paper, we build upon previous work [19] by presenting a discussion on techniques of modeling video game narrative. This is followed by thorough presentation and demonstration of our game-centric theoretical model of interactive narrative, *Novella 2.0*, which builds upon our previous contributions. This model is then positioned within a novel architecture for the authoring, interchange, integration, and simulation of video game narrative. We present alongside the architecture four key innovations towards supporting game narrative. We include support for Discoverable Narrative and other game narrative content alongside structural features in a deference of responsibility to game engines and our own approach to mixing calligraphic and sculptural hypertext structure.

## CCS CONCEPTS

• **Human-centered computing → Hypertext / hypermedia**.

## KEYWORDS

interactive narrative, narrative modeling, video games

**ACM Reference Format:**
Daniel Green, Charlie Hargood, and Fred Charles. 2019. Novella 2.0: A Hypertextual Architecture for Interactive Narrative in Games. In *ACM Hypertext '19: 30th Conference on Hypertext and Social Media, September 17–20, 2019, Hof, Germany*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/1122445.1122456


*Hypertext '19, September 17–20, 2019, Hof, Germany*

ACM ISBN 978-1-4503-9999-9/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

## 1 INTRODUCTION

IDN and hypertext are two fields of research and creativity that share a history of common works (such as Joyce's *afternoon, a story*[1], games like *Myst*[2], and more recently the broad range of work created in Twine[3]), experimental pieces [27], and technological tools and systems [6, 22, 28]. Much contemporary interactive narrative can be seen in the area of games, where the ideas of hypertext can be used as a lens to understand the structures and patterns created therein, and provide systems to enable those stories. Works from the hypertext community have identified that research presenting new original hypertext systems has diminished in recent years [3, 22] when much still may be learned about structured and linked content from new implementations.

It is with this in mind that we presented our initial work towards a new hypertext system for game narrative, *Novella* [19]. Our early work showed how existing models and systems captured aspects of game storytelling, and we proposed a new approach towards supporting this major area of interactive narrative that utilized the ideas of hypertext structure. However, this fell short in providing a representation or vocabulary in some areas of game narrative content. This early work has now been refined beyond an observation and proposition into a more mature model and system architecture which we present here as *Novella 2.0*.

In this work, we present a hypertext systems paper that brings together modern game-based IDN and hypertextural structure in a new systems architecture. We begin with a discussion of existing models of interactive narrative and hypertext fiction, and how we have built on top of these towards a new platform for game narrative. We then present the *Novella 2.0* architecture and describe how it supports game narrative, and demonstrate its capabilities. We conclude by describing its specific innovations and how this has built upon the state-of-the-art.

## 2 BACKGROUND

A range of existing approaches have explored support for interactive narrative models and frameworks - we review here work from both the hypertext and games research communities.

### 2.1 Hypertext Models

This community has its own long history of working with interactive narrative, from classic works of hyperfiction such as Joyce's *afternoon, a story* to research on new mediums on which to deliver them such as the seminal HyperCafe [27]. As part of this, hypertext research has also grappled with conceptual structural models

[1] *afternoon, a story* by Michael Joyce, published by Eastgate, 1990
[2] *Myst*, Cyan Worlds Inc., 1993
[3] http://www.twinery.org as of 22 Apr 2019

of interactive narrative. While game narrative used to be more closely associated with hyperfiction (the early 90s game *Myst* was originally developed in HyperCard [28], for example), we are still able to consider contemporary game narrative through the lens of hypertext as it stands today - as structured documents of content, connected by links, that a player or reader navigates through with their interactions.

Mark Bernstein has been one of the principle researchers of models of interactive narrative in the domain of hypertext. His seminal work on the patterns of hypertext [4] identified repeating structures within this space and established a vocabulary for representations of hyperfiction. Similarly, he, alongside Weal and Millard (who developed their own system on top of the FOHM model in Auld Leaky [30]), identified the key differences between the classical calligraphic structural approach (wherein pages are connected by links) and the sculptural approach (wherein conditional guard fields and state changing functions govern the way in which links are prevented/sculpted away) [5, 7]. More recently, Bernstein has combined the ideas of calligraphic and sculptural structures in his latest version of StorySpace [6]. Beyond that, Hargood et al. have explored how sculptural hypertext is ideally suited to support locative hyperfiction [22] and have built up Bernstein's patterns with their own work identifying high-level structures [25] and a new vocabulary of patterns for sculptural hypertext [21].

While these models ably describe structure, and in our previous work [19] we have explored applying some of them to a number of modern game narratives with success, they provide less in terms of a vocabulary or representation of the content within that structure. Modern game narrative is rich with varied mediums and storytelling techniques from environmental storytelling [23] to branching dialogue, mechanics-as-metaphor [20] to cut-scenes, and to better support this medium from a technological perspective, we may want to capture with our models the content as well as the structure. For example, while structure might tell us that one scene follows another, or may cause divergence based on player choice, it is harder to use these models to accurately represent a variety of different forms of interaction or play, let alone the subtleties of Discoverable Narrative [17].

## 2.2 Game Modeling Approaches

In our previous work [19], we described an application of three models to video games to determine their narrative representation successes and downfalls. We have since expanded upon this and examined two more broad approaches to game narrative modeling.

**Adding Interactivity to Narratological Models**

A common approach is to take existing theories that are not based on games and attempt to add support for interactivity or otherwise modify them to better suit games representation and analysis.

One such attempt was to inject interactivity into Campbell's Hero's Journey [15]. The original theory by Campbell [13] specifies 17 phases and is ideal for mapping linear narratives that conform to the progression outlined by the model. The authors highlight the lack of support for interactivity due to having no way to handle player choice and actions undertaken, which are fundamental to most gaming experiences. The authors suggest firstly keeping the player at the center of the action, and that resolution must be a result of player choice. Second, that when a choice is given to a player (begin journey, accept/refuse help, etc.) that each possible option must be valid and allow for progression of the narrative. They also note that a lot of games cause failure or backtracking by picking the 'wrong' choice, and suggest instead sanctions to be placed on players instead of halting. Third, that some optional phases must be conditional based on player ability (i.e., the phases activate based on the player's prior ability to succeed). In their final model, a number of additional phases are added particularly to choice and agency, and phases can link to any neighboring phase, including repetition of already passed phases. As the player encounters phases that could have more than one outcome, the pathway taken is now dependent upon the player's choice. Sequences ahead of the player's progress can also dynamically adapt to the success and ability of the player. In *Road of Trials*, for example, a setup could be made so that the player can only exit the sequence once adequate performance was reached, otherwise subjecting the player to repeated testing.

