Demeter: An Implementation of the Marlinspike Interactive Drama System

Zach Tomaszewski & Kim Binsted

Information and Computer Science Department,
University of Hawaii—Manoa, Honolulu, HI 96822
(ztomasze@hawaii.edu & binsted@hawaii.edu)
http://zach.tomaszewski.name/argax/

Abstract

Demeter is a text-based prototype game that demonstrates the Marlinspike architecture for producing computer-based interactive dramas. Marlinspike uses a top-down, directed approach to generating stories: a central drama manager agent responds to story-significant user actions by selecting the next scene to play from a pre-authored collection. This drama manager strives to explicitly reincorporate past user actions into the story in order to meet an Aristotelean definition of narrative unity.

Introduction

Marlinspike is designed to produce a *directed interactive drama*. An *interactive drama* is a system in which the player assumes the role of a character in an unfolding story and, through their actions within the virtual story world, influences the outcome of that story. Marlinspike is *directed* in that it includes a centralized drama manager that attempts to choose the next story event in such a way as to produce a well-formed, coherent plot that includes the user's actions. This is in contrast to a purely emergent approach, in which a story is meant to emerge solely from the player's interactions with autonomous character agents or the static rules of the story world.

Marlinspike Architecture

A Marlinspike drama takes place within a particular *story world*—a collection of simulated objects, including *locations*, *props*, and *characters*. The player dictates the actions of one of these characters (the *player character*, or PC); the other *non-player characters* (NPCs) are controlled by the drama manager.

The player interacts with the objects of the story world by selecting one of a number of *verbs* to perform. Verbs specify simple physical actions, such as taking, dropping, kissing, or talking. Based on the current story context, each verb is then translated into one or more *actions* that represent what that verb signifies within the current story. For example, a kiss may have different "meanings" depending on the context: it could be an attempt at romance, it could be done simply to break a magical enchantment, or it could be a Judas-like betrayal.

The Marlinspike *drama manager* (DM) then responds to user actions by selecting the next *scene* to play. Whenever possible, scenes are selected so as to reincorporate user actions into later story events. This reincorporation forms a story *thread* of connected events that makes earlier user actions narratively necessary to the finished story.

The details of this architecture—particularly as

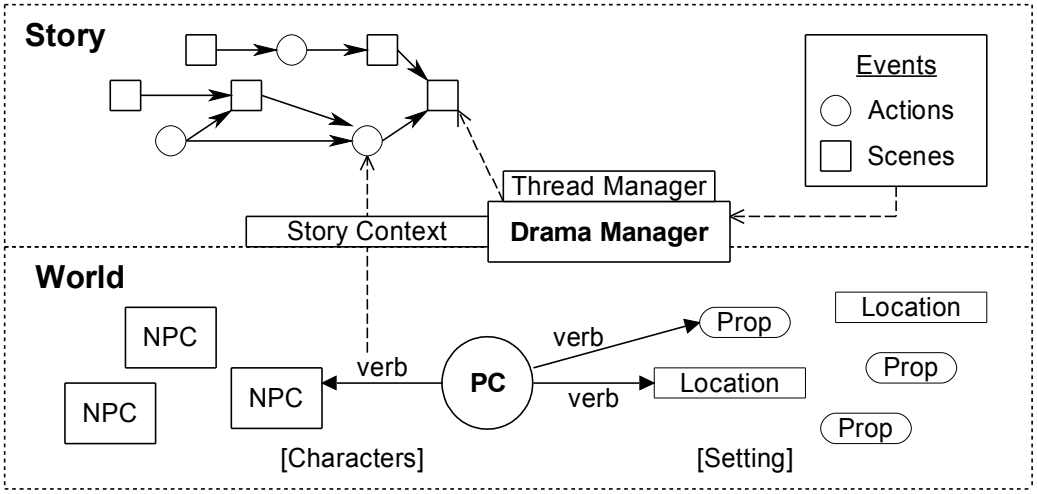

Figure 1: Overview of the Marlinspike architecture

implemented in the current prototype—are described below.[1]

Story World: Characters and Objects

Objects in the Marlinspike story world are simply bundles of state and invoked behaviors. For examples, props include such state as their current location in the world; they may also define their own specific responses to being affected by different verbs.

