Iterations of an Open-Source 3D Game Engine: Multiplayer Environments for Learners

Jon Scoresby, Brett E. Shelton, Tim Stowell, Chad Coats, Michael R. Capell
Department of Instructional Technology and Learning Sciences
College of Education and Human Services
Utah State University, Logan, UT 84322; Tel: 435-797-2694, FAX: 435-797-2693
Email: jon.scoresby@aggiemail.usu.edu; brett.shelton@usu.edu; stowellt@gmail.com; kchadcoats@gmail.com; mcapell@ionicdesign.com

Abstract: As virtual 3D environments become more common in education, it is important to pay attention to and learn from the experiences that take place during the development of these environments. A team made up of graduate, undergraduate, and recently graduated students, took a conglomeration of open-source libraries and glued them together to create 3D simulation called *HEAT* (Hazard Emergency and Accident Training). The development team has gone through many iterations during the development of *HEAT,* while learning what it takes to build a fully functional simulation and 3D engine. We discuss the experiences and some of the processes the development team has gone through to create *HEAT,* the 3D engine, and the separation of the two for a more general use for other simulations. We discuss some of the benefits and challenges recognized during this experience.

Introduction

A team made up of graduate, undergraduate, and recently graduated students, took a conglomeration of open-source libraries and glued them together to create an effective 3D simulation engine (Shelton et al., under review). *HEAT* (Hazard, Emergency and Accident Training) is an open-source 3D simulation, built around the 3D engine, and is designed to train first responders in emergency situations (Scoresby, Stowell, Coats, & Shelton, 2007). Since the unveiling in 2007, we have refocused our efforts toward creating a more sustainable design for the engine, namely fleshing out the engine infrastructure to make it more reusable for the development of other simulation-based and game-based activities. This effort has resulted in many changes in the way we consider our open-source structure, and has led to significant trade-offs between the time spent on upgrading the features of our initial application (training of first responders) versus the time dedicated to making a reusable, sustainable tool with increased potential for completely different applications. Our efforts for this project are to offer a multiplayer tool that includes full facilitator interaction with game/simulation participants to demonstrate the power and usefulness of the learning environment.

Background

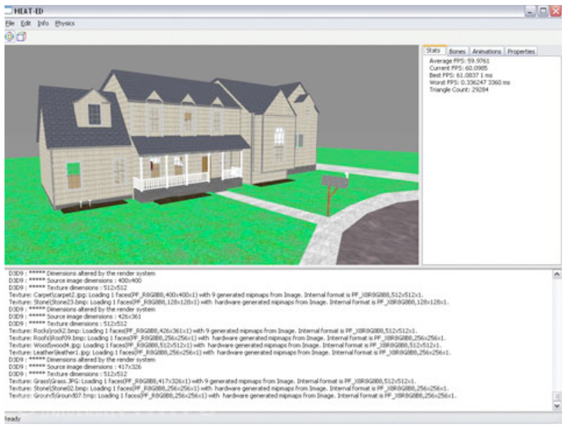
Virtual 3D environments are becoming more commonplace in their use for educational applications (Squire, 2007; Van Eck, 2007). The range of their use has been noted across disciplines and audiences (Gee, 2003; Barron & Bransford, 1993), and used in formal and informal learning environments (Aldrich, 2003; Kafai, 2006; Steinkuehler, 2006). Emphasis has recently been given to complex, problem-based and discovery types of learning within virtual 3D worlds, given their inherent properties that allow for rich, social interactions, task-based functionality and character empathy (Barab, S. et al, 2005; Dede, C. et al, 2004).

Other projects have noted the value of how constructing virtual environments, and even game environments, can provide students with special design-related and group-based learning experiences (Robertson & Good 2005, Barab, Hay, et.al. 2000). Most of these projects have relied on a pre-existing development platform on which to program. In other words, students work with a program to build new pieces of virtual spaces that are pre-rendered and pre-packaged; there requires little understanding of how the development platforms operate. Here, we present a project that relies on the creation of such a platform as a learning experience.

Market 3D engines have all the capabilities needed to build full-featured 3D environments. Development of these market 3D engines often requires vast resources of money and people and, therefore, market 3D engines cost a significant amount to purchase. Educational institutions rarely have the funds to purchase a current top-of-the-line engine, choosing instead to research alternate avenues. These avenues focus mainly on current open source engines or older engines that could be purchased for a lower price (Cruz, 2006). We found that these engines would not support our needs or were too dated to use for cutting edge applications. We chose instead to create our own 3D engine to do what we needed to, the way we wanted it done.

