GDC

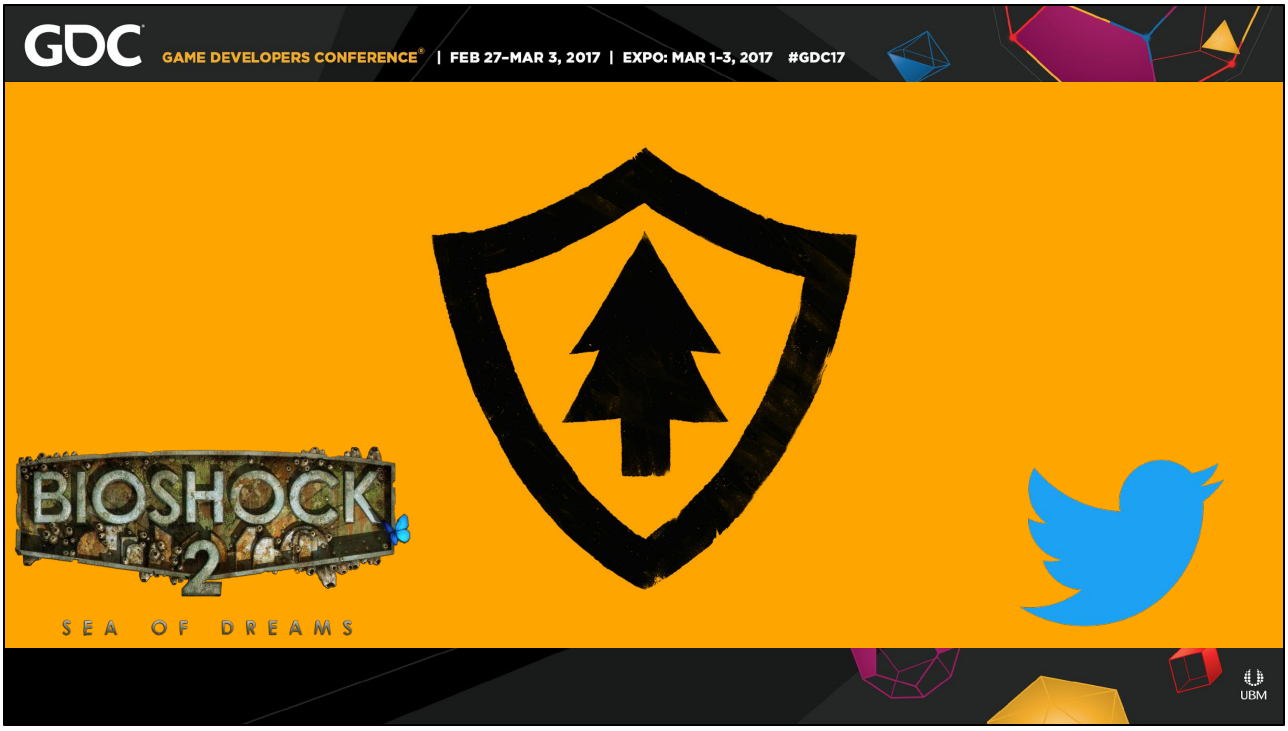
# Do You Copy?

## The dialog system in Firewatch

**Patrick Ewing**
**& William Armstrong**
Programmers on Firewatch



UBM




<<BOTH>>

We made Firewatch together
William Armstrong - Tools and Systems Programmer
Bioshock 2

Patrick Ewing — Tools and Gameplay programmer
Twitter engineering, Campo Santo, now working on a new game with Chance Agency

# Prior Art

Left 4 Dead's Barks - From GDC 2012 Elan Ruskin's talk on Dynamic Dialog
http://www.gdcvault.com/play/1015317/AI-driven-Dynamic-Dialog-through

Naughty Dog - Context Aware Dialog at GDC 2014
http://www.gdcvault.com/play/1020951/A-Context-Aware-Character-Dialog

Prince of Persia (2008) - A player initiated chat about the game button



<<WILL>>

- Prior work focused on barks. Having natural sounding conversations that can be interrupted.
- We aren't going to go into technical detail of the fact system and the requirement checking, Elan's talk does a perfect job of that. Go watch it.
- Our game wanted to put bark's in the hands of the player. Use the radio to say a thing, then have the game strike up a conversation with you about it.
- Made the prior systems seem a perfect fit

# What we thought we were doing

- Initial prototype late 2014
- Player can always start talking
- Only Henry starts talking



<<WILL>>
Initial prototypes
Player initiated conversations
Never take away player agency - Like a real conversation, you can interrupt… or let the other perso just TALK
Player can always interrupt
Player always starts conversations
This sounded A LOT like barks

# What we built

Generic Event System w/ Blackboards



<<WILL>>
Quick overview of the Event System implementation
Anywhere in the game can trigger an event, code, script, other events
When an event is raised, we find the best response.

# What we built

Generic Event System w/ Blackboards

Requirements -> Responses



<<WILL>>
When some game state is reached -> alter the game state

# What we built

Generic Event System w/ Blackboards

Requirements -> Responses
[Event Name]+[List of Boolean tests] -> Responses



<<WILL>>
Ideally, an Event Happens + Required State => Desired New State

# What we built

Generic Event System w/ Blackboards

Requirements -> Responses
[Event Name]+[List of Boolean tests] -> Responses
[Event Name]+[Target Name]+[Sender Name]+[Boolean Tests] -> Responses



<<WILL>>
Ideally, facts could contain strings, but, we were in a rush and the tools for that are trickey so we special cased who sent the event and who was targeted by the event. Either of these can be null. This was enough for handling barks so seemed reasonable.

# What we built

Generic Event System w/ Blackboards

Requirements -> Responses
[Event Name]+[List of Boolean tests] -> Responses
[Event Name]+[Target Name]+[Sender Name]+[Boolean Tests] -> Responses
OnTossed + BeerCan + Player + (LinePlayed == FALSE) -> Say "F@#$ it, I'm not the maid."



<<WILL>>
Concrete example

OnTossed, the beer can, by Henry, Haven't Played

# What we built

## Requirements

Event Name - Designer created name describing what happened

Target and Sender - the name of the game object that is sending or receiving an event

Facts - floating point comparisons against Facts on various Blackboards



<<WILL>>
How we match events and store facts

# What we built

## Blackboards

Simple Key-Value Pair dictionaries

Use different Blackboards to scope Facts to a relevant context



<<WILL>>

How we match events and store facts

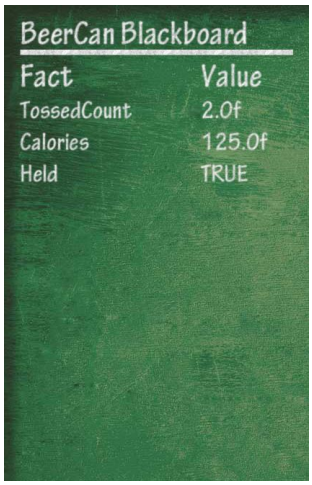
# What we built

## Blackboards

Simple Key-Value Pair dictionaries

Use different Blackboards to scope Facts to a relevant context

- Game Object Blackboard






<<WILL>>

How we match events and store facts

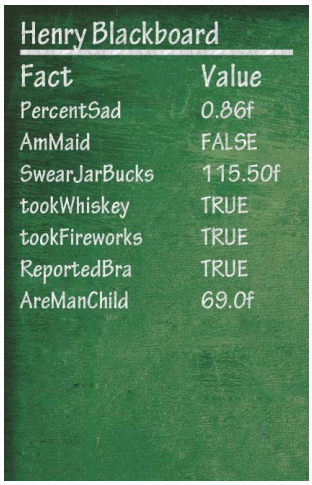
# What we built

## Blackboards

Simple Key-Value Pair dictionaries

Use different Blackboards to scope Facts to a relevant context

- Game Object Blackboard
- Player Blackboard






<<WILL>>

How we match events and store facts

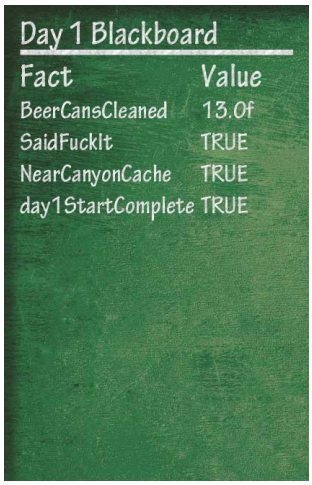
# What we built

## Blackboards

Simple Key-Value Pair dictionaries

Use different Blackboards to scope Facts to a relevant context

- Game Object Blackboard
- Player Blackboard
- Day Specific Blackboard






<<WILL>>

How we match events and store facts

# What we built

## Blackboards

Simple Key-Value Pair dictionaries

Use Blackboards to scope Facts to context

- Game Object Blackboard
- Player Blackboard
- Day Specific Blackboard
- Global Blackboard

Global Blackboard

| Fact | Value |
|---|---|
| MysteryDone | 0.72f |
| QUEST_GoodEnding | 0.0f |
| NamedDogBucket | FALSE |
| NamedDogMayhem | TRUE |
| TimeSinceSpeech | 0.0001f |
| LightsAreOn | FALSE |
| StrobesAreOn | TRUE |
| BoomboxOn | TRUE |
| PartyStarted | TRUE |



<<WILL>>

How we match events and store facts


<<WILL>>
Here is a concrete example of our entire tool chain. These are the tools we used to make the whole game. Notice how bare-bones everything is. Low abstraction.

