Test Plan

Team Name: Cloud Nine
Team Members: Jeremy Maas, Matt Burton, McHale Trotter, Kevin Sampson, Justin Chen, Ryan Hirscher

Before merging any changes into the main branch, new code must be tested to prevent the introduction of bugs. Cloud Nine's test plan includes three parts:

1. Unit testing
2. Module testing
3. Integration testing

Who Conducts Tests?

As the author, it is expected that you have tested your code with at a basic level before passing it to the formal testers. Formal tests must be **conducted by members of the team that did not author the code**.

Where are Tests Stored?

/documents/tests

Test scripts and data are stored in the *\/documents/tests* directory. Each module has its own sub-directory within the */tests* directory.

documents/tests/module

Testers must create a new directory **named after the module** to store all of the test scripts and data

Scripts

Test scripts should be stored in the */documents/tests/module* directory.

documents/tests/module/inputs

If a module needs external input, a sub directory named */inputs* can be used to contain a set of input files, each file containing one test.

When applicable, test sets should include all of the following:

1. An example from each equivalence case
2. Additional tests around the boundaries of each equivalence case range
3. An input of an incorrect data type

documents/tests/module/outputs

If testing requires an input set, a sub-directory named */outputs* must contain the set of expected values from each input test.

Unit Tests

Unit tests ensure that each individual method within a module works as expected.

Develop tests that exercise each module and ensure the methods behave as defined in the functional specification document.

If each method works as intended, continue on to module testing. If there is an issue with one or more methods, report the issue(s) to the author and track the issue(s) in the defect log.

Module Tests

Module tests are isolated tests where new code is tested independent of its connections to other classes/modules.

Now exercise the entire module, not just individual methods.

If the entire module behaves as described by the functional specification document, move on to integration testing. If there is an issue with the module as a whole, notify the author and track the issue in the defect log.

Integration Tests

Finally, test the entire project and esnure that it behaves in the manner outlined by the requirements document. If the project still performs as intended, the module is ready to be merged into the main branch. If an issue is found upon integrating the module into the project, report the issue to the author and track the bug in the defect log.

Finding a Balance

Cloud Nine's test plan is designed to reduce the overall number of bugs entered into our main codebase while maximizing the amount of time we have to develop new features. We recognize the benefit of having more team members test each module, but we also acknowledge our limited development time and as such have decided to keep the number of testers to a minimum. Generally, we can expect that all code that is ready to be tested will undergo inspection through GitHub's pull request system, and ideally they will go through the testing listed above. This level of testing is not necessarily cost effective for simple modules, however we will make sure to exercise this plan for more complicated and frequently used systems as they are developed.

With more time, or for a more performance critical project we would be sure to exercise black box testing, glass box testing, and inspections by a multitude of team members as we recognize the value those strategies provide when tryinng to reduce bugs. However, with time as another pressing factor our testing may be less robust than that of a larger software company.

Tests

The Airport Class

Must ensure that adding an entry creates an object with the passed values. Validate that the methods set and return lists or values that are accurate. documents/tests/airport/test_airport.py

```
python3 test_airport.py
```

Function	Input	Assertion
test_get_airport_id()	1	1
test_get_airport_name()	"John F. Kennedy International Airport"	"John F. Kennedy International Airport"
test_get_airport_abbreviation()	"JFK"	"JFK"
test_get_airport_latitude()	40.641766	40.641766
test_get_airport_longitude()	-73.780968	-73.780968
test_get_airport_timezone_offset()	-5	-5
test_get_metro_population()	19034000	19034000
test_get_total_gates()	5	5
test_get_available_gates()	3	3
test_get_is_hub()	1	1
test_remove_gate()	Call remove_gate()	2
test_add_gate()	Call add_gate()	4

The Database

Test database connector and validate a connection to the database. Ensure that basic queries like select, delete, update, and insert are possible and validated. documents/tests/database/test_database.py

```
python3 test_database.py
```

