Critical Factors and Resources in Developing a Game-Based Learning (GBL) Environment Using Free and Open Source Software (FOSS)

http://dx.doi.org/10.3991/ijet.v8i6.2918

J. M. Lothian[1], J. Ryoo[2]
[1]The Pennsylvania State University, University Park, United States
[2]The Pennsylvania State University, Altoona, United States

Abstract—**Engaging students in learning is often a challenge. It is even more so when the subject matter is non-trivial and requires a significant effort to master. Game-Based Learning (GBL) makes learning more interesting and appealing by seamlessly incorporating educational lessons into competitive games. Students naturally develop their interest in the materials and are immersed into learning as they compete with each other or against themselves in the game. To be effective, the game itself should be fun and engaging as well as accommodating the intended learning objectives. Although many people are aware of how effective GBL can be, it is overwhelming for a beginner to master the tools and techniques quickly to have GBL implemented in a classroom environment. We recognize this lack of guidance in the existing GBL literature and discuss critical factors in developing a GBL environment using the free and open source software (FOSS) resources available as of this writing.**

Index Terms—**game-based learning, free and open source, platform, critical factors**

I. INTRODUCTION

Game-Based Learning (GBL) uses competitive exercises to motivate student learning according to specific learning objectives [1]. Serious games also have a similar goal but differ from GBL since the former tend to imply that the only purposes of the game is educational in nature while the latter can include the use of non-educational games for educational purposes.

The students can compete either with other students or against themselves. The games used in GBL usually employ an interesting narrative deliberately designed to engage the students in learning. Although GBL does not necessarily have to rely on computers, the type of GBL discussed in this paper only refers to those implemented in digital media.

In recent years, attempts to use video games for educational purposes have received much attention across the globe, with studies being conducted in several countries including Australia [2], China [3, 4], Greece [5], Norway [6], and the United States [7-9]. These games have been used at several levels of education: elementary (primary) school [5], middle school [10, 11], high school [6, 9], university (post-secondary) [2, 3, 7], and even education for retired seniors [4]. Additionally, they include a wide variety of game genres: virtual worlds and environments [2, 7, 8, 10, 12], simulations [13], turn-based strategy [10, 14], and card games [5]. These studies cover a diverse range of educational topics including healthcare, physics, biology, engineering, mathematics, physical education, social studies, economics, geography, and history.

Results from these studies have been mixed from reasonably effective [2, 5, 12] to limited or no gains over more traditional methods [7, 9], but the literature does not indicate that they are ineffective or less effective than traditional methods. Researchers have indicated that poor communication between the game designers and the instructor can lead to a game that does not integrate well into the existing lesson plan or teaching style of the instructor [9] and that a lack of performance might be mitigated by better understanding which game features can teach what concepts more effectively [2].

De Freitas [15] and Wilson et al. [16] provide more information on the theory, design, and evaluation of GBL activities and Serious Games, with explanation of why certain features of games are more effective at teaching certain types of material than other features.

Interestingly, games have also been demonstrated as an effective way to close learning gaps among previously underperforming students [12] or in situations where the topic generally demonstrates a gender preference for learning [11].

In addition to more effectively motivating students to learn, GBL environments also encourage them to learn from and adapt to each other's tactics and play styles [17].

These attributes are of particular importance when teaching the "net-generation" of students. As Van Eck suggests, these students "require multiple streams of information, prefer inductive reasoning, want frequent and quick interactions with content, and have exceptional visual literacy skills," all of which, he notes, match features provided by GBL [18].

Despite the many benefits of GBL, the adoption of GBL presents a number of significant challenges. Two of these are the steep learning curve and expense associated with developing the game itself. Luckily, there are many existing free and open source game development platforms, and as a result, a GBL advocate does not have to start from scratch to build a game. However, the sheer number and variety of factors to consider in developing an effective GBL environment are overwhelming to beginners. Therefore, some guidance is critical for a user to be able to successfully navigate a development process as well as choose an appropriate GBL platform for his or her needs, but the current literature fails to provide appropriate level of guidance.

To address this deficiency, this paper focuses on the technical and logistic components of developing effective GBL modules rather than delving deeply into the educational learning theories behind them, which also involve learning objectives, assessment, pedagogy, etc. However, this paper will still point to resources for the educational side of GBL as well. We also provide practical advice to those newly adopting GBL in their curriculum, which makes this paper serve as a primer for the new adopters of the GBL platforms. We assume some familiarity with software development and video games, but more extensive resources for understanding these concepts are provided in section IV.

II. Elements of Educational Game Creation

For game-based learning, careful construction or selection of a game is essential. In either case, the game needs to represent the instructional topic well, while still being fun to the students. If a research team is implementing a game, then it should work closely with the instructors using the game in the classroom, or field experts to incorporate the learning objectives directly into the game design[9]. The more seamlessly integrated the learning materials are to the game experience, the more potential there is for learning [18, 19]. This integration needs to occur at multiple levels; game production, pedagogical composition, classroom facilitation, and the game itself are all highly interconnected.

A. Game Production Team

Game design, much like educational instruction, is a specialized discipline that requires unique knowledge, skills, and abilities for good performance and productivity. While game designers who can play more than a single role certainly exist, it is more likely that the game production team will consist of at least a few individuals who specialize in different areas. There might be a **producer** – the individual in charge of the overall feel of the game, its intention and direction, and potentially the project and team management. In a research setting, the producer is probably the principal investigator (PI).

The team will also need one or more **programmers**. As with any programming project, programmers should be able to dedicate significant time towards the project. Games can be highly complex, and having to train new programmers can be a significant overhead. Depending on the complexity of the game or engine, and the time the programmer can dedicate to the project, simply learning an existing system could even take weeks, while halting continued production.

Additionally, the game may require custom graphics or sound. Often in a university setting, it may be possible to outsource these tasks to students in art, media, or theater departments. Sometimes this work can be free, or conducted as part of a class project. Other times if the game requires significant custom media, it may be necessary to hire one or more dedicated team members for this area. A dedicated team member in this area would also be useful for providing insight into creating a consistent look or feel to the game.

In the context of a research-oriented educational game (i.e., a game whose main purpose is to test one or more research hypothesis in education or other fields), the **game designer** is a vital component. This individual should have knowledge in a broad range of topics. Designers should understand the concepts of game balance, enjoyment, and player motivations. In addition, they should have an understanding of the requirements for implementing features. While they are not responsible for programming on the project, designers must comprehend the complexities of adding different kinds of features or mechanics to the game. As Bartle notes, one of the primary traits a designer should have is the ability to look at the game objectively, knowing when a feature concept is bad, and avoiding suggesting features just because they sound "cool"[20]. If the team is not very large, the game designer may also take some of the responsibilities normally given to a producer, and act as a liaison between the customers and the other developers.

