# Expansion of the xPST Framework to Enable Non-Programmers to Create Intelligent Tutoring Systems in 3D Game Environments

Sateesh Kumar Kodavali$^a$ , Stephen Gilbert$^a$ , Stephen B. Blessing$^b$

Virtual Reality Applications Center, Iowa State University$^a$
University of Tampa$^b$

**Abstract.** Our previous work has demonstrated that the Extensible Problem Specific Tutor (xPST) framework lowers the bar for non-programmers to author model tracing intelligent tutoring systems (ITSs) on top of existing software and websites. In this work we extend xPST to enable authoring of tutors in 3D games. This process differs substantially from authoring tutors for traditional GUI software in terms of the inherent domain complexity involved, different types of feedback required and interactions generated by various entities apart from the student. A tutor for a village evacuation task has been constructed in order to demonstrate the capabilities of using the extended xPST system to create a game-based tutor.

**Key words:** Intelligent Tutoring System, xPST, 3D Games, Authoring, Cognitive Tutor

## 1 Background: xPST Authoring System

Re-using an existing interface with a tutor reduces the time required to develop the tutor and any issues of learning transfer, a concern of past researchers [1]. If the ITS environment is the same as the non-ITS environment then such issues of transfer largely disappear. We developed the Extensible Problem-Specific Tutor (xPST) [`http://code.google.com/p/xpst/`] in order to create ITS-based software training within the software itself. Traditional software training often uses videos based on screen-recordings, but this passive technique to learning has been shown to be less effective than an ITS [2]. xPST is similar to the Cognitive Tutor Authoring Tool (CTAT; [3]). Both allow for the creation of problem-specific tutors with programming-by-demonstration means. CTAT allows for a more visual programming approach. xPST was designed specifically to overlay a tutor on top of existing software.

The xPST architecture is an instantiation of the architecture of plug-in tutor agents described in [4]. The xPST file, which contains information that allows for instruction akin to a model-tracing tutor, describes the objects within the learning domain and rules that determine which feedback the student will receive. Every interface element of the application for which one needs learning instruction is mapped to an object (goalnode) and has one or more rules associated with it. The rules contain the instructional feedback. A Listener plugin or module eavesdrops on user actions in the third party software and sends them to the xPST Tutoring Engine, which checks them with the xPST file. We use the terminology that a goalnode is *completed* when the correct *answer* to that particular goalnode is given by the student at an appropriate point in the problem-solving process. Feedback is mapped back to the client UI control and displayed appropriately. Because software training does not typically require as many repetitive tasks as academic learning domains such as mathematics, such problem-specific models are simpler to produce and manage.

The xPST file is a text file written using a syntax that is designed to be easy to read and write for an inexperienced cognitive modeler. The file contains three sections: Sequence, Mappings, and Feedback. The sequence identifies possible paths of steps the user might take through the problem space to achieve the goal specified in the task. The feedback section provides hints and error messages for each step within the sequence. Finally, the mappings section maps interface identifiers to the steps noted in the sequence that the user takes. Because of this relatively simple syntax, the authoring tool for xPST can be a text editor. We have also built an online text editor for creating web-based xPST tutors in which authors can immediately jump to their target website and test the current xPST file [5].

We have confirmed that the xPST approach can be used to develop real tutors rapidly; our most extensive effort is described in [6], in which a tutor taught university faculty how to use a complex web-based homework authoring tool. Other efforts include tutoring on the NCBI biotechnology site and on the Slashdot website. These examples used the xPST Firefox plugin to allow xPST to eavesdrop on a user's interactions with any website that uses static html, which includes those that do not use significant Flash or AJAX-style interactions.

## 2 Emergence of Games in Tutoring

For learning to be effective it should be scaffolded or guided [7]. During the last few decades, the very nature of teaching in modern universities has changed [8]. Motivating students by setting challenges, goals and problems which are engaging is being seen as a key factor in the learning process. Research has shown that students learn better and retain more when they actively engage in the learning process. Tutoring using games provides these advantages compared to tutoring with traditional software. Pedagogy researchers have shown an increased interest in incorporating gaming principles into teaching and learning [9]. Games manage to maintain the user's attention with a background story, high-end graphics and the feeling of immersion within a simulated environment. Games can encourage active learning and motivate participation by giving rewards when students complete a task.

