Simulated Evolutionary Environment

CIS 470 Project

Taylor Benner

Contents

[1. Project Plan 3](#_Toc431717304)

[1.1 System Overview 3](#_Toc431717305)

[1.2 Project Development Management 3](#_Toc431717306)

[1.2.1 Organization and Resources 4](#_Toc431717307)

[1.2.2 Personnel 4](#_Toc431717308)

[1.3 Schedule and Milestones 4](#_Toc431717309)

[1.3.1 Scheduled Activities, Tasks, and Assignments 4](#_Toc431717310)

[1.3.2 Delivery Milestones and Baselines 4](#_Toc431717311)

[1.4 Risk Analysis 6](#_Toc431717312)

[1.5 Software Engineering 7](#_Toc431717313)

[1.5.1 Standards and Procedures 7](#_Toc431717314)

[1.5.2 Development Methodology 8](#_Toc431717315)

[1.5.3 Development Resources 8](#_Toc431717316)

[1.6 Testing Procedures 8](#_Toc431717317)

[1.7 Configuration Management 9](#_Toc431717318)

[2. Requirements Specification 10](#_Toc431717319)

[2.1 Introduction 10](#_Toc431717320)

[2.2 Purpose 10](#_Toc431717321)

[2.3 Overview 10](#_Toc431717322)

[2.4 Definitions 10](#_Toc431717323)

[2.5 Methodology 11](#_Toc431717324)

[2.6 Requirements 11](#_Toc431717325)

[2.6.1 Interface Requirements 11](#_Toc431717326)

[2.6.2 Class Requirements 11](#_Toc431717327)

[2.6.2.1 Member Class Requirements 11](#_Toc431717328)

[3.2.2. Population Class Requirements 12](#_Toc431717329)

[3.2.3. Brain Class Requirements 12](#_Toc431717330)

[3.2.4. Food Class Requirements 13](#_Toc431717331)

[3.2.5. Toxin Class Requirements 13](#_Toc431717332)

[3.2.6. Database Requirements 13](#_Toc431717333)

[3.2.7. Reporting Requirements 13](#_Toc431717334)

[3. Design Specification 14](#_Toc431717335)

[3.1 Scope 14](#_Toc431717336)

[3.1.1 System Description 14](#_Toc431717337)

[3.1.2 Major Software Functions 15](#_Toc431717338)

[3.1.3 Database Description 16](#_Toc431717339)

[3.1.4 Design Constraints and Limitations 16](#_Toc431717340)

[3.2 Design Description 17](#_Toc431717341)

[3.2.1 Data Description 17](#_Toc431717342)

[3.2.2 Data Flow 17](#_Toc431717343)

[3.2.3 Program Architecture 18](#_Toc431717344)

[3.2.4 Component Interfaces 19](#_Toc431717345)

[3.3 Detailed Design 21](#_Toc431717346)

[3.3.1 Processing Description 21](#_Toc431717347)

[3.3.2 Interface Description 23](#_Toc431717348)

[3.3.3 Pseudocode 24](#_Toc431717349)

[3.3.4 Modules Uses 27](#_Toc431717350)

[4. Test Plan 31](#_Toc431717351)

[5. Application Files 32](#_Toc431717352)

[6. Project Presentation 33](#_Toc431717353)

[7. Deployment Plan 34](#_Toc431717354)

[8. Peer Review 35](#_Toc431717355)

# 1. Project Plan

## 1.1 System Overview

This system will perform the task of simulating evolution on a localized scale. In order to do so, an environment will be created that will be populated with a varying amount of entities and obstacles. The entities will be composed of sensory input, mechanical output, simulated brain, and survival needs that must be fulfilled. The obstacles will consist of hazards and benefits such as toxins or food. The entities will breed, creating subsequent generations, subject to optimized selection and mutation.

The primary mechanisms of the entities mentioned herein are grouped into 3 primary collections. Input, Output, and Needs. Inputs will be composed of sensory input in the form of vision, food intake, and damage intake. The vision of an entity will be a conic ray that judges distance to an interception point. An additional layer of vision will translate the properties of the interception point. The output of the entity will be a hyperbolic tangent between -1 and 1 indicating the magnitude of motion on left and right tracks. The needs will be described as hunger and health.

The primary mechanisms of the environment will be limited to food and toxins at first. Food will be an interception point that decreases the hunger need. Toxins will be an interception point that decrease the health need. Additional environmental mechanisms will be added, time and scope permitting.

The purpose of this system is to generate a report on the progression of optimal weights and structure of a neural network given a problem domain. For the purpose of this project, the problem domain is described as survival in a simulated environment for an entity emulating basic biological mechanisms. The key metric of success of the project will be measured by the ability to produce a report from stored data indicating optimization over time.

In terms of team composition, since the team is only composed of one member, subsequent sections will completed to the best of my ability however concessions will need to be made for Personnel, Management, and Task allocation.

