SHOWRUNNER: A Tool for Storyline Execution/Visualization in 3D Game Environments

Rushit Sanghrajka, R. Michael Young, Brian Salisbury, and Eric W. Lang

University of Utah, Salt Lake City, UT
rush.sanghrajka@utah.edu
young@eae.utah.edu
salisbury@eae.utah.edu
ewlang@cs.utah.edu

Abstract. We introduce SHOWRUNNER , a tool for visualizing story world execution within a 3D game environment. SHOWRUNNER takes as input an abstract, declarative specification of a story script and a set of mappings between terms in the story and data elements in the game engine and executes the story's actions, using virtual cameras to film and present the action to a user. The implementation details on the working of the tool, as well as instructions on how users with various design and API constraints can utilize the tool are discussed in this paper.

Keywords: Story execution · game environments · cinematic visualization of stories

1 Introduction

The domain of computational narrative is growing steadily, using algorithmic approaches to construct plot and narrative. An important capability for systems that synthesize stories is to be able to present the output of a story's action sequences to human users in some conventional narrative medium. We have developed a tool called SHOWRUNNER that allows story action sequences generated by external systems (e.g., narrative planners, screenwriting graphical user interfaces, or human editors) to be automatically executed within a 3D virtual world, creating a machinima-based visualization for the action sequence.

SHOWRUNNER provides a specific story world built within a commercial 3D game engine (i.e., Unity) and a means for reading in a story line and a mapping used to translate from some external naming of story entities and the internal game engine entities in the game engine's data model. In this sense, SHOWRUNNER abstracts away the details of the game engine's execution of story actions and allows some exogenous system or a human author to specify the dynamics of a story in a declarative language appropriate for the external model's representational needs. Our intent in building SHOWRUNNER is to increase the accessibility of a 3D game engine as a resource for visualization for intelligent story generators, although the interface SHOWRUNNER provides can be used by any external source capable of creating story specifications. Further, SHOWRUNNER is extensible, in that developers wishing to add new scenes, set elements, objects, actions and animations can create and incorporate that content via Unity's development tools.

This paper details the design and functionality of the SHOWRUNNER system, and provides information about how SHOWRUNNER can be used by researchers in order to visualize story execution from a variety of input sources.

2 Related Work

A number of previous projects have developed the capability to execute story lines within 3D game engines. Many of these explicitly provided a tight coupling between an AI system generating the story and the game engine used to execute the story line. Pollack and Ringuette's Tileworld [8] was one of the first systems to build a game-like environment for executing the output of experimental agent architectures controlling agents in unpredictable and dynamic worlds. Laird [5] created one of the initial systems to connect an external AI controller – the SOAR Architecture [6] – to a commercial 3D game engine. Young developed a system, *Mimesis*, which created virtual world narratives using a bipartite approach of a story-world planner and discourse planner [14]. Cavazza and his collaborators [1, 2] also built a number of systems that were driven by AI planning systems executing in game engines. Thuene et al. built the Virtual Storyteller system [11], which is a framework for creating plot, narrative and presentation of the narrative. Kapadia and his collaborators [4] developed a framework for authoring multi-agent environments with behaviors. Screenwriting software (e.g. [3, 7] allow for features like tracking character trajectories, conflict, emotions, and even provide for previsualizations of stories.

SHOWRUNNER expands the capabilities of related work by acting as a test-bed where one could connect story input from a range of input sources: human authored, computationally generated or computer-assisted. SHOWRUNNER hopes to provide one common environment for execution of these experiences and provide one automated pipeline towards that end.

3 System Overview

SHOWRUNNER provides a layer of abstraction over action execution in pre-defined Unity game environments. These pre-defined game environments contain a set of data structures and code that hold assets corresponding to the characters and their action animations, objects, locations, and the code used to perform the actions of the story world. Extending a SHOWRUNNER level is discussed briefly in Section 4.

