Evolutionary Simulation: Unity-Based Genetic Algorithm

Project Documentation

December 20, 2024

Contents

Chapter 1

Introduction

1.1 Motivation

This project theme was chosen partly because of my previous experience with creating unity applications, as well as the ability to easily represent the individuals and their characteristics in a visual manner by using the build-in mesh renderers that Unity provides.

1.2 Project Objectives

The primary objectives of this project are:

- Model basic evolutionary mechanisms such as reproduction, mutation, and survival
- Demonstrate how genetic traits influence the survival of the individual and population

Chapter 2

Theoretical Background

2.1 Genetic Algorithms

Genetic algorithms are