B. Pedagogical Components and Classroom Facilitation

An educational video game should not simply be forced into an existing instructional setting or plan. They are not modular components that fit into every educational scenario, or a magic tool that makes learning (and teaching) fun and easy [7, 9, 15, 16, 21]. Game-based learning requires that (1) a knowledgeable instructor facilitate the game properly, (2) the game is carefully integrated with the learning material, and (3) the students are receptive to the game as an educational tool.

Annetta derived 6 "I's" of educational game design integrated into a classroom setting from several years of experience and research [22]. He presents them as nested elements in the order from inner to outer: Identity, Immersion, Interactivity, Increasing Complexity, Informed Teaching, and Instructional.

The first four of these components refer to the players' perception of and their interactions within the game. The last two encapsulating the others refer to the integration of the game within the educational environment.

Informed Teaching is the idea that the integration of an educational game is not a one-time event, but requires the instructor to observe how the game is used by students. They can then adjust the instrumentation of the game during subsequent iterations. That is, if the usage of a game is not a complete success in the first try, the instructor and designer can be informed by the attempt and make alterations to increase the game's effectiveness in the future.

In the final outer layer, Instructional, Annetta mentions the importance of the instructor when using an educational game: "the teacher is responsible for creating scaffold-structuring interactions and developing instruction in small steps based on tasks the learner is already capable of performing independently" [22]. Instructors should not expect the game to teach the students independently. Games should be integrated into the scaffolding of a larger lesson, and facilitated with additional instructions that guide the students in the learning activity [5, 22].

Additionally, Annetta mentions that one area where video games work well is with implicit (or incidental) learning, where the student is not actively aware of the educational content embedded in the game.

This issue is addressed by a framework for evaluating the effectiveness of educational games provided by de Freitas and Oliver [21]. They present the concept of debriefing from studies on educational simulations to educational games. It is important to facilitate discussions after playing an educational game not only to determine what each individual student has learned, but also for students to share what they have learned with each other. By facilitating proper debriefing, the instructor assists the students in teaching each other, while ensuring that the class as a whole has acquired complete mental representation of the learning objectives.

1) Students

It is important to consider the target audience for the educational game as well, the students. A 2008 survey by Pew Internet supported by the MacArthur Foundation found that 97% of students between the ages of 12 and 17 years old play video games [24]. This provides an immediate advantage to instructors implementing games for education, since most (if not all) of their students will already be familiar with the media.

However, it also presents two distinct disadvantages to developers implementing educational games. Firstly, because students' primary exposure is with commercially available, high-budget video games, they have significantly high expectations in terms of the quality of the final product [9, 25]. Specifically, students expect high quality graphics [9, 26, 27]. This can create a relatively high barrier for creating a video game from scratch, which is acceptable to students. Nevertheless, by utilizing existing software libraries, and even some commercial video games, this obstruction can be successfully mitigated.

The second issue, as Virvou et al. mention, is that some students may not like using games for the purposes of education, or may not be proficient with the specific type of game being used [12]. Rather than encouraging these students, video games may discourage them from the learning process. While unfortunate, the instructor should prepare for this and may choose to handle it on a case-by-case basis. Nevertheless, if resources and time allow, it might be beneficial to include a non-videogame version of the lesson to accommodate these students in the rare instances that occur.

2) Learning theory within the design-motivations

Deci et al., using the Self Determination Theory (SDT), argue that students who are more intrinsically motivated are more likely to have a positive experience and succeed in an educational activity [28]. For this reason, it is important for game designers to understand the motivations of their player students in creating a productive educational gaming experience. Fortunately, there has been a recent surge in research attempting to understand the reasons that people enjoy playing video games.

Recently, the discussion of motivating factors for playing video games has been an important topic among many researchers [29-34]. The factors from these studies are not new, however. Malone and Lepper described many of them much earlier [19]. Rather than simply describing motivations to play video games, they described a taxonomy of motivations within the context of an educational situation, with each subcategory grounded in existing educational theory.

This taxonomy is divided into two major sections. The first is Individual Motivations. This contains the categories of challenge (goals, uncertain outcomes, performance feedback, and self-esteem), curiosity (audio/visual sensory enhancement and cognitive curiosity), control (responsive learning environment, ability of the student to make their own choices, and whether those choices should generate powerful effects), and fantasy (identification of students with game characters, the game fiction closely matching the intended educational topic). The second section is Interpersonal Motivations that contain the student-to-student interactions of cooperation, competition, and recognition. In addition to including many of the factors established by Yee and Sherry, the taxonomy of Malone and Lepper integrates many of Annetta's layers (Identity, Immersion, Interactivity, and Increasing Complexity).

C. Game Application Components

The implementation of a game can go from relatively simple, to systems that are more complex than some of the most sophisticated commercial productivity software. There are four primary components to consider as part of the game itself: the game engine, the game mechanics, narrative components, the visual and audio media assets needed for the game, and game types.

1) Game Engine

The game engine is the primary set of programming logic that facilitates game play. This can be as simple as a single webpage, or as complex as a complete, distributed client-server architecture. The engine may support a single player at a time, or multiple players interacting with the game and each other. It might be implemented entirely as a text interface, as 2D graphics, or a sophisticated 3D representation. There may be simple game rules, or detailed physics.

The scope of the game engine should fit within the facilities of the available tools, and match the skills of the team. This is one of the aspects where having a good designer or project manager is essential. The designer or manager should be able to limit the scope of the game engine to help fit both the educational needs of the game and development within the time constraints of the project, and the skill sets and resources of the other developers.

2) Game Mechanics

A game mechanic is a rule or structure provided by the game, which describes how features work within the game. With game-based learning, the mechanics need to provide both an education, and a fun experience. They should also closely match the topic of intended learning. The mechanics of a real-time strategy game may not be the best one to teach digital privacy awareness (although clever individuals may figure out a creative way to make this work). Game mechanics also regulate game balance.

Game balance is a complex topic itself, but simply put, it is the equilibrium among achievement in the game, the difficulty of the task, and the enjoyment of each player. One indication of a good balance is described in the media effects flow theory, originally defined by Csikszentmihalyi, and applied to video games by several others [22, 35-37]. If players cannot achieve the goals of the game, they will not enjoy playing it. Likewise, if the game is not very difficult, their enjoyment will decrease. This theory is supported by the motivational factors proposed by Malone and Lepper (challenge and control), Sherry et al. (challenge and arousal), and Annetta (increasing complexity) [19, 22, 34].