Development Experiences

In the beginning of *HEAT* development, the simulation and the engine were not two separate software applications as seen in Figure 1. In many ways the *HEAT* program and the 3D engine are still very much interconnected, but the development team is making efforts to standardize and generalize the 3D engine to make it completely separate from the current and future simulations. Over time the development team makes improvements to both the simulation and 3D engine. Each addition or improvement to the 3D engine is cause for a larger separation of the two applications. Finishing *HEAT* and separating the engine from *HEAT* is an ongoing process until the two are completely separate.



```
Average FPS: 59.9761
Current FPS: 60.0985
Best FPS: 61.0837 1 ms
Worst FPS: 0.336247 3360 ms
Triangle Count: 29284
```

```
D3D9 : ***** Dimensions altered by the render system
D3D9 : ***** Source image dimensions : 400x400
D3D9 : ***** Texture dimensions : 512x512
Texture: Carpet\carpet2.jpg: Loading 1 faces(PF_R8G8B8,400x400x1) with 9 generated mipmaps from Image. Internal format is PF_X8R8G8B8,512x512x1.
Texture: Stone\Stone23.bmp: Loading 1 faces(PF_R8G8B8,128x128x1) with  hardware generated mipmaps from Image. Internal format is PF_X8R8G8B8,128x128x1.
D3D9 : ***** Dimensions altered by the render system
D3D9 : ***** Source image dimensions : 426x361
D3D9 : ***** Texture dimensions : 512x512
Texture: Rocks\rock2.bmp: Loading 1 faces(PF_R8G8B8,426x361x1) with 9 generated mipmaps from Image. Internal format is PF_X8R8G8B8,512x512x1.
Texture: Roofs\Roof09.bmp: Loading 1 faces(PF_R8G8B8,256x256x1) with  hardware generated mipmaps from Image. Internal format is PF_X8R8G8B8,256x256x1.
Texture: Wood\wood4.jpg: Loading 1 faces(PF_R8G8B8,512x512x1) with  hardware generated mipmaps from Image. Internal format is PF_X8R8G8B8,512x512x1.
Texture: Leather\leather1.jpg: Loading 1 faces(PF_R8G8B8,256x256x1) with  hardware generated mipmaps from Image. Internal format is PF_X8R8G8B8,256x256x1.
D3D9 : ***** Dimensions altered by the render system
D3D9 : ***** Source image dimensions : 417x326
D3D9 : ***** Texture dimensions : 512x512
Texture: Grass\Grass.JPG: Loading 1 faces(PF_R8G8B8,417x326x1) with 9 generated mipmaps from Image. Internal format is PF_X8R8G8B8,512x512x1.
Texture: Stone\Stone02.bmp: Loading 1 faces(PF_R8G8B8,256x256x1) with  hardware generated mipmaps from Image. Internal format is PF_X8R8G8B8,256x256x1.
Texture: Ground\Ground07.bmp: Loading 1 faces(PF_R8G8B8,256x256x1) with  hardware generated mipmaps from Image. Internal format is PF_X8R8G8B8,256x256x1.
```

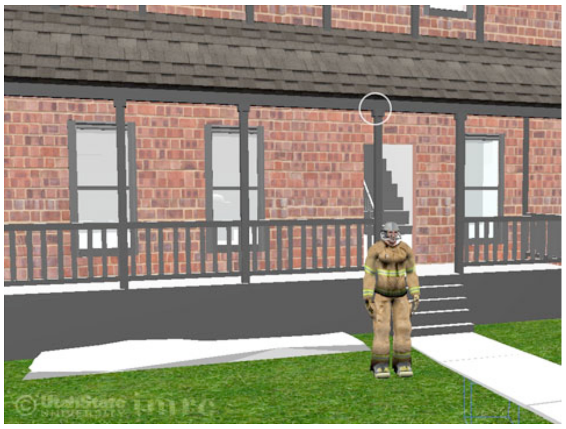
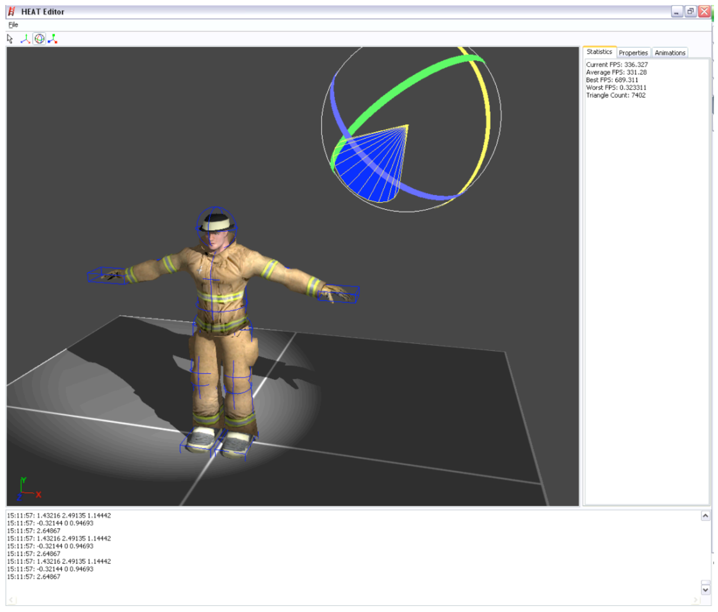


