Campfire Tales: Augmenting Modern Oral Traditions Through Performance and Procedural Generation

Matthew Deline, MA Independent Games and Playable Experience Design, Goldsmiths

Abstract—Performance is one of the oldest, if not *the* oldest, forms of human creative expression. It was used to develop language, social structures, remember historical events, as a tool for survival, and gave us a voice as storytellers. Compared to many millennia of development in practice and form for these traditions, modern artificial intelligence algorithms that allow computers to create stories of their own are still in their infancy. Yet their journey is shockingly similar. What would happen if we were to combine the two?

In this paper, we discuss the transformation of ancient oral traditions to modern western fireside stories, examine several examples of procedural storytelling in interactive media and games, and show how they informed the development of *Campfire Tales*, an immersive generative narrative experience shown at the *Playful Experiences* exhibition at Goldsmiths, University of London in September 2018.

1 Introduction

1.1 What is Campfire Tales?

WHEN I was growing up, I learned the alphabet by playing a storytelling game with my dad we called the "Alphabet Game." He would tell me a story at bedtime where he would periodically pause because the main character encountered a "blank," where the blank would start with the next letter of the alphabet. This led to lots of stories with alligators or apples going on boating adventures with cobras. I adapted this storytelling game later in life with friends on camping trips, where you would go around in a circle and each time a new blank occurs, the next person would then become the storyteller. It's the kind of experience that in either case is impossible without some kind of improvisation, and dialogue between audience and orator. It's also incredibly difficult to simulate this type of relationship using computer software. And by taking cues from storytelling games like the Alphabet Game that use a human agent as a moderator between the ruleset and shared performance, we can address some of that difficulty.

As a former actor and sometimes writer turned game developer, I find the intersection between art and technology particularly compelling when they meet in performance. So, the goal for *Campfire Tales* was to be able to replicate my childhood memories for others while using all of the various skills that I have learned to make something unique. The end result is an installation based experience that combines the most important parts of interactive media through performance and group interaction, and the symbolic relationship between procedural generation and storytelling through oral traditions.

• *Matthew Deline is a member of the 2018 graduate class of the MA Independent Games and Playable Experience Design at Goldsmiths, University of London. E-mail: deline.matt@gmail.com*

Campfire Tales is a storytelling game where players interact with a generated narrative around a digital campfire, where they are periodically given the opportunity to introduce new words into the story that are incorporated into the telling. When a story has finished, a banner image of their complete tale is saved so their favorites can be shared with their friends, or kept for rereading aloud. In the version to be displayed at *Playful Experiences*, the game will be *performed*, where the story is read aloud by a human moderator who interacts with the audience to make group decisions on added elements, and the game system manipulates a theatrical campfire that will change color and react based on player interaction. It is a symbiotic digital recreation of an oral tradition.

2 Oral traditions, campfire tales and storytelling in games

2.1 What is an Oral Tradition?

Oral traditions, defined as both the process and the products of "verbal messages which are reported statements from the past beyond the present generation" are some of the earliest forms of human performance. The process is the repeated spoken telling of an original message over time until it has been lost, and the product is the new message produced by each new repetition [1]. This method of storytelling predates the concept of the stage, and has been used by numerous cultures, nations and civilizations as a way to transmit knowledge throughout human history. Oral traditions themselves are a form of ritual, where there is importance placed not only upon the tales themselves, but to the reverence placed upon the places and ways in which they were told [2].

Modern campfire stories, told at firesides at campsites

around America during summer camps, scout events, and countless individual camping trips every year *are* their own form of oral tradition. Like the warnings of magical danger seen in the stories from other modern oral traditions elsewhere in the world, they allow us to pass on warnings and cautionary tales in the form of urban legends [3]. They allow us to spread information organically through pre-existing social constructs, and share vital information in an interesting and entertaining manner.

2.2 What Makes a Good Campfire Story?

After reviewing dozens of campfire stories published online and in book form, speaking with former scout leaders, and drawing upon my own experiences from summer camps as a child, I've reached a framework for what I consider to be a *good* campfire story [4], [5], [6].

They must have a proclaimed truth, a spooky setting, a twist, and the ability to change to fit the audience. They can have a moral, a warning, new knowledge, or mythical characters. Most of these assertions that I made before writing the stories ended up being confirmed through player feedback during the development of the game as well (See Appendix A).

2.3 Example: The Evolution of Stories and Campfire Americana in *Where the Water Tastes Like Wine*

Figure 1: Telling stories at a campfire with another lost soul.

In *Where the Water Tastes Like Wine*, players have made an unfortunate bet with the Dire Wolf and are forced to travel across the United States of America collecting stories as payment to free their soul [7]. Players collect stories by interacting with points of interest on the map that are indicated with a special prompt, and are added to their inventory to be used later. These collected stories can change over time as they are re-told in roadside homes, cities, and campfires, and players often have the ability to affect *how* they change when they are given a choice to say they are true or not. For example, a story about a strange woman that you encounter in a town can change and grow to become a story about a woman who was actually a ghost! These stories fit certain moods in the game, and are used to fulfill requests in special conversations around a campfire with other fellow lost souls found throughout the game world. By matching the right stories, like a funny story for someone who wants to laugh, the player reveals the true nature of these special characters, freeing them from their falsehoods and in turn helping the player free themselves from their own curse.

While *Where the Water Tastes Like Wine* doesn't generate text, as all of the potential variations are pre-written, and the only procedural generation that the game has is in the distribution of the stories across the landscape, it's still an incredibly interesting take on the evolution of stories over time, and is an excellent example of the model of oral traditions in games.

2.4 Example: Interactive Storytelling through Dice in Rory's Story Cubes

Figure 2: Rolling the dice to create a story.

Rory's Story Cubes is a great example of an interactive narrative with audience participation that uses images on die faces to represent the individual segments of the game's possibility space [8]. In the game, players are encouraged to roll a certain number of dice, and to improvise a story based on the images seen on the die faces. In the Epic Stories variation of the rules, nine cubes are rolled and used by the first player to create the opening chapter of a new story.

Using the cubes in the image above, we could tell a story about Rob the Alien who is flying on an airplane from the moon to go to the theater. When he arrives, he discovers that the entrance to the door is locked, and his monstrous shadow has taken the key! In each subsequent turn, other players switch from audience member to storyteller, and roll the dice to create a new chapter of their own. The last player is tasked with bringing all the stories together to a satisfying conclusion, perhaps one where Rob has become best friends with his shadow and they now both travel the stars as a comedic duo inspired by the show that they were locked out of in the first chapter.

3 Procedural Content Generation (PCG)

3.1 What is PCG?

For *Campfire Tales* to present this type of interactive performance that incorporates audience participation by using software, it needs to be able to improvise or create new content throughout the experience. This could be like generating the stories from the cube faces as we did with *Rory's Story Cubes*, or by creating the images that are used as

prompts for the players. We can do this digitally through *Procedural Content Generation* (PCG), which is "the algorithmic creation of game content with limited or indirect user input, [where] computer software can create game content on its own, or together with one or many human players or designers" [9]. Content that is created can be anything produced by the software, from enemy statistics, levels, and location names to unique audio and imagery. For Campfire Tales, content that is created could be written pieces of story to be read by a moderator, and color choices for environmental lighting and effects.

There are six established methods that can be used to create PCG: Distribution, Parametric, Tile-based, Grammars, Constraint-Solvers, and Agents and Simulations [10].

3.2 Distribution

Distributive methods are used when you have a large number of objects that you would like to place around an environment algorithmically. Examples of this can be the selective placement of specific types of Pokémon and the likelihood that they might appear in *Pokémon Go*, or the location of the stories and storytellers in *Where the Water Tastes Like Wine* [7], [10], [11].

3.3 Parametric

Parametric methods involve the manipulation of specific parameters of predefined objects. In a game like *No Man's Sky*, this could affect the characteristics of certain types of foliage, landscapes, or environment geometry. They can also be seen in music-based games that generate effects based on audio parameters that are manipulated by the player [10], [12].

3.4 Grammars

Grammars are generators that use recursive algorithms to *expand* predefined text based on a series of rules to create new and interesting combinations. It evokes the cut-up method used by Burroughs, and although this method uses text in its construction, the output does not necessarily need to be in words [13]. Several examples output SVG images, or ASCII art to great effect [10].

3.5 Constraint-Solvers

Constraint solving methods generate solutions from large sets of possibilities. Pathfinding, for example when you click on a destination in a strategy game like *StarCraft* and your units must find the best way to navigate around obstacles to reach it, is a form of constraint solving [14]. Procedural Generation can utilize these methods by generating all of the possibilities from an initial set, and selecting the best possible outcome based on a set of constraints. They can be used to create incredible variety in content, but also require a significant amount of computational power that increases exponentially with complexity [10].

3.6 Agents and Simulations

These are fascinating tools that come closest to replicating naturally occurring behavior. Genetic Algorithms, famously demonstrated in John Conway's *Game of Life*, extrapolate accurate evolutionary patterns from simulations produced from a defined initial state [15]. More recently, *Dwarf Fortress,* an indie management simulation game about dwarves building a fortress uses simulated agency in the dwarves themselves to create truly emergent behavior in fascinating ways [10], [16].