Similarly, non-player characters are not autonomous agents, but simply puppets of the drama manager. NPCs are comprised of a number of attributes, including *morality* (how altruistic or self-interested that character is) and *affinity* (how much they like or dislike another particular character). These attributes affect how certain verbs are translated to actions and how scenes are selected (see below). NPCs also contain their own set of potential conversation responses, which can be used by scenes to customize the story's narration depending on which particular NPC has been selected to perform the events of that scene.

World-Level Events: Verbs and Deeds

As the Marlinspike implementation is currently text-based, example player commands might include `take tennis ball`, `go north`, `talk to Alice`, or `punch Fred`. These illustrate the verbs `Take`, `Go`, `Talk`, and `Attack` (for which "`punch`" is a synonym). When we include the direct object each verb affects in the story world, we produce a number of world-level events called *deeds*: `Take(ball)`, `Talk(Alice)`, `Go(n_obj)`, `Attack(Fred)`. Occasionally, a deed requires a second object, as in: `Show(Alice, ball)`.

The physical outcome of deeds is usually presented to the player as a single sentence of narration, such as "Taken." or "You take a swing at Fred... but he ducks your blow!"

Story-Level Events: Actions

Every deed is then translated into a story-level *event* based on the current story context. These *action* events are story-level representations of player deeds. Action events have a structure like that of deeds that includes both the particular action and its affected objects or characters.

Through a processes called *casting*, every verb has a default translation to an action. For example, the verb `Drop` generally translates to the MANIPULATE action.

This default translation may take the current world-state into account. So the deed `Kiss(Alice)` should translate to the event ROMANCE(Alice) only if Alice already has a high affinity for the player. If Alice detests the player, this same deed would be better translated to ASSAULT(Alice).

[1] The Marlinspike architecture, as described here, was originally presented in (Tomaszewski & Binsted 2008), although it has been refined somewhat since then. Please see this earlier work for more on Marlinspike's theoretical inspiration as well as a comparison to other interactive drama systems.

Occasionally, this default casting can be completely overridden by the story context. For instance, if Alice has been transformed into a frog, then `Kiss(Alice)` should perhaps instead become RESCUE(Alice).

A deed can also be translated into more than one action. In this case, the extra action events form of a tree of sub-events called *recasts*. So, if the player has already established a girlfriend relationship with the character Betty, Betty might take offense at this kissing of Alice. This secondary effect would be appended as a recast, producing the following structure:

RESCUE(Alice)
|— OFFEND(Betty)

Actions have effects associated with them. Since only deeds change the "physical" state of the world, action effects are generally limited to character state changes. So RESCUE(Alice) will probably increase Alice's affinity for the player, and possibly increase the player character's morality value.

Unlike deeds, actions do not produce any narration. After each event has occurred, it (including all of its recasts) is appended to an *event history* list, which forms a transcript of story-level events that have occurred so far. Once recorded there, the DM can respond to the action with a scene.

Story-Level Events: Scenes

Scene events—the other kind of story event—represent a bundle of narration and world-level manipulations made by the DM. Scenes are pre-authored components that serve three purposes in Marlinspike. The first is to provide reactions to player actions. While verbs provide narration of their world-level effects, only scenes provide narration of NPC responses and other story-level effects. The second purpose of scenes is to advance the story by introducing new incidents and material. Finally, scenes can provide a story context that may affect later verb-to-action translations (as previously described above).

Every scene has a list of preconditions that determines whether it can currently be performed and appended to the story-so-far. Preconditions might include the current story-world time, character locations, prop states, character attributes, or previous story events.

While pre-authored, scenes can be heavily templated so that specific details are filled in at run-time. For example, a scene could be written to handle any character's reaction to being ASSAULT-ed by the player. This could be written generically enough to occur at any location, and even the particular reaction can be customized with dialog or other details pulled from the reacting NPC or the event history. Similarly, any character with a strong grudge against the player—due to some earlier event—could be cast to appear in a scene requiring a betrayal of the player.

Besides preconditions that must be true for a particular scene to run, scenes can also include *hooks*, which are previous events that the scene can refer to or otherwise

reincorporate if they have occurred. This aids in threading (described below).

Each scene belongs to one of three *functions*. *Beginning* scenes have no preconditions and are selected by the drama manager to start a new story. The bulk of scenes are *middle* scenes, which serve to advance the story in some way. *Ending* scenes have preconditions, but provide a conclusion to the current story.