## 1.2 Project Development Management

The project management process selected for this project will be in the traditional style. This is in order to determine the best course of action for the development and implementation phases given the limited resources available, as discussed in the next few sections. The initial specifications of the project will be outlined in the sections to come, and the team should strive to accomplish the key deliverables within the allocated timeframe to ensure the project is completed within budget and schedule. This is also in an effort to reduce the impact of change within the lifetime of the project to ensure dependencies remain fixed, and work can progress in a linear fashion.

### 1.2.1 Organization and Resources

Since the personnel of the team are limited in scope, the work to be done will be broken down into primary phases of development focusing on individual development phases. These phases will be discussed more in the coming sections, but a cursory overview is as follows:

1. Development Environment
2. Rendering Engine
3. Creature Development using Neural Networks
4. Simple Simulated Environment
5. Genetics
6. Simulation and Reporting

The resources available are limited in scope to 4 hours development per weekday, and 6-8 hours development per weekend. The language used will be Python incorporating abstracted libraries to reduce the amount of new code to be written. This section will be expanded upon on in Section 5.

### 1.2.2 Personnel

The personnel assigned to this team is composed of one member. This member will create the documentation, software, tests, and planning to the best of their ability within the lifetime of the project. Since the team is working on limited resource availability, a baseline of key features will be established in Section 3 that details the critical path of the project. The Work Breakdown Structure covered in Section 3.2 will identify the key features the system needs in order to accomplish the initial specifications as outlined in the System Overview, and to make concessions for features that are not along the critical path but may be added, time permitting.

## 1.3 Schedule and Milestones

This project’s lifespan will be from Monday, August 31st 2015 to Saturday, October 24th 2015. The weekly allocation for this project is 8 Weeks. The primary milestones of the project are identified in Section 3.1 and assigned as Work Packages in Section 3.2.

### 1.3.1 Scheduled Activities, Tasks, and Assignments

As indicated in the personnel section of this paper, all assignments are to be completed by the same individual. As such, the “Assigned To” portion of the Work Breakdown Structure in Section 3.2 will be removed. The milestones identified below will be assigned as work packages with specified duration and estimated delivery date via Section 3.2.

### 1.3.2 Delivery Milestones and Baselines

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Activity  Name** | **Description** | **Start Date** | **End Date** | **Dependencies** |
| 1 | Management | The ongoing processes and documents needed to manage the project | 8/31/2015 | 10/24/2015 | -- |
| 1.1 | Project Plan | The initial project document | 8/31/2015 | 9/6/2015 | -- |
| 1.2 | Team Charter | Detail responsibilities of team member | 9/6/2015 | 9/13/2015 | 1.1 |
| 1.3 | Requirements Specification | Documentation detailing business needs and schedule composition | 9/6/2015 | 9/13/2015 | 1.1, 1.2 |
| 1.4 | Design Specifications | Document detailing the design and specifications of the software | 9/13/2015 | 9/20/2015 | 1.3 |
| 1.5 | Test Plan | Create implementation documentation and user guides | 10/11/2015 | 10/18/2015 | 1.4, 2, 3 |
| 1.6 | Final Deliverable | Organize and revise documentation pertaining to project for delivery | 10/18/2015 | 10/24/2015 | 1, 2, 3 |
| 2 | Environment Setup | The initial environment set up | 9/6/2015 | 9/20/2015 | 1 |
| 2.1 | Development Environment | Select needed technologies based on requirements and create the development environment | 9/13/2015 | 9/20/2015 | -- |
| 2.2 | Library Installation | Acquire and Install software dependencies | 9/13/2015 | 9/20/2015 | -- |
| 2.3 | Code Repo | Create subversion repository for code on GitHub | 9/6/2015 | 9/13/2015 | -- |
| 2.4 | Development Environment | Install Sublime Text 2 and Python 2.7 | 9/6/2015 | 9/13/2015 | -- |
| 3 | Software Development | The creation of the software packages | 9/20/2015 | 10/11/2015 | 1, 2 |
| 3.1 | Rendering | Using PyGame to create a modular rendering system | 9/20/2015 | 9/24/2015 | - |
| 3.2 | Entity Development | Using PyBrain to create the entity class | 9/24/2015 | 9/27/2015 | 3.1 |
| 3.3 | Environment Development | Creating simple environment and data gathering | 9/27/2015 | 9/30/2015 | 3.2 |
| 3.4 | Genetics Implamen. | Implement genetic algorithm to optimize entity | 9/30/2015 | 10/06/2015 | 3.3 |
| 3.5 | Reporting | Create reports showing optimization over time | 10/06/2015 | 10/11/2015 | 3.4 |

