On Preventing Sequence Breaking in Video Games

Matthew Shelley
Carleton
1125 Colonel By Drive
Ottawa, ON, Canada K1S 5B6
MatthewShelley@carleton.ca

Wei Shi
UOIT
2000 Simcoe North
Oshawa, CANADA, L1H 7K4
Wei.Shi@uoit.ca

J.-Pierre Corriveau
Carleton
1125 Colonel By Drive
Ottawa, ON, Canada K1S 5B6
jeanpier@scs.carleton.ca

ABSTRACT

Sequence Breaking exists in video games where the player gains access to a portion of a game that should be inaccessible. In such instances, a game's storyline is disrupted. That is, the predefined set of valid event sequences—events being uninterruptable units of functionality that further the game's story—is not honored. We postulate that sequence breaking most often arises through bypassing geographic barriers, cheating, and misunderstanding on the player's behalf. Here, we summarize an approach to preventing sequence breaking at run-time with the help of Use Case Maps.

Keywords

Sequence Breaking, Narrative, Games, Use Case Maps

1. OVERVIEW

Sequence breaking exists in the domain of game narrative when a predefined storyline is not followed as per the game designers' intentions. It can be viewed as a form of cheating: When a 'narrative' feature, hereafter referred to as an *event*, is called outside of the game's set of predefined narrative sequences, there exists unwanted interference with the storyline as its integrity has not been honored. When the player starts an invalid sequence of events, they are breaking the predefined narrative sequence.

Video games have been subject to sequence breaking since their inception. For example, in 2011, Nintendo's *The Legend of Zelda: Skyward Sword* became impossible to finish if players completed tasks in a certain order [1]. Several other examples are described in [2]. In summary, sequence breaking conflicts have existed for decades and continue to exist to this day.

Such conflicts are detrimental to players in two ways. First, as skilled players are often the ones to perform sequence breaking, for the sake of cheating or their own enjoyment, other players may fall victim to such unfair advantages. Second and worse, when sequence breaking occurs without the player's knowledge, the game's story may make no sense, leading to confusion, or the game may become unwinnable. Because the player's experience is reduced, sequence breaking poses a significant problem for an industry that relies on creating 'fun' in order to sell its products.

Some solutions to sequence breaking have been attempted. For example, Eladhari [3] proposes the creation of causal relationships between events, as well as "Object Oriented Story Construction". The latter requires that all entities of a game be given knowledge of the game's story in order to "make them more intelligent with respect to the overall narrative goals". The main disadvantage with existing research in narratives however remains that it is essentially theoretical: existing proposals are not implemented in an actual game. Consequently, unfortunately, the verification of a computational version of such proposals is generally not addressed.

We can think of narrative elements as the 'who,' 'what,' and 'where' of a game, while the 'when,' 'why,' and 'how' of the game's story are either told or shown to the player. For instance, within the game's fictive world, the player (who) may battle enemies (who) or interact with other characters (who); the player may collect objects (what); and, the player may explore geography (where). The time of the story (when), motivations of characters (why), and actions that occur (how) can be denoted using events.

From reviewing the literature and studying five popular games, we compiled a list of narrative elements that appear to be necessary and sufficient to represent many storylines. We then observed that a subset of Use Case Maps (UCMs) [4] could readily capture these narrative elements and represent the valid sequences of a narrative (in order to compare a player's progress against the designer's set of valid storylines).

Based on these initial observations, we propose a 'narrative manager' and traversal algorithm in order to prevent sequence breaking within a game environment at run-time. We represent a set of valid narrative sequences using UCMs; keep track of the player's progress along such a representation; and then, check if attempted event calls are legal using this combined knowledge. The narrative manager stores the narrative representation, player's progress, and set of legal events. This manager can determine if attempted event calls are legal by checking against the current legal set of events. The traversal algorithm accepts an event identifier for a legal event, and then updates both the player's progress and the set of next legal events. We can prevent sequence breaking by rejecting any attempted calls to illegal events. In our work [5], we have implemented a narrative manager and a UCM traversal algorithm in order to monitor the player's progress and prevent sequence breaking. We then developed an extensive set of test cases that address all the UCM elements we use, as well as cover all the branches of our algorithm. Finally, we developed a game to illustrate the feasibility and performance of our solution. Most importantly, this game demonstrates that our run-time approach to sequence breaking works within the limitations of a game's environment *without* hindering the player's experience (as we discuss at length in [5]). Let us briefly elaborate.