SHOWRUNNER takes as inputs three elements: a specification of the actions in a story, a (possibly empty) specification of a custom starting state for the story, and a data dictionary that maps descriptors in the start state and action descriptions to corresponding descriptors in the Unity game environment. The system first creates an internal database used to map the input story references to game engine-internal objects. Next, it modifies the starting state of the story world according to any customizations detailed in the input files. Finally, it begins executing the actions enumerated in the story script, ensuring that each one executes correctly. During execution, a camera system automatically films the unfolding action, visualizing it for the user.

We have built and tested a Western cowboy-themed story world. We are developing similarly instrumented story worlds for feudal Japan, medieval Europe and a dungeon-focused fantasy world.

The SHOWRUNNER system is built using C# and its code and art assets exist within a collection of scenes within the Unity game engine [13]. The system consists of a number of customizable, modular components, described in more detail below.

Input and Output. SHOWRUNNER input consists of two required elements and one optional element:

1. A Story Script. A story script is a file containing a declarative representation of the actions in a story along with ordering dependencies between those actions.
2. A Mapping Definition. A mapping definition is a file containing associations between the symbols used as identifiers in the story script (i.e., action types, characters, locations and objects) and the unique IDs provided by SHOWRUNNER to the corresponding Unity GameObjects in the story world scene.
3. An optional Initial State Description. An initial state description is used to specify any ground literals whose truth value is required to be different than their truth values in the default start configuration of the story world's scene.

Output from SHOWRUNNER is a 3D visualization of the execution of the actions in the story plan running within the virtual set of the story world. These actions are filmed by virtual cameras pre-placed within the Unity scene. Dynamic camera selection is managed automatically by Unity's Cinemachine [12] camera control system, which optimizes camera selection based on visibility of target characters, user-defined weights, and other cinematic factors.

The Story World's Scene. The process of executing stories runs entirely within a Unity scene. The scene specifies the virtual world of the story, the art and code assets for the entities in the story as well as the code to manage the system's behavior.

Each SHOWRUNNER Unity scene contains the following elements:

1. The *virtual set*. This set includes the 3D space of the story world, including buildings, exterior landscape and any objects in the story world that have no dynamic state properties associated with them.
2. The *world objects*. World objects in SHOWRUNNER are Unity GameObjects that have a physical representation in the story world and are distinguished from elements of the virtual set because characters can interact with them.
3. The *character models*. In our current implementation, the characters are human figures, but future implementations may extend our character model set to include horses, fantastical beasts, or other entities with agency. These are represented within the game engine as Unity GameObjects with specific components.
4. The *animations*. Animations capture the movement of the elements of the 3D models of characters or the dynamics of other animated objects.
5. The *animlocations*. Animlocations are specially designated Unity GameObjects that are placed within the virtual set. Each animlocation is associated with one or

more animations and designates a physical location and orientation that any character model performing the associated animation must be placed in. These locator objects serve a role similar to location vectors originally used by the Steve virtual agent [9].
6. The *action classes*. Action Classes are C# classes that define the processes for running an individual story action in the story world. Each action class defines a set of methods for checking that the action's preconditions hold in the game world state, performing the animations and game state changes that form the body of the action, and confirming that the action has successfully established its effects in-game. Action class methods are written as co-routines, allowing a form of concurrent execution between actions that are not temporally ordered with respect to one another.
7. The world's *virtual cameras*. Virtual cameras are placed throughout the virtual set and are accessed by the execution manager to provide the visualization of the story world action as it unfolds.
8. The Execution Manager. The SHOWRUNNER Execution Manager is the main control point for the story's execution in Unity. SHOWRUNNER is, in effect, a scheduler responsible for initiating actions for execution and tracking the success/completion of the methods used by each action's action class instance.

