Bringing Authoring Tools for Intelligent Tutoring Systems and Serious Games Closer Together: Integrating GIFT with the Unity Game Engine

Colin Ray, Stephen Gilbert

Iowa State University, Ames, IA, USA
{rcray, gilbert}@iastate.edu
http://www.iastate.edu

Abstract. In an effort to bring intelligent tutoring system (ITS) authoring tools closer to content authoring tools, the authors are working to integrate GIFT with the Unity game engine and editor. The paper begins by describing challenges faced by modern intelligent tutors and the motivation behind the integration effort, with special consideration given to how this work will better meet the needs of future serious games. The next three sections expand on these major hurdles more thoroughly, followed by proposed design enhancements that would allow GIFT to overcome these issues. Finally, an overview is given of the authors' cur- rent progress towards implementing the proposed design. The key contribution of this work is an abstraction of the interface between intelligent tutoring systems and serious games, thus enabling ITS authors to implement more complex training behaviors.

Keywords: intelligent tutoring, serious games, virtual environments, game engines

1 Introduction

Experience with the Generalized Intelligent Framework for Tutoring (GIFT) has shown that authoring new courses, domain knowledge, and learner configurations requires little-to-no programming experience. A basic understanding of XML and how the modules of GIFT interact is sufficient to design and configure a course for one of the supported training applications. When it comes to extending the framework to support new training applications, however, each interface module must be hand-crafted. Reducing the amount of effort required to author a tutor and its content is a desirable quality of future authoring tools [1], therefore the task of integrating new training applications should be made as seamless as possible.

Serious games are one example of training applications that are well-suited for integration with ITSs; two such games are already supported by GIFT: Virtual Battlespace 2 (VBS2) and TC3 vMedic. These games encompass a only a subset of the training material that is possible with serious games, however. There are certain aspects of this genre of game are common across all individual applications, meaning that it may be possible to create a single abstraction layer capable of decoupling GIFT from the training application. This approach is recommended by Sottilare and Gilbert, who suggest that such an abstraction layer might be able to translate learning objectives into meaningful objects actions in the game world, and vice versa [2].

In addition to adapting data about the game state to a format that the ITS expects, it is also desirable for the ITS to have a finer degree of control over the scenario itself. These so-called “branching” or “conditional” scenarios [2] are difficult to achieve if the serious game and its plugin API are not designed with such functionality in mind. Therefore, it may also be necessary to “standardize” the ability to branch scenarios in the design of serious games.

To these ends, our proposed solution is to bring the ITS authoring tools closer to the content authoring tools used to create a given serious game. In the case of this paper, we have chosen to work with the popular Unity game engine. In the following sections we will show how integration with Unity and other serious game authoring tools can achieve the functionality that is currently desired in a modern ITS authoring suite.

2 Current Authoring Capabilities

As stated by Sottilare et. al, authoring new intelligent tutors is one of the three primary functions of GIFT [3]. To this end, the framework already contains authoring tools that enable users to create and configure the essential elements of an intelligent tutoring program. The following list gives a brief overview of the current authoring capabilities supported by GIFT:

- Authoring learner models through the Learner Configuration Authoring Tool (LCAT)
- Configuring sensors through the Sensor Configuration Authoring Tool (SCAT)
- Authoring Domain Knowledge Files (DKFs) through the DKF Authoring Tool (DAT)
- Creating and presenting surveys through the Survey Authoring Tool (SAT)

By using good design principles, the authors of GIFT have been able to effectively decouple the authoring of individual tutor components from one another. The decoupling of different program elements is important for improving the maintainability and extensibility of large pieces of software such as GIFT. One area of the framework design that suffers from tight coupling is the integration of third-party training applications, e.g. VBS2, vMedic, etc.

The development of these authoring tools is guided by several design goals, one of which is to “Employ standards to support rapid integration of external training/tutoring environments.” [3] In this regard, the current GIFT authoring construct can benefit from design enhancements that standardize this process across a range of training applications. Through the work outlined in this paper, we aim to generalize the process of integrating serious games with GIFT by creating an abstraction layer between GIFT and the game engine itself.

3 Related Work

Prior work in integrating serious games and intelligent tutors has demonstrated that ITS authoring tools can be easily adapted to work with individual games. Research conducted by Gilbert et al. demonstrated interoperation between the Extensible Problem-Specific Tutor (xPST) and a scenario created in the Torque game engine [4].