A similar approach was taken in the Narratification project [29], which attempted to unite narrative and gameplay in an analytical framework. The model upon which this work is based is Dixon's Goal, Motivation, and Conflict (GMC) model [16]. Dixon's model defines three categories for characters, each having internal (known to themselves) and external (public knowledge) factors. *Goal* refers to the objective the character would like to achieve, *Motivation* defines why they want to achieve their objectives, and *Conflict* determines what is stopping them from reaching their objectives. The authors highlight that this model is ideal for developing meaningful plots based on a character's internal and external factors, but that it does not capture any dimension of interaction from the player. They then extended this model into what they call Interactive Goal, Motivation, and Conflict (IGMC) by adding an additional column for the player alongside the character's column. The player and character columns are interconnected via the game experience. The GMC entries of the character must satisfy the entries of the player and vice versa. The authors note that this presented solution is aimed at designers being able to consciously plan out the relationship between player and character GMCs.

Propp's Morphology [26] has also been indirectly applied to video game narrative. In Brusentsev's work [12], Propp's Morphology is the basis of a game analysis framework in an effort to discover how well the original form of the theory applies to video game narrative. In this framework, a number of key sections (some of which use Propp's functions and character archetypes) are used to deconstruct the game. The *Game Structure Overview* provides a brief description of the game, highlighting unique narrative techniques. *Character Archetypes* lists all characters present in the game and assigns them to one of Propp's character archetypes. *Overall Story Arc* assigns Proppian functions to the overall story to allow for a high-level description of the structure. *Level Narrative* then breaks down the story into acts (or similar) and maps the functions again. *Impact of Player Decisions* assesses how player choice alters the narrative. *Dialogue* looks at how the in-game player dialogue choices move the narrative. Finally, *Morality* sees if ethical choices in the game alter the narrative. These elements are all considered for any given game. The authors conclude that by using Propp's functions, they can gain oversight of the structure of video game narrative, but

receive no further insight into it. They also note that the functions, in their original form is unable to handle any form of choice or agency, particularly from the player. One solution that they suggest without breaking Propp's original rules is a 'Decision Function', but this is purely speculative and was not further developed. This concept was further tackled by Bostan [10]. This project took a different approach by modifying and appending Propp's functions and their associated rule set to better suit games. Six of the original functions were modified and 15 new game-specific functions were added. In this framework, the story is broken down into any format the author sees fit (such as acts, chapters, levels) and then the extended list of Proppian functions are mapped in any order, can repeat any number of times, and can have branching simply by specifying connections between the functions. Although this approach is less rigorous than Propp's constraints, it does have the advantage of letting us analyze repeating patterns within game narrative. For instance, if there is a sequence of three functions repeated at numerous points that indicate a climax and battle, we could partly infer the pacing of the narrative from these repetitions, as well as begin to balance out the pacing by adjusting the spacing between the patterns.

**Componentization & Factorization**

Another approach is the use of Componentization and Factorization. These phrases are derived from the computing terms meaning to take a given system and reduce it into a number of logical constituents that work together. By using these methods on video games, we can develop a better understanding of their structure and look at how each component contributes to narrative.

One approach is to develop a taxonomy of game elements which can then be used for reference and analysis. An example of this is the interaction-centric structural framework by Björk [9], further refined in his later contributions [24]. This work represented all objects as components and defined their involvement as one of three types of actions: those instigated by the player, those of components with prescribed agency, and those originating from the game system itself. Another example is the Game Ontology Project [31] which similarly relies on a set of entity manipulation actions to declare events within the game. These high-level concepts can be used to broadly represent connected events within a narrative. For example, it's possible to take logs of interactions between these components to begin to build a picture of the experienced narrative after such events have taken place [14]. This concept has also been used to determine the effect of these kinds of components on the narrative [11] by surveying 80 games and identifying the relationships that emerge between components and various narrative forms.

A similar approach was taken by Bizzocchi [8], but with a particular focus on narrative rather than general structure. In this work, he identifies a number of factors of narrative in video games and how players interface with them. While this work defines its factors at an observational level, it can serve as initial groundwork for further structured analysis. A more component-oriented approach can be seen in Aarseth's framework [1]. In this framework, he breaks up games into the game worlds, objects, agents, and events. The world represents the physical structure of the game's level layout and differentiates between ludic and extra-ludic spaces in a way reminiscent of Adams' layout principles [2]. The objects are entities within the game world, categorized by their malleability and interactivity. Agents represent characters and are likewise classified by malleability but can be further divided into Bots, Shallow characters, or Deep characters. Events represent individual moments of narrative within the game and are considered either Satellites, which are supplementary events, or Kernels, which define particular stories. This is extended to say that all kinds of narrative can be defined as weighting of combined Satellites and Kernels, and that the type of narrative (linear, branching, etc.) can be determined from the ratio.

## 3 THE NOVELLA 2.0 MODEL

In our previous work [19], we proposed an early model that targeted video game narrative. This has since been refined into the system presented in this paper. The evolution of this model is significant as we subsequently felt that the previous model did not capture the concept of Discoverable Narrative in a meaningful way and that the model lacked any form of meaningful extensibility.

The concept behind this new model is a three-layered system of *Groups*, *Sequences*, and *Events*, and how they structured hierarchically as well as their connectedness. The three layers are similar in function but differ enough that when combined can create great complexity whilst remaining easy to understand. This model is designed with implementation and integration into engines, such as Unity[4] or Unreal[5], and other runtimes in mind. It does so by structuring itself to provide a specification of core functionality, yet deferring element functionality to a particular implementation. Figure 1 shows a high-level UML diagram of the model.

**Story.** The Story is a custodian of sorts, responsible for creation and management of all narrative elements. All Variables are globally accessible and are also managed by the Story. There always exists a single top-level Group in which all other content exists, which is stored in the Story. At any point, the Story can execute a set of Logic functions. This is of particular use when external runtimes which to query or modify the state of the Story, such as triggering an in-model element from an external source.

**Variables.** The state of the Story is controlled by a set of Variables. A Variable is a type-restricted piece of data, such as Boolean or integer, which can store arbitrary information. All Variables are mutable by default but can be declared constant. Variables have an initial starting value that is restored when Story execution begins. Since Variables are global, their identifying names must be unique.