Additionally, game balance also includes the balance between different avatars and how the player progresses. If the game allows selection of different avatar types, or customization of skills as the player advances, then the different possible avatar choices the player makes should provide equal, or similar, opportunities for the player to progress.

3) Narrative

The depth of narrative for the game will vary by the specific context. In some instances, it may even be possible for the narrative to be entirely external to the game, delivered by the instructor. In others, it may be necessary to include a deep narrative within the game itself.

In some cases, the narrative itself may be part of the subject being researched. For instance, Aylett et al. examined an AI-driven emergent narrative in their game called "FearNot!," which is used to teach anti-bullying behaviors [38].

Nevertheless, while there has been considerable research in narratives as they apply to education, within the specialized context of educational video games the field is open for additional study.

4) Media Assets

Whether the game runs from a webpage, or from a complex 3D engine, artistic assets are often necessary. These include audio files, 2D graphics, 3D models, and textures, and in the case of webpages, this might be CSS layouts and Flash objects. The quality of the game assets can be as important as (sometimes more important than) the game engine and mechanics. This is not to say that a great game requires the most detailed 3D models available – recent hits such as World of Warcraft, Magicka, and Minecraft have used simplified 3D models to incredible success. Even Terraria, with its 2D graphics reminiscent of a video game from the 1980's, was an instant sensation. However, the quality and theme of the game assets should provide a consistent look and feel for the game [20].

D. Single versus Multi-Player Games

Multi-player support currently drives much of the video game market from consoles to PCs. However, the decision on whether to implement a single player or multi-player game can have significant implications for both the classroom facilitation and the game development process.

In a competitive, multi-player game where students interact with each other, they may be far more sensitive to the issues of game balance and fair play, which could significantly increase the amount of testing required before the game can be used in a classroom. Additionally, even with a cooperative game, the designer must pay attention to the design for the interactions between different players and how those interactions affect the educational context of the game.

From a technical standpoint, multiplayer games can require a significant amount of additional time and resources to implement. Not all game engines or libraries provide an easy-to-use system for implementing networked game play. Additionally, many multi-player situations include an additional point of failure – the server. The hardware aspect of the server can increase the difficulty of maintaining a functional game environment, and may require additional support and configuration from an IT department. The software aspect requires developers to program and maintain an additional (non-trivial) component in the system.

Some uses of multi-player games can also be simulated using single-player games with an adequate scoring system. Scores in single player games can allow players to both compete against each other, and in some cases, cooperate.

III. FREE AND OPEN SOURCE PLATFORMS FOR GAME-BASED LEARNING (GBL)

A. Free Software versus Free Open-Source Software

One important consideration when selecting a platform for game development is whether the development will require access to the source code of the game engine itself. Often, game engines may restrict access to their source code and provide alternative methods for customizing their source code. This is the primary separation between free game engines and free open-source software (FOSS) engines. With a FOSS game engine, the developer has complete access to all of the game engine's programming source code, and is free to modify the engine as needed. For researchers, this generally means that either the engine is used as-is, only adding onto it, or that they must be willing to learn the internals of a system that can be quite complex if they want to make any significant changes.

For instance, the OpenSimulator (OpenSim) project is an open-source version of the SecondLife server architecture, programmed in C# for the Mono implementation of Microsoft's .Net framework. Development of environments within OpenSim follows the same basic procedure as SecondLife – users connect through a client, and create objects and scripts within the environment itself, never having to alter the underlying server architecture. Scripts are written in the standard Linden Scripting Language, or other programming languages that Mono recognizes, such as C#. In this way, it is similar to many closed-source projects. However, the scripts only have access to libraries that were included when the server architecture was compiled. Including additional libraries, or altering the way scripts behave requires the developer to make modifications to the architecture source code. Additionally, complex modifications may also result in the changes in the behavior of the client used to connect to OpenSim.

Closed-source alternatives, however, generally contain more features, and are specifically designed to be flexible and make game creation easier. For instance, the Unity3D game engine, in addition to including most of the features provided by OpenSim, also includes several tools specifically for developing games and managing game content.

While these additional features may be useful in some situations, they are not always required. Picking the proper game engine should not rely on the total number of features it has, but rather on how they fit with the specific project. For example, OpenSim provides avatar customization to the user, complete with animations, while a closed-source free platform may not include those customizable 3D models or animations, leaving them up to the developer to create or otherwise acquire. Depending on the game being developed, a pre-existing avatar system may be easier to work with than developing a custom-made system.

Additionally, working with FOSS game engines will generally mean interacting with the community developing the engine and understanding how the engines are licensed.

B. Open Source Licenses

A developer considering FOSS game engines will undoubtedly encounter a few key acronyms and license names during the selection process. Terms seen frequently might include GPL, BSD, Apache, and LGPL. Explaining all of the subtle intricacies of these licenses is beyond the scope of this article. However, some of the key differences are discussed in this section.

Software licensed under the GNU Public License (GPL) can be used internally by an organization without releasing the source code to the public [39]. If a modified version of the software or a project using the GPL-licensed software is released to the public, then the entire source code of the project must also be released. Alternatively, software licensed under the BSD or Apache licenses allows the developer to publish modified versions of the software, or programs using the software without releasing any source code.

The GPL should not be confused with the LGPL (Lesser GPL) license. This license is an alternative form of the GPL license that allows developers to use libraries or engines, and only requires developers to release changes they make to the original source code. This means that developers are free to use the engine or library in their project without releasing the source code of the entire project except for the changes they make to the library.

C. Acquiring an Open-Source Engine or Library

The first challenge of any researcher or developer who has selected an open source game platform is getting it. Depending on the project, the software developers may have a variety of methods for acquiring the engine and setting it up for use. Different methods each have their own drawbacks and benefits.

1) Pre-Compiled Binaries

The easiest method of using an engine is when the developer provides a pre-compiled binary (PCB) download. With a PCB, generally, the source-code is not immediately available, but the developers have provided all the files needed to run the game engine, or to use the library in a project. Developers may still need to configure certain settings, or their development environment, but they do not need to compile the engine source code themselves. This can be a tremendous timesaver during the initial setup of a project, or during the evaluation of multiple engines.