Like a paragraph in a novel, scenes vary in how much action they present. A scene may be only a single line of dialog delivered by an NPC in response to a question asked by the PC. On the other hand, a scene may be paragraphs long or summarize the passage of hours. Regardless of length, no user input—and thus no deeds or actions—are possible during a scene. Like actions, completed scenes are appended to the story event history.

Story Context: Triggers

Scenes can add to the story context by introducing a *trigger*. A trigger simply watches for the later occurrence of a certain verb or action and can then interrupt or append to its translation into an action.

For instance, in an interactive Bluebeard fairy tale, a scene could have Bluebeard tell the player not to open a certain closet. The scene then ends, but it would also add a single trigger to watch for the `Open(closet)` deed. Should this deed occur, the trigger can then recast the default MANIPULATE(closet) event to include a DEFY(Bluebeard) sub-event. Now both the MANIPULATE and the DEFY actions can provide material for future scenes. Certainly such a fairy tale would include a scene that provides a response to DEFY-ing Bluebeard.

Sometimes a scene may need to establish a more complex context than can be managed with only one or two separate triggers. In this case, a *trigger bundle* can be used. Whenever a bundled trigger fires, it also alerts the bundle so that the state of the other related triggers can updated if necessary. Such bundles can be used to represent important story-level states, including *roles*. For example, a scene may cast an NPC into the role of a Friend. This Friend role may then serve as a precondition or a hook for certain scenes. However, this role state is essentially self-managing in that it sets recast triggers for conditions that would end the role—such as if the player severely affronts the Friend character.

Narrative Unity: Threads

It is not sufficient for an interactive drama system to simply provide believable responses to separate player actions. Such an approach may present a very believable virtual world, but it will frequently fail to produce a coherent story that also includes the player's diverse actions in that world. Therefore, Marlinspike strives to weave both user actions and authored scenes together to produce a finished story that meets an Aristotelean definition of narrative unity. Specifically, Aristotle claims the events of a story must be connected by necessary or probable cause. These causal connections produce a single narrative Action that is both whole and complete, such that

> if any [part] is displaced or removed, the whole will be disjointed and disturbed. For a thing whose presence or absence makes no visible difference is not an organic part of the whole (Aristotle 1961, Chapter VIII).

Taking inspiration from Keith Johnstone's (1979) work in improvisational theater, Marlinspike attempts to form such a unified Action by *reincorporating* previous events. If an earlier event *A* is required before the current event *B* can occur, then we can say that event *A* is necessary for event *B*. Therefore, Marlinspike attempts to select the next scene so that it makes necessary to the story as many important earlier events as possible. For example, suppose that, for no apparent reason, the player chooses to insult an NPC in a bar. Even if the NPC reacts immediately by getting upset and storming off, the player's action has not yet had a significant effect on the story if that NPC does not appear again. However, if the system responds at some point later in the story by reincorporating this action—for instance, by having the now-antagonistic NPC turn out to be the father of the PC's new girlfriend—then the player's INSULT action is made necessary to the finished story: it is a direct result of the insult that the player cannot now take his new girlfriend out on a date.

There are two primary ways to form reincorporation connections between events in Marlinspike. The first is through a scene's preconditions and hooks: whenever a scene requires or refers to a previous event in its own narrated details, it is considered to be reincorporating that earlier event. The second form of reincorporation occurs through story context triggers: if an earlier scene set a trigger that is involved in casting or recasting the current action, then it means that the earlier event has some direct relevance to the current action. We saw each of these kinds of connection in the earlier Bluebeard example. (See Figure 2.) Such a sequence of reincorporated events is called a *thread*.

Marlinspike does not attempt to reincorporate every action, however. Every event—whether scene or action—includes a pre-defined import value which suggests how dramatically exciting it is. For instance, the MANIPULATE action (which represents such verbs as `Take`, `Drop`, and `Push`) has a very low import while the MURDER action has a very high import. Though it reincorporates whenever it can, Marlinspike is only required to reincorporate events of

Figure 2: A simple thread demonstrating the two types of reincorporation.

high import. In addition, all of an action's recasts (sub-events) are considered to part of that event; therefore, the whole event is considered to be reincorporated if any of its sub-events is reincorporated.