3.7 What are the Difficulties of Using PCG?

These methods are powerful, and have the ability to produce incredibly high numbers of possible variations, but they have their limitations. An issue that they all share is the considerable problem of creating uniqueness. Kate Compton describes this as the "10,000 bowls of oatmeal problem", where you can generate a mathematically significant number of bowls with individual oats in different locations or orientations that all appear to the viewer as a bowl of oatmeal [10]. In this case, using generative techniques doesn't result in any meaningful output that justifies their use. The real problem lies in creating *perceptual uniqueness*, or "the difference between being an actor being a face in a crowd scene and a character that is memorable" [10].

I like to think of it as similar to the old adage that if you put a million monkeys in a room with typewriters, and they could write for eternity, eventually they would create Shakespeare's Hamlet. Procedural Generation is significantly more structured, as computers aren't randomly typing characters until they eventually arrive at the expected result, but the problem is the same. How many plays did those million monkeys write that were nonsensical, or even worse, *boring*?

Returning to our *Rory's Story Cube* example, it would be much easier to write a program that generated the story prompts as images than to write a program that generates stories or *meaning* from the prompts themselves.

3.8 Example: The Difficulties of Developing with PCG in *No Man's Sky*

Figure 3 A procedurally generated alien on an unknown planet

No current conversation about the challenges of developing procedural content would be complete without discussing Hello Games' *No Man's Sky* [12]. It's a game about experiencing the infinite possibilities of space through exploration, and is a marvelous technological achievement by utilizing several of the previously discussed methods successfully. The universe is expanded from an initial seed, and through a series of parametric, distributive, and simulative algorithms that are constrained by a set of universal

rules to allow players to explore many of the generated galaxy's "18,446,744,073,709,551,616 unique planets" [17].

However, that focus on the sheer number of available possibilities did not demonstrate the uniqueness of each possible generation, and the overwhelming response from critics and players during its original launch indicated that the game was too repetitive and didn't deliver on what people expected, that every planet would be wholly unique. Much of this problem lies in public perception of what procedural generation is capable of, and that the potential variety of content is of greater value than the moment to moment experience in game.

4 Procedural Storytelling

4.1 How PCG Can be Used to Create Narratives

Hello Games has, worked tirelessly to address this over the two years since release. The addition of improved visuals, tailored variation in locations and tasks, multiplayer, and a narrative arc that ties the experience together by giving players a story to follow have largely addressed many misgivings from the original version. Introducing authorial control by producing a specific *narrative* from these generated elements is particularly effective because we as humans enjoy piecing together threads of stories to make them our own.

These are the stories we create where if we hadn't gotten off the train at a specific time, we never would have bumped into someone who gave us our next job, or introduced us to our future partner. In the case of *No Man's Sky* they might be tracked to specific encounters on certain planets that led to a mystery solved elsewhere, and that journey is entirely our own. And that is what makes this kind of linear building block storytelling so powerful, as it's quintessentially human. Therefore, providing a space for people to create their own stories through procedurally generated content was a key goal for *Campfire Tales.*

Doctor Chris Martens defines procedural narrative generation in her 2017 GDC talk with Rogelio Cardinal-Rivera as "any automatic process that creates a narrative over time, where that narrative is not determined before the process begins." [18]. As discussed in *Procedural Narrative Generation*, the accepted process for a simulation-centric approach requires an initial state with which to begin the story, a set of interaction rules, and a story generator. This method would allow designers to set up a rich world that could be observed by the player, and could produce story through the interaction of different elements from an initial state. For example, a game that has a squirrel that loves to find nuts in trees has just moved into an area with trees full of squirrel-eating monsters.

Alternatively, a plot-centric approach could be used to encourage player interaction with the world as well as greater authorship from the designer. This approach also requires an initial state and a set of interaction rules, but adds an interactive narrative that is produced using both a story generator and a story mediator to allow the player to proceed through a series of "nodes" or story "beats" that occur in response to player actions [18].

4.2 Example: Procedural Narrative Generation in Reigns: Her Majesty

Figures 4 & 5 Shuffling the deck and making decisions.

Reigns: Her Majesty is an excellent example of a game which utilizes these elements to drive a narrative forward [19]. In the game, players take the role of a queen ruling her kingdom by making small binary decisions on cards that appear each time the deck is shuffled. A player might choose to placate the church by not wearing the color red or decide whether or not to invest in a foolish endeavor to seek out a new world. These actions affect the levels of the relationships with the church, people, military, and coin, and when any of the levels are too high (or low) the player is killed and a new ruler is born to replace them (thus shuffling the deck).

While a story could be told about the individual lives of each ruler and the decisions they made that led to their untimely deaths, *Reigns: Her Majesty* does have an overarching narrative that is told through a series of beats. By utilizing a distributive method of PCG by adding weighted values to each card to ensure that certain cards appear when needed to move the connected narrative in the direction intended by the designers. This includes encountering specific characters once a certain number of rulers have died, as well as giving subtle instructions that help the player solve puzzles hidden throughout the game's world. For example, at one point the queen is given perfume through a coronation ceremony that the player must perform properly. This is an important item that is kept by all subsequent generations, and the encounter is weighted so that it occurs early on in the game, and then is changed such that it won't happen again once the player has the perfume.

5 Exhibiting PCG

5.1 Using PCG Outside the Game

Because Campfire Tales is intended to be experienced in a performative space, I needed to look beyond games that used PCG as pure content creation towards games that used said content to evoke specific responses from their players in the real world. Here are some examples of games that use procedural content generation to encourage players to act within or manipulate the spaces around them.

5.2 Experiencing Parametric PCG in Exhibition with *PANORAMICAL*

Figures 6 & 7 Visual output from play at Now Play This!

PANORAMICAL is a game that allows players to generate striking visual landscapes through audio manipulation [20]. By adjusting various parameters in the soundscape, players are given the chance to create and modify the on-screen visuals in real time in ways that map accordingly to their actions. In the home version, players can save screenshots of their unique landscapes, and can share them by connecting the application to their personal Twitter account.

PANORAMICAL has also been shown at exhibitions like *Now Play This!* in London, where players are able to control the landscape with others in person using a MIDI controller with a series of dials [21]. In this way, the experience becomes a form of collaborative performance as players connect to create landscapes together through quick improvised decisions and by controlling multiple parameters simultaneously to achieve the desired result.

5.3 Creating an Immersive Experience with *Pixel Fireplace*

Figure 8 Pixel Fireplace on display with 8-bit decor

Like *PANORAMICAL*, *Pixel Fireplace* is a game that benefits from the place and environment in which it is being played [22]. In it, players are tasked with keeping a pixelated fireplace ablaze by typing in the things that they need, like "log" or "match" to add logs or to start the fire. It's a relatively simple and relaxing experience when played on a computer at home, that is *transformed* when introduced to other surroundings that complement it. Some users have built their own fireplaces [23] to encase the game and give it additional meaning, whereas others have used old monitors and keyboards to evoke the era that the pixelated art hearkens back to [24]. In either case, they are digital reproductions of their analog originals, much in the same way I hoped to add a physical representation of a campfire to fundamentally change the space surrounding it.

6 Connecting the Dots

6.1 Why it is Important to Examine the Link Between Oral Tradition, Interactivity, and Procedural Generation

Written records of oral traditions are themselves a form of reproduction that raises questions of their own authenticity. Do they accurately represent the source material, and are they themselves changed by the new medium in which they exist?

When Walter Benjamin wrote in his 1936 paper, *The Work of Art in the Age of Mechanical Reproduction* that "technical reproduction can put the copy of the original into situations which would be out of reach for the original itself," he was advocating that there are unique qualities inherent in the reproduction of existing works either "in the form of a photograph or a phonograph record" that don't exist in their source materials [25]. Whereas his examples describing the ability for camera lenses to highlight features present that might be invisible to the naked eye and others are firmly rooted in technology that has been markedly changed over the past century, the sentiment is the same. In an age where technology is no longer just capable of reproducing art, but also *creating* it as well we need to reevaluate our assumptions.

When we consider Campfire Tales as an experiment in technically reproducing an art form, rather than an individual piece, things become much more interesting. By having a computer act as both the orator and mediator while stories are told, they are also *recorded*, and though they may change with repeated tellings, a record of each can be kept. This is in direct contrast with the ephemeral nature of oral traditions throughout human history as their original messages are lost in the telling [1].

7 Development Process - Research

7.1 Introduction

I set out to develop *Campfire Tales* by focusing on three major components of the experience: a central program that utilizes procedurally generated content to tell stories with audience interaction, optional room-scale environmental effects driven by the central program to facilitate new types of generative expressions, and introducing performers to direct the player experience in a public space. To reach as wide an audience as possible, I planned to release *Campfire Tales* on the web so that people could interact with, save, and share their own stories while being encouraged to act

them out live with their friends in a similar way to the version being publicly exhibited. Through repeated play, we create our own digital oral traditions as players, performers, and storytellers.

7.1 Why Procedural Generation?

Because the player interaction that I want to provide generally consists of a single word during transitions between story beats, and that kind of input can be replicated in code by saving player text input as a variable, I need a way to tell stories that include the possibility for improvisation as a result of that user interaction in order for it to feel meaningful. To do so, I would need to find the right tool that would allow me to generate text that could exist as a part of a larger narrative, respond to user interaction in a meaningful way, and allow me to use the content generated to drive a microcontroller or other external electronic hardware component.