For continued development, however, it may eventually become difficult to work with. For example, compiler settings used to create the PCB may limit developers. These settings can include options such as enabling debugging information, threading options, or speed and memory optimizations. Additionally, PCB releases are usually "stable" releases of the source code. These releases compile properly, with a minimal number of known bugs, and all the intended features for the version. In general, this is a benefit, as the developer knows that they are using something that should work most of the time. However, it also means that they may not have access to the latest bug fixes, or feature improvements.

2) Source Access – Compressed Formats

To address some of these issues, the developer can consider downloading a source package in a compressed format such as tar.gz, zip, or rar. Source packages will allow the developer to compile their own version of the engine or library with their own compiler settings and provide complete access to the source code for debugging. These packages may still only be available as stable releases, but with access to the source code, developers can include their own bug fixes if necessary.

There is also an additional complexity added, in that the project source may require other projects, called *dependencies*, to compile properly. In particularly complex scenarios, the dependencies may have their own dependencies, which can significantly add to setup time. In a Linux environment, the dependency issue may be mitigated by using a package management system (e.g. YUM and APT), which handles the entire process of finding and installing dependencies for the user.

That said, many open-source game engines attempt to limit the number of dependencies, and minimize the effort used to gather and compile them with very specific and thorough instructions.

3) Source Access - Version Control Systems

If a developer needs up-to-the-moment source code for the game engine or library, most projects provide access to their source code versioning repository. Source code versioning can include (but are not limited to) systems such as CVS, SVN, GIT, and Mercurial. Each of these systems requires a different client interface, which is available as a command-line tool, as well as a graphical program for most operating systems. The operations of each of these tools is beyond the scope of this paper, but suffice it to say that they provide a means to browse different versions of the source code, and download source to be compiled. This gives the researcher access to the latest code available, which can include critical bug fixes for issues they may have encountered. The primary drawback also stems from this ability – because the code is highly recent and possibly not well tested, it is easy to encounter new bugs introduced between stable versions, which others have not yet discovered.

D. Interacting with Developers

Open-source software is often community-driven. This requires some methods of maintaining communication between different developers and tracking tasks. By utilizing an open-source platform, the researcher has become a part of that development community to a certain degree. Thus, they should be familiar with how the community is organized. Many open-source projects utilize a variety of methods for communication and task-tracking. E-mail lists are very common, and they are often open to any individual to participate. E-mail lists are also often archived online, which provides the research team with an additional source of information regarding the project. These lists might require a user to register before participation. This is usually done either by filling out a form on a webpage, or emailing a registration service requesting to be added. Once registered, researchers can participate in the community discussion and ask questions to the developers.

We found that the lead developers were highly accessible via their developer email list while working on a project using OpenSim. The developers responded quite quickly, with very informative replies that greatly reduced the amount of time required to implement some of the features we needed.

In addition to E-mail lists, the development community often uses some form of project management or bug-tracking system. This can be useful to developers in a number of ways. Firstly, they can track the development of specific features of the platform – features that the research project might require. Secondly, if the researcher discovers what he or she thinks is a bug, or strange behavior, the person can check the tracking system to see if others have discovered the same thing, and see if anyone has provided a solution or fix for the issue. Lastly, as part of the development community, the researchers can use these system to provide their own bug reports and feature requests, as well as provide any improvements they have developed for their own use to improve the project as a whole.

E. Examples in Research

Many free and open-source software libraries and game development platforms have already been used for educational purposes in research. The following is a non-exhaustive summary of examples.

1) Virtual Worlds – Second Life Alternatives

With the proprietary and difficult nature of working within the virtual world environment called Second Life, several alternatives have been developed recently. Two examples of these alternatives are OpenSim (mentioned previously) and Open Wonderland [40, 41]. We have been working on a game-based learning environment called Immersive Security Education Environment (I-SEE) and originally implemented it using Second Life, but found that environment too constraining and moved our project to OpenSim [8]. Because OpenSim is compatible with the Second Life architecture, much of the existing code could be moved without considerable modification. This shows that OpenSim is a viable platform for researchers currently using Second Life but are looking for more flexibility.

As an additional alternative to Second Life, Open Wonderland was originally developed by employees at Sun Microsystems (now Oracle) to provide a much more extensible and controllable platform than offered with Second Life [42]. This toolkit utilizes a Java programming codebase, potentially giving it the same kind of platform independence as OpenSim [40]. Additionally, it supports features for voice communication, and advanced extensibility.

Similar to I-SEE, Parsons and Stockdale report their project moving from Second Life to Open Wonderland [43]. They provide a technical evaluation of Open Wonderland for this purpose. One of the important contributions of this evaluation was the finding that despite Java's platform independence, applications written in it still suffer from various configuration problems on certain systems. Additionally, the Open Wonderland client must download the 3D content of the virtual world when it first connects. If the content is large, it can take a considerable amount of time before a user can participate in the environment, leading Parsons and Stockdale to suggest "priming" the clients before actually using them in a classroom.

2) 3D Engines

While the open-source virtual world packages offer flexibility and extensibility, researchers are still confined by certain limitations imposed by the design decisions and assumptions of the original developers. Additionally, some educational games may not need the full implementation of a virtual world. For these reasons, a developer may want to consider using an openly available 3D game engine.

Panda3D is a game engine originally developed by Disney for virtual reality theme park attractions, and eventually used for other video games such as Disney's MMORPG Toontown [44-46]. Originally being a closed-source, purely commercial project, Panda3D was open-sourced, and Carnegie Mellon University became involved in its development [47]. Developers using Panda3D have a choice between using Python and C++ for programming.

Henrich and Reuter used Panda3D to implement a game that teaches safer driving [48]. They integrated an open-source ridged-body physics library (Open Dynamics Engine) into their game to provide a realistic simulation of driving a car. Although having had utilized open-source libraries, there were still many issues to consider for their project. The integration of two different open-source tools solved some problems but generated additional challenges. Because they were using a physics library that was separate from Panda3D's integrated system, it required them to write functions to translate between the different physical systems. Additionally, for the physics to be accurate, the 3D models had to be constructed to be mechanically consistent with real-world cars. Since enabling precise physics on every object caused considerable performance loss, the team implemented a Level of Detail algorithm that selectively turned on physics for objects near the car, and kept physics off for objects far away.

On the opposite end of the commercial and institutionally supported spectrum of Panda3D is the purely open-source community-supported project called Ogre3D [49]. While Ogre3D incorporates many advanced graphics features such as shaders, flexible shapes and skeletal animations, and customizable scene management, it is not as fully featured as the Panda3D engine. Nevertheless, sometimes a developer only needs a graphics engine, and working around the integrated components of a full game-ready platform can be more difficult. For instance, Ogre3D is used as the graphics front-end for the ION Framework environment for intelligent agents [50]. ION was created to facilitate separating an agent-based virtual environment from what Vala et al. call the *realization engine*, which is the visual representation of that environment including the graphics engine and potentially physics engine.

