Artemis

Artemis User Documentation

Dev Build 2.27.2023

Written by Nicholas Perell

Table of Contents

Introduction

Hello! This document is for anyone looking for a more in-depth explanation of *Artemis* than what's included in the base README.md. Here you will find:

- Who and **what *Artemis* is built for.**
- The same **file-by-file explanation** you can find in the README.md.
- A step-by-step **pipeline for using *Artemis* in a project;** from spreadsheet, to asset generation, to archer, and thence delivery.
- **Optimization tips** for the programmers (or conscientious designers and producers).
- A message of **encouragement** to programmers: ideas on how to mess with *Artemis*'s code for your own uses.
- **Credits and special thanks.**

The biggest goal is to both inform you as a developer of how to get *Artemis* working in your projects, and inspire you to use it in creative and unique ways. All I hope to do is make it something useful for other folks, let them focus on design and writing while *Artemis* manages a good amount of the backend. I wish you luck, and

May your aim be true.

What *Artemis* is *For*

For games where the order of who you talk to or what you do is variable, *Artemis* accesses rules and world state data to give the most appropriate and important delivery. It's not about the means of delivery, like *Ink* or *Yarn Spinner*, but instead about deciding what should be delivered.

Artemis took inspiration from *Hades*'s priority queues[1], *Firewatch*'s Delilah brain[2], and *Left 4 Dead 2*'s Dynamic Dialog[3].

[1] People Make Games's video on *Hades*
[2] Chris Remo's 2019 GDC talk on *Firewatch*
[3] Elan Ruskin's 2012 GDC talk on Valve's games

What's Included

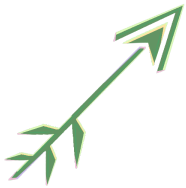
- The base code for *Artemis*'s...
    - Arrows for each deliverable narrative beat, with ID's, priority values, and what criteria need to be met.
    - Archers, which use the priority values (and when they were added to the archer) to determine which of a group of arrows should be delivered.
    - Arrow Bundles, which can be prompted to dump into (or drop from) archers of the designer's choosing.
    - Flags, which store the values that are evaluated for an Arrow's criteria.
    - Flag Bundles, which bundle up flags into groups, to load and unload them as necessary to optimize the process.
    - Fletchers, which parse .CSV's to generate the databases full of the relevant information needed to direct the...
    - Bows, which are the monobehaviors that use the incoming data to deliver the narrative.
    - Goddess (Narrative System), which tracks all the true/false flags the arrows use.
- Debug Example for how the code can be used...
    - Example .CSV files.
    - Children scripts of the Fletchers & Bows.
    - Scene demonstrating *Artemis* in action.

File-by-file Explanation

Although one of the best ways to get an understanding of *Artemis* would be to check out the examples made, it's worth documenting the target purpose of each of the previously listed items. A good way to imagine it:

> "The *fletcher* makes and stockpiles the *arrows*, and the *archer* decides which arrow to shoot. The archer can get more arrows from (or throw away some in) an *arrow bundle*, and she uses her *bow* to fire them. An arrow checks if certain *flags* are met to consider it appropriate to use, and *flag bundles* can be loaded as needed to optimize the process."

Arrows



Stores the most basic information for each possible piece of narrative delivery. This includes:

- ID: used to access the database found in the fletcher it is connected to.
- Priority Value: int value used by the archer. Can use the "COND" keyword to get the number of criteria in the rule.
- Rule: flags that must be set to specified criteria (otherwise the arrow will be skipped over by the archer)
- How to handle busy: if the archer tries to fire the arrow, but the bow is busy, what is done? There are a couple options:
    - CANCEL: Retreat! Return the arrow back to the archer that chose it.
    - QUEUE: Add the arrow to a queue and wait until the fletcher/bow gets around to it.
    - INTERRUPT: Abruptly stop what the delivery actor is doing to deliver a narrative beat, and have it do this one.
    - INTERRUPT_CLEAR_QUEUE: Same as above, as well as clearing out the queue that was there.
    - DELETE: Don't play it, but don't return it to the archer. If the Archer discards arrows, this'll discard the arrow entirely.
    - FRONT_OF_QUEUE: Similar to the queue, but make it cut to the front of the queue.