7.2 Improv

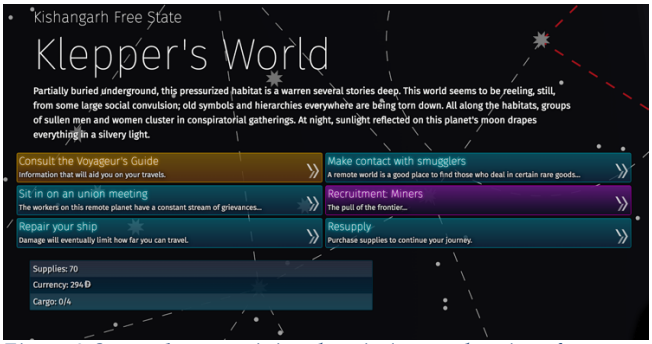


Figure 9 Screenshot containing descriptions and options for new worlds

Improv is a "model-backed generative text grammar tool for JavaScript" by Bruno Dias that "can generate random, procedurally generated text recursively" [26]. It was used to create the details for the randomly generated elements (like descriptions for planets, characters, pathways, and events) in his 2017 game Voyageur [27]. Improv works by creating JavaScript grammar objects from a possibility space with a deeply nested structure consisting of phrases and objects that can be tagged or filtered. A recursive function will read through this structure (which contains a set of terminal and non-terminal values) to return a generated text string that can used in the game.

What sets Improv apart from other solutions is the ability to create a model, or "object that holds data about the text," through the reincorporation of tags that can be filtered to ensure more predictable or authored generation. In his PROCJAM tutorial, Bruno provides an example that would allow developers to generate types of ships that are named specifically because they are either military or civilian [28].

7.3 Rant

From the Rant website: "Rant is a free, all-purpose procedural text generation language developed for .NET. It has been refined over several years of development to include a dizzying array of features designed to handle everything from the most rudimentary of string generation tasks to dynamic game dialogue generation, code templating, automatic formatting, poetry, and much more" [29].

At first glance, Rant appears to have quite a lot of the functionality for text generation that I'm looking for, as well as documentation specifically for writers in established projects. At the time of this writing, I had issues integrating Rant into the latest version of Unity, and the .NET platform doesn't have the same kind of flexibility with external components that I'm looking for.

7.4 Tracery

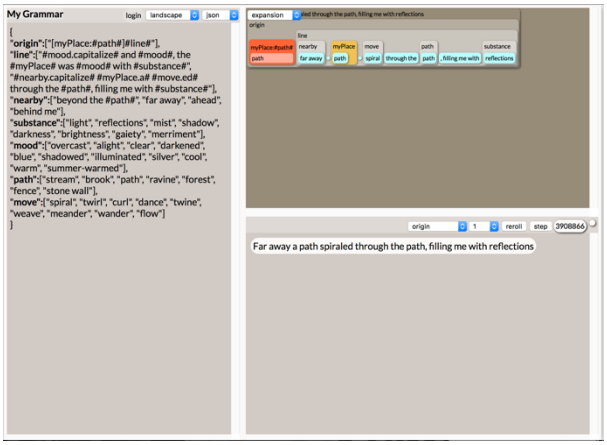


Figure 10 The web editor at tracery.io for rapidly prototyping new grammars

From the Tracery website: "Tracery is a super-simple tool and language to generate text, by GalaxyKate. It's been used by middle school students, humanities professors, indie game developers, professional bot makers, and lots of regular people, too. Give it a try today!" [30]

Tracery, like Improv, uses a recursive function to generate text from an authored set of possibilities. While it lacks native support for data modeling and filtering, it is extremely well documented and has been used in a number of projects that served as an inspiration for this game. It has also been ported to other languages (such as ruby and python) and there is even an integration into Twine!

7.5 RitaJS

From the RitaJS website: "Designed to support the creation of new works of computational literature, the RiTa† library provides tools for artists and writers working with natural language in programmable media. The library is designed to be simple while still enabling a range of powerful features, from grammar and Markov-based generation to text-mining, to feature-analysis (part-of-speech, phonemes, stresses, etc.). All RiTa functions are heuristic and do not require training data, thus making the core library quite compact. RiTa can also be integrated with its own user-customisable lexicon, or with the WordNet database. RiTa is implemented in both Java and JavaScript, is free/libre and open-source, and runs in a number of popular programming environments including Android, Processing, Node, and p5.js." [31]

At first glance, RitaJS appears to have all the functionality that I need to get things up and running, with the added bonus of having feature analysis to allow me to parse and

modify generated text based on English grammar structures. I used RitaJS during my research to create a basic Mad Libs Generator that replaces specific particles of speech with words selected from its built-in lexicon.

7.6 Deciding on Which Platforms to Use

Ultimately, I settled on using Tracery as the library for grammar generation based on the simplified nature of its grammar structure. Whereas Improv has the ability to filter text so that I would be less likely to create impossible scenarios, the fact that objects within its grammar generation require tagging meant that it would be a nightmare for users to interact with the program. I didn't want people to have to type more than one thing per interaction, and it would be especially disruptive to live performances if people had to wait while someone typed in "animal" and "extinct" every time that they wanted to enter the word "dinosaur." Tracery was powerful enough for my purposes, and allowed me to use a JSON data structure to create and save modified grammars as people played.

I began working on *Campfire Tales* using JavaScript, Tracery, Processing, RitaJS, and NodeJS so that I could extend and update the project as needed during development with additional packages for functionality. This environment allows for deployment to the web to reach a greater potential audience than I likely could using Unity (which meant that Rant was unsuitable). Additionally, I could use Processing for visual effects and for serial communication to drive an external electronic component (like an Arduino for example) and RitaJS to parse text intelligently if I wanted to implement changes to already generated stories over time (like replacing key words with synonyms from the built-in lexicon to create a sense of evolutionary growth).

8 Prototyping

8.1 Twitter Bot @CampfireTale

As I'm fairly new to JavaScript development, I decided to make a Twitter Bot to both learn how to build a project using NodeJS and interact with an external API as well as find a place to test generative text in tweet-sized chunks and user interactions. I followed the excellent video tutorial by Daniel Shiffman of "The Coding Train" to set up basic tweet interactions while the program is running [32]. To do so, I had to apply for a Twitter development account. For now, @CampfireTale only has a few basic test posts that included running a generated image, and an automatic response [33]. Going forward, I plan to use the account with the Tracery based Twitter Bot generator "Cheap Bots, Done Quick!" to quickly iterate my grammar design before attempting to integrate the Tracery library directly into my existing code [34]. This way, I can continue to iterate without having to spend valuable development time implementing my own solution in code that doesn't necessarily fit into the core product.

8.2 Processing, Tracery, and Node Prototype with User Interaction

My first completed technical prototype started off as an exercise in getting Improv functionality working with P5js, where I could output the text generated by Improv to the screen using Processing. I started by following the example in the Improv Tutorial to generate text to a terminal window, and then began to write a sample program that generated a variable from an Improv object in a Javascript program [28]. I was able to get it running using a local simple HTTP server in python to test results, but it became really inefficient to set up every time that I wanted to make changes and see the results in the browser console.

By using the node package "budō" I was able to package all the local files and serve them locally for testing, and should include functionality for packaging them up for deployment in a live environment [35]. I used Processing to display text to the screen, and to add a text input box for players. For usability, I didn't want to have more than one point of entry on the screen at any given time. This is when I realized that Improv may not be the right tool for the job, given the need for tagging objects that are added to the grammar's structure.

I then decided to test with the node package for Tracery (tracery-grammar), which requires including jQuery as well in order to function [36]. Here, I was able to push inputted values to the array of values for characters in the story, thus increasing the possibility space for potential characters with each new entry. Each time the page is refreshed, the short story is regenerated from all of the values contained within the grammar.

It was a very basic interaction that demonstrated the functionality that I was hoping would exist at the core of the experience. However, I still have some major problems to solve in future iterations. I can use the performer to filter entry at the exhibition, but there is no easy way to filter language (though this may be a good use for implementing RitaJS), and I'm not certain that I should restrict the language for any personal copy of the game. I also need to find a way to connect the generated grammars in such a way that I can create a narrative, and track inputted variables for the course of an individual story.

8.3 Web Hosting with GitHub Pages

My eventual goal for *Campfire Tales* is to have it widely available on the web. Once I had completed my initial technical prototype, I needed a way to publish it for people to access it. My first test was to create a sketch in processing that generated text on a curve over a static image and deployed the sketch to GitHub and published it as a GitHub page [37]. The sketch was visible, but had issues displaying across a wide variety of devices. I collected a screenshot of the sketch, and adjusted it to deploy a static teaser website at campfiretales.info, where the final version of the game would be released and take the place of the screenshot in the canvas.

8.4 Tuya Smart Light Integration

My original intention was to use Processing to send data over a serial port to an Arduino or other microcontroller

for environmental lighting or other effects. While I still may use both for mechanical interactions, I decided to explore smart products for environmental lighting. This was both to learn something new, and as a forward-thinking move, as it could be kept and implemented into future iterations to allow users with smart light bulbs to experience some of the designed lighting effects from the exhibition version of *Campfire Tales*.

To keep costs down, my first test was performed using a generic Tuya compatible smart light. One of the major advantages was the availability of a node package for controlling Tuya devices locally [38]. Getting the API to function properly within my main program was significantly more difficult. Once I retrieved specific values by decoding HTTPS traffic for a local key, local IP address and device ID for my test bulb, I did manage to get the test program to turn the light on and off when a key was pressed. Unfortunately, there appears to be no documentation for writing color values to the bulb (while there is a dataset that appears to contain values, they refer to a state that is locked behind developer access to the API, which at the time of this writing I haven't received).