Devasani et al. built upon this work and demonstrated how an authoring tool for interpreting game state and player actions might be designed [5]. For their work, xPST was integrated with a VBS2 scenario. An important revelation made by the authors was that the author of the tutor need not define a complete state machine with transitions, since these transitions are implicit when the game engine changes state each frame.

Another of the GIFT design goals is to "Develop interfaces/gateways to widely used commercial and academic tools." [2] As previously mentioned, the current GIFT release has support for two serious games, one of which is VBS2, and the other being vMedic. This work and the previous two examples highlight the usefulness of integrating intelligent tutors with serious games, as well as the need for a standardized interface for authoring relationships between the game objects and tutor concepts. There are currently no concrete examples of a standard for quickly integrating serious games and intelligent tutors, although Sottilare and Gilbert make recommendations on how this problem might be approached [2].

4 Design Enhancements

As noted by previous authors [2, 4], one of the key challenges of tutoring in a virtual environment is mapping relevant game states to subgoals defined by the training curriculum. If the learner's goal is to move to a specific location, for example, the tutor author may not be interested in how the learner reached that state (e.g., driving, walking, or running). Thus, the tutor would have to know to filter out information from the game engine about modality of movement, and attend only to the location. If, however, the trainer wants to focus on exactly how best to move to that location (e.g., stealthily), then the tutor does need to monitor movement information. Using this example, we see that the context of the pedagogical goal influences the type of and granularity of tutor monitoring. From here on, we will refer to this challenge as the "observation granularity challenge."

In the process of reaching each pedagogical goal, the learner will build up a history of actions. Similar to the concept of a context attached to goals, there can also be context attached to patterns of actions over time. As an example, there may be cases where a tutor would permit errors in subgoals within a larger pattern of actions that it would still deem "successful." This history is essentially a recording of the virtual environment state over the course of the training. The amount and diversity of data in this history stream is potentially massive, creating a major challenge when attempting to recognize patterns. The problem of recognizing these patterns is crucial for identifying the learner's progress. From here on, we will refer to this challenge as the "history challenge."

In addition, because game environments afford interaction among multiple simultaneous entities, the tutor's reaction to actions and other new game states may be dependent on the actor. This context dependence suggests that it would be a valuable to add game entity attributes to state updates, and for GIFT to be able to process logic such as, "If the gunshot action came from an entity that is unknown or hostile, then take action X. If the gunshot came from a friend entity, take action Y." The additional layer of entity attributes adds complexity to authoring, but will be necessary for modeling team and social interactions. Devasani et al. describes a possible state-based architecture that could be the basis for such an approach, and it could be incorporated into GIFT [4]. From here on, we will refer to this challenge as the "actor-context challenge."

4.1 Abstraction Layer

A core aspect of the design principles behind GIFT is its generalizability to new training applications and scenarios. For this reason it is critical that the representations of data in GIFT and in the training application be allowed to remain independent. It is infeasible to force training applications to adapt to the interfaces that GIFT provides. However, a layer of abstraction that adapts messages from a sender into a form that can be consumed by a receiver is similar to the classic Adapter design pattern in software engineering. This design pattern has the useful property of enabling two otherwise incompatible interfaces to communicate, in addition to decreasing the coupling between them. In the case of GIFT, the abstraction layer would handle the mapping of objects from one service into a representation that makes sense to the other. As an example, this module might receive a message from the game engine containing the new location of the learner in the virtual environment which might then be interpreted for the tutor as "the learner reached the checkpoint."

In addition to mapping game engine concepts to tutor concepts, the abstraction layer can act as a filter in order to solve the observation granularity and history challenges. The scripting language achieves this by affording "do not care" conditions that would then trigger the abstraction layer to interpret only the relevant messages and discard everything else.

One proposed method for implementing this mapping is a scripting language and engine that allows the author to define the mapping themselves. Although it is far from being an automated solution, a scripting language would allow the ITS and content authors to hook into more complex behaviors with very little learning overhead. Scripting languages can be more user-friendly than XML by virtue of their syntactical similarities to written English. Furthermore, within the context of the Unity development environment we can expect users to have familiarity with scripting languages such as JavaScript and Boo (similar to Python). For these reasons, a scripting language is a natural choice for abstracting communication between GIFT and Unity. It is important for the simplicity of tutor authoring that this messaging abstraction layer have the tutor-side representation use language that a trainer would naturally use. If this is the case, the trainer can more easily author feedback and adaptive scenarios.