Arrows can also be prompted to fire on their own without going through the process of being in an archer. This is useful for cases where an arrow is fired in response to a different arrow finishing.

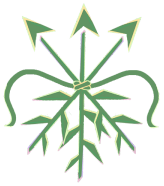
Archers

This is what tries to choose which arrow should be shot. Arrows with a priority of zero are placed in a "general pool" with a random order. Higher values are given priority above lower values. The data presented to the user includes:

- Default Contents: the arrows the archer has on hand when she's initialized.
- Decision Making: the logic of the archer, as set by the developers.
- Handling Same Priority: when there are arrows of the same priority value greater than zero, how does the archer determine which arrow to shoot?
    - QUEUE: First of those arrows to go in, the first of those arrows to go out.
    - STACK: Latest of those arrows to go in, the first of those arrows to go out.
    - RANDOM: Choose which of those arrows at random.
- Discard Arrows After Use: is an arrow removed from circulation after being used?
- Loops: when the archer is out of any arrows to consider, is there a refresh that occurs?
    - Include Bundles: if refreshing includes the arrow bundles' dump and drop history.
    - Include Higher Priorities: if refreshing includes arrows with a priority of greater than zero.
- Arrow Bundles: the history of dropping (adding) or dumping (removing) arrows from an Archer is saved.
- Partitioning Flags: an important piece of optimization the developers can decide to use. By choosing SYMBOL flags to be in every single arrow an Archer will consider, the arrows can be divvied up into separate tables.

The archer is prompted by calling `bool AttemptDelivery(FlagBundle[] importedStates, FlagID[] all = null)`. `importedStates` is the list of flag bundles evaluated for considering arrow criteria. Any FlagIDs in `all` will be skipped over and assumed as being met—the use being it could essentially allow *any* character to respond (instead of just one), making for self-branching conversations.

Arrow Bundles



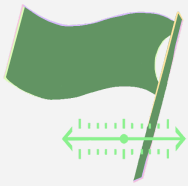
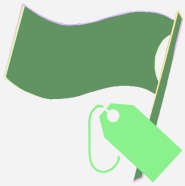
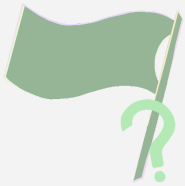
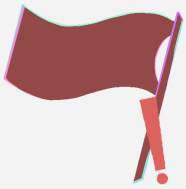
List of arrows. Can be dumped into or dropped from an archer. These dumps are where Handling Same Priority on an archer is very important.

Flags

Objects which store the values that are evaluated for an arrow's criteria. Data stored in each flag include:

- ID: enum value used to sort flags.
- Value: this value is internally stored as a float, but can be presented a variety of ways:
    - FLOAT: a number.
    - BOOL: true or false.
    - SYMBOL: an enum value.

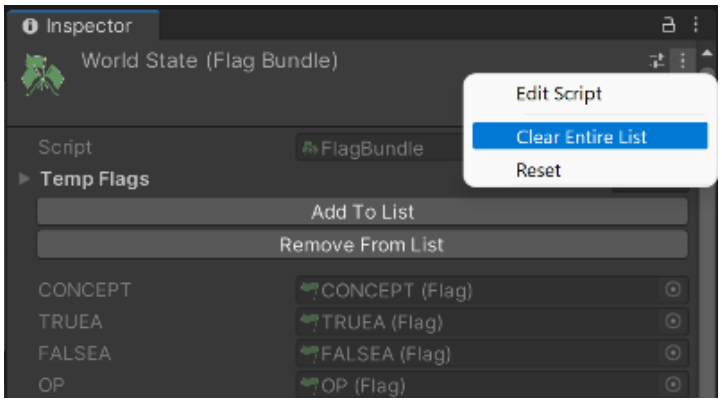
When in the inspector, the flag uses variant icons to indicate what type of flag it is.

