Coding Standard

for

Pittsburgh Train Automation System

Version 0.1

Prepared by Alexander Mcmoil, Cameron Kirsch, Kyle Kessler, Justin Pacella, Yuheng Lin, and Yun Dong

Aurora, Inc.

9/26/2023

1 Naming Convention

1.1 Class

The Aurora team will name classes using camel-case with a capitalized first letter. The names themselves will use complete words to be perfectly clear to all Aurora team members.

- `class TrainController(object): pass`

1.2 Properties or Attributes

We will use snake-case for naming class properties. The names do not necessarily need complete words so long as the meaning is clear. Private properties will be preceded by two underscores and protected properties by one.

- `self.num_passengers: int`

1.3 Methods

We will name class methods using snake-case. Each method name must contain at least one verb to make the method's purpose explicitly clear.

- `self.engage_emergency_brake()`

1.4 Variables

We will name variables the same way we name class properties with a few exceptions:

- We may use `i`, `j`, or `k` to denote loop indices
  - `for i in range(1,10): pass`
- Any named Boolean variable outside of a class must be prefixed with "`is_`" as if the name were a true/false question
  - `is_door_open: bool`
- We will add a plural suffix to all container names
  - `trains: dict(int,TrainModel)`

1.5 Constants

We will name constants using only uppercase letters and underscores.

- `PASSENGER_CAPACITY: int`

1.6 Comments and Comment Style

The Aurora team will write comments preceding each class, method, and function. Each of these will contain sections for:

- @brief – brief description
- @params – defines each function argument
- @returns – defines the value(s) returned
- @raises – lists any exceptions the function may raise

Aside from class/method descriptor functions, comments will be kept relatively sparse. We will only use extra comments to summarize substantially large sections of code whose purposes may be unclear.

2 Programming Conventions

2.1 Indentation and Brackets

The Aurora team will universally use a four-space indent. Since Python does not employ brackets to separate programming scopes, we will not use them. For the hardware portion of the project, we will use C++. Opening brackets will be placed on the same line as their decorators and closing brackets after the last line of code in the scope.

```
int set_switch(int: switch_id) {
      /* code */
}
```

2.2 Exception Handling

The Aurora team will not employ Python's assert statements outside of testbenches for our software. Instead, we will use if-raise statements to trigger exceptions. To catch errors, we will use Python's try-except-else-finally blocks. Each except block will catch one specific error and handle it according to its severity.

```
try:
      if not track.get_stuff():
raise TrackFault("A track fault has occurred")
except TrackFault as error:
log.print(f"Caught track fault: {error}")
except TypeError as error:
log.print(f"{error}")
raise error
else:
pass
finally:
pass
```