Marlinspike: An Integrated Approach to Interactive Drama

Zach Tomaszewski & Kim Binsted
Information and Computer Science Department,
University of Hawaii Manoa, Honolulu, HI 96822
http://zach.tomaszewski.name/argax/
ztomasze@hawaii.edu & binsted@hawaii.edu

ABSTRACT
Marlinspike—a directed interactive drama system—generates stories by interleaving story-significant user actions with a collection of pre-authored scenes, guided by a simple neo-Aristotelean model of story structure. Its design integrates elements from a number of previous interactive narrative architectures, as well as ideas from improv and table-top role-playing games. This design intends to strike a better balance between user agency and system control, thereby overcoming certain limitations of its predecessors.

Categories and Subject Descriptors
J.5 [**Arts and Humanities**] – *fine arts;* **I.2.1** [**Artificial Intelligence**]: Applications and Expert System – *games*.

General Terms
Design, Theory

Keywords
Interactive narrative, interactive drama, directed narrative, Marlinspike.

1. INTRODUCTION
The Marlinspike architecture 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, can influence the outcome of that story. By *directed* we mean that the system has some form of centralized control over the story world. This centralized *drama manager* component works to direct the story, thus generating a well-formed, coherent plot that incorporates the user's actions. (This is in contrast to an *emergent* approach, in which the story is meant to emerge solely from the player's interactions with autonomous character agents or the static rules of the story world.)

Here we describe the basic design of Marlinspike, and then explore how this both builds upon, and yet differs, from previous approaches.

2. MARLINSPIKE DESIGN
In brief, Marlinspike translates user-entered, world-level *verbs* into story-level *actions*. It then responds to these actions and advances the story by selecting a *scene* from a pre-authored collection. This dialog between system and user produces a *thread*—a series of causally-related scenes and actions. Threads are eventually woven together to form a complete story. The details of this process are as follows.

2.1 Story World
A Marlinspike drama takes place within a particular *story world*—a collection of simulated objects, including *locations*, *props*, and *characters*. Each object supports certain relevant world-level actions, such as being picked up, opened, dropped, etc.

Marlinspike characters are not autonomous agents, but simple puppets of the drama manager. Characters 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 characters are cast into roles (see below). Characters also contain their own set of potential conversation responses.

2.2 Verbs
Marlinspike includes a number of predefined *verbs*, which are used by the player to interact with the objects of the story world. Example input from the player might include `take tennis ball`, `go north`, `talk to Alice`, or `kill Fred`. These illustrate the verbs `Take`, `Go`, `Talk`, and `Attack` (for which "`kill`" is a synonym). When we include the direct object each verb refers to in the story world, we produce a number of *commands*: `Take(ball)`, `Talk(Alice)`, `Go(n_obj)`, `Attack(Fred)`. Occasionally, a command requires a second object, as in: `Show(Alice, ball)`.

When processing user input, the system first determines whether the resulting command is possible. For example, the tennis ball might be nailed to the floor or Alice might not be in the room, making the input `take ball` or `talk to Alice` impossible to complete. Every valid command (and its component verb) is then translated to one or more *events* (each with its component *action*; see below) based on the current story context.

2.3 Actions
Actions represent story-level actions. Every defined action includes an *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. On the other hand, the MURDER action (which might be produced by the verb `Attack`) has a very high import.

Actions are the primary components of *events*, which have a sentence-like data structure similar to commands. That is, an event may include the object that receives the effects of the action, as well as any second object necessary to the event. So an event is simply the occurrence of a particular action, including any object(s) directly affected by that action.

Every verb translates, by default, to an action. For example, `Drop` generally translates to MANIPULATE. The default translation may take the current world-state into account. So the command `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 command would be better translated to ASSAULT(Alice).

However, within the context of a particular scene, this default translation can be overridden. For instance, if Alice has been transformed into a frog, then `Kiss(Alice)` should instead become RESCUE(Alice).

