Swarm Visualiser - COS 301 Main Project
Testing Specifications

Team: Dragon Brain
Members:
Matheu Botha u14284104
Renton McInytre u14312710
Emilio Singh u14006512
Gerard van Wyk u14101263

Contents

1 Introduction

1.1 Purpose

This document describes the testing methodologies and frameworks used in the Swarm Visualiser project by team DragonBrain. The general purpose of the project is to create a functional experimental and teaching tool that allows the functioning of a Particle Swarm Optimisation problem solver to be conceptualised and visualised to display a more comprehensible impression of the inner workings of such systems.
This document serves as a recording of the methods and specific tests used to ensure proper functionality of the project, thus permitting proper test driven development processes to be followed. This is a necessity in order to ensure that the system in question has minimal risk of failure as the system develops.

1.2 Scope

This document is structured as follows:

- Tests that have been identified are specified in section 2.
- Features to be tested are specified in section 3.
- Sections 4 through 6 will discuss the tests indepth.
- Section 7 will discuss the results of the testing.
- Sections 8 and 9 will conclude with additional comments and explanations.
- The remainder of this section will be used to discuss the testing environment, as well as assumptions and dependencies.

1.3 Test Environment

The environment of the testing system is as follows:

- Programming Languages: C++ has been used as the base language with which the system is coded. Additionally, graphical subsections of the system make use of OpenGL and the Graphical User Interface is made using Qt libraries.

- Testing Frameworks: The system's unit tests are run and handled using the cross-platform Unit Testing library, Google Test, Google's own testing framework for C++ applications.
- Coding Environment: All coding has been done using CLion, Jetbrains' IDE for C++ (which has integrated support for Google Tests), along with CMake for automated building of the system.
- Operating System: In true spirit of cross-compatibility, tests have been run under two particular operating systems, namely Windows 10 and Linux Mint (a distribution of Linux).

1.4 Assumptions and Dependencies

For the sake of these tests, the following dependencies between subsystems are assumed:

- The Manager depends upon the Settings Package, the Graphics Pipeline, the General Optimiser and the Snapshot Manager.
- The Graphics Pipeline depends on the Snapshot Manager and the Settings Package.
- The General Optimiser depends on the Snapshot Manager and the Settings Package.

2 Test Items

Following is a list of the unit tests performed:

- Snapshot Manager subsystem functionality.
- General Optimiser subsystem functionality.
- Settings Package subsystem functionality.

3 Functional Features to be Tested

3.1 Snapshot Manager

The features to be tested are as follows:

- The ability of the subsystem to create a correct snapshot of the current particle situation.

- The ability of the subsystem to correctly function as a standard queue of snapshots (hence, the ability to enqueue and dequeue snapshots correctly).
- The ability of the subsystem to handle additional bounded queue functionality.

These are all low level tests that will be tested with a series of simple assertion-like methods, to ensure that the queue functions as it is meant to. The queue's main design requirement is Scalability.

3.2 General Optimiser

The features to be tested are as follows:

- The ability of the subsystem to correctly generate particle objects.
- The ability of the subsystem to correctly generate a swarm with a valid n-matrix.

3.3 Settings Package

The features to be tested are as follows:

- The ability of the subsystem to create a valid Graphics Settings Package.
- The ability of the subsystem to create a valid Problem Domain Settings Package.
- The ability of the subsystem to create a valid Optimiser Settings Package.

4 Test Cases

4.1 Snapshot Manager

4.1.1 Case 1: Generating a Snapshot

- Objective: To ensure basic generation of a Snapshot is functional.
- Input: An array of Particles and an array of integers representing the links between them.
- To assume a pass result, the expected outcome is for a single Snapshot which has the Particles and links as members to be created.

4.1.2 Case 2: Generating a Snapshot Queue

- Objective: To ensure basic generation of a Snapshot is functional.
- Input: An array of Particles and an array of integers representing the links between them.
- To assume a pass result, the expected outcome is for a single Snapshot which has the Particles and links as members to be created, which is then expected to be enqueued into the Snapshot Manager and dequeued successfully.