Thus, a story starts with a beginning scene. This initial thread can be extended by some action building upon a context established by this scene. However, the player might instead start a new thread by performing some unrelated action of high import. In response, the drama manager will select the next scene from those that it can perform. It will choose the scene that extends the most threads of high import, giving some preference to first replying to the most recent action. Even if the drama manager cannot immediately reincorporate the most recent action, it may be able to do so later in the story.

When a scene plays, it will thread all the previous events that it can. If the last event of more than one thread can serve as preconditions for the scene, then the drama manager can effectively *splice* two (or more) threads together into one thread. Contrarily, if the system cannot currently extend any existing thread, it may be forced to *fork* an earlier thread event, thereby creating a new thread which it will hopefully be able to splice back in later.

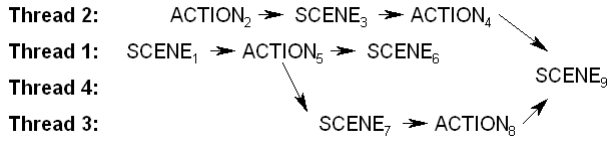

Figure 3: Thread map demonstrating a fork (producing Thread 3 from Thread 1) and a splice (producing Thread 4 from Threads 2 and 3)

In the Figure above, $SCENE_6$ is still pending. If its thread (Thread 1) is not extended, then the story is poorly-formed, as it contains unnecessary events. However, this evaluation is tempered by the rule that not every action or scene needs to be spliced into a thread, but only those of high import.

A story may end as soon as the drama manager can meet the preconditions of an ending scene. However, if the drama manager can play a different scene that will extend threads of high import that would have been left pending by the ending scene, it will continue the story instead.

Demeter: Blood in the Sky

Demeter is our prototype game intended to demonstrate the feasibility of Marlinspike. It will also be used to formally evaluate the notion that reincorporation increases the perceived coherence of an interactive story. We will use the interactive fiction system Inform to define the story world objects and to process user input. Therefore, the playing experience will be very much like that of interactive fiction, where the user types in commands in natural language and the system responds with text describing the results of their actions. The exception to this will be conversations with characters, which will be menu-driven to improve system affordances in the absence of any natural language processing beyond Inform's existing parsing capabilities.

The story is set in an alternate 1923. Seven affluent passengers—one of whom is the player's character—leave England on a four-day trans-Atlantic flight aboard the Zeppelin-class airship *Demeter*. However, just before the dawn of the third day, the passengers are roused from their beds by the captain on the intercom. The captain urges them to bar the hatchway to their passenger gondola, for someone... or some*thing*... is loose within the narrow walkways of the Zeppelin above. The crew is dead, and the captain himself is succumbing to blood loss. As the intercom falls silent, the passengers are left to decide what to do—and what to believe—as the *Demeter* drifts slowly westwards over the Atlantic.

The number of verbs required for *Demeter* will be comparable to interactive fiction games—about forty or so. These will translate to approximately twenty different actions. Scenes will vary in size, but most will result in about a half-screen of text. We expect to need about sixty scenes to produce an average story of twenty scenes in length; such a story should take about thirty minutes to play. Demeter has only a single beginning scene and about six possible ending scenes.

Conclusion

Marlinspike's design strives to balance strong narrative control with high user agency. The user is immersed in a story world of simulated objects and characters. Their deeds in this world can have different meanings and effects based on the preceding story context, which is managed through a very simple mechanism of triggers. The Marlinspike drama manager then furthers the plot by using customizable, pre-authored scenes, working to reincorporate the user's significant actions and thereby make them necessary to the resulting story. The game *Demeter* is a specific implementation meant to demonstrate Marlinspike's functionality in practice.

References

Aristotle. *Poetics*. Trans. S. H. Butcher. Ed. Francis Fergusson. New York: Hill and Wang, 1999.

"Inform." <http://www.inform-fiction.org/inform6.html>

Johnstone, Keith. *Impro: Improvisation and the Theatre*. New York: Routledge, 1979.

Tomaszewski, Zach, and Kim Binsted. "Marlinspike: An Integrated Approach to Interactive Drama." *Integrating Technologies for Interactive Stories: Papers from the INTETAIN 2008 Workshop.* Playa del Carmen, Mexico. 7th Jan 2008. <http://zach.tomaszewski.name/argax/pubs/2008-TomaszewskiBinsted-Marlinspike.pdf>