Figure1. Images of the house in the early stages of development. The image on the left is showing the house in an early iteration of simulation and engine combined. The image on the right shows a later iteration with the fireman and house in the environment without the surrounding engine properties showing.

Through the different cycles of the iterations, the developers have added to and taken away code to make *HEAT* work and separate the two applications. The process of separation requires moving the core engine code into its own dynamic link library (DLL). In this way, other applications can link to that DLL for the functionality needed. For example, the team has been working on an editor tool (e.g. Figure 2) to make the manipulation of the 3D environment easier. This tool needs access to the core engine code. In the past, the team tried to include the files from the *HEAT* engine project into the editor project, but this had several drawbacks. If someone is working on the engine and adds a new file, another developer may not know about it until later, and then would need to manually add it to the project. If the core engine was a DLL, the developers who link to it don't have to do as much manual work. All they need is the updated engine DLL. Another advantage to having the core engine in a DLL is it can provide a common interface for rendering, input, and relieving the need of every application that uses the engine to do the same.



```
Current FPS: 336.327
Average FPS: 331.28
Best FPS: 689.311
Worst FPS: 0.323311
Triangle Count: 7402
```

```
15:11:57: 1.43216 2.49135 1.14442
15:11:57: -0.32144 0 0.94693
15:11:57: 2.64867
15:11:57: 1.43216 2.49135 1.14442
15:11:57: -0.32144 0 0.94693
15:11:57: 2.64867
15:11:57: 1.43216 2.49135 1.14442
15:11:57: -0.32144 0 0.94693
15:11:57: 2.64867
```

Figure 2. A screen shot of the new editor. The editor uses the core engine code in the background to allow for development of current and future simulation without the use of *HEAT* specific code.

Another reason for separating the core engine code from *HEAT* is to define a totally generic message passing interface. The engine code does not need to know anything about users of the engine. For example, if an application wants to make a tree that moved and bounced around, the engine would need to provide the building blocks to make the tree object but not have a specific tree object within the engine about bouncing trees. When the 3D engine is completed, it will be used for more general purposes and not those only specific to *HEAT*.

As time and the experience of the developer progresses, improvements to *HEAT* and the engine are implemented with better technique and more understanding of how the software should run. Sometimes programmers have to take the long route to find the best answer. For example, the team's lead programmer researched and found the GUI (Graphical User Interface) library that was the most appropriate for the needs of the *HEAT* application (CEGUI). As progress continued, the creator of CEGUI stopped support so the development team stopped using this library out of fear of not having the support needed and switch GUI libraries for a lesser known but supported library (QuickGUI). The change caused problems because QuickGUI was not as powerful as the CEGUI and did not function as hoped. Support for the CEGUI system was later restarted and the team had to decide whether or not to switch back to the previous system. The decision was made to use the original GUI system and this proved to be a good decision because it is more mature and robust and there are not as many problems to worry about as in the newer, lesser known GUI system. Some problems the developers had with QuickGUI were how it displayed text and how it rendered multiple images. Text would randomly not show up and rendered images files had to be created every time the engine was started, adding extra time to startup. From this experience the team better understands the benefits and pitfalls of dealing with open-source code and community, as well as gaining experience with implementing different types of open-source libraries to better fit their needs.