4.2 General Optimiser

4.2.1 Case 1: Hill-climber OPT Process

- Objective: To ensure that during a single iteration, using the Hill-Climber process,that the particles are able to perform an optimisation action.
- Input:
  - A Snapshot consisting of the following:
    - A particle Swarm
    - An Objective Function
- To assume a pass result, the expected outcome is for at least one particle in the swarm to be at a position better than it originally was. In ideal conditions, the swarm would have every particle reach a better position, the best, and reach a convergence point but for the purposes of testing, at least one particle needs to be in a position better as defined by its objective function.

4.2.2 Case 2: Conical PSO OPT Process

- Objective: To ensure that during a single iteration, using the Conical Particle Swarm Optimisation process,that the particles are able to perform an optimisation action.
- Input:
  - A Snapshot consisting of the following:
    - A particle Swarm

        * An Objective Function

- To assume a pass result, the expected outcome is for at least one particle in the swarm to be at a position better than it originally was. In ideal conditions, the swarm would have every particle reach a better position, the best, and reach a convergence point but for the purposes of testing, at least one particle needs to be in a position better as defined by its objective function.

4.2.3 Case 3: General PSO OPT Process

- Objective: To ensure that during a single iteration, using the general particle swarm optimisation process,that the particles are able to perform an optimisation action.

- Input:

  - A Snapshot consisting of the following:
    * A particle Swarm
    * An Objective Function

- To assume a pass result, the expected outcome is for at least one particle in the swarm to be at a position better than it originally was. In ideal conditions, the swarm would have every particle reach a better position, the best, and reach a convergence point but for the purposes of testing, at least one particle needs to be in a position better as defined by its objective function.

4.3 Settings Package

4.4 Case 1: Generatng a SettingsPackage Object

- Objective: To instantiate a SettingsPackage Object comprised of A ProblemDomainSettingsPackage, GraphicsSettingsPackage, and an OptimizerSettingsPackage, as well as some local independent variables.

- Input: A series of values for initialization of the various components

- : To assume a pass result, it must be confirmed that all composing objects of the SettingsPackage Object contain the attributes that are expected.

4.5 Objective functions

4.6 Case 1: Generating a sinObjective

- Objective: To instantiate a specific objective function(sinObjective) and test that it returns a value in the correct range, for a given random input.
- Function: sinObjective calculates the formula sin(x)+sin(y), where x and y are the 2 input parameters.
- Input:
  - a sinObjective instance does not require any parameters for its constructor.
  - the function in sinObjective is called with random parameters to assure that it is tested with a diversity of values, making the test more robust.
- To achieve a pass result the value returned by sinObjective(after being called with random parameters) should always be within the range [0,2], if the returned value is outside of that range it is a fail result.

4.7 Case 2: Checking the integrity of the AckleyObjective

- Objective: Determine if the Ackley Objective function is functioning as expected.
- Function:

$$f(x_0 \cdots x_n) = -20exp(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}) - exp(\frac{1}{n}\sum_{i=1}^{n} cos(2\pi x_i)) + 20 + e$$

$$-32 \leq x_i \leq 32$$

`minimum at f(0, ... ,0) =0`

- Input:
  - An AckleyObjective instance does not require any parameters for its constructor.

  - The function in AckleyObjective is called with the specific value of (0,0) and then with a random value at any point other than (0,0).

- If the function returns 0 for (0,0) and any positive value for the randomly chosen point, then it passes, otherwise it fails.

4.8 Particle

4.8.1 setVelocity

- Objective: To ensure that a particle's velocity value can be set by being passed an external vector.
- Input: An array consisting of double values of a pre-specified length.
- To assume a pass result, the expected outcome is for the private array representing the particle's position vector to be set to exactly the values specified by the passed in array.

4.8.2 setPositionAtDimension

- Objective: To ensure that a particle's position vector at a specific dimension can be set by being passed an external value.
- Input: Two double values: one representing the position value, the other the dimension.
- To assume a pass result, the expected outcome is for the private array at the specified dimension, representing the particle's position vector, to be set to exactly the value specified by the parameter passed in.

