Game programming environments for musical interactions

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
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.

There are many music-oriented video games in existence. This article explores the potential and rationale for using robust programming environments like Construct 2 and Unity 3D for individuals who lack formal programming skills to develop video games that facilitate their musical objectives; performance, composition, education, research or other.

Keywords
Interactive media, music systems, gaming environments, music programming, music technology, music education, informal music learning

BACKGROUND

An interactive music system is a hardware or software configuration that allows an individual to accomplish a musical task, typically in real-time, through some interaction. Though commonly associated with composition and performance, the tasks associated with interactive music systems can include analysis, instruction, assessment, rehearsal, research, therapy, synthesis, and more. These systems typically have some set of controls, hardware or software, such as switches, keys, buttons, and sensors by which musical elements like harmony, rhythm, dynamics, and timbre can be manipulated in real-time through user interaction."

Electronic musical instruments can then be thought of in terms of having controls over some musical elements. A variable is something that changes. A control is something that changes a variable. In music, there are many variables, such as pitch, dynamics, and timbre that change as a result of the instrument's control device, also known as a control interface.

The control interface for a violin is typically a bow. Without buttons, knobs, or sensors, the bow is capable of controlling numerous variables within a single, simple, interface. For example, if you angle the bow differently as it hits the strings, the timbre will change; apply more pressure and the dynamics will change.

The Buchla 200e, shown in Figure 1, for example, is a modular synthesizer also capable of controlling numerous musical variables. In fact, the Buchla is capable of creating more diverse timbres than the violin. However, controlling musical variables on the Buchla, with the control interface of knobs, buttons, and patch cables, involves more gestures than the violinist and the bow.

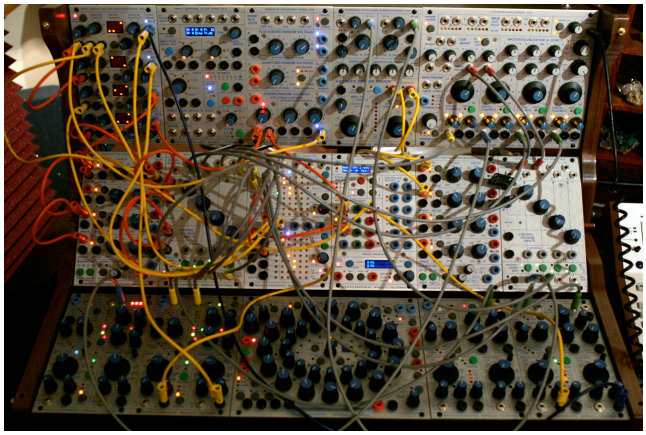


Figure 1. The Buchla 200e modular synthesizer

For the intent of performance, some control interfaces are more accessible than others for real-time use. With a computer, you can arguably achieve any sound imaginable if you tweak the right numbers and press the right buttons. It is a well-designed control interface, however, that allows a performer to readily control musical variables in a less cumbersome way than clicking on menu items from pull-down lists and checking boxes.

Throughout history, people have created new musical instruments, and the instruments created generally reflect the technological resources available at the time. Early primitive instruments had few moving parts, if any. The Industrial Revolution made way for the modern piano to evolve using steel and iron. In the Information Age, it stands to reason that newly created instruments may largely involve computers and electronics.

New Interfaces for Musical Expression or NIME, is an international conference in which researchers and musicians share

their knowledge of new instruments and interface design. Session topics include controllers for performers of any skill level as well as the pedagogical implications of using these controllers.

Tod Machover, of the Hyperinstruments group (Machover, et al., 1986) states, "Traditional instruments are hard to play. It takes a long time to [acquire] physical skills which aren't necessarily the essential qualities of making music. It takes years just to get good tone quality on a violin or to play in tune. If we could find a way to allow people to spend the same amount of concentration and effort on listening and thinking and evaluating the difference between things and thinking about how to communicate musical ideas to somebody else, how to make music with somebody else, it would be a great advantage. Not only would the general level of musical creativity go up, but you'd have a much more aware, educated, sensitive, listening, and participatory public." (Oteri, 1999).

With practice, an individual can control most variables of an instrument well and at very fast speeds. However, the initial performance accessibility of an instrument or control interface has definite implications for its use by individuals as a musical instrument—in particular, those individuals who lack formal musical training and those who have physical or mental impairments.

In computer science, the term "mapping" is used to describe the correspondence of one set of data with another set. The potential mappings of musical variables to software controls has been the subject of recent experimental research (Couturier, Kessous, & Verfaille, 2002; Goudeseune, 2002; Levitin, McAdams, & Adams, 2002). Hunt and Wanderly (2002) conducted studies in which participants performed music making tasks using four control interfaces exemplifying two mapping types: one-to-one and many-to-one. One-to-one mapping types allow single musical variables to be controlled by a single controls mechanism of an interactive system. A many-to-one map allows numerous musical variables to be controlled by a single controls mechanism in an interactive system; more similar to the example of a violin bow controlling numerous musical variables as described earlier. In this research, the interfaces with many-to-one mappings were more engaging for subjects during the musical activities, yet both types of interfaces allowed subjects to perform the required tasks.

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 can be acoustic or electronic in design and Crowe (2004) has reviewed the literature of electronic adaptive instruments used to assist in music making. Recent 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).

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). The development of such systems may be accomplished using game development environments as these also allow music and sound variables to be mapped and manipulated with 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 2. Character cannot avoid enemy without an action.

In Figure 2, 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.

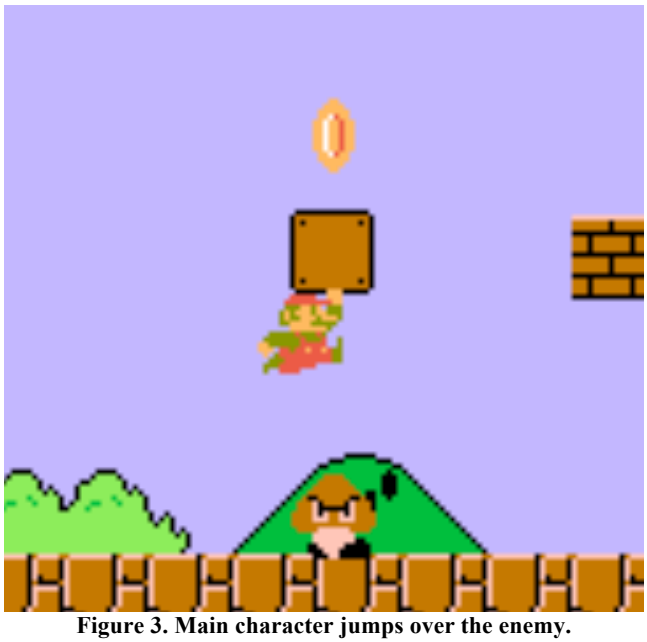


Figure 3. Main character jumps over the enemy.

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 developed by Scirra. It is free for non-commercial use.

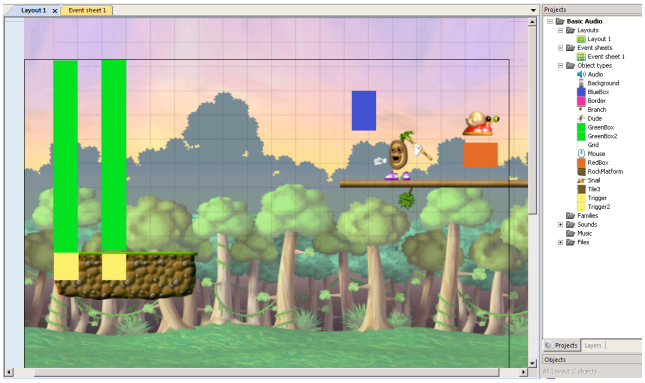


Figure 4. Construct 2 main layout showing assets

As shown in Figure 4, *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 5, 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.

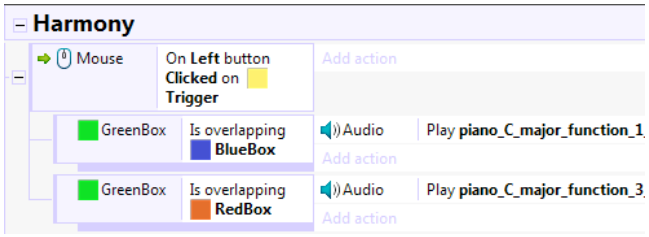


Figure 5. Construct 2 event-based programming sheet

As shown in Figure 5, 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 6, 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 6. Note-making characters are stacked as pitches

Unity 3D

Unity 3D 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 7, 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.

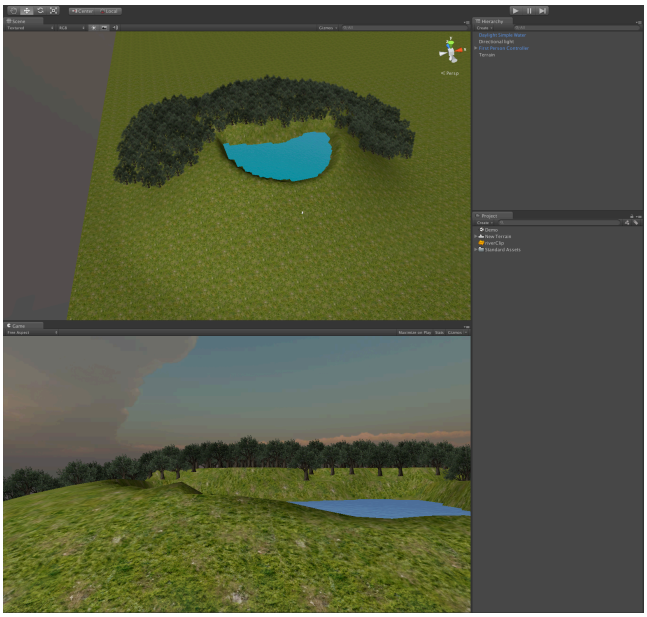


Figure 7. Basic Unity 3D terrain showing shapes with textures

As one develops in Unity, they 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 8. 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.