By separating these factors from the virtual environment, their engine is suitable for a number of agent-based simulations. They use a case study of an educational game as an example of the usage of this framework. The game, FearNot!, is designed to teach anti-bullying awareness and techniques [50-52]. Having a separate graphics engine allowed them to separate the agent simulation from their graphical representation, which permits the re-use of framework components for multiple simulations.

Certain research projects may require specific features from the game environment. This is generally the case for games implemented within the Delta3D gaming and simulation engine [53]. Delta3D was developed with a very specific purpose in mind – support for Department of Defense (DoD) simulations and games. This engine integrates several other open-source projects (Open Scene Graph, Open Dynamics Engine, Character Animation Library, and OpenAL) into a single framework that supports DoD community standards for simulation such as the

High Level Architecture (HLA), After Action Review (AAR), and SCORM Learning Management Systems (LMS) [53-55].

Delta3D has already been used to move Marine Corps training simulations from a commercial solution to lower cost of deployment and increase utilization [55]. An additional case-study implements a simulation to train Forward Air Controllers (Airborn) (FAC-A). McDowell et al. also mention the recent inclusion of an artificial intelligence planning framework derived from a model implemented in the popular video game F.E.A.R [55, 56]. The flexibility of their approach developing Delta3D has allowed them to integrate several useful open-source projects, as well as concepts from commercial games into an extensible framework useful for many kinds of training simulations.

3) *Specialized Frameworks*

Because virtual worlds do not fit the needs of every educational game project, completing an entire game using a framework graphics engine may be excessive in some circumstances. For some applications, a specialized game engine may need to be used. This section summarizes four specialized game engines and how they have been used for educational research.

The Spring Engine is an open-source game engine specifically designed to support building 3D Real Time Strategy (RTS) games in which the player controls multiple "units" rather than a single avatar [57]. RTSes are generally used for a variety of war or combat simulations. Kernel Panic is one such game built using the Spring Engine, and is likewise open-source. Muratet et al. chose Spring and specifically, Kernel Panic over another open-source project (Open Real-Time Strategy) to implement their serious game because Spring has a larger community of both developers and players [26, 58]. They utilized these to create a serious game to teach programming topics, by allowing students to program AI into the existing game. By modifying an existing open-source game called Kernel Panic, they were able to concentrate their efforts on mapping the educational context to the existing features

Another example of a specialized open-source game engine is The Open Racing Car Simulator (TORCS) [59]. As the name implies, TORCS provides the graphics and physics components necessary to implement a car racing game. Coller and Shernoff used TORCS in a game for mechanical engineering students to teach computational numerical methods [60]. As they mention, "engineers like to tinker" [60]. They were able to use TORCS to create a game environment that directly supports this motivation, while being closely linked to the subject matter (mechanical engineering). As a result, they found that students were more engaged and intellectually stimulated than in previous versions of the same class using traditional methods. Additionally, while they used TORCS as a base for their project, they also implemented several new features specifically related to more realistic physics and simulation of different car components.

F. *Platforms to Consider*

While not explicitly used in much (if any) of the current educational research, there are additional platforms and technologies to consider. These consist of other open-source graphics engines, commercially available game engines, as well as commercial games that provide extension mechanisms that can be used to develop entirely new games.

1) *Engines and Component Frameworks*

Anderson and Peters argue that many of the components necessary to build games already exist in many forms [61]. They provide a comprehensive list of several technologies, organized by component types (graphics, audio, multi-platform GUI systems, physics, 3D model-loading, and many others), many of which are open-source projects. Covering their entire paper is beyond the scope of this article, but it is important to note that not only do they provide a list of software components that can be used to create a game engine, but they also include some nearly complete open-source 3D game engines.

These engines generally include all the facilities for the creation of a full game, incorporating a graphics rendering architecture, audio and physics libraries, as well as components to handle input and networking. One of these engines, Ogre3D has been used in a few case studies in educational gaming [51, 62].

One of the major drawbacks of these free open-source engines is that they are not always able to facilitate easily creating a commercial-level quality game. The design team may also wish to consider a commercial game engine, many of which offer free versions, or significant educational discounts. For instance, the game creation framework called Unity 3D is available as a free product for many platforms (including mobile and Web browser in addition to Windows, Mac OSX, and Linux).

Not only does Unity 3D offer the same kinds of components as an engine like Ogre3D, but it also integrates editing and creation of the game directly into the engine. That is, it provides the developer with many additional game editing tools that can significantly reduce the time and resources required to create an entire game. These include features such as advanced asset management pipelines to terrain editing and audio manipulation. Additionally, Unity3D offers a commercial service for support and consultation if the developers encounter difficulties that they are not able to overcome. Their services range from software design to teaching and training as well as design of the visual elements.

However, it is important to note that a functional and enjoyable educational game does not always need a sophisticated 3D engine. Games using 2D graphics may be completely adequate for the task, and may significantly decrease the time required for development.

PyGame is a multi-platform library for creating 2D games with the Python programming language [63]. It combines a simple programmer interface for manipulating 2D graphics with the ease and power of Python. However, using an interpreted programing language and the goal of keeping the architecture simple have their drawbacks such as limited additional functionality and possible performance issues. The first issue is easy to overcome as Python has many other libraries available for a variety of functions.

However, for more advanced applications, the C++ library called ClanLib is a viable option. ClanLib provides all the functionality of PyGame with additional features such as database support (SQLite), built-in collision detection, a CSS-based GUI framework, XML support, resource management with support for sprite-based animations, access to hardware shaders, and more. However, these features come with the additional complexity of C++ and could add significant development time.

2) Existing Games as Platforms

While the existing games and their engines may not always be completely free, they can be relatively cheap as a development platform. One such game called Half-Life 2 provides a software development kit (SDK) to facilitate development using the game engine. The Half-Life 2 SDK has been a successful tool to teach game programing concepts to students at the high-school level, which reflects its ease of use for specific scenarios [64, 65].

Additionally, Half-Life 2 (and its predecessor, Half-Life) have been used extensively to create scenarios in game effects research, algorithm analysis, server networking, and several other areas [66-72]. The wide-range of research topics covered in a multitude of papers further indicates that the Half-Life 2 SDK is a viable platform to implement games.