Color changing appears to be possible by setting up an open source controller called "Homebridge," but it seems to make less sense to continue developing for a platform that much less people would have access to, and requires a significant amount of user setup that would significantly reduce my ability to share these features with others in the future. Additionally, there is a likely constraint that I won't be able to access external internet from the venue that this will be installed in, so using a platform that requires access to an external source for setup and control isn't suitable. I still plan to use colored lighting for the room, and more specifically for the centerpiece campfire that I would like the program to control.

8.5 Fire Effect Prototype

The original inspiration for this effect was a YouTube video by the Daniels Wood Land Show detailing how to make an "Insane Fake Fire Special Effect" using water vapor created by a sonic ionizer, halogen lights, a Tupperware container, and an old paint can [39]. Because I'll be setting *Campfire Tales* up in an indoor environment with limited ventilation, it's critical that I make something that is both safe and not actually fire.

I need to build something similar to the effect shown in the previous video and scale it up to become a centerpiece for the exhibition display. To do so, my first prototype includes an ultrasonic mist maker, a plastic shoe box, a plastic bin that I cut holes into, and two portable fans. The mist maker is essentially a piezo vibrating at a very specific frequency that causes the water to split into microscopic droplets that form a mist.

Initial tests with the mist show that it works well at generating a large volume of airborne particles, but they tend to sit in a layer about 2 to 3 inches above the water level. To introduce movement, I added the two portable fans for testing, and held the light above the lip of the bin. The effect is close, but will require a significant amount of additional development to get working right. The plan is to find a way to "chamber" the mist so that it could be blown (potentially from multiple sources) into an opening that will be covered by burnt wood to give the impression of a campfire. I'll be adding cade and birch tar essential oils to the mixture to give the impression of a smoky, burning wood smell, and can build a stone circle to hide the functioning electronics and protect them from wayward visitors. Lastly, I'll be able to integrate one of the smart light solutions that I'm working on for environmental lighting to give the campfire an otherworldly flavor with unnatural colors.

9 Iterating Prototypes Based on Feedback

9.1 Research Methodology

Campfire Tales was built on a foundation of iterative design, and therefore required a significant amount of player feedback to inform and validate design choices and the project's effectiveness in completing research goals. This was done with several rounds of feedback during its development using a mixed-method approach to gather information from peers, friends and family, professionals, and through a small set of public tests before release.

Qualitative feedback was gathered from my peers in the form of interviews and recorded responses during our weekly development and degree show planning meetings over the summer. Once development reached a playable stage, a google form was created that is linked to the completion of the game that allows testers to input feedback in the form of short answer survey questions and metrics based on a Likert scale to determine their response to various core features in the game (See Appendix A for questions and results).

9.2 Feedback and Iteration

The first round of peer feedback on my original prototype raised significant concerns about what it was that made *Campfire Tales* special. In its earliest form, the user interaction doesn't extend beyond an individual page. And while I had managed to implement my own version of a model using JSON, my classmates were genuinely concerned that I wouldn't be able to deliver a compelling experience with a series of disconnected paragraphs. They were, however, excited by the possibility of generating special effects with the results of these generated sections.

At this point I realized that I really needed to double down on the development of the actual story generation itself, and to determine how to make the experience itself enjoyable. I started by researching aspects of oral traditions in performance, interactive dramaturgies, design for procedural narratives, and reading hundreds of campfire stories to learn what makes them work. I came across a fundamental structure for building these narratives as a series of beats that connect nodes in a possibility space.

My writing process was to create a set of possible events on index cards, and to map out how a player could get from start to finish while still allowing for the entry of their own characters. I would write short paragraphs with blanks (for generative text additions like adjectives) and special notes for saved variables that the player will have

or should have named, and planned to use these when designing Tracery "grammars" in future iterations of the game.

I then diagrammed the entirety of how I imagined the user experience would be, to make sure that I had a clearer idea of all of the various game states and features that I would need to implement during development.

9.3 Game Flow Diagram

Figure 11 Game Flow Diagram

My original vision for the entirety of the game works like this: When a player first starts the game, they would see a title screen indicating them to click to start the game (as well as introducing clicking as a valid interaction). They would then see a secondary title screen with three buttons to start a new story, load a saved story, or reach an options menu.

If the player chose to go to the options menu they could find a set of instructions, load a saved world model that included things that had been added by other players, and see a list of game credits (that for now includes myself and the creator of the public domain stock background image that I am using for the game).

If the player chose to start a new story, they would be given a series of screens with no more text than you would see on a teleprompter on each (so they could easily be read by a performer). Each screen would either give the option to click to continue or to enter some text that could be added to the game world and reincorporated into the story. When the story ends, the player is then given the option to save the story to replay it later or to restart.

If the player chose to *load* a saved story, either created themselves or someone else playing the game, the experience is similar to that of a new story except that players would have the option to add new elements or keep them the way that they were. This would allow players to keep the same structure as a favorite story, such as a sequence of events that they really enjoyed, but change certain elements to create a slightly different version that changes with the telling. This is much like the ritual action and gradual change of stories that I found occurred during my research on storytelling in oral traditions.

With this vision in mind, I set out to develop my second functional prototype of the game.

9.4 Saving Variables into Existing Grammars

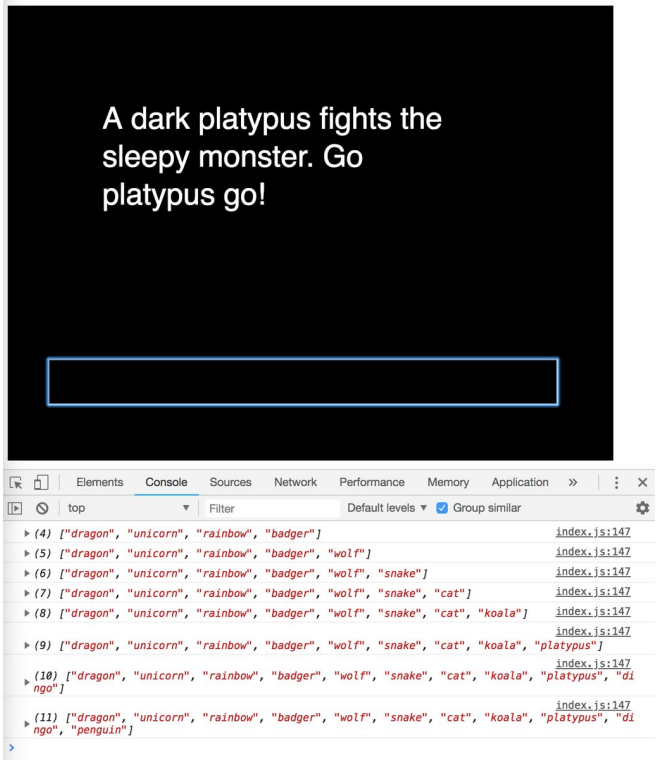


Figure 12 Grammar Manipulation Prototype

As mentioned previously, Tracery uses a recursive function to expand a data structure called a *grammar* to generate text. Generally, this allows the programmer to loop through text and save an individual result from multiple possibilities to generate different outputs each time the function is run using the same grammar as a source. This is done by creating a series of rules within the grammar for Tracery to follow. For example, if we wanted to write a very short story with a small amount of variation, we could create something like this:

```
var grammar = {
     story: "#beginning# there was an #adj# #character#",
     beginning: ["Once upon a time", "It was a dark and stormy night, and", "A long time ago, in a galaxy far, far away"],
     adj: ["green", "yellow", "gigantic", "evil"],
     character: ["squirrel", "scientist", "unicorn"]
}
```

Expanding this grammar with a starting point uses Tracery's functions to select an item from each rule that it encounters with the '#' character. In this case, by starting with the story rule, Tracery would select a beginning then an adjective and a character and output one possible combination of these elements. This might result in our story being "Once upon a time there was an evil squirrel."

What if we wanted to save a selection from one of the rules and keep it for the remainder of the generation? We could change the "story" rule to look like this:

```
story: "#beginning# there was an #adj# [hero:#character#]. #newSentence#"
```

And then add a new rule that calls back our hero character in a new sentence, like this:

```
newSentence: “The #hero# was awesome”
```

One possible outcome of this new grammar might be “It was a dark and stormy night, and there was an gigantic unicorn. The unicorn was awesome.”

What this *doesn't* do is save the variable between multiple calls to expand the grammar. Nor does it allow us to incorporate new values (such as text entry or defined values) into the structure for further use. This is where we can use a combination of JavaScript variables and Tracery's pushRules function to address this problem. To make my second prototype, I created a very simple short story with very little variation to test this functionality. In it, a person (whose name is generated by Tracery) meets a new friend (defined by the player) and they go to a place (also defined by the player) and have fun!

To make this work, I would need to run Tracery's algorithms each time there was something new added to the story. To save a permanent addition to the grammar (so that the generated name is consistent for the remainder of the program) I wrote the grammar to look like this:

```
var grammar = {
     names: [“Molly”, “Billy”, “Yosuke”, “Amanda”]
}
var story = tracery.createGrammar(grammar);
var hero = story.flatten(#names#);
story.pushRules(“hero”, [hero]);
```

This code takes a grammar structure with one rule composed of a series of names that when “flattened” selects one of the names in the list and saves it to the variable “hero.” That variable is then pushed as a new rule to the grammar called “hero” and containing only one possibility (the previously saved name), thus making sure that the value is consistent for all future expansions.