A command can also be translated into more than one event. In this case, the extra events form of a tree of *sub-events*. 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 sub-event, producing the following structure:

RESCUE(Alice)
|— OFFEND(Betty)

Actions also have effects associated with them. This can include world or character state changes. So RESCUE(Alice) will probably increase Alice's affinity for the player, and possibly increase the player's morality value. Actions also provide some textual narration or graphical presentation, so the player can be made aware of the action's effects.

After each event has occurred, it is appended to an *event history* list, which forms a transcript of story-level actions that have occurred so far.

Like verbs, actions are player-centric. That is, the subject of an event—the character that undertakes the action—is always the player's character. The actions of non-player characters is instead handled solely through scenes.

2.4 Scenes

Scenes are pre-authored components that serve two purposes in Marlinspike. The first is to advance the story. Since characters are not autonomous and all story control is centralized in the Marlinspike drama manager, it is only through scenes that characters respond to player actions and the story is advanced by the introduction of new material. Secondly, as we've seen, scenes provide a story context that may override some of the default verb-to-action translations.

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

Scenes can be either *durative* or *instant*. Durative scenes introduce some event in the story and then remain active, preventing further scenes from occurring until its own ending conditions have been met. This is handy for introducing complex choices that have many possible player responses. For example, if a vampire offers the player immortal undeath, the scene may want to remain active so it can determine what actions corresponds to the player accepting, fleeing, or fighting the vampire.

Instant scenes introduce some event and then end. This is best for providing character reactions to previous events or for story elements that don't need much control of the story context.

Instant scenes can still add to the story context by introducing a *trigger*. A trigger watches for the later occurrence of a certain verb and can then interrupt or append to its translation into an action. For instance, in an interactive Bluebeard fairy tale, an instant scene could have Bluebeard tell the player not to open a certain closet. The scene then ends (giving other appropriate scenes a chance to play), but it would also add a single trigger to watch for the `Open(closet)` command. Should this command occur (even within another scene's context), the trigger can then append the DEFY(Bluebeard) sub-event to the resulting default MANIPULATE(closet) event.

Alternately, a trigger can be set for an action instead of a verb. So, in a previous example, the player's jealous friend Betty took offense at the `Kiss(Alice)` command, even though Alice was an icky frog that needed disenchanting at the time. If we wanted to make Betty more forgiving, we could set her trigger for the ROMANCE action instead of the `Kiss` verb.

Aside from being durative or instant, scenes also belong 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.

Scenes are represented by a SCENE action, and so every scene becomes an event (with no subject or direct object) that can be appended to the event history. If a scene is durative, its record may include, as sub-events, the player actions that occurred within its context.

2.5 Roles

A scene or action can also cast characters into certain *roles* specific to the particular story. For instance, a scene might establish that a princess has been imprisoned by an evil wizard. If the player then rescues the princess, the RESCUE(princess) action

may cast the princess into a Friend role and the evil wizard as an Enemy.

In many ways, roles simply serve as a short-hand notation for all the possible actions or scenes that could produce a certain relationship or state. For instance, there might be a number of ways to make a Friend in the game—only one of which is RESCUE-ing them. But once the player has a Friend (by some means), that relationship can serve as an precondition for a number of later scenes that require a Friend.

Additionally, roles serve to separate relationships from character attributes. For example, the player might be greatly esteemed by a number of characters in the story world; that is, they all have high affinity for the player. Yet a Friend role can only be cast through a significant story event. Therefore, if a later scene has misfortune befall a Friend character, it should be more significant and relevant to the story than if that same misfortune simply befell one the player's many fans.

Like scene triggers, roles can also interrupt verb-to-action translation, as we've seen with Betty's jealous Girlfriend role in previous examples.

2.6 Threads

Events—arising from both user-entered commands and system-selected scenes—can be connected in terms of necessity. That is, if one event must precede another, we can say that the first is necessary for the second.

In terms of the Marlinspike system, these connections of necessity can be equated with preconditions, scene context, and the casting of roles.