However, designers should use caution when selecting to use a game or SDK for their particular application. The Half-Life 2 game engine and its associated SDK were designed primarily for first-person shooter (FPS) games in mind. This means that it may be more difficult to implement games in other genres or formats using this tool. It may not be impossible but require more time and resources.

G. Assets – Where and How

Developing custom graphics and audio for a game can take a significant amount of time and dedication. Professional development of these assets can also be quite expensive in many cases. However, a few resources are useful for reducing the time and development cost. The previously mentioned Unity 3D engine provides some assets with the platform and through supporting webpages, but it also integrates an asset store where users sell or give away assets they have created for their own games (similar in some ways to the Second Life marketplace, but using real-world currency).

Additionally, websites such as OpenGameArt.org provide graphics and audio content free of charge, specifically for open-source video games. The assets hosted on OpenGameArt are available under a variety of licenses, similar to (or in some cases using) the licenses mentioned in the previous section. Much of the content is high quality, and with the correct software should be usable in many different game engines. Using websites such as this, or the Unity asset store could save a considerable amount of development time as well as lower the costs of production significantly.

IV. Additional Resources

As game design itself is a specialized field, there are also some comprehensive resources available for further consideration. The following books and websites cover a wide range of topics, generally in great depth and should be helpful to different individuals depending on their involvement in a game development project.

Firstly, searching for a game engine to use can be a difficult process. The website *DevMaster* (http://www.devmaster.net) makes this process much easier by providing a comprehensive database of 3D game engines. Game engines can be searched and sorted by several categories including closed or open source, platform support (Windows, OS-X, Linux, etc.), programing languages used, physics availability, networking availability, scripting support, and many more. Additionally, users, giving detailed accounts of their experiences, can review each engine.

Both game development and game play include many cultural, social, and psychological factors. Raessens and Goldstein cover many relevant topics in The Handbook of Computer Game Studies [73]. This book is a compilation of sections from many subject experts in fields ranging from artificial intelligence to narrative development. Rather than simply being a reference book on the subject, this book is intended as an overview textbook for students interested in learning more about video games as a new medium. While the 2004 publication date may seem a bit dated in the rapidly evolving scope of video game research, this book is still quite relevant and a very good introduction to the field of research and understanding of video games.

As the name suggests, Designing Virtual Worlds by Richard Bartle is a guide for the design and implementation of virtual environments for game play [20]. The book provides a history of virtual worlds from MUDs (Multi-User Dungeons) to contemporary MMORPGs (Massively Multi-player Online Role Playing Games). In this book, Bartle provides an overview of development team construction and the development process, and eventually moves into discussions about understanding players, game world and narrative design, and game balance. While many of the topics covered address issues from the standpoint of a multi-player game, most are generalizable to single player games as well (for instance, understanding player motivations). While Raessens and Goldstein's book is more of an introduction to video games for researchers, Bartle's book fills this niche for beginning developers.

Bridging these two books, and providing theoretical formalization to much of Bartle's discussions is Salen and Zimmerman's Rules of Play – Game Design Fundamentals [74]. This book provides a much more in-depth and comprehensive look at game development, game play, and game research than the other two. Their approach presents a series of non-exclusive conceptual frameworks for understanding not just video games, but also games and play in a broader sense.

Whether they are using an existing game engine or implementing a custom one, any researchers or developers considering a long-term pursuit of game development may need to understand the theoretical implementations of different game components, architectures, or engines. In his book Game Engine Architecture, Jason Gregory provides a deep technical review of architecture design, development tools, mathematical concepts used in games, and game play [75]. While the book contains many examples given in C++, each concept is described in a way that can be implemented in many programming languages. Nevertheless, this book contains many advanced examples and may not be helpful to a programming or software development beginner.

A good resource for any researchers interested in a deeper understanding of the development methodologies used for open-source projects is the essay titled *The Cathedral and the Bazaar* by Eric Raymond [76]. This essay outlines two processes used by open source projects for releasing their source code – either keeping their source code open to the public during all development (the Bazaar model) or only releasing their source code at specific times and only allowing certain developers to contribute to major features (the Cathedral model). Additionally, the author outlines 19 guidelines for creating good open source software projects.

V. CONCLUSION

Game Based Learning integrates two very complex topics – education and game implementation. By including open source software, a team can significantly reduce the burden of creating an educational game, without considerably reducing the quality of the experience. The resources provided in this paper have illustrated several factors researchers should be aware of, provided example resources to consider, and reflected on the experiences of other researchers implementing open source software for video games in research. Through the consideration of the game as three interconnected parts (the design team, pedagogical components, and the video game itself), we have outlined a structure to aid researchers in pursuit of developing their own educational games while avoiding and being aware of potential pitfalls and difficulties.

REFERENCES

[1] R. Teed. *Game-Based Learning*. Available: http://serc.carleton.edu/introgeo/games/index.html

[2] S. Kennedy-Clark and K. Thompson, "What Do Students Learn When Collaboratively Using A Computer Game in the Study of Historical Disease Epidemics, and Why?," *Games and Culture,* vol. 6, pp. 513-537, November 1, 2011 2011.

[3] H. Lu, "Rethinking game-based learning from a gender perspective: a case study of a male English language learner in China," *Gender, technology and development,* vol. 16, pp. 49-70, 2012. http://dx.doi.org/10.1177/097185241101600103

[4] F. Wang, *et al.*, "Computer Game-Based Learning: Perceptions and Experiences of Senior Chinese Adults," *Journal of Educational Technology Systems,* vol. 40, pp. 45-58, 2012. http://dx.doi.org/10.2190/ET.40.1.e

[5] M. Kordaki, "A computer card game for the learning of basic aspects of the binary system in primary education: Design and pilot evaluation," *Education and Information Technologies,* vol. 16, pp. 395-421, 2011. http://dx.doi.org/10.1007/s10639-010-9136-6

[6] K. Silseth, "The multivoicedness of game play: Exploring the unfolding of a student's learning trajectory in a gaming context at school," *International Journal of Computer-Supported Collaborative Learning,* vol. 7, pp. 63-84, 2012. http://dx.doi.org/10.1007/s11412-011-9132-x

[7] D. M. Adams, *et al.*, "Narrative games for learning: Testing the discovery and narrative hypotheses," *Journal of Educational Psychology,* vol. 104, pp. 235-249, 2012. http://dx.doi.org/10.1037/a0025595

[8] J. Ryoo, Techatassanasoontorn, A.A., Lee, D., and Lothian, J., "Game-Based InfoSec Education Using OpenSim," in *Colloquium for Information Systems Security Education*, Fairborn, Ohio, 2011.

