1. Input Space Partitioning

a) Any other parameters

Other parameters:

1. File or data received from the serverURL - weatherData, as the AI analyse depends on the returned object to process.
2. The to-do list, as the AI use the taskID to access the item information from the to-do list.

b) Input Space Partitioning process

functions: assessWeatherImpact();

parameters: taskID, city, serverURL, to-doList, weatherData(assume it's a number)

Some possible characteristics:

taskID:

- TaskID is null – is null and not null
- If not null it must have a Length - length = 0, length > 0

City:

- City is null – is null and not null
- If not null it must have a Length - length = 0, length = 1, length > 1
- If length > 1 is the city valid – valid city, not valid city

serverURL

- Is URL: true , false
- Is the server reachable- true, false

to-doList

- Array empty
- Number of occurrences of taskID in list: occurrences=0, occurrences=1, occurrences>1

weatherData

- Is a number – true, false?
- If is a number, Is the weatherData negative, zero, or positive.

Test case 1

Test description: The taskID is not listed in the to-do list

Test setup:

- to-do list should be set up before the test

Inputs:

- taskID_# not in the to-do list
- Actual City name
- The weather server URL
- E.g.(taskID_1234, perth, www.weather.com)

Expected result:

- The system should display an error message and return -1, the error message that tell the client taskID_# not found.

Assume there's a sub system used to valid the city name.

Test case 2

Test description: City is unrecognized

Test setup:

- to-do list should be set up before the test
- weather server should be ready.
- Both main system and subsystem should be in ready state.

Inputs:

- taskID_# not in the to-do list
- Random string not a city name
- The weather server URL

Expected result:

- The sub system should return false after validating, and the print the function error message and return -1 as error. the error message that tell the client city not found.

c) Test doubles

Yes, we need test double in those two test cases:

1. This function requires The URL of a web-server providing weather updates, which as it requires network access, but in test situation, access a services over the internet may be slow or not even possible. So not applicable.
2. The function require a to-doList that contain information about a task, in which we do not have any task while we testing.

For URL request from weather server, we fake a weather server or class while we are testing. The idea is the fake server can be implemented locally, so it will not engage with internet, but will store predefined data, and return data in same format as the actual weather server.

For to-doList use stub, so it can return canned answers during each test.

2. Logic-based testing

a)

Let w(x) denote weather is wet

Let C(x) denote Conditions is x

Let O(x) denote x is Outdoor tasks

Flag() be a function takes an input and flag it need a weather impact assessment

Then an appropriate predicate would be (assuming t is some task):

(W(wet) v C(extreme)) ^ O(t) -> flag(t)

b) Make each clause in the predicate active

To make a clause active, we need to assign values to variables in a predicate such that the truth-value of the whole predicate depends on the clause. To make each cluse active we need to make sure the test cover Active Clause Coverage, which need to make each cluse in each predicate active.

To make W(wet) to be active we need to set:

C(extreme) = false, O(t)=true, flag(t)=false.

So (W(wet) ^ true) ^ true -> false

When W(wet) is true

- (true v false) ^ true -> false
- true ^ true -> false
- true -> false
- false

When W(wet) is false

- (false v false) ^ true -> false
- false ^ true -> false
- false -> false
- true

3. Syntax-based testing

Assume URL should not contain subpages.

BNF:

<invocation> ::= "agendum " (<invokes>) | "agendum " (<filename> " " <url>)

<invokes> ::= "--help" | "--version"

<url> ::= <http> <domian> "." <domianExtension>

<filename> ::= <fileChars> | <fileChars> <filename>

<domian> ::= <domianChars> | <domianChars> <domian>

<domianChars> ::= <letter> | <digit> | <urlsymnol>

<domianExtension> ::= <letterLower> | <letterLower> <domianExtension>

<fileChars> ::= <letter> | <digit> | <filesymbol>

<letter> ::= <letterLower> | <letterUpper>

<http> ::= "https://" | "http://"

<letterLower> ::= [a-z]

<letterUpper> ::= [A-Z]

<digit> ::= [0-9]

<filesymbol> ::= "!" | "#" | "$" | "%" | "&" | "(" | ")" | "+" | "," | "-" | "." | ";" | "=" | "@" | "[" | "]" | "^" | "_" | "`" | "{" | "}" | "~ " | "-"

<urlsymnol> ::= "@" | ":" | "%" | "." | "_" | "/" | "+" | "~" | "#" | "="

b) How many tests

Terminals:

urlsymnol=10, Filesymbol=23, digit=9, letterUpper=26, letterLower=26, http=2, invokes=2, agendum=1

10+23+9+26+26+2+2+1=99

Production:

1. <invocation> ::= "agendum " (<invokes>)
2. <invocation> ::= "agendum " (<filename> " " <url>)
3. <url> ::= <http> <domian> "." <domianExtension>
4. <filename> ::= <fileChars>
5. <filename> ::= <fileChars> <filename>
6. <domian> ::= <domianChars>
7. <domian> ::= <domianChars> <domian>

8. <domianChars> ::= <letter>
9. <domianChars> ::= <digit>
10. <domianChars> ::= <urlsymnol>
11. <domianExtension> ::= <letterLower>
12. <domianExtension> ::= <letterLower> <domianExtension>
13. <fileChars> ::= <letter>
14. <fileChars> ::= <digit>
15. <fileChars> ::= <filesymbol>
16. <letter> ::= <letterLower>
17. <letter> ::= <letterUpper>

Additional 17 tests

Total 17 + 99 = 10=116

c) Test case <invocation> ::= "agendum " (<invokes>)

Assume the other agendum uninstalled in the system and before the test the current version of the agendum installed before the test (i.e., agendum-1.5)

Test description: test the agendum should display help and version information inside the terminal.

Test value: the string - agendum --help and agendum --version

Expected result:

agendum -- help

- Should display a predefined manual in the terminal.

agendum –version

- Should display the current version of the agendum in the terminal – version: agendum-1.5.