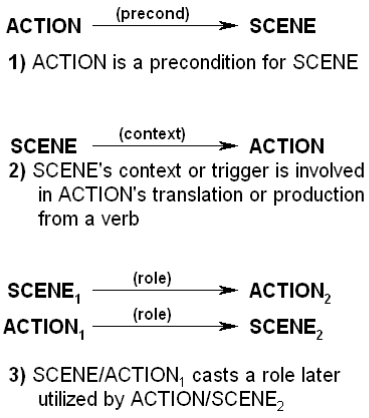

Figure 1: Three kinds of necessity

So, if a scene lists a certain action in its preconditions, then that action is necessary for the scene. Similarly, if a verb is translated into an action based on the currently-active durative scene or based on a trigger from a previous instant scene, then that scene's context is necessary for the action. Finally, if an action or scene changes, ends, or otherwise requires a role, then that role (and whichever scene or action originally cast that role) is necessary for the current action. For instance, if a scene requires that the player already have an Friend, the earlier action that resulted in making that Friend is necessary for the scene to occur.

These three simple rules allows us to plot *threads* of necessity through the events of the story. So, to return to the Bluebeard example, a scene may establish that the player is not to open a certain door and then sets an appropriate trigger to enforce this context. When that trigger produces the DEFY(Bluebeard) event, we can say that the earlier scene was necessary for the DEFY-ing. In turn, the DEFY(Bluebeard) action can now serve as a precondition for an enraged Bluebeard scene.

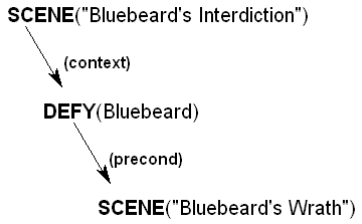

Figure 2: Necessity in the Bluebeard example

Threads serve to incorporate events into a coherent storyline. As mentioned, a storyline starts with a *beginning* scene. This thread can be continued by some action building on a role or context established by this scene. However, the player might also start a new thread by performing some unrelated action of high import.

The drama manager works to extend an existing thread with the next selected scene. If the last event of more than one thread can serve as preconditions for the next 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. (See Figure 3.) This thread-extending behavior is the source of the *Marlinspike* name, which refers to the splicing and fancy rope work of marlinspike seamanship.

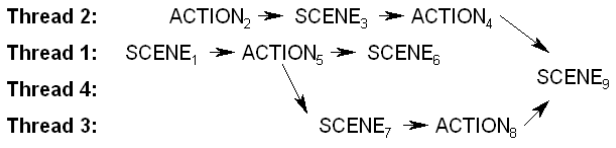

Figure 3: Thread map demonstrating a fork (producing Thread 3 from Thread 1) and a splice (producing Thread 4 from Threads 2 and 3)

In the above example, $SCENE_6$ 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.

3. THEORETICAL FOUNDATION

Marlinspike is based on a neo-Aristotelean poetics [10], which specifies that an interactive drama can be defined in terms of multiple levels. Its highest level—Action—represents the significant events of the story. These events materially rely upon the underlying level of the story world—comprised of Characters and Setting—while at the same time also formally specifying that story world's nature (See Figure 4).

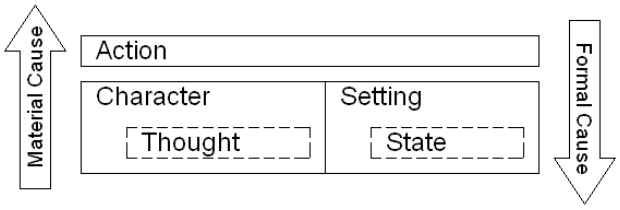

Figure 4: Partial poetics model of story (see [10] for more)

As a character in the story world, the player interacts through verbs and commands. The alternative—letting players specify actions and events directly—would make the user more of a director than an actor in the drama. It is hoped that using verbs will increase the player's sense of playing a character within the story. It also allows multiple possible ways to achieve the same story-level action. For example, the player might RESCUE(princess) by breaking down her door, picking the lock, or magically teleporting into her cell.