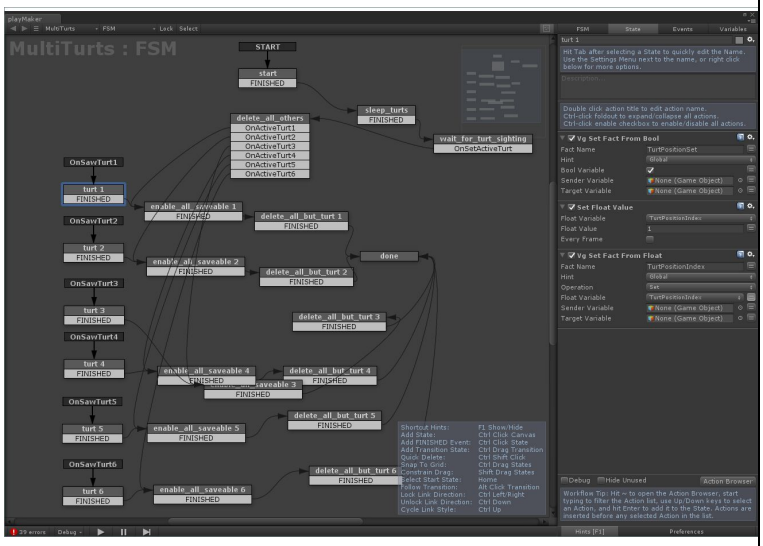



<<WILL>>
Notice how one of the names is based on a Twine choice
Notice how the display text changes we when put the Turtle back down

# Responses

## Secret Multi Turt

## Event Driven Level Script






<<WILL>>

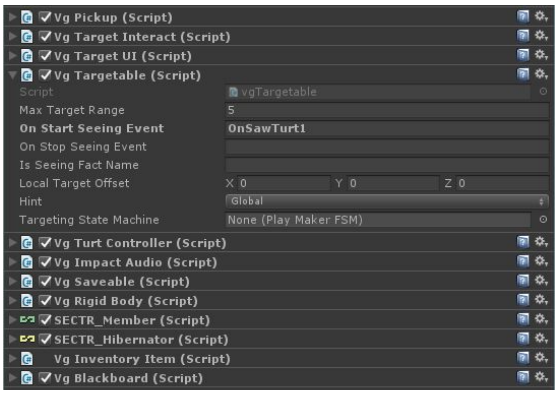
This the PlayMaker state machine that manages the Turt Pickups

Dirty Secret of Turt, there are many of them out in the world. The first one you see becomes your Canonical Turt, and we turn off the rest of them. We wanted the turtle to be unique, and fun little hidden surprise, but our world was too big, and players weren't finding it often enough.

While complicated, and not pretty, this state machine was made late in the game, using only existing systems. Other than advice and bug hunting, no programmer time was required.

# Gameplay Event Triggers

A wild Turt
Sends ‘OnSawTurt1’



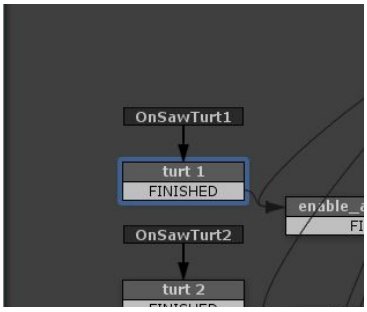



<<WILL>>

An example of a gameplay system triggering an event
By default just sends a generic OnSeen event with this object as the Target
Specialized for ease of use / searching

Script waits on an Event from the Event System

Setting facts from PlayMaker



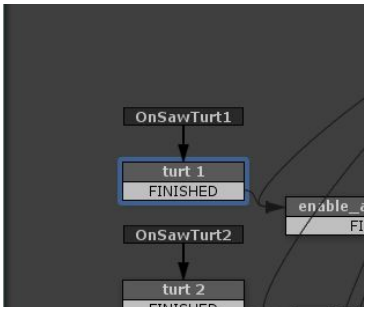


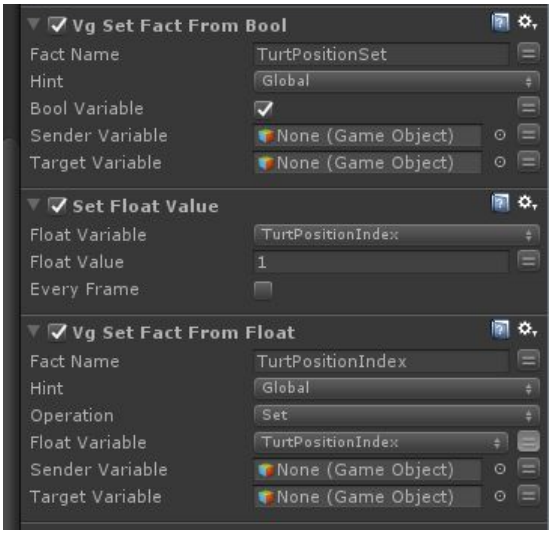

<<WILL>>

Here you can see a state that happens when a specific event occurs, and then sets facts to a blackboard, in this case the global one
This makes the first Turt you see the Canonical Turt for your playthrough

Script waits on an Event from the Event System

Setting facts from PlayMaker

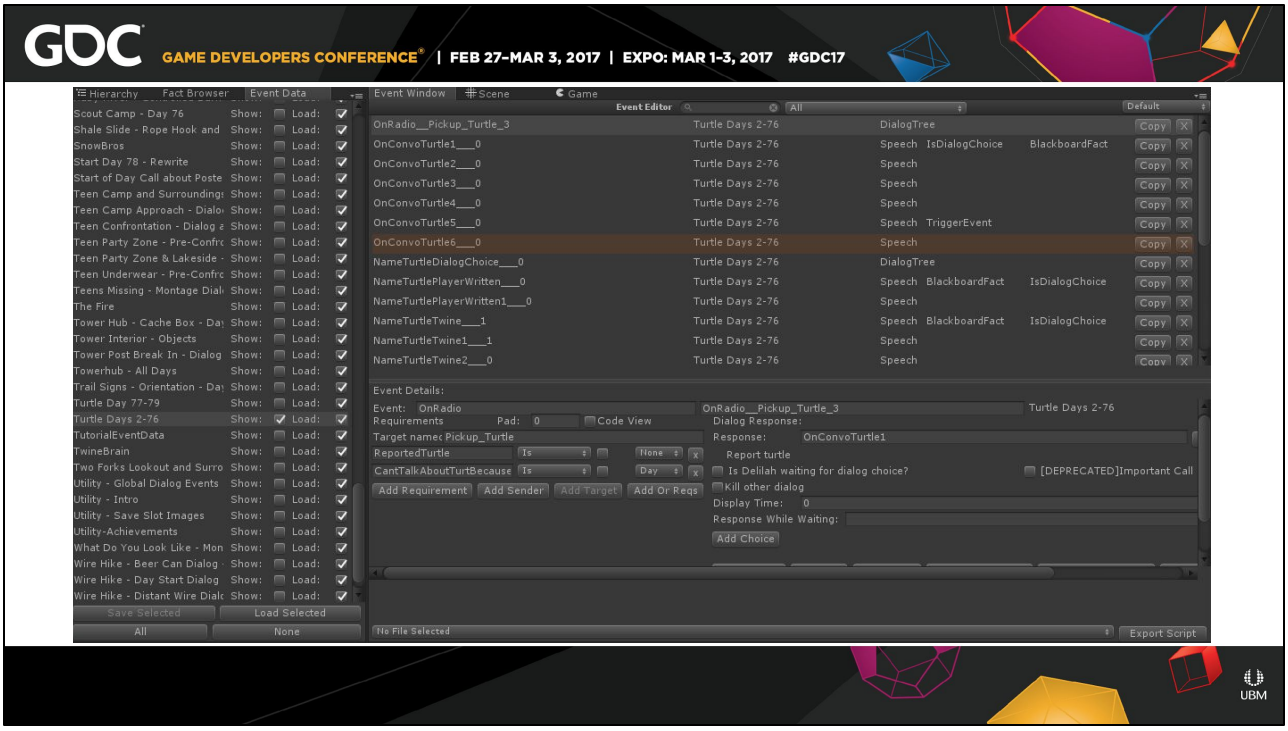





<<WILL>>

Here you can see a state that happens when a specific event occurs, and then sets facts to a blackboard, in this case the global one
This makes the first Turt you see the Canonical Turt for your playthrough



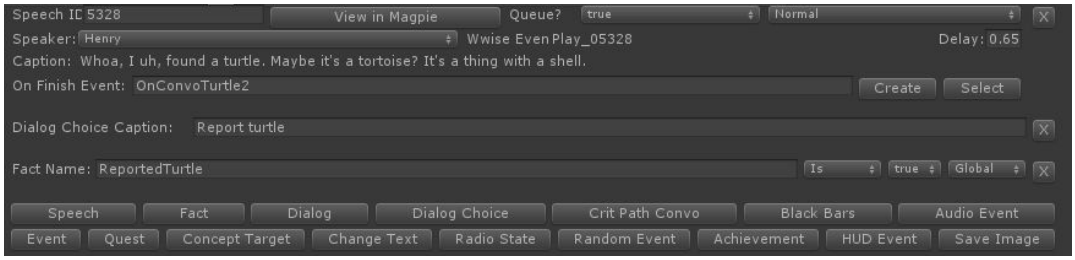

<<WILL>>

Here is our in engine editor for the event data

Lots of events about turt, all sorted into their own event list
We have tools to multi-select and jump quickly from a dialog line to any of its followups.
The selected line is in fetching Campo Orange (156,70,5)