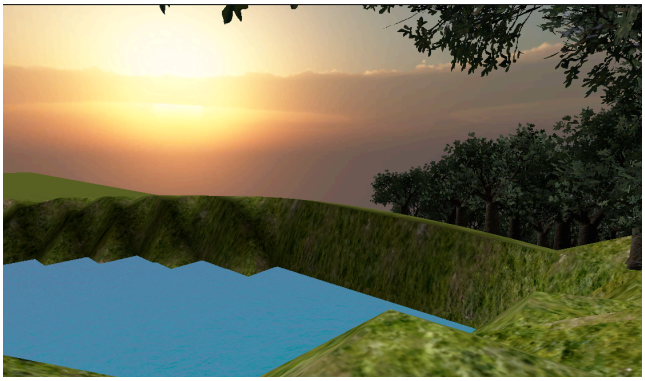


Figure 7. First-person character explores the environment

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. An expanded Unity 3D tutorial is included in Appendix A.

DISCUSSION

Gaming development programming environments allow for a convergence of multimedia elements within a single environment. This articles mentions several game programming environments and focuses on two specifically, noting how they may be used to create rich, immersive, interactive music systems that supported the composition and performance efforts of non-musicians. Research into the efficacy of such systems to support musicianship stems from prior interactive music projects that involved the development and use of software applications designed to support musical creativity by musicians and non-musicians.

There are implications for the use of such systems, particularly by programming novices, in university music classes. Games similar to those mentioned in this paper can be constructed similarly yet modified in terms of their objectives, musical results and rewards. Concepts of theory and composition can be demonstrated and explained through expansion on the Construct 2 demonstration. Similarly, aspects of critical listening and timbral recognition can be demonstrated and explained by expanding the Unity 3D demonstration. The gaming environments themselves are intuitive tools that can provide an accessible development interface for educators and researcher, and facilitate informal music learning opportunities for students. The use of gaming systems for such applications may be useful to individuals with an interest in using multimedia tools to create interactive music systems that allow end-users to compose and perform through software.

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] Arfib, D., Couturier J.M., Kessous L., & Verfaille V. (2002). Strategies of mapping between gesture data and synthesis model parameters using perceptual spaces. Organised Sound: Cambridge University Press, 7(2) 127-144.

[2] Crowe, B. J. (2004, Winter). 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] Goudeseune, C. (2002). Interpolated mappings for musical instruments. Organised Sound: Cambridge University Press, 7(2) 85-96.

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

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

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

[8] Levitin D. J., McAdams, S., & Adams, R. (2002). Control parameters for musical instruments: a foundation for new mappings of gesture to sound. Organised Sound: Cambridge University Press, 7(2) 171-189.

[9] Oteri, F. J. (Interview with Todd Machover). (1999). Technology and the future of music. Retrieved May 25, 2011, from NewMusicBox: http://www.newmusicbox.org

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

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

[12] Pask, A. (Interviewer) & Oliveros, P. (Interviewee). (2007). The adaptive use instruments project. Retrieved July 11, 2011, from Cycling '74: http://cycling74.com/2007/12/07/the-adaptive-use-instruments-project/

[13] Schogler, B. (2010). Skoog music. Retrieved Oct. 24, 2011, from Http://www.skoogmusic.com: Http://www.skoogmusic.com

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

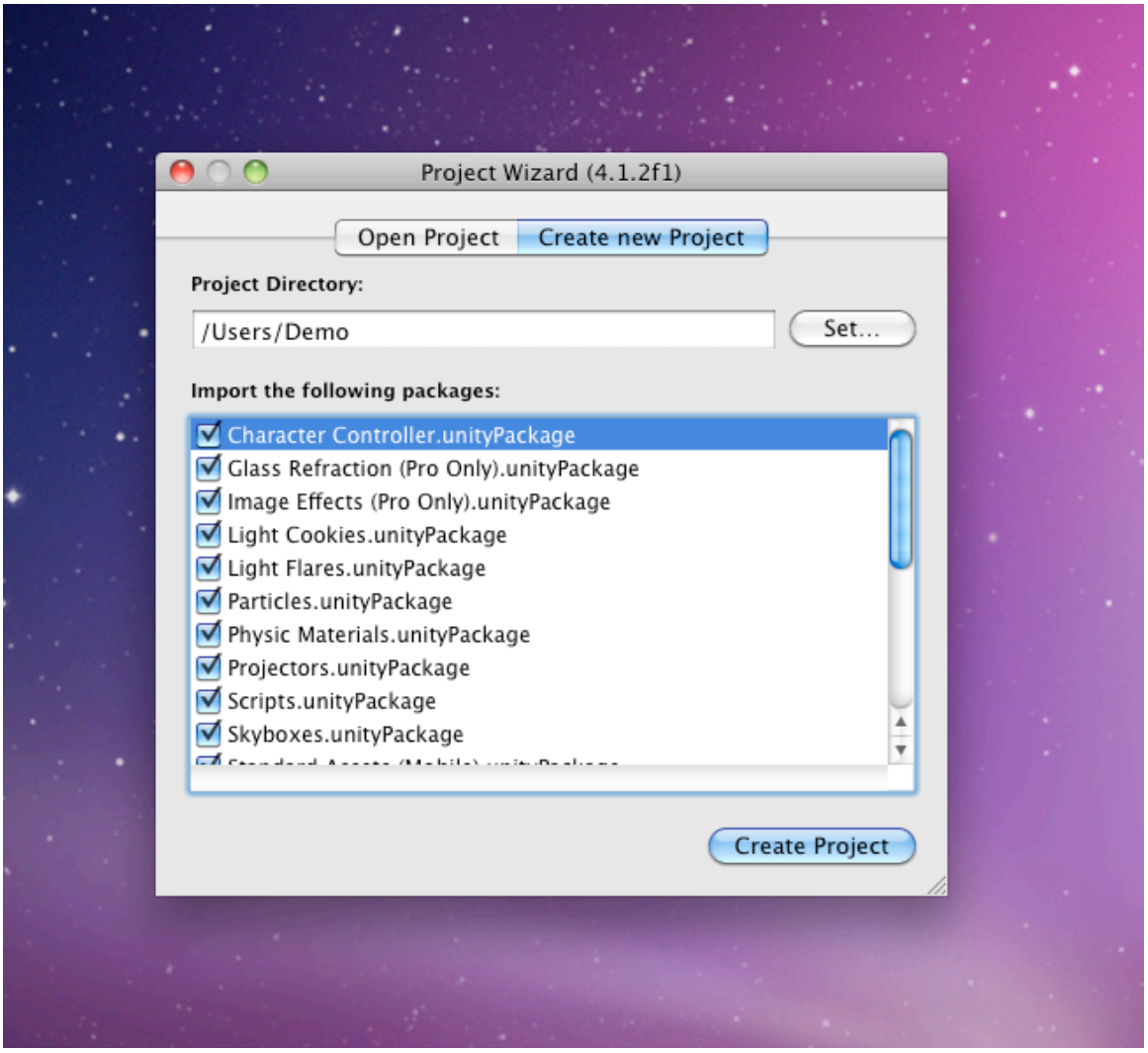
[15] Wel, R. V. D. (2011). . Retrieved July 5, 2011, from My Breathe My Music Foundation: http://www.mybreathmymusic.com

APPENDIX A

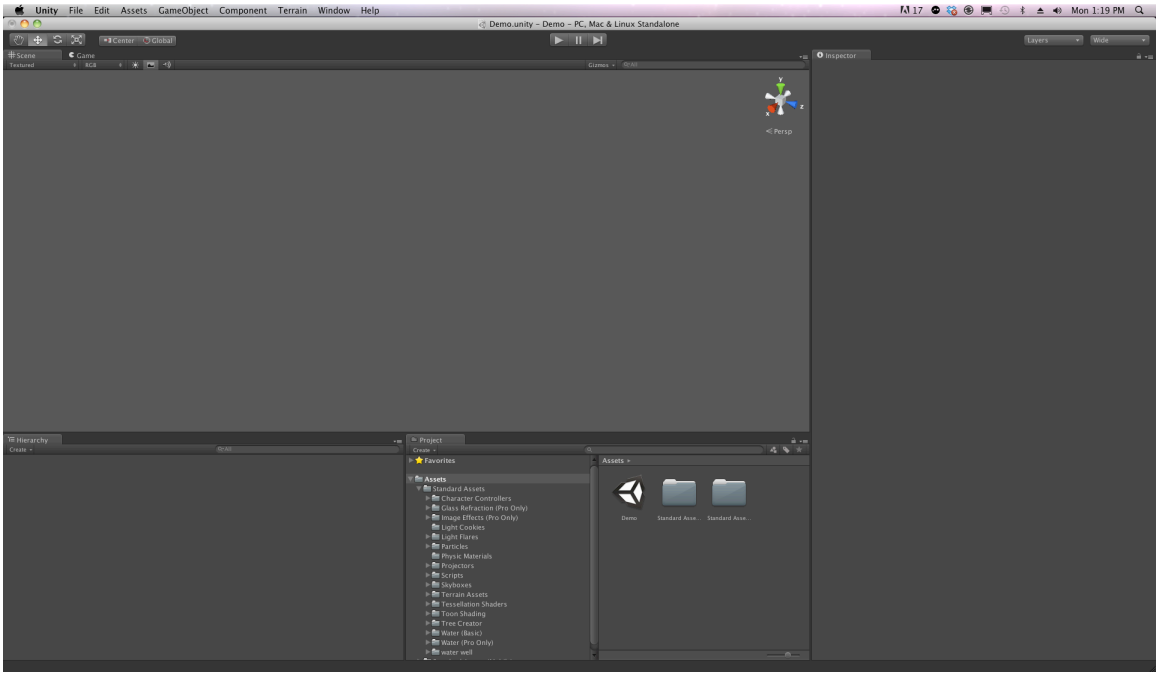
Unity 3D Expanded Tutorial

Below is an introductory step-by-step tutorial for creating an audio project in Unity 3D. Download and install Unity 3D for free from **www.unity3d.com**.

