Before work on the project begins, there needs to be some groundwork laid so that we don't run around like chickens with our heads cut off, or step on anyone’s toes while trying to get the project done. Also, Unity is like a whole other language, and if someone working on the project doesn't know how to speak it, then production slows. If you have a question about one small thing, ask someone else that might know. If you have a big question on an entire topic (like how do I animate something), then go find a video or ask if someone can walk through an example with you. Unity has a lot of documentation and tutorials that will probably benefit you the most, and using Unity resources instead of having someone teach will keep efficiency up. These rules are subject to change. If you have suggestions or questions then tell me, because I'm not a dictator and this is not just MY project, it's all of yours too.

Tutorials:

https://unity3d.com/learn/tutorials

These are not basic coding tutorials, they are tutorials for how to use Unity. Follow the link, click on “engine”, and scroll down to the topics section. Some topics that I recommend to check out are Interface & Essentials, 2D Game Creation, and Animation. There is also a Scripting topic, which could be used for miscellaneous questions that come up when writing scripts for specialized functionality with a component.

In the Interface and Essentials topic, check out the USING THE UNITY INTERFACE and ESSENTIAL UNITY CONCEPTS sections. For the 2D Game Creation topic, watch the UNITY FOR 2D and 2D PHYSICS sections (I know 2D physics sounds boring but it’s actually kind of cool and probably not what you think). Under the Animation topic, look at the ANIMATING (except for video 4), CONTROLLING ANIMATION, and CHARACTER ANIMATION sections. Animating is kind of complicated so you might have to rewind the videos during some parts.

Game Object Documentation:

Every game object or other important piece of our game should have a text file of the same name, but with "\_log" added to the end. Some examples would be: a game object called "WorldMap" would have an accompanying text file called "WorldMap\_log", and another game object called "PlayerAvatar" would have an accompanying text file called "PlayerAvatar\_log". These \_log files will basically be a diary of all the changes made to that game object. Things that should be included in every log entry are: author's name, modification date, and things changed. The changes section should be generalized. For example, if I changed where the PlayerAvatar spawns on a certain level, I would put: "I changed where the player spawns on level 3". This lets other people stay updated easily by just looking at the logs, and if your change creates a bug that screws something else up, we know what changes to undo. If you go to work on a game object that doesn't have a \_log file, please add one and just say how far you got on the object. To create a new log file, just say: “Created game object with rigidbody2d and box collider components”. Log file entries don’t need to be long and bloated, but if there’s important information that people working on the file in the future should know, then you can add that too.

File Management:

We will be using github to move around files. Make sure that when you work on a file, that you have the latest version. You also need to upload any scripts or files that you create or modify. Game files on github should be in the same directory that they would normally be in the unity editor so they are easy to find. Github also has a file revision history that you can use if changes need to be reverted. Whenever reverting to an earlier version of a file, update the log file.

Communication:

For voice communication, I recommend discord because it is simple to use, easy to download, and free. We should have a general time frame during the day that is designated for working on the game. This doesn’t mean you have to work for the entire time, every day, or that you can only work during those hours. Think of it more like TA office hours: optional, helpful, and scheduled. During this designated time, there will be a discord voice channel where people working at the same time can collaborate. You can work outside the designated time but there is less chance that someone else will be on to help you.

Code Documentation/Style:

Each script should have a comment at the very top that describes the general functionality of the script. Please comment all code that is even remotely complex. If you code something that looks simple, like adding three numbers together, but someone might look at it and ask “why did they add these numbers together”, then just add a little comment. Comments should be at the end of the line of code, not above the line. Comments should explain why, not what you are doing.

Function lengths shouldn’t be longer than about 40 lines. If you have a function that is 41 lines, I don’t think anyone will care, but if you can, try to split up functions when possible. Try to aim for average function lengths of about 20 lines so that we can confine bugs to smaller portions of code, and so the functions will be easier to read. Simple switch statements and variable declarations don’t count when tallying up the function length.

Try to keep single word variables and function names in all lower case, like the variable “sum”, or the function name “insert”. When something is longer than one word, Capitalize all words after the first, like the variable “distanceMoved”, or the function name “findMin”. Common abbreviations like AM or PM can be in all caps. User-defined macros and constants should be in all caps, like “INFINITY”, or “TRUE”. If a counter variable is only used in a simple loop, use any variable name you like, but if a counter is used outside of a loop, or in a big loop, then try to give it a meaningful name, like “lineNumber”, or “numCollisions”. Classes should have every word capitalized, like the “DestroyPowerupObject” class.

Indents should be two spaces. Use blank lines to separate different sections of your code. Both starting and ending brackets for functions should be lined up with the first character of the function name (bottom left), and “if” statements or loops with only one line of code don’t need brackets (bottom right). For example:

void split() if(// condition)

{ // code

// code else

// code // code

}