ITS researchers have begun exploring how games and features that are found in games (e.g., embodied agents) can be used in intelligent tutors. For example, McQuiggan, et al. [10] have examined how topics in middle school science could be taught using a tutor built on a commercial 3D-game engine. Students search an island science post to find clues to solve a mystery. While interacting with non-player characters and making observations in the virtual world, students learn scientific principles. [11] describes a tutor in which users learn cultural issues while interacting in a serious game. Gomez-Martin, et al. [12] have developed a system called *JV2M* which borrows ideas from games to teach programmers with Java knowledge the internal workings of the Java Virtual Machine. In some knowledge domains, games may be the only possible means of simulating and practicing real world problems. In the military for instance, simulations have been used for teaching pilots to fly as well as for training of combat scenarios that would otherwise be too deadly or expensive to train in the field [13].

Our work focuses on helping military trainers create these tutors easily for various training tasks using 3D games, typically first-person shooters (FPSs), as the simulation environment. Authoring of 3D game-based tutors is challenging due to their inherent domain complexity, the different kinds of feedback required and the interactions generated by various entities in the game apart from the player. These changes require an extension of xPST to enable a military trainer to author a game tutor. Before discussing the extensions required for xPST to enable authoring in 3D games, we discuss the core differences between 3D games and traditional GUI software or websites from the perspective of authoring ITSs.

## 3 3D Games Vs Traditional GUI Software or Websites

In a traditional GUI application or website, there are usually a set of controls (e.g. buttons, menus) that correspond one-to-one with a set of features. Some of these controls typically remain on screen while the user works. The two-part architecture typically consists of an application (e.g. Microsoft Word or Adobe Photoshop, or Amazon.com or Google.com) that has particular state while user-created content (e.g. a document, an image, or a query) is shaped by the user. A user's actions will typically evoke similar responses if done repetitively. In a game, on the other hand, a user's actions are frequently dependent on the context of other entities and the timecourse within the game. Rather than the user changing a file or query within an application that maintains a state, the gamer is focused on changing the application's state within the game state space. The states can be discretely defined, and they then act as the goalnodes: the user's goal is not to complete a textbox with a certain correct answer but rather to reach a specific state. The granularity at which these states needs to be defined depends on the author and the complexity of the task. The traditional GUI software and websites are typically a static system where all the events are triggered by the student (player). But 3D games are more like a dynamic system where interactions can happen between various entities in the game apart from the player and the events can be triggered by different entities in the game (e.g. by automated enemy players, or "constructive forces" in military terminology).

We categorize the events generated by the player and the events generated by other entities into player events and non-player events.

Unlike the traditional GUI software or websites, 3D games require the student to navigate through a simulated environment, the map, and sometimes communicate with the other entities in the game. This calls for the authoring system to provide tools to support tutoring on communication-based and location-based subtasks, e.g., "Identify yourself to the guard" or "Return to mission headquarters to give a report."

3D games and traditional GUI software also differ in the kinds of feedback required for tutoring. In GUI-based tutors, we have seen that two typical types of feedback *Hint* and *Just in Time error messages*, or *JITs*, are sufficient. A *Hint* corresponds to information about how to complete next goalnode in the sequence and is given on student's request ("What do I do next?"). A *JIT* corresponds to corrective feedback when the answer to a particular goalnode is wrong ("That's not quite right, because..."). *JIT*s are prompted by the incorrect answer itself. In 3D games, because the goalnodes can represent a much broader range of subtasks to accomplish than simply typing an answer in a box or choose a drop down menu option, and because it is sometimes not visually apparent whether the goalnode has been completed ("Did I reach the location?"), it is sometimes important for the learner to have feedback from the tutor that is neither requested, like a Hint, or based on an incorrect event (like a JIT). We call these Prompts. For example ("Good Job"),("Do this next.").

## 4 Extensions to xPST

In order to accommodate the differences between 3D games and traditional GUI software or websites, we have added the following extensions to the xPST architecture to facilitate easy authoring of ITSs in 3D games.