Marlinspike scenes are partly inspired by the *encounters* of table-top role-playing games [3]. An encounter is an event structured in if/then terms—if the players do this, then this will happen. Sometimes this is can be as simple as "if the players enter this room, this monster will attack them." Just as all encounters are linked together to form an *adventure*, Marlinspike's scenes are strung together as threads.

These threads, when taken together, meet Aristotle's definition of a plot as being whole and complete, such that no part can be removed without leaving the whole "disjointed and disturbed" [1]. Specifically, Aristotle claims that the unity of the plot comes from its parts being connected by necessary or probable cause. He goes on to define the beginning, middle, and end in terms of this necessity [1]. Marlinspike strives to connect earlier actions to later actions precisely in accordance with this definition, so that no user action (of high import) is left unnecessary to the finished story.

The way Marlinspike selects current scenes that build on past events is inspired by Keith Johnstone's work in improvisational theatre [7]. Specifically, he claims that if one focuses on the structure produced by reincorporating previous events, the content of stories tends to take care of itself.

4. INFLUENCES AND PREVIOUS WORK

The design of Marlinspike also owes much to certain existing interactive drama systems.

Marlinspike's actions and events are much like Chris Crawford's verbs and events as used in his Erasmatron system [2]. However, unlike Crawford, we make the distinction between world-level and story-level events. Instead, Crawford only concerns himself with the story-level. (For instance, his verbs take only characters as subject or direct object.) Additionally, any character can be the subject of Crawford's verbs, while all Marlinspike actions are player-centric. Finally, Crawford's system is largely decentralized, with the rules of system (specifically, inclination formulae) determining how characters respond to the player's actions. As such, his system has no significant model of the plot beyond an event history and certain global variables.

The process of stringing pre-authored scenes together at run-time based on scene preconditions has also been used, among others, by Grasbon and Braun in GEIST [5,9] and by Mateas and Stern in Facade [8]. Neither approach includes user actions as atoms in the story model, however. Instead, user actions within a scene (or within a *beat*, as Mateas and Stern call their story atoms) simply determine the outcome of that scene.

Marlinspike uses a much more flexible story model than either Grasbon and Braun's Propp-based morphological approach or Mateas and Stern's tension-based story arc model. While this will hopefully mean a system more adaptive to user action, it also means that Marlinspike dramas may not be as well-structured in terms of rising and falling tension.

Fairclough's OPIATE system [4] includes a similar dynamic casting of characters into roles based on affinity for the player's character. However, OPIATE used case-based reasoning to build complete storylines at once. If the player failed three times to cooperate with a storyline's required choices, that storyline would be put on hold and a new one generated. Instead, Marlinspike strives to incorporate any important user actions into a single (albeit multi-threaded) story-line generated piece-by-piece.

Building on our own previous work with a scene-based approach [11], we have redefined the atoms of our story model to include both scenes and user-entered actions. Additionally, in the process of reincorporating past actions when selecting the next scene, it will be possible to reveal to the player exactly how their actions are impacting the story. For example, characters may refer to the previous event that resulted in their assumption of a role that is now relevant to the current scene. This is intended to offer the player a greater sense of story-level agency [8,11] than existing systems.

Another step towards greater user agency is that Marlinspike always gives the player a chance to act before starting the next scene. This is meant to avoid long strings of non-interactive instant scenes, a problem that plagued our last design [11].

Finally, it is hoped that Marlinspike dramas will be faster to author than current scene-based approaches. While still dependent on a large number of pre-authored scenes, with Marlinspike not all significant action must be within the bounds a scene. This can potentially limit the number of scenes required for a successful story. Additionally, a scene needs only specify its exceptions to the underlying default verb-to-action translations rules, rather than directly providing the regular means of user interaction.