# Dialog Responses

Speech to play
Dialog caption to display
Setting a fact






<<WILL>>

This is what at basic Dialog Response looks like, how we play lines

# Dialog Tree Responses

List of other events to trigger
Finds valid responses on display
Previews choices




<<WILL>>

Here is our dialog tree response

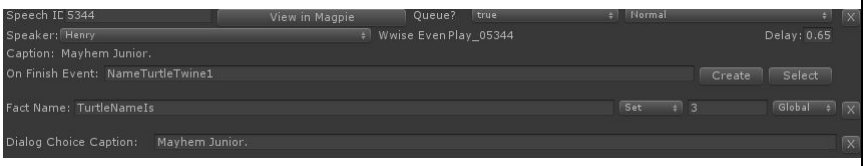
All it does is fire off other events with their own responses
We have a custom response type that sets the Caption for each choice, and when we first set an active dialog we go find those captions based on the game state at that moment

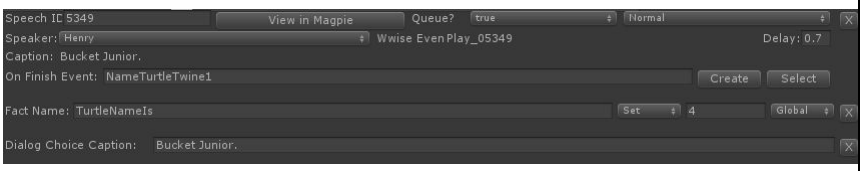
# Memory

Use the blackboard to select name

Based on player choices

Selects the event with the most matching requirements



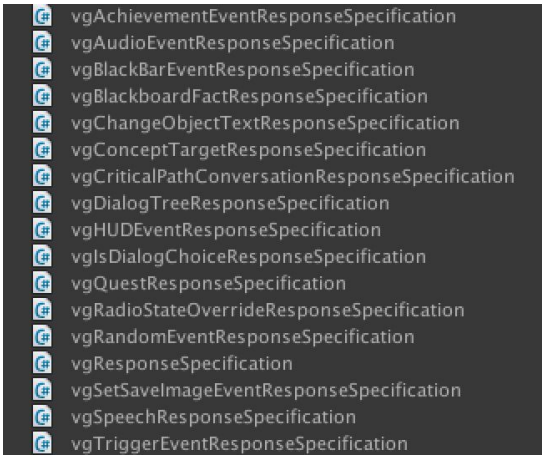

<<WILL>>

How we use our Blackboard system to store game state and remember things that happened

# Responses

Much more than just dialog

- Grant Achievements
- Play Sounds
- Alter object display text
- Bring up Letterboxing
- Lock player into a conversation
- Trigger UI events
- Override the default Radio animations
- Change the save game image
- Trigger more events or manipulate blackboard facts
- Integrated fully with level scripting






<<WILL>>

All sorts of things can be a response.
All events can trigger move events, set facts, trigger game logic, and all sorts of hooks into scripting

Examples - Save Game image swapping, setting an index into a list of pre-generated save game images
Examples - Event Listeners in PlayMaker

# Response Code

```
using UnityEngine;
using System.Collections;

[System.Serializable]
public class vgDialogTreeResponseSpecification : vgResponseSpecification
{
    public vgDialogDefinition DialogToShow = new vgDialogDefinition();

    public override void Execute(vgEventResponseSpecification sourceSpec = null)
    {
        if( DialogToShow != null )
        {
            vgLocalizedData locData = sourceSpec != null ? sourceSpec.sourceData.localizedEventData : null;
            DialogToShow.localizedData = locData;
            DialogToShow.eventDataName = sourceSpec != null ? sourceSpec.sourceData.name : null;

            vgDialogTreeManager.Instance.TryStartDialogOptions(DialogToShow, vgEventManager.Instance.currentSender, vgEventManager.Instance.currentTarget);
        }
    }

    public override void OnCustomGUI(vgEventEditorCallbacks callbacks)
    {
    #if UNITY_EDITOR
        DialogToShow.OnCustomGUI( callbacks );
    #endif
    }

    public override string GetDebugText()
    {
        if( DialogToShow != null && DialogToShow.Choices != null )
        {
            string outString =  "EVENT LOG: Dialog Event:" + DialogToShow.Choices.Count + " Choices: ";
            foreach( vgDialogChoice choice in DialogToShow.Choices )
            {
                outString += "\nEVENT LOG: Display: " + choice.displayText + " Event: " + choice.EventResponse;
            }
            return outString;
        }
        return "EVENT LOG: Empty Dialog Response";
    }
}
```



<<WILL>>

Here is what basically all of the responses ended up looking like in code. Dead simple.
Marshall state from the event system, then pass the data onto a more complicated specialized system.
Code to alter how event responses are displayed in the editor for validation
Lots of code for printing debug text, debugging this system is more important than it working

# Response Code

Easy to add

Thin wrapper

```
using UnityEngine;
using System.Collections;

[System.Serializable]
public class vgDialogTreeResponseSpecification : vgResponseSpecification
{
    public vgDialogDefinition DialogToShow = new vgDialogDefinition();

    public override void Execute(vgEventResponseSpecification sourceSpec = null)
    {
        if( DialogToShow != null )
        {
            vgLocalizedData locData = sourceSpec != null ? sourceSpec.sourceData.localizedEventData : null;
            DialogToShow.localizedData = locData;
            DialogToShow.eventDataName = sourceSpec != null ? sourceSpec.sourceData.name : null;

            vgDialogTreeManager.Instance.TryStartDialogOptions(DialogToShow, vgEventManager.Instance.currentSender, vgEventManager.Instance.currentTarget);
        }
    }

    public override void OnCustomGUI(vgEventEditorCallbacks callbacks)
    {
    #if UNITY_EDITOR
        DialogToShow.OnCustomGUI( callbacks );
    #endif
    }

    public override string GetDebugText()
    {
        if( DialogToShow != null && DialogToShow.Choices != null )
        {
            string outString =  "EVENT LOG: Dialog Event:" + DialogToShow.Choices.Count + " Choices: ";
            foreach( vgDialogChoice choice in DialogToShow.Choices )
            {
                outString += "\nEVENT LOG: Display: " + choice.displayText + " Event: " + choice.EventResponse;
            }
            return outString;
        }
        return "EVENT LOG: Empty Dialog Response";
    }
}
```



<<WILL>>

Here is what basically all of the responses ended up looking like in code. Dead simple.
Marshall state from the event system, then pass the data onto a more complicated specialized system.
Code to alter how event responses are displayed in the editor for validation
Lots of code for printing debug text, debugging this system is more important than it working

# Response Code

Easy to add

Thin wrapper

Focused on debugging

```
using UnityEngine;
using System.Collections;

[System.Serializable]
public class vgDialogTreeResponseSpecification : vgResponseSpecification
{
    public vgDialogDefinition DialogToShow = new vgDialogDefinition();

    public override void Execute(vgEventResponseSpecification sourceSpec = null)
    {
        if( DialogToShow != null )
        {
            vgLocalizedData locData = sourceSpec != null ? sourceSpec.sourceData.localizedEventData : null;
            DialogToShow.localizedData = locData;
            DialogToShow.eventDataName = sourceSpec != null ? sourceSpec.sourceData.name : null;

            vgDialogTreeManager.Instance.TryStartDialogOptions(DialogToShow, vgEventManager.Instance.currentSender, vgEventManager.Instance.currentTarget);
        }
    }

    public override void OnCustomGUI(vgEventEditorCallbacks callbacks)
    {
    #if UNITY_EDITOR
        DialogToShow.OnCustomGUI( callbacks );
    #endif
    }

    public override string GetDebugText()
    {
        if( DialogToShow != null && DialogToShow.Choices != null )
        {
            string outString =  "EVENT LOG: Dialog Event:" + DialogToShow.Choices.Count + " Choices: ";
            foreach( vgDialogChoice choice in DialogToShow.Choices )
            {
                outString += "\nEVENT LOG: Display: " + choice.displayText + " Event: " + choice.EventResponse;
            }
            return outString;
        }
        return "EVENT LOG: Empty Dialog Response";
    }
}
```



<<WILL>>

Here is what basically all of the responses ended up looking like in code. Dead simple.
Marshall state from the event system, then pass the data onto a more complicated specialized system.
Code to alter how event responses are displayed in the editor for validation
Lots of code for printing debug text, debugging this system is more important than it working

# In-Engine Tools

**Pros**

- Cheap to make
- Flexible
- Extensible
- Discoverable
- Integrated runtime

**Cons**

- Visually unappealing
- Complex
- Ever changing



<<WILL>>

- No assumptions. Everything is an event or a response, a fact or a condition
- Used for much more than dialog.
  - UI
  - Animation
  - Level Scripting
  - Conditional Spawning
- This allows for a multiplicative explosion of functionality. Anything that can trigger an event can cause any of the above as a response. Any new response can be triggered from every event in the gamee

This all has a downside though. A tool that is build to iterate functionality on everything will not be perfect for anything. This is, at the end of the day, an editor for the event system, not an editor for Dialog.