4.8.3 getVelocity

- Objective: To ensure that a particle's velocity value can be set by being passed an external vector.
- Input: No input.
- To assume a pass result, the value returned must be the same as the value stored by the particle that getVelocity was called on.

4.9 Graphics Processor

4.9.1 Case 1: Generating a Graphics Processor

- Objective: Ensure that a Graphics Processor is correctly instantiated with an objective function and a snapshot manager.
- Input: An objective function representing some mathematical formula.
- To assume a pass result, the expected outcome is for the graphics processor to correctly create a mesh from the given objective function as well as be able to dequeue from the snapshot manager and create a particle system from the snapshot.

4.9.2 Case 2: Generating a Landscape Mesh

- Objective: Ensure that a mesh can be generated from a objective function.
- Input: An objective function representing some mathematical formula.
- To assume a pass result, it is expected that the given objective function will produce values from given co ordinates and create an accurate mesh that can be drawn to the screen.

4.9.3 Case 3: Generating a particle System

- Objective: Ensure that a particle system can be generated from the snapshot manager.
- Input: A valid snapshot manager.
- To assume a pass result, it is expected that the graphics processor will be able to dequeue from the snapshot and use the particle co ordinates to create a particle system.s

5 Item Pass/Fail Criteria

Each item tested must meet criteria specific to its particular scenario in order to be considered passed or failed. These are tested with assertions and similar methodologies, hence if an item is failed it will be detected as such by Google Tests.

5.1 General Optimiser: OPT Process

In general, the difficulty of testing is predicated on the stochastic nature of the particle system, exact precise testing is difficult. However, that being said, there are certain capacities that would be considered fail and considered a pass.

The passing condition would require any combination of the following:

- At least one particle is in a better position according to the appropriate function.

The failing condition would require the following:

- More than 50 percent of the particle swarm ended in a worse position than the previous iteration and it continues for at least some proportional(x) degree of the total number of iterations.

6 Detailed Test Results

6.1 Snapshot Manager

6.1.1 Case 1: Generating a Snapshot

The following results were obtained from the tests conducted. The tests to produce the following results have **passed** and the reasons are stated below.

- Created Snapshot Object and confirmed that the links are as they were expected to me.
- Snapshot Object has a list of particles matching the expectation.

6.1.2 Case 2: Generating a Snapshot Queue

The following results were obtained from the tests conducted. The tests to produce the following results have passed and the reasons are stated below.

- Basic Enqueue/Dequeue check passes (item enqueued correctly matches dequeued item).
- Bounded Queue behaviour demonstrated correctly (no further items enqueued after [bound] number of items have been enqueued).
- Dequeue fails correctly when attempting to dequeue from an empty queue.

6.2 General Optimiser

6.2.1 Case 1: Hill-climber OPT Process

The following results were obtained from the tests conducted. The tests to produce the following results have passed and the reasons are stated below.

- The Hill-Climber process is considered to be stable. By this, this means that the probability of any of the particles not becoming even slightly more improved is very low. Even in stable stochastic settings, it is very unlikely that, for sufficiently large swarms, none of the particles achieve some position at termination of the run less optimal than their position at the start.

    1. A sufficiently large test swarm, 20 particles, was used.
    2. The Marsenne Twister Pseudo Random number generator improves the stochastic reliability of the test.

6.2.2 Case 2: Conical PSO OPT Process

The following results were obtained from the tests conducted. The tests to produce the following results have passed/failed and the reasons are stated below.

- The Conical PSO process is considered to be stable. By this, this means that the probability of any of the particles not becoming even slightly more improved is very low. Even in stable stochastic settings, it is very unlikely that, for sufficiently large swarms, none of the particles achieve some position at termination of the run less optimal than their position at the start.

    1. A sufficiently large test swarm, 20 particles, was used.
    2. The Marsenne Twister Pseudo Random number generator improves the stochastic reliability of the test.

6.2.3 Case 3: General PSO OPT Process

The following results were obtained from the tests conducted. The tests to produce the following results have passed/failed and the reasons are stated below.

