Contents

[1. Overview 2](#_Toc432006952)

[1.1 Testing Scope 2](#_Toc432006953)

[1.2 Roles and Responsibilities 2](#_Toc432006954)

[1.3 Organization and Resources 2](#_Toc432006955)

[2. Unit Testing 3](#_Toc432006956)

[2.1 Unit Test Policy 3](#_Toc432006957)

[2.2 Unit Tests 3](#_Toc432006958)

[3. Integration Testing 5](#_Toc432006959)

[3.1 Integration Policy 5](#_Toc432006960)

[3.2 Integration Tests 5](#_Toc432006961)

[4. Use Case Examples 6](#_Toc432006962)

[4.1 Environment Use Cases 6](#_Toc432006963)

[4.1.1 Score Members 6](#_Toc432006964)

[4.1.2 Select Parents 6](#_Toc432006965)

[4.1.3 Perform Mutation 7](#_Toc432006966)

[4.2 Brain Use Cases 7](#_Toc432006967)

[4.2.1 Add New Neuron 7](#_Toc432006968)

[4.2.2 Add New Connection 7](#_Toc432006969)

[4.2.3 Randomize Connection Weight 8](#_Toc432006970)

[4.2.4 Activate Network 8](#_Toc432006971)

[4.3 Member Use Cases 9](#_Toc432006972)

[4.3.1 Check Collisions 9](#_Toc432006973)

[4.3.2 Update Position 9](#_Toc432006974)

[4.4 Database Use Cases 10](#_Toc432006975)

[4.4.1 Record Members 10](#_Toc432006976)

# 1. Overview

## 1.1 Testing Scope

The tests used in this project follow a traditional top-down approach. The program was initially built using an iterative methodology prior to the creation of a test plan; however the procedures used can be detailed and written down. The program consists of a number of high level modules, which are in turn composed of smaller units. The test plan will extend to the top-down overview of the primary systems and key sub-systems required to get the program operational. Since the project has already gone through a number of iterations, this document will cover the strategies already used for the primary functions critical to system operation.

The most important parts of the program are the interactions, genetic processes, and neural network operations. These facets must be tested to ensure operation under any condition as the environment progresses over time. The more intensive parts of the member class that will be tested are: target interaction, movement, and data recording. The portions of the environment class that will be tested are: scoring, mutation, crossover, selection, and target updates. The brain class will be tested for the following: adding of new neurons, adding of new connections, randomizing of existing connections, and activation of the network. Targets will be tested to ensure they initialize with the proper coordinates. The database class will be tested to ensure that storing and retrieval of data works correctly, as well as the generation of an optimization report.

Helper class functions will not be included in this documentation as their use is dependent on the success of parent functions. We will also not test the rendering system as visual confirmation of success is integrated directly into the routines.

## 1.2 Roles and Responsibilities

Typically the test cases would be allocated based on the team member’s areas of expertise and experience, with considerations to the portions of the project they worked on directly. Since this project is limited in resources to one member, all testing responsibilities will be allocated accordingly. Unit Test construction and code, Use Case Modeling, and Integration Test construction and code will be completed by Taylor Benner.

## 1.3 Organization and Resources

Coded unit tests will self-contained in their own module files located in the tests sub-directory of the project directory. The file name will be the class that the tests are constructed for, and each file will contain its own Suite Runner. Tests for a module will be contained in a test suite for that class, and will be available as a function list within the module.

As the program is written in the Python Programming Language, the integrated Unit Testing module, PyUnit, will be used to run the code. Each test defined in a suite will assert one or several test cases to ensure that the expected output of functions and code blocks conforms to the standards of the program. These tests cover base program functionality and should be run on each new project build.

# 2. Unit Testing

## 2.1 Unit Test Policy

Due to limited time and resources, system critical function will be the only units considered. Particular units or functions will be tested to ensure operability in action. This will allow us to asset expected results within the framework of the system. Units that require specific input and provide discrete output will be tested using values indicated in the expected results section of the plan in section 2.2. The Test ID will be related to the specific requirement that may require the functionality, as well as being used later in the Use Case Examples to indicated inclusions and requirement references. Given that much of the code has been written already and has been shown to work, the Actual Results indicated that the program is currently exhibiting the expected behavior. In a Test Driven Development paradigm, this section would have been used to indicate if the test has been run, and the bug-like status of the output.

