Naming

* Descriptive, avoiding acronyms or abbreviations
* Classes
  + PascalCase for class names
  + Interfaces should begin with an “I”
  + Examples
    - LoadLevel
    - Player
    - ClassName
    - IClassInterface
* Functions and function parameters
  + camelCase for function names and parameters
  + Examples
    - getMonster()
    - shootWeapon()
    - pause()
    - parameterFunction(int parameterExample, bool isExample)
* Public Variables
  + camelCase, similar to function names
  + Examples
    - publicVariable
    - bossName
    - playerHealth
* Private Variables
  + Begin private variables with an underscore (\_), and then use camelCase for the rest of the name
  + Examples
    - \_privateVariable
    - \_playerSpeed
    - \_bossAnimator
* Static Variables
  + Begin static variable names with an “s”, then use camalCase
  + Examples
    - public static: sPlayerName
    - private static: s\_privateVariable
* Protected Variables
  + Begin protected variables with a “p\_”, then use camelCase for the rest of the name
  + Examples
    - protected float p\_protectedVariable
* File names
  + File names should be all lowercase with underscores (\_) separating each word
  + Examples
    - high\_score.txt
* Exceptions:
  + If a variable, function or class is predefined by Unity or another library, then these naming standards are not used
  + If a variable has a generally accepted use and name, then that use/name can be used
  + Example
    - Using ‘i’ as an iterator variable in a for-loop

Comments

* Comments are important to communicate ideas to other developers. Comments should focus more on *Why* a section of code exists and less on *What* the code does.
* Comments should help describe the relationships between functions, classes, and variables.
* If a comment is created on a new line, each new concept within the comment should be listed on a new line.
* If a comment exceeds one line, the new line should start with “ \* “ before the comment resumes
* Inline
  + If an inline comment is used the comment should be short and will be preceded by “//”.
  + Start comments by indenting at least once past the code in that line
  + If there are several inline comments in a row, make sure all comments in that block line up
  + Example:
    - private \_weaponProjectile; //Holds projectile info to be used by weapon
  + Example of several comments in a row
    - private \_foo; //Foo variable to give example
    - static private s\_fooBarBaz; //Variable to add more to the example
    - public testVariableOne; //Even more of an example comment variable
* Function Heading
  + Before a function, a comment should be used to describe a high level view of what the function does and how it interacts with other functions, variables, and classes.
  + Comment describes why the function was created
  + Function heading comments should start with “/\*” and end with “\*/”
  + Example
    - /\*This function is here to shoot the weapon
    - \* Provide more details about the function
    - \* Make sure to explain why the function is being used\*/
    - shootWeapon()
* File Heading
  + At the beginning of each script file there should be a heading that explains what the file is and why it is there.
  + Follows the same rules as function header comments
  + The end of each line in the File Header comment should end with “\*”
    - All the right-hand asterisks (\*) should line up
  + Example
    - /\*
    - \* This is an example comment to show a file header \*
    - \* The purpose of this explanation is to ensure consistency within our code \*

Error Handling

* In general use try-catch error handling, but use if-else if it is more appropriate

Layout

* Indentation helps code appear more organized and improves readability
* Each indentation should be 4 spaces
* The following apply to all Classes, Functions, Conditional statements, Switch statements, and Iterating statements.
  + The opening brace ( { ) should be placed on the next line down from the class name with no additional indentation.
  + The closing brace ( } ) should be placed on the line after the end of the class definition with no additional indentation.
  + All content within a class should appear between the opening and closing braces, and nothing should be placed on the same line as a curly brace
  + Example
    - Statement
    - {
    - /\*Place all Statement Content Here\*/
    - }
* Classes
  + For consistency, elements should be placed in a class in the following order:
    - Private data members, arranged in the order they will appear followed by an empty line
    - Public data members, arranged in the order they appear followed by an empty line
    - The void Start() method (if used)
    - The Update() method (if used)
    - Any Unity-specific methods like OnCollisionEnter2D()
    - Other custom-made functions
  + Examples
    - public class ExampleClassIndentation : Examples
    - {
    - private int \_privateNumber1;
    - private int \_privateNumber2;
    - private bool \_privateBool;
    - public int publicNumber1;
    - public bool publicBool;
    - void Start()
    - {
    - /\*Starting Code\*/
    - }
    - void Update()
    - {
    - /\*Updating Code\*/
    - }
    - void OnCollisionEnter2D(Collision2D collision)
    - {
    - /\*Code to run on collision\*/
    - }
    - void customFunction()
    - {
    - /\*customFunction() code\*/
    - }
    - }
* Functions
  + The function parameters (if any) should be separated by a comma and a space
  + Functions should start with any variable definitions that need to take place followed by an empty line
  + Example
    - int findHypotenuse(int side1, int side2)
    - {
    - double hypotenuse;
    - hypotenuse = Math.sqrt((side1^2) + (side2^2));
    - return hypotenuse;
    - }
* If-statements
  + Example
    - if ((number1 <= number2) && (number1 > 0))
    - {
    - /\*Execute First Condition\*/
    - }
    - else if (number1 > number2)
    - {
    - /\*Execute Second Condition\*/
    - }
    - else
    - {
    - /\*Execute else statement\*/
    - }
* Switch Statements
  + Example
    - switch (switchVariable)
    - {
    - case 1:
    - /\*case 1 code\*/
    - break;
    - case 2:
    - /\*case 2 code\*/
    - break;
    - case 3:
    - /\*case 3 code\*/
    - break;
    - default:
    - /\*default code\*/
    - break;
    - }
* Iterating Statements
  + Example
    - while (True)
    - {
    - /\*While-loop code\*/
    - }
  + Example
    - for (int i = 0; i <3; i++)
    - {
    - /\*For-Loop Code\*/
    - }