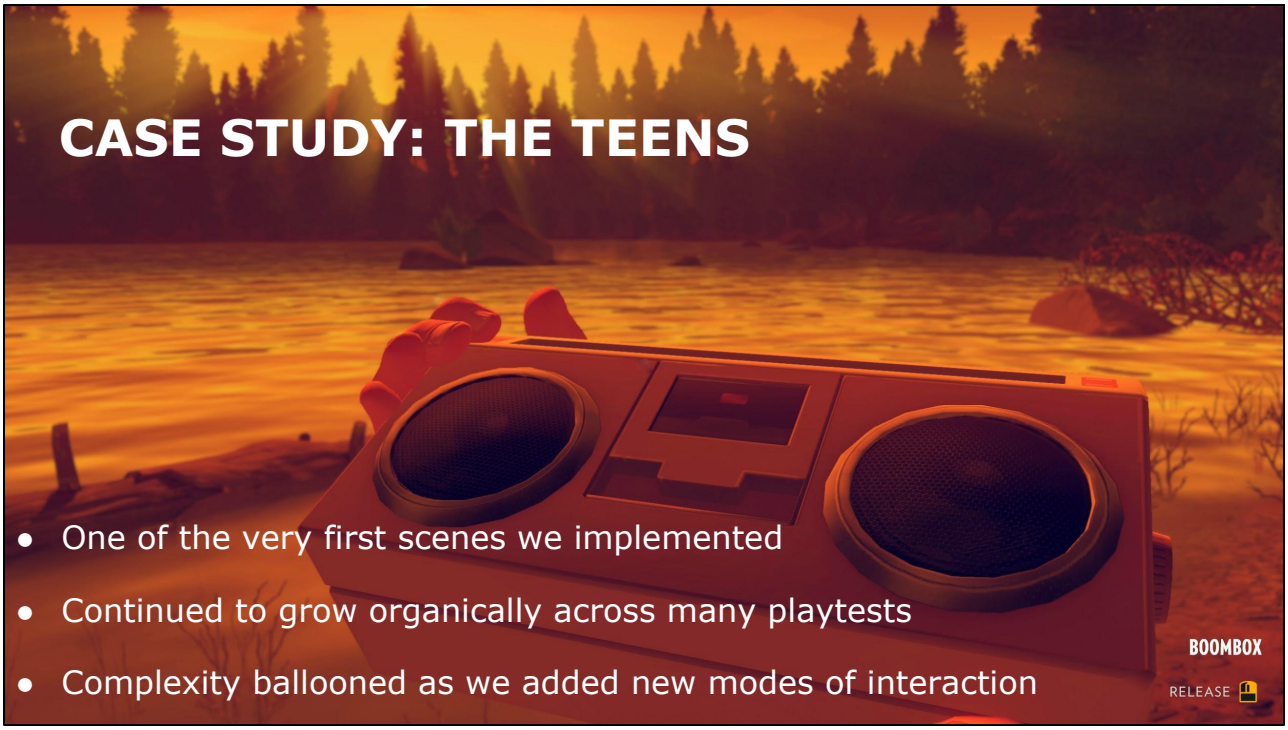
# Now for the hard part!

From a working system to a content pipeline



<<WILL>> Hand off to Patrick— he will change the slide

<<Patrick>>
With all of these systems in place and working, we needed to figure out a pipeline that could fill a 4 hour game with massively branching content. An example of how this content looked while we were just getting started is The Teens.




<<PATRICK>>

This is one of the first scenes we implemented, and it grew to be one of the most complex. Updated continuously throughout development.

- We always wanted to add more
- Responded to player feedback constantly
- ‘What happens if I throw the boombox in the lake’? What if I just sneak by them?
- Organically growing over time. Entire team involved.
- Debugging stress test. We had enough rope to hang the whole team here, and did. Repeatedly.
- All content, very little code needed as we iterated.

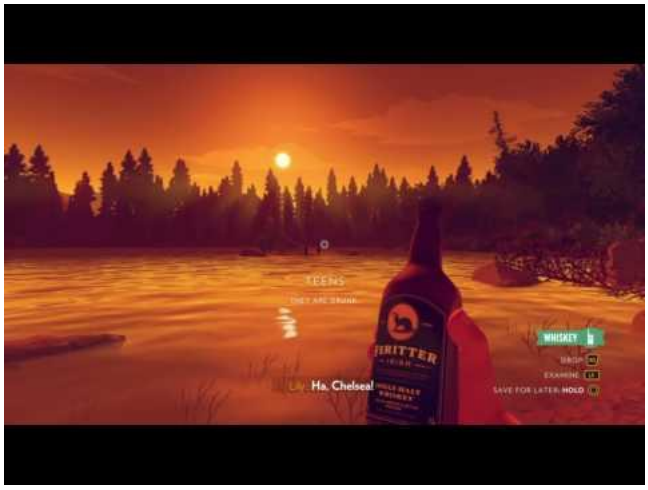
# Case Study: The Teens




<<PATRICK>>

Updated continuously through the whole game. Drove much of our tools development. If we could do all this, we could do anything.

# Case Study: The Teens - Polite Run

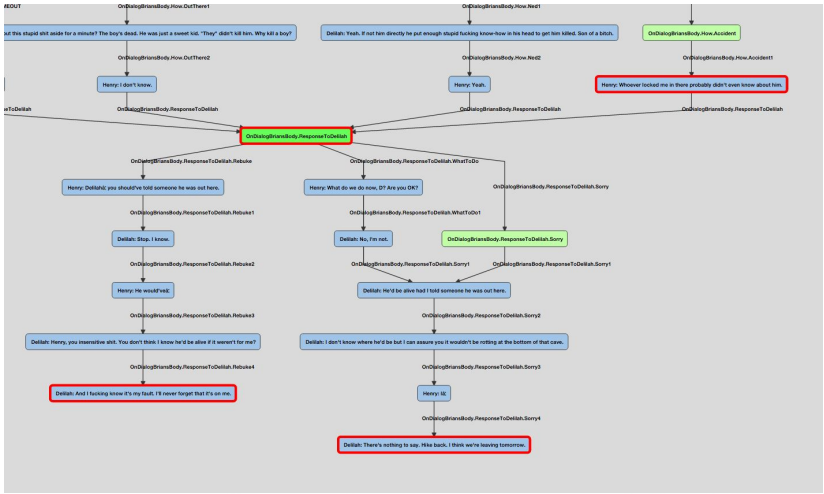





<<PATRICK>>

Pros of this approach: We had to write almost no new code as we developed the game. It was all event data.
Cons: New content could often disrupt old content. Events interrupting events— new meanings given to older facts; thousands of permutations to test.

# Case Study: The Teens






<<PATRICK>>

- Pros of this approach: We had to write almost no new code as we developed the game. It was all event data.
- Cons: New content could often disrupt old content. Events interrupting events— new meanings given to older facts; thousands of permutations to test.

So what could we learn from this vertical slice? How could we make writing, updating and changing content easier as we went forward?

# Writing for Games is **HARD**

- Massively branching content
- Non-linear = a web of independent lines
- Nearly every line is context-aware
- Firewatch is full of interruptible dialog
- Excel is a terrible creative writing tool



<<PATRICK>>

Interactive fiction writers have a super unique talent— it's unlike anything else.

- Massively branching content
  - Writers keep track of everything of massive amounts of info in their heads— but then need to capture that on paper. Sean prototyped the vignettes that start Firewatch in Twine, a visual tool
- Non-linear = a web of independent lines
  - But Twine isn't sufficient for a game like Firewatch. Lines need to feed into a Voice recording and localization pipeline.
- Nearly every line is context-aware
  - State needs to be checked on every frame. State must be maintained in a vast number of blackboards.
- Firewatch is full of interruptible dialog
  - From the get go, we knew that Henry could call Delilah and change the subject as she rambled on. We later added the ability for Delilah to call Henry. This means thinking about event chains and all the possible ways they can be broken, recovered from, or guarded.
- Excel is a terrible creative writing tool

- ...and yet it’s what a large part of the industry uses to track lines

# Magpie

**Speech & Dialog CMS**

# Goals

- Track a line's progress through the pipeline: Write → Record → Wire → Localize
- Make line edits & story rewrites less buggy
- Keep track of past versions
- Let writers use their preferred text editor
- Seamlessly integrate with Unity
- Grow into a localization solution
- Web based