Execution Manager. The Execution Manager operates in two phases: Start Up and Running. In the start-up phase, the Execution Manager reads in the input files and first creates a MapDB database that translates from story entity references to SHOWRUNNER -internal object references. Next, it reviews the content of the Initial State Revision file, making any modifications to the game world the file specifies. Finally, it creates a directed acyclic graph (DAG), where each node in the graph is one of the actions listed in the input script. Orderings between nodes are created based on the partial ordering over story actions specified in the script.

Once this execution DAG has been created, the Execution Manager switches to Running Mode. The Execution Manager iterates in Running Mode by (a) checking to see if the DAG is empty, in which case SHOWRUNNER halts, and (b) selecting the minimal elements in the DAG and constructing a method call for each from the action specifics in the node and Unity method names and GameObject references provided via look-up in the MapDB. This method is then invoked with the identified parameters. As the code for each action terminates, the code removes its corresponding node from the DAG.

4 Discussion

SHOWRUNNER provides a useful level of abstraction away from the details of a game engine's coding and operation. SHOWRUNNER is designed to support at least two distinct use cases. One is its use essentially as an off-the-shelf story visualization tool. In this use case, story scripts are built using references just to SHOWRUNNER 's default virtual set, characters and actions. In a power user use case, a user can create new actions by adding new action classes, animations, etc, within the SHOWRUNNER Unity project. The code for the system is available in the project's Gitlab repository [10].

References

1. Cavazza, M., Charles, F., Mead, S.J.: Character-based interactive storytelling. IEEE Intelligent systems **17**(4), 17–24 (2002)
2. Cavazza, M., Lugrin, J.L., Pizzi, D., Charles, F.: Madame bovary on the holodeck: immersive interactive storytelling. In: Proceedings of the 15th ACM international conference on Multimedia. pp. 651–660. ACM (2007)
3. Hollywood Camera Work: Causality story sequencer, `https://www.hollywoodcamerawork.com/causality.html`
4. Kapadia, M., Singh, S., Reinman, G., Faloutsos, P.: A behavior-authoring framework for multiactor simulations. IEEE Computer Graphics and Applications **31**(6), 45–55 (2011)
5. Laird, J.E.: It knows what you're going to do: adding anticipation to a quakebot. In: Proceedings of the fifth international conference on Autonomous agents. pp. 385–392. ACM (2001)
6. Laird, J.E., Newell, A., Rosenbloom, P.S.: Soar: An architecture for general intelligence. Artificial intelligence **33**(1), 1–64 (1987)
7. Marti, M., Vieli, J., Witoń, W., Sanghrajka, R., Inversini, D., Wotruba, D., Simo, I., Schriber, S., Kapadia, M., Gross, M.: Cardinal: Computer assisted authoring of movie scripts. In: 23rd International Conference on Intelligent User Interfaces. pp. 509–519. ACM (2018)
8. Pollack, M.E., Ringuette, M.: Introducing the tileworld: Experimentally evaluating agent architectures. In: AAAI. vol. 90, pp. p183–189 (1990)
9. Rickel, J., Johnson, L.: Integrating pedagogical capabilities in a virtual environment agent. In: Proceedings of the First International Conference on Autonomous Agents. pp. 30–38 (1997)
10. Sanghrajka, Rushit and Young, R. Michael and Salisbury, Brian and Lang, Eric W.: ShowRunner GitLab Repo. GitLab (2019), `https://eae-git.eng.utah.edu/01221789/utahpia2`
11. Theune, M., Faas, S., Nijholt, A., Heylen, D.: The virtual storyteller: Story creation by intelligent agents. In: Proceedings of the Technologies for Interactive Digital Storytelling and Entertainment (TIDSE) Conference. vol. 204215 (2003)
12. Unity Technologies: Cinemachine, `https://learn.unity.com/tutorial/cinemachine`
13. Unity Technologies: Unity, `https://unity3d.com`
14. Young, R.M.: Story and discourse: A bipartite model of narrative generation in virtual worlds. Interaction Studies **8**(2), 177–208 (2007)