The Serious Gaming Surgical Cognitive Education and Training Framework and SKY Script Scripting Language

Brent Cowan, Bill Kapralos
Business and Information Technology, University of Ontario Institute of Technology.
Oshawa, Ontario, Canada.
{brent.cowan, bill.kapralos}@uoit.ca

Fuad Moussa
Division of Cardiac and Vascular Surgery,
Sunnybrook Health Sciences Centre.
Toronto, Ontario, Canada.
Fuad.Moussa@sunnybrook.ca

Adam Dubrowski
Faculty of Medicine,
Memorial University.
St. John's, Newfoundland, Canada.
adam.dubrowski@gmail.com

ABSTRACT
Developing effective serious games is a difficult and time consuming process often requiring technical expertise which is lacking by many educators employing such tools within their curriculum. We have recently begun development of a serious game surgical cognitive education and training framework (SCETF) for use in surgical skills training and education. Novel to the SCETF is the Sky Script scripting language that balances the functionality, and flexibility, allowing educators to design and build their own interactive content without any prior programming knowledge. Here we provide a brief overview of the SCETF and introduce Sky Script, both of which are currently under development.

Categories and Subject Descriptors
I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism – *Virtual reality.* H.5.1 [**Information Interfaces and Presentation**]: Multimedia Information Systems – *Artificial, augmented, and virtual realities.* D.3.3 [**Programming Languages**]: Formal Definitions and Theory – *semantics, syntax.*

General Terms
Algorithms, Design.

Keywords
Virtual simulation, serious gaming, cognitive surgical

1. INTRODUCTION
Serious games have been applied within a wide range of applications from surgical skills acquisition to military strategy training, to interpersonal skills development. Given the ubiquity of video game use across a large demographic (i.e., male, female, from the very young to the very old), and their ability to engage and motivate learners, the popularity of serious games has seen a recent surge across all areas of education and training. Despite the growing popularity of serious games and the benefits that they afford, the development of effective serious games is difficult and time consuming, requiring an appropriate balance between game design and instructional design (it has been suggested that a lack of proper instructional design will lead to ineffective serious games or "instructional games") [1,2]. Further complicating matters, there are currently no standard development tools (game engines and frameworks in particular), available that emphasize (and perhaps to some degree force), designers and developers of serious games to follow any set of best practices. Sherrod defines a game engine as "a framework comprised of a collection of different tools, utilities, and interfaces that hide the low-level details of the various tasks that make up a video game" [3]. The terms game engine and framework are often used interchangeably. For the purpose of this paper, the term game engine refers to the functionality and features that become part of the completed game. A framework includes a game engine in addition to external tools and resources that simplify the process of game development. Many serious game developers are using game engines and frameworks that were developed for the creation of commercial entertainment games despite the inherent differences between serious game and entertainment game development [4] (see [5] for a summary of the most commonly identified frameworks and game engines along with a description of their features).

Serious games are often developed by game developers with limited knowledge regarding the game's educational content and instructional design. Educators could take on a greater role in the design and development of educational software and serious games if tools were made available to them that greatly simplified the game development process. We believe that the availability of such tools coupled with an educator's educational background (instructional design in particular), will lead to more effective serious games. Although it is unrealistic to expect educators to be highly skilled in game development and programming, one should expect that some educators would be willing to take the time to learn some basic skills in order to provide a better learning environment for their students. If a game creation framework could be made "simple enough", a layperson could become competent enough to develop their own games, or at least modify existing games developed by others using the framework.

Empowering content experts and educators to build their own interactive software and serious games may allow them to better meet the needs of their students. It could be argued that educators who do not have a background in game development may create games that do not offer an enjoyable experience for the players (students). However, it could also be argued that game developers who do not understand the educational content of the game or the needs of the students, may create games that do not meet the educational needs of the students.

To empower educators and provide them the opportunity to have a greater role within the serious game development process, we have assembled an interdisciplinary group of researchers, educators, computer scientists/engineers, and game developers,

SIMUTOOLS 2015, August 24-26, Athens, Greece

DOI 10.4108/eai.24-8-2015.2261370

and have begun development of a novel game development framework that will greatly simplify the three primary areas of game development (art and animation, sound and music, and programming). The framework allows for the creation of serious games by those who are not necessarily game developers or expert programmers (e.g., many educators). Although the framework described here may be of value to educators in general, we are currently targeting medical educators specifically. Serious gaming is growing in popularity within the medical education and training realm for a number of reasons including their ability to motivate and engage the trainees, and their cost effectiveness. However, medical procedures and practices are constantly changing in response to new technologies, new treatments, and a shifting patient demographic. By providing medical educators with the tools necessary to easily develop and/or easily modify their own interactive content (i.e., serious games), we are aiming for a proliferation of serious games for medical education and training.