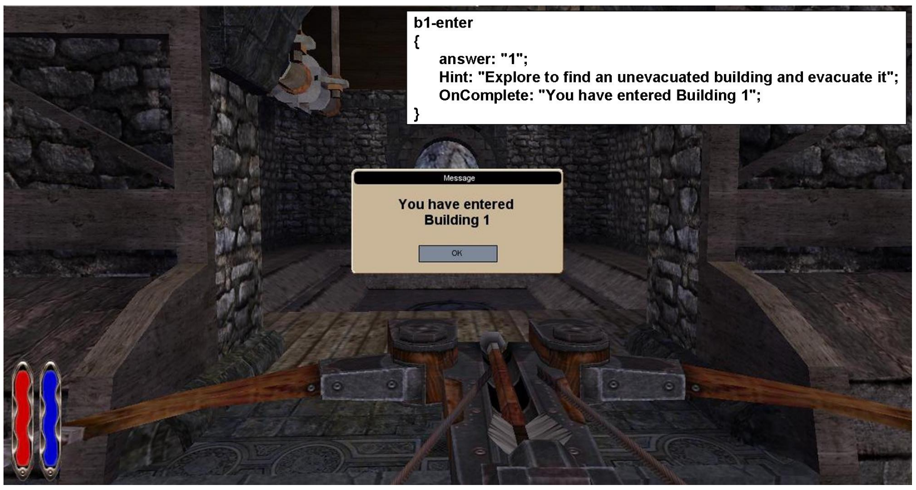
### 4.1 Actions by Non-Player Objects

Events can be triggered by non-player objects in 3D games. These events are modeled as hypothetical events by a Player-class object which is not the user.

For example *Avatar1:request-answer* is the goalnode corresponding to requesting an answer from the Avatar1 entity, where *Avatar1* is the unique id of the entity and *request-answer* is the associated action. This approach is useful in tutoring on generic actions associated with any entity in the game, such as Tanker1:explode, Enemy1:attack. This is a more generic way of handling events compared to the previous xPST architecture in which the unique ID attribute always corresponds to the Player class object and was hence ignored while writing in the file. This is because the previous xPST architecture could support tutoring only on events generated by the Player.

### 4.2 Proactive Hints or Prompts

Since the state space of a 3D game is quite complex with interactions between various entities, and since it is sometimes not obvious what the current game state is, it is sometimes useful for the learner to have direct feedback when the current goalnode is completed or to receive some reminders about the next goalnode. So we have included a new type of feedback in xPST, "*OnComplete*", supplementing the potential *Hints* and *JITs* for each goalnode. This feedback is proactively provided to the student on completion of that particular goalnode. Fig. 1 shows the scenario using the *OnComplete* feedback, which lets the learner know that he or she has entered Building 1 upon doing so (a location-based event). The screenshot shows a fantasy game environment. The media representing a more realistic scenario could easily be created and added by a graphic designer.



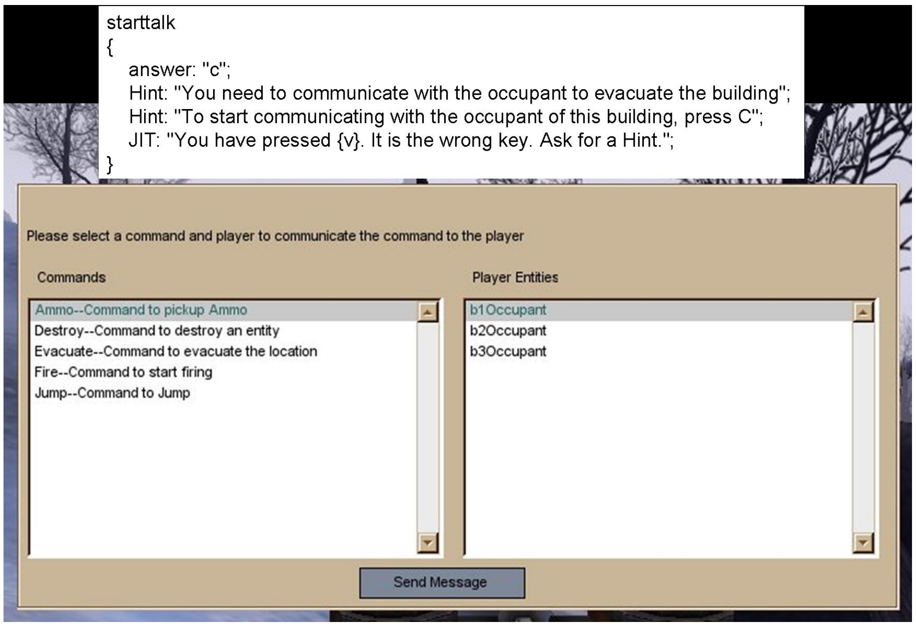
```
b1-enter
{
    answer: "1";
    Hint: "Explore to find an unevacuated building and evacuate it";
    OnComplete: "You have entered Building 1";
}
```

