Design Document
EVOLUTION WILDFIRE

Nawar Ismail

September 7th, 2019

Contents

List of Source Code Listings

List of Figures

Part I

Project Organization and Workflow

Chapter 1

Overview

The purpose of this document is to formalize, justify, and explain the design of this software. The associated code is primarily concerned with the evolution of entities in a 3D environment. The decisions laid out in this document are to justify their design with the aim of usability, extendability, and maintainability.

1.1 Context

The evolutionary algorithm is exciting due to the relatively unpredictable behavior, while being able to solve difficult problems. On top of this, neural networks have also become increasing popular with their applications ranging across almost every technological domain. Thus a combination is surely worth exploring. Despite this, there are few examples where these are combined in the evolution of entities in a 3D environment, with sufficient interaction from the user. Thus, this software finds its place there.

1.2 Goals and Non-Goals

The goal of this software is not exactly pinned down. It may end up as a simulator or a game. The development so far would lend itself well to controlling a single creature in a competitive environment. Possibly the user could guide the evolution, or unlock upgrades / augmentations as their creature completes tasks / progresses. Once the prerequisite problems have been solved, more consideration would be added. Regardless, real-time evolution is an important aspect.

There are several tools, algorithms, theories, and techniques that are required to get all the machinery working. As a result, this provides a good opportunity to learn about physics, machine learning, and software design. Although not a primary focus, knowledge of 3D graphics is required to create a visually appealing environment. Other considerations like audio could also be implemented. These are largely considerations for the future, since the graphics will likely be ported over to Unity when a base use has been established. Far in the future, online play with a database, levels etc. may be added.

Chapter 2

Milestones and Events

Description	**Start-End**
First Version of Design Document	Sept 7$^{\text{th}}$, 2019

Chapter 3

Dependencies

Part II

Machinery

Chapter 4

Overview

In general, there are two approaches to evolve creatures, what we will call *Instance Evolution*, and *Continuous Evolution*. Instance evolution takes place on a single creature and progress occurs in discrete iterations. By contrast, Continuous Evolution simulates multiple creatures simultaneously resulting in a more realistic continuous evolution. The former has a more straightforward implementation (including parallelization) and is expected to improve faster. The latter however greatly reduces computational cost for large scale simulations. This has not yet been implemented, but may be when creatures can detect and react to other creatures. The following will only discuss instance evolution.

There are several components that work together to allow for high configurability and expandability. The simplest component is the **gene**. These encoded the properties of a component of a **creature**. The set of these are packaged into a **genome**, which in turn can be decoded into a physical entity according to a **body**'s interpretation of the genes, resulting in a **creature**. This is sufficient to create an entity but to simulate and evolve it, it requires both a **fitness** metric to strive for and [This is a poor design and should be directly connected to the Evolution Algorithm] an **environment** that it may interact with, in what is referred to as a **scenario**. Multiple scenarios, contained in a **population** are required to be simulated for the **evolution algorithm** to be used effectively. Finally, the **user** may interact / view aspects entities in the population through the **Viewer** class. An overview of these components can be found in Fig. 4.1. They will be individually discussed in the following sections.

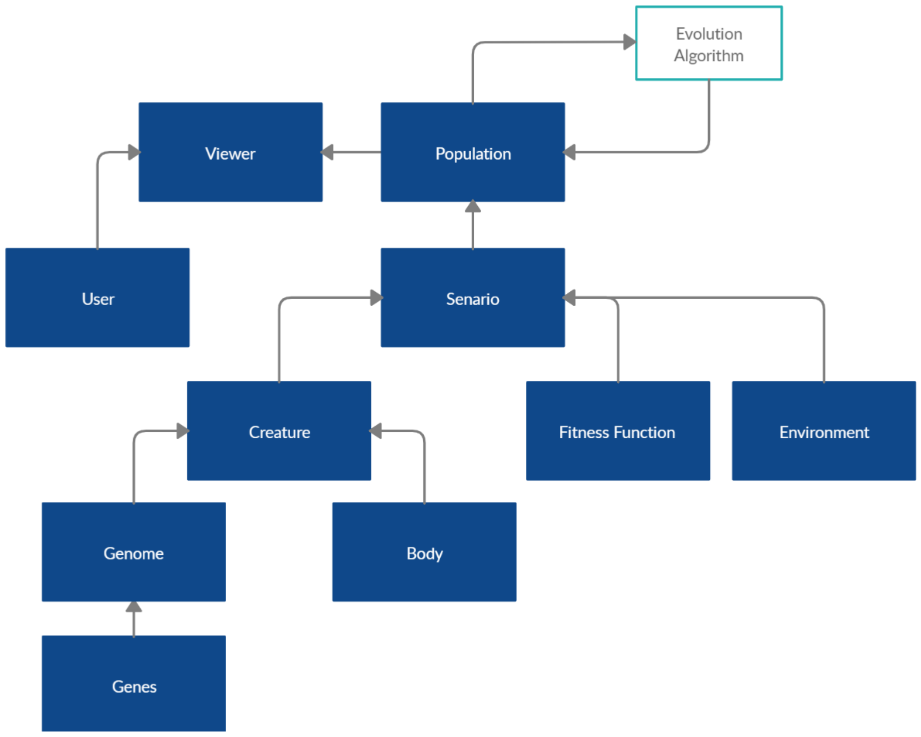

Figure 4.1: The machinery that allows the user to view the evolution of a set of genes expressed by a body in a given environment according to a given fitness function. Opaque boxes correspond to classes, while clear boxes correspond to procedures.

4.1 Genetics

4.1.1 Genes

A gene is very simple as it only contains a set of attributes representing the properties of a component of a creature. They may be read to/from a string. Every gene has a unique (single-character) symbol associated with it, so that the string representation is uniquely identified. Several genes are already implemented:

```
Gene
├─NodeGene ..............Vec3 position, double mass, double friction
├─BoneGene ..............................................int a, int b
│ └─MuscleGene ...........................int a, int b, double speed
├─CubeGene ................................Vec3 position, double mass
└─AxonGene ...................int a, int b, int layer, double weight
```

4.1.2 Genome

A genome is a collection of genes. It can be written to/from a string. It can also be mutated, resulting in potentially altered genes. The usability of this is relatively low and currently is manually controlled.

4.1.3 Open-Questions

- The single-character notation may quickly cause expressiveness to degrade as name collisions are avoided. A string symbol would avoid this.
- How should mutations for different creatures be managed? Most likely each `body` should be in control of this.

4.2 Physical

4.2.1 Body

4.2.2 Creature

4.2.3 Environment

4.2.4 Scenario

4.3 Algorithm

4.3.1 Population

4.4 Viewer

4.5 Special Details

The evolution algorithm is multi-threaded.

Part III

Physics

Chapter 5

Diffusion

Chapter 6

Vision

Part IV

Discoveries