	**FLOAT.** Flag presents its Value this way.
	**BOOL.** Flag presents its Value this way.
	**SYMBOL.** Flag presents its Value this way.
	**UNKNOWN.** Flag ID is not set to something valid. Usually seen when creating a new flag asset for the first time.
	**ISSUE.** The flag thinks it needs to present the stored Value a certain way, but the Flag ID isn't set to something valid. Common cause of this may be the Flag ID of this flag has been deleted or modified.

Flag Bundles

Sorted list of flags. Flags in these lists can't have the same flag ID. Loading these in or out helps speed through the evaluation process on arrows.

If there are any null items in your flag bundle, you can try to remove an empty list. If there are missing items (you deleted the flag assets before removing it from the bundle) the previous solution doesn't work on, you can clear the bundle in the context menu ⋮.



Fletchers

Artemis's base fletchers script is an abstract template class, where you will want to define:

1. The information that needs to be stored in a database for the bow to deliver the narrative how you want it.
2. The `bool SetUpDataFromCells(string[] dataToInterpret, out T valueDetermined)` function that validates the string array intake from the .CSV and uses those strings to generate the information that needs to be stored.
3. The length of the string array. Based on the value of an int named columnsToReadFrom.

Bows

Another whose base script is an abstract template class. The typing on the template class should be the same as the fletchers you want it to work with. This is where things go from decision to full delivery.

To properly set up a script for a delivery actor:

1. Define `void Send(T data)`. Using this data, how does this gameobject facilitate delivery.
2. Define `bool IsBusy()`. Is the actor still in the middle of delivery?
3. Define `void AbruptEnd()`. If the fletcher wants to interrupt with a new arrow being sent, how do you wrap up what's going on?
4. When you're done delivering, be sure to call `ReportEnd();`. This allows the fletchers to see if there were any arrows with QUEUE or FRONT_OF_QUEUE stored for later.
5. If you define `OnEnable()`, be sure to call `base.OnEnable();` in the function.

When attaching the bow monobehavior to a game object, make sure the Fletcher in the inspector is set to the fletcher you want the actor to be paired with.

Goddess (Narrative System)

Singleton that facilitates the flag IDs. Found at `Window/Artemis Goddess`.

The narrative system keeps track of if flag IDs are being used by any of the arrows generated by the fletchers. By default, if a flag IDs has not a single arrow checking it for being true or false, that flag ID will be deleted. However, in the inspector the

narrative system has a Flag IDs to Keep array. Flag IDs in this array will not be deleted by this scrubbing.

Another important value the Goddess has is the Globally Loaded Flag Bundles array. All attempts at delivery from archers or arrows will take the flags here into account.

Using *Artemis*

CSV Layout & Formating

For what the format of the .CSV's should be like, here is an example format on Google Sheets. You're encouraged to make a copy and use it as a basis for yours. Make sure to use the `Dev: Artemis Debug Console` sheet, as the other sheet uses the old formatting from version 0.1.