# 

## 1.4 Risk Analysis

* Limited Personnel Available  
  Strategy – Mitigation  
  This is the project’s greatest risk. Due to the lack of team members, if the involved member falls behind schedule, the rest of the project’s quality or implicit budget may suffer. The method employed to deal with this is to ensure that scope is kept under control in terms of applicable features of the system and preemptive documentation efforts. This risk will require active mitigation efforts to ensure it does not pose a problem over the lifetime of the project. Risk mitigation will be monitored via the baseline established in Section 3.2 to ensure the project does not fall behind schedule
* Scope Creep  
  Strategy – Avoidance   
  Given the nature of this project, scope creep is a very real risk that needs to be avoided. In conjunction with limited personnel, feature bloat would cause the project to fall behind schedule and may contribute to not being able to complete. As such, the team must adhere to the baseline established in Section 3.2 and the specifications determined in the Design Plan to be completed at a later time. Preemptive efforts have already been made to ensure that the development is set up prior to the specified date of 9/6/2015. This ensures that any potential problems with environment are addressed early.
* Dependency Failure  
  Strategy – Mitigation  
  Since the project relies heavily on external dependencies, each one provides a single point of failure for the project. As such, in conjunction with the Scope Creep Mitigation Plan, the dependencies have already been tested on the development platform and ensured that they offer the features needed to complete the project. This is elaborated on in Section 5.

## 1.5 Software Engineering

The following section details the methodologies, resources, and procedures the team will use in the production of the project system.

### 1.5.1 Standards and Procedures

A cut down version of IEEE’s Process Tree development methodology will be used to ensure that the elements of the project are completed correctly and in an acceptable time frame. The project will be broken down into 3 subsequent branches containing Primary, Supporting, and Organizational structures.

The Primary Branch will focus on the development, deployment, and operation of the project program. All development will be completed under this classification and further documents will reference the Primary Branch when referring to the technical development.

The Supporting Branch will consist primarily of the documentation and configuration management portions of the project. Any potential issues that arise during the lifetime of the project such as conflict or additional risk assessment and management will be handled through this branch.

Finally, the Organizational Branch will house the management initiatives, process control and management, and user guide specifications.

### 1.5.2 Development Methodology

The software development employed over this project will follow the IEC 12207 Standards of Development. Initially, a high level overview will be drafted that explores the specifications of the project and how this relates to the system. Modular design will be an important piece of the development lifecycle. The system will be broken down into the following coding classes:

* Rendering
* Entity
* Environment
* Genetics
* Reporting

The methods and members of each will be explored as each class is designed via the Design Specifications document. After the design plan has been created for the above modules / classes, coding will commence. Each class will have Unit Tests created for the primary routines that ensure the module works correctly. Finally, a systems test will be drafted to ensure the system meets the minimum requirements of the project specification.

### 1.5.3 Development Resources

* Languages
  + Python
* Libraries and Dependencies
  + PyGame
  + PyBrain
  + Numpy
  + Shelve
* Software Resources
  + Sublime Text 2
  + Python 2.7
  + GitHub Online Repo
  + Microsoft Word
  + Microsoft Excel
  + Microsoft Project
* Online Information Resources
  + Stack Overflow
  + Python Documentation
  + Wikipedia
  + PyBrain.org
  + PyGame.org

## 1.6 Testing Procedures

In order to assess the project’s health over time, weekly status reports and time sheets will be provided by the team leader. These weekly reports will indicate how the project is progressing over the lifespan, and if additional risks are identified along the critical path. Using the Work Breakdown Structure as a baseline metric, we will be able to monitor the project to ensure it remains on schedule and accomplishes the project requirements.

Software will routinely be tested for bugs in development. The design phase of the project will flesh out specifications for each individual module or class. These specifications will provide a more granular approach to testing the program as we can actively monitor for bugs in the code that prevent specification competition, as well as ensure that implementation with additional modules will happen in a way that produces the desired output.

Both static and dynamic testing method will be employed based on the deliverable in question for that period of time. Time permitted, documentation will be submitted for review prior to final submission to ensure all areas of the document are covered and are in line with project specification.

Dynamic Testing will involve creating test cases for a method or member of a class to ensure it performs the desired function. An example may be that if an entity’s’ coordinates intersect that of a toxin, that entity’s health must decrease proportionally to the strength of the toxin and the longevity of interception.

As each class is designed, these test cases will be created for the primary methods that class must perform.

## 1.7 Configuration Management

Every day, changes to the code base and documentation base will be submitted to a GitHub Repository online. This is to provide versioning control over time for use if a version needs to be rolled back. The version number of the project will increment by 1 for each major revision which includes: documentation, class introduction, milestones, or weekly rollover. The secondary version number will increment for each commit to the repository which tracks change over version.

The versioning standard will be Project X.YY. As the project progresses the version number will be tracked along with the corresponding baseline package as determined in Section 3.2. This will provide a means of monitoring progress as a definable metric and correlate it to the actual development of the project over time.

In this way, the team will be able to track change over time in the form of definable change logs that may be used in future implementation phases. This also provides the ability for any additional team members that may be added in the future to follow how the project has progressed, as well as be able to routinely obtain the newest copy of the project documentation and code bases.