- The PSO process is considered to be stable. By this, this means that the probability of any of the particles not becoming even slightly more improved is very low. Even in stable stochastic settings, it is very unlikely that, for sufficiently large swarms, none of the particles achieve some position at termination of the run less optimal than their position at the start.

    1. A sufficiently large test swarm, 20 particles, was used.
    2. The Marsenne Twister Pseudo Random number generator improves the stochastic reliability of the test.

6.3 Particle

6.3.1 setVelocity

The following results were obtained from the tests conducted. The tests to produce the following results have passed and the reasons are stated below.

- The velocity passed in was a double value in the range of allowable values.
- The particle was correctly initialised.

6.3.2 setPositionAtDimension

The following results were obtained from the tests conducted. The tests to produce the following results have passed/failed and the reasons are stated below.

- The position value passed in was a double value in the range of allowable values.
- The dimension value was within the range of dimensionality of the position vector.
- The particle was correctly initialised.

6.3.3 getVelocity

The following results were obtained from the tests conducted. The tests to produce the following results have passed/failed and the reasons are stated below.

- The Velocity value provided was within the range of allowable velocities.
- The particle was correctly initialised with correct initial values.

6.4 Settings Package

6.4.1 Case 1: Generating a SettingsPackage Object

The following results were obtained from the tests conducted. The tests to produce the following results have **passed** and the reasons are stated below.

- Object's own attributes are confirmed equal to the expected values.
- ProblemDomainSettingsPackage Object's attributes are confirmed equal to expected values.
- GraphicsSettingsPackage Object's attributes are confirmed equal to expected values.
- OptimizerSettingsPackage Object's attributes are confirmed equal to expected values.

7 Other

- Firstly, a comment needs to be made regarding the use of Maven. Maven is a build system that is somewhat at odds with the project structure and project specifications. C++ is a language that encourages the use of two kinds of files, h and cpp, in order to construct a single class. Couple this together with inherently standalone nature of the project and it creates an additional, but not necessarily better or needed, requirement to configure a Maven build system. Currently, the existing system is an integration of the CLion IDE by Jet Brains with Github. The IDE is capable of opening and launching a project using MakeList files written by us directly out of a branch from Github. To that end, our current strategy is to have stable and development branches of any component and develop directly out of the branches including building.
- Contracts and Mock Objects have been used in the project. The latter used heavily during earlier development of the project was required in

order to test the Graphics Processor as it could not be tested without Mock Objects. The former has been extremely useful in terms of clearly specifying the requirements of components of the system. System components would have their requirements and services specified clearly by contracts and those would be used to ensure system correctness.

- The deployability of the project to an application server is an interesting question. In general the project is stated to have performance as a key quality requirement. Extrapolating the service to an application server would require some significant changes to enable high performance over a network. Application servers generally are a Java specific component and again, that stands at odds with the C++ that was required for the project so we believe that deployment to an application server is not advisable with the project.

8 Conclusions and Recommendations

There are two fundamental limitations to the testing included here in this document. The first is that due to the stochastic nature of the underlying optimisation strategies, it is essentially impossible to precisely define parameters during testing. So largely, this means that testing consists of observing that general positive or negative trends become apparent rather than being able to accurately and specifically check for values. The second limitation would be the Graphics Processor itself. The Graphics Processor produces visual output mapped to a screen which is difficult to rigorously test according to unit testing specifications.

A mention needs paying to the testing environment which has been slightly troublesome. Early on in the development of the project, CLion was chosen as the development environment because it offered very many features and integrations that did not exist in many other competing products. However, its one limitation would be that only one testing framework is implemented, Google Test, for it. Google Test requires a specific project structure and that imposition is unfortunately one of the trade-offs made in the design process.

In conclusion, the document presented here is a summation of the most important aspects to the testing and refining of the project. The provisions

made here hopefully illuminate the reader on the more technical aspects of the project.

9 Appendix- Unit Testing Examples

Below are all of the unit test example screenshots presented here because of the size of the screenshots and to prevent a cluttered paragraph structure.

