Document Overview

This document contains the programming style guide for the Art Generator Project for CSCI 2340: Software Engineering, Spring 2024. This file will be broken into multiple sections, one for each programming language used in the project. There may be additional files mentioned in each subsection that shows a template file of implementation in each programming language. For coding style relating to error logging, refer to the Error Logging Guide.

Python Programming Standard

This section details the required programming style that should be adhered to for any Python programming in the Art Generator Project. There is a reference template file in the templates folder that demonstrates the guidelines mentioned in this section in practice.

Variables

This section overviews variable naming for all Python code for the Art Generator Project. As a general rule, all variables must be descriptive. At the end of this section is a table that will demonstrate how each of the variable types should be implemented.

Classes

Class names should have two parts. The first part is an all-capitals abbreviation of the code area in which the class exists. The second part is in Pascal Case with no spaces describing the specific class. These two components are then connected via an underscore. This rule applies for all types of classes including enumerations. A table of abbreviations can be found below:

Code Area	**Abbreviation**
Art Generation Driver	AGD
Back-End Command Interface	BCI
Common	CMN
Database	DB

Table 1: Code Area Abbreviations for Class Naming

Class Members

Class members should be in camel case followed by an underscore with no spaces.

Constants

Constants should be defined using capital letters with words separated by underscores.

All known constants that should be predefined by specific value or at start-up should be defined in classes with either Enum, IntEnum, or StrEnum type, depending on what their type of constant is. Constants should be grouped together by their purpose. All constants should be defined in a definitions Python file that is specific to the code area in which the constants are used.

Global Class Members

Global class members should be all lowercase, separated by underscores with a trailing underscore.

Global Variables

Global variables should be in Pascal Case with no spaces and no trailing underscore.

Local Variables

Local variables, whether in a function, class, or file, should be Camel Case with no spacing and no trailing underscore.

Variable Type	Example Implementation
Class	ABV_ClassName
Class Member	exampleClassMember_
Constant	EXAMPLE_CONSTANT
Global Class Member	example_global_class_member_
Global Variable	ExampleGlobalVariable
Local Variable	exampelLocalVariable

Table 2: Variable Naming Examples

Functions and Methods

This section overviews function and class method naming for all Python code for the Art Generator Project. Note that for some of the Touch Designer built in functionality, naming conventions are unable to be changed, leading to some potential violations of these guidelines. That is acceptable.

Class Methods

All class methods should be Camel Case with no spaces and no trailing underscore. There is one exception to this, which is the initializer function, which is defaulted to __init__ in Python. This is acceptable.

Enum Methods

Enumerations in Python are technically a class object, however, to ensure that there is a difference in an enumeration class and a normal class, enumeration methods have a different naming convention. The proper naming convention for an enumeration method is lower-case separated by underscores with no trailing underscore.

Functions

All functions should be written in Pascal Case with no spaces or underscores. Note that Touch Designer built-in functions will not align with this standard, and that is okay only for Touch Designer functions.

Code Area	Abbreviation
Class Method	exampleClassMethod()
Enum Method	example_enum_method()
Function	ExampleFunction()

Table 3: Function and Method Naming Convention Examples

Comments

This section gives an overview of commenting guidelines for all Python code for the Art Generator Project. Since some of the comment blocks can be large, it is important to refer to the template Python file to view actual implementation of the comments.

General Comments

In general, a few quick words throughout the code base to describe what the code is doing or a thought process behind a certain design is natural and helps other developers understand the software easier. It is acceptable to leave general comments through out the code. Ensure that all comments left through the code are concise and clear. In general, there should not be more comments in a function or method than lines of code.

File Header Comments

Each file should contain a header that clearly lists the name of the file, the purpose or description of the file, the creation date of the file, and the author of the file. See the Python template file for implementation.

Function and Method Comments

Each function or method should contain a block comment above its definition containing the name, the purpose, the inputs, and the outputs of, as well as the requirements satisfied by the function or method. See the Python template file for implementation.

Enum Comments

Each enumeration class should contain a block comment above its definition including the name, the enumeration type, the description, and the values of the enumeration. When listing the values of the enumeration, give brief descriptions of each member so their purpose is understood. See the Python template file for implementation.

Requirement Comments

Wherever a requirement is satisfied in the code base, it is important to directly above the line of code leave a comment simply saying which requirement has been satisfied. This makes the process of finding where certain features are implemented in the code base easier for all developers. See the Python template file for implementation.

Section Comments

Each section of code should be separated by a standard comment showing the area for public modules, project modules, function definitions, class definitions, local variable definitions, etc. See the Python template file for implementation. If a certain section of code does not exist in the code, for example public modules, then that comment can be omitted.

Other

Use of Constants

Constants should be defined for all known values that are to be used in the Art Generator Project. All hardcoded integer or path values should be removed and replaced with a constant with the proper value. This should help the code be more readable, and reduce the chance of introducing bugs.

Indentations

All indentations should be set to four spaces. This can be done either by manually entering four spaces, or by adjusting the tab width to four spaces. This applies for any indentation for functions, loops, conditionals, etc.

Termination of Lines

All code should be terminated with a semi-colon. This includes any assignments, function calls, or return statements. This does not include importing, conditional statements, loop statements, or class definitions. Follow C / C++ syntax on how to approach using semi-colons to terminate lines.

Importing External Modules

When importing modules into Python source files, only the required modules should be imported into the code base to prevent adding unnecessary amounts of characters on the screen. For example, if importing ceil() from math, import as [from math import ceil]. Do not import the entire math library and call [math.ceil()].

Importing Project Modules

Contrary to the previous section, the project files are set up in a way that does not allow for specific components from the project modules to be loaded in. In this case, it is important to load in module objects like this [from <file name> import <object> as <abbreviation>]. This type of importing reduces the overall amount of text required when actually programming the system. Note that when using a class

within the same file that it is defined in, the full name must be used instead of an abbreviation. This is acceptable. A table below is a reference to the official abbreviations that should be used when importing project modules:

File	Module	Abbreviation
AGD_Definitions	AGD_LengthUnits	AGD_LU
AGD_Definitions	AGD_RecordingParameters	AGD_RP
AGD_Definitions	AGD_TouchDesignerNodes	AGD_TDN
AGD_Definitions	AGD_TouchDesignerInstance	AGD_TDI
AGD_Definitions	AGD_Directories	AGD_DIR
CMN_ErrorLogging	CMN_LoggingLevels	CMN_LL
CMN_ErrorLogging	CMN_Logging	CMN_LOG

Table 4: Sample Abbreviations for Importing Project Modules

Using Classes vs Functions

There are instances in the software design process where it makes sense to use helper functions that do not need to belong to a class. A good example of this is a function that is used in a Touch Designer Callback. In this case, Touch Designer requires that these functions be accessible through their API, so it is OK to not include these functions in a Python Class. It is best practice to utilize an Object-Oriented approach while designing this software; however, in instances like mentioned above, it is OK to not utilize a class based structure when designing software.