In congruence with section 3.2, the GitHub Repository will be created and linked to in the second week.

# 2. Requirements Specification

## 2.1 Introduction

The purpose of this section Is to provide a high level overview of the System Requirements Specification Document, as well as provide a system overview and definitions of key terms used herein.

## 2.2 Purpose

The document will provide the requirements specifications for the Simulated Evolutionary Project ( SEP ) as being completed by Taylor Benner. This document will contain the high level overview of the system, the breakdown of the system hierarchy, the requirements of the classes involved, as well as detail peripheral considerations for review by the stakeholder of the project.

## 2.3 Overview

The SEP is a simulated evolution platform in which the primary interaction method will be a visual monitoring of a member’s progress within a problem domain. The project will seek to visually represent the growth progress of a series of members controlled by neural networks in non-descript pathfinding and simple input / output categorization. The system should be used under the guise of an informational or entertainment based premise in which the observer understands the system is akin to an accelerated ant farm.

An interested party may run the simulation to see how members in a controlled environment learn to navigate the environment under the premise of survival. They may also elect to generate reports based on the progress of the members over time. The member’s progress from generation to generation will employ methods derived from genetic algorithms in which the key components of a member’s ability to survive will be used in the creation of new generations.

## 2.4 Definitions

|  |  |
| --- | --- |
| Member | An entity that exists within the system and is governed by a neural network. |
| Food | An object that when collided with by a member, increments a health value |
| Toxin | An object that when collided with by a member, decrements health value |
| Environment | The object that contains references to all objects |
| Brain | A neural network unique to a member that is responsible for receiving input and processing output |
| Vision | The set of parameters used to determine the distance away from an object and the object type. |

## 2.5 Methodology

The primary method used for requirements analysis during the lifetime of the project will be iterative prototyping. Class based requirements will be expanded on as the code is developed to account for unknown unknowns in the project development lifecycle. Due to the pseudo-agile project management methodology employed, requirements will be expanded on to conform to the system’s overview as the project progresses.

Each major phasic class of the system will be subject to various iterative prototypes. Once the development phase has been drafted and initialized, the requirements of the class will be noted as the initial prototype. The class can then be tested to ensure it meets the specifications of the system as determined in the overview.

In this way, requirements of the classes that are unknown will be discovered and accounted for in the final delivery of the system. When designing the system, actions that a class must perform will be noted as a requirement. Secondary and tertiary support functions will not be labeled as requirements.

## 2.6 Requirements

### 2.6.1 Interface Requirements

On running the program a window should open containing a view screen for the environment and a side panel for statistical readout. There will be no interactive elements as the system is a pre-determined simulation.

The rendering system will draw each member as a circle with a line beginning at the center and extending to the edge of the circle indicating the direction the member is facing. Food will be represented visually by small green circles. Toxins will be represented visually by small red circles.

The side panel readout will contain descriptive information about the environment and population. The information contained includes: population size, generation number, food pellets, toxin pellets, highest fitness, and generation duration.

## 2.6.2 Class Requirements

### 2.6.2.1 Member Class Requirements

2.6.2.1.1 Data storage  
Member data stored every generation iteration to a shelve file

2.6.2.1.2 Calculate distance  
Member should be able to perform a ray cast to determine distance away from an object within a conic view.

2.6.2.1.3 Identify Object  
Member should be able to identify the closest target

2.6.2.1.4 Move member  
Member should have move function that finds the new X and Y coordinate based on rotational angle and velocity

2.6.2.1.5 Rotate member  
Member needs to calculate rotational based on radian change per second which is derived from track differential

3.2.1.6 - Eat food  
When a member collides with a food object, a counter should increment indicating that the member has eaten the food. Health should increase, and the food pellet must be removed.

3.2.1.7 – Eat toxin  
When a member collides with a toxin object, a counter should increment indicating the member has eaten a toxin. Health should decrease, and the toxin should be removed.

3.2.1.8 – Has brain  
The member should store an instance of the brain class that is unique to the member.

3.2.1.9 – Draw member  
Member has a function that renders a circle to screen based on the stored values of radius, x, y, and color. Function should also draw a line from the center of the member to the perimeter indicating rotational angle.

### 3.2.2. Population Class Requirements

3.2.2.1 – Generate members  
When a population is instantiated, the collection of member objects should be initialized.

3.2.2.2 – Perform scoring  
The population class has a function that scores the members based on health times food, divided by toxins.

3.2.2.3 – Perform selection  
After scoring, the population class should sort by score and select the top members as parents.

3.2.2.4 – Perform crossover  
A series of new members is created from a 2 point crossover of the parent’s weights.

3.2.2.5 – Perform mutation  
Each weight has a small change to be initialized as a random value.

### 3.2.3. Brain Class Requirements

3.2.3.1 – Construct network  
Given a shape of 2,5,2, the class will create a neural network using Linear layer input, Sigmoid hidden layer, and Tanh output layer.