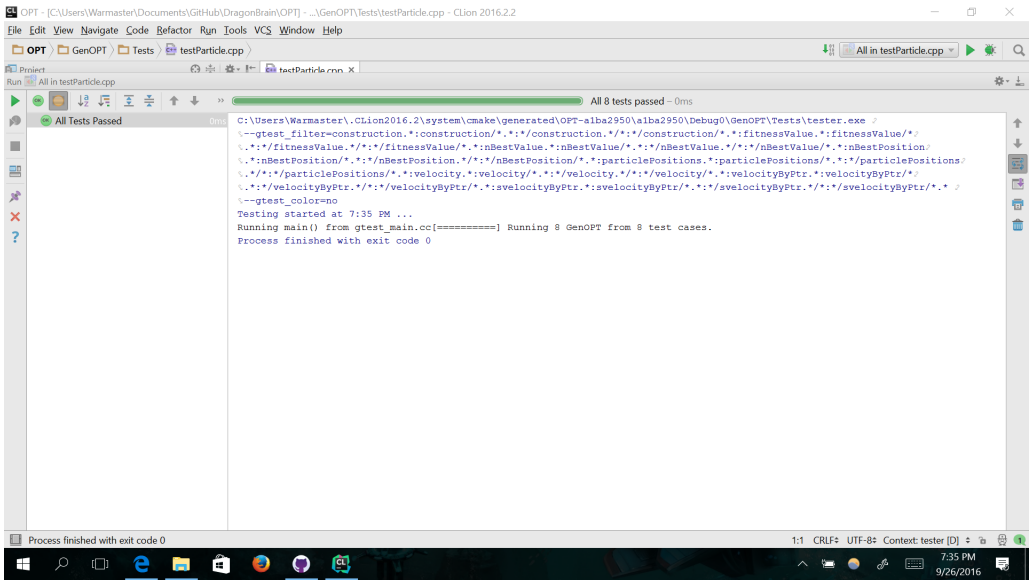

Figure 1: Example Unit Test Performed

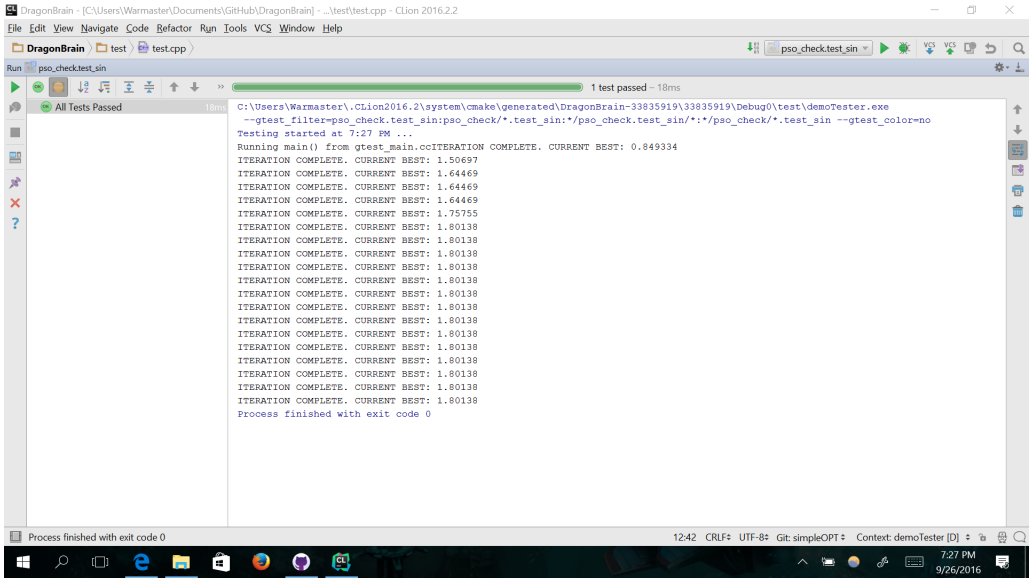

Figure 2: Example Unit Test Performed

