Interactive music systems within multimedia game development environments

V.J. Manzo, PhD
Worcester Polytechnic Institute
100 Institute Road
Worcester, MA 01609
+1 (508) 831-5246
vj@wpi.edu

Dan Manzo
Worcester Polytechnic Institute
100 Institute Road
Worcester, MA 01609
+1 (508) 831-5246
dvmanzo@wpi.edu

ABSTRACT

Gaming development programming environments allow for a convergence of multimedia elements within a single environment. This article mentions several game programming environments and focuses on two specifically, noting how they may be used bob non-programmers to create rich, immersive, interactive music systems that support the composition and performance efforts of non-musicians. Research into the efficacy of such systems to support musicianship stems from prior interactive music projects (EAMIR, IMTCP) that involved the development and use of software applications designed to support musical creativity by musicians and non-musicians. The use of gaming systems for such applications may be useful to individuals without formal programming experience who have an interest in utilizing multimedia tools to create interactive music systems. Such systems traditionally allowed end-users to compose and perform through software. Issues of accessibility, pedagogy, design, and creativity within these development environments are discussed.

Keywords

Interactive media, music systems, music theory, ear-training, harmony, adaptive instruments, games, music programming, music technology, music education, informal music learning

Adaptive musical instruments can provide scaffolding by which disabled and special needs populations acquire musicianship skills. The instruments themselves are created with accessibility in mind for a specific purpose, such as to play chords or percussive sounds, with a specific individual or group in mind with which the instrument will help overcome some limitation, perhaps physical or mental, on the part of the performer. Adaptive instruments are acoustic or electronic in design and have been shown by Crowe (2004) to be useful in music education and music therapy contexts. Advances in technology have helped many new adaptive instrument projects to form including Skoog (Schogler, 2010), AUMI (Pask, 2007), My Breath My Music Foundation (Wel, 2011), and EAMIR (Manzo, 2007).

As Hunt and Wanderly (2002) explain, the paradigm of instrument design traditionally has focused primarily on principals of acoustics. Design concepts that inhibit string vibration or airflow in ways that compromise musical variables such as timbral qualities and dynamic range in undesirable ways were and are concerns for makers of acoustic instruments. With electronic instruments, the mapping of musical variables to control can be similar to traditional instruments or completely unrelated. In this way, instrument designers can pursue concepts that allow for novel idiomatic writing and performance without the acoustical concerns of sound reproduction.

Developing musicianship skills using the assistance of a traditional acoustic instrument like guitar or piano requires some level of proficiency on that instrument. During the process of developing proficiency on that instrument, the participant might be expected to focus his or her attention to specific cognitive musicianship skills such as hearing the chord progression and the harmonic flow as they are playing, and not focus on the physical task of performing on the instrument. To this extent, if asked to listen to a recording and identify the chord progression using a traditional instrument as an aid, a musician who is unfamiliar with the traditional instrument, might focus most of his or her attention on ensuring that he or she is playing the chords correctly and miss the purpose of the activity altogether: identifying the chord progression used on the recording. Using a software-based musical instrument could remove some of the need for attention to physical performance issues that one might encounter while performing an unfamiliar acoustic chordal instrument.

Separating the physical act of performing from the cognitive function of hearing harmony is important to educators because it allows *musicing* (Elliott, 1995) to occur by students without making them wait until they have learned the performance skills of a traditional instrument in order to play chords. In this way, playing chords and, conceivably, being able to compose and perform with them can occur much sooner with a software-instrument than with acoustic instruments. By minimizing the number of layers between the student and the task, the musical concept can be isolated to some extent and understood apart from the context of it being performed on a particular instrument.

The Interactive Music Technology Curriculum Project (Manzo & Dammers, 2010), or IMTCP, was a study in which students learned to compose and perform informally using non-traditional software-based instruments. Musical concepts and compositional and performance techniques were explained and demonstrated to non-traditional music students and students who were not involved in their school's music program through the use of software-based musical systems in an informal manner similar to that of Green (2002, 2008). Similar systems may be developed using video game programming environments, since video games commonly map music and sound variables to accessible controls.