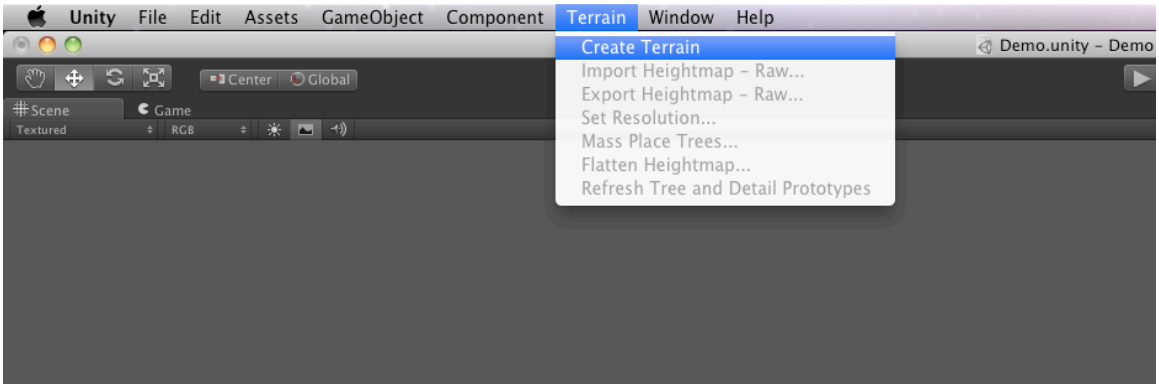
- Open up Unity3D and select the tab "Create New Project"
    - Select a location to save the files
    - Import any additional packages that you would like to use
        - For this example, we will use the Character Controller, Water, and Light Flare packages

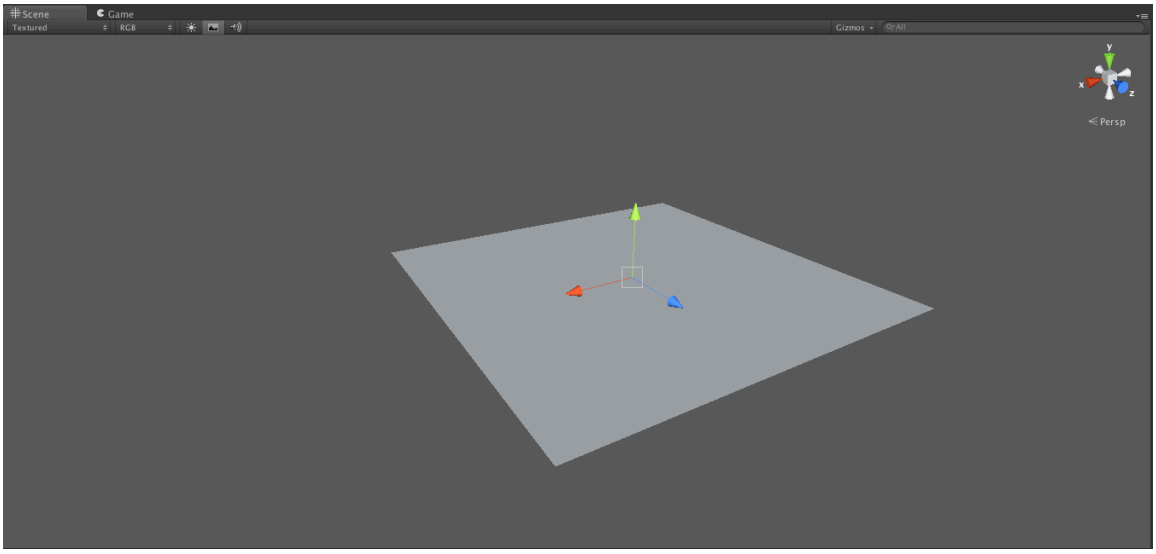


- Once the packages load, we are brought to a blank scene

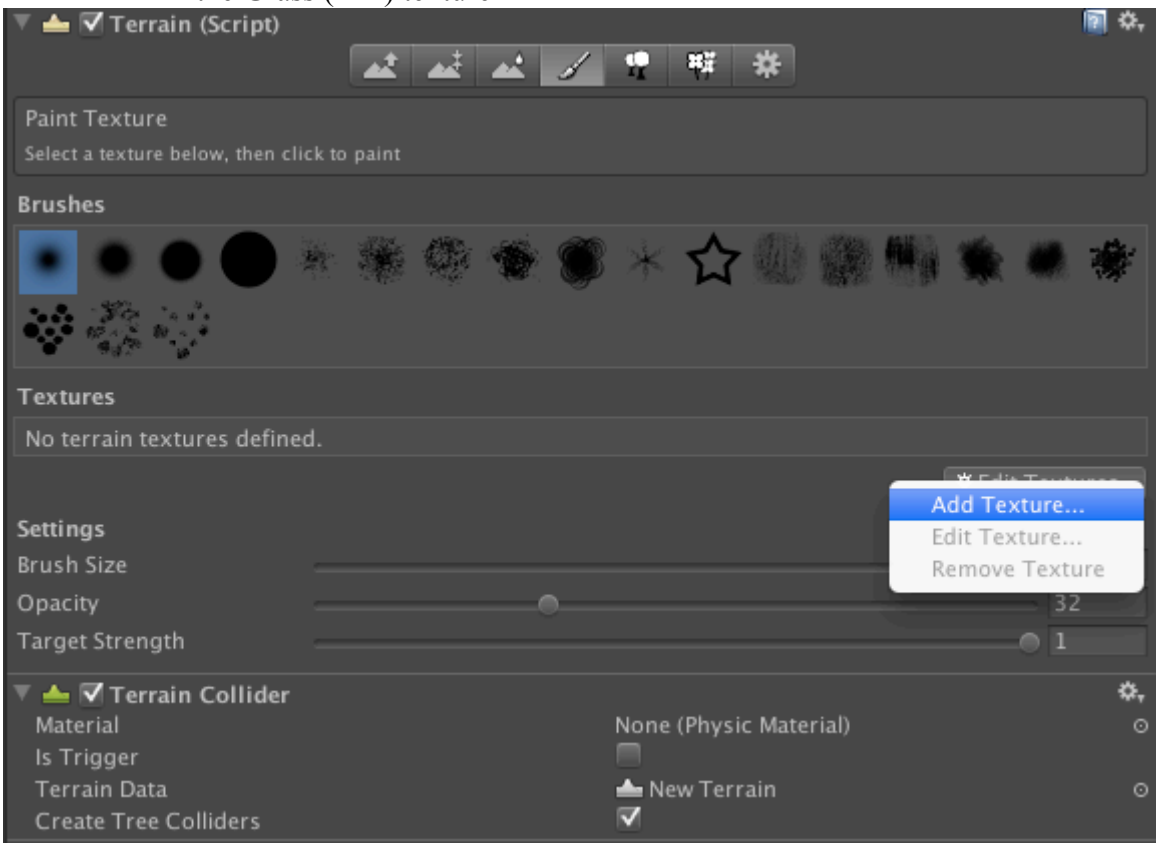


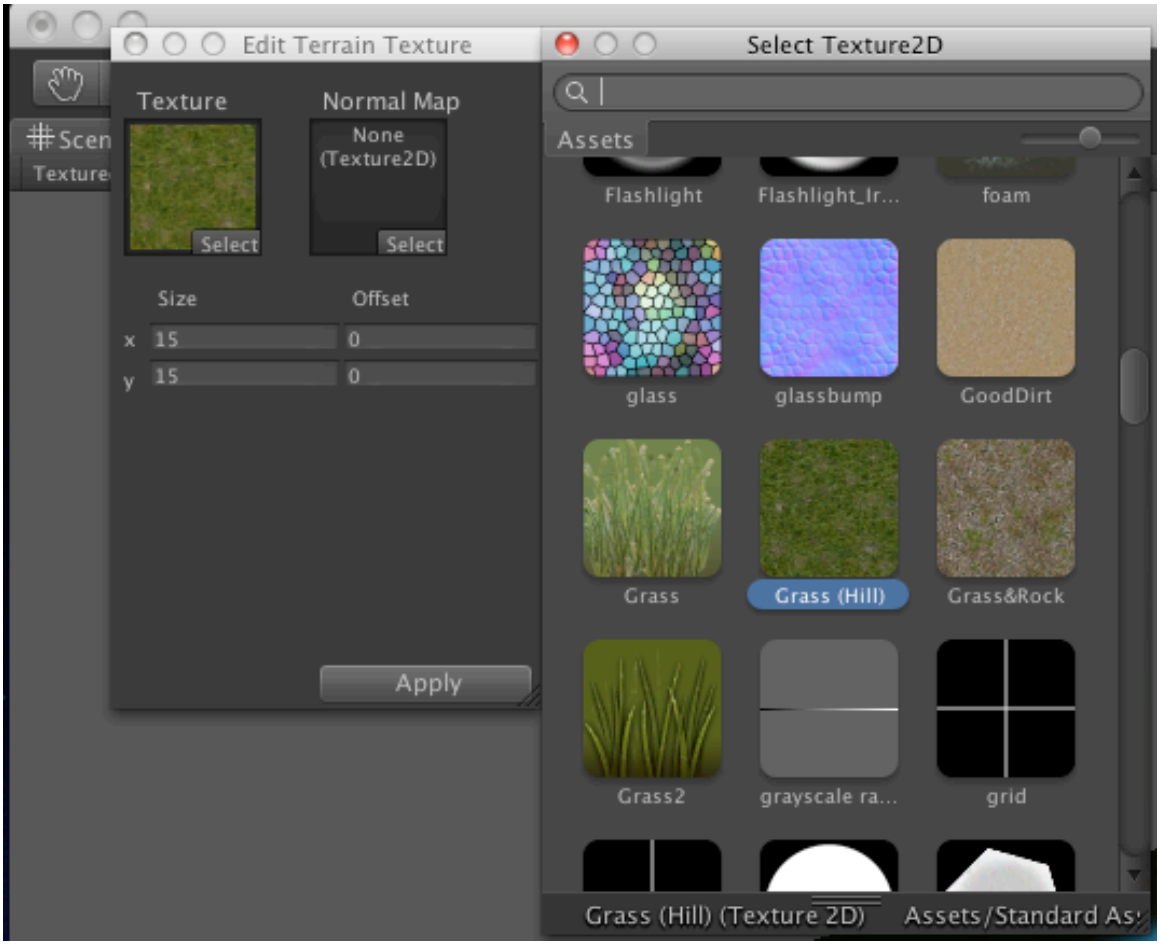
- First, let's make a terrain for our character to walk on. We can achieve this by going to the Terrain menu at the top and selecting Create Terrain