5. PROTOTYPE IMPLEMENTATION

Marlinspike has not yet been implemented, but a prototype is currently under development. While the Marlinspike architecture would also work in a mouse-based, graphical environment (though the range of verbs may be smaller), our prototype game will be text-based. Specifically, we will use the interactive fiction system, Inform [6], 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-based. This is meant to improve system affordances, since we do not plan to implement any natural language processing beyond Inform's existing parsing capabilities.

The number of verbs required will be comparable to interactive fiction games—about fifty to sixty. These will translate to approximately thirty 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. Stories will end as soon as the drama manager can meet all the preconditions of one of about seven ending scenes.

6. CONCLUSION

Marlinspike's design strives to better merge the strong story-control of scene-based approaches with the high user agency of Crawford's verb-based approach. In this system, the player interacts as a character at the story-world level of *verbs*. The translation from these verbs to story-level *actions* can then be affected or overridden by the story context established by earlier story events. Marlinspike responds to user actions by selecting the next pre-authored *scene* that will further the story. The primary feature of Marlinspike is that, beyond just finding the next scene it *can* play, it strives to play scenes that actually make earlier user actions integral and necessary to the resulting plot. Thus, the system offers the user a wide range of possible actions at any point in the story, but also works to then incorporate those actions into its own model of the story.

Further work—starting with the implementation of the planned prototype game—will reveal whether Marlinspike actually lives up to its potential.

7. References

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

[2] Crawford, Chris. *Chris Crawford on Interactive Storytelling*. Berkeley, CA: New Riders, 2004.

[3] *Dungeons & Dragons: Dungeon Master's Guide: Core Rulebook II v3.5*. Renton, WA: Wizards of the Coast, 2003.

[4] Fairclough, Chris R. "Story Games and the OPIATE System: Using Case-Based Planning for Structuring Plots with an Expert Story Director Agent and Enacting them in a Socially Simulated Game World." PhD thesis. University of Dublin, Trinity College, 2005. <http://www.cs.tcd.ie/publications/tech-reports/reports.05/TCD-CS-2005-59.pdf>

[5] Grasbon, Dieter, and Norbert Braun. "A Morpholgical Approach to Interactive Storytelling." *Proceedings of cast01//Living in Mixed Realities, 2001*. <http://netzspannung.org/version1/extensions/cast01-proceedings/pdf/by_name/Grasbon.pdf>

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

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

[8] Mateas, Michael. "Interactive Drama, Art, and Artificial Intelligence." PhD Thesis. Technical Report CMU-CS-02-206. Carnegie Mellon University, Pittsburgh, PA, 2002. <http://www.cs.cmu.edu/afs/cs.cmu.edu/misc/mosaic/common/omega/Web/Groups/oz/papers/CMU-CS-02-206.pdf>

[9] Spierling, Ulrike, Dieter Grasbon, Norbert Braun, and Ido Iurgel. "Setting the Scene: Playing the Digital Director in Interactive Storytelling and Creation." *Computer & Graphics* 26 (2002): 31–44.

[10] Tomaszewski, Zach, and Kim Binsted. "A Reconstructed Neo-Aristotelian Theory of Interactive Drama." *Computational Aesthetics: Artificial Intelligence Approaches to Beauty and Happiness: Papers from the 2006 AAAI Workshop*. Technical Report WS-06-04. Menlo Park, CA: AAAI Press, 2006. <http://www.zach.tomaszewski.name/argax/pubs/2006-TomaszewskiBinsted-Drama.pdf>

[11] Tomaszewski, Zach, and Kim Binsted. "The Limitations of a Propp-based Approach to Interactive Drama." *Intelligent Narrative Technologies: Papers from the AAAI Fall Symposium.* Technical Report FS-07-05. Menlo Park, CA: AAAI Press, 2007. <http://www.zach.tomaszewski.name/argax/pubs/2007-TomaszewskiBinsted-ProppLimitations.pdf>