Function	Input	Assertion
test_connect(mock_connect)	mock_connect	mock_connect.assert_called_once()
test_disconnect()	db.disconnect()	assertIsNone(connection)
test_execute_query(mock_connect)	SELECT * FROM airports	assertIsNotNone(result)
test_execute_query_to_dataframe(mock_connect)	SELECT * FROM airports	assertListEqual(list(df.columns), [id, name])
test_execute_insert_update_delete_query(mock_connect)	INSERT mock data, UPDATE mock data, DELETE mock data	execute insert_query, update_query, delete_query

The Great Circle

Test different distance airports which are short distance and long distance and ensure that the function returns expected values. Ensure that the same airport returns 0. documents/tests/great-circle/test_great_circle.py

```
python3 test_great_circle.py
```

Function	Input	Assertion
test_same_airport()	Call great_circle(airportSF, airportSF)	0
test_far_away_airports()	Call great_circle(airportSF, airportNY)	2572.08

Turn Around Time

Assert that the turn around time values are correct. documents/tests/turn-around-time/test_turn_around_time.py

```
python3 turn_around_time.py
```

Function	Input	Assertion
testfuncRefuelCorrect()	50	True
testfuncNoRefuleCorrect()	40	False
testfuncRefuelIncorrect()	100	True

Incomplete Tests

The Flight Class

Must ensure that adding an entry creates an object with the passed values. Validate that the methods set and return lists or values that are accurate. documents/tests/aircraft/test_flight.py

```
python3 test_flight.py
```

The Aircraft Class

Must ensure that adding an entry creates an object with the passed values. Validate that the methods set and return lists or values that are accurate. documents/tests/aircraft/test_aircraft.py

```
python3 test_aircraft.py
```

The Aircraft Instantiation

Must ensure that the database conneciton is made. Must ensure that the select statement from the database returns as expected so that we can instantiate the object list of aircrafts. documents/tests/aircraft/test_aircraft_objects.py

```
python3 test_aircraft_objects.py
```

The Airport Instantiation

Must ensure that the database conneciton is made. Must ensure that the select statement from the database returns as expected so that we can instantiate the object list of airports. documents/tests/airports/test_airport_objects.py

```
python3 test_airport_objects.py
```

The Timetable Generation

Must ensure that all aircrafts are placed at airports as expected including reserve aircrafts, paris aircraft, and the remaining. Must ensure that the algorithm does not generate a timetable that has aircrafts landing late by design. Must ensure that gates are reserved as intended and the count of reserved gates is true. Must ensure that a flight path does not include previous airports unless it is the last leg flying home. Must ensure that only valid airports are chosen in an flight path that exist within the airports list. Must fly only valid aircrafts in the aircrafts list. Must ensure that each flight path includes at least one hub. documents/tests/generate_timetable/test_generate_timetable.py

```
python3 test_generate_timetable.py
```

The View Timetable

Must ensure that viewing the timetable returns the correct select statement of the database flights table. Must ensure that the headers are printed and match the database attributes. documents/tests/timetable/test_timetable.py

```
python3 test_timetable.py
```

Aircraft Menu

Must ensure that options in the menu behave as expected and returns the correct information including: selecting from the database aircrafts table with view_aircraft(), add or remove an aircraft with edit_aircraft(), adding aircraft is inserted into the database with add_aircraft(), removing an aircraft is deleted from the datbase with remove_aircraft() as long as the aircraft is not assigned to a flight. Get valid input must be applied to each of the functions and tested invidividually. documents/tests/menu/test_aircraft_menu.py

```
python3 test_aircraft_menu.py
```

Airport Menu

Must ensure that options in the menu behave as expected and returns the correct information including: selecting from the database airports table with view_airports(), add or remove an airport with edit_airport(), adding airport is inserted into the database with add_aircraft() and the appropriate gates are assigned to it, removing an airport is deleted from the datbase with remove_airport() as long as the airport is not receiving or sending flights. Get valid input must be applied to each of the functions and tested individually. documents/tests/menu/test_airport_menu.py

```
python3 test_airport_menu.py
```

