Assignment 2: Godot Engine

Student: av222vu – Course: 1DV609

Project Description:
Godot is an open-source game engine used to create 2D and 3D games. It aims to provide a user-friendly platform for game developers by offering a flexible and extensible architecture. Godot includes tools for scripting, rendering, physics, and animation, supporting a wide range of platforms (Windows, macOS, Linux, and consoles).

Purpose and Goals:
Godot's purpose is to empower developers to build games without licensing fees or proprietary constraints. The engine promotes collaboration and continuous improvement through community contributions. Its goals include providing a lightweight yet powerful engine that is customizable and easily accessible.

Overview of Requirements and Specifications:

- **Core Functionalities:** Scene management, scripting (GDScript, C#, C++), and rendering.
- **Platforms Supported:** Windows, macOS, Linux, Android, iOS, Web.
- **Key Features:** Node system for modular design, physics engine, visual scripting, and animation tools.
- **Specifications:** Written primarily in C++ with optional scripting in GDScript and other languages. Modular architecture with editor customization.

Stakeholders, Risks, and Evaluation:

- **Stakeholders:** Game developers, indie studios but also large studios, educational institutions.
- **Risks:** Compatibility issues, performance bottlenecks, and bugs affecting game development.
- **Evaluation:** Continuous feedback from the community, regular bug tracking, and contribution to GitHub repositories.

Past, Present, and Future Development:

- **Past:** Originating in 2007 by Juan Linietsky and Ariel Manzur, Godot was publicly released as open-source in 2014.
- **Present:** Ongoing development with significant contributions from the community. Godot 4.x includes Vulkan rendering and major performance improvements.
- **Future:** Focus on enhancing 3D capabilities, implementing more advanced AI systems, and supporting modern rendering techniques (ray tracing, etc.).

Current Testing Strategy:
Godot employs unit testing, integration testing, and regression testing. The primary framework used is **doctest**, which allows developers to create, run, and verify tests during the build process.

- **Unit Tests:** Focused on testing individual modules and classes.
- **Integration Tests:** Ensure modules interact correctly.
- **Regression Tests:** Run periodically to detect bugs from recent changes.

Tools and Bug Management:

- **Tools:** Godot uses scons for building the project and doctest for running tests. Developers may also use external debuggers and performance profilers.
- **Bug Tracking:** Managed via GitHub issues. Critical bugs are prioritized, and developers submit pull requests with fixes and tests.

Testing and Documentation of tests:

1. **Running Existing Tests:**
   → Firstly build the tests using SCons.
   scons tests=yes

   → Once the build is done, run the tests with:
   ./bin/<godot_binary> --test

   Example:
   ./bin/godot.windows.editor.x86_32.exe --test

   Results are output in the terminal, detailing passing and failing tests.
   Running all tests:

```
[doctest] test cases:    1138 |    1138 passed | 0 failed | 1 skipped
[doctest] assertions: 2421432 | 2421432 passed | 0 failed |
[doctest] Status: SUCCESS!
```

2. **Writing New Tests:**
   - New tests can be generated using:
     " *python tests/create_test.py MyNewTest -i* "
   - This creates a test file in the tests/ directory with a basic structure.

1. **Test Case: [String] Find Character**

   Purpose: This test verifies the functionality of *find_char* and *rfind_char*, ensuring correct behavior when searching for characters in a string. It checks both forward and reverse searches.

```
TEST_CASE("[String] Find character") {
    String s = "racecar";
    CHECK_EQ(s.find_char('r'), 0);
    CHECK_EQ(s.find_char('r', 1), 6);
    CHECK_EQ(s.find_char('e'), 3);
    CHECK_EQ(s.find_char('e', 4), -1);

    CHECK_EQ(s.rfind_char('r'), 6);
    CHECK_EQ(s.rfind_char('r', 5), 0);
    CHECK_EQ(s.rfind_char('e'), 3);
    CHECK_EQ(s.rfind_char('e', 2), -1);
}
```

```
===============================================================================
[doctest] test cases: 1 | 1 passed | 0 failed | 1138 skipped
[doctest] assertions: 8 | 8 passed | 0 failed |
[doctest] Status: SUCCESS!
```