Although JavaScript and Boo are well-suited as tools to implement complex behaviors for game objects, they overcomplicate the task of describing interactions between the game world and the tutor. Instead of complex behaviors, we seek to enable the tutor author to quickly declare relationships between objects in the game, domain knowledge, and pedagogical goals.

In order to avoid burdening the author with the challenge of authoring different components in different languages, it may be advantageous to use XML for authoring abstraction layer rules. The declarative nature of XML makes it ideal for this role, although as mentioned previously, it suffers from poor readability. An alternative to XML is TutorScript, a scripting language developed for use in ITSs [6]. The design of TutorScript centers around the sequences of goals or contexts called a predicate tree. TutorScript's primary advantage over the previously mentioned alternatives is that it was designed with the goal of relating domain knowledge to learner actions in the training application. Additionally, TutorScript takes inspiration from Apple script in regards to syntax, allowing non-programmers to write scripts that read like English. For our work, TutorScript would allow us to hook into objects in both GIFT and Unity, where we can then create interactions using simple language.

4.2 Unity Editor

One of the main benefits of the Unity editor is that it is extensible to support user-created tools for custom workflows, or to fill in functionality lacked by the default editor. Some examples of editor plugins authored by users have added advanced level building tools, cut-scene editors, and even node-based visual scripting interfaces. The ultimate goal of this project is to completely integrate GIFT's authoring tools with the Unity ecosystem. This entails creating editor plugins for the entire suite of GIFT authoring tools, thereby enabling content authors to generate serious game and tutor content side-by-side using a single development environment.

An added benefit of integration with the Unity editor is its powerful rapid-prototyping abilities. Scenarios in Unity are organized into "Scenes" which can be loaded individually, played, and paused within Unity's built-in player. Current work to develop a proof-of-concept has demonstrated that it is possible to interact with the tutor within this player, thereby enabling the author to perform debugging on the training scenario to an extent.

It is considered good practice when authoring Unity games to "tag" game objects with names that encode the meaningful behavior that the game object performs. Assuming that the author adheres to this practice, the tagging mechanism combined with the abstraction layer will solve the actor-context challenge. Tags can be transmitted with game state updates that pass through the abstraction layer, which will then interpret the tags into context that is meaningful to the tutor. Since the abstraction layer is scripted by the author, it is essential for the abstraction layer script editor to be included in Unity's authoring suite. Making these tools easily accessible from one or the other allows the author to update changes to the scripts as soon as he or she makes changes to game object tags and other metadata.

As stated previously, the scripting languages provided by Unity may not be ideally suited to the task of communicating between the game engine and the tutor. Additional modifications will need to be made to MonoDevelop, the highly extensible IDE distributed as part of Unity, in order to support TutorScript or a variant of it. MonoDevelop greatly simplifies the creation of helpful programming tools such as syntax-highlighting and auto-completion that assist users with no prior programming background. Developing a MonoDevelop add-on for TutorScript also allows the author to more easily manage large or complex scripts needed to address the history and actor-context challenges via the built-in code organization features such as collapsing scopes. Taken together, Unity and MonoDevelop can be used as a suite of tools for authoring not only serious game content, but also advanced tutor behaviors, curriculum, and domain knowledge that will drive the training scenarios.

5 Recommendations

We project that the design enhancements recommended in this paper will assist in improving time savings and reducing cost involved with authoring an intelligent tutor. Specifically, we are aiming to reduce the time required to integrate GIFT with a new serious game by instead integrating it with the game engine itself. Our reasoning is that there are relatively few game engines that would need to be integrated, compared to the number of games with potential for enhancement through tutoring. Additionally, code reuse is facilitated by employing a standard format for describing relationships between game and tutor objects. If successful, this work will introduce a new abstraction layer between GIFT and the game engines that drive serious serious games, enabling a single tutor configuration to be deployed across a wide range of scenarios. For your convenience, the recommendations have been consolidated and figured in the table below.