## 2.2 Unit Tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Test ID | Description | Expected Results | Actual Results |
| 1 | 2.6.2.1.1 | Member should save data to database | New record | PASS |
| 2 | 2.6.2.1.2 | Recorded closest target to member | Smallest Distance | PASS |
| 3 | 2.6.2.1.3 | Calculate distance to reference point | [0,100][100,200] => 141.421 | PASS |
| 4 | 2.6.2.1.4 | Calculate relative angle to reference point | [0,100,90][100,200] => 45.3 | PASS |
| 5 | 2.6.2.1.5 | Given speed and angle, move member to new position | [.41,6.1][288.3,151] => [295,284] | PASS |
| 6 | 2.6.2.1.6 | Given track speeds, calculate rotational change | 0 + 0.1689 = 0.1689 | PASS |
| 7 | 2.6.2.1.7 | Collision increment counter, add energy, mark target for removal | [ 1, 1, 1] | PASS |
| 8 | 2.6.2.1.8 | Member initializes instance of Brain | Brain != None | PASS |
| 9 | 2.6.2.2.1 | A population of N members should be created when none exist | len(members) = N | PASS |
| 10 | 2.6.2.2.2 | The sum of all member scores should be 1 | 1 | PASS |
| 11 | 2.6.2.2.3 | Should select N parents leaving 0 members | [ N, 0 ] | PASS |
| 12 | 2.6.2.2.4 | Should produce N members leaving 0 parents | [ N, 0 ] | PASS |
| 13 | 2.6.2.2.5 | Should run functions successfully to mutate | Mutations >= 0 | PASS |
| 14 | 2.6.2.2.6 | Target list is updated to remove or add targets | +/- len(targets) | PASS |
| 15 | 2.6.2.3.1 | Network should initialize with shape [ 4, 1, 2 ] | Shape = [ 4, 1, 2 ] | PASS |
| 16 | 2.6.2.3.2 | Should add random neuron | Initial Layers + 1 | PASS |
| 17 | 2.6.2.3.3 | Should add random connection | Initial Connections + 1 | PASS |
| 18 | 2.6.2.3.4 | Should accept parameters and return 2 floats between -1 and 1 | [ -1/1, -1/1 ] | PASS |
| 19 | 2.6.2.3.5 | Should randomize random connection | NW does not equal OW | PASS |
| 20 | 2.6.2.4.1 | Target initializes within bounds of view | R <= X <= W || R <= Y <= H | PASS |
| 21 | 2.6.2.5.1 | Key is generated for a member using Timestamp – Lifespan – Member – Generation | K = time-life-member-generation | PASS |
| 22 | 2.6.2.5.2 | Value is saved to database by key | After save: key, value = input | PASS |

# 3. Integration Testing

## 3.1 Integration Policy

Each module exists as its own class and serves a specific set of purposes. Since abstraction and inheritance is not used, templates or interfaces are not used as the primary means of integration testing. In this way, and as described in the design documentation, integration is a series of encapsulated pointers to instances of subsequent classes. The Main class should hold an instance of the Environment and Database classes. The Environment class should hold instances of Member and Target classes. The Member class should hold an instance of the Brain class. Peripheral helper functions exist outside of class structure and are available as needed to all classes and functions.

At points in the program’s logical flow, component functions must behave in a specific linear fashion, such as the propagation of a new generation, or the mutation chance. The bulk of the integration problems could arise in the update routine of the Main class, specifically where environmental processes are concerned.

Since no discrete output is expected from the integration testing, the program will be tested for fatal errors, asserting only when a return value is expected from the integration. In the case of specific linear progression as mentioned above, the transformed data will be tested against expected results.

## 3.2 Integration Tests

|  |  |  |
| --- | --- | --- |
| Test ID | Description | Actual Results |
| 3.2.1 | Main should initialize pygame. A pygame instance should be available. | PASS |
| 3.2.2 | Main should initialize Environment. Environment functions should be accessible from the primary Main class routines. | PASS |
| 3.2.3 | Environment should initialize N members and targets, making member functions available. | PASS |
| 3.2.4 | Environment should be able to perform scoring on the collection of members | PASS |
| 3.2.5 | Environment should be able to perform selection on the collection of members, resulting in a cleared list of members, and a filled list of parents. | PASS |
| 3.2.6 | Environment should update the list of targets to remove targets marked for consumption and add new targets to ensure target count remains the same | PASS |
| 3.2.7 | Member should initialized an instance of the Brain class | PASS |
| 3.2.8 | Member abstracts the brain’s activation function to reduce the amount of chaining required. Member.brain.activate() should produce expected network output. | PASS |