In the game *Super Mario Brothers* (1985), the controls for the main character allow him to jump and move forward or backward.

When the game begins, there is no explanation given to describe the gameplay or the mechanics of the controls; players learn the basic gameplay informally through interactions within the environment.

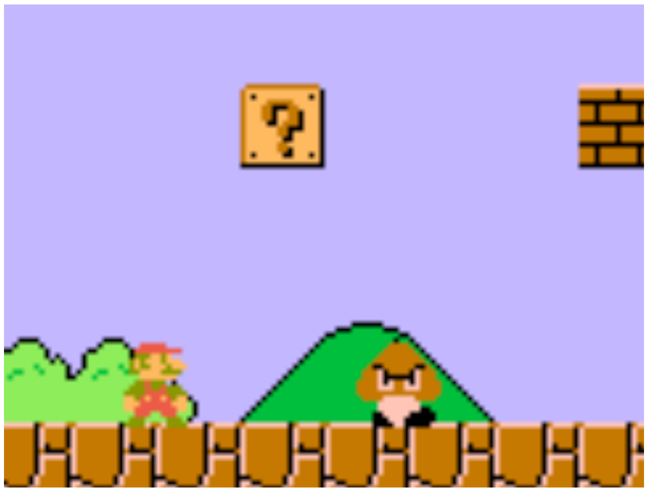


Figure 1. Character cannot avoid enemy without an action.

In Figure 1, we see that an approaching enemy encroaches upon the main character. If the player does not explore the controls presented to him on the game controller, the enemy will touch the character and the level will restart.

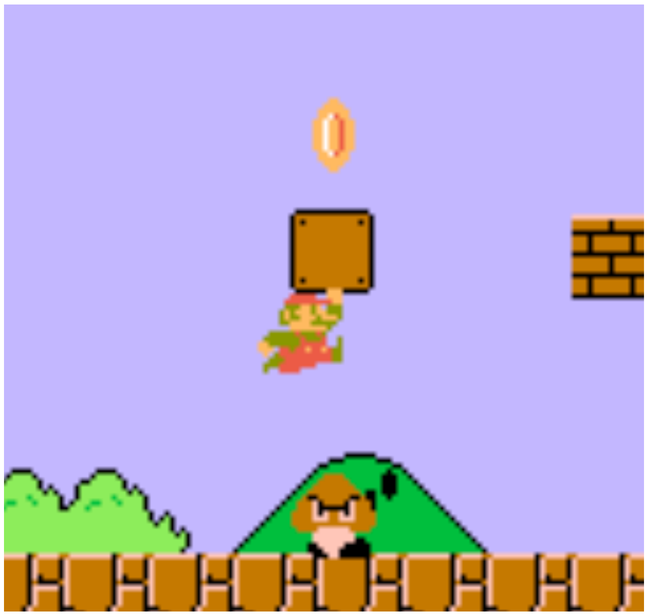


Figure 2. Main character jumps over the enemy.

Exploration of the game controls leads the player to learn that he can cause the main character to jump. Besides running, this is the only action that main character can make in this game. As the enemy approaches, the player causes the main character to jump. The environment within the game is ordered so that jumping over the enemy simultaneously, and inadvertently, hits the question mark box above the enemy. This results in the character being rewarded with one coin. Through this design, the player informally learns that hitting boxes yield positive rewards and jumping over the enemy is necessary. Informal instructional techniques are possible with thoughtful level design.

DEVELOPMENT ENVIRONMENTS

Lack of formal programming skills can be a hindrance to music researchers who seek to develop rich music-oriented tools. However, videogame programming environments already contain tools and frameworks for generating and manipulating audio since these are generally a major component of videogame design. As these environments, and the video game development profession, have grown in popularity through the years, new development architectures have emerged that facilitate game development with minimal programming skills. Game development environments also allow developers to deploy their games to multiple devices and platforms such as desktop computers, browsers (HTML5), and mobile devices.

Construct 2

One such development environment is the PC-based application Construct 2 (2014) developed by Scirra. It is free for non-commercial use.