2. The Serious Game Framework

The multi-modal framework that we are developing is known as the serious game surgical cognitive education and training framework (SCETF) (see [6] for greater details). The SCETF focus is on the cognitive components of a surgical procedure and more specifically, the proper identification of the sequence of steps comprising a procedure, the instruments and anatomical/physiological knowledge required for performing each step, and the ability to respond to unexpected events while carrying out the procedure. By clearly understanding the steps of a procedure and the surgical knowledge that goes along with each step, trainees are able to focus solely on the technical aspect of the procedure. The SCETF consists of graphical and spatial sound rendering engines, a graphical user interface (GUI), and a library of game assets (sounds, 2D and 3D models, and textures, amongst others). The GUI provides a simple interface for script editing, and level creation. Sound and graphic libraries relieve the user of the responsibility of creating in game assets; editing sounds, graphics, and models requires a considerable amount of skill. Users with these specialized skills will have the option to add their creations to the existing library. Programming at some level is unavoidable if the user is going to have any control over the game's logic. Domain-specific surgical modules are built on top of the existing framework, utilizing common simulation elements/assets and ultimately reducing development costs. A module is specific to particular surgical field (e.g., cardiac, or orthopedic surgery), and within each module specific scenarios can be defined and developed (e.g., a scenario in the cardiac surgery module may focus on the cognitive aspects of the off-pump coronary artery bypass (OPCAB) grafting procedure [7]).

To facilitate scenario development and thus increase the flexibility and utility of the framework, the SCETF includes a scenario editor (currently being developed) that allows users of the module (educators/instructors) to create and/or modify/edit specific scenarios using a graphical-based user interface. The scenario editor allows a scenario to be easily developed by clicking and dragging various interface components in a "what you see is what you get" (WSYWIG) manner. Finally, the SCETF is also being developed as a research tool where various simulation parameters (e.g., levels of audio/visual fidelity) can be easily adjusted allowing for the controlled testing of such factors on knowledge transfer learning and retention and this will ultimately lead to more effective serious games. For example, a preliminary version of the SCETF was used to facilitate an experiment that examined the influence of sound of visual realism and task performance within a virtual operating room [6].

Although a number of tools (game engines in particular) are available to facilitate the development of serious games and video games in general (e.g., the Unity 3D, and Unreal video game engines), they have not been developed specifically for, and marketed to educators who will typically have limited programming experience. They are instead intended for skilled programmers and artists. Creating a modern video game is not a simple process, and often requires dozens of highly skilled people and budgets that can reach tens of millions of dollars [8].

As an alternative to programming, several serious games allow educators to insert their own content. In-game menus may offer some flexibility with respect to the appearance or overall flow of the game. The educators act as content experts with little or no control over the game play mechanics. However, programming can provide the end user with complete control over both content and interaction. Generally, the greater the functionality afforded by the engine, the greater the prior (programming) knowledge required by the user (educator/instructor) to use it. Striking a balance between an educator's prior programming knowledge and experience, and the functionality, and ease of use of the framework, we are developing Sky Script, a novel programming (scripting) language that is being incorporated into the SCETF. It allows educators to design and develop their own interactive content without any prior programming knowledge.

3. The Sky Script Language

Sky Script includes a simple and consistent syntax, yet it is powerful enough not to restrict the type of games that can be created with it. Sky Script script is translated into another language when compiled allowing the script to add a layer of abstraction over existing game engines and languages. Sky Script has been designed for programmers with various skill levels. Expert programmers are needed to write the "glue code" that defines how Sky Script will translate each line to the destination language used by an existing engine such as the Panda3D, or the Unity 3D game engines. Amateur programmers will then be able to use this script to write programs even if they have little or no experience with the underlying engine. Beginners will be able to make modifications to modules created by others. In addition to being simple, it is anticipated that some of the performance issues often associated with scripting languages will be addressed. Sky Script does not incur any additional overhead due to interpretation. The compiler translates the Sky Script script directly to another language leaving no trace of the original script. For example, Sky Script can compile to C++ code so that the output from the compilation process is complete C++ code, which can in turn be compiled using a C++ compiler.

Functions in Sky Script must belong to a class, and functions do not return data (void). In order for a function to alter other objects (outside of the object calling the function) they must be supplied as arguments. Future versions will allow the user to specify whether or not an argument is 'read only', a designation that will be enforced by the compiler. The code-based examples outlined in Fig. 1 – Fig. 3 demonstrates how to create a simple class called 'Number' with two user defined member functions; 'Multiply' and 'Print' (see Fig. 1). This is an example of 'glue code' which defines how the class will compile to another language, in this case C++. Each class comes with a constructor named 'Create', a destructor named 'Destroy' and a function named 'Set'. By

default, every class contains a member variable called ‘data’ which is variant, and string variables called ‘name’ and ‘type’ which are set when the object is created. Any number of additional member functions and variables may be added to a class.

```
Sky.AddType(Number)
Number.Create.AddDefinition(
        float name = 0.0f;
)
Number.Set.AddDefinition(
        name = value;
 )

Number.AddFunction(Multiply)
Number.Multiply.AddArgument(Number other)
Number.Multiply.AddDefinition(
        name *= other;
)

Number.AddFunction(Print)
Number.Print.AddDefinition(
        cout << name;
)
```

Fig. 1. A simple Sky Script script class that compiles or translates to a C++ type.