ID	Priority Va	Flags	How to handle busy	Debug Message	Log Type	Time (s)
Test_Debug_000	0	Concept = Life, TrueA, !FalseA, 100 > Op >0, EQ = 0, 100 >= CI >= 50, 50 > OC >= 0, 50 >= CO >0, WHO = Marcus	CANCEL	Test 0	DEFAULT	0.3
Test_Debug_001	0	Concept = Life, TrueA,TrueB_1, !FalseA,!FalseB_1, WHO = Marcus	QUEUE	Test 1	DEFAULT	0.3
Test_Debug_002	3	Concept = MomDad, TrueA,TrueB_2, !FalseA,WHO = Liam,!FalseB_2	INTERRUPT	Test 2	ERROR	0.3
Test_Debug_003	2	Concept = Reloading, TrueA,TrueB_3, !FalseA,!FalseB_3	DELETE	Test 3	DEFAULT	0.3
Test_Debug_004	1	Concept = Reloading, TrueA,TrueB_4, !FalseA,!FalseB_4	FRONT_OF_QUEUE	Test 4	WARNING	0.3
Test_Debug_005	1 + COND	Concept = Idle, TrueA,TrueB_4, !FalseA,!FalseB_4	INTERRUPT_CLEAR_QUEUE	Test 5	DEFAULT	0.3
Test_Debug_006	COND	Concept = Idle, Who = Liam, TrueA,TrueB_4, !FalseA,!FalseB_4	QUEUE	Test 6	DEFAULT	0.3

Make sure the **ID** values are unique. Don't make any of the ID values equal "END."

In the **Priority Value** column, COND is a stand in for the number of flags that need to be met. In the above screenshot, `Test_Debug_006` will have a priority value of `6`.

For the **Flags** column, you can use the values for ranges (e.g. "50 > OC >= 0"), symbols (e.g. "WHO = Marcus"), and bool (e.g. "TrueA", "!FalseA"). The flags are converted into internal symbols, and are *not* case sensitive. "TrueA" and "TRUEA" will be understood to mean the same thing. *Artemis* will not accept using a flag for multiple value types (e.g. "Who > 10").

How to handle busy defaults to `CANCEL` if it can't recognize the input.

Test_Debug_030	2	Concept = Life, Who = Nick, TrueA,TrueB_4, !FalseA,!FalseB_13	QUEUE	Test 30	DEFAULT	0.3
Test_Debug_031	2	TrueA, Who = Bob, !FalseA, Gr > 0,Eq = 5, Le < 4, GE >= 5, LE<= 5, 100 > Op >0, 100 >= CI >= 50, 50 > OC >= 0, 5	QUEUE	Test 31	DEFAULT	0.3
END						

Include the END at the bottom line.

In Google Sheets, you can download a specific sheet as a .CSV file. This is the file you can then bring into your project to be used by *Artemis*.

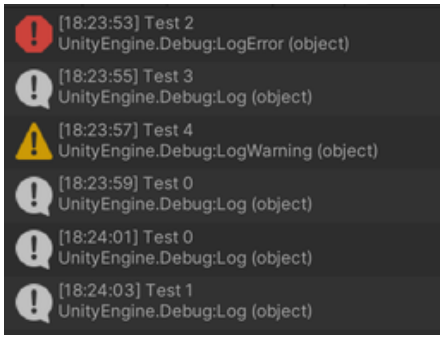
Namespace

The code for *Artemis* uses `namespace Artemis`. The example code uses `namespace Artemis.Example`. For your own code, it is instead recommended to have the line `using Artemis;`.

