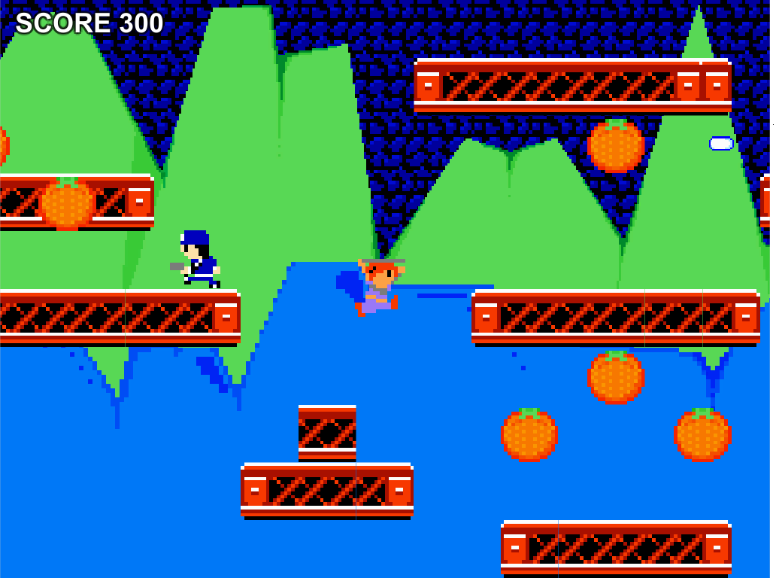
**Readme for IGME-119 Platformer**

**By Jonathan So**



Github Link: <https://github.com/jon101514/IGME-119-Platformer>

**Introductory Notes**

Welcome to IGME-119, 2D Animation and Asset Production! Over the semester, you will be creating graphics for game sprites, UI elements, and even your own cutscene to replace the default ones found in this Unity project. If you wish to add your own gameplay-related personal touches to your copy of the game, I’ve developed tools in-engine which allow you to modify gameplay without programming knowledge.

Please use **Unity 5.6.0f3** for this project; this is the version used in the classroom for other courses in the department. Opening this project with other versions of Unity (especially older ones) may result in errors.

Thank you very much to **Prof. Elouise Oyzon** for affording me this opportunity to create a tool and game that can be helpful to the students. Thank you to **Dan Giaime** for creating the previous version of this project and for the cutscene placeholder video.

**Importing Sprites**

Enclosed in the Sprites folder (IGME-119 Platformer > Assets > Sprites), you will find multiple images that will be overwritten by your artwork. For the UI elements, background, and two parallax layers, **saving your graphics in place of the file** is fine; however, the objects displayed during gameplay, such as collectibles, NPCs, etc., require a different solution.

All of the gameplay sprites are stored in a spritesheet, and Active\_Sheet.png is the spritesheet currently being used by Unity to display all of the objects during gameplay; the engine pulls graphics from that sheet and places them into the game with carefully-calculated bounding boxes created for every sprite. To make sure that your own sprites make it into the game without any errors, I have created Blank\_Sheet.png, which is a guideline that displays proper sprite positions and resolutions on the sheet.

I suggest that in a graphics editor, you work on your sprites on a layer above Blank\_Sheet.png and, when completed, save the whole file over Completed\_Sheet. The layer containing none of the guidelines and only the sprites should be saved as Active\_Sheet, and once you have saved over Active\_Sheet, your sprites should be in the game if you view it in the Unity Editor.