By communicating with the open-source community, the progress for engine functionality has increased dramatically. The engine is becoming more stable, increasingly separate from the simulation, and will be used for other types of simulations. The team's programmers frequent the online forums in the open-source community to find what is being and has been done when creating software applications like *HEAT* and the 3D engine. The programmers find answers to questions and gather ideas as to what the next steps should be for the 3D engine development. Ideas also come to the programmers when developing the simulation side of the application. For example, in the early stages of *HEAT* development, the programmers wanted to have the ability to make changes to the simulation during runtime and not have to hardcode the changes and recompile. *LUA* is a scripting language that once implemented into the software, will allow the user to do user to make changes to the application during runtime. The programmers started with *LUA* because they wanted to move the logic of a simulation away from the engine. They realized that *LUA* did not have a well defined type structure and did not have native object oriented capability. So they looked at other scripting languages and found *Python*, which seemed more suitable for larger applications. Also it was object oriented and did have a well defined type system allowing them to quickly prototype logic without having to recompile the engine every time a simulation rule changed. Although the team can also code in C++ outside of the engine, *Python* is still faster a faster language to use for prototyping. Another example of engine functionality is the object system that was implemented. In the case of *HEAT*, the programmers would create a fireman in the simulation and apply different properties to that fireman. The programmers faced challenges after trying to create multiple firemen and sharing the properties across the network. For example, if one fireman had the hose or was running, the challenge was in updating the system so the other firemen knew who had the hose and could see a fireman running on their individual computers. These challenges were solved after creating and implementing an object system that applies to everything created within the virtual simulated world. The object system will allow the engine to create other scenarios now and not just *HEAT* related scenarios.

Building 3D Engine Functionality

The previous examples discuss how the programmers learned from the open-source community and their own experiences to improve the 3D engine. We next discuss the other functions being added to the engine to make it more functional and usable to create many different scenarios.

Network functionality has been added for multi-player, group activity, sound effects and voice over internet protocol (*VOIP*). The team began implementing the networking system by using the *Raknet* library but found it to be unpredictable in the way it affected the simulation. In response, the team switched to a "lower level" library called *Enet* that has proven much better because it is easier to find and fix bugs. *Raknet* has so layers of code that if something went wrong it was very hard to find the problem. *Enet* has also proven to load all data correctly and completely everything *HEAT* is run. For the current fire fighting training scenario, it was desirable to allow multiple users to participate in the simulation. This more accurately mimics a real-life emergency response situation where many different personnel arrive at the scene of a fire to choreograph an effective strategy. However, most full-featured commercial game engines also come with networking capability because many games benefit from the ability to have multiple players compete with each other at the same time. Along these lines, the team felt that the engine would benefit in a generic sense as well by having networking capabilities. For example different instructional games and simulations can benefit by allowing multiple players to interact and help each other.

Audio has been added to increase feelings of engagement and to emulate reality within the simulation activity. The audio portions also include the investigation of *VOIP* for communications between players at different locations. The team attempted to use the *OpenAL* library but found it very low level, thus requiring more work to implement. The team searched and found the *FMOD* library, which takes care of many details such as making sound come from the left or right speaker as the player turns different directions. *FMOD* can also handle occlusion, meaning that if there is a wall between two players, the sound produced by one player will sound muffled to the other player because of the wall between them. In most cases, the *HEAT* project is made up of open-source software mixed with the code created by *HEAT* programmers. Although the *FMOD* library is a commercial library, the team was given permission to use the library because of the *HEAT* project's non-profit status. Even though the programmers have had success and been helped with open-source libraries and the community, not all problems can be fixed within that area. If the team had not branched out beyond the borders of open-source, *FMOD* would not have been found and implemented. This was a positive learning experience for the team because they have worked with others in both the open-source and commercial world.

Inventory functionality has been implemented and expanded to include a variety of useful functions, with the potential to add more diverse features to the simulation. The realization that many simulations may require the use of an inventory system led the team to generalize such a structure independent of the needs of any specific simulation. Under this system the item itself, whether it is a fire hose or a breathing apparatus as in the case of the firefighter scenarios, acts and behaves according to a few select settings that could govern any item. Such concerns could be something like how an item is used in the simulation (is it worn, or held?). Beyond these simple features any simulation-specific requirements can then be addressed separately. The team can associate an item with an icon to be used to identify it in a menu setting and can define how the item responds to changes in a network environment. This way the team was can provide a general interface that may be used by any simulation to fill the need for an inventory system.

