Runarc: A Parser-Free Interactive Fiction Framework for the Web Browser[1]

Augusto Goulart[2]

Tuesday, 26th of May – MMXXV

This document describes the overall architecture of the Runarc interactive fiction framework. It also provides the structure of objects and comments on how to use it. Only a brief overview is presented here; some discussion on similar tools, story writing, and interactive fiction as an art form is also included.

[1] Source code not publicly available.

[2] Contact me at the address `a_goulart (at) icloud (dot) com`

Interactive fiction (IF) writers tend to be unacquainted with programming, as most IF projects are text-based choose-your-own-adventure (CYOA) games. These projects have a heavy focus on story-building and are analogous to CYOA books; for that reason, tools available for IF writing are prone to shy away from any programming.[3] This is the case for most popular tools like ChoiceScript, ink, Twine – although possible to insert JavaScript code; and Inform.

Runarc is the opposite of IF-making tools like Inform 7 – to a tee. The goal of this framework is to keep the story development as close as possible to the code; whereas other tools parse either a near-natural language text or a markup script, with Runarc the code and the story are one and the same. This allows the inclusion of further visuals and effects directly from the story's flow, which in turn creates a wider distinction between IFs and text-based games.[4]

But, why? Pourquoi? Waarom? Por que? I'm glad you've asked, dear reader. This framework is not meant for those who want to start making IFs; it is not meant for seasoned programmers; and, it is not meant for seasoned IF writers either. Runarc is meant for projects with a distinct artistic vision, where all the story-telling of text-based games meets with the visual possibilities of web browser technologies.

As for its implementation, Runarc was coded with TypeScript. This choice was due to my familiarity with type-strict languages and not because of any hard requirement. Runarc is built and packaged with Parcel, which allows the usage of other pre-processing plugins such as Sass. The newer SCSS syntax – from Sass – is used to generate all style sheets of Runarc.

However, there are limitations that are purposefully placed on Runarc based on my personal inclinations – such as not being compatible with mobile devices. Those are implementation-specific and are not required for the architecture of the framework, neither for it to be functional.

[3] Taking both tasks alone and with no pre-written story is a burden most won't dare pick – for good reason; it is better to focus on the story.

[4] Text-based games, at their earliest, were made with CLI-like interaction; nowadays, most of us are familiar with visual novels, an evolution of that concept – which are often mislabeled as interactive fictions.

Disclaimer: No part of this document or code of this project has been written or created with generative AI; it is my personal belief that all generative AI is anti-human and therefore unethical – more so than just plagiarism.

Architecture

Events

Events are the smallest logic component of the framework. They can be either *(a)* linear events, that **cannot** branch the story; or *(b)* choice events, that *can* create branches in the story. All other events are derived from these two objects.

LINEAR EVENTS — such as in figure 1, may only have one "result" event. This will be the next event once the linear event is finished; it cannot be skipped. The result may be any event or none (i.e. `null`). Derived events include: textual events, meant to display text to the reader; visual events, that change the document's elements or styling; and, data events, which update the story's internal values (e.g. stats).

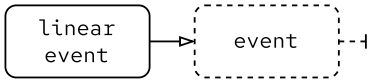

Figure 1: Linear events cannot branch.

CHOICE EVENTS — may have any number of "options". These are used to *(a)* check this choice condition (i.e. whether or not this branch is available to be chosen); and *(b)* define this choice's result – which may also be any event or none. A three branch choice event can be seen on figure 2. Derived events include: pause events, that create a break in the story's flow, until the reader proceeds – it has a single option; branch events, meant to have two or more story branches.

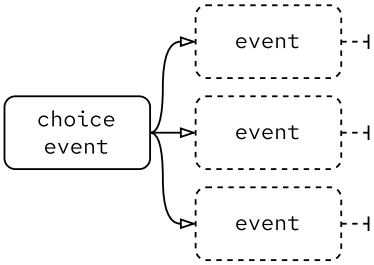

Figure 2: A choice event that starts three branches in the story.

Event Tree

The most important data structure of this architecture is the event tree. It represents everything that *might* happen in a specific part of the story. This allows Runarc to not have a parser for its story events.

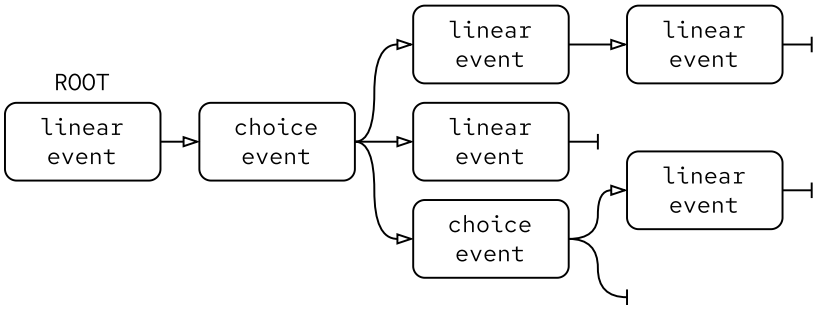

Figure 3: An event tree; its starting point is a linear event. *Note: this is an unbalanced tree, here represented horizontally.*

To do that, the event tree is populated recursively at *construction* from its starting point (i.e. root); see figure 3. Since *(a)* every event is derived from the same object; and *(b)* every event must have a result at *construction*, `null` or otherwise[5]; then: every subsequent event from a given start point must be defined in code.

[5] Not to be confused with *undefined*, a `null` value is still a defined value.