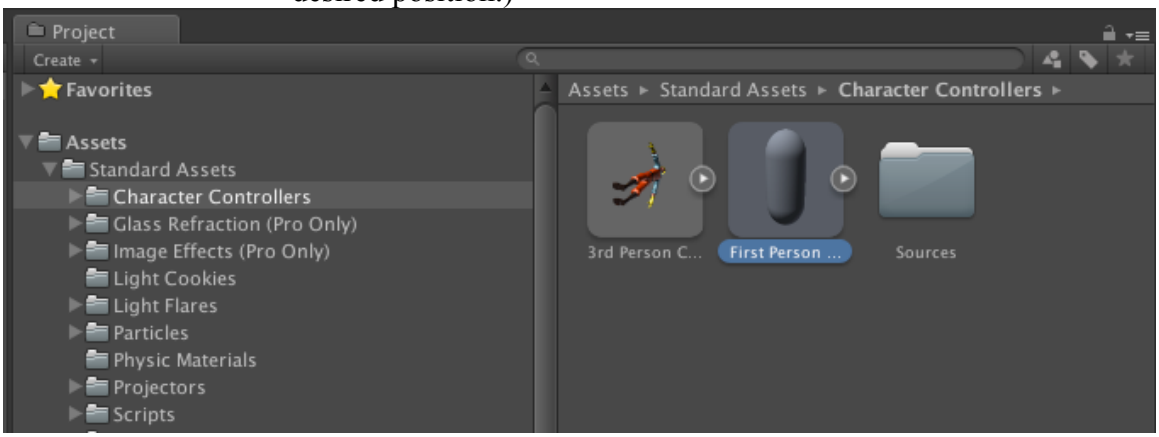


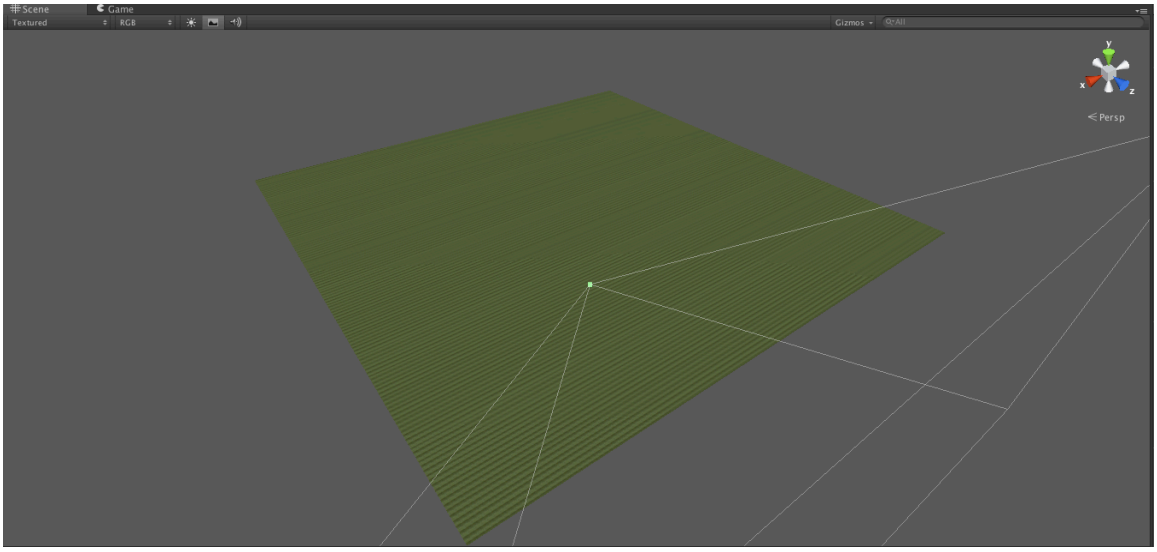
- Next, let's dress our terrain up and make it look a little nicer. To do this, we will go to the terrain options on the inspector panel.
    - Using the paint option, we'll add a base texture for our terrain. In this example, we'll use the Grass (Hill) texture



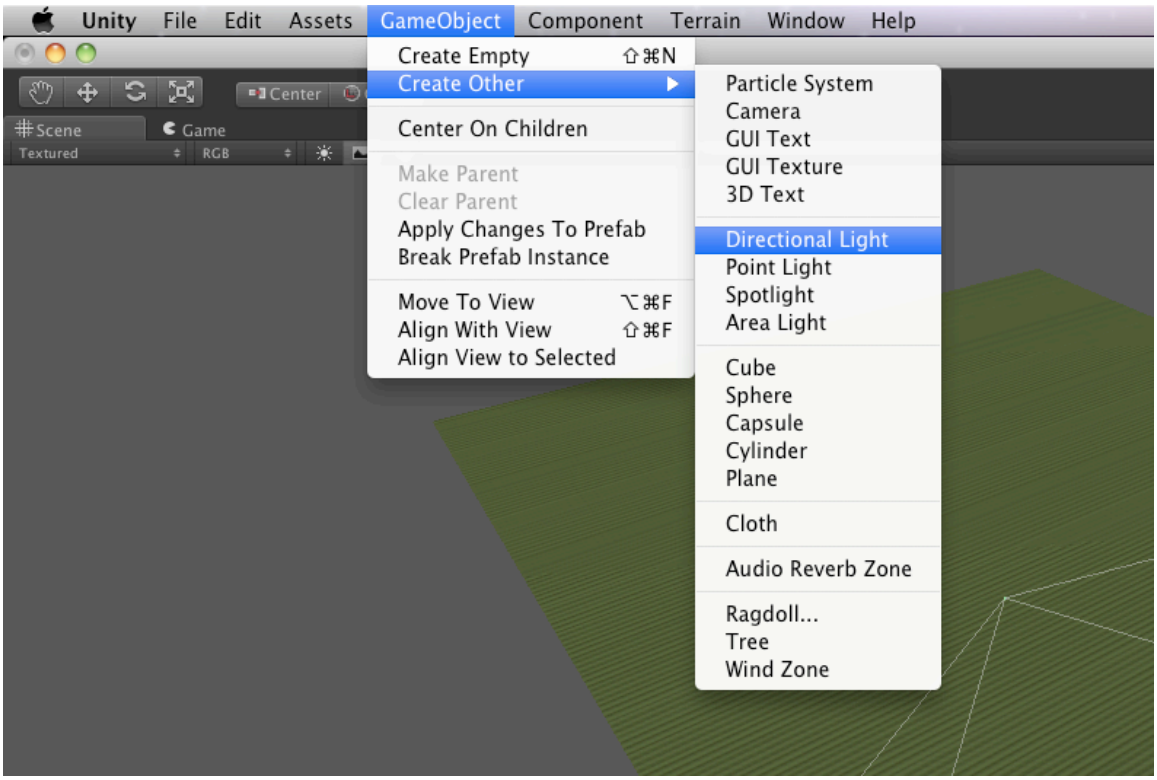


- Now, that we have some land to walk around on. We'll add our character. In this example, we'll use Unity's First-Person Controller prefab to get us started.
    - Go into Assets -> Standard Assets -> Character Controllers
        - Grab the First-Person Controller object and drag it onto the terrain where you would like the character to start
        - (NOTE: the controller may be placed underneath the map depending on where your camera position is. Use the inspector to change the XYZ position to the desired position.)

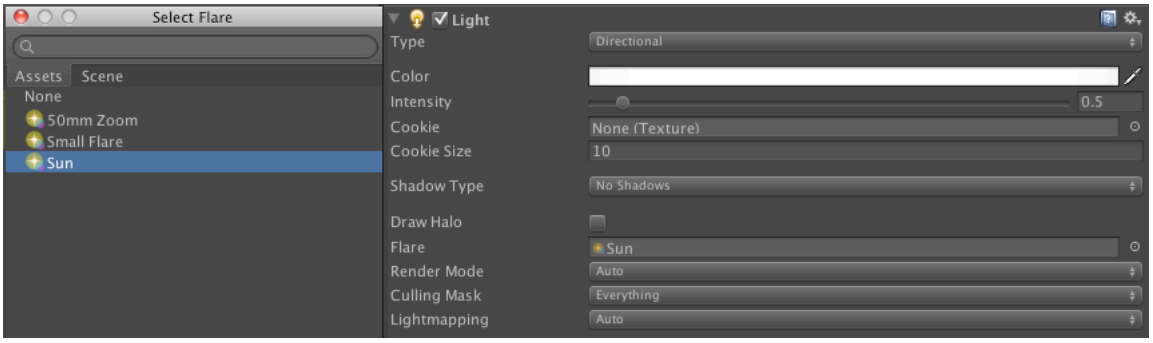




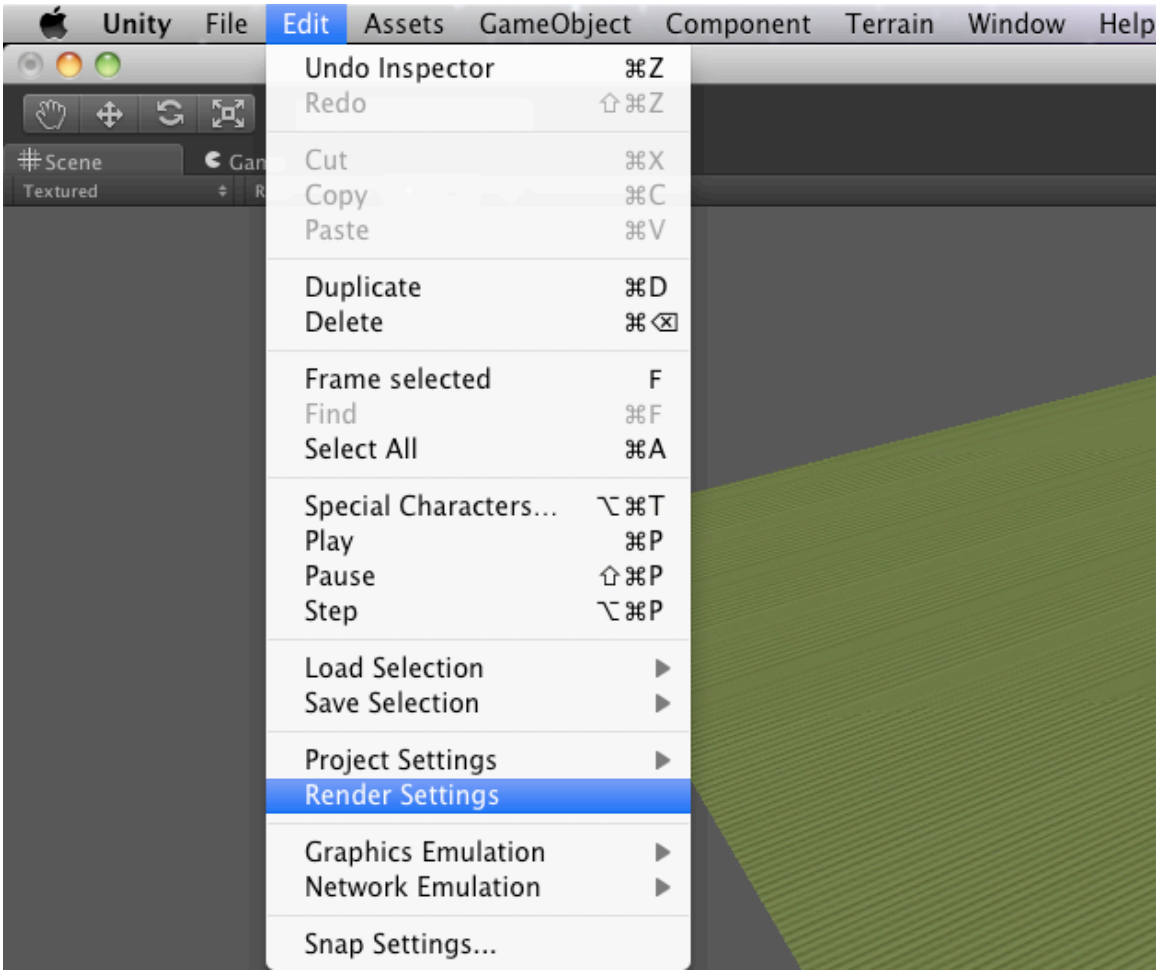
- Great! Now, it'll be a little dark on the map for our character so let's add a sun and sky to our level. Go to the GameObject menu at the top -> Create Other -> Directional Light (NOTE: the directional light may be placed underneath the map depending on where your camera position is. Use the inspector to change the XYZ position to the desired position.)