# 4. Use Case Examples

## 4.1 Environment Use Cases

### 4.1.1 Score Members

|  |  |
| --- | --- |
| Use Case ID: | 2.6.2.2.2 |
| Use Case Name: | Score Members |
| Actors: | Environment, Member |
| Description: | The sum of all member scores should be 1 |
| Trigger: | Perform scoring function |
| Preconditions: | All members dead  Perform Scoring function called |
| Post conditions: | All members have scores that when summed equal 1 |
| Normal Flow: | For each member  Calculate score using lifespan plus food \* 1000  For each member  Divide member score by total score  Set member score to step 4  Add normalized scores |
| Exceptions: | Score is less than 0, set score to 0 |
| Includes: | None |
| Frequency of Use: | Once per generation |
| Special Requirements: | 2.6.2.2.1 |
| Assumptions: | Lifespan and food consumption equate to higher survivability |

### 4.1.2 Select Parents

|  |  |
| --- | --- |
| Use Case ID: | 2.6.2.2.3 |
| Use Case Name: | Select Parents |
| Actors: | Environment, Member |
| Description: | Should select a number of parents are determined by the elitist parameter, and clear the remaining members from the population. |
| Trigger: | Perform crossover function |
| Preconditions: | All members dead  All members scored  Perform crossover function called |
| Post conditions: | Parents attributes is populated with the selected members |
| Normal Flow: | Sort members by fitness values in descending order  Calculate the number of parents to select  Splice members attribute  Set members attribute to NULL |
| Exceptions: | Rounded value selection may produce a number of members smaller than population size.  Finds remaining and creates new random members |
| Includes: | None |
| Frequency of Use: | Once per generation |
| Special Requirements: | 2.6.2.2.2 |

### 4.1.3 Perform Mutation

|  |  |
| --- | --- |
| Use Case ID: | 2.6.2.2.5 |
| Use Case Name: | Perform Mutation |
| Actors: | Environment, Brain |
| Description: | This function should successfully perform 3 units resulting in mutation |
| Trigger: | Perform mutation function |
| Preconditions: | All members dead  Members scored  Parents selected  New members created from crossover |
| Post conditions: | New members have gone through mutation to produce new values |
| Normal Flow: | For each member  Iterate over connection  Random chance to randomize connection weight  Random chance to add new logic neuron  Random chance to add new connection |
| Exceptions: | Refer to related use cases:  4. 2.6.2.3.3  5. 2.6.2.3.2 |
| Includes: | 2.6.2.3.3, 2.6.2.3.2, 2.6.2.3.5 |
| Frequency of Use: | Once per generation |

## 4.2 Brain Use Cases

### 4.2.1 Add New Neuron

|  |  |
| --- | --- |
| Use Case ID: | 2.6.2.3.2 |
| Use Case Name: | Add New Neuron |
| Actors: | Brain |
| Description: | This function should successfully add a new logic neuron and randomly connect it to another layer. |
| Trigger: | Perform mutation function, add random neuron function |
| Preconditions: | Brain is initialized  Mutation is performed  Random number is less than or equal to Mutation Rate |
| Post conditions: | New neuron is added to existing brain |
| Normal Flow: | Create new layer with random type ( Sigmoid, Linear, Tanh, Gaussian )  Randomly pick layer from existing network  Add new layer as logic neuron  Add connection between two layers  Sort Modules |
| Exceptions: | Connection creates a logic loop  Remove connection  Leave new neuron unconnected |
| Includes: | Remove\_connection(), add\_logic(), add\_connection() |
| Frequency of Use: | .8% chance every member every generation |
| Notes and Issues: | Unconnected neuron may spontaneously be connected at some point in the future allowing for the creation of new neural pathways. |

### 4.2.2 Add New Connection

|  |  |
| --- | --- |
| Use Case ID: | 2.6.2.3.3 |
| Use Case Name: | Add New Connection |
| Actors: | Brain |
| Description: | This function should successfully add a new connection between all pre-existing layers |
| Trigger: | Perform mutation function, add random connection function |
| Preconditions: | Brain is initialized  Mutation is performed  Random number is less than or equal to Mutation Rate |
| Post conditions: | New connection is added to the brain |
| Normal Flow: | Select two random layers  Create Full Connection  If two layers are not the same, add connection  Sort network modules |
| Exceptions: | Connection creates a logic loop  Remove connection and continue |
| Includes: | Remove\_connection(), add\_connection() |
| Frequency of Use: | .8% chance every member every generation |
| Notes and Issues: | Connections that create a loop between layers produce fatal errors in Pybrain. A try / catch block is used to handle this exception and remove the connection after it has been added. |

