TEAM VULPECULA

Adrian Buenavista - Chris Darnell - Johannes Sunarto

CS467

06.11.2017

INTRODUCTION

After a semester-long project, the culmination of our efforts is at hand. Our project group set out to create a Zelda-Clone 2-Dimensional Action Adventure / Roleplaying Game using the Unity game engine with the intention to replicate certain aspects of the retro Zelda series while adding our own personal twist. This project would prove to be one of great fun but with a very steep learning curve as we had to acclimate ourselves to using and acquiring a bunch of new tools, assets, and services. As we set out to develop our game, we had to overcome obstacles in regards to all working on the same files (which Unity is notoriously bad at), finding the appropriate ways to use the C# programming language in a complementary fashion with Unity's built-in Game Inspector, as well as the working with a system where every single game object can update frame by frame.

In the most basic terms, our program is a computer-based video game. The user is given the chance to take control of the Zelda series' main protagonist, Link, as he journeys through the tailored adventure that we created. The user must navigate through two dungeons full of traps and peril, fight numerous enemies, gain two new abilities, and defeat all the bosses to gather the three pieces of the sacred *Triforce* which will grant you victory.

Setup/Usage

Executable Game Application

Our game is pre-compiled into an executable called Zelda.exe that runs on Windows x86 systems. You will probably get a *Windows Defender* / antivirus warning when opening it for the first time; just ignore this and open anyway.

A Unity-Launcher splash screen will come up, the default settings are good (make sure it says “Beautiful” for graphics quality:

Example Preferred Game Settings

Controls:

W-S-A-D or Arrow Keys for movement

[c] for character stats menu

[e] to interact

[space] to sword attack / continue interaction [left ctrl] for bow

[z] for bomb

[g] for god (or grader) mode

Playthrough Instructions

1.

Software: Unity Overview

Our project relied almost exclusively on unity as our core Software System.

Unity - Setup

1. Download/install latest version of Unity Game Engine at https://unity3d.com/ 2. If Unity asks to make an account, just make a free personal use one

3. Extract or git clone our project repository.

4. Open the project repository folder in unity as a project

Unity - Layout & Functions

o Hierarchy On the left of the screen you will notice the Project “Hierarchy”. This shows exactly what game objects are set up in the current scene. Clicking on these objects will allow you to use the “Inspector” which defaults to the right side of the screen.

o Inspector TheUnityInspectordisplaysallthedetailsofthegameobjectsfromthings like the scripts attached, components used, etc. It is an incredibly powerful tool because it provides a visual, programmable interface to game objects and scripts.

o Scene Thesceneiswherewecanmanipulateeverythinginthescene.Thisis where we place our objects, place our levels, and do most of the visual front end work.

o Game Thegameiswherewecanrunthecurrentbuildofourgamewithout having to create an executable. You can press the play button at the top of the screen to play the current scene though this may not always work depending on how your objects carry over from prior scenes.

o Animator Theanimatorisaniftytoolthatcanhelplayoutanimationsforindividual game objects. For example, a sprite sheet will usually have sprites that correspond to walking, standing, jumping, attacking, and more. You can use the animator to create states for these sets of animations. Then you can set up transitions between animation states (such as walking to running, idle to walking, etc.).

o Animation Animationisusedforlookingatindividualanimationsforasprite.Youcan alter the timing or add additional frames amongst many other useful features.

o Project Thisisessentiallytheentirerepositoryoffiles.Everythingthatisanything in our project is located here. The files are split into their corresponding folders based on whether they are scripts, sprites, sounds, prefabs (which are just set up game objects that we wanted to save for later), *Tiled2Unity* files (which is where our levels were created), and more. Clicking on any of these files will open their information in the inspector. Clicking on a script will open it up in Visual Studio if you have it, or *MonoDevelop* which is Unity's IDE.

o Console LikemostIDEsthisiswherewarnings,errors,orgeneraldebugstatements can be monitored that occur when playing the Game using the play button

Graphics

Example 1 - A Map of Our Game World

Everything accomplished when it comes to graphical presentation of moving elements in 2D gaming via Unity is done with sprites. Down below you will see an example of the kind of files you will see within the sprite folder of a Unity project. A lot of these images have an arrow next to them to represent that each contain sub-images which compose those corresponding spritesheets.