**Fig. 1.** A Location Event with feedback: "You have entered Building 1."

### 4.3 Communication Events

Unlike the traditional GUI software or websites, many of the tasks in 3D games require the student to be able to communicate with other player entities in the game. We have extended xPST to support tutoring on communication events by using a special goalnode *starttalk*, to initiate the communication with other entities. The student will be able to choose the entity with which to communicate and the message to communicate. This approach facilitates tutoring on the protocol of communication and the message that is being communicated, a common training task in the military, where communication is frequently highly-structured.

For example, Fig. 2 shows the instantiation of *starttalk* goalnode along with the UI for the user to communicate with the other players.



```
starttalk
{
    answer: "c";
    Hint: "You need to communicate with the occupant to evacuate the building";
    Hint: "To start communicating with the occupant of this building, press C";
    JIT: "You have pressed {v}. It is the wrong key. Ask for a Hint.";
}
```

Please select a command and player to communicate the command to the player

| Commands | Player Entities |
|---|---|
| Ammo--Command to pickup Ammo | b1Occupant |
| Destroy--Command to destroy an entity | b2Occupant |
| Evacuate--Command to evacuate the location | b3Occupant |
| Fire--Command to start firing | |
| Jump--Command to Jump | |

**Fig. 2.** Framework of Communication Events

If the student is supposed to choose *Evacuate* command for the task and if he chooses a different command, say, the Fire command, a JIT can be fired saying "*You used Fire command on this occupant. That's not something you need to do right now.*" This multiple-choice user interface for tutoring on communication is designed to tutor on communication protocol and procedure: what to say and when and how to say it. Future research will evaluate its effectiveness within military scenarios.

## 4.4 Location Events

Location events facilitate tutoring on the navigational aspects of the player's performance. Unlike the traditional GUI software or websites, almost every task in a 3D game requires the player to move within the virtual environment. The author can use the *entityid-enter* goalnode to tutor on when the player enters a particular designated location in the game.

For example, the goalnode *b1-enter* is triggered when the player enters building1 (b1). Fig. 1 shows the *b1-enter* goalnode along with the appropriate feedback given to the user in the game.

## 5 Torque Game Engine Advanced and TorqueScript

We have used Torque Game Engine Advanced (TGEA) as our simulation engine. It is a commercial off-the-shelf game engine from GarageGames. It provides various core functionalities required for game development like the rendering engine, physics engine, 3D graphs, collision detection etc. Instead of starting from the scratch, using an off-the-shelf game engine drastically reduces the game development time and helps the author concentrate more on the tutoring task.

TGEA supports scripting using TorqueScript. TorqueScript is similar in syntax to JavaScript and allows the developer to create modifications (mods) of the existing games. We have used TorqueScript to create the xPST Torque driver which contains two major modules, the Listener module and the Presentation module. The Listener module listens to the various events happening in the game and sends them to the xPST engine over the network. Then the xPST engine sends the appropriate tutoring feedback to the Listener module. This feedback is then presented to the user through the Presentation module.

The framework of the xPST driver can be leveraged to various other game engines by making the syntactical script changes required to be able to suit with that particular game engine. The Torque driver enables tutoring in 3D games created with Torque Game Engine. The Torque driver is one of the many possible interfaces to xPST and one of the several we have built. The xPST Firefox Plugin is an interface which is used to tutor on websites. The Paint.NET Driver is another interface which is used to tutor on Paint.NET, an image editing application. Likewise, many interfaces could be built to tutor with xPST on different existing software applications.

Fig. 3 shows the extended xPST architecture along with the relationship between various components of the system.