****

**Please ensure that your artwork is the same resolution as the file you’re saving it over to avoid problems.**

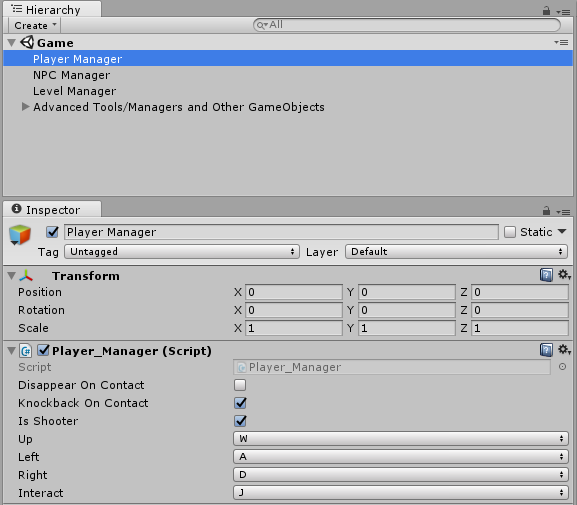
**Importing the Cutscene**

In this course, you will create a cutscene which plays before gameplay begins. To override the cutscene placeholder video, I suggest uploading your cutscene to YouTube and downloading it with a website, then saving the output from that website over Cutscene.mov. I have tested this method with y2mate.com, pickvideo.net, and keepvid.com, which all work for getting a file playable by Unity’s video player without error.

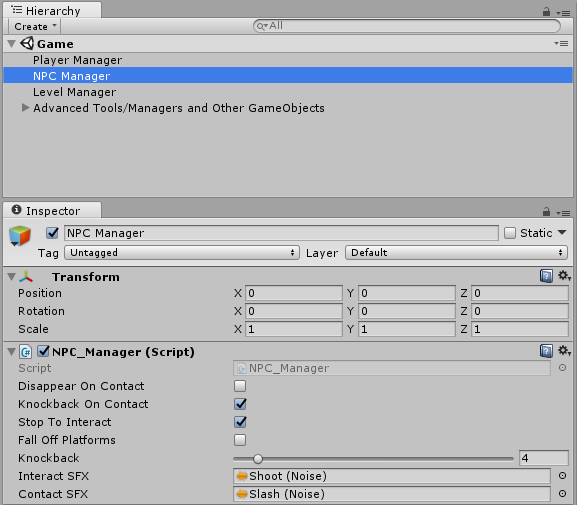
**Adding Personal Touches to the Game**

Basic

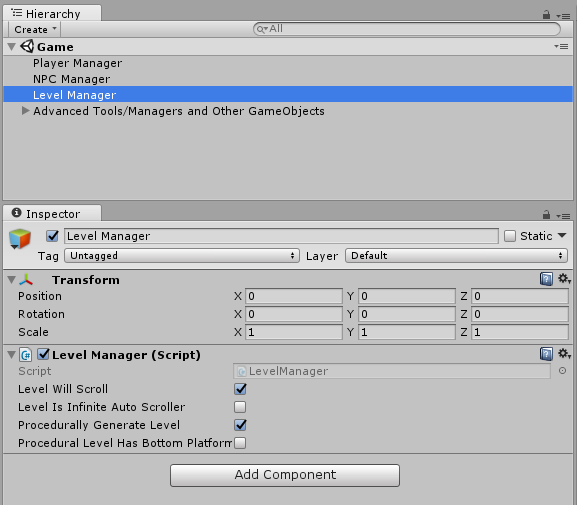
In the Hierarchy, clicking on the Player Manager, NPC Manager, or Level Manager will display in the Inspector variables which you may change. For example, selecting the **Player Manager** allows you to determine whether or not the player will disappear on contact with an NPC, if they will have knockback on contact with an NPC, their interaction style (shooter or melee-style interaction), and even the controls of the game.



The **NPC Manager** handles every NPC in the game, and you may control whether or not they disappear upon contact with the player’s interaction, whether or not to have knockback, if they stop to fire their projectiles, if they know how to avoid falling off platforms, and even adjust the amount of knockback they have.



The **Level Manager** determines what kind of level is played. If LevelWillScroll is checked, then the level will scroll instead of being a fixed camera. If LevelIsInfiniteAutoScroller, then this overrides all other variables in the manager and procedurally generates an infinitely scrolling level. If ProcedurallyGenerateLevel, then it erases the level architecture built by hand and the computer creates a level randomly. If ProceduralLevelHasBottomPlatform, then the player may not lose the game by falling off platforms because a platform at the bottom prevents them from doing so.



Advanced

Every object’s behavior in the game, from the player character to the level, is handled via **scripts**. Scripts are code files which dictate how these objects move, make sound, react to keyboard input, and more, and **you can modify them** if you want to add a personal touch to your copy of the game. Just head over to the Scripts folder in the Project tab and double-click on the script you want to change; I have provided detailed documentation on the code I have written. **If you make a mistake in programming and want to revert your project back to its original state, *backup your work* and download the original Unity project at the Github link.**