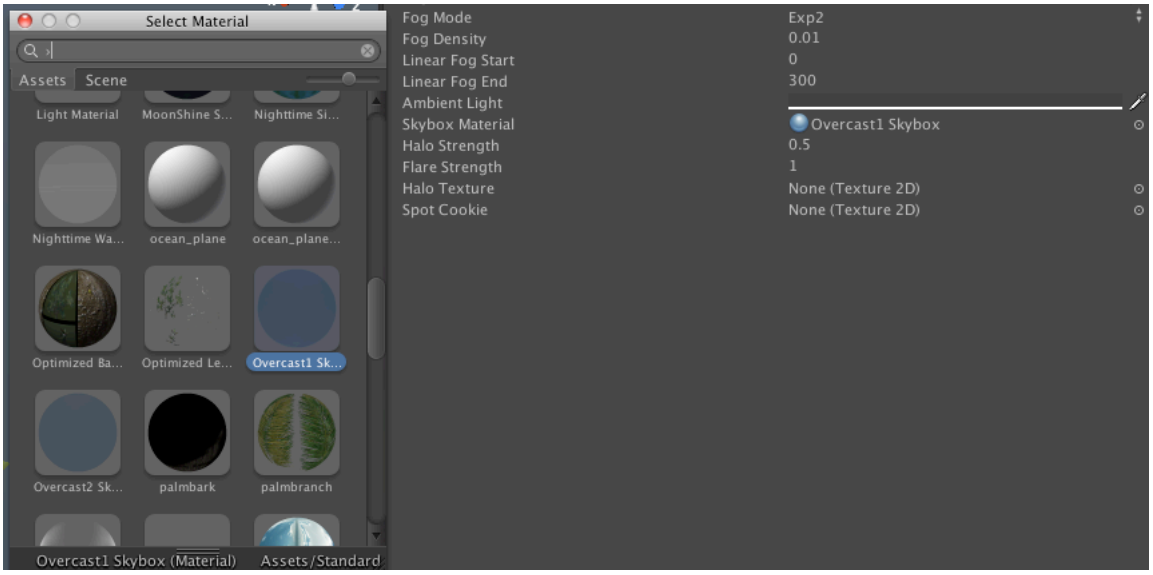


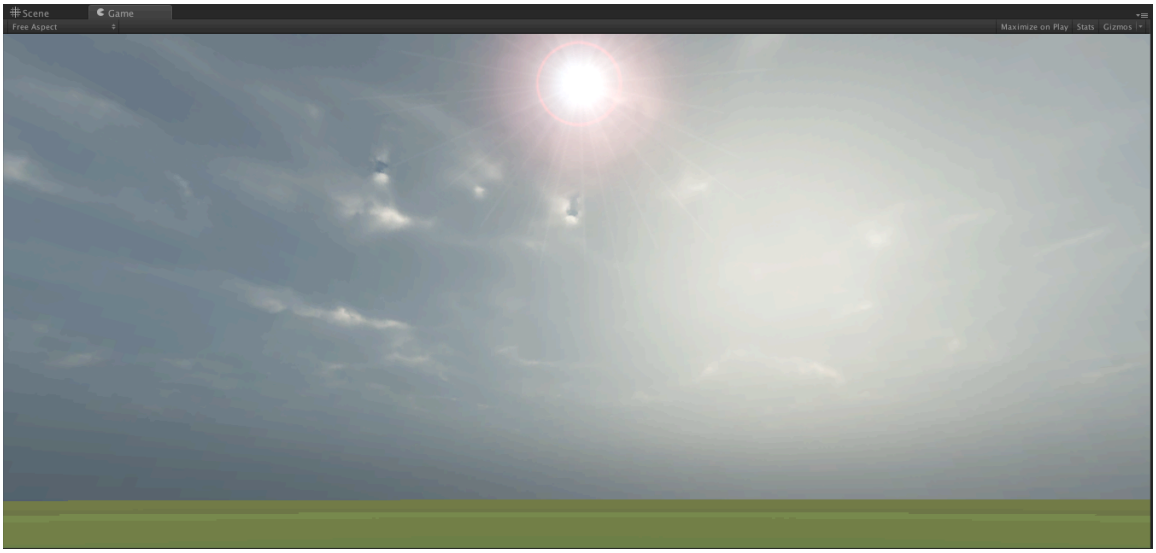
- For a more realistic look, we'll add a sun flare to our directional light "Flare" option in the inspector.



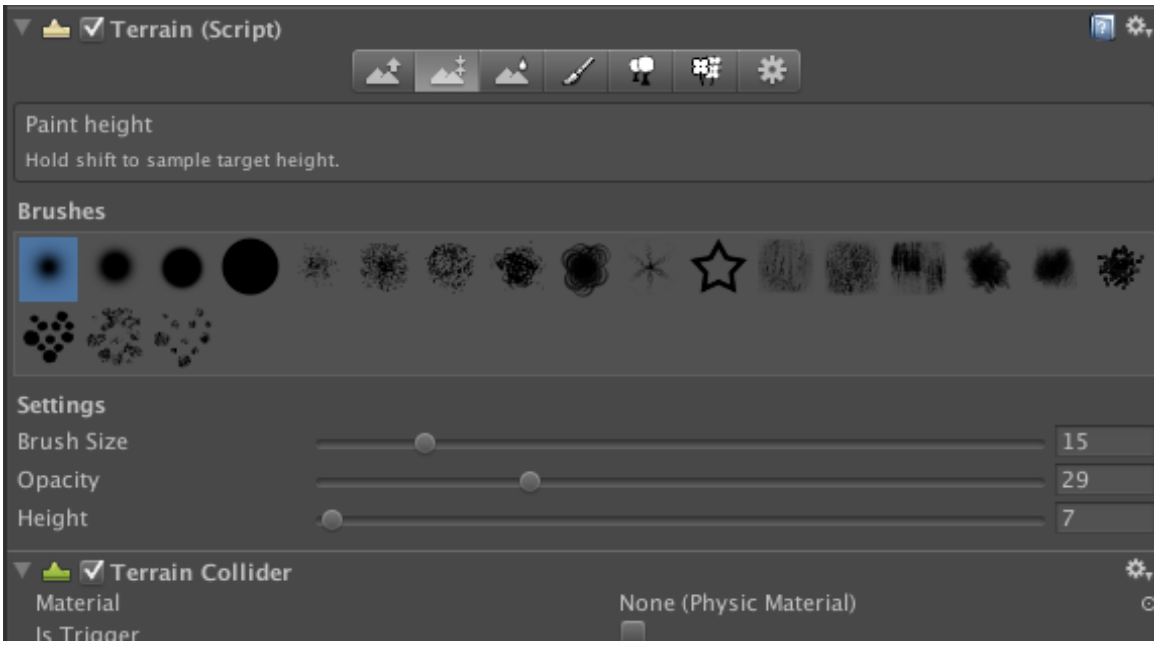
- For our sky, go to Edit -> Render Settings and change our “Skybox Material” in the inspector. Let’s use the Overcast 1 texture.