**Groups.** Groups are the highest level container. Within a Story there is only one main Group and every other Group within the story is nested. Groups can be nested indefinitely for structural or organizational purposes. Groups act as a scope for their contents; elements inside are only able to act while their parent Group is active. All Groups nested at the same depth (i.e., are siblings) run in parallel during Story execution. However, Groups do not necessarily execute immediately as they are guarded by a Condition which determines when the Group will trigger. This means that while all Groups are parallel to their siblings, their Conditions may result in different execution timing. Groups therefore may execute instantly,

[4]Unity 3D, https://unity.com as of 22 Apr 2019
[5]Unreal Engine, https://unrealengine.com as of 22 Apr 2019

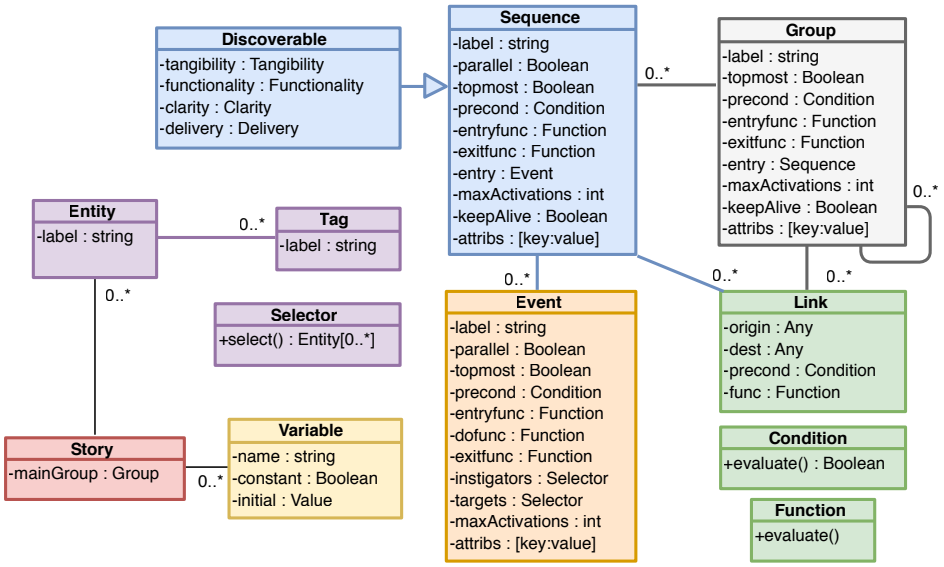



**Figure 1: High-level *Novella* model UML.**

with delay, or not at all. Groups also contain a list of Sequences. The connectedness of these sequences is determined by a list of Links. Each Group has an entry point which is either empty or one of the contained non-parallel Sequences. When a Group is entered and exited, a Function is optionally run that can modify the Story state. Groups can be marked as topmost which means that when entered, everything in the model pauses execution until the Group exits, at which point paused elements automatically resume.

**Sequences.** Sequences are individual segments of narrative made up of Events. Sequences are sequential as defined by their link structure in their owning Group, but can be optionally marked as parallel. A parallel Sequence will attempt to trigger when its parent Group is active (first at entry and then every tick), providing its Condition is met. Within a Sequence is a list of contained Events as well as a list of Links determining their connectedness. There is an entry point which is either empty or one of the contained non-parallel Events. As with Groups, when a Sequence is entered and exited, a Function optionally runs. Sequences, like Groups, can also be marked as topmost.

**Discoverables.** Discoverables are a specialization of Sequence that partially implements Discoverable Narrative. It appends the four matrix dimensions of Discoverable Narrative as enumerations and is always parallel. Due to the hierarchical scoping of this model, the parent Group determines when the Discoverable can be found. The added enumerations act as labels and have no direct influence on its contents. Instead, contained elements can use Logic to query the state of these enumerations and respond accordingly. This means that Discoverables are essentially a parallel Sequence with extra information that is triggered by a given Condition within a given scope. Because of this, more abstract kinds of Discoverable Narrative such as mechanics as metaphor are not easily represented. This is intentional as to not overcomplicate the model; attempting to wrangle every form of Discoverable Narrative into this model would increase its complexity without substantial return.

**Events.** An Event is a representation of a single narrative event. Where Groups and Sequences are containers, Events are leaves of such a tree and cannot be divided further. Sibling connectedness is defined by a set of Links of the parent Sequence. Similar to previously, Events can also be declared as parallel and topmost. Functions also fire on entry and exit as before, but Events are unique in that they also have an additional do Function. This Function is responsible for the actual behavior of the Event and is therefore delegated to the particular implementation via Logic. This is the primary manner in which the model defers Event specifics to a runtime. Events also contain two Selectors. The first determines the set of Entities that are instigating the Event, and the second determines the set of Entities that are targets or otherwise involved in the Event. The results of these Selectors can be queried and used by the Event's Functions.

**Attributes.** Groups, Sequences, and Events all have a dictionary that maps a string-based key to a value of any type. Using this Attribute system, it is possible to represent features such as the location within which an element takes place. To illustrate, let's take the "Medical Pavilion" level of *BioShock*[6], which is broken up into 11 subsections. We can represent the larger level as a Group and subsections as Sequences. Locations can then be represented by an Attribute with the key "location" and value of the location name. The Logic or runtime can then query this Attribute and respond accordingly. Earlier we acknowledged that some forms of Discoverable Narrative were not plausible in this model. However, using Attributes we can emulate simplified forms of environmental storytelling. An Attribute could be used, such as "environment", that stores a textual string written by designers instructing the implementation on how elements should be laid out. An interchange specification could be used for writers to adhere to (or a tool can provide an interface for) which could then could be parsed by

[6] *BioShock*, Irrational Games, 2007

an implementation. This is beyond the scope of this work but is important to consider.

**Simulation.** Groups, Sequences, and Events have an integer determining the number of times that they are allowed to (re)activate during simulation. This integer can either provide a positive upper bound to limit the executions, or be set to zero, meaning there is no limit. This means that if an activation request would put an element past its threshold, then the request should be declined. Additionally, a single element can only run one instance at a time, i.e., an already active element cannot be activated again until it terminates. By default, all container-like elements only remain active while their contents are currently active or have the potential to activate. Once all contents that could possibly activate have done so, the parent element will automatically terminate. If this behavior is not desirable, all elements contain a Keep Alive Boolean. When true, an element will maintain its active state even when its contents has expired. This could be useful, for example, to force an exit Function to not trigger unless explicitly desired. This does not apply to Events as they have no contents. Instead, their lifetime is determined by the duration of the in-engine event that takes place in their do Function and are "oneshot" by nature (i.e., activate, do, deactivate).

**Links.** Links tie together two elements of the same type. Sequences can only link to other non-parallel Sequences, and the same applies for Events. Each Link has a static origin, a mutable destination, a Condition determining if the Link can be traversed, and a Function fired upon traversal. During execution, if a set of Links are siblings (i.e., are outputs of the same element) then their Conditions are evaluated to determine availability. If more than one Link remains valid, then the implementation is required to resolve the stalemate to proceed. The concept of choice here is deferred to the implementation and does not necessarily result in explicit questioning à la a dialogue wheel. It simply means that a particular narrative chain cannot progress until one option is chosen.