Specific educational features have been added to the engine to enhance teacher-student interactions and facilitate good feedback for formal and informal assessments as seen in Figure 3. One of these features includes DVR-type playback of the simulation, including fast-forward, rewind and pause. Another feature includes a "scenario regeneration" function that allows a (potentially) unlimited number of different new scenarios branching from an existing one—this function allows a group of learners to see how different decisions during a simulation can result in completely different outcomes without having to "recreate" the initial conditions of that scenario. The team developed a generic state system to make the timeline system possible. This was needed to allow playback at any given point in the recorded simulation. The states are also sent across the network so that each remote player gets data in a uniform way. In this manner, the core engine can supply or accept state data in a uniform way, and the game logic can supply that data in any way it wants, as long as the interface matches. The state system can also apply in a generic sense to other simulations because all objects in the virtual world have states. The team eventually plans on creating "state machines" which allow game logic to exist as states, i.e. "the hose is in the spraying state," or "the hose is in the off state."

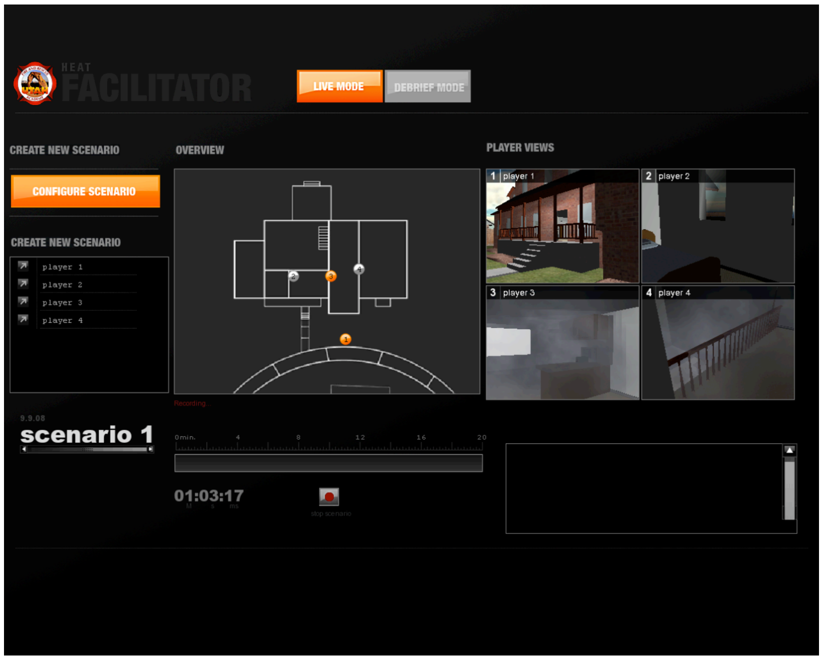


Figure 3. Specific educational features of the facilitator view showing, picture in picture, the scenario creation, and the DVR like playback function.

Future Possibilities and Educational Significance

Once the 3D engine is completed by generalizing the functions to the engine and not the simulation, the possibility to develop future simulations and games can be easily attainable for this team. They have the experience and knowledge of what developing a 3D engine entails and how it should run in an efficient manner. They have learned ways to find answers to questions, fix problems, and also help others. Software engineering is iterative in nature of development in general and its significance has certainly not been lost on the programming team. The revisions they have made to the system over the lifetime of the project to meet evolving standards and necessities supply the obvious evidence of this. In addition, the learning that has taken place during this project has refined the knowledge and skills used in all aspects of design and development.

By highlighting the changes and choices in the directions we have taken with the open-source 3D game engine, we hope to help inform educators, designers and developers interested in open technology and gaming resources navigate both pitfalls and beneficial threads of this type of work. We offered changes to the project's infrastructure and, hopefully, will start conversation about the basis of those decisions and exploring their repercussions. We discussed what aspects of our efforts to create a sustainable design can be generalized to other open-source projects. We also present new opportunities resulting from our sustainable design (e.g. Figure 4), including possibilities for projects with aircraft maintenance, hazardous materials, and information visualization.