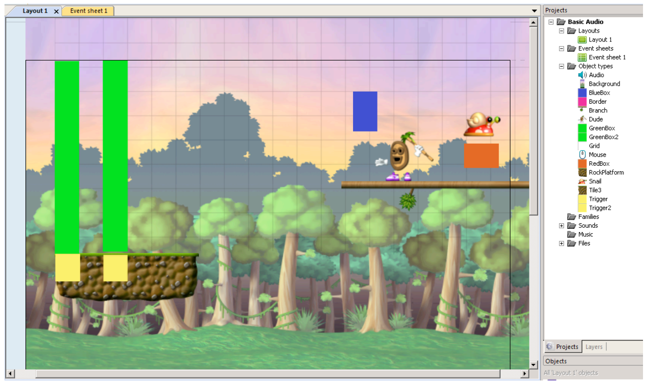
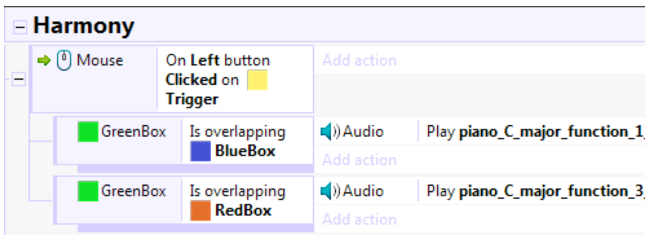


Figure 3. Construct 2 main layout showing assets

As shown in Figure 3, *Construct 2* allows you to drag and drop images, audio, and other assets onto a blank canvas called the *Layout*. For each asset in this game world, a number of characteristics can be defined. Is it a solid? Is it heavy? Is it visible? As the Layout is built, the game may be previewed within a web browser. This simple game is used to allow the end-user to click on game characters to play back pitched sounds.

The *Event Sheet* presents a simplified programming approach. As shown in Figure 4, the Event Sheet allows the developer to use conditional statements to define the rules of the game world. Each asset or control mechanism may be used to complete these statements. *Construct 2* asks a number of questions in order to complete this task.


Harmony

Mouse — On **Left** button **Clicked** on 🟨 **Trigger** — Add action

🟩 GreenBox — Is overlapping 🟦 **BlueBox** — Audio — Play **piano_C_major_function_1** — Add action

🟩 GreenBox — Is overlapping 🟧 **RedBox** — Audio — Play **piano_C_major_function_3** — Add action

Figure 4. Construct 2 event-based programming sheet

As shown in Figure 4, the developer has selected the mouse as the control to be defined. Construct 2 then prompts the developer with actions germane to the mouse such as "when left button is down" , "when left button is up", and so on. The prompts continue allowing the developer to select objects that are changed by the initial action. This type of event-based programming may be useful to non-programmers because there is no scripting syntax to learn. There are only logic statements that the developer must think through.

A demonstration level is available (see *Discussion* section) that shows how audio clips have been assigned to two characters. When the mouse clicks on a certain area of the platform, it plays the sound of each character standing on that area. As shown in Figure 5, two characters are stacked on top of each other. The end-user of this game would click on the platform and hear the interval of a perfect fifth performed; one note derived from each of the characters.