**Functions & Conditions.** A Function is a set of logical statements that are able to, at a minimum, read and modify the Story state. Functions are used throughout the model at important structural points, commonly entries and exits. Conditions are a specialized Function in that they must result in a Boolean and are not able to mutate the Story state. They are used as guards in the model.

**Entities and Tags.** Characters and other narrative objects are enumerated as Entities. An individual Entity is a unique instance of an object. Entities have zero or more Tags, where a Tag is a unique identifier that partially declares the owner's role within the narrative. All Entities sharing a common Tag are equally viable to fulfill that role. Tags can be used as wildcards to enforce restrictions on Entity participation. For example, we may request all Entities of Tag `Enemy` and `Weaponized`, which would select all Entities that have these two tags (i.e., enemies alone wouldn't suffice as they must also be weaponized to be included).

**Selectors.** A Selector is a logical statement that returns an array of Entities. It is not able to mutate Story state. Selectors are used to determine which Entities are involved in an Event. As Selectors use Logic, they are dynamic by nature and can resolve in runtime.

**Logic.** The Logic of this model is a set of functions for interacting with the Story state. This collection is designed to provide minimal mandatory functionality for an implementation to fulfill. Logic does not detail itself with the particular language of implementation, as long as the core functionality is met. A software implementation of this model could use LUA, JavaScript, or anything else, as long as the requirements are met. This area is also the primary method of interaction between the model and runtime system. This is ideal for implementing custom per-game functions that may or may not concern themselves with the model. For example, an implementation could define a function to check if the player is near an entity, `isPlayerNearEntity(withTag: Tag, radius: Double) -> Boolean` , which when evaluated would return a Boolean identifying if an Entity with a given Tag was within a radius of the player. This could be used in a Condition, for example, to trigger an element within the model. A similar approach could be taken to query and modify the actual game's state. This Logic is a bidirectional communication layer between the model and a runtime. The mandatory functions can be found in Table 1.

**Table 1: Mandatory Logic declarations.**

**Variables**

`setValue(var: Variable, value: Type) -> Void`
Sets a Variable value. Present for all data types.

`getValue(var: Variable) -> Type`
Gets a Variable value. Present for all data types.

**Entities**

`entity(named: String) -> Entity`
Gets an Entity by name.

`entities(withTags: [Tag]) -> [Entity]`
Gets all Entities with the given Tags.

`add(entity: Entity, tag: Tag) -> Void`
Adds a Tag to an Entity.

`del(entity: Entity, tag: Tag) -> Void`
Removes a Tag from an Entity.

**Components**

`trigger(type: Type, delay: Int) -> Void`
Triggers a given Group/Sequence/Event with a delay in ms. Present for all three.

`stop(type: Type, delay: Int) -> Void`
Stops a given Group/Sequence/Event with a delay in ms. Present for all three.

`pause(type: Type) -> Void`
Pauses a given Group/Sequence/Event. Present for all three.

`resume(type: Type) -> Void`
Resumes a given paused Group/Sequence/Event. Present for all three.

`tangibility(disc: Discoverable) -> Tangibility`
Gets the Tangibility of a Discoverable Sequence.

`functionality(disc: Discoverable) -> Functionality`
Gets the Functionality of a Discoverable Sequence.

`clarity(disc: Discoverable) -> Clarity`
Gets the Clarity of a Discoverable Sequence.

`delivery(disc: Discoverable) -> Delivery`
Gets the Delivery of a Discoverable Sequence.

`activations(type: Type) -> Int`
Gets number of times the Group/Sequence/Event has activated. Present for all three.

## 3.1 Worked Example

In order to demonstrate the capability of this model, we will now explore a worked example using a sequence from *Life is Strange*[7]. A simplified overview is displayed in Figure 2. Following the opening nightmare sequence, Max jolts awake during a photography lesson (1). She then gathers her bearings while a lecture takes place (2).

[7] *Life is Strange*, Don't Nod Entertainment SA, 2015


**Figure 2: High-level overview of the sequence. 1) Max wakes up in class. 2) A lecture is taking place. 3) Stella drops her pen. 4) Taylor bullies Kate. 5) Victoria's phone vibrates. 6) Max's desk interactions. 7) Max's camera usage prompt. 8) Max's questioning for disturbing class.**

During this, Stella drops her pen (3), Taylor bullies Kate (4), and Victoria's phone rings (5). These three events happen in parallel to the lecture and Max's pondering. At this stage, Max can interact with a photo on her desk, and after doing so, four more items become available (6). Interacting with all objects but the camera will not halt the lecture, but have Max narrate them in parallel. If the camera is not selected in due time, the lecture begins to loop a series of questions to the class. When Max interacts with her camera (7), which disturbs the lesson, the narrative continues. Max is then punished for her intervention with a question (8), after which the bell rings to end class. This sequence was chosen as it demonstrates typical narrative actions in video games, but also includes parallel events, gating of progression, and player choice.

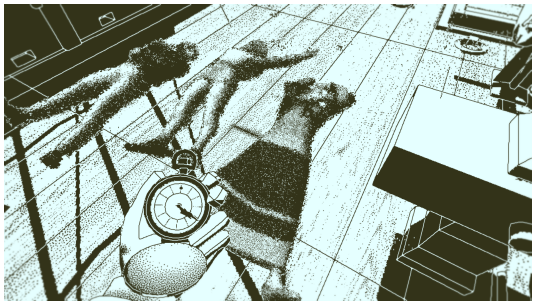
**Entities.** We must first decide and enumerate the involved Entities. Depending on the granularity desired, different objects may be included. For instance, fine-grained approaches may consider the paper ball Taylor throws at Kate as an Entity and model the individual Event for throwing it, whereas a more coarse approach may represent the short bullying scenario as a single Event including two Entities. This is usually influenced by the particular use-case. This concept of granularity applies to all of the model. For simplicity, we will take a coarse approach. As such, as can enumerate our character as Entities: `Max`, `Jefferson`, `Stella`, `Kate`, and `Victoria`. Tags are not necessary as we can rely on direct referencing.