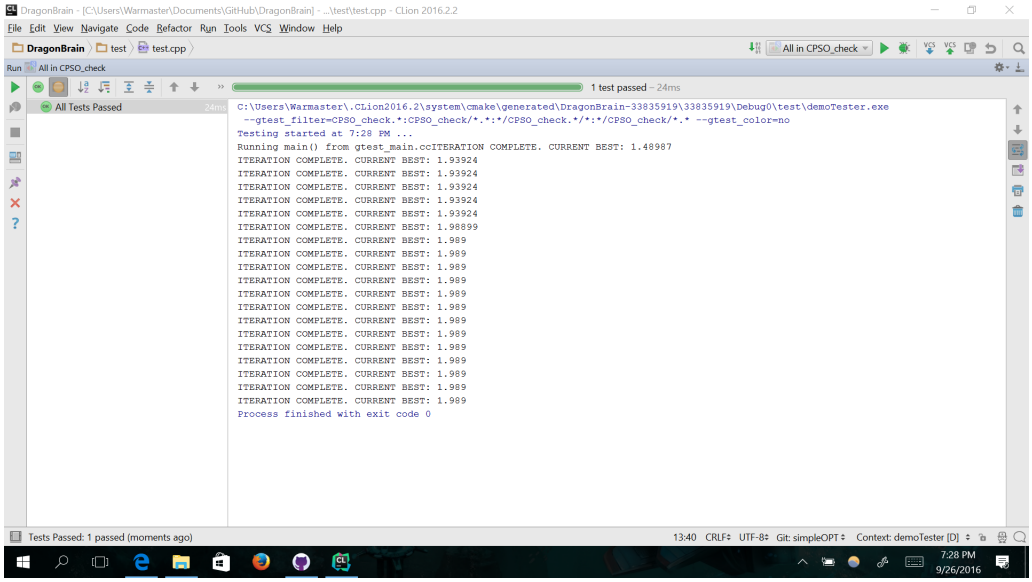

Figure 3: Example Unit Test Performed

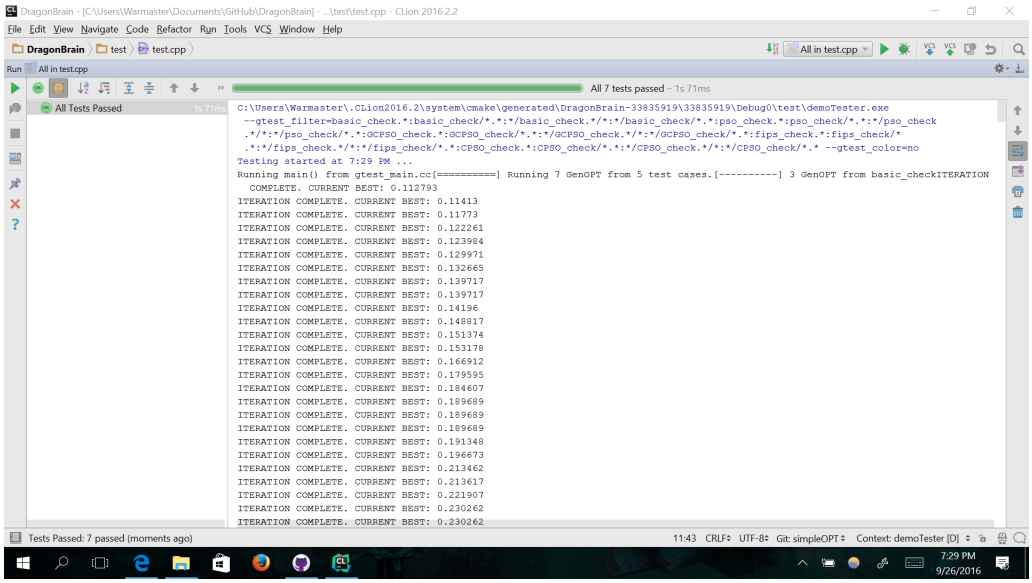

Figure 4: Example Unit Test Performed