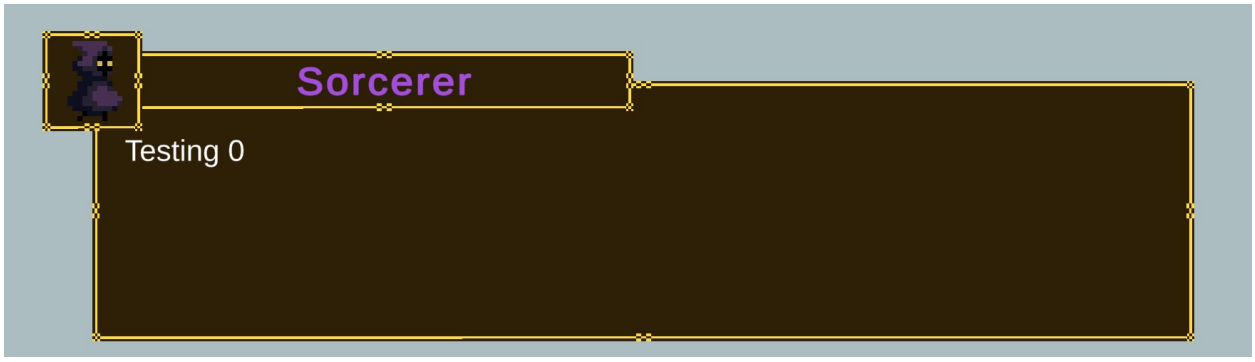
Making Your Own Fletcher & Bow

Most of what there is to say about the Fletcher & Bow are in the file-by-file explanation. Additionally, here are some links that might be helpful:

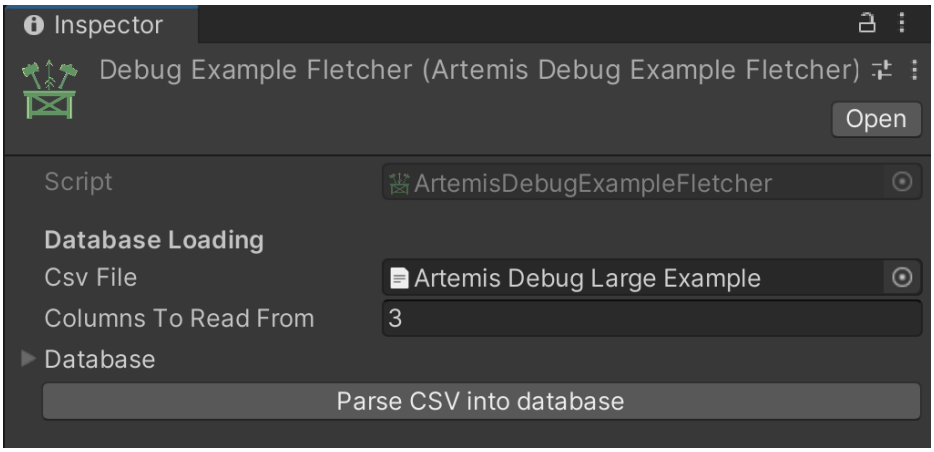
- Example Fletcher & Bow that sends debug messages with a delay timer before the next one can be sent.



- Example Fletcher & Bow for a conversation involving dialogue boxes. Used for *Rituals*, an example project for *Artemis* currently in the works.



Using a Fletcher



Be sure to set the Columns To Read From to the correct value. This is the number of columns in the CSV used to generate the data structures in each database. Number does not include the base 4 columns.

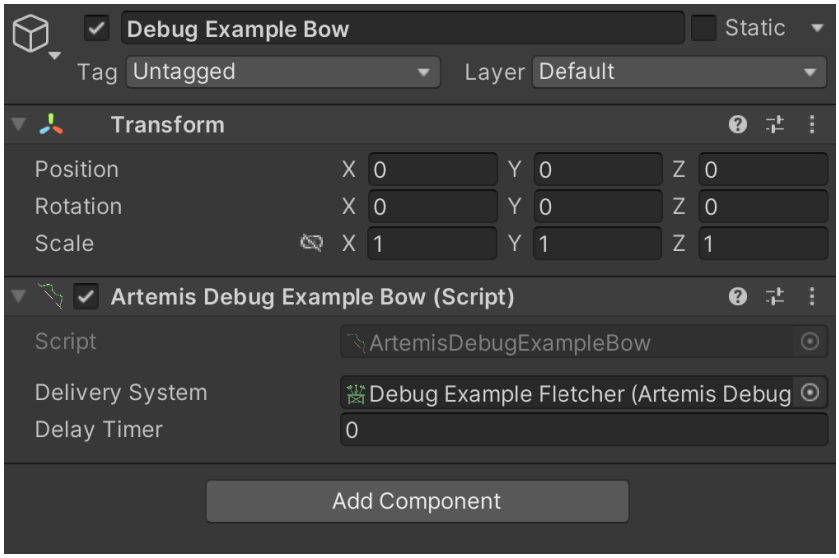
To generate the arrow (and the database), click the button once the CSV and Columns To Read From value are in there correctly.