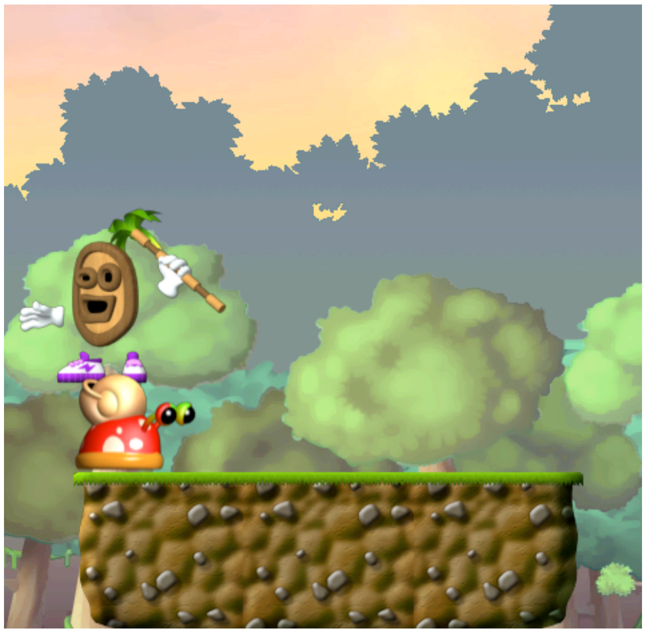


Figure 5. Note-making characters are stacked as pitches

Unity 3D

Unity 3D (2014) is another popular game development environment. It is free for non-commercial use and available for Mac and PC platforms. Unlike Construct 2, Unity 3D is useful for building three-dimensional game environments with support for two-dimensional worlds as well.

Like Construct 2, Unity 3D uses a drag and drop approach to building the gaming world, though experience with Javascript or C# programming, even a superficial understanding, will allow for more sophisticated games to be created. Still, assets like characters, camera views, props, sound sources, and more can simply be dragged into the game world and repositioned. As shown in Figure 6, a Unity use may add geometric shapes and then assign textured graphic patterns to them. The environment pictured was made from a single three-dimension object with an image of a grassy texture applied to it. The sky is also an image. The developer may add a light source to the world, which will be perceived by the end-user as a sun. Unity allows for control over the physics of the world you create. This may be used to create an environment in which sound exploration is possible.

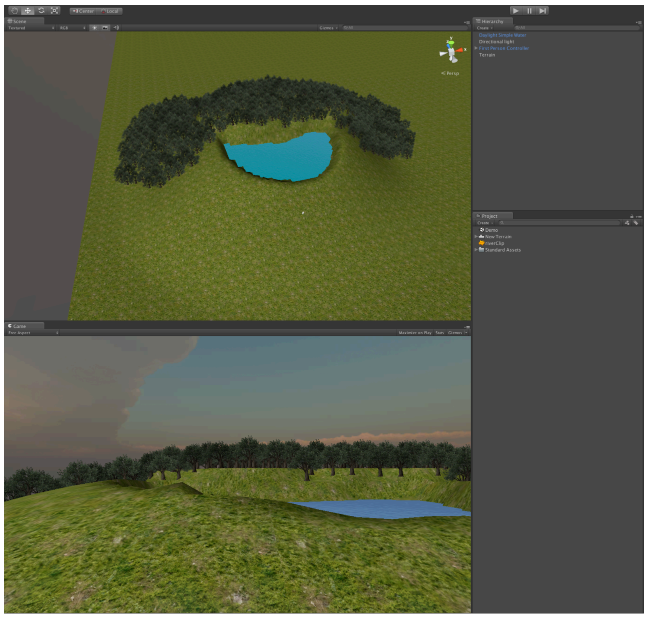


Figure 6. Basic Unity 3D terrain showing shapes with textures

As one develops in Unity, he or she may preview the virtual world from within the development environment. In this game, a simple hill, a pond, and trees have been added using Unity's built-in models. However, models by animators and artists are readily available online, and can be imported into Unity with ease.

A demonstration level is available (see *Discussion* section) as shown in Figure 7. The sound of flowing water has been dragged onto the model of the pond. Unity automatically applies the physics of our world to this gaming environment, so as the game character approaches the pond, the sound of the water increases slowly in volume, simulating the way one would perceive these sounds in real life.

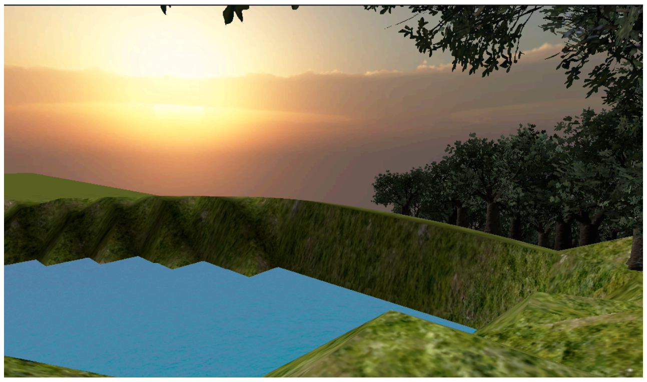


Figure 7. First-person character explores the environment

Such an environment could be used for developing interactive listening environments, where the end-user is expected to detect a timbre sound from among others, or to explore a virtual world using listening skills to identify a specific sound.