**Groups & Sequences.** We can assume that the classroom scene is part of a single Group. Since there are a number of individual happenings that run in parallel, and some that halt until given conditions are met, we need to define Sequences to represent this structure. Max talks to herself four times before being able to interact with all five items on and around her table. This happens while Jefferson's lecture is ongoing and therefore we can put it in a Sequence named `Max`. Jefferson's lecture can be broken up into two Sequences: `Jefferson1` and `Jefferson2`. The former runs until Max interacts with her camera, looping otherwise. The latter involves querying Max and class ending, which is instigated by Max interacting with her camera. We can also enumerate the Sequences `Stella`, `Taylor`, and `Victoria` for Stella dropping her pen, Taylor bullying Kate, and Victoria's phone vibrating. These short Sequences happen in parallel to `Jefferson1` and `Max`.

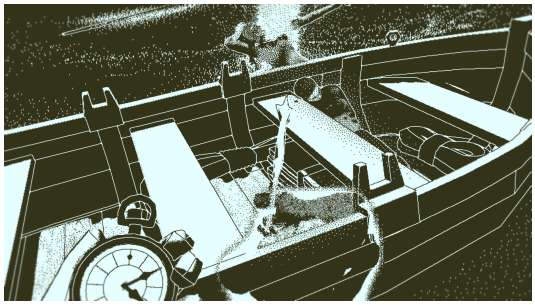
**Max's Inner Thoughts.** During the `Max` Sequence, Max narrates to herself four times. This can be thought of as dialogue with oneself. However, since the speech is inner (i.e., thinking), it may be advantageous to differentiate from regular dialogue. Let's assume our implementation has defined a `thinking` function that requires an Entity and textual string. This can then be used within an Event's do Function, reading the instigating Entity and providing the spoken string. For Max's thoughts, these Events would simply have Max as the instigator and call the implementation's `thinking` function, passing in the instigator and a text string for whatever Max is thinking for the particular Event.

**General Dialogue.** More general dialogue could be handled with a defined `dialogue` function, again provided by the implementation. This would largely mimic the `thinking` function but differ in that it takes two sets of Entities read from the instigators and targets. For example, in the `Jefferson1` Sequence, Victoria and Jefferson engage in conversation. These individual speech acts could be represented by a sequential set of connected Events that use the `Dialogue` function with appropriate Entities selected for involvement. In the case where Jefferson addresses the whole class, if we wanted to include all students as target Entities, then we could use a Selector to directly reference them all, or if a `Student` Tag had been assigned, select all Entities with that Tag instead.

**Parallel Sequences.** The `Stella`, `Taylor`, and `Victoria` Sequences all execute during `Jefferson1` at fixed times. We could handle this in a variety of ways. If specific timing is known, all three Sequences could be started upon entry of `Jefferson1` with a delay using the `trigger(seq: Sequence, delay: Int)` Logic function. They could also be triggered after a particular Event for more fine-grained control with or without delay. Alternatively, the Sequences could have a starting Condition that checks a truth value of a Boolean Variable which could be set to true, triggering the Sequences. As mentioned earlier, Max's four thoughts are parallel to Jefferson's lecture. This can be implemented by having the `Max` Sequence as a sibling of `Jefferson1`, and ensuring that they both execute at the same time.



**(a) Entering Chapter IV, Part 4 by interacting with Bun-Lan Lin's body. Parts 1, 2, and 3 unavailable at this point.**



**(b) Entering Part 2 from within Part 4 by locating and interacting with Patrick O'Hagen's body.**

**Figure 3: Entering Part 4 and discovering Part 2.**

**Gating Progress.** In the `Max` Sequence, there are five items on and around Max's table that can be interacted with: her camera, pencil case, photograph, journal, and backpack. However, four remain inactive until the photograph is first picked up. This could likewise be implemented multiple ways. An easy way is to have all five elements as non-parallel Events within the `Max` Sequence. The photograph Event would follow Max's pondering Event chain. This Event would then connect to all four other Events, as well as back to itself. All other Events except the camera would likewise link to all five Events. The result is that the player must first interact with the photograph before being able to access the remaining objects. Interaction with these objects can then be looped until the camera is chosen because of their connectivity. Of course, using the `activations(type: Type)-> Int` Logic function, we could alter the narrative content for variance each time the Events are visited by responding to the returned count.

If Max does not interact with her camera in time, `Jefferson1` begins to loop a set of four dialogues from Jefferson to the class. Once Max interacts with her camera at *any* point that it is available during `Jefferson1`, the Sequence terminates and `Jefferson2` initiates. This can be handled by setting the Function of the Event representing the camera interaction to use the `stop(Jefferson1,0)` and `trigger(Jefferson2,0)` Logic functions in order. This would firstly terminate `Jefferson1` regardless of its progress, and start `Jefferson2` immediately thereafter, which is how the game works.

**Player Choice.** `Jefferson2` starts after Max interacts with her camera. In this Sequence, Max is made to answer a question as punishment for disturbing class. This presents two options to the player, both of which reconcile to the same Event to give a perception of agency over the narrative. We simply need to link the preceding Event for dialogue to both potential answers, as multiple outputs must be resolved before continuing. The way in which the choice is presented is left up to the implementation. In the case of this example, a dialogue wheel is presented to make a binary choice which would in turn trigger the model to follow one pathway. Similar cases later in the game are more complex as they pause all other elements during execution. This can be achieved by manually using the `pause` and `resume` functions, or by marking the appropriate element as topmost.

## 3.2 Discoverable Narrative via Structure

We will now look at how the model's structure allows for some forms of Discoverable Narrative. *Return of the Obra Dinn*[8] is a contemporary murder mystery game that makes extensive use of Discoverable Narrative concepts. The game is presented in chapters that are made up of a varying number of parts that form a sequential narrative. The player does not necessarily access them in such a way, often beginning in the latter half of a chapter and then working backwards by finding the deceased within each part. For example, if the player enters part 4, they will be able to find and enter at least the previous part. Players must locate the deceased (or related items) and interact with them, transporting them to a freeze-frame at the moment of death, aiding them in solving the mysteries.

Figure 3a shows Bun-Lan Lin's body on the main deck, which when interacted with will transport the player to Chapter IV, Part 4. There is no way to access previous parts without firstly entering here. Once the player enters this scene, they are able to locate and interact with previously deceased characters which act as triggers to backtrack throughout the rest of the chapter. Depending on the scene, zero or more parts will be accessible. In this scene, the player can interact with three of the deceased (parts 1, 2, and 3). This can be seen in Figure 3b with Bun-Lan Lin on the right and Patrick O'Hagen in the foreground. Locating and interacting the latter of these two will teleport the player back to part 3.