It is important to note that since the nature of sequence breaking involves the unintended sequential ordering of events, it cannot be assumed that any arbitrary event can be guaranteed to not be called. Instead, we have to assume that any event can attempt to be called regardless of the player's progress. Therefore, it seems to be necessary to validate at least $O(n^2)$ sequences, where n is the number of events, prior to the game's execution (e.g. at design-time or during play-testing), but such validation is intractable. It is unreasonable to expect designers or play-testers to consider an exponential number of sequences as such an approach simply does not scale. Thus, it is necessary in our opinion to offer a run-time solution. Indeed, instead of testing and resolving (not only pair-wise but) all sequences of events to ensure only valid sequences are allowed, it is preferable to create a means of checking if a requested event is legal based on the player's progress within the storyline at a given point in the game. When an attempted event call is deemed to be legal, it may proceed as expected; otherwise, the event call is rejected (and an appropriate warning can be given to the player). In this approach, designers need only consider valid sequences, while game testers verify that this preventative procedure works.

We believe sequence breaking to be preventable at run-time when, in a time undetectable by the player and within a game environment, we can perform the following algorithm, given an event identifier, narrative representation, and the player's progress:

1. Determine if the event associated with the given event identifier is legal to call at this point of the narrative.
2. If 'yes', return a 'success' value, call the associated event, update the player's progress, and then:
   a. Update the game's world to allow the player to call events that are now considered legal; and,
   b. Update the game's world to prevent now-illegal events from being called, effectively preventing the player from accessing illegal events.
3. If 'no':
   a. Ignore the attempted event call; and,
   b. Return a 'failure' value, so an optional developer-defined function can be triggered to handle the detected sequence breaking, possibly by using the current set of legal events.

One derived benefit of our solution is that, having access to the current legal set of events, a designer may draw on this knowledge to have the game offer better context-sensitive behavior (such as improved dialogues, better behavior for non-player characters, etc.). In particular, we explain elsewhere [5] how, beyond preventing sequence breaking by ignoring attempted calls to illegal events, we try to reduce such calls by altering the game world to exclude related triggers or entities.

There is however presently a drawback in our approach: Because the player's progress is currently only updated when a legal event is called, it is possible for events that depend on conditions to not be 'moved' to the legal set before another event has been triggered. Similarly, an event that has become illegal based on a precondition may remain in the legal set of events. A solution would require rethinking the current handling of narrative-related variables. At this point in time, it is not clear whether such rethinking should occur only in the limited scope of single-player adventure-type games that we have chosen, or consider the complexities that will be unavoidably introduced if we tackle massively multiplayer online games.

2. REFERENCES

[1] Kotaku: Brian Ashcraft, "Game-Breaking Skyward Sword Bug Confirmed. Here's How to Avoid It." December 7 2011: http://kotaku.com/5865426/game+breaking-skyward-sword-bug-confirmed-heres-how-to-avoid-it [Accessed August 29 2013].

[2] tvtropes.org, "Sequence Breaking" May 9 2012: http://tvtropes.org/pmwiki/pmwiki.php/Main/SequenceBreaking [Accessed August 29, 2013].

[3] M. Eladhari, Object Oriented Story Construction in Story Driven Computer Games, Master's Thesis, Stockholm University, Stockholm, Sweden, 2002.

[4] M. Shelley, On the Feasibility of using Use Case Maps for the Prevention of Sequence Breaking in Video Games, Master's thesis, Carleton University, Ottawa, 2013.

[5] D. Amyot and G. Mussbacher, "User Requirements Notation: The First Ten Years, The Next Ten Years", Journal of Software, vol. 6, no. 5, pp. 747-768, 2011.