3.2.3.2 – Perform reinforcement  
When a favorable action is performed, the neural network must reinforce that action by adjusting the weights to increase the probability of that action.

3.2.3.4 – Perform activation  
When requested, the Brain class will activate the neural network using two inputs mapped to the member’s vision properties. The output will be two values between -1 and 1 which are mapped to the left and right track speeds.

### 3.2.4. Food Class Requirements

3.2.4.1 – Perform initialization  
Object will initialize with two random values between 0 and WIDTH, HEIGHT for placement on the canvas.

3.2.4.2 – Draw object  
Renders the object as a small green circle at the initialized coordinates.

### 3.2.5. Toxin Class Requirements

3.2.5.1 – Perform initialization  
Object will initialize with two random values between 0 and WIDTH, HEIGHT for placement on the canvas.

3.2.5.2 – Draw object  
Renders the object as a small green circle at the initialized coordinates.

### 3.2.6. Database Requirements

3.2.6.1 – Create key  
Generate key for a member using member number, generation number, parent’s numbers, and simulation time. This allows for fast indexing without needing to pull the value data.

3.2.6.2 – Store value  
Store data for a member using food, toxins, and neural network weights.

### 3.2.7. Reporting Requirements

3.2.7.1 – Show optimization  
Generate report or graph showing optimization progress over time as measured by normalized fitness values of the population.

3.2.7.2 – Show optimal weight distribution  
Generate a report indicating the optimal weight distribution per layer as determine via the fittest members from all generations.

# 3. Design Specification

## 3.1 Scope

### 3.1.1 System Description

This document seeks to outline the design specifications for the Simulated Evolutionary Environment program detailed in previous documentation. The system will have limited user interface, and will seek to optimize a neural network controller using a genetic algorithm. The output of the neural network controller will drive a simulated creature’s ability to move through a virtual environment, in parallel to an autonomous robot. Sensory input will be a conic range in which the creature will select the nearest target to itself as the source of the primary sensor data. Object type, angle, and distance will be data transmitted to the neural network controller, and the output will be tied to a member’s left and right movement expressed as values ranging from -1 to 1.

Each member will be allocated a maximum energy potential and each update of the member’s primary attributes will consume 1 energy point. Food objects will increase energy, and toxin objects will decrease energy. If the member falls to zero energy, or increases by two times the maximum energy, the member will die and will be inert for the remainder of the generation.

Once all members of a generation are rendered inert, the population will be scored taking into account time alive, number of food eaten, number of toxins eaten. The top 50% of the population will be selected for asexual reproduction, in which N children will created for that member where N is population count / number of parents. In the case of an odd number of members, the top parent will produce the final children.

In lieu of a traditional crossover function requiring two parents, the crossover function of the asexual reproduction will work as follows. Each weight will have a 50% chance to be scaled by a constrained percentage. Each weight will also have a 5% of mutating to a random value within the constraint of the layer.

### 3.1.2 Major Software Functions

The following section lists the primary functions this system will require.

* **Main Class**
  + Initialization
  + Events
  + Update
  + Render
  + Cleanup
  + Execute
* **Environment Class**
  + Initialization
  + Get Objects
  + Check Collisions
  + Score Members
  + Select Parents
  + Reproduce
  + Mutate
  + Generate Next Population
* **Member Class**
  + Initialization
  + Update Member Attributes
  + Render Member
  + Get Nearest
  + Get Object Type
* **Brain Class**
  + Initialization
  + Build Network
  + Activate Network
* **Target Class**
  + Initialization
  + Draw Target
* **Helper Class**
  + Random Coordinates
  + To Fixed
  + Delta X
  + Delta Y
  + To Radians
  + To Degrees
  + Random Color
* **Database Class**
  + Lookup By Key
  + To CSV

### 3.1.3 Database Description

The database technology employed will be a derivative of a Berkeley Database dictionary file. Functionality will be extended to the program through use of the Shelve Library in python. By using a dictionary structure, we can encode certain amounts of data via a key, and the bulk raw data as the value associated with the key.

The key will composed of the following parts:

* Generation Number – The generational number the member was initialized in.
* Member Number – The order in which the member was created in the generation.
* Lifespan – The amount of time the member was alive as expressed in milliseconds.

The value of each entry will store the primary data in a list-object formation that includes the following data:

* Weights – The weights of the neural network stored as an array going from Input 0 to Output 2
* Topology – Time permitting, the topology of the neural network will stored as an encoded genome.
* Targets – A count of each target the member interacted with. Sub classifications for food and toxins will be included.

### 3.1.4 Design Constraints and Limitations

The database technology being used is Object Oriented storage and as such employs the NoSQL paradigm of data storage. Primary and Foreign Key Constraints and Indices are not applicable to this technology. The tradeoff is that the data is less accessible outside the program. As such, a CSV generation function will be included that allows each dictionary entry to be expressed as a row in a CSV file.