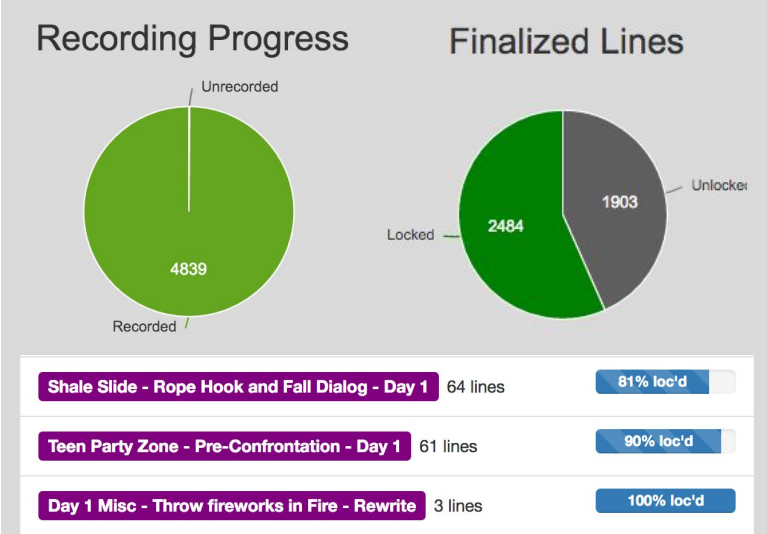


<<PATRICK>>

- Magpie is an end-to-end CMS— a database-backed web application for managing dialog.
- Writers use their own tools, then bulk import into our line database
- One line = one recording = many usages = many translations
- Non destructive editing- all changes tracked & synced
- Edit inline, unity refreshes automatically
- Listen inline / Translate inline
- Web based tools— quick to make, lives in the cloud, easy to share with non programmers

# Magpie

## Progress Tracking






<<PATRICK>>
Benefits of a web-based CMS— easily set up dashboards like this one
Recording progress, Wiring progress, Loc progress

# Magpie

## Progress Tracking

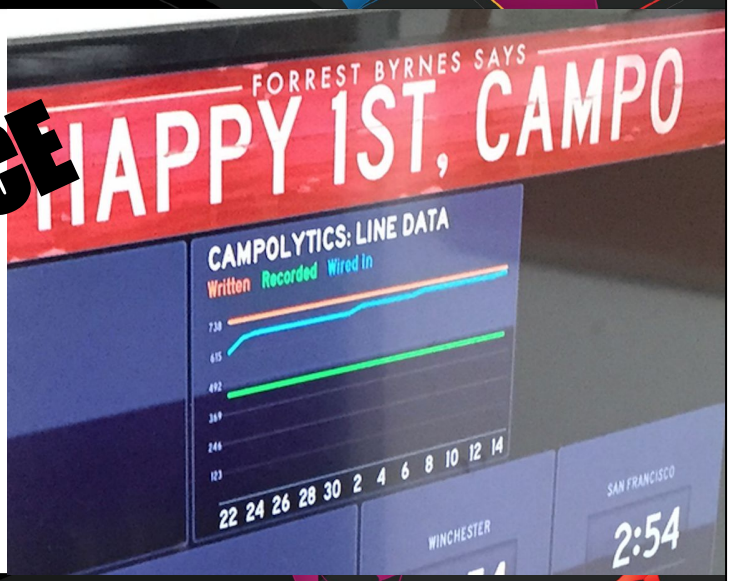



<<PATRICK>>
Panic's Status Board was our friend
The bus times up in the right, our graph of how many times employees were fired for terrible puns in the right

# Magpie

## Progress Tracking

ENHANCE






<<PATRICK>>

And our line data pipeline is tracked too. A status board widget talks to Magpie's JSON API

# Process

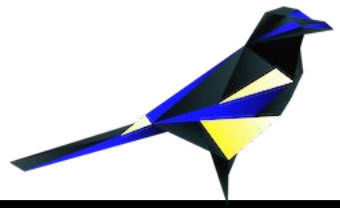
## First, writers deliver a plain-text document

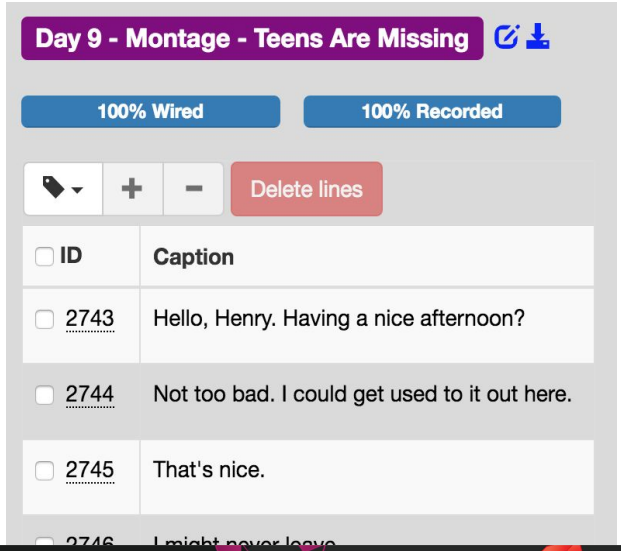
```
# Day 9 - Teens are Missing
    DELILAH
    Hello, Henry. Having a nice afternoon?

    <<1. Yeah, great.>>
    HENRY
    Not too bad. I could get used to it out here.

      DELILAH
      That's nice.

    <<2. I might never leave.>>
    HENRY
    I might never leave.

      DELILAH
      Well, I called with some bad news.
```



<<PATRICK>>
This is a simple format based on “Markdown” that makes it easy for Magpie to parse the script.

Magpie parses the doc into a list of **lines**, each with a **unique ID**

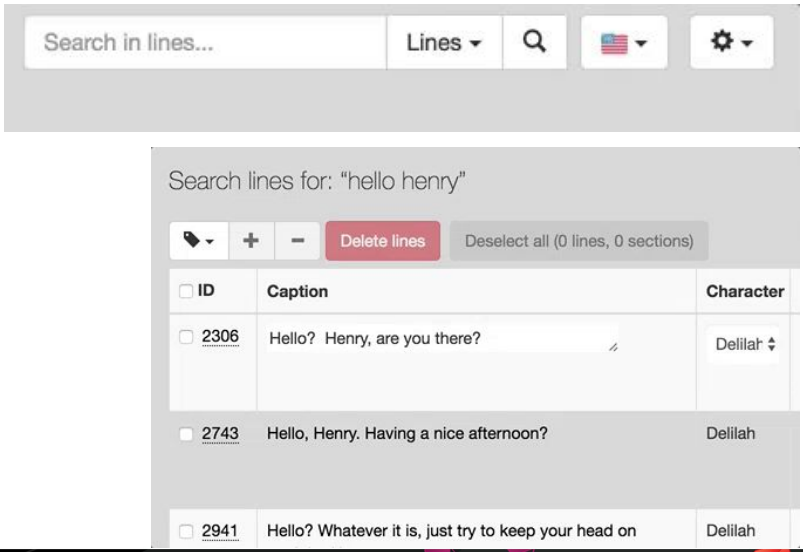







<<PATRICK>>
Magpie creates a unique Line ID and populates our database with all of the metadata you see here (character, notes, line type etc.)

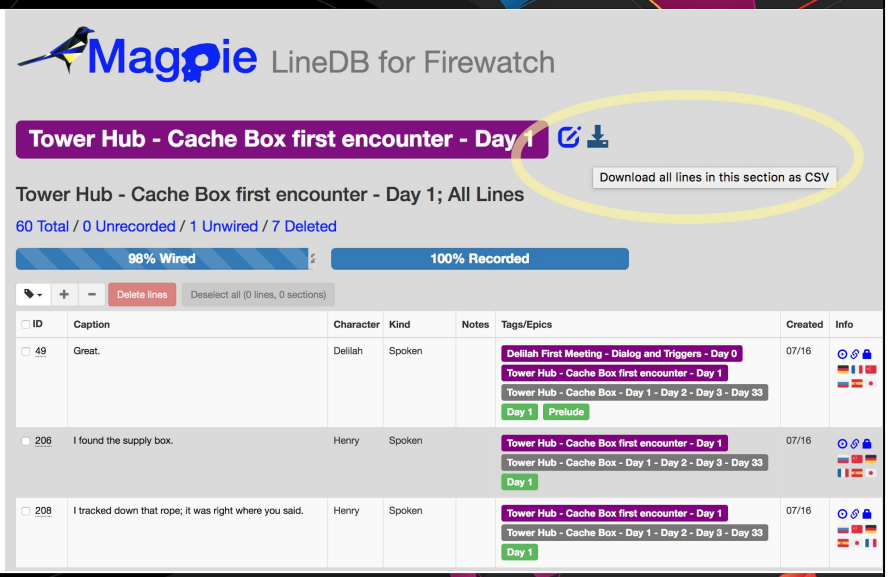
# Lines can be easily searched, grouped and edited inline






<<PATRICK>>
Lines can be grouped into lists, tagged by theme or day, and searched easily from anywhere. No need to open unity, you can even use it from your mobile device.