We can model this using Groups, Sequences, and Logic. We can declare a Group for the overall chapter, `ChapterIV`. Implementing each part as a Sequence would be unwise as only Events can be contained within. Instead, we can declare each part also as a Group and nest them as siblings within `ChapterIV`. This form can be seen in Figure 4 where Group `Part4` is expanded. Within this expanded Group is three Sequences, one for each of the preceding parts. We can assume for this example that the contents of such a Group has been implemented and that each of the previous part Groups have this format replicated for their own preceding parts respectively. Now we can define a Condition for each of the Sequences using Logic decided by the implementation. In this case, we could perhaps define a function for checking interaction with a given Entity, or alternatively handle interaction in the runtime and manually

[8] *Return of the Obra Dinn*, Lucas Pope, 2018

trigger these Sequences by name. We can then use the exit Function of the Sequences to handle the consequences of interacting with the deceased characters. In this case, we could terminate the current Group by using the `stop(group: Group, delay: Int)` function, and start the new Group by using the `trigger(group: Group, delay: Int)` function. If we want a given set of Events to trigger before switching Group, then we can embed them within the appropriate Sequence, but in this case, this is not necessary. With this solution, we could have used Discoverable elements instead of Sequences, which would provide us with a queryable set of enumerations that could further enrich the possibilities both from within and external to the Sequence. It could also be useful in that an implementation may wish to differ between what is discoverable and what is not regardless of functionality.

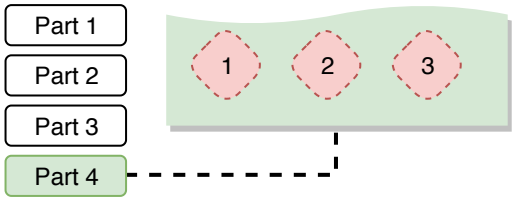



**Figure 4: High-level possibility of Discoverable Sequences in *Return of the Obra Dinn* Chapter IV - The Calling.**

## 4 SYSTEM ARCHITECTURE

The *Novella* platform consists of an authoring tool and an API. The architecture of this platform and the typical pipeline it follows is demonstrated in Figure 5. *Novella* is a general framework for interactive narrative that respects that a game story might be delivered in any one of a range of game engines. An author will use the authoring tool to create a narrative plan by specifying content, structure, and connections to the game engine, which is then exported as JSON ready for interpretation by the *Novella* API in any given game engine. This is then integrated by the game developer using the API. This potentially allows for the same story to be delivered in multiple engines, and for games to interpret and incorporate the content in ways appropriate for each piece. The *Novella* implementation is fully open-source[9] and a screenshot of the authoring tool can be seen in Figure 6.

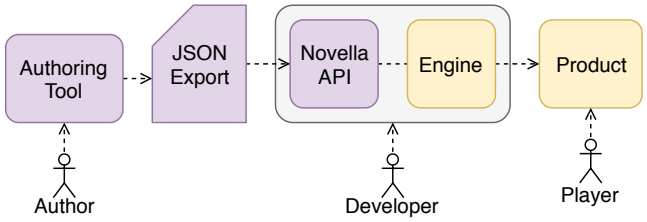



**Figure 5: *Novella* system architecture.**

The authoring tool is an environment that implements the theoretical model for visual creation of the story structure and content.

[9]Available at https://github.com/KasumiL5x/novella as of 22 Apr 2019

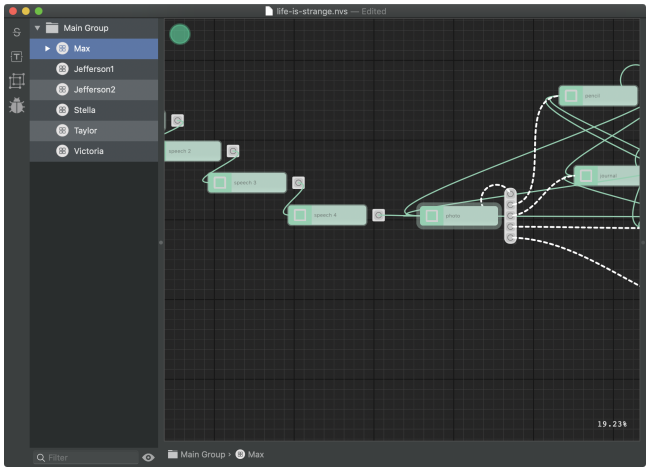



**Figure 6: *Novella* authoring tool prototype.**

In our case, we have implemented a standalone authoring application for macOS using Swift. The advantage of creating a standalone tool is that the editor is no longer accustomed to a given engine or product. An inherent limitation of this approach with regards to integration is that the tool is unable to directly reference in-game object instances and other engine assets without some form of interop (such as a UUID), and even then may not be possible. A custom authoring tool built directly into an engine would not have such limitations, and is a feasible approach, but removes the engine neutrality of our own approach. As our implemented tool is standalone, the authors must export their story to an intermediate JSON representation for interpretation by the API.

The *Novella* API is an implementation of the theoretical model for direct use within an existing codebase, essentially as a plugin for a game engine. A compliant API will be wholly usable by a developer without use of the authoring tool. It is responsible for the creation, management, and simulation of the story, whereas the engine is responsible for implementation of external Logic that connects the two. Typically, a developer loads the JSON file exported from the authoring tool into the API, which in turn parses and automatically configures the story accordingly. The developer can then fine-tune the loaded story as they see fit, such as linking variables to actual in-engine object instances or connecting story Events with particular gameplay mechanics (for example, if a scene calls for the player to become dazed, this is the point where the story model will communicate this and the developer will connect this to the appropriate part of the game engine to instantiate the effect). The resulting simulation contributes to the final game (or otherwise) that the player interacts with.

### 4.1 API Implementation

For this project we have implemented a *Novella* API for Unity as a proof of concept - while only developed for this game engine currently, similar APIs could be developed for other engines such as Unreal. Targeting Unity meant our API was restricted to C# 4.x and we were unable to use multithreading as a number of Unity functions must be called from the main thread only.

We chose to use LUA to fulfill the Logic component. The main Story class maintains an instance of the LUA session. When the session is configured, all C# data types that are to be used in LUA (Groups, Sequences, Variables, and so on) are registered, and functions written in C# fulfilling the mandatory Logic of the model are also mapped to LUA identifiers so that they can be called from scripts. A wrapper for the LUA session is exposed in the API and provides the ability for developers to register custom types (such as Unity types), custom variables, custom functions, and to at any time run arbitrary LUA code for querying and instructing the model. The contents of Functions, Conditions, and Selectors are automatically wrapped in LUA `function` code. To avoid name clashes, a unique function name is generated automatically for each function using their UUID. Conditions and Selectors also contain failsafes in the case where the script does not return the expected data.