## 6 Evacuate Demo Task

We have developed a demonstration task called *Evacuate* to show that the extended xPST Framework can be used to create ITSs in 3D games. The task teaches the learner (player) how to evacuate the civilians from all the buildings in the scenario. The scenario has three buildings, each with one civilian inside. The player enters each building, checks for civilians present in it, communicates the *Evacuate* command, and waits for the civilian to come out. When the learner does this for all the buildings in the scenario, then the task is said to be successfully completed.

The xPST file or the cognitive model for this task contains three major goalnodes which illustrate the game-enabling extensions of xPST. The location-based *buildingid-enter* goalnode is completed when the player enters the building with id *buildingid.* The *buildingid-evacuate* goalnode is completed when the player sends the evacuate command to the civilian in the building with id *buildingid.* The *starttalk* goalnode is completed when the player initiates communication with the civilian. All these goalnodes are provided with appropriate *Hint*, *JIT*

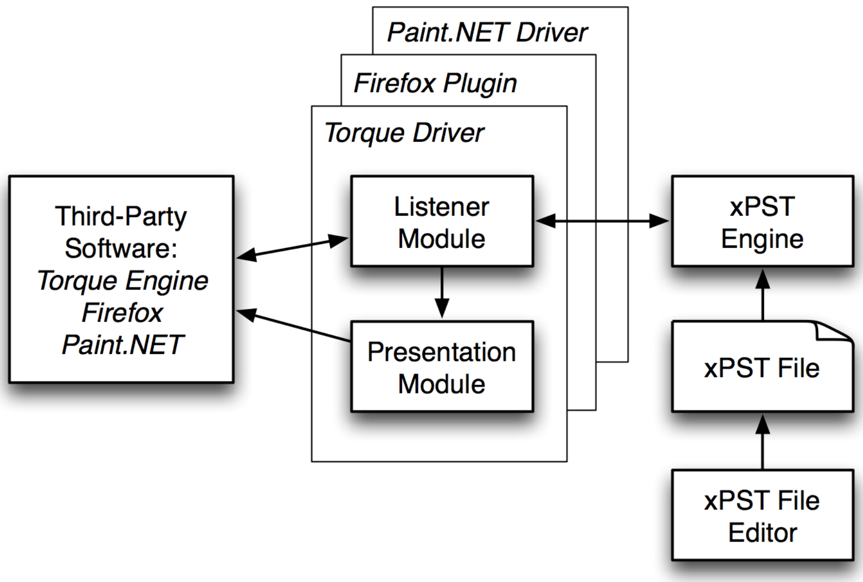


**Fig. 3.** Extended xPST architecture

and *OnComplete* feedback to guide the student in successfully completing the task. There are three sets of these goalnodes, one for each of the buildings. The purpose of choosing more than one building is to showcase the goalnode sequencing capabilities of the xPST framework; the order of building evacuation does not matter.

In general, we see that this framework makes authoring of tutors easy in a complex domain like 3D games. The four important steps required to create a game tutor from scratch include: 1) Create the tutoring scenario, which in effect consists of building the game map. 2) Give unique identifiers to each entity in the game on which you plan to tutor. 3) Make a list of the events corresponding to the entities chosen to tutor on and give appropriate mapping names in the mappings section of the xPST file. The mappings associate game-based events with the goalnodes that need to evaluate those events within the tutor. 4) Complete the feedback and sequence sections of the xPST file, listing the appropriate feedback for each goalnode and the sequence(s) in which the goalnodes may to be accomplished. Step 1 is perhaps the most difficult step, and is required of the scenario author even in the absence of a tutor. We have demonstrated in past research [5] that novice authors can accomplish steps 2-4 with little training in a simpler software setting, that of using a website to search an online database. Future research will explore whether military trainers could use a modification of those previous tools to overlay an xPST tutor on existing virtual training scenarios effectively.

This approach is very much extensible for tutoring on other game tasks. The potential author would be provided with a library of actions associated with different generic entities in games. The author would then use them in his or her xPST file as required by the tutoring task. Authors with programming knowledge requiring more specific tutoring actions could create more of these using TorqueScripting and add them to a library for their specific tutoring needs.