This same method can be used when saving player input from a text box to push new rules to the grammar that can be called later. And because we can define the name ourselves, we can write rules that anticipate these rules being pushed later, even if they don't exist when the program is first run. For example, I could use the same rule from our previous example

```
newSentence: “The #hero# was awesome”
```

which would result in a sentence like “The Amanda was awesome” once the algorithm is run using the newSentence rule.

Where my earlier prototype had values added directly to the arrays in the grammar themselves, and were saved for future iterations. Players were essentially adding words to an ever-increasing list of monster names, and with each new addition they were decreasing the possibility that any of these would be recalled in future stories. While it's great to have variety, it made it more difficult for the player to feel like they were having a direct influence on the story itself. This newer structure felt much more dynamic, and served as the basis for the playable and final versions of the game.

Lastly, I added a title screen and option to save the story as an image from the flow diagram I had made previously to test the player experience from start to finish. Doing so made me realize that I didn't necessarily need to give players the option to share the actual data of their grammars with each other, nor did I need most of the additional screens and options from my flow diagram to create a compelling experience.

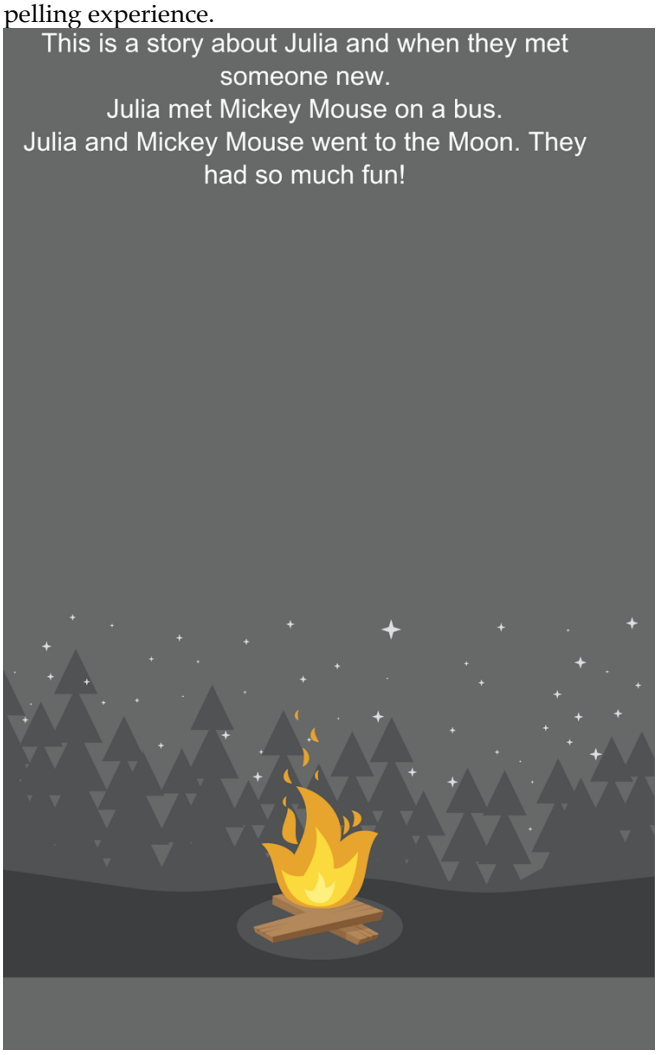


Figure 12 Example Story Poster

9.5 Player Feedback

Peer feedback received in class and a limited number of responses to a google feedback form were instrumental in helping me develop the first fully playable version of the game. Players indicated that they liked the idea of being able to print their stories at the end of the game, but wouldn't be interested in keeping or sharing it at this time. Multiple responses asked for changes to the font, animated fire, and a longer story. One player enjoyed “being able to choose the characters and destination but nothing really happened when they got there” (See Appendix A). These responses gave me three clear goals for developing the next iteration of the game: to improve the visuals by adjusting text appearance and adding fire, to create more a substantial narrative with greater variation, and to make sure those stories are impacted by the input from the player.

10 Playable prototypes

10.1 Developing the First Playable Version

I began development on the first playable version by creating scene management functions that would allow me to keep track of content created for longer stories, and allow the user to progress through either mouse clicks or text entry depending on the scene. This would allow me to make sure that something "happens" as a result of the story progressing, and ideally as a result of the elements introduced by player text input. My first test was to create a series of generated scenes that were a mix of random questions and unrelated sentences. The questions were framed as someone who is sitting with you at a fire and wanting to get to know you better. Like the original prototype, the answers weren't yet used in an interesting way beyond being collected and displayed as a talking point for testing.

However, because these questions were ones that I considered keeping for the final version and I wanted to see how they worked in context, I brought them to a group of professional developers for a bit of improvisational feedback. One of the elements I was trying to gather for the story was a name, and an example question asked players to name "a friend that they grew up with." The response from the group was overwhelmingly negative, because when someone is giving something to the story with an explicit personal attachment (like a friend's name) they will expect that the story's representation matches their own experiences. For example, if I were to say my mother's name to a storyteller, and they incorporated it into the tale they were weaving, it would be jarring to hear her do something completely out of character.

We found that when asking for a name that someone likes or dislikes, they are able to define something that they have a personal connection with but not any clearly defined expectations for. They are being asked for the name, and not the *person*. Using this frame of reference, I rewrote my questions for the audience to be less explicitly personal, and began brainstorming potential narratives in the game.

My next step before adding complete stories was to improve the audiovisual aspects of the game. After reducing the text size and changing the font, I found an excellent open source fire animation in processing by Julien of Kampeki Factory to use as the basis for the background of each text-based scene [40]. By converting the code to my node instance of P5.js and drawing the output to an offscreen buffer, I could position the fire wherever I wanted on screen with minimal impact to performance. I reduced the buffer size while keeping the same aspect ratio to improve speed on mobile devices and further enhance the haziness of the image to give the appearance of heat.

To make the effect better, I added campfire audio, and three additional background tracks that loop to give the impression that you are in a forest at night [41], [42], [43], [44]. Because all of these assets are preloaded, I added a loading animation to replace the default P5.js text that would appear when the webpage is first opened [45].

10.2 Iterating the Environmental Effects

I continued developing the fire effect by using a larger plastic storage bin than the one from the first technical prototype, and laser cut an acrylic cover that could be sealed properly to prevent mist leaking. Adding additional atomizers increased the volume of mist being produced so that it became noticeable at higher altitudes when being blown by fans. Testing the effect with various configurations of low and high-speed USB fans showed that the high-speed fans dissipated the mist too thinly to give the appearance of fire. The low speed fans worked better, but had to be placed in very particular angles to create the airflow circulation inside of the bin to push the mist out in a fire-like pattern.

Putting a colored light in the mist, and covering the opening with small logs gave the appearance of a smoldering wood oven, but not that of a roaring campfire. It seemed like the effect was not going to work the way I intended, and I nearly gave up when I realized that *blowing* into the opening worked much better than the fans I had been using. Even better, using your breath as if you were starting the fire felt like a natural interaction, and one that I could use both as a part of the performance and in the game.

10.3 Combining the Two

Discovering the blowing mechanic led to a major shift in the design of the experience as a whole. Given that there was limited development time remaining, I needed to make a choice, to either keep working on environmental lighting that is driven by the game or to double down on this new method of interaction. I decided that it would be much more interesting to give players a physical fire that they could play with by trying to keep it alight, than it would be to generate different colors while playing the game.

To do so, I began by testing my assumptions with the web interface for the game. I added a scaling method to the fire animation in the game where it would stretch vertically over time and give the appearance that it was dying out. And when players breathe into their microphones, which gives audio input over a specific threshold, the scaling would reduce and appear as if it was coming back to life.

Originally, I had the program test for a sustained input of twenty seconds before returning to the original scale, but after some players had issues noticing whether or not it was working, I changed the function so that the scale is adjusted whenever microphone input is received over a certain point. Because the fire is constantly receding, outlier input like ambient room noise or conversations that intermittently exceed the threshold didn't result in noticeable change. The result ends up being an animated reaction to the player's breath that feels responsive.

10.4 Philips Hue Prototyping

The next step to developing the physical portion of the installation was to set up functionality with Philips Hue lights to react to the breath mechanic introduced in the most recent version. This could be done by setting the brightness level of the lights to match a counter that is increased when inputted volume is over a certain level. Compared to the previous smart lights that I had been using, the Hue lighting system is much better documented and

has a fully featured npm package for controlling the lights in a connected node JS application. This would however work much better within a controlled local environment, where you can establish the hardware addresses and user credentials for controlling the lights without having to build a user interface to do the process for you [46].

To get the lights to work, I built a separate application to run a local http server that facilitates communication between the Hue lights and a second JavaScript that collects microphone information and determines how the lights should change. My initial tests with just a counter being updated on screen indicated that the message and response system of standard HTTP functioned much too slowly to feel natural, so I decided to rewrite the server using websockets [47]. This was near instantaneous and allowed me to write new functions that would have much quicker responses.

The first test used the node-hue-api function to turn on the light when the counter reached 100. It would then stay on for a period of time before returning to the off state to represent the fire either being lit or not lit. Next, I mapped the brightness of the Hue lightbulb to the value of the counter, but noticed a pretty significant delay in the update to the bulb from the bridge.