A naive solution to executing all parallel elements would be to run each in a separate thread, terminating only when its child threads join. With complex stories, this could quickly reach sensible threading limits. Since we were unable to use threading regardless, we opted for a hierarchical stepped simulation where each step forwards the simulation by one tick. In the case of Unity, this is called in the `Update` function of an object present in the scene. Stepping a single element will in turn step all of its children providing they are active. So, by stepping the main Group of the story, we inherently step everything in the model, providing it is active. Groups, Sequences, and Events all follow a three-stage life cycle during simulation: Activation, Stepping, and Deactivation. Activation and deactivation are where construction and deconstruction happen, such as handling entry points and running valid Functions. When an element is stepped, it attempts to enable all of its inactive children, steps its already active ones, and then handled its own behavior. Events differ in that they simply run their `do` function and then terminate. For Groups and Sequences, a sequential link of child elements must be followed (if present) in addition to those that are parallel. This is achieved by child elements, when deactivating, notifying their parent. Then, the parent determines, where applicable, the next available connected node(s) and either follows it or requests that the engine resolves the stalemate in the case where multiple exist.

## 5 INNOVATIONS

The *Novella* model takes a unique approach to modeling of game narrative and contributes a number of innovations in its methods.

### Extensibility and Deference of Responsibility

In our previous iteration of the model, we had a limited set of individual events (such as Dialog) that contained predetermined data. This was limiting, as it's not really feasible to capture every kind of event, nor to generalize them, and it is perhaps not our place to determine what data each event consists of. In the model iteration presented in this paper, we have taken a deferred approach to solve this problem. We provide a basic Event, as described earlier, that maintains a dictionary of arbitrary data to be used in the Event. This way, an implementation can easily add any data it requires; the data is deferred and is not predetermined. By doing this, we are providing the *functionality* of an Event, but leaving the details up to the implementation. This philosophy is followed elsewhere throughout the model. Additionally, the Logic of this model serves as a connection between the model and an engine. The specifics of functionality, such as what Events actually do in-game, are likewise deferred through the use and extension of Logic. The language used to fulfill the requirements of Logic is arbitrary; in the case of our example, we used LUA, but any other language could just as readily be used in its place.

This approach presents a story model that can richly define the structure and content of a game's narrative without having to accustom itself with the rules, systems, or mechanics within the game. The *Novella* API will interpret the story structure, leaving the engine free to connect and apply the detailed functionality.

### Game Content Support

Through the use of Logic, Attributes, Tags, and Selectors, we are able to build up a machine-readable language for the content of our Groups, Events and Sequences. This moves beyond what has been provided by some models of interactive narrative which supply rules and descriptions of structure but not the content itself within the nodes. We can use this additional content model to indicate and communicate the effect of the story on characters or the environment, changes in rules and systems, or to communicate how something is displayed or presented within the interface. For example, making a specific choice might put a particular item into a character's inventory (use of two Entities), a scene may change if there are hostile enemies nearby (use of Selectors, Logic, and Tags), or a scene might occur that is not directly delivered to the player in the foreground but occurs in the background of an area as a piece of environmental storytelling (use of Attributes).

It is to be noted that the *Novella* authoring tool and model do not directly handle this functionality - as previously stated, that is deferred to the runtime. What it does provide is a vocabulary and model for the author to express this story structure and content in a machine-readable fashion. Applying this specific content within a game is deferred to the runtime itself.

### Support for Discoverable Narrative

Our approach incorporates features that aid it for modeling games whilst still retaining a general enough system to be used for other hypertext-based narrative modeling. We incorporate a limited set of Discoverable Narrative elements by extending a standard Sequence to support the enumerations allowing for queries so that its contents (or other) can react accordingly. This is an area we wish to improve clarity and support for in future iterations of this model. Additionally, while not particularly new, we acknowledge the need for broad groupings of objects and dynamic selection of those objects in games, which is fulfilled through the use of Tags and Selectors respectively.

### Calligraphic Structure in a Sculptural Context

Mixing calligraphic and sculptural hypertext for interactive narrative is not new; Storyspace 3 does this with sculptural elements existing within a broader calligraphic context [6]. However, we have approached this in a slightly different way in that while individual narrative sequences are essentially calligraphic, they exist

within a broader sculptural framework whereby several parallel Groups or Sequences may be active simultaneously listening for relevant triggers and interactions. As such, the Events within these Sequences, while linked calligraphically to their siblings, are implicitly linked in potential to the Events of other Sequences. A player in an RPG, for example, may pursue the main storyline only to trigger a sidequest half way through and divert their attention to that. While structurally the main quest and side quest are made up of calligraphically-ordered Sequences of narrative Events, the potential to shift between the two is closer on a structural level to sculptural hypertext. As such, as have created a calligraphic structure for our individual elements and their constituents, but they exist within a sculptural context to enable this form of flexible storytelling that is common within games.

## 6 CONCLUSION AND FUTURE WORK

In this paper, we have presented our theoretical model of interactive narrative, *Novella 2.0*, an iteration upon our previous work which is better suited to supporting video game narrative. This model is presented as part of the *Novella* platform, a systems architecture composed of an authoring system, the model, and an example implementation in the form of a C# API.

Our model, as part of the described system architecture, takes novel approaches in its explicit deference of responsibility to other systems (such as game engines) supporting narrative content and structure, but recognizing that in a game this sits within other structures and systems. Taking lessons from hypertext research we support our own approach to mixing sculptural and calligraphic structures but rather than sculptural structures within a calligraphic structure (as in StorySpace 3 [6]) we inverse this as calligraphic structures within a sculptural structure. Finally we also expand our vocabulary to address specific aspects of game narrative content beyond structure, particularly in improving upon our previous model iteration by providing better and more explicit support for Discoverable Narrative.

Our future work includes working towards unification of our theoretical model with a sophisticated User Experience for authors. This refers to the experience of the authoring process, and the interface of our authoring tools is a large part of this. We have previously made initial explorations into User Experience and interface paradigms in authoring tools [18]. We intend to further continue this work with experiments to discover hindering and aiding practices within User Experience and interface design for interactive narrative authoring tools. We also intend to continue the refinement of the theoretical model to ensure that it is best capable of capturing the nuances of video game narrative while still maintaining an understandable and eloquent model design.

## REFERENCES


[1] Espen Aarseth. 2012. A Narrative Theory of Games. In *Proceedings of the International Conference on the Foundations of Digital Games (FDG '12)*. ACM, New York, NY, USA, 129–133.

[2] Ernest Adams. 2014. *Fundamentals of Game Design* (3rd ed.). New Riders Publishing, Thousand Oaks, CA, USA.