# Magpie exports any line list as a CSV file...






<<PATRICK>>
For any set of Lines, Magpie produces a JSON or CSV export that can be imported into other tools. JSON syncs automatically from a simple C# script in Unity. CSV was useful for building recording scripts for V/O

# Which becomes a V/O recording script in Google Sheets

FW Dialog Session 9.15.15

File Edit Rename Format Data Tools Add-ons Help All changes saved in Drive

caption

| | A | C | D | E | F | H |
|---|---|---|---|---|---|---|
| 1 | id | caption | character | session take | chosen_read | notes |
| 2 | 6088 | Well, I'm flush with ropes now. | Henry | 12 | 1 | <<#OnPickup_ |
| 3 | 6089 | I just found an old pack full of them. In decent condition too. | Henry | 12 | 1 | <<didn't reporte |
| 4 | 6090 | That pack was full of them. In decent condition too. | Henry | 12 | 1 | <<reported bag |
| 5 | 6091 | That's lucky. | Delilah | 12 | 1 | <<from either>> <<go to either t |
| 6 | 6092 | There's enough rope here that I can just leave them hooked up, I think. Oh, get this -- this pack came with one of those cardboard single-use cameras. | Henry | 12 | 1 | <<Don't know F |
| 7 | 6093 | Thanks Brian Goodwin. | Henry | 12 | 1 | |
| 8 | 6094 | ...Who? | Delilah | 12 | 1 | |
| 9 | 6095 | The bag had the name Brian Goodwin sewn into the top. | Henry | 12 | 2 | |
| 10 | | | | | | |
| 11 | 6096 | I actually found some of his stuff of back in my desk. | Henry | 13 | 1 | <<FOUND fold <<If not, just go |
| 12 | 6097 | Huh. Wow. | Delilah | 13 | 2 | |
| 13 | 6098 | Do you know him? | Henry | 13 | 2 | |
| 14 | 6099 | He was a lookout? | Henry | 13 | 2 | <<1. Lookout?> |
| 15 | 6100 | Ha, kinda, I guess. | Delilah | 13 | last | <<go to AboutE |



<<PATRICK>>

Google Sheets is a godsend for voice recording. Rich & Cissy and Sean & I could all be in the same doc at once, watching each other's cursors flitting around.

# Voice Acting Pipeline

- Producing a recording script for our actors
- Selecting, chopping and tagging “takes”
- Tag alternates for later use
- Processing, tagging and importing thousands of .wav files
- Allowing for interruptions and recoveries

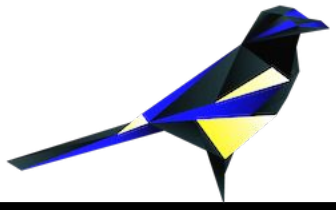


<<PATRICK>>

Rich and Cissy worked in their own home studios. We’d all be on Skype— Sean and I listneing, and the tow of them while they recording local, high quality copies. Sean would tag the takes he liked in Google sheets.

Later, an intern could chope and collate these session wav files into thousands of individual wavs, named by line ID

# Writers can listen to V/O recording easily






<<PATRICK>>
One click import/export of lines as a CSV file, for use in Google Docs, Excel, or whatever external tools you use

This became especially invaluable when we needed to do a late-game rewrite. The story changed— This meant combing through everything we had, finding lines that needed to be deleted or replaced, and creating new recording scripts that sorted brand new lines with rewrites.

A tagging system was key here— for instance, find me all the lines ABOUT Brian Goodwin, or all the Goodwin lines on Day 3 or after.

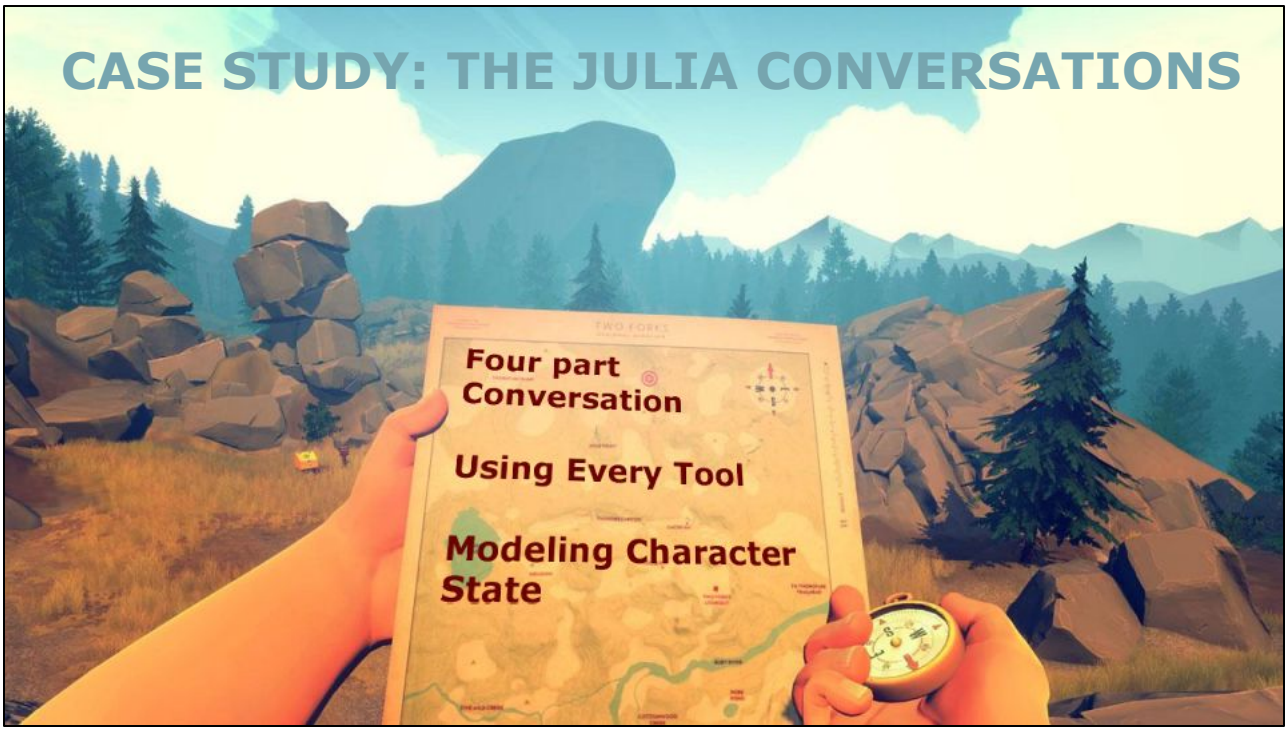
# Generation vs Maintenance

- Creating content is about speed and reliability
- Automate repetitive fragile work
- Should have done this much sooner
- Spent a lot of time trying to solve both problems



<<PATRICK>>

Talk about or original plan and attempts to make the visual graph editor version of this. The growing pains there. How much easier things were once we limited the problem to writing lots of lines of dialog, not building and maintaining a reflection of the full event system




<<PATRICK>>

- System used properly, since we had 3+ big revisions on the teens.
- The creation of Delilah and Henry brains
- Multiple Conversation entry points— triggers in world world, time since last speech, every piece of state.
- Totally interruptable, VERY state dependent.
- This needed to feel very natural, with her remembering all she said

# Case Study: Day 2 Julia Conversations (Take 1)




<<PATRICK>>

- Interruptions here became really tricky because if thread didn't finish than internal state could be wrong. Delilah would know things the player didn't.
- Once we had the structure in place for the facts and the trigger volumes, most of our issues were with surrounding conversations.
- This is the use case that ended up getting us Crit Path conversations, that couldn't be halted. This is where we had to formalize this pattern.

# Case Study: Day 2 Julia Conversations (Take 2)



<<PATRICK>>

- System used properly, since we had 3+ big revisions on the teens.
- The creation of Delilah and Henry brains
- Multiple Conversation entry and exit points
- This needed to feel very natural, with her remembering all she said
- Show the actual event lists in question (maybe make a state graph?) Show the facts names for the Delilah Brain.
- Interruptions here became really tricky because if thread didn't finish than internal state could be wrong. Delilah would know things the player didn't.
- Once we had the structure in place for the facts and the trigger volumes, most of our issues were with surrounding conversations.
- This is the use case that ended up getting us Crit Path conversations, that couldn't be halted. This is where we had to formalize this pattern.

# Keeping your facts straight

- Sort events by number of requirements
- Take first matching
- Most specific response is best