To give the player a way to tell whether they had an effect on the fire quicker, I added two audio tracks. One base layer fire crackling, and a second roaring bonfire sound that would increase in volume along with the counter [41], [48]. Because the audio doesn't have to communicate further with the light system, the feedback is immediate. Combined they both give the impression that you are lighting a flame when blowing on the microphone, with an added unintentional flickering effect from the delay in changing brightness values.

Lastly, I added color to the lighting system by creating a function that sets the default color during startup, and changes it rapidly when the brightness is full, giving additional response for the player and a fun interaction.

10.5 Interpreting Feedback from the First Playable Version

Player feedback from both new and repeat play testers enjoyed the response from the addition of the fire animation and sound. Additionally, the overall scores from the Likert scale results began to demonstrate an upward positive trend between feedback rounds. This helped me realize that while I was moving in the right direction, even if I still had quite a lot of work to do.

While observing players try to blow into their microphones, I noticed that many players couldn't easily find the location on their laptop where they needed to blow and some became frustrated trying to set things up. Also, while I had become very comfortable with the long slow breaths that I had been using to trigger the game start, some people tried short powerful breaths that didn't quite trigger the opening.

Nearly every player wanted longer stories, and better text placement on the screen. Some wanted the ability to see text from previous sections, and one player suggested colorizing the final screen text to showcase the generated words or sentences in different colors. Many players pointed out that it was difficult to see the continue button during play, and wanted a way to continue with keyboard input rather than clicking during each screen.

Additionally, while players enjoyed the audio, some found the frequency of high pitched sounds (like crickets) in the background occurred too often, and wanted a less distracting soundscape.

11 Final Playable Versions in Performance

11.1 Developing the Final Version

I started addressing these issues immediately during the development of the final version. First, by decreasing the amount of microphone input needed to start the fire during the opening screen, by adding a function to skip the scene if microphones weren't detected, and by adding a continue button to skip the scene if they weren't able to get it to work correctly.

Next, I began to address the requests for a longer story, by starting to write a short story that followed some of the necessary qualities that I determined during my research. The game begins with a storyteller inviting you to share their true tale over a fire, and invites you to help them make a story with them by giving names to protect innocents in the story to give the audience ownership. After an accidental, and presumed death, the true nature of the storyteller is revealed, and the player is confronted with the story's twist.

I moved the text so that it was centered beginning at a quarter of the screen height, and reduced the size of the fire on the screen to give more focus to the words. I also changed the soundscape of the game by replacing one of the background audio tracks with a more sparsely populated biosphere and implemented a function that allows me to select specific tracks to play or pause as a group (rather than the default audio functions in processing).

I also ran one final round of user testing to gather feedback on the changes, and test what the experience was like when vocalized. One of the shyer testers tried acting the story out loud, and felt empowered by the experience, as it gave them the ability to improvise in tandem. This was reassuring, as it was my hope that it would encourage others to try when the game is demonstrated and performed by others during the *Playful Experiences* Exhibition.

11.2 Organizing *Playful Experiences*

One of the core development goals for Campfire Tales has always been to find a way to use the game in an exhibition space. This also meant that I needed a way to display it publicly to demonstrate the performative and hardware features that I have spent so much time making. Concurrent with my work researching and developing the game, I have acted as student organizer for the *Playful Experiences* show that it will be seen in first.

Throughout the summer, I held weekly planning meetings with classmates, collaborated remotely with those who couldn't be present, delegated tasks to each team member that fit their desired interests, facilitated communication between task teams, provided clearly defined

goals and timelines for the process, managed each team member by keeping track of progress and provided advice and assistance when needed, helped design a website to showcase our work online using GitHub Pages, and handled a number of administrative tasks for making sure that the show runs smoothly [49].

This gave me an opportunity to support the space that I used to build the final version of my fire effect. To do so I combined the previous setup I created with the original lighting prototype with the functioning setup from the Philips Hue system. I then attached the power system underneath the underside of a display table, which I experimented with at different heights, and found that a slightly higher table was safer, and had less chance of being knocked over without being seen if it was prominently displayed.

I ran the local websocket server and application from a Raspberry Pi using a USB lapel microphone for input, and a rounded white Bluetooth speaker that looked like a marshmallow to output sound within the context of the environment. I sealed the plastic compartment and covered it with a black drop cloth, and added a physical campfire built from fire logs and gathered sticks suspended by hot glue. I experimented with using reptile terrarium heat lamps to provide a source of heat during the show, and plan to keep them operational so long as they are supervised (as there is heat being generated directly beneath dry twigs, I didn't want to start an *actual* fire). Lastly, I added a green shaggy carpet to give the impression of grass, and to provide a seating area for audiences, and set up a portable projector on a telescopic stand to display starry skies.

12 Future Development

12.1 What Worked, What Didn't, and Where Do We Go from Here?

At the time of this writing, *Campfire Tales* has yet to be seen in public. My plan is to use feedback from our final degree exhibition to inform the development for future versions, and to comment on whether or not I succeeded in reaching my objective with a much wider audience. I'm extraordinarily excited to see the response to both the things that I felt worked well, and those that I would like to improve further.

Firstly, I feel like I have been very successful in my learning objectives for the project. When I started, I had extremely limited experience with JavaScript, and beyond that, less than a year's worth of programming experience period. Taking on a project and making the decision to not only use full-stack JavaScript development as a platform, but to also *learn* how to use it, integrate it with experimental hardware techniques, and still complete the project at a postgraduate level was a major achievement for me.

With that said, if given the choice to revisit the project again, I wouldn't use Processing as the main tool for creating the game. It was an easy to use JavaScript library that factored into my current knowledge (I had some familiarity in converting some processing sketches to C++ as an exercise in one of my taught modules) but it didn't work well with a project based in nodeJS. A few game focused development environments that use JavaScript and nodeJS that would be much better options would be CreateJS, PixiJS, and Phaser.

Additionally, while I am excited to have had the opportunity to push myself to experiment with smart lighting as an option, I feel that I might have better control over more complex systems of local effects (including lighting or mechanical) using a system of wirelessly connected microcontrollers. I would love to see how a version of this experience might be iterated upon further by focusing on the strengths of the experience as an installation. On the flipside, laying the groundwork for the smart lights was in the hope of potentially exploring a version of this game in the future that runs on the Amazon Alexa platform, entirely using microphone input, and could interact with the personal setups of the players at home.

And while I am happy with the current structure of the story that has been created, there is a portion of me that also feels that by choosing to focus specifically on campfire stories, I haven't yet reached the type of limitless imagination inspired by the alphabet stories I told when I was younger. I would love to create more variety within the stories themselves, to further integrate player choice, and to improve people's ability to share their stories beyond a printable image.

I could do this by adding an online database service, like Google's Firebase, to the game as a way to collect the stories created by people, and to use the Twitter Bot that I created early on in the development process as a method of sharing them. Over time, as the stories become more complex, there could be a genuine record of how they change, and would be an excellent way to illustrate the ritualistic repetition through play that evokes the oral traditions that inspired this project.

12.2 Final Thoughts

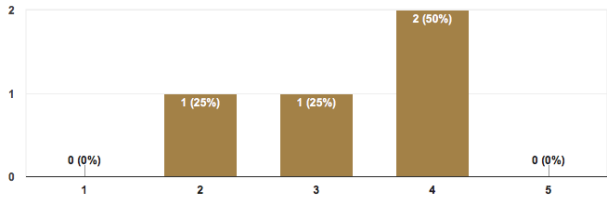
In conclusion, I am extremely happy with the current result of *Campfire Tales* as the start of a greater ambition in combining generative storytelling and immersive theatre. It manages to provide the illusion of improvisational interaction with a storyteller, and demonstrates how stories can be changed through repeated tellings with a written record. Mostly, I feel as though it has contributed in its own small way by demonstrating a new method of using generative techniques in combination with real world spaces, and might be used to help people learn to embrace their own innate storyteller through improvisational techniques that can be taught through software of this type.

APPENDIX A: USER TESTING RESPONSES

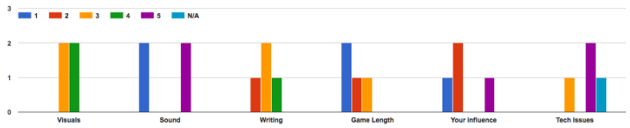
Prototype Feedback

Overall, how much did you enjoy playing Campfire Tales?

4 responses



How satisfied were you with these elements in Campfire Tales?



Optional: Please describe why you chose the scores above

4 responses

The whole experience was really short. I did enjoy being able to choose the characters and destination but nothing really happened when they got there.
its short, visuals could be animated, writing is really basic, i didnt have any song
I would've liked more interaction and for my choices to affect the gameplay a little more, also I would've liked it to be a bit longer.
The game is too short in its current state and i didnt feel like i could influence the story. The sound is really good, and the fire visual works well.

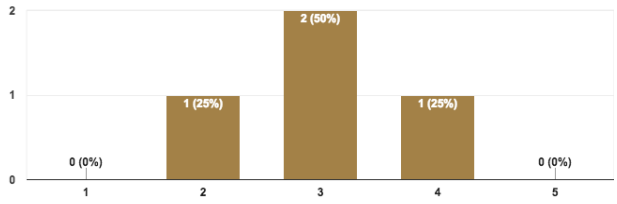
Optional: If you encountered any technical issues, can you please describe them here?