[3] Claus Atzenbeck, Thomas Schedel, Manolis Tzagarakis, Daniel Roßner, and Lucas Mages. 2017. Revisiting Hypertext Infrastructure. In *Proceedings of the 28th ACM Conference on Hypertext and Social Media*. ACM, 35–44.

[4] Mark Bernstein. 1998. Patterns of Hypertext. In *Proceedings of the Ninth ACM Conference on Hypertext and Hypermedia*. ACM, New York, NY, USA, 21–29.

[5] Mark Bernstein. 2001. Card shark and thespis: exotic tools for hypertext narrative. In *Proceedings of the twelfth ACM conference on Hypertext and Hypermedia*.

[6] Mark Bernstein. 2016. Storyspace 3. In *Proceedings of the 27th ACM Conference on Hypertext and Social Media*. ACM, New York, NY, USA.

[7] Mark Bernstein, David E. Millard, and Mark J. Weal. 2002. On Writing Sculptural Hypertext. In *Proceedings of the Thirteenth ACM Conference on Hypertext and Hypermedia (HYPERTEXT '02)*. ACM, New York, NY, USA, 65–66.

[8] Jim Bizzochi. 2007. Games and Narrative: An Analytical Framework. *Loading - The Journal of the Canadian Games Studies Association* 1, 1 (2007), 5–10.

[9] Staffan Björk and Jussi Holopainen. 2003. Describing Games - An Interaction-Centric Structural Framework. In *Proceedings of the 2003 DiGRA International Conference: Level Up*, Vol. 2. 4–6.

[10] Barbaros Bostan and Orcun Turan. 2017. Deconstructing Game Stories with Propp's Morphology *(system)*, Vol. 17. Ärstanbul, TURKEY, 18.

[11] Jeffrey E. Brand and Scott J. Knight. 2005. The Narrative and Ludic Nexus in Computer Games: Diverse Worlds II. In *Proceedings of the 2005 DiGRA International Conference: Changing Views: Worlds in Play*, Vol. 3.

[12] Andrew Brusentsev, Michael Hitchens, and Deborah Richards. 2012. An Investigation of Vladimir Propp's 31 Functions and 8 Broad Character Types and How They Apply to the Analysis of Video Games. In *Proceedings of The 8th Australasian Conference on Interactive Entertainment: Playing the System (IE '12)*. ACM, New York, NY, USA, 2:1–2:10.

[13] Joseph Campbell. 2008. *The Hero with a Thousand Faces*. New World Library.

[14] Yun-Gyung Cheong and R. Michael Young. 2006. A Framework for Summarizing Game Experiences As Narratives. In *Proceedings of the Second AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE'06)*. AAAI Press, Marina del Rey, California, 106–108.

[15] Guylain Delmas, Ronan Champagnat, and Michel Augeraud. 2007. Bringing Interactivity into Campbell's Hero's Journey. In *Proceedings of the 4th International Conference on Virtual Storytelling: Using Virtual Reality Technologies for Storytelling (ICVS'07)*. Springer-Verlag, Berlin, Heidelberg, 187–195.

[16] Debra Dixon. 2013. *Goal, Motivation and Conflict: The Building Blocks of Good Fiction*. Bell Bridge Books.

[17] Daniel Green. 2019. *Don't Forget to Save! The Impact of User Experience Design on Effectiveness of Authoring Video Game Narratives*. Transfer Thesis. Bournemouth University.

[18] Daniel Green, Charlie Hargood, and Fred Charles. 2018. Contemporary issues in interactive storytelling authoring systems. In *International Conference on Interactive Digital Storytelling*. Springer, 501–513.

[19] Daniel Green, Charlie Hargood, Fred Charles, and Alexander Jones. 2018. Novella: A Proposition for Game-Based Storytelling. In *Narrative and Hypertext 2018*. ACM, Baltimore, Maryland.

[20] Mads Haahr. 2018. Playing with Vision: Sight and Seeing as Narrative and Game Mechanics in Survival Horror. In *International Conference on Interactive Digital Storytelling*. Springer, 193–205.

[21] Charlie Hargood, Verity Hunt, Mark J. Weal, and David E. Millard. 2016. Patterns of Sculptural Hypertext in Location Based Narratives. In *Proceedings of the 27th ACM Conference on Hypertext and Social Media*. ACM, New York, NY, USA, 61–70.

[22] Charlie Hargood, Mark J. Weal, and David E. Millard. 2018. The StoryPlaces Platform: Building a Web-Based Locative Hypertext System. In *Proceedings of the 29th on Hypertext and Social Media (HT '18)*. ACM, New York, NY, USA, 128–135.

[23] Henry Jenkins. 2004. Game Design as Narrative Architecture. *First Person. New Media as Story, Performance, and Game (eds.) Pat Harrigan and Noah Wardrip-Fruin* 44 (2004), 53.

[24] Petri Lankoski and Staffan Björk. 2015. Formal Analysis of Gameplay. In *Game Research Methods*. ETC Press, Pittsburgh, PA, USA, 23–35.

[25] David E. Millard, Charlie Hargood, Michael O. Jewell, and Mark J. Weal. 2013. Canyons, Deltas and Plains: Towards a Unified Sculptural Model of Location-based Hypertext. In *Proceedings of the 24th ACM Conference on Hypertext and Social Media*. ACM, New York, NY, USA, 109–118.

[26] V. Propp. 2010. *Morphology of the Folktale: 2nd Edition*. University of Texas Press.

[27] Nitin Sawhney, David Balcom, and Ian Smith. 1996. HyperCafe: Narrative and Aesthetic Properties of Hypervideo. In *Proceedings of the the Seventh ACM Conference on Hypertext (HYPERTEXT '96)*. ACM, New York, NY, USA, 1–10.

[28] Ted Smith and Steve Bernhardt. 1988. Expectations and experiences with HyperCard: a pilot study. In *Proceedings of the 6th annual international conference on Systems documentation*. ACM, 47–56.

[29] Nils Sørensen and Mette Pødenphant. 2013. Narratification: Unifying Narrative and Gameplay. (2013).

[30] Mark J. Weal, David E. Millard, Danius T. Michaelides, and David C. De Roure. 2001. Building Narrative Structures Using Context Based Linking. In *Hypertext '01. Proceedings of the Twelfth ACM conference on Hypertext, Aarhus, Denmark*. 37–38.

[31] José P. Zagal, Michael Mateas, Clara Fernández-Vara, Brian Hochhalter, and Nolan Lichti. 2005. Towards an Ontological Language for Game Analysis. In *Proceedings of the 2005 DiGRA International Conference: Changing Views: Worlds in Play*, Vol. 3.