2. **Test Case: [Array] remove_at()**

   Purpose: This test checks the behavior of the *remove_at* method, ensuring that elements are correctly removed by index and verifying behavior when attempting to remove from an empty array.

```
TEST_CASE("[Array] remove_at()") {
    Array arr;
    arr.push_back(1);
    arr.push_back(2);
    arr.remove_at(0);
    CHECK(arr.size() == 1);
    CHECK(int(arr[0]) == 2);
    arr.remove_at(0);
    CHECK(arr.size() == 0);

    // The array is now empty; try to use `remove_at()` again.
    // Normally, this prints an error message so we silence it.
    ERR_PRINT_OFF;
    arr.remove_at(0);
    ERR_PRINT_ON;

    CHECK(arr.size() == 0);
}
```

```
===============================================================================
[doctest] test cases: 1 | 1 passed | 0 failed | 1138 skipped
[doctest] assertions: 4 | 4 passed | 0 failed |
[doctest] Status: SUCCESS!
```

3. **Test Case: [OS] UTF-8 environment variables**

   Purpose: This test ensures that UTF-8 encoded environment variables are correctly

set, retrieved, and unset. It tests string encoding and decoding for proper handling of non-ASCII characters.

```
TEST_CASE("[OS] UTF-8 environment variables") {
    String value = String::utf8("hell\xc3\xb6"); // "hellö", UTF-8 encoded

    OS::get_singleton()->set_environment("HELLO", value);
    String val = OS::get_singleton()->get_environment("HELLO");
    CHECK_MESSAGE(
            val == value,
            "The previously-set HELLO environment variable"
            "should return the expected value.");
    CHECK_MESSAGE(
            val.length() == 5,
            "The previously-set HELLO environment variable"
            "was decoded as UTF-8 and should have a length of 5.");
    OS::get_singleton()->unset_environment("HELLO");
}
```

```
===============================================================================
[doctest] test cases: 1 | 1 passed | 0 failed | 1138 skipped
[doctest] assertions: 2 | 2 passed | 0 failed |
[doctest] Status: SUCCESS!
```

4. **Test Case: [Plane] Constructor methods**

   Purpose: Verifies that planes created with the same values but different methods are equal.

```
TEST_CASE("[Plane] Constructor methods") {
    const Plane plane = Plane(32, 22, 16, 3);
    const Plane plane_vector = Plane(Vector3(32, 22, 16), 3);
    const Plane plane_copy_plane = Plane(plane);

    CHECK_MESSAGE(
            plane == plane_vector,
            "Planes created with same values but different methods should be equal.");

    CHECK_MESSAGE(
            plane == plane_copy_plane,
            "Planes created with same values but different methods should be equal.");
}
```

```
===============================================================================
[doctest] test cases: 1 | 1 passed | 0 failed | 1138 skipped
[doctest] assertions: 2 | 2 passed | 0 failed |
[doctest] Status: SUCCESS!
```

5. **Test Case: Resource Duplication**

   Purpose: Verifies the behaviour of resource duplication and its impact on the original resource.

```
TEST_CASE("[Resource] Duplication") {
    Ref<Resource> resource = memnew(Resource);
    resource->set_name("Hello world");
    Ref<Resource> child_resource = memnew(Resource);
    child_resource->set_name("I'm a child resource");
    resource->set_meta("other_resource", child_resource);

    Ref<Resource> resource_dupe = resource->duplicate();
    const Ref<Resource> &resource_dupe_reference = resource_dupe;
    resource_dupe->set_name("Changed name");
    child_resource->set_name("My name was changed too");

    CHECK_MESSAGE(
            resource_dupe->get_name() == "Changed name",
            "Duplicated resource should have the new name.");
    CHECK_MESSAGE(
            resource_dupe_reference->get_name() == "Changed name",
            "Reference to the duplicated resource should have the new name.");
    CHECK_MESSAGE(
            resource->get_name() == "Hello world",
            "Original resource name should not be affected after editing the duplicate's name.");
    CHECK_MESSAGE(
            Ref<Resource>(resource_dupe->get_meta("other_resource"))->get_name() == "My name was changed too",
            "Duplicated resource should share its child resource with the original.");
}
```

```
[doctest] test cases: 1 | 1 passed | 0 failed | 1138 skipped
[doctest] assertions: 4 | 4 passed | 0 failed |
[doctest] Status: SUCCESS!
```