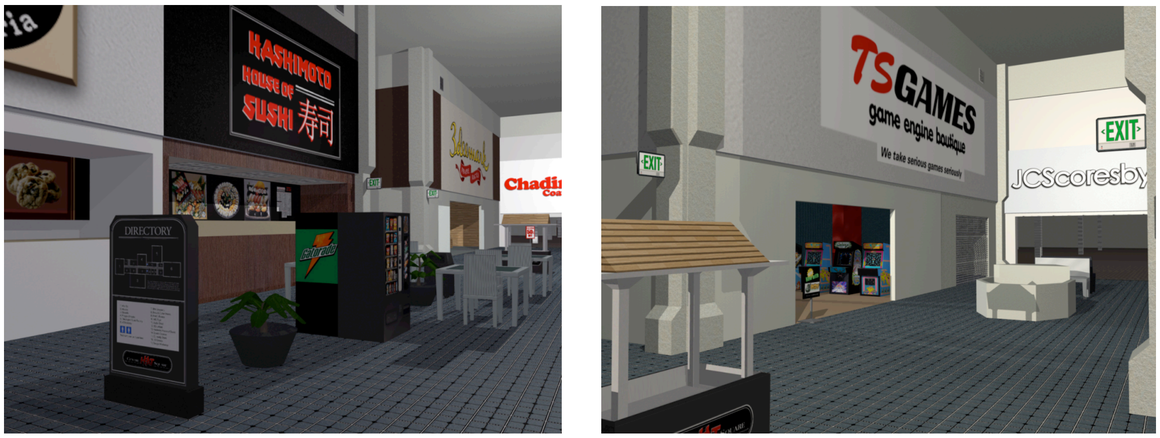

HASHIMOTO HOUSE OF SUSHI 寿司
DIRECTORY
EXIT
TS GAMES
game engine boutique
We take serious games seriously
EXIT
JCScoresby

Figure 4. Images for future simulation development: hostage situation in a mall.

References

Aldrich, C. (2003). *Simulations and the future of learning : An innovative (and perhaps revolutionary) Approach to e-Learning.* New York: Pfeiffer.

Barab, S. A., Hay, K. E., Barnett, M., & Keating, T. (2000). Virtual solar system project: Building understanding through model building. *Journal of Research in Science Teaching*, *37*(7), 719-756.

Barab, S., Thomas, M., Dodge, T., Carteaux, R., & Tuzun, H. (2005). Making learning fun: Quest Atlantis, a game without guns. *Educational Technology, Research, and Development,* 53(1), 86-107.

Barron, L., & Bransford, J. (1993). The Jasper experiment: Using video to furnish real world problem solving contexts. *Arithmetic Teacher*, 40(8), 474479.

Cruz D, Wieland T, & Ziegler, A. (2006). Evaluation criteria for free/open source software products based on project analysis. Software process: Improvement and Practice, 11(2), 107–122.

Dede, C., Ketelhut D., & Nelson B. (2004). Design-Based Research on Gender, Class, Race, and Ethnicity in a Multi-User Virtual Environment. Retrieved October 15, 2005, from http://muve.gse.harvard.edu/muvees2003/documents/AERADede04.pdf

Gee, J. P. (2003). *What Video Games Have to Teach Us About Learning and Literacy*. New York: Palgrave MacMillan.

Kafai, Y. B. (2006). Playing and making games for learning: Instructionist and constructionist perspectives for game studies. *Games and Culture*, 1(1), 36-40.

OGRE 3D: Open source graphics engine - What Is OGRE? Retrieved July, 30, 2007 from *http://www.ogre3d.org/index.php?option=com_content&task=view&id=19&Itemid=79*

Robertson, J.& Good, J. (2005). Story creation in virtual game worlds, *Communications of the ACM* 48(1)

Scoresby, J., Stowell, T., Coats, K. C., & Shelton, B. E. (2007). Remixing open-source materials for creating a 3D game engine: A developer's diary. Paper presented at the OpenEducation Conference 2007, Logan, UT.

Shelton, B.E., Scoresby, J., Stowell, T., Capell, M., Alvarez, M., & Coats, K.C., (Under Review). A Frankenstein Approach to Open Source: The Construction of a 3D Game Engine as Meaningful Educational Process.

Squire, K. (2007). Whereever You Go, There you are: Place-Based Augmented Reality Games for Learning. In B. E. Shelton & D. Wiley (Eds.), *The Design and Use of Simulation Computer Games in Education*. Rotterdam, The Netherlands: Sense Publishers

Steinkuehler, C. (2006a). The mangle of play. *Games & Culture*, 1(3), 1–14.

Van Eck, R. (2007). Six Ideas in Search of a Discipline. In B. E. Shelton & D. Wiley (Eds.), *The Design and Use of Simulation Computer Games in Education*. Rotterdam, The Netherlands: Sense Publishers