Event Window | Scene | Game

Event Editor DelilahCallsDay2 All

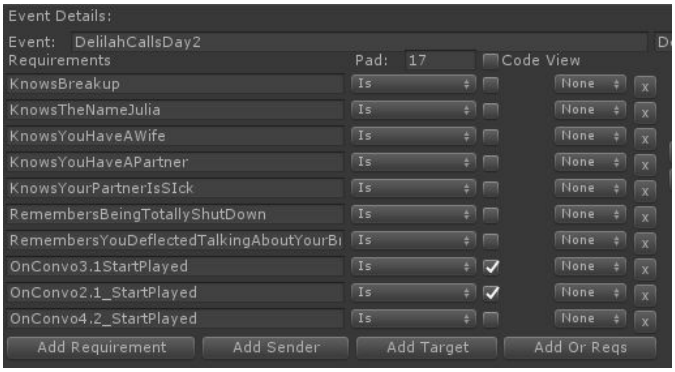
| | | | |
|---|---|---|---|
| DelilahCallsDay2___19 | Julia Convos Sorter | TriggerEvent | IsDialogChoice |
| DelilahCallsDay2___18 | Julia Convos Sorter | TriggerEvent | IsDialogChoice |
| DelilahCallsDay2___21 | Julia Convos Sorter | TriggerEvent | IsDialogChoice |
| DelilahCallsDay2___23 | Julia Convos Sorter | TriggerEvent | IsDialogChoice |
| DelilahCallsDay2___26 | Julia Convos Sorter | TriggerEvent | IsDialogChoice |
| DelilahCallsDay2___27 | Julia Convos Sorter | TriggerEvent | IsDialogChoice |
| DelilahCallsDay2___24 | Julia Convos Sorter | TriggerEvent | IsDialogChoice |
| DelilahCallsDay2___17 | Julia Convos Sorter | TriggerEvent | IsDialogChoice |
| DelilahCallsDay2___20 | Julia Convos Sorter | TriggerEvent | IsDialogChoice |
| DelilahCallsDay2___25 | Julia Convos Sorter | TriggerEvent | IsDialogChoice |
| DelilahCallsDay2___20 | Julia Convos Sorter | TriggerEvent | IsDialogChoice |
| DelilahCallsDay2___14 | Julia Convos Sorter | TriggerEvent | IsDialogChoice |
| DelilahCallsDay2___22 | Julia Convos Sorter | TriggerEvent | IsDialogChoice |
| DelilahCallsDay2___15 | Julia Convos Sorter | TriggerEvent | IsDialogChoice |
| DelilahCallsDay2___16 | Julia Convos Sorter | TriggerEvent | IsDialogChoice |
| DelilahCallsDay2___11 | Julia Convos Sorter | TriggerEvent | IsDialogChoice |
| DelilahCallsDay2___12 | Julia Convos Sorter | TriggerEvent | IsDialogChoice |
| DelilahCallsDay2___13 | Julia Convos Sorter | TriggerEvent | IsDialogChoice |
| SinceLastJuliaConvo___1 | Julia Convos Sorter | TriggerEvent | |



<<WILL>>

Callback to Goons Bark system
Instead, indy film about romance and friendship

# Keeping your facts straight




"You're not one of those guys who is building the "Great Plan" to get her back while you're out here, are you?"

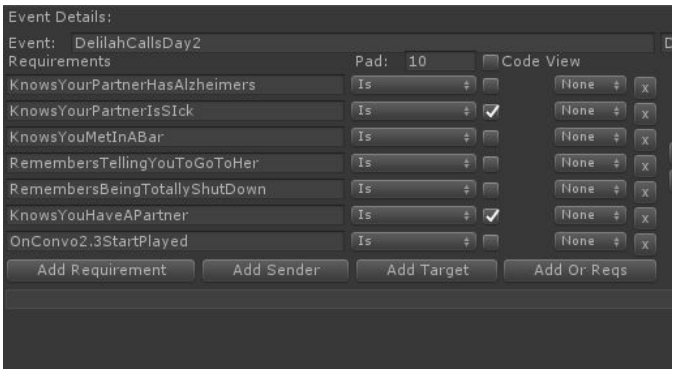


<<WILL>>

Padding lets design prioritize as needed

This line only plays when Delilah knows nothing about Julia.

# Keeping your facts straight




"What does she have?"



<<WILL>>
Here she knows about Julia, knows she isn't well, but not why.

All these different conversation lines have the same basic start, and we just drill down through game state until we find the most perfect line we have for your playthrough.

# The Conversation Problem

- Conversation Problem
- Non-linear content
- Modal State

*Prevent Interruptions*

*Line 1*

*Line 2*

*Stop Preventing Interruptions*



<<WILL>>

Conversations as a First Class Citizen. Critical Path convo hell.
We used start and stop events, with ref counting, to turn on and off normal player control. Ideally, we would have had a nested structure that allowed for the creation of a conversation that would fire off certain events on start and on end, no matter what the details of that conversation were. Really just a specialized tool for automatically adding the correct responses to all the right places.

# The Conversation Problem

- Conversation Problem
- Non-linear content
- Modal State

*Prevent Interruptions*

*Line 1*

*Dialog Choice*

*Line 2*

*Stop Preventing Interruptions*

*Line 3*



<<WILL>>

Alternatively, if you jump from before Line 1 to Line 2, you are also in a bad place, never allowing the player to start talking ever again

# The Conversation Problem

- Conversation Problem
- Non-linear content
- Modal State

*Prevent Interruptions*

*Line 1*

*Dialog Choice*

*Line 2*

**Stop Preventing Interruptions**

*Line 3*



<<WILL>>

Alternatively, if you jump from before Line 1 to Line 2, you are also in a bad place, never allowing the player to start talking ever again

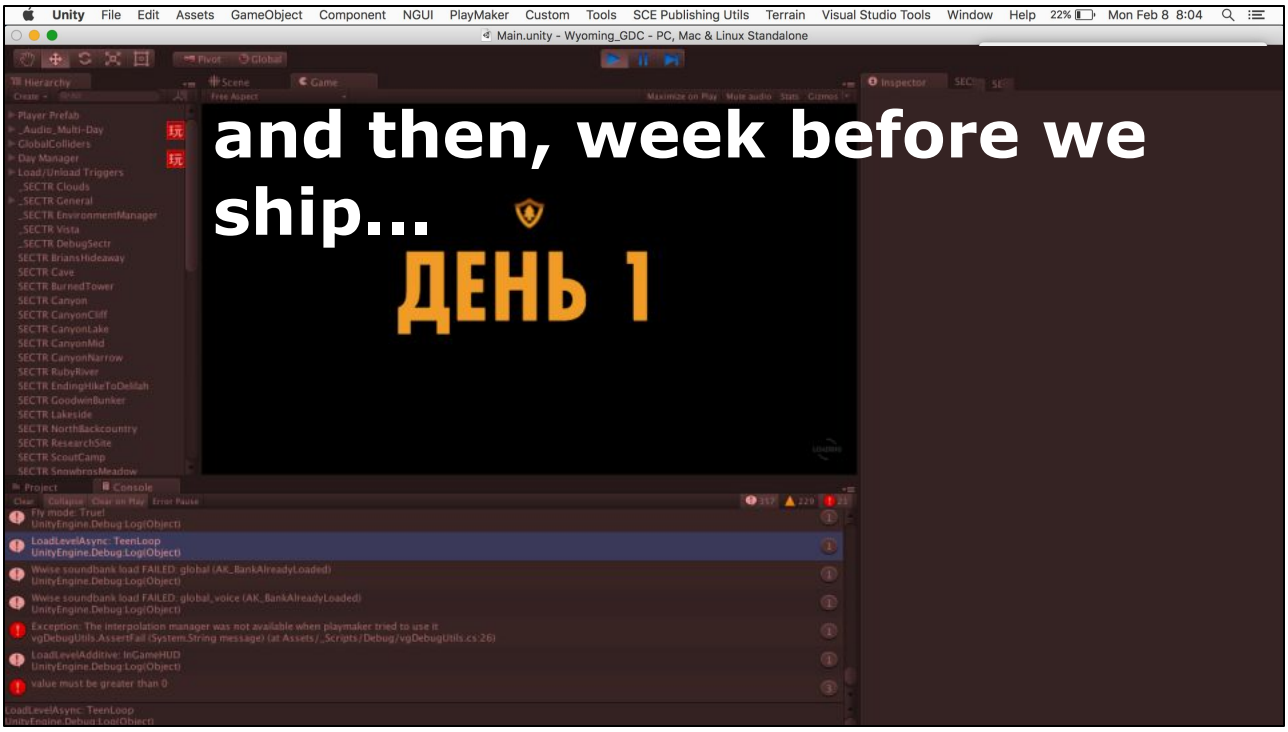
# The Conversation Problem

- Flexible tools are fragile
- Letting you do anything means you must do everything
- Keep experimenting even when things break
- Learn best practices, then automate



<<WILL>>

Touch on the problems with our system here




<<WILL>>

- SURPRISE! We decided to ship localization

**ENGLISH**
It's actually pretty damn cold out here.

**MËTAL**
Ìt's áçtúállÿ prèttÿ dámñ çòld òút hèrè.

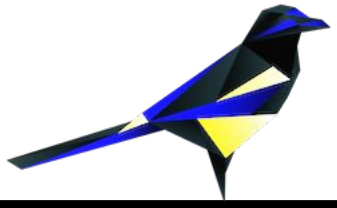
**WIIIDE**
It's actually pretty damn cold out here. (mn cold out here)

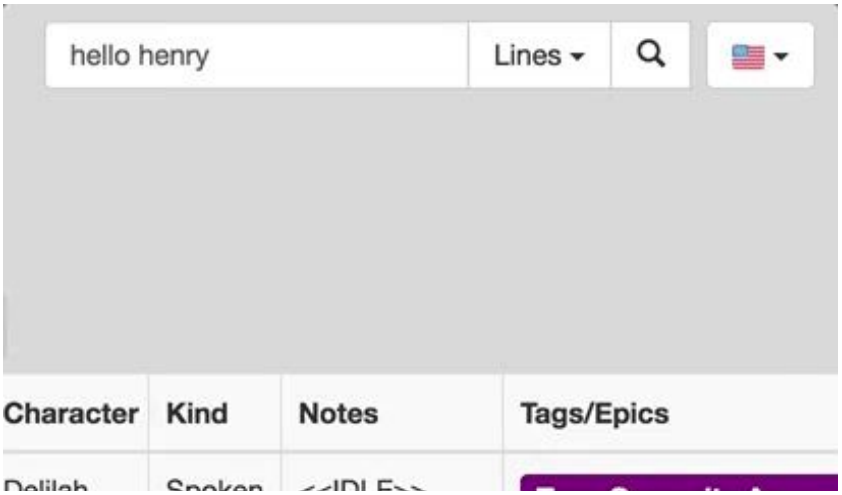
# Pseudo Languages: **Mëtal & Wiiide**



<<PATRICK>>

# We quickly added a localization interface to Magpie…

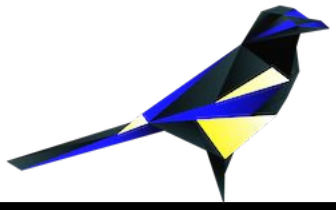







<<PATRICK>>
Mention that we were working with Slava, our amazing Russian translator, to pull this rabbit out of a hat

# We quickly added a localization interface to Magpie…






<<PATRICK>>
Mention that we were working with Slava, our amazing Russian translator, to pull this rabbit out of a hat

# Localization Runtime

- Fields tagged with [vgLocalizeString]
- Dump all strings to JSON as a [English <-> English] dictionary
- Add [English <-> Language] values



<<WILL>>
Already in place!
Already tested-ish
Use C# reflection to find all tagged fields
We had a component on some UI fields that were directly input that would track down the text component.
Otherwise everything was simply a attribute
vgLocalizeString worked on strings and string enumerations
Loc builds were able to be generated manually, but they also just build automatically with every build, so you could never get the data out of date.