Defining classes that compile to another language requires a thorough understanding of the destination language. However, once the ‘glue code’ has been written, amateur programmers with limited or no experience using the destination language (C++ in this example) will be able to write programs using the defined classes. In addition, they will be able to define their own classes containing other classes as member variables, without ever having to look at the underlying C++ code. The sample code illustrated in Fig. 2 demonstrates how to use the class created in Fig. 1 within a program. The example in Fig. 2 will compile or translate to the C++ code shown in Fig. 3. If this code were part of a complete C++ program, it could be compiled to machine language with a C++ compiler and would not incur any loss of efficiency due to the scripting language.

```
Number.Create(price)
price.Set(3.95)

Number.Create(quantity)
quantity.Set(4)

price.Multiply(quantity)
price.Print()
```

Fig. 2. A simple program utilizing the class created in Figure 1.

```
        float price = 0.0f;
        price = 3.95;
        float quantity = 0.0f;
        quantity = 4;
        price *= quantity;
        cout << price;
```

Fig. 3. The C++ code generated by compiling the simple code listed in Figs. 1 and 2.

The Sky Script scripting language provides a simple object orientated syntax and is compatible with many game engines; Sky Script can be configured to compile to, or translate to most other languages. The Sky Writer graphical user interface (GUI) that is currently under development, will provide visually guided coding to prevent syntax errors before they are made by validating each line as it is added. Finally, the compiler will be lightweight, free, and open source.

4. Summary and Future Work

The design and development of serious games is a difficult task that requires knowledge of instructional design in addition to game and software development. Here we have described our ongoing work that is seeing the development of a serious game framework to empower educators/instructional designers and provide them the opportunity to develop their own, custom serious games with a limited technical (software and game development) background. Fundamental to our framework is Sky Script, a novel programming (scripting) language that allows educators to design and build their own interactive content without any prior programming knowledge. Although significant progress has been made, greater work remains. In addition to completing Sky Script, future work will also see completion of the Sky Writer, the GUI that will allow users to develop scenarios in a “point and click” manner. Finally, once completed, a series of user-based effectiveness tests will be conducted to examine the usability of the tool and more specifically, can it be used to develop an effective serious game by someone with a limited programming and game development background?

ACKNOWLEDGMENTS

The financial support of the Social Sciences and Humanities Research Council of Canada (SSHRC), in support of the Interactive & Multi-Modal Experience Research Syndicate (IMMERSe) initiative, and the Natural Sciences and Engineering Research Council of Canada (NSERC) in the form a Post-Graduate scholarship to B. Cowan and a Discovery grant to B. Kapralos is gratefully acknowledged.

REFERENCES

[1] Becker, K., and Parker, J. The Guide to Computer Simulations and Games, John Wiley and Sons, Inc., Indianapolis, IN USA, 2011.

[2] Iuppa, N., and Borst, T. End-to-End Game Development: Creating Independent Serious Games and Simulations from Start to Finish, Focal Press, Oxford, UK, 2010.

[3] A. Sherrod, “Ultimate 3D Game Engine Design & Architecture,” Charles River Media, 2007. Boston, MA.P.

[4] Petridis, I. Dunwell, D. Panzoli, S. Arnab, A. Protopsaltis, M. Hendrix, and S. de Freitas, “Game engines selection framework for high-fidelity serious applications,” *International Journal of Interactive Worlds*, vol. 2012 (2012), Article ID 418638.

[5] Cowan, B., and Kapralos, B. A survey of frameworks and game engines for serious game development. Proc. Int. Conf. on Advanced Learning Technologies, Athens, Greece, July 7-10, 2014, pp. 662-664.

[6] Cowan, B., Rojas, D., Kapralos, B., Moussa, F., and Dubrowski, A. “Effects of sound on visual realism perception and task performance,” *Vis. Comput.*, July, 2014, 1-10.

[7] Cowan, B., Sabri, H., Kapralos, B., Cristancho, S., Moussa, F., and Dubrowski, A. “SCETF: Serious game surgical cognitive education and training framework,” *Proc. 3rd IEEE Int. Games Innovation Conference*, City of Orange, CA, USA, Nov. 2-4, 2011, pp. 130-133.

[8] Overmars, M. H. 2004. Teaching computer science through game design. *IEEE Computer*. 37, 4 (Apr. 2004), 81-88.