## 3.2 Design Description

### 3.2.1 Data Description

The program uses primarily class specific members. Each major component of the program is represented by a loosely coupled class object. The objects, when instantiated, create a series of publically accessible variables that are encapsulated within the class. Members defined in the initialization function of a class are limited in scope to that instance, whereas members defined prior to the initialization function are shared by all instances of that class.

These attributes are accessible within the scope the class is accessible. This is a feature of the python language. Furthermore, inherited methods within the scope of a class are considered virtual attributes and may only perform operations within the lexical scope of the class.

There are a few globally accessible variables in the spiritual flavor of global constants. These global variables are defined in the beginning of the source and are expressed as all capital names. These are globally modifying attributes such as environment width, height, and debug parameters.

The data majority will be class objects with instance members as floats. Calculations are performed on list objects of 2 to 7 elements. Floats are limited to an inherit 8 points of precision.

### 3.2.2 Data Flow



### 3.2.3 Program Architecture



### 3.2.4 Component Interfaces

This program does not require a high level of breakdown beyond the standard class models already described. Each class can be considered a baseline package with the components being the members and methods that make up that class. Detailed below is the relationship between the classes.



A target and member should never have to interface directly as they are processed independently via the Environment class. Both members and targets are added, removed, and compare at the Environment level to isolate structural elements to each class. In this way, the properties of both targets and members may be changed without impacting the program as a whole.

The Main class will coordinate routine as well as data storage and report generation. Publically accessible members in the Environment class will be used by Main to be stored and retrieved as needed. The report class will generate a CSV file output as well as a visual graph using Matplotlib Library with the data sent to it by Main.

The Brain class contains all structural logic behind the neural networks used by the member. This is to keep the topography and weight structure loosely coupled to the Environment so it can be changed within the Brain class to impact how the Member behaves.

Finally the Main class interfaces directly with the viewport, not shown above. The members and targets have their own draw method which is referenced through an Environment function. The Main class holds the display object which PyGame represents visually.

# 3.3 Detailed Design

### 3.3.1 Processing Description

The processing method employed will be a continuous loop until the PyGame Object is destroyed via a quit event. This routine will be managed by the *execute* function in which a While Loop checks the boolean flag *Running.* On each iteration of the primary game loop, the system will perform an update and render pass. The update function calls ass needed update functions sequentially, and checks to make sure each member is still alive. If all members are dead, the genetic algorithm is activated and a new population is generated.

The update function is defined by the following logic. First, take a time snapshot and set the active flag to false. For every member, check if that member is alive. If they are, update their position, update their state, process collisions with targets, and set active to true. Then update the state of all targets. Afterwards, check if active is still false. This means all members are dead. If active is false, score the members, select the fittest, breed them using crossover, perform mutations, and increment the generation number. Finally, find the time difference and save the value if it exceeds the maximum processing time for this function.

After update has run, the loop will perform all required renderings. This function first fills the screen with white, then iterates over every object with a draw function. The display object is passed to the draw function and the object renders tits components to the display. A label is added for the generation count, and the display is flipped to display the new renderings.

When a member is updated, we perform the following processing flow. First, update the position by taking a time snapshot. We check for the inclusion of a close by object as determined in the collision detection later on. If the member is close to an object, calculate the distance and reference angle and get the object code. Otherwise set these values to 0. Then we activate the neural network passing in the member’s position, distance to target , angle to target, target code, and the member’s energy amount. The output of the neural network is assigned to the left and right tracks. Then we calculate the radian change from the difference of the tracks divided by the member’s radius. Velocity is then calculated using the sum of both tracks. Based on the radian change and velocity, we calculate the delta of the X and Y values using sin and cosine respectively added to the current position. If the delta of the X value plus the radius is beyond the view limits, we set the new X to the corresponding limit ( 0 or WIDTH ). We do this as well for the Y value. Otherwise, we set the X and Y value to the delta values and the member moves in the appropriate direction. We end with calculating the time difference from the snapshot and setting to the maximum processing time for this function.

The update state function decreases the member’s energy and checks the alive state and sets it accordingly. If the member is less than 0 energy, they are listed as dead. If the member’s energy is greater than 2000, they have overfed and are dead. If the member is dead, we clear the close to object, capture the time of death, and process the lifespan of the member for scoring purposes. This function is also timed for processing bottlenecks.

The check collision function is the next to be activated. We pass in a list of targets and compare the member’s position against each of them in iterative passes to reduce processing overhead. For each target passed in, we check to see if the target’s X and Y values are within a tolerance range of 10 times the member’s radius. If the target is close to the member, we check to see if it is the closest, and capture it if it is. Then, we perform an absolute collision check by checking the left, right, top, and bottom edges of the member against the left, right, top, and bottom edges of the target. If the hit boxes overlap, we consider a collision to have taken place and we mark the target for removal. We also increase the member’s energy amount and target count. This function is also timed for processing bottlenecks.