Table 3. GIFT Design Enhancement Recommendations

Improve decoupling of potential learner actions and other game-specific data from the gateway and other GIFT modules.
Define a new XML schema for constructing game-tutor object relationships.
Develop a new authoring tool capable of authoring and validating these relationships.
Integrate new and existing authoring tools with the Unity editor.

6 Current Work

At this point we have successfully developed a proof-of-concept plugin that demonstrates basic communication between GIFT and Unity-driven games, similar to the interoperation module developed for VBS2. The extent of this functionality encompasses connecting to the Unity plugin from GIFT and then issuing playback commands such as pause and resume to the Unity player. This work has helped to increase our understanding of the inner workings of GIFT with regard to the augmentation required to communicate with our abstraction layer. In particular, the extent to which GIFT is tailored to each training application became apparent. In addition, we were able to leverage support for C# .NET 2.0 in Unity to move a great deal of the supporting code into components attached to game objects. This design allows the three services (Unity, Abstraction Layer, and GIFT) to remain isolated from one another during development, encouraging loose coupling across service boundaries and portability to other serious game authoring tools.

Before any work on the abstraction layer can begin, the language used to define object relationships must first be well-defined. Once this step is completed, we can begin abstracting away the elements of third-party application integration in GIFT that are currently hard-coded. Ultimately, these elements will be encapsulated by the proposed abstraction layer.

7 Conclusion

In this paper we proposed a handful of major design enhancements to GIFT with the overarching goal of bringing the ITS authoring workflow into the game content creation pipeline. The first task in realizing this vision is to create an abstraction layer comprised of a scripting engine tailored for ITSs. The second and final task is to integrate the GIFT authoring tools into Unity, in order to encourage side-by-side development of game and tutor content. The Unity game engine has been chosen for this work due to its ease of use, cross-platform support, and high extensibility. It is our hope that such a comprehensive suite of tools will help to drive a new generation of high-quality serious games.

8 References

1. Brawner, K., Holden, H., Goldberg, B., Sottilare, R.: Recommendations for Modern Tools to Author Tutoring Systems. (2012)
2. Sottilare, R.A., Gilbert, S.: Considerations for adaptive tutoring within serious games: authoring cognitive models and game interfaces. (2011)
3. Sottilare, R.A., Brawner, K.W., Goldberg, B.S., Holden, H.K.,: The Generalized Intelligent Framework for Tutoring (GIFT). Technical report (2013)
4. Gilbert, S., Devasani, S., Kodavali, S., Blessing, S.: Easy Authoring of Intelligent Tutoring Systems for Synthetic Environments. (2011)
5. Devasani, S., Gilbert, S. B., Shetty, S., Ramaswamy, N., Blessing, S.: Authoring Intelligent Tutoring Systems for 3D Game Environments. (2011)
6. Blessing, S.B., Gilbert, S., Ritter, S.: Developing an authoring system for cognitive models within commercial-quality ITSs. (2006) 497–502

Authors:

Colin Ray is a graduate student at Iowa State University, where he is pursuing an M.S. in Human Computer Interaction and Computer Science under the guidance of Stephen Gilbert, Ph.D. He possesses a B.S., also from Iowa State University, in the field of Electrical Engineering. His current research is focused on integrating intelligent tutoring systems with entertainment technology. In addition to ITS research, he is also conducting research and development in the areas of wireless video streaming and mobile surveillance to develop a platform for studying 360-degree video interfaces.

Stephen Gilbert, Ph. D., is the associate director of the virtual reality applications center (VRAC) and human computer interaction (HCI) graduate program at Iowa State University. He is also assistant professor of industrial and manufacturing systems engineering in the human factors division. His research focuses on intelligent tutoring systems. While he has built tutors for engineering education and more traditional classroom environments, his particular interest is their use in whole-body real-world tasks such as training for soldiers and first responders or for machine maintenance. He has supported research integrating four virtual and three live environments in a simultaneous capability demonstration for the Air Force Office of Scientific Research. He is currently leading an effort to develop a next-generation mixed-reality virtual and constructive training environment for the U.S. Army. This environment will allow 20-minute reconfiguration of walls, building textures, and displays in a fully tracked environment to produce radically different scenarios for warfighter training. Dr. Gilbert has over 15 years of experience working with emerging technologies for training and education.