Creating Archer, Bundling, and Flag Assets

Right click the project window, mouse over “Create,” and then over “Artemis.” From there, you’ll see the options to create the assets you are free to on your own.

Firing the Arrows

As mentioned in the file-by-file explanation, the archer is prompted by calling `bool AttemptDelivery(FlagBundle[] importedStates, FlagID[] all = null)`. The Debug Example’s scene has a script prompting it, as well as initializing the archer at start.



Additionally, ensure in the scene that there's a game object with the custom Bow component, and the Bow has the correct Fletcher dragged into its `Delivery System` variable.

There you have it!

Optimization

Artemis has been made with many built-in optimization features, as well as ways you can optimize the game as needed for yourself. Some built-in optimization includes:

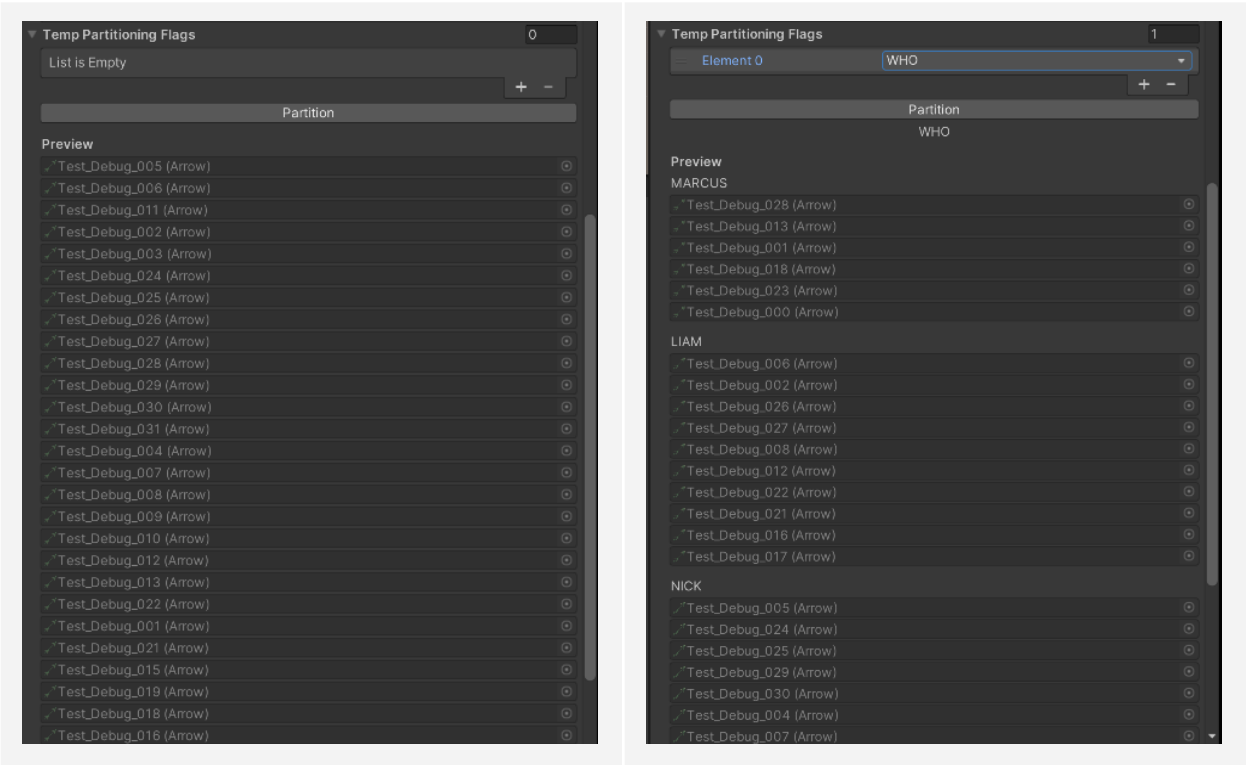
- Compiling Arrow IDs, Flag IDs, and values used by SYMBOL flags as enums instead of strings.
- Partitioning arrows in an archer using an integer array for the key instead of a concatenated string.
- Sorting by Flag ID internally so evaluating an arrow runs faster.
- Turning any criterion check into the same range function.