[9] J. Elliott, *et al.*, "No magic bullet: 3D video games in education," 2002, pp. 23-26.

[10] K. Squire, "From content to context: Videogames as designed experience," *Educational researcher,* vol. 35, pp. 19-29, 2006. http://dx.doi.org/10.3102/0013189X035008019

[11] M. J. Mayo, "Games for science and engineering education," *Communications of the ACM,* vol. 50, pp. 30-35, 2007. http://dx.doi.org/10.1145/1272516.1272536

[12] M. Virvou, *et al.*, "Combining software games with education: Evaluation of its educational effectiveness," *Educational Technology & Society,* vol. 8, pp. 54-65, 2005.

[13] B. Kapralos, *et al.*, "A Serious Game for Training Health Care Providers in Interprofessional Care of Critically-Ill and Chronic Care Patients," *Journal of Emerging Technologies in Web Intelligence,* vol. 3, pp. 273-281, 2011. http://dx.doi.org/10.4304/jetwi.3.4.273-281

[14] K. Squire, "Changing the game: What happens when video games enter the classroom," *Innovate: Journal of online education,* vol. 1, 2005.

[15] S. de Freitas. (2007). *Learning in Immersive Worlds - A review of game-based learning.*

[16] K. A. Wilson, *et al.*, "Relationships between game attributes and learning outcomes," *SIMULATION & GAMING,* vol. 40, p. 217, 2009. http://dx.doi.org/10.1177/1046878108321866

[17] D. Oblinger, "Games and Learning," *EDUCAUSE Quarterly,* vol. 29, pp. 5-7, 2006.

[18] R. van Eck, "Digital game-based learning: It's not just the digital natives who are restless," *EDUCAUSE review,* vol. 41, p. 16, 2006.

[19] T. W. Malone and M. R. Lepper, "Making Learning Fun: A Taxonomy of Intrinsic Motivations for Learning," in *Aptitude, learning, and instruction*, R. E. Snow, Farr, M.J., Ed., ed, 1987, pp. 223-253.

[20] R. Bartle, *Designing virtual worlds*, 1st ed. Berkeley, CA: New Riders, 2003.

[21] S. de Freitas and M. Oliver, "How can exploratory learning with games and simulations within the curriculum be most effectively evaluated?," *Computers & Education,* vol. 46, pp. 249-264, 2006. http://dx.doi.org/10.1016/j.compedu.2005.11.007

[22] L. A. Annetta, "The "I's" have it: A framework for serious educational game design," *Review of General Psychology,* vol. 14, p. 105, 2010. http://dx.doi.org/10.1037/a0018985

[23] R. Rosas, *et al.*, "Beyond Nintendo: design and assessment of educational video games for first and second grade students," *Computers & Education,* vol. 40, pp. 71-94, 2003. http://dx.doi.org/10.1016/S0360-1315(02)00099-4

[24] A. Lenhart, *et al.*, "Teens, Video Games, and Civics," Pew Internet & American Life Project2008.

[25] J. Robertson and J. Good, "Story creation in virtual game worlds," *Communications of the ACM,* vol. 48, pp. 61-65, 2005. http://dx.doi.org/10.1145/1039539.1039571

[26] M. Muratet, *et al.*, "Experimental Feedback on Prog&Play: A Serious Game for Programming Practice," presented at the EUROGRAPHICS 2010, Norrköping, Sweden, 2010.

[27] J. W. Rice, "New media resistance: Barriers to implementation of computer video games in the classroom," *Journal of Educational Multimedia and Hypermedia,* vol. 16, pp. 249-261, 2007.

[28] E. L. Deci, *et al.*, "Motivation and education: The self-determination perspective," *Educational psychologist,* vol. 26, pp. 325-346, 1991.

[29] N. Yee, "Motivations for play in online games," *CyberPsychology & Behavior,* vol. 9, pp. 772-775, 2006. http://dx.doi.org/10.1089/cpb.2006.9.772

[30] N. Yee, "The demographics, motivations, and derived experiences of users of massively multi-user online graphical environments," *Presence: Teleoperators and virtual environments,* vol. 15, pp. 309-329, 2006. http://dx.doi.org/10.1162/pres.15.3.309

[31] R. Bartle, "Hearts, clubs, diamonds, spades: Players who suit MUDs," *Journal of MUD research,* vol. 1, p. 19, 1996.

[32] R. L. M. van Meurs, "How to play the game?," Masters, Department of Social Sciences, Tilburg University, Tilburg, 2007.

[33] A. Tychsen, *et al.*, "Motivations for play in computer role-playing games," 2008, pp. 57-64.

[34] J. L. Sherry and K. Lachlan, "Video game uses and gratifications as predictors of use and game preference," *Playing video games. Motives, responses, and consequences,* pp. 213-224, 2006.

[35] M. Csikszentmihalyi, *Flow: The psychology of optimal experience: Steps toward enhancing the quality of life*: Harper Collins Publishers, 1991.

[36] J. Chen, "Flow in games (and everything else)," *Communications of the ACM,* vol. 50, pp. 31-34, 2007. http://dx.doi.org/10.1145/1232743.1232769

[37] P. Sweetser and P. Wyeth, "GameFlow: a model for evaluating player enjoyment in games," *Computers in Entertainment (CIE),* vol. 3, pp. 3-3, 2005. http://dx.doi.org/10.1145/1077246.1077253

[38] R. Aylett, *et al.*, "Unscripted narrative for affectively driven characters," *Computer Graphics and Applications, IEEE,* vol. 26, pp. 42-52, 2006. http://dx.doi.org/10.1109/MCG.2006.71

[39] (2011). *Frequently Asked Questions about the GNU Licenses*. Available: http://www.gnu.org/licenses/gpl-faq.html

[40] (2011, 2011-06-10). *Open Wonderland.* Available: http://openwonderland.org/