The draw function of the member first determines the color and stroke of the member. If the member is dead, we’ll use a grey color to indicate as such. Delta X and Delta Y are calculated using the member’s radian amount and radius to draw the internal line indicating direction. One circle is drawn with the member’s color, another is drawn on top of that to provide an outside stroke, and finally the directional line is draw. If DEBUG is enabled, we also draw the hitbox for the member. The energy bar for each member is then processed and drawn if the member is alive. We do this by defining the width and height of the bar. Then we calculate the percentage of the bar that is filled based on the percentage of energy vs max energy the member has. The fill amount is drawn followed by the bar itself over top to give the appearance that the bar is filling or draining over time. If DEBUG is enabled, we also draw a line to the closest target.

The member contains an instance of the brain class. When the brain is initialized, it builds a new network if none has been provided. The network is built using the following topology. An input layer is constructed using the sigmoid activation functions to normalize the input. A hidden layer is created using a Gaussian distribution followed by a Linear distribution layer. The output is a sigmoid layer to ensure members move in a smoother fashion. We then add each module to the network as well as a bias unit. Connections are then added between the layers, and the bias unit is connected to the output layer. Finally, we sort the modules and return the network for assignment to the instance of the Brain class.

When all members of a population are dead, we begin the next generation process. First, we call the score members function in the Environment class. Out fitness function calculates member lifespan times food less toxins consumed. The score is stored with the member. All scores are then totaled and normalized to within a range of 0 to 1. The sum of all normalized scores should be 1.

After scoring the members, we perform selection on them. Current this function sorts the members by score in descending order and selects the top half of the population. These members are stored as parents, and the members list is cleared out.

In the crossover function we calculate the number of children each parent must produce. We then instantiate new members for each parent, with a 5% chance to generate a completely random child that does not inherit. We pass the network and color to the new member function for inheritance.

Finally we perform mutation on each new member. This iterates over each weight in the network and has a 5% chance to random reinitialize it between -2 and 2. The function also finds a weighted mix of the colors to visually represent mutation.

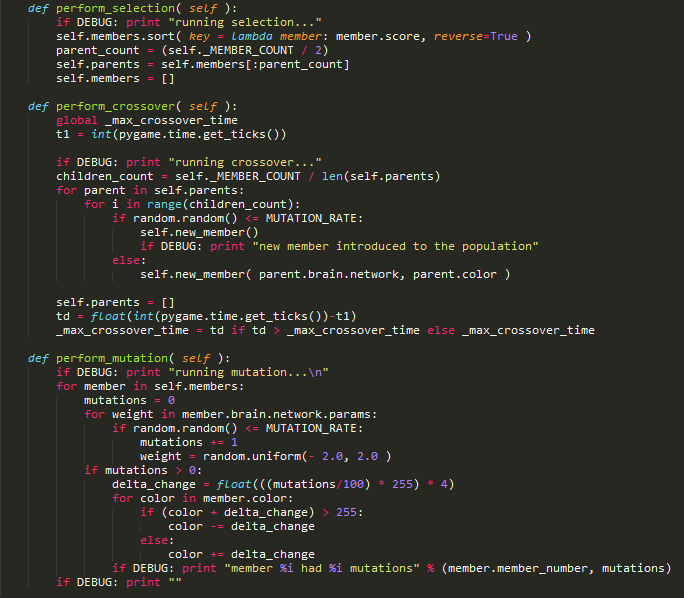
The above process chain is repeated as many times as desired within the system.

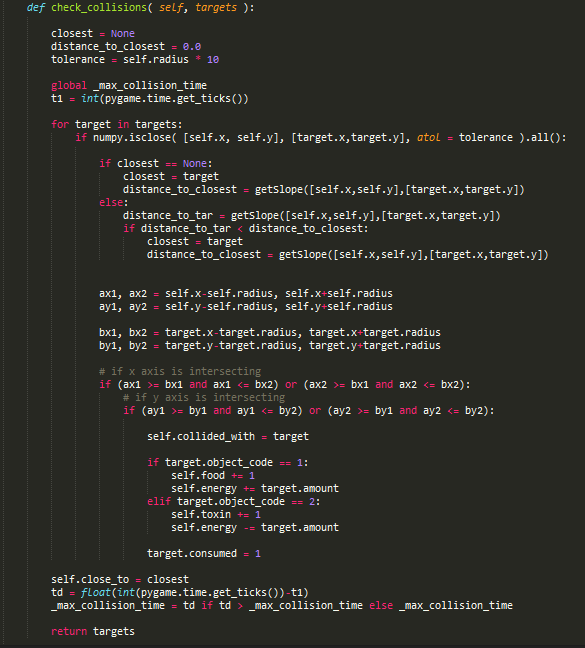
### 3.3.2 Interface Description

The main class stores an instance of the Environment class. The Environment class stores lists for instances of the Member class, the Target class, and references to parents as a temporary list. The Environment class has functions to instantiate new members and targets, and passes an object number and object type to the new instances. The member class receives a pointer to a list of target when processing collisions and can modify the list as a temporary object, but passes the modified list back to the environment for further processing and assignment. The member instantiates a copy of the brain class when created.