Example 2A - Sprites

Below we have a complete sprite sheet that corresponds to Link. You will see that he has a corresponding image for each direction that he runs, attacks, etc. We used multiple types of sheets like this to compose Links move set. The idea is to group them together to create the appropriate animation.

Example 2B - Spritesheet

Here is a close look at a spritesheet that was cropped up to make some of Links actions. To accomplish this though the spritesheet require a transparent background and clearly defined borders between the images. Otherwise it becomes exponentially more difficult to define individual subimages.

Example 2C - Spritesheet in Frames

With these sprite sets we can compose the action states of the “animator”. The states can be “blend trees” or just individual state machines. A blend tree relies on different directional parameters to output the correct animation which is useful for blending one animation into another. While individual state machines can have simple bool/trigger transition events to do the same thing. All these things stringed together can produce a fully functional animating object.

Example 2D - Animator

Additional Tools, APIs, Libraries & etc.

Github

Over the course of the past 10 weeks, we have relied exclusively on our GitHub repository to keep track of our version control and collaboration techniques. As such we have made 468 commits to Master, 59 pull requests, and countless branches. Unity, however, is notoriously unfriendly to git, and we had many merge conflict issues with Unity scenes which had to be painstakingly manually resolved or re-done. However, git was essential because many times we deleted and re-cloned the local repository because it is relatively easy to get put into a bad state locally due to Unity's massive bulk.

We should also mention that we used Git Large File Storage to host over 50 MB of large binaries such as PNG images and WAV files for sprites and sound effects, respectively. Git LFS was brilliant because it automatically formed a pre-push hook that intercepted our large files with minimal configuration.

Adobe Animator & Photoshop

Adobe Photoshop and paint.net were used to edit source images from the internet and crop spritesheets into usable sprites. Adobe audition was also used to modify sound effects and crop music.

C# and .NET

At the start of this project, none of us had any C# or .NET experience. The Microsoft C# Programming Guide was extremely useful in developing a solid baseline for this language. We are all very fond of C# now as it maintains a good balance between many features, but clean C-style (better-than-Java) syntax.

Tiled is a program that has been essential in level design for our game development project. Back in the earlier days, retro-styled games like the older Zelda games had to find the best methods of utilizing memory especially when it came to the graphic design. They used tiled maps which could be composed of 8x8 or 16x16 in our case. These tilesets could be used together to create a larger image. Luckily for us memory is not much of an issue especially when we are designing a SNES fashioned game on vastly stronger computers.

Instead Tiled is just a method for making this process much more simplified. With Tiled all you must do is find the appropriately sized tilesets, make sure to input correct margins and spacing, then place them in the desired fashion. This allowed us to create different types of levels with whatever topography matches our intent.

Tiled2Unity

The Tiled2Unity program is essentially a plugin that allows Tiled maps to be imported into Unity as a prefab. These prefabs are then able to be dragged into a scene to be used as seen fit. The only gimmick to this tool is that the user needs to be familiar with pixel per unit ratios. Some basic boundaries for collisions are also able to be imported with this method but using too many collision detectors can result in slowdown during runtime especially in larger sized maps.

Over the course of the past 10 weeks, we have relied exclusively on our GitHub repository to keep track of our version control and collaboration techniques. As such we have made 468 commits to Master, 59 pull requests, and countless branches. Unity, however, is notoriously unfriendly to git, and we had many merge conflict issues with Unity scenes which had to be painstakingly manually resolved or re-done. However, git was essential because many times we deleted and re-cloned the local repository because it is relatively easy to get put into a bad state locally due to Unity's massive bulk.

We should also mention that we used Git Large File Storage to host over 50 MB of large binaries such as PNG images and WAV files for sprites and sound effects, respectively. Git LFS was brilliant because it automatically formed a pre-push hook that intercepted our large files with minimal configuration.

