the zoidberg engine

Search this site

Welcome to the zoidberg engine Infosphere
Project Definition
Resources
Game Ideas
Documentation
The Team
To-Dos
Project updates
Files
Release Dates
Sitemap

136
days since
Project Due Date

Recent site activity

removed stuff
edited by Joe Balough

The Team
edited by rmb5299

removed stuff
edited by Joe Balough

cliCreator
edited by Joe Balough

Testing Framework
edited by Joe Balough

View All

Documentation >

Testing Framework

Intro

The zoidberg engine has a testing framework that is capable of running two different kinds of tests, functional and graphical. Functional tests are designed to test the zoidberg engine custom datatypes that do not have any graphical representation and cannot be tested by showing the user something. Graphical tests are designed to test graphical methods that have no way of confirming that they are working programmatically and input must come from the user.

Contents

Functional Tests

Functional tests should be used to test any and all functions of the zoidberg engine that can be tested prommatically. For example, the first functional test added to the framework was a test to determine whether or not the collisionMatrix class operated correctly. This class is an ideal candidate for a functional test since it has no graphical representation and all of its functions can be tested by simply making a dummy collisionMatrix and running its functions.

Adding a Functional Test

Functional tests are added by making a new class that inherits the abstract `functionalTest` class in the `testing.cpp` file. Within this class there must be a `virtual bool run()` function that runs the test and returns whether the test was successful or not and a constructor that initializes the `name` variable with the name of the test (should be like "foobar Test").
This class should follow zoidberg engine Code Standards and should contain a doxygen comment before it explaining what it does.

```
/**
 * myTest functional test
 *
 * This is a very, very simple test that only prints hello world on the screen
 * when it is run. It is only used as an example in the documentation.
 *
 * @author Joe Balough
 */
class myTest : public functionalTest
{
public:
    myTest()
    {
        // Set the test name
        name="myTest Test";

        // Do construction stuff
    }

    virtual bool run()
    {
        // Run test here
        iprintf("hello world!\n");
    }
};
```

After defining the test class, it is necessary to add it to the list of tests in the `void getFunctionalTests(vector<functionalTest*> &tests)` function by creating a new instance of the new test on heap and pushing it on the tests vector. It will be automatically deleted later.
This code, too, should follow the zoidberg engine Code Standards and be prefixed with a comment with a brief description.

```
void getFunctionalTests(vector<functionalTest*> &tests)
{
    ...

    // A simple example functional test that just prints "hello world!"
    myTest *myt = new myTest;
    tests.push_back((functionalTest*) myt);

    ...
}
```

After completing those two steps, a new functional test will appear in the functional test list and will be run when running all tests.

Graphical Tests

Graphical tests are to be used for game functions that cannot be tested for correctness using a function. The general steps to running a graphical test are the following:

1. Tell the user what is *supposed* to happen in the following test, wait for user to press a button to continue.
2. Run the test for a predetermined amount of time.
3. Ask the user if the test ran correctly
4. If the user says that it did not run correctly, print a debug message that describes what *might* have gone wrong.

The graphical tests for the testing build are defined in the assets file as levels but include three additional attributes: a `timer` attribute in the `<level>` tag, an `<exp>` group to explain what should happen, and a `<debug>` group to explain what might have gone wrong. See the XML Description section of the cliCreator page for more details.

When listing the graphical tests, the level's `<name>` value is used as a test name. When selected from the list, the text in the `<exp>` tag is presented to the user and should explain what the test is supposed to do. After the user presses a button, the engine will run that level for `timer` screen blanks then ask the user if the test ran correctly. If the user says that it did not run correctly, it will print the contents of the `<debug>` tag to tell the user where something probably went wrong.

Adding a Graphical Test

Since graphical tests are just levels in a special zbe file, all that needs to be done is that a new level needs to be added to the testing.xml file that generates the testing.zbe assets file. Each new graphical test should define a test that tests one simple things. The following is the list of some of the currently implemented graphical tests and what they test:

- Scrolling Background
  - Tests the `update()` function of the `background` class.
  - Allows the user to scroll a hero around the screen for 15 seconds to determine if the background scrolls correctly.
  - If the test runs correctly, the background will scroll properly without distorting the background pattern.
  - If the test fails, the message "something went wrong in the background::update() function" is given to the user.
- Basic Collision Resolution
  - Tests basic collision detection and resolution.
  - Defines two objects, one resting on a platform with one above it. When the test starts, the second object falls to land on top of the first object.
  - If the test runs correctly, that second object should stop moving immediately above the first.
  - If the test fails, the message "There is a problem with the default collision resolution."
- Weighted Collision Resolution
  - Tests collision resolution based on object weight.
  - Defines a confined tunnel with a Hero on the left end and an object immediately to the right of it with the path on the left blocked.
  - If the test runs correctly, the user can move the hero to the right and, because the hero is defined to be heavier than the object, push the object along the tunnel.
  - If the test fails, the message "There is a problem with the default collision resolution."

- Moving Platform
  - Tests collision resolution with moving platforms.
  - Defines a vertical tunnel with a platform at the bottom and a hero on top of it.
  - If the test runs correctly, when the test begins, the platform begins moving up, pushing the hero along with it.
  - If the test fails, the message "There is a problem with the default collision resolution."

Each of these tests are short and designed to test only one small part of the zoidberg engine.

Making the testing build

The testing build can be can be made by building the engine with `ZBE_TESTING` defined. This can be done simply with the makefile by running

```
make testing
```

Issues come up when switching between builds, so if the last target built was `all` (default) and the `testing` build is desired, it is necessary to run

```
make clean
```

before running

```
make testing
```

The same goes for switching from the testing build to the all build. In this situation,

```
make clean
```

must be run before running

```
make
```