### 3.3.3 Pseudocode

As I have already written a large portion of the code, I’ll include key portions here as an iterative approach to fine tuning the code.







### 3.3.4 Modules Uses

2.1.4.1 – Init

This function is responsible for setting the initial view state and initializing an Environment class.

2.1.4.2 – Events

This function interprets the PyGame Events and reacts accordingly.

2.1.4.3 – Update

This is the primary function of the main game loop. This function calls several update functions for the classes involved in the program and contains the primary logic for the order of processing. This function is timed to identify bottlenecks in debug.

2.1.4.4 – Render

All draw routines and visual aspects are processed in this function.

2.1.4.5 – Cleanup

Called on quit, this function trashes the PyGame object and prints debugging information.

2.1.4.6 – Execute

This is the game loop controller. It calls the above functions in sequence to ensure the game remains running. The clock is incremented here to control game loop speed.

2.1.4.7 – Get Object

This function returns a concatenated list of all members and targets for iterative processing.

2.1.4.8 – Get Targets

This function returns a concatenated list of all food and toxin objects for iterative processing.

2.1.4.9 – New Member

Accepts optional network and color parameters. This function initializes a new member class and passes the optional arguments to the member \_\_init\_\_. Keeps track of how many members are created.

2.1.4.10 - New Food

This function creates a new target instance with food like attributes.

2.1.4.11 – New Toxin

This function creates a new target instance with toxin like attributes.

2.1.4.12 – Score Members

This function processes the fitness function for all members and normalizes the values to percentages of 1. All normalized scores should add up to a total of 1 to indicate the scoring is being calculated correctly. The fitness function currently employed is lifespan by food minus toxins.

2.1.4.13 – Perform Selection

This function sorts and selects members for breeding based on the score of each. Currently the top 50% of the members are used.

2.1.4.14 – Perform Crossover

This is the breeding function for the members. Currently an asexual breeding method is used in which each parent splits to an even distribution of the total population size. Each new member has a chance to be completely randomized.

2.1.4.15 – Perform Mutation

This function iterates over each weight in a member’s neural network and has a chance to randomly set it to a value between -2 and 2. The number of mutations is kept track of and is used in modifying the member’s color for a visual distinction.

2.1.4.16 – Update Targets

This function removes consumed targets from the target list and initializes new targets for any consumed.

2.1.4.17 – Draw

Any and all draw methods across the member and target classes simply render the object given the parameters defined.

2.1.4.18 – Update Position

This function performs all primary calculations for the member instances. Distance and angle to a nearby target are calculated and passed into the activation function of the member’s brain instance along with member position, energy, and object code. Rotational amount is determined by the difference between the left and right tracks, divided by radius. Velocity is determined from the speed of both tracks. If the new position is beyond the limits of the viewing area, the member is prevented from moving in that direction. This function is timed to debug bottlenecks in processing.

2.1.4.19 – Update State

This function sets energy and alive status after updating the member. It also captures death time and lifespan when a member runs out of energy. This function is timed to debug bottlenecks in processing.

2.1.4.20 – Check Collisions

This temperamental function is a key function to the success of the simulation. The function iterates over each target for each member and performs two collision passes. The first checks for objects within ten times the radius of the member. If a target is close, the distance to the target is calculated and stored if smaller than the current minimum. For any object that is within the defined range, an absolute collision check is performed. Each edge of the target is compared to each edge of the member to find intersecting X and Y points. If a collision occurs, the target is recorded, member energy is adjusted, and the target is marked for consumption. This function is timed to debug bottlenecks in processing.

2.1.4.21 – Build

This function is responsible for constructing the neural network for a member. The shape, layer types, and connection types can all be adjusted to fine tune the topography of the network.

2.1.4.22 – Activate

This function passes a list of parameters into the neural network and returns the output.

2.1.4.23 – Calculate Contrast

A helper function for determining the lightness of a color in order to adjust the color of the lines used.

2.1.4.24 – Random Color

Returns a randomized RGB value as a list

2.1.4.25 – Random Coordinates

Returns a random position within the bounds of the viewing area with consideration to a radius.

2.1.4.26 – To Fixed

Rounds a number and returns as an integer type. Useful for converting floating point values to screen coordinates.

2.1.4.27 – Delta X

Returns the sin of an angle by a number. Used in calculating change in X.

2.1.4.28 – Delta Y

Returns the cosine of an angle by a number. Used in calculating change in Y.

2.1.4.29 – Get Slope

Returns the length of the hypotenuse between two points. Used to calculate distance.

2.1.4.30 – Get Angle

Returns the arc tangents of two points. Used to calculate angle of change as radians.

# 4. Test Plan

# 5. Application Files

# 6. Project Presentation

# 7. Deployment Plan

# 8. Peer Review