**Question 1**

Sprite Class Test Plan:

* Objectives for the class
  + The Sprite class is designed to represent any spawn-able object within the game from the player character to items.
  + The class must be usable as a base building block to all in-game objects.
  + The class is defined to provide simple functionality when working with objects.
  + The class must include private fields that keep track on the object’s location within the game world. Often specified in form of a 2D coordinate system.
  + The class must be able to track the associated bitmap used to visually represent the object on the user display.
  + The class must provide simple movement functionality so that each item can be moved within the game.
  + The class must also provide collision detection functionality so that two items cannot be overlapping in-game.
* Any requirements for inspection/review of the class.
  + As a base layer for the game, the Sprite class must be robust, reliable, and maintainable.
  + In terms of robustness requirements, the class must show that it can maintain its base functionalities within unanticipated conditions. This is quite crucial as games are quite complex with lots of moving factors, often leading to unknown conditions.
  + In terms of reliability, the class must show high tolerance when encountering faults through rigorous exception handling. The class must also be able to recover from illegal interactions such as in-game item collisions as seamlessly as possible.
  + Finally, in terms of maintainability, the Sprite class and its test plan must be documented in full detail, in addition to taking down any specific notes or instructions that would help programmers in future. The class must be easily customizable and accommodate for further modification to the Sprite by structuring the code in a logical and adequate manner.
* Brief statement of objectives for specification-based testing for the class.
  + In specification-based testing, the software is tested against the requirements given by stakeholders. It is used in a black-box manner where the testing does not know anything about what is inside the program.
  + For the Sprite class, the testing must use the prototype interface to construct and instantiate a Sprite object. The objective is then to test the given specification such as object location, object movement, bitmap association and collision detection.
* Brief statement of objectives for implementation-based testing for the class.
  + In implementation-based testing, the primary focus includes the technical design decisions taken to translate the business requirements to working software. Despite Specification based testing, implementation-based testing is often used in a white box testing where the approach of the programmer to writing code is tested.
  + For the Sprite class, the tester would need to analyze the coding approach used by the programmer to define unit tests corresponding to functions within Sprite methods. These unit tests can expand to develop integration tests with combined components.
* Brief statement of objectives for interaction-based testing (i.e., interactions between methods of the class)
  + Interaction-based testing is quite crucial in developing games as tens of instantiated objects maybe interacting with each other at any given time through different relations. Often done through grey-box testing, we can use partial information regarding the implementation in testing interactions and observing the results.
  + For the Sprite class, the objective is to use various methods such as movement in specific timing so that two items may undergo an interaction such as a collision. With different type of items, the unique interactions can quickly grow however using knowledge from implementation we can reduce this number.

**Question 3**

Given Diesel is a Low Volatility and Propane a High Volatility fuel, the test cases are as follow:

Test 1: safe fuel load test

* Description: This test is used to demonstrate the fuel load functionality where loading is allowed as the load amount is equal to or less than the “safe” tank capacity.
* Input: Fuel = Diesel, Load = 1200
* Pre-condition: Tank Empty
* Testing condition: if FuelType == LowVolatility then Load <= 1200
* Expected Output: Tank is successfully filled
* Expected Post-Condition: Tank Full

Test 2: unsafe fuel load test

* Description: This test is used to demonstrate the fuel load functionality where loading is not allowed as the load amount is more than the “safe” tank capacity.
* Input: Fuel = Propane, Load = 1200
* Pre-condition: Tank Empty.
* Testing condition: if FuelType == HighVolatility then Load <= 800
* Expected Output: Load is more than safe capacity
* Expected Post-Condition: Tank Empty

Test 3: list tank capacity without expansion space test

* Description: This test is used to list the tank capacities for Low Volatility fuel type.
* Input: Fuel = Diesel
* Testing condition: FuelType == LowVolatility
* Expected Output: Safe capacity is 1200

Test 4: list tank capacity with expansion space test

* Description: This test is used to list the tank capacities for High Volatility fuel type
* Input: Fuel = Propane
* Testing condition: FuelType == HighVolatility
* Expected Output: Safe capacity is 800

Test 5: illegal data string input

* Description: This test is to demonstrate the software’s capabilities in handling exceptions, specifically with incorrect fuel name or type.
* Input: Fuel = Di3sel
* Testing condition: Di3sel == Low Volatility || Di3sel == High Volatility
* Expected Output: Exception caught. Incorrect fuel entry, Di3sel is not classified as fuel.

Test 6: illegal data number input

* Description: This test is to demonstrate the software’s capabilities in handling exceptions, specifically with incorrect fuel tank capacity or load.
* Input: Fuel = Diesel, Load = 13oo
* Testing condition: if FuelType == Low Volatility then Load <= 800
* Expected Output: Exception caught. Incorrect fuel load entry, 13oo is not a valid number.

Test 7: missing field (exit when done)

* Description: This test is used to demonstrate program’s ability to prevent fuel discharge before proper set up. This method can also be used to show to assert the automatic exit requirement, as the user cannot initiate a fueling sequence with the previous configured fuel.
* Input: Load = 800
* Testing condition: FuelType == High Volatility || FuelType == LowVolatility
* Expected Output: Exception caught. Fuel not specified, please specify the fuel name.