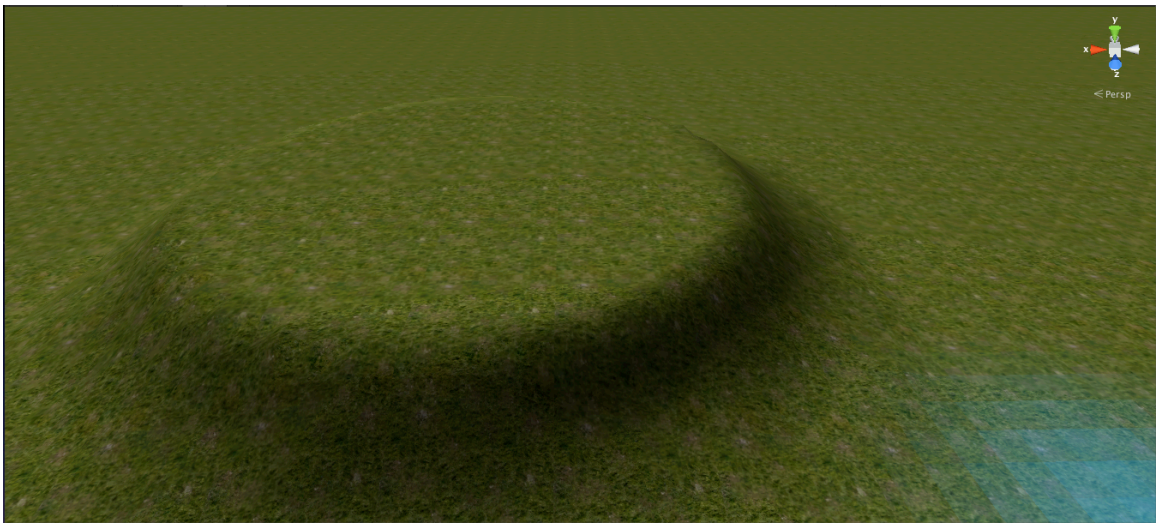


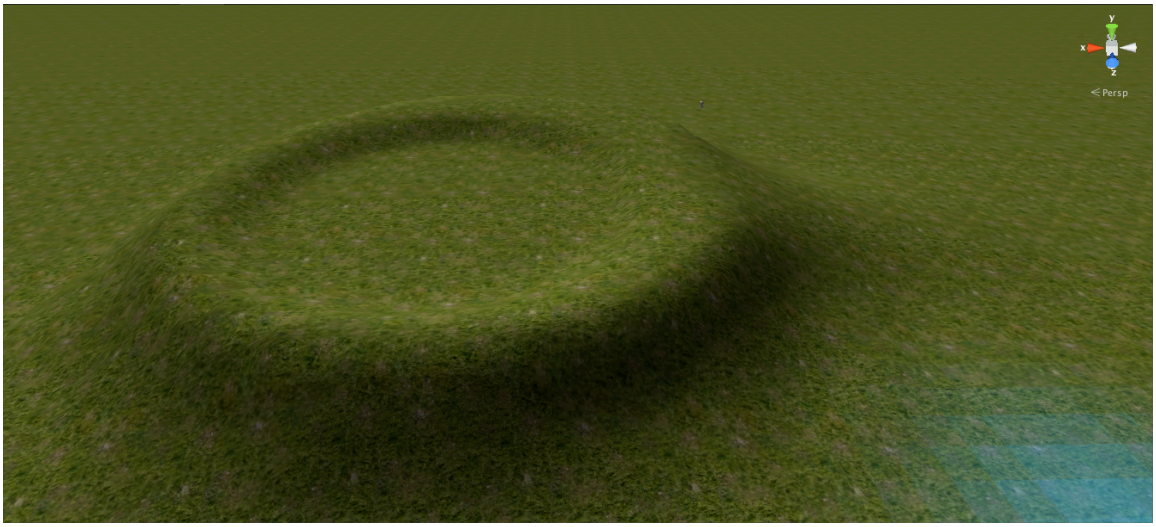




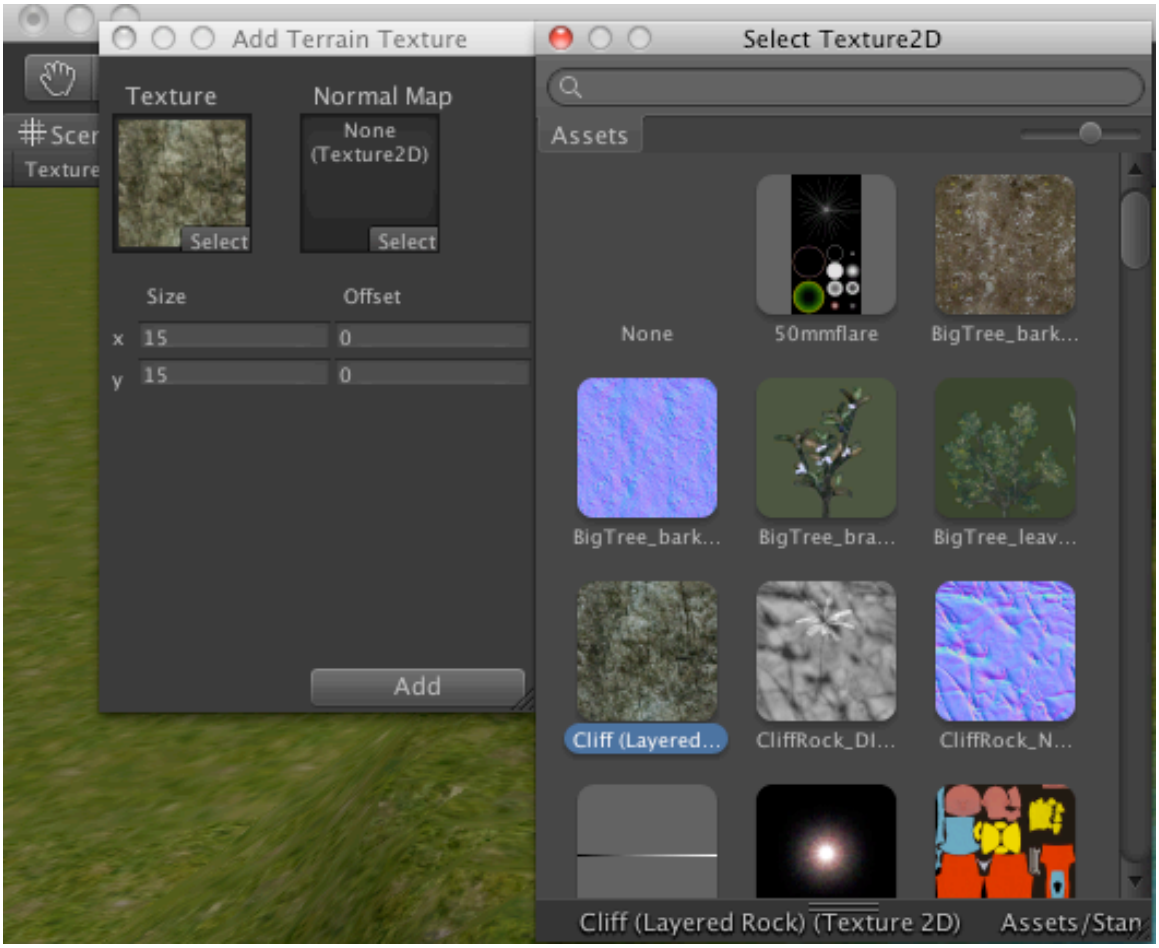
- Okay, it's starting to look like a good level. Let's make a hill that our character can walk up to and listen to the water. In the terrain inspector, let's change the height of a patch of land.
    - Click the mountain icon with the two arrows pointing inward
    - Select your desired brush and settings and begin to change up your terrain
        - The current height of the terrain is at 0, but you can always sample the current height of a particular terrain by HOLDING SHIFT and clicking an area

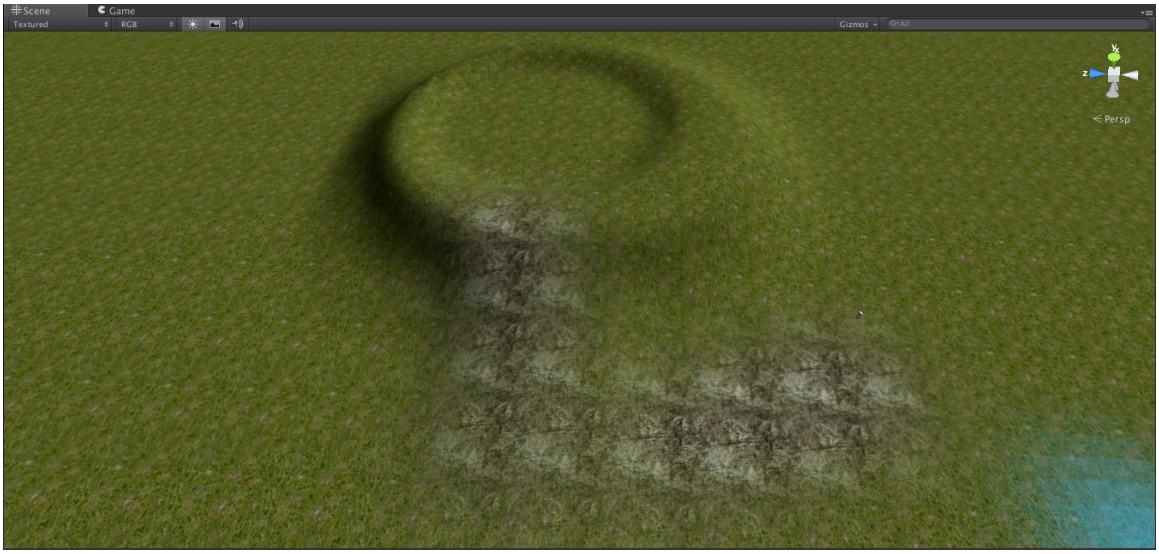

Terrain (Script)
Paint height
Hold shift to sample target height.
Brushes
Settings
Brush Size
15
Opacity
29
Height
7
Terrain Collider
Material
None (Physic Material)
Is Trigger


Persp

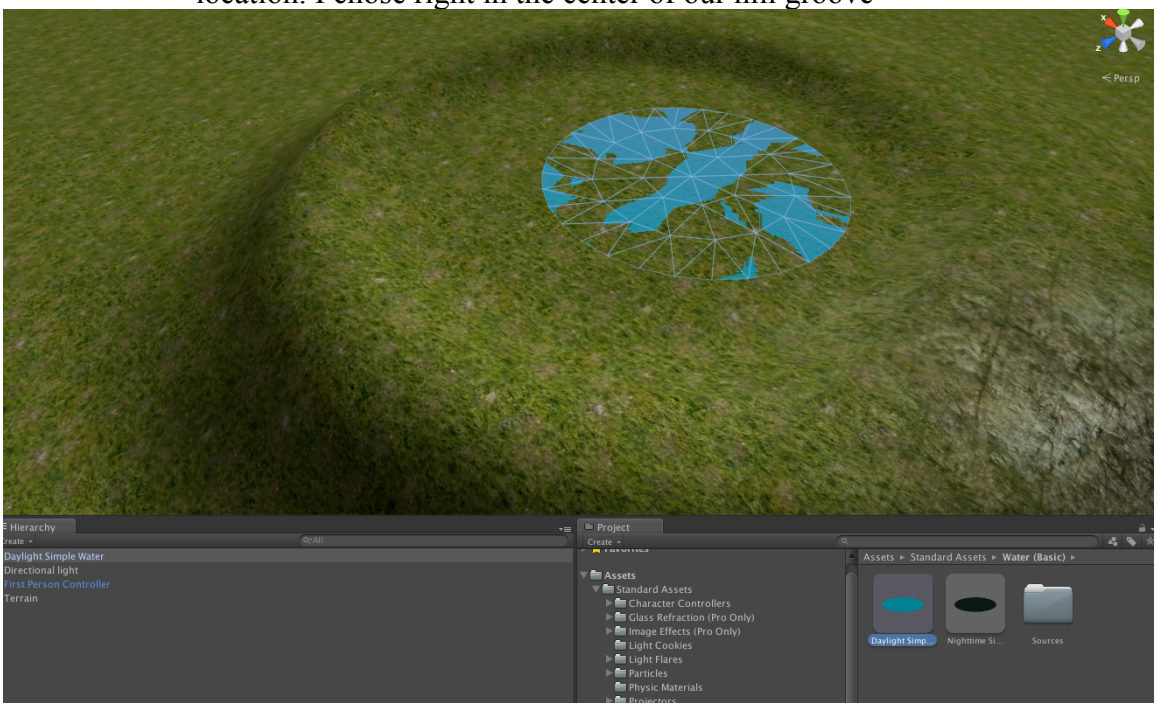


- As you can see, I made a groove for the water to go into and even a small walkway to the right of the hill. Let's add another texture to the terrain and paint a distinct walkway for our character to walk up to back in the paint option of the terrain inspector.