More important for this document are the ways users can optimize their use of *Artemis*.

1) Partitioning Flags for Archers

Mentioned in the file explanation for archers, setting up partitioning flags is an important piece of optimization the developers can decide to use. By choosing SYMBOL flags to be in every single arrow an archer will consider, the arrows can be

divvied up into separate tables. Now there's less arrows that have to be evaluated by the archer, speeding up the process.



This one example is from just using one SYMBOL flag to partition. More than one SYMBOL flag can be used to partition the arrows up even further.

What sort of SYMBOL flag(s) every arrow used by an archer would use will be specific to each project.[4]

2) Bundles

Similar to why it's useful to partition the arrows used by an archer, it may be appropriate to only add arrows in when they might be relevant, and to remove the arrows once they no longer are. This is one of the purposes of **arrow bundles**.

The other sort of bundle, **flag bundles**, can affect the speed of every single evaluation of an arrow by being there or not. If the player is in an underwater cavern off the coast of New Zealand, none of the relevant arrows are going to check the number of people waiting in line to ride the Cyclone at Coney Island. Don't load the Coney Island-specific flags in the first place.

[4] According to Elan Ruskin's 2012 GDC talk on Valve's games, in *Left 4 Dead 2*, "who" and "concept" would be good examples.

3) Cleaning Up the Enums

Artemis keeps track of the enum usage for the IDs of arrows and flags, deleting the unused ones from the enum script. This is not the case for SYMBOL flags. It's worth looking at their enum scripts to make sure a typo'd item or since-deleted character isn't taking up space.

Another thing recommended to do early on is set the most important flag IDs to the smallest values. This lets the linear searches for these values run as fast as possible. Be mindful that editing its numbers in the script will require going to the arrows or flags that use that flag ID and changing it there, too.

Encouragement to Programmers

Artemis was initially a 6-week project back in the spring of 2022. Version 0.1 was based on the narrative pipeline and systems I created for a small game called *Project Nautilus*.[5]

Version 0.2 took a longer period of time, and a lot more deliberate work. In the process of making the package more robust, some of the scripts were completely redone. One of the things I learnt during development was that sometimes the best thing we can do as programmers is to know who we're making something for. Try to empower your narrative team.

The writers on that game were comfortable using Google Sheets, but that might not be for everyone. If working in a Google Sheet and exporting a .CSV file isn't a good pipeline for them, the code base is there for a reason. I encourage you to get messy, to rip apart that Fletcher code, and make it import data the way they want it to. There are no wrong answers, and every project is different. Hopefully *Artemis* gives you a good place to start.

Credits & Special Thanks

Artemis is an ongoing narrative programming project by Nicholas Perell.

CSV Parsing Scripts contributed by Brandon "bb" Boras.

Art made by Crystal Wong.

[5] Due to a sudden upset in the development timeline for the narrative team, we never got to use it to any worthwhile potential. The game is out there in the wild, but don't play it expecting any sort of demonstration of an *Artemis* predecessor.

A special thanks to Scott Barret, Max Anderson, Shannon Mitchell, Aaron Pastor, and Eric Harvey for their support and advice.

A special thanks to Elan Ruskin, Chris Remo, Chris Bratt, Anni Sayers, Greg Kasavin, and Amir Rao. Your videos and talks both inspired and informed this project. You have my gratitude for sharing your insights with those around you.