Team Member Contributions

Adrian Buenavista

Chris Darnell

Johannes Sunarto

In many ways our team leader and trailblazer, Ellard lead the design of the Village, Forest sections, and Castle sections, which were integral to getting the project started. This included the procurement/allocation of sprites and tilesets to create said sections within the Tiled level creator program, as well as actual implementation of boundaries, collisions, warp targets, enemy spawn locations. Ellard created the dialogue/interaction system that is used by NPCS (non-playable characters) and chests located in the open world. Designed and implemented the game objects and inclusive scripts for the forest dungeon boss, Phantom Ganon of the castle section, spike traps, as well as a few other enemies. He helped create and implement the bow attack ability for Link as well as create the system for unlocking new abilities. He also implemented the in-game shop that allows the player to upgrade their basic stats which can be seen from the in-game RPG character sheet. Lastly, Ellard was the first one to outline our reports.

Michael Hueter

Michael was the Git master of the team from the beginning. He setup the initial Git repository and frequently sorted out merge conflicts every week for the other team members. He also spearheaded code cleanup and refactoring to bring our code within style guidelines and to better leverage C# class inheritance and polymorphism rather than redundant (and less performant) code. He designed the scene transition and warp-between-scenes systems, scripted all the cinematic events, and added most the music and sound effects in the game. Lastly, Michael formatted, cleaned up, and submitted reports and files

Peter Murphy

Peter was an expert at designing enemies and levels. He found and edited many of the enemy sprite sheets which were used to create custom animations for many of the enemies. He created the patrol script used to move the enemies when not active in combat, and he created the tiled maps for the river and river cave scenes. Peter singlehandedly built and scripted the Minotaur and Ganon boss fights, and the entire bomb ability. He also adjusted bow ability to fire from link's position when running diagonally. He implemented an enemy line of sight via unity's raycast system for the generic enemy manager script. Finally, Peter was great at coordinating team efforts by scheduling meetings and organizing our product priorities.

Deviations From Original Plan

Our original project plan included the implementation of a third desert based dungeon. After coming to terms with the software we had to use to carry out our task and the sheer amount of work that goes into making a game, we decided to cut back to just two dungeons followed by a final section. We felt that trying to create the third dungeon would subtract from the overall quality of the product since less time would be spent fine-tuning what we had/creating more meaningful content. The decision was mutually decided and allowed us to create more interesting dungeons/enemies/boss fights. In addition, for the sake of creating a product that could be graded/tested within a reasonable amount of time, we felt that the removal of the desert dungeon would allow our game to fall within this amount of time.

Besides this change, the project plan was essentially followed with maybe the addition of every project team member having to assist others with their work due to the amount of overlap to maintain possible scalability and to downsize the amount of potential runtime conflict.

It should also be noted that we did not utilize Asana as much as we originally intended; this is because we had an excellent system of meeting every Monday night and divvying up individual tasks while noting collaborative opportunities in the coming week. We also found that the static analysis tool Fxcop was not necessary because the combination of VisualStudio and Unity were extremely verbose and detailed in pointing out warnings, errors, and in many cases best practices.

CONCLUSION

Overall this has been a very enlightening experience into the world of game development. The simplest way we can summarize this entire process is that game development is very easy to get into and startup but quite frankly extremely difficult to perfect. There are numerous considerations that must be considered when creating a game. For example, there are dozens of ways to accomplish the same goal but some are more resource intensive than others. In addition, whenever creating something, it should always be a general goal to make it as scalable as possible so that it can be used again later. Creating individual objects with specific scripts have their place but putting in the extra work to make some of the objects usable in other areas makes the overall product more maintainable and will shorten the work in the long run.

Another essential point to bring up is the sheer importance of the artistic aspect of game design. A game is after all interactive art. We used assets that were readily available to us online (though some took quite a bit of manipulation and effort to find) but if we had to create our own we would find ourselves deeply stretched for creating something decent looking. While what we have created is already quite a feat, it is unfortunately also not ever able to be published to the public because it uses copyrighted artistic materials.

In the end the experience has been very enjoyable and not to mention has honed in our programming skills to create some meaningful and personal. We hope you enjoy playing the game as much as we enjoyed creating it.

REFERENCES