2 responses

no
No technical issues

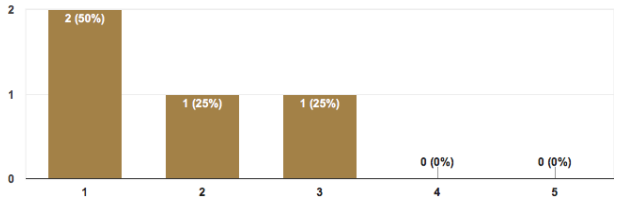
I enjoyed the story that I made

4 responses



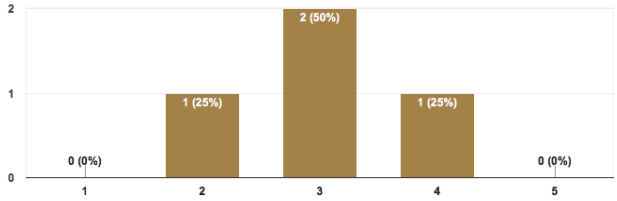
I would like to share the story I made with others

4 responses



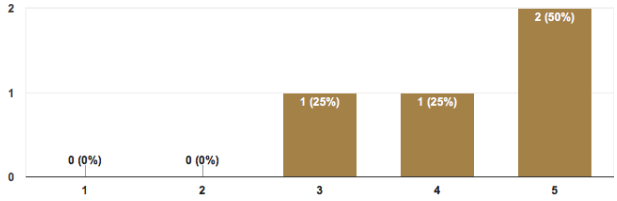
I would recommend Campfire Tales to friends

4 responses



I plan to play Campfire Tales again

4 responses



If you could add anything to the game, what would it be?

4 responses

More options, a longer story.
longer strings of text, less general stories, maybe animated fire
More visuals
Maybe custom buttons clickables for next or save

What would you remove from the game?

4 responses

NA (2)
I dont think anything needs to be removed.
Perhaps make the loading gif., it looked quite pixelated , but not sure it is on purpose

What would you like to change?

4 responses

Perhaps the font?
the font should be changed, less bold
Make the game longer.
I think the text needs bleed from sides. and top, it is too close to the edges of the screen. Seeing the last text when you get a new text could be cool too.

How would you define a "campfire story"

4 responses

- A scary story that borders on the unbelievable
- I assume(mainly from films) its a scary story that is told in the woods mostly common among teens to scare each other out.
- Fun and nostalgic
- A scary story told in night in a camping trip, next to a camp fire (All my knowledge is from movies)

Anything else?

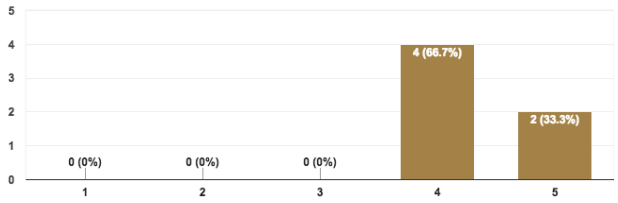
4 responses

- Looking forward to future iterations.
- It has potential, only it looks like a early prototype atm
- You're doing great, I love the concept!
- I like the sounds a lot, the fire animation is cool too, but maybe pixel density could be increased a little bit. Having low pixels is a cool abstraction, and maybe using a font like pixelite,(https://www.dafont.com/pixelite.font) can fit into the general aesthetic very well. If the fire is left like this retro style. Another font that might fit this game is, (https://www.dafont.com/dk-lemon-yellow-sun.font) DK yellow sun. This font is a little bit like life is strange style. It goes good with hand drawn doodles. I feel like it could fit into a campfire story. I am unsure too, arial, calibri, roboto, or helvetica is a safer choice, they are generally good on web, they are really good in a traditional web page, or if they are condensed they are really good as subtitles. I see that you plan to use text on image, so putting a black shadow on white text, and making it a little bit condensed (not too much, no 2 letters should touch) could work better. I hope this help, it is okay the way it is as well, good luck with the project. :)

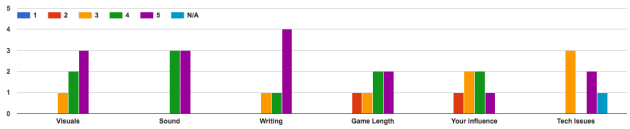
Playable Version Feedback

Overall, how much did you enjoy playing Campfire Tales?

6 responses



How satisfied were you with these elements in Campfire Tales?



Optional: Please describe why you chose the scores above

6 responses

- The overall game is excellent. Reminds me of short story adventures. Would like more input from player
- I loved the sound, especially with headphones. I'm curious if the narrative has many paths that it can run through? It's nice to be able to choose names but it would be interesting to steer the path of the narrative in a similar way. Substituting names feels like it could work better with a group.
- I was pretty immersed with the sound of the campfire so I enjoyed that. It could be even better if there were tense sounds during scary moments, and silence during rest moments. I didn't think anything that I wrote actually had an impact, I never saw any data coming back. The story ended so suddenly which left me confused even though it was interesting.
- The immersion from the campfire visuals and sounds really added a lot to the atmosphere. I quite liked that a lot. The writing was also really engaging and I wanted to know how the story ended. In the current state, I did not see where the names I entered had an influence on the game.
- This game is very fun and really had me engaged. It really scared me too! Perfect campfire tale.
- Text placement and colours could be adjusted to make more ledgible

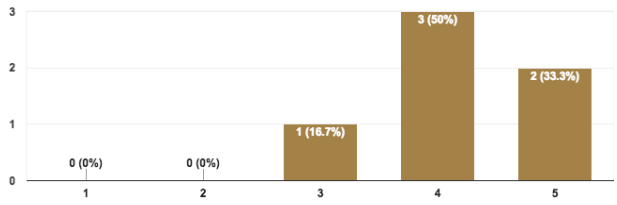
Optional: If you encountered any technical issues, can you please describe them here?

5 responses

- finding my builtin mic, but got it in the end
- The only issue I ran into was that when the fire was fully white at the bottom the text for the "continue" button was obscured, as it was also white.
- I didn't get the feedback form when the story finished
- no issues! :)
- Only that it was difficult to see the Continue button. A background colour might help.

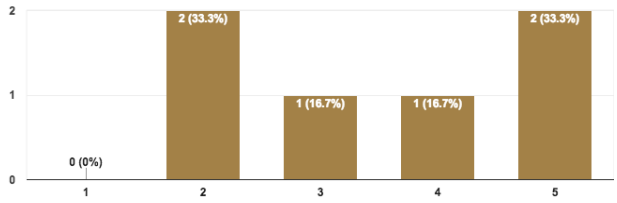
I enjoyed the story that I made

6 responses



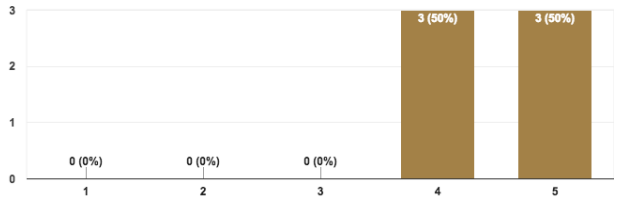
I would like to share the story I made with others

6 responses



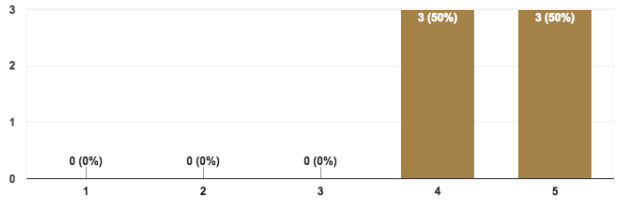
I would recommend Campfire Tales to friends

6 responses



I plan to play Campfire Tales again

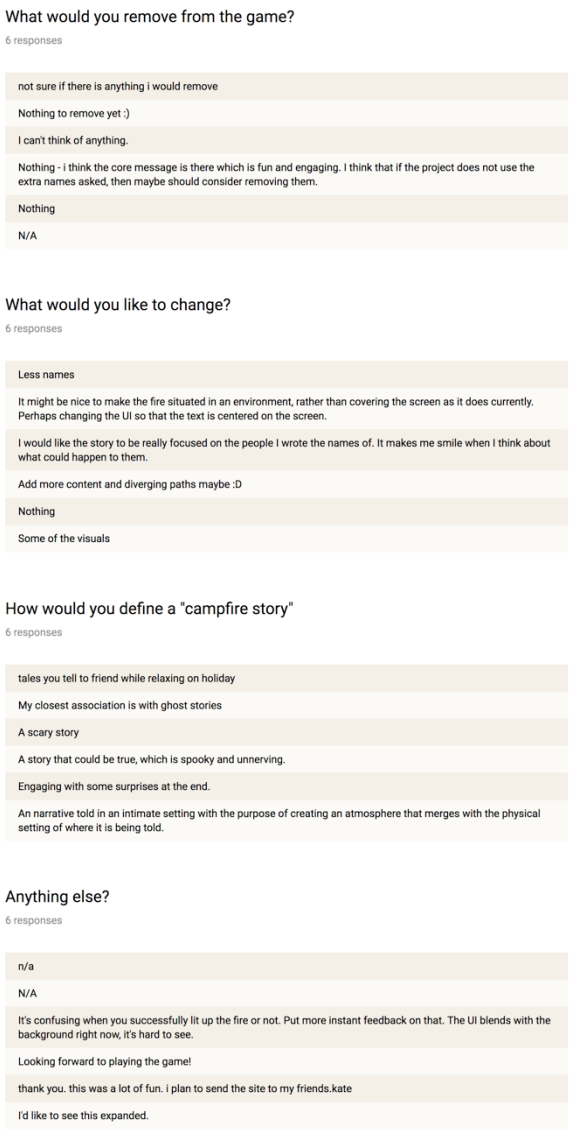
6 responses



If you could add anything to the game, what would it be?