## 7 Conclusions

We have discussed the xPST framework which allows for fast creation of model-tracing tutor for a specific problem. We have also discussed the prime differences between 3D games and traditional GUI software or websites from the perspective of authoring ITSs. Understanding the differences between 3D games and traditional GUI software or websites, we have described the extensions that were required for the xPST framework to enable it to be able to tutor in 3D games. We have also described the game engine used and important subcomponents of the xPST Torque driver used to communicate with the xPST engine. Finally, we have discussed a demonstration task showing that the extended xPST framework can be used to tutor in 3D games.

As stated previously, our goal is to provide military trainers, as well as others who author scenarios in 3D environments, the ability to create in-scenario tutoring in an appropriate and easy-to-author manner. In the future, we would like to evaluate the programmability of this system by conducting a study where we examine how novice xPST authors with little programming experience can learn to create to these kinds of tutors.

**Acknowledgments.** We thank Steven Ourada as the senior architect of xPST. This work was supported in part by the National Science Foundation under OII-0548754 and by the Air Force Office of Scientific Research.

## References

1. Corbett, A. T., Koedinger, K. R., Anderson, J. R.: Intelligent tutoring systems. M. G. Helander, T. K. Landauer, P. Prabhu, (eds.) Handbook of Human-Computer Interaction. 2nd edition Elsevier Science, 849–874. (1997)
2. Hategekimana, C., Gilbert, S. Blessing, S.: Effectiveness of using an intelligent tutoring system to train users on off-the-shelf software. In: K. McFerrin et al. (eds.), Proc. of Society for Info. Tech. and Teacher Education Int?l Conf., AACE., pp. 414–419. (2008)
3. Aleven, V., Sewall, J., McLaren, B. M., Koedinger, K. R.: Rapid authoring of intelligent tutors for real-world and experimental use. In: Kinshuk, R. Koper, P. Kommers, P. Kirschner, D. G. Sampson, W. Didderen (eds.), Proceedings of the 6th IEEE International Conference on Advanced Learning Technologies, pp. 847–851. (2006)
4. Ritter, S., Koedinger, K.: An architecture for plug-in tutor agents. Journal of AIED, 7(3-4), 315–347 (1992)

5. Gilbert, S., Blessing, S. B., Kodavali, S.: The Extensible Problem-Specific Tutor (xPST): Evaluation of an API for Tutoring on Existing Interfaces. In: Proceedings of the 14th International Conference on Artificial Intelligence in Education (2009)
6. Roselli, R.J., Gilbert, S., Howard, L., Blessing, S. B., Raut, A., Pandian, P.: Integration of an Intelligent Tutoring System with a Web-based Authoring System to Develop Online Homework Assignments with Formative Feedback. American Society for Engineering Education Conference (2008)
7. Kirschner, P. A., Sweller, J., Clark, R. E.: Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching. Educational Psychologist, 41(2), 75–86 (2006)
8. D. Laurillard.: Rethinking university teaching: A framework for the effective use of educational technology. Routledge (1993)
9. J. Kirriemuir, A. McFarlane.: Literature reviews in games and learning. Technical Report Report 8, Nesta FutureLab Series (2004)
10. McQuiggan, S., Rowe, J., Lee, S., Lester, J. (2008).: Story-based learning: The impact of narrative on learning experiences and outcomes. In: Proceedings of the Ninth International Conference on Intelligent Tutoring Systems, Montreal, Canada, pp. 530-539. (2008)
11. Johnson, W. L.: A simulation-based approach to training operational cultural competence. In: Proceedings of ModSIM, Virginia Beach, VA. (2009)
12. M. Gomez-Martin, P. Gomez-Martin, and P. Gonzalez-Calero.: Game-driven intelligent tutoring systems. In: Proceedings of the Third International Conference on Entertainment Computing (ICEC)., pp. 108-?113. (2004)
13. R. H. Stottler.: Tactical action officer intelligent tutoring system(tao its). In: Proceedings of the Industry/Interservice, Training, Simulation and Education Conference (I/ITSEC 2000) (2000)
14. Blessing, S., Gilbert, S., Blankenship, L., Sanghvi, B.: From SDK to xPST: A New Way to Overlay a Tutor on Existing Software. In: Proceedings of the Twenty-Second International FLAIRS Conference (2009)