Configuration Menu

Must ensure that configurations are set as intended according to a JSON formatted document. Ensure that configuration start and duration are set for the intended time. Ensure that print functions are formatted as intended and print the correct values. documents/tests/menu/test_configuration_menu.py

```
python3 test_configuration_menu.py
```

Cost Menu

Must ensure that configurations are set as intended according to a JSON formatted document. Ensure that fuel costs, takeoff costs, landing costs, and leasing costs are set for the intended value and handled appropriately for input. Ensure that the datbase connection is made and the intended leasing costs are updated in the database with the intended values. Ensure that print functions are formatted as intended and print the correct values. documents/tests/menu/test_configuration_menu.py

```
python3 test_cost_menu.py
```

Timetable Menu

Must ensure that output is formatted as intended for viewing timetable options. Including search_routes(), view_timetable(), edit_timetable(), and submenus associated with each. documents/tests/menu/test_timetable_menu.py

```
python3 test_timetable_menu.py
```

Simulation Menu

Must ensure that configurations are received as intended. Ensure that configuration options are printed in correct format. Ensure that display menu is called correclty with new menu options. documents/tests/menu/test_simulation_menu.py

```
python3 test_simulation_menu.py
```

The Simulation

Ensure that the events can be rescheduled. Ensure taht the output is formatted as intended. documents/tests/menu/test_simulation.py

```
python3 test_simulation.py
```

The Report

Must ensure that time values funciton as intended and are shown appropriately in the report. Must ensure that the report is formatted as indended as is downloadable. documents/tests/report/test_report.py

```
python3 test_report.py
```

The Schedule

Must ensure that clear schedule empty's the list. Must be able to ensure that add_event can avoid event conflicts and add to the timeline if available, show that this can behave as intended. documents/tests/schedule/test_schedule.py

```
python3 test_schedule.py
```

The Scheduled Event

Must ensure that departure events and arrival events are used as intended and are accurate according to the database. Ensure that the retrieved attriutes from the database are accurate. documents/tests/schedule/test_scheduled_event.py

```
python3 test_scheduled_event.py
```

Clock

Ensure that get_time() converts minutes input to days, hours, and remaining minutes. Ensure that print_time() output is formatted as intended. Ensure that get_flight_time never exceeds 24 hours and only manages hours and minutes. documents/tests/utilities/test_clock.py

```
python3 test_clock.py
```

Display Menu

Must ensure that menu options are available as intended. Ensure that input is validated and shown options are the only options available. documents/tests/display/test_display_menu.py

```
python3 test_display_menu.py
```

Flight Angle

Must ensure that wind speed is accurate calculated and the percentage is accurately calculated and returned as intended. documents/tests/utilities/test_flight_angle.py

```
python3 test_flight_angle.py
```

Flight Demand

Ensure that the flight demand for an airport is accurate calculated accurately based on metro population. documents/tests/utilities/test_flight_demand.py

```
python3 test_flight_demand.py
```

Flight Duration

Ensure that the flight duration of a flight is accurately calculated and returned. Ensure that if an aircraft needs to be refueled for total flight duration, the time will be adjusted. documents/tests/utilities/test_flight_duration.py

```
python3 test_flight_duration.py
```

Flight Takeoff

Ensure that the takeoff to cruising altitude and descent to destination airport of a flight is accurately calculated and returned as intended. documents/tests/utilities/test_flight_takeoff_.py

```
python3 test_flight_takeoff.py
```

Comfort Airlines

Ensure that the menu options are displayed will always allow the user to choose an available option as intended. documents/tests/comfort_airlines/test_comfort_airlines.py

```
python3 test_comfort_airlines.py
```