6 responses

- More player input
- An option to skip the microphone section for people playing the game on their browser would be great!
- More impact from the data I put in. A story about twice as long.
- More divergence on story paths, maybe different stories for different playthroughs.
- It is fine the way it is right now.
- More sound effects that relate to the story



ACKNOWLEDGMENT

I wish to thank Phoenix Perry for lending the use of her Philips Hue kit during development, Hugh S. Kennedy for JavaScript guidance, and the rest of my MA cohort for testing *Campfire Tales* and contributing to the success of *Playful Experiences*.

REFERENCES

[1] J. Vansina, Oral Tradition as History, Madison, WI: University of Wisconsin Press, 1985, pp. 3,27.

[2] H. Hagebölling, "Elements of a History of Interactive Dramaturgy: Cultural Fingerprints in the Digital Net," in *Interactive Dramaturgies: New Approaches in Multimedia Content and Design*, Berlin ; New York, Springer, 2004, pp. 9-16.

[3] P. W. Wiessner, "Embers of society: Firelight talk among the Ju/'hoansi Bushmen," *Proceedings of the National Academy of Sciences*, vol. 111, no. 39, p. 14027, 2014.

[4] R. Wilson, Wild and Weird Campfire Stories, Moab: Yellow Cat Publishing, 2012.

[5] S. Friedman, "How to Tell a Good Campfire Story," 27 October 2014. [Online]. Available: https://www.backpacker.com/survival/how-to-tell-a-good-campfire-story. [Accessed 1 August 2018].

[6] "Campfire Stories - Ultimate Camp Resource," [Online]. Available: http://www.ultimatecampresource.com/site/camp-activities/campfire-stories.html. [Accessed 8 August 2018].

[7] Dim Bulb Games, "Where the Water Tastes Like Wine," Good Shepherd Entertainment, 2018.

[8] Gamewright, *Rory's Story Cubes*, 2010.

[9] N. a. T. J. a. N. M. J. Shaker, Procedural Content Generation in Games: A Textbook and an Overview of Current Research, Springer, 2016.

[10] K. Compton, "So you want to build a generator...," 22 February 2016. [Online]. Available: http://galaxykate0.tumblr.com/post/139774965871/so-you-want-to-build-a-generator. [Accessed 27 July 2018].

[11] Niantic, "Pokemon Go," Niantic, 2016.

[12] Hello Games, "No Man's Sky," Hello Games, 2016.

[13] K. Hollings, "What is the Cut-Up Method?," 25 June 2015. [Online]. Available: https://www.bbc.com/news/magazine-33254672. [Accessed 17 September 2018].

[14] Blizzard Entertainment, "Starcraft," Blizzard Entertainment, 1998.

[15] J. H. Conway, "Game of Life," 1970.

[16] T. Adams and Z. Adams, "Dwarf Fortress," Bay 12 Games, 2006.

[17] M. Cook, "Alien Languages: How We Talk About Procedural Generation," 18 August 2016. [Online]. Available: http://www.gamesbyangelina.org/2016/08/proceduralanguage/. [Accessed 27 July 2018].

[18] C. Martens and R. Cardona-Rivera, "Procedural Narrative Generation," 2017. [Online]. Available: https://www.gdcvault.com/play/1024143/Procedural-Narrative. [Accessed 27 July 2018].

[19] Nerial, "Reigns: Her Majesty," Devolver Digital, 2017.

[20] F. Ramallo, "PANORAMICAL," Finji, 2015.

[21] Somerset House, *Now Play This!*, London, 2018.

[22] Hex-Ray Studios, "Pixel Fireplace," Hex-Ray Studios, 2015.

[23] A. Judge, "Tweet," 2017. [Online]. Available: https://twitter.com/ahjudge/status/940704412930117633. [Accessed 8 August 2018].

[24] T. Martens, "Pixel Fireplace," [Online]. Available: https://hammertail.itch.io/pixel-fireplace. [Accessed 8 August 2018].

[25] W. Benjamin, "The Work of Art in the Age of Mechanical Reproduction," 1936. [Online]. Available: https://www.marxists.org/reference/subject/philosophy/works/ge/benjamin.htm. [Accessed 7 August 2018].

[26] B. Dias, "Improv," [Online]. Available: https://github.com/sequitur/improv. [Accessed 1 July 2018].

[27] B. Dias, "Voyageur," Fundbetter, 2017.

[28] B. Dias, "Generating Text With Improv," 2017. [Online]. Available: http://www.procjam.com/tutorials/en/improv/. [Accessed 19 June 2018].

[29] N. Fleck, "Rant," 2017. [Online]. Available: https://berkin.me/rant/. [Accessed 25 June 2018].

[30] K. Compton, "Tracery: generate text, graphics and mroe," [Online]. Available: http://tracery.io/. [Accessed 25 June 2018].

[31] D. C. Howe, "RiTa a software toolkit for computational literature," [Online]. Available: http://rednoise.org/rita/. [Accessed 25 June 2018].

[32] D. Shiffman, "15: Twitter Bot Tutorial - Node.js and Processing," 19 September 2017. [Online]. Available: https://www.youtube.com/playlist?list=PLRqwX-V7Uu6atTSxoRiVnSuOn6JHnq2yV. [Accessed July 2018].

[33] M. Deline, "Campfire Tales," 2018. [Online]. Available: https://twitter.com/CampfireTale. [Accessed 2018].

[34] G. Buckenham, "Cheap Bots, Done Quick!," [Online]. Available: https://cheapbotsdonequick.com. [Accessed July 2018].

[35] M. DesLauriers and yoshuawuyts, "budō," June 2018. [Online]. Available: https://www.npmjs.com/package/budo. [Accessed June 2018].

[36] G. Buckenham, "tracery-grammar," 2016. [Online]. Available: https://www.npmjs.com/package/tracery-grammar. [Accessed June 2018].

[37] oksmith, "Bonfire," 24 April 2018. [Online]. Available: https://openclipart.org/detail/300895/bonfire. [Accessed July 2018].

[38] codetheweb, "TuyAPI," [Online]. Available: https://github.com/codetheweb/tuyapi. [Accessed July 2018].

[39] The Daniels Wood Land Show, "Insane Fake Fire Special Effect!," 1 March 2016. [Online]. Available: https://www.youtube.com/watch?v=htBYgRNvkmk. [Accessed July 2018].

[40] Kampeki Factory, "Set Your Browser On Fire With P5.Js," 7 March 2018. [Online]. Available: https://kampeki-factory.blogspot.com/2018/03/set-your-browser-on-fire-with-p5js.html. [Accessed August 2018].

[41] aerror, "campfire.wav," 25 July 2016. [Online]. Available: https://freesound.org/people/aerror/sounds/350757/. [Accessed August 2018].

[42] felix.blume, "Forest at dawn with birds, crickets and insects in the Sian Ka'an Biosphere Reserve," 9 November 2015. [Online]. Available: https://freesound.org/people/felix.blume/sounds/328296/. [Accessed August 2018].

[43] felix.blume, "Forest at night, crickets, cicadas and insects in the Sian Ka'an Biosphere Reserve," 9 November 2015. [Online]. Available: https://freesound.org/people/felix.blume/sounds/328293/. [Accessed August 2018].

[44] A. Fletcher, Composer, *Magical Forest*. [Sound Recording]. 2018.

[45] Gifer, "Transparent Loading Bar Gif," [Online]. Available: https://gifer.com/en/YCZH. [Accessed August 2018].

[46] P. Murray, "node-hue-api Node.js Library for interacting with the Philips Hue Bridge and Lights," 29 August 2018. [Online]. Available: https://github.com/peter-murray/node-hue-api. [Accessed 29 August 2018].

[47] socket.io, "socket.io," [Online]. Available: https://socket.io. [Accessed August 2018].

[48] homejrande, "fire wood bonfire low eq.aif," 28 March 2006. [Online]. Available: https://freesound.org/people/homejrande/sounds/17375/. [Accessed August 2018].

[49] Playful Experiences, "Playful Experiences," August 2018. [Online]. Available: http://playfulexperiences.com. [Accessed August 2018].

[50] H. Hagebölling, "Aspects of Interactive Dramaturgies: Thematic Frame and Authors' Contributions," in *Interactive Dramaturgies: New Approaches in Multimedia Content and Design*, H. Hagebölling, Ed., Berlin; New York, Springer, 2004, p. 260.

Matthew Deline is currently completing his MA Independent Games and Playable Experience Design with Goldsmiths, University of London. He has a BA in English and Comparative Literature, and was the 2009 Outstanding Graduate in the subject at San Diego State University. Recently, he has worked as a contributing artist for *The Long Run*, a commissioned work by the British Medical Association for the 70th anniversary of the NHS, runs a travel blog under the name *The Radical Dreamer*, and formerly worked as a senior technical support agent for the Worldwide Operations group at Apple Computer, Inc. He looks forward to using his expertise to make new types of games and interactive experiences.