# Localization Schemata


[English <-> Spanish]

[This cant possibly go wrong <-> Esto no puede salir mal]



<<WILL>>
Tricky because -

Multi-User editing
Make a range of ID's, and grab them per user
Hash the string - DANGER

# Localization Schemata


[English <-> Spanish]

[This can't possibly go wrong <->
???? - No entiendo]

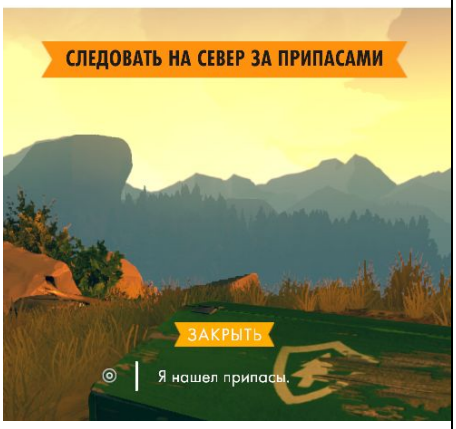


<<WILL>>
Tricky because -

Multi-User editing
Make a range of ID's, and grab them per user
Hash the string - DANGER

# Localization Schemata

We could have done better!






<<WILL>>
When to bake in assumptions - String IDs for localization.

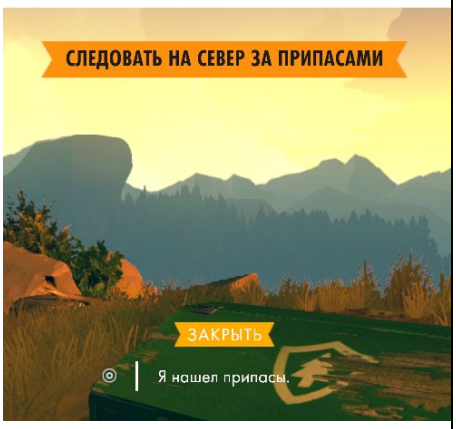
English <-> English dictionaries for translation are easy to read, and great if you are planning on having fan's loc the game.

However, once we decided to go with professional localization, we should have switched to using a UniqueID per string, so that we could alter the English without breaking localization. Everytime we fixed a typo in the English, we broke the localization. We were already doing somethign like this for dialog lines and audio files. Just didn't have time at the end.

# Localization Schemata

We could have done better!

'Loc Lock' - Lock the Keys in your dictionary. Translate [Locked English <-> Proper English]






<<WILL>>
Tricky because -

Multi-User editing
Make a range of ID's, and grab them per user
Hash the string - DANGER

# Localization Schemata

We could have done better!

- ‘Loc Lock’ - Lock the Keys in your dictionary. Translate [Locked English <-> Proper English]
- String IDs - Generate a Unique ID for each line. Works perfectly! But is tricky to implement.



<<WILL>>
Tricky because -

Multi-User editing
Make a range of ID’s, and grab them per user
Hash the string - DANGER

# Localization Success!

But maybe we should have waited..

Patch Notes

- 2/9/16 - Fix several small issues with Russian localization
- 2/10/16 - Fix many issues with the Russian localization
- 2/16/16 - Fix several Russian localization bugs
- 2/18/16 - Fix a few issues with the game showing English when set to Russian
  Fix issues switching language while playing the game
- 2/19/16 - Fix a few missing Russian
- 3/16 - Update some strings in



<<PATRICK>>

Game shipped with support for Russian

First full localization test on Sunday

Game launched that Tuesday

# Lessons Learned- Tools

- Small team
- AAA quality with Indie Scope
- Tools supported our iterative design process
- Built this complexity over time



<<PATRICK>>

One year later - What went write
Small team
AAA production with indie scope
Iterative Design Process
Change Tolerant
Organically built up complexity over time

# Lessons Learned - Systems

- Assume nothing
- Build for flexibility early
- Build for special cases with real content

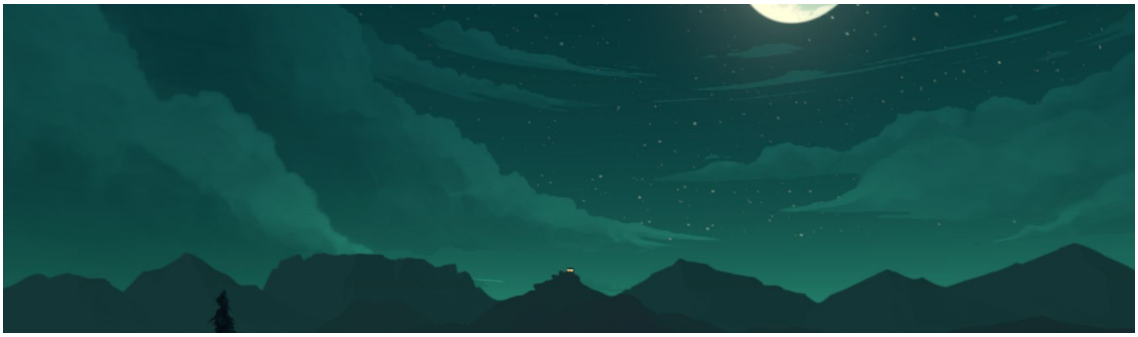


<<WILL>>

- Make general purpose tools early
- Specialize as needed, but only as needed.
- Factor in time to re-do the tool / editor after your first milestone
- Work with content producers, and make some content yourself.

# Further Work

## Conversations as building blocks





<<PATRICK>>

Conversations as a First Class Citizen. Critical Path convo hell.
We used start and stop events, with ref counting, to turn on and off normal player control. Ideally, we would have had a nested structure that allowed for the creation of a conversation that would fire off certain events on start and on end, no matter what the details of that conversation were. Really just a specialized tool for automatically adding the correct responses to all the right places.

# Further Work

- Automate follow up lines
- Set state on enter and exit
- Know the state that should exist per line

*Prevent Interruptions*

*Line 1*

*Dialog Choice*

*Line 2*

**Stop Preventing Interruptions**

*Line 3*



<<PATRICK>>

We should have had a ‘Conversation’ type that would serve as a statefull container of lines. Any lines in that conversation set the state needed by that conversation. Any time the dialog system stops playing any lines in that conversation, clears all state. Automatically generate the busy work of making a series of back and forth or follow up lines

All of this only makes sense for conversations, and would need to be bolted on top of the generic, flexible event system.

# Further Work

- Automate follow up lines
- Set state on enter and exit
- Know the state that should exist per line

```
Conversation 16
Interrupt = FALSE
  Line 1
  Dialog Choice
    Line 2
    Line 3
```

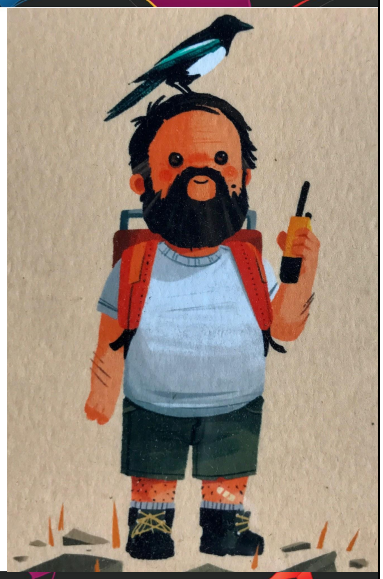


<<PATRICK>>

We should have had a ‘Conversation’ type that would serve as a statefull container of lines. Any lines in that conversation set the state needed by that conversation. Any time the dialog system stops playing any lines in that conversation, clears all state. Automatically generate the busy work of making a series of back and forth or follow up lines

All of this only makes sense for conversations, and would need to be bolted on top of the generic, flexible event system.

# OVER & OUT!



William Armstrong
williama@unity3d.com
@WillWArmstrong

Patrick Ewing
patrick@chanceagency.com
@hoverbird

# Q & A

William Armstrong
williama@unity3d.com
@WillWArmstrong

Patrick Ewing
patrick@chanceagency.com
@hoverbird