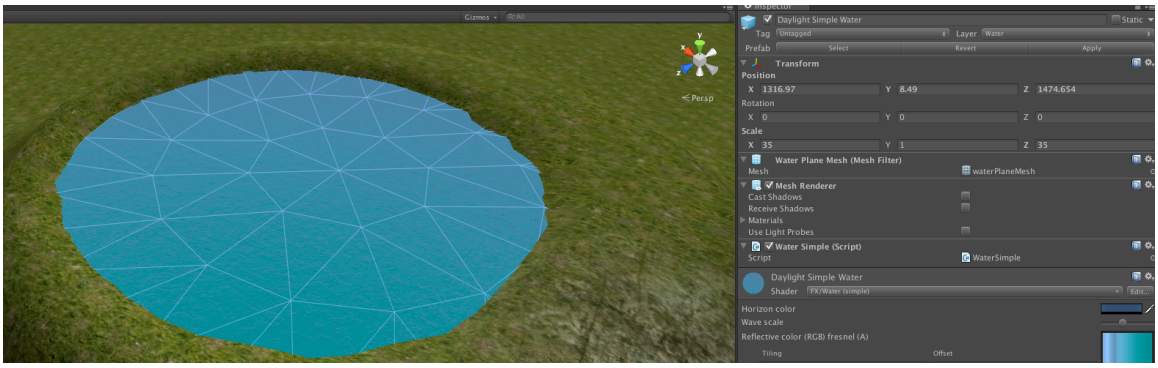


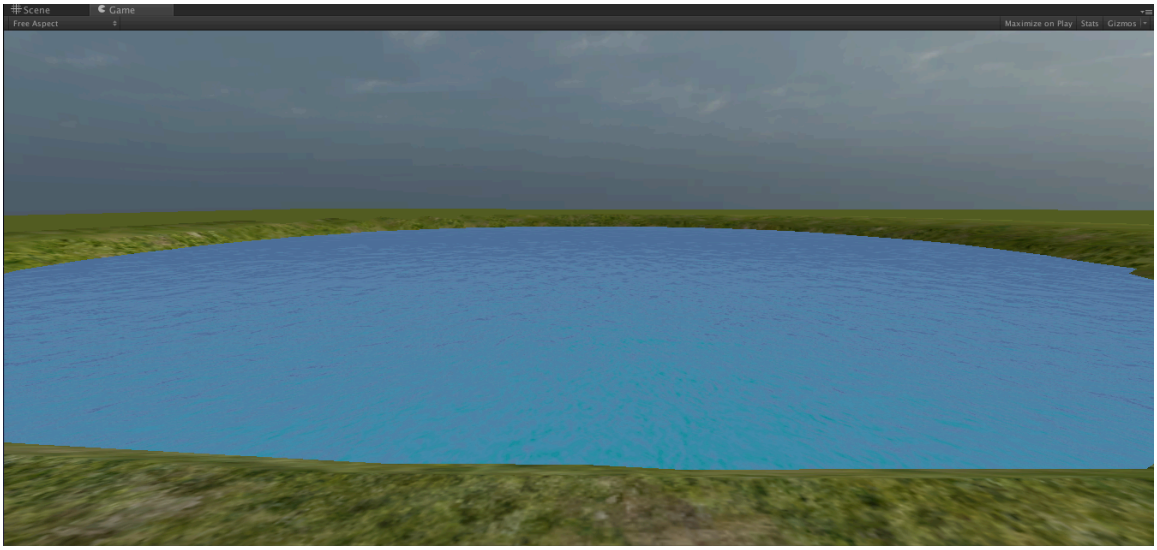


- I placed our character controller at the foot of the path so that he won't have to walk too far to get to the hill
- Now, let's add some water by going to Assets -> Standard Assets -> Water (Basic)
    - For this example, we'll use the Daylight Simple Water so just drag it onto the desired location. I chose right in the center of our hill groove

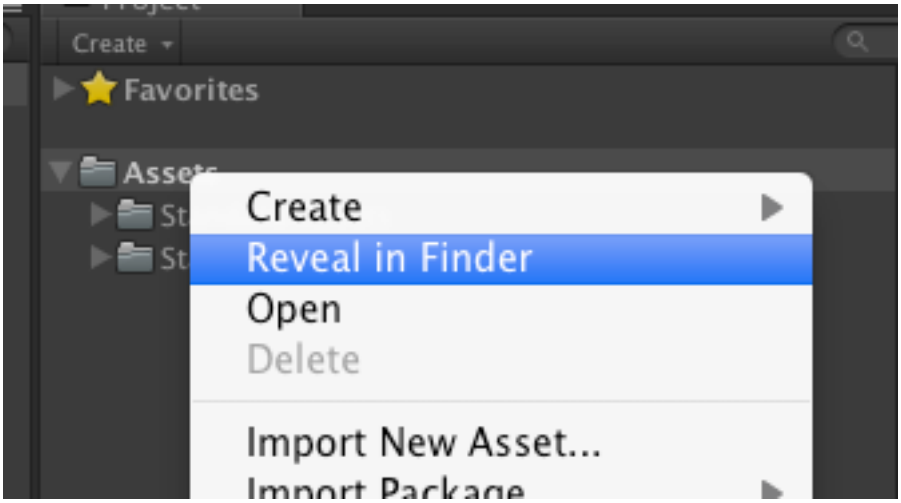


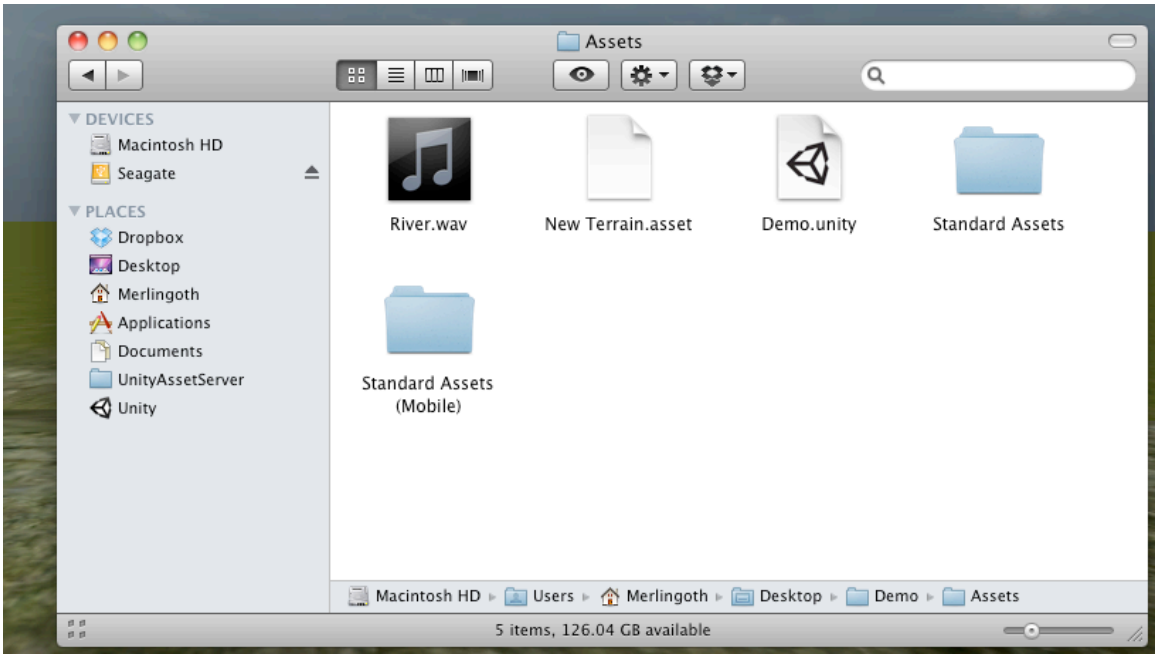
- As you can see, it doesn't look very good. It's because it is at the same level as the hill. Let's change the XYZ position coordinates AND XYZ SCALE it so it fits our groove better.