[41] (2011, 2011-06-10). *OpenSimulator*. Available: http://opensimulator.org/wiki/Main_Page
[42] M. Gardner, *et al.*, "Reflections on the use of Project Wonderland as a mixed-reality environment for teaching and learning," in *ReLIVE 08 - Researching Learning in Virtual Environments International Conference*, Milton Keynes, UK, 2008, pp. 130-141.
[43] D. Parsons and R. Stockdale, "Cloud as Context: Virtual World Learning with Open Wonderland," presented at the 9th World Conference on Mobile and Contextual Learning (mLearn 2010), Valetta, Malta, 2010.
[44] M. Goslin. (2004, 2011-06-09). *Postmortem: Disney Online's Toontown*. Available: http://www.gamasutra.com/view/feature/2027/postmortem_disney_onlines_.php
[45] M. R. Mine, *et al.*, "Building a massively multiplayer game for the million: Disney's Toontown Online," *Computers in Entertainment (CIE),* vol. 1, p. 6, 2003. http://dx.doi.org/10.1145/950566.950589
[46] (2011, 2011-06-10). *Panda3D - Free 3D Game Engine*. Available: http://www.panda3d.org/
[47] M. Goslin and M. R. Mine, "The Panda3D graphics engine," *Computer,* vol. 37, pp. 112-114, 2004. http://dx.doi.org/10.1109/MC.2004.180
[48] V. Henrich and T. Reuter, "CarDriver–Using Python and Panda3D to construct a Virtual Environment for Teaching Driving," Reykjavík University RUTR-CS08003, May 2008.
[49] (2011, 2011-06-10). *OGRE - Open Source 3D Graphics Engine*. Available: http://www.ogre3d.org
[50] M. Vala, *et al.*, "ION Framework – A Simulation Environment for Worlds with Virtual Agents," in *Intelligent Virtual Agents*. vol. 5773, Z. Ruttkay, *et al.*, Eds., ed: Springer Berlin / Heidelberg, 2009, pp. 418-424.
[51] R. Aylett, *et al.*, "Fearnot!–an emergent narrative approach to virtual dramas for anti-bullying education," *Virtual Storytelling. Using Virtual Reality Technologies for Storytelling,* pp. 202-205, 2007.
[52] R. S. Aylett, *et al.*, "FearNot!-an experiment in emergent narrative," in *Fifth International Conference on Intelligent Virtual Agents*, 2005, pp. 305-316.
[53] (2011, 2011-06-10). *Delta3D - Open source gaming & simulation engine*. Available: http://www.delta3d.org
[54] R. Darken, *et al.*, "The Delta3D open source game engine," *IEEE computer graphics and applications,* pp. 10-12, 2005. http://dx.doi.org/10.1109/MCG.2005.67
[55] P. McDowell, *et al.*, "Delta3D: a complete open source game and simulation engine for building military training systems," *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology,* vol. 3, p. 143, 2006. http://dx.doi.org/10.1177/154851290600300302
[56] J. Orkin, "Three states and a plan: the AI of FEAR," in *Proceedings of the Game Developer's Conference* 2006.
[57] (2011, 2011-06-10). *The Spring Project*. Available: http://springrts.com/
[58] M. Muratet, *et al.*, "Towards a serious game to help students learn computer programming," *International Journal of Computer Games Technology,* vol. 2009, pp. 1-12, 2009. http://dx.doi.org/10.1155/2009/470590
[59] (2011, 2011-06-10). *TORCS*. Available: http://torcs.sourceforge.net/
[60] B. D. Coller and D. J. Shernoff, "Video game-based education in mechanical engineering: A look at student engagement," *International Journal of Engineering Education,* vol. 25, pp. 308-317, 2009.
[61] E. F. Anderson and C. E. Peters, "No More Reinventing the Virtual Wheel: Middleware for Use in Computer Games and Interactive Computer Graphics Education," presented at the 31st Annual Conference of the European Association for Computer Graphics - Eurographics 2010 Education Papers, Norrköping, Sweden.
[62] O. Shabalina, *et al.*, "GAME-BASED APROACH IN IT EDUCATION," in *Human Aspects of Artificial Intelligence*, ed Sofia, Bulgaria: Institute of Information Theories and Applications, 2009, pp. 63-69.
[63] (2011). *Pygame - Python Game Development*. Available: http://pygame.org
[64] A. Eliens and S. Bhikharie, "Game@ VU–developing a masterclass for high-school students using the Half-life 2 SDK," *Proc. GAME'ONNA,* pp. 19-21, 2006.
[65] M. S. El-Nasr and B. K. Smith, "Learning through game modding," *Computers in Entertainment (CIE),* vol. 4, pp. 1-20, 2006. http://dx.doi.org/10.1145/1111293.1111301
[66] T. Wright, *et al.*, "Creative player actions in FPS online video games: Playing Counter-Strike," *Game studies,* vol. 2, pp. 103-123, 2002.
[67] N. Cole, *et al.*, "Using a genetic algorithm to tune first-person shooter bots," in *Proceedings of the 2004 Congress on Evolutionary Computation IEEE*, 2004, pp. 139-145.
[68] Y. W. Bernier, "Latency compensating methods in client/server in-game protocol design and optimization," in *Game Developers Conference*, 2001.
[69] J. T. Bell and H. S. Fogler, "Implementing virtual reality laboratory accidents using the Half-Life game engine, WorldUp, and Java3D," in *Proceedings of the 2003 American Society for Engineering Education Annual Conference & Exposition*, 2003.
[70] L. Nacke, *et al.*, "Log who's playing: psychophysiological game analysis made easy through event logging," *Fun and Games,* pp. 150-157, 2008.
[71] A. K. Przybylski, *et al.*, "The motivating role of violence in video games," *Personality and Social Psychology Bulletin,* vol. 35, p. 243, 2009. http://dx.doi.org/10.1177/0146167208327216
[72] S. McQuiggan, *et al.*, "Story-based learning: The impact of narrative on learning experiences and outcomes," in *Intelligent Tutoring Systems 2008*, Montreal, Canada, 2008, pp. 530-539.
[73] J. Raessens and J. H. Goldstein, *Handbook of Computer Game Studies*. Cambridge: MIT Press, 2005.
[74] K. Salen and E. Zimmerman, *Rules of Play: Game Design Fundamentals*: The MIT Press, 2004.
[75] J. Gregory, *Game Engine Architecture*. Natick, Massachusetts: A K Peters Ltd, 2009.
[76] E. Raymond, "The cathedral and the bazaar," *Knowledge, Technology & Policy,* vol. 12, pp. 23-49, 1999. http://dx.doi.org/10.1007/s12130-999-1026-0

AUTHORS

J.M. Lothian is with The Pennsylvania State University, University Park, PA 16802 (e-mail: jlothian@psu.edu).

J.Ryoo is with The Pennsylvania State University, Altoona, PA 16601 (e-mail: jryoo@psu.edu)

This article is an extended and modified version of a paper presented at the 2011 Interdisciplinary Conference of AHLiST, held May 20-22, 2011, at College of Technology, University of Houston, TX, USA. Submitted 16 June 2013. Published as re-submitted by the authors 08 November 2013.