Event Stack

To allow "backtracking" from a story branch, it requires a event stack – see figure 4. This is a path from the current event down to the root of the event tree (i.e. starting point). However, to fully integrate this feature, every event's logic must have a *do* and *undo* function, to go forwards and backwards, respectively.[6] Once gone backwards, current progress is erased and the reader is presented with the last "control" event in the stack. This may be any event, as defined by the story's writer.

[6] This is particularly important for events that change internal data values.

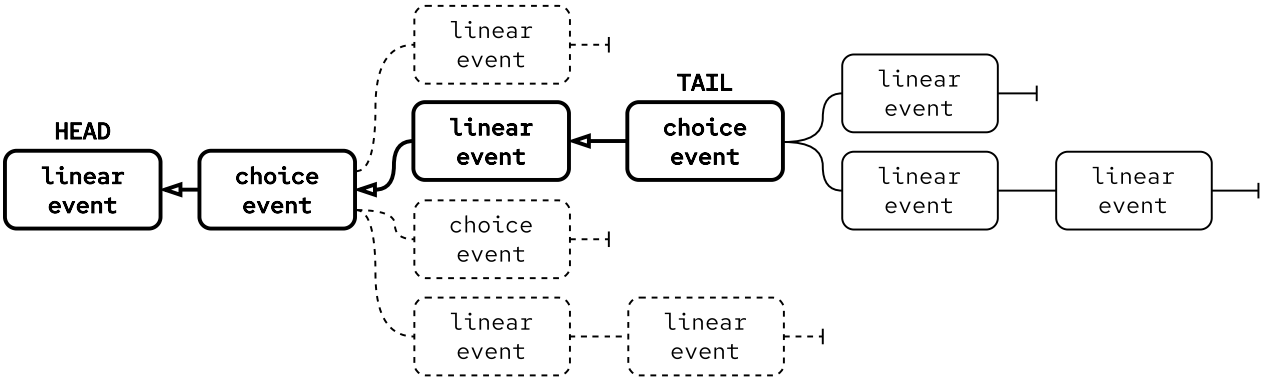

Figure 4: This event stack (in bold) is currently at the third level of the event tree. You can see the future branches in solid lines and past branches in dashed lines.

Scenes

To control the story's flow, it must be divided into manageable chunks. Since we cannot define labels and jump from one to another, as is the case in parsed stories, then we must define an object to handle the underlining logic; this object is a scene. Each scene has an event tree object – think of it like a **decision** tree; it also has an event stack object, to keep track of the readers **path**. Now, do you remember that every event has a result `Event` or `null`? Here is why that matters: a `null` result represents the end of the scene. Each scene is a *state* in a *FSM*.

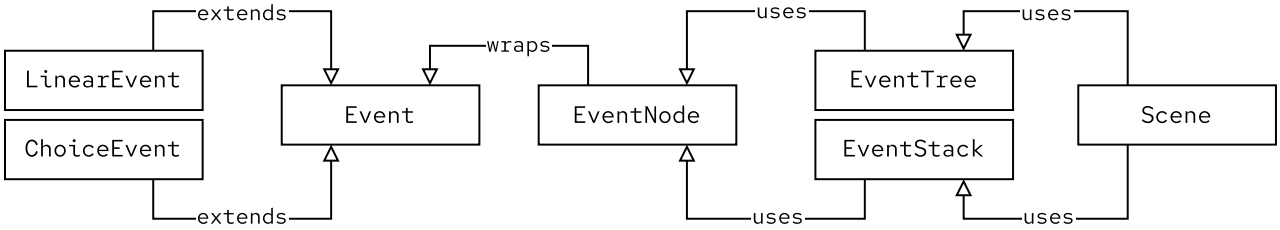

Figure 5: An overview of Runarc's architecture from a scene's point-of-view.

Data Flow

To allow the use of *global* stats throughout the story, Runarc provides a stats manager. Each scene can safely update any stat and create new ones if *future* conditions are met.

Data Persistence

One of the most important questions for web-based IFs is: *"how will I make sure the reader's progress is saved for later?"* – All global stats and current state of the story's flow can be stored on device, this is done in Runarc by using *IndexedDB*.[7]

[7] Some framework-specific values might be stored using *WebStorage*.

Reader Interaction

"Hold up! That's a lot of technical stuff; but how does the reader interact with the novel?" – That is the neat part: **they do not**; this is a lie of course, what I mean is: they do not interact with Runarc. All interactions with the reader are up to the IF writer, even for story events; it is implementation-specific.

Visual Identity

Runarc uses Kaushan Script for its typeface and a "stroke" animation for its splashscreen. This animation consists of each stroke for the letters being drawn out one at a time. Only its first 'R' should be capitalized; it is not an acronym.

Runarc

Considerations

Use Case

This framework is being used to create The Dossier, an IF adaptation of the 1867 crime fiction le Dossier n° 113 by Émile Gaboriau. The first step is to translate the original text from French to English – which I am having way more fun with than I should. There is an English translation on public domain, but it uses British (and old) vocabulary. I plan to write about that process in the future.

Accessibility

As mentioned before, reader interaction is implementation-specific, it is up to the writer to ensure their story's visuals and logic is consistent with accessibility standards.

Source Code

This project has been designed and implemented for use within my own interactive fictions and will remain closed-source; if you are interested in maintaining an open-source version of this project, then feel free to contact me.

Acknowledgments

I would like to recognize how much I have been inspired by the interactive fiction community and the many stories I have read. Without their inspiration I would not have made Runarc, neither would I have taken a step into this art form. And for that – and much more – I say: thanks to all of you![8]

[8] To you too, reader ;)