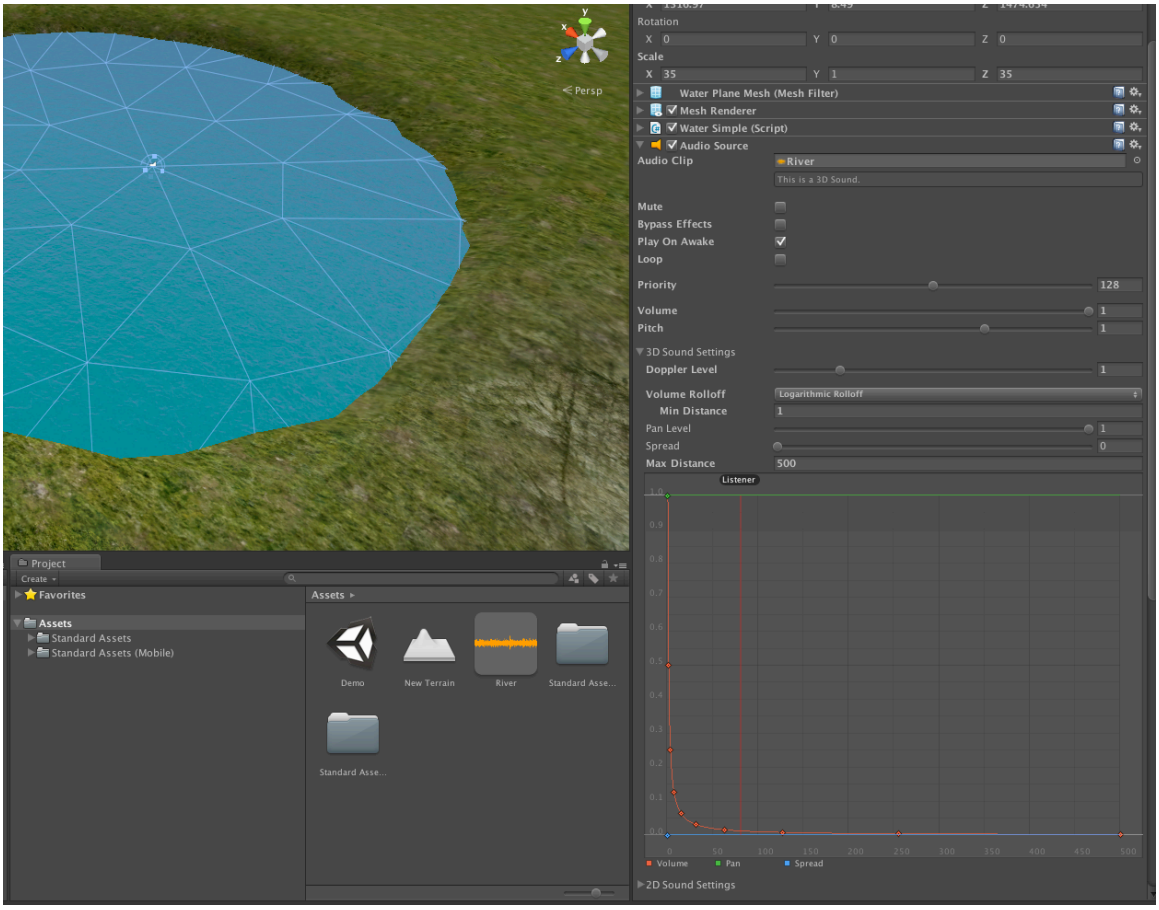


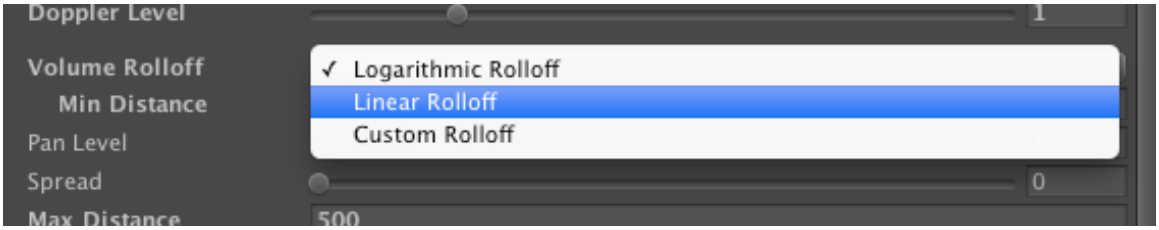
- Much better! Now, the last thing we need to do is set up our audio clip so that when our character walks by he can hear the sound of water. I found my audio clip on freesound.org, but any clip will do.
    - You can add a clip by simply dragging it into your assets folder

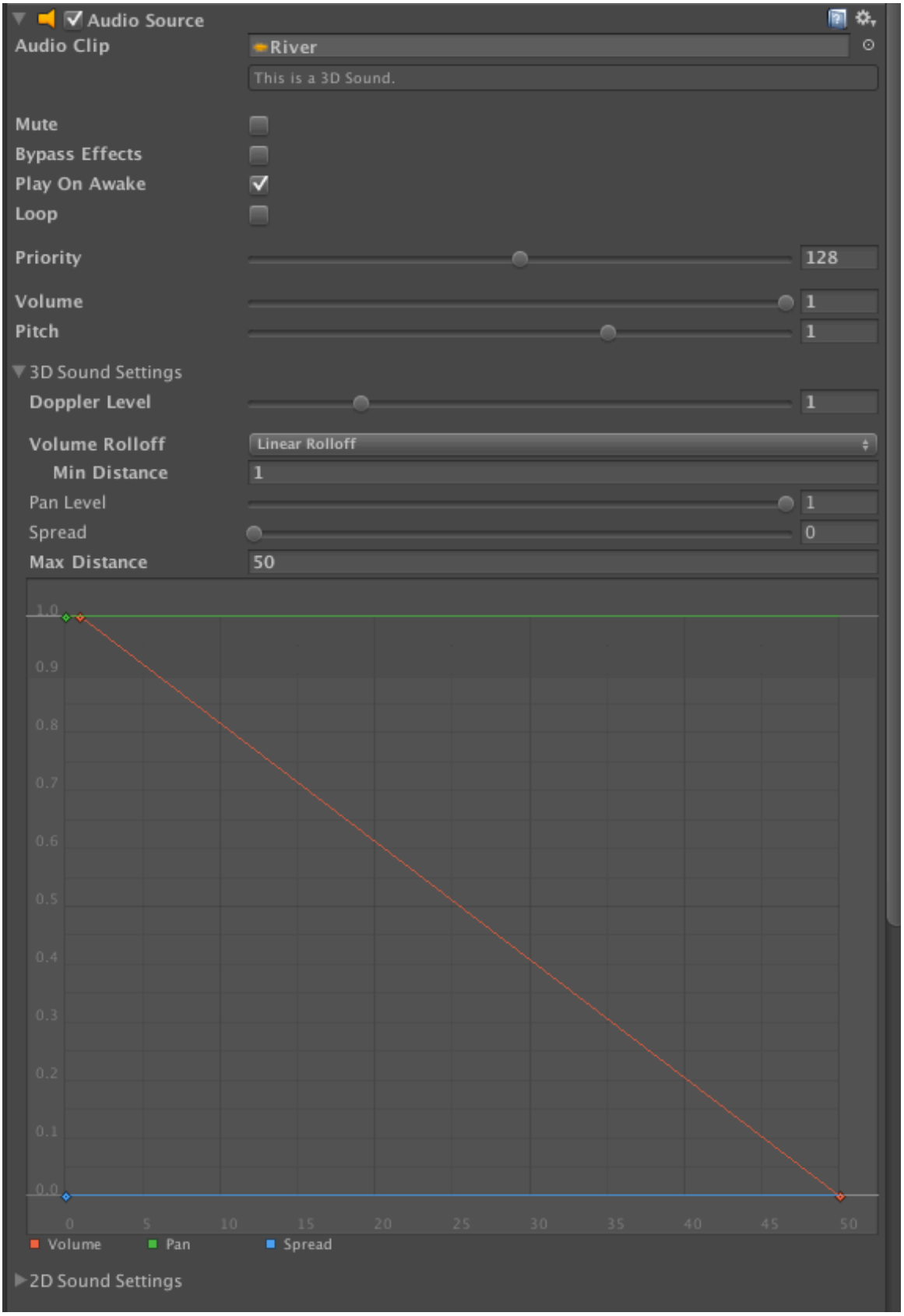




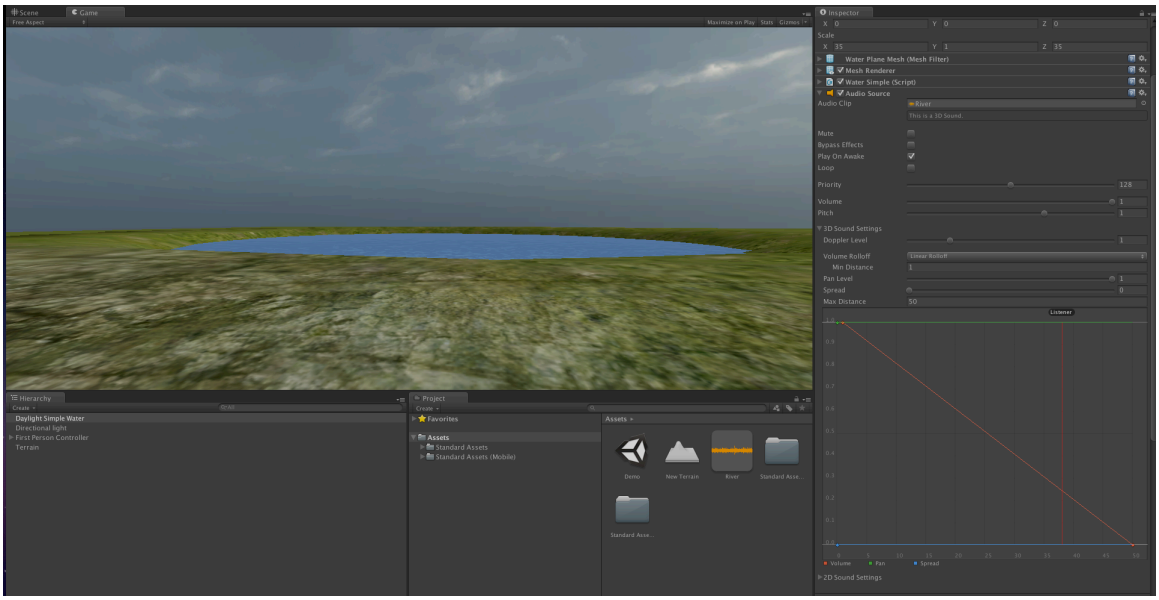
- Now that our audio clip is in our assets folder, simply drag it onto the water object. It automatically adds the clip to the center of the audio object and we can adjust the settings depending on our preferences
    - I used the linear roll-off option so that the volume increases as our character gets closer to the audio source


Persp
Rotation
X 0 Y 0 Z 0
Scale
X 35 Y 1 Z 35
Water Plane Mesh (Mesh Filter)
Mesh Renderer
Water Simple (Script)
Audio Source
Audio Clip
River
This is a 3D Sound.
Mute
Bypass Effects
Play On Awake
Loop
Priority
128
Volume
1
Pitch
1
3D Sound Settings
Doppler Level
1
Volume Rolloff
Logarithmic Rolloff
Min Distance
1
Pan Level
1
Spread
0
Max Distance
500
Listener
Volume
Pan
Spread
2D Sound Settings
Project
Create
Favorites
Assets
Standard Assets
Standard Assets (Mobile)
Demo
New Terrain
River
Standard Asse...
Standard Asse...


Doppler Level
1
Volume Rolloff
Logarithmic Rolloff
Min Distance
Linear Rolloff
Pan Level
Custom Rolloff
Spread
0
Max Distance
500


Audio Source
Audio Clip
River
This is a 3D Sound.
Mute
Bypass Effects
Play On Awake
Loop
Priority
128
Volume
1
Pitch
1
3D Sound Settings
Doppler Level
1
Volume Rolloff
Linear Rolloff
Min Distance
1
Pan Level
1
Spread
0
Max Distance
50
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
5
10
15
20
25
30
35
40
45
50
Volume
Pan
Spread
2D Sound Settings

- As you test it and walk near the source, you will see the Listener object that is attached to our Character show up



- We’re finished! Now when we press the play button on top, we can walk up our hill and hear the audio clip as we get closer to the water!!