DISCUSSION

There are numerous music-oriented video games in existence. Game development programming environments make use of many of the same sound generation technologies that music technology applications use. There are mechanics for playing, organizing, listening, recording, and manipulating sounds, all of which can be delivered in a controlled environment. By developing the environment in which players interact with the mechanics of gameplay, a researcher can structure the order in which events occur and to teach musical concepts informally. Such informal learning can occur through exploration of the game world with accessible controls and limited written or verbal communication.

Accessible game development environments allow the creation of rich musical applications, but programming skills are not a prerequisite. Games similar to those mentioned in this paper can be constructed similarly, modifying their objectives, musical results, and rewards. Concepts of theory and composition can be demonstrated and explained by further developing the Construct 2 example used in this document. Similarly, aspects of critical listening and timbral recognition can be created by further development of the Unity 3D demonstration. The game development environments themselves are intuitive tools that can provide an accessible development interface for educators and researchers, and facilitate informal music learning opportunities for students.

Construct 2 and Unity 3D are available for download from www.scirra.com and www.unity3d.com respectively. The demonstration levels created for this article may be downloaded from www.vjmanzo.com/demos/game_devs/.

BIOGRAPHIES

V.J. Manzo (PhD Temple University, M.M. New York University) is Assistant Professor of Music Technology and Cognition at Worcester Polytechnic Institute (WPI). He is a composer and guitarist with research interests in theory and composition, artificial intelligence, interactive music systems, and music cognition. V.J. is the Oxford University Press author of the book MAX/MSP/Jitter for Music (2011) on developing software-based interactive music systems for composition, performance, instruction, and research.

Dan Manzo (BA New Jersey Institute of Technology, MS candidate at Worcester Polytechnic Institute) is a programmer, pedagogue, and musician with interests in web applications, interactive media & gaming, information technology education, and multimedia performance. He is the founder of Knockout Media and has authored numerous projects in these genres.

REFERENCES

[1] Construct 2. (February 2014). Scirra Construct 2. Retrieved from http://www.scirra.com

[2] Crowe, B. J. (Winter, 2004). Implications of technology in music therapy practice and research for music therapy education: A review of literature. Journal of Music Therapy, 41(4), 282-320.

[3] Elliott, D. (1995). Music matters: A new philosophy of music education. New York: Oxford University Press.

[4] Green, L. (2002). How popular musicians learn. Aldershot, England: Ashgate Publishing Limited.

[5] Green, L. (2008). Music, informal learning and the school: A new classroom pedagogy. Surrey, England: Ashgate Publishing Limited.

[6] Hunt, A. & Wanderly, M. (2002). Mapping performer parameters to synthesis engines. Organised Sound: Cambridge University Press, 7(2) 97-108.

[7] Manzo, V. J., & Dammers, R. (August, 2010). Interactive music technology curriculum project (IMTCP). Retrieved from http://www.imtcp.org

[8] Manzo, V. (Winter, 2007). EAMIR [The electro-acoustic musically interactive room]. Retrieved from http://www.eamir.org

[9] Pask, A. (Interviewer) & Oliveros, P. (Interviewee). (July, 2011). The adaptive use instruments project. Retrieved from http://cycling74.com/2007/12/07/the-adaptive-use-instruments-project/

[10] Schogler, B. (July, 2010). Skoog music. Retrieved from http://www.skoogmusic.com

[11] Super Mario Brothers [video game]. (1985). Tokyo, Japan: Nintendo EAD

[12] Unity 3D. (February 2014). Unity. Retrieved from http://www.unity3d.com

[13] Wel, R. V. D. (July 2011) . My Breathe My Music Foundation. Retrieved from http://www.mybreathmymusic.com