### 4.2.3 Randomize Connection Weight

|  |  |
| --- | --- |
| Use Case ID: | 2.6.2.3.5 |
| Use Case Name: | Randomize Connection Weight |
| Actors: | Brain |
| Description: | This function should successfully randomize the weight of a pre-existing connection. |
| Trigger: | Perform mutation function, random chance is less than or equal to Mutation Rate. |
| Preconditions: | Brain is initialized  Mutation is performed  For each module in network  Iterate over connections in module  Random number is less than or equal to Mutation Rate |
| Post conditions: | Connection is randomized, but network remains otherwise unchanged |
| Normal Flow: | For each member  Iterate over modules  Iterate over connections for module  If random number is less than or equal to Mutation Rate  Call randomize function |
| Exceptions: | None |
| Includes: | Pybrain.structure.connection.randomize |
| Frequency of Use: | .8% chance every connection every module every member every generation |
| Notes and Issues: | This loop may double the chance of randomizing a connection weight due to multiple modules containing the same connection. |

### 4.2.4 Activate Network

|  |  |
| --- | --- |
| Use Case ID: | 2.6.2.3.4 |
| Use Case Name: | Activate Network |
| Actors: | Brain |
| Description: | This function accepts a list of parameters and returns output from the neural network. |
| Trigger: | Member update, activate function called |
| Preconditions: | Member is initialized  Brain is initialized  Member’s state is updated  Activate Function is called |
| Postconditions: | Member state is updated with new values for position |
| Normal Flow: | Member finds nearest target  Member calculates distance  Member calculate relational angle  Values are normalized  Brain is activated with input parameters |
| Exceptions: | None |
| Includes: | Pybrain.\*, Brain |
| Frequency of Use: | 60 \* Member Count per second |
| Notes and Issues: | The activation function’s description is left a bit ambiguous due to the evolving nature of the network topology. |

## 4.3 Member Use Cases

### 4.3.1 Check Collisions

|  |  |
| --- | --- |
| Use Case ID: | 2.6.2.1.7 |
| Use Case Name: | Check Collision |
| Actors: | Member, Target |
| Description: | The member should identify the closest target, calculate distance and angle, and processes collisions with targets. |
| Trigger: | Member update, target is close, target intersects |
| Preconditions: | Member is initialized  Member updates state  Member updates position  Targets are initialized |
| Postconditions: | Member gains energy  Member has closest target  Targets are marked for removal |
| Normal Flow: | Iterate over targets  Check if target coordinates are within tolerance limit of member  Check if close target is closer than current minimum  Set closest target if distance is less than current minimum  Find hypotenuse between member and target  If distance is less than the sum of each radius  Mark target for removal  Add energy to member  Increment member’s food consumption counter |
| Exceptions: | Targets are same distance away  Select last target measured |
| Includes: | 2.6.2.1.5 |
| Frequency of Use: | 60 \* Member Count per second |
| Special Requirements: | 2.6.2.1.2, 2.6.1.3, 2.6.1.4 |

### 4.3.2 Update Position

|  |  |
| --- | --- |
| Use Case ID: | 2.6.2.1.5 |
| Use Case Name: | Move member |
| Actors: | Member |
| Description: | Based on network output, a member must calculate its new position and set the new coordinates correctly. |
| Trigger: | Update State, Update position |
| Preconditions: | Member is initialized  Brain is activated  Update State is called  Update position is called |
| Postconditions: | Member is moved according to the calculated coordinates |
| Normal Flow: | Member state is updated  Left and right track values are determined from network activation  Speed is calculated  Rotational change is calculated  Change in X is calculated  Change in Y is calculated  Check boundaries of new X and Y  Set new X and Y |
| Exceptions: | Member is outside bounds of viewport  Set member to opposite edge of screen  Rotational change is > 2pi or < 0  Set rotational change to 0 or 2pi |
| Includes: | 2.6.2.3.4, 2.6.2.1.6 |
| Frequency of Use: | 60 \* member per second |
| Special Requirements: | 2.6.2.1.2, 2.6.1.3, 2.6.1.4 |

## 4.4 Database Use Cases

### 4.4.1 Record Members

|  |  |
| --- | --- |
| Use Case ID: | 2.6.2.1.1 |
| Use Case Name: | Data Storage |
| Actors: | Environment, Member, Database |
| Description: | Each generation, the environment should send member data to the database class to be saved. |
| Trigger: | Generation ends  Members are scored |
| Preconditions: | Members are initialized  Members die  Members are scored |
| Postconditions: | Database contains entries for members |
| Normal Flow: | Members are initialized  Members persist until energy is expended  Members are scored  Save function is called |
| Exceptions: | Member keys are not unique  Add timestamps to keys |
| Includes: | 2.6.2.5 |
| Frequency of Use: | Once per generation |
| Special Requirements: | 2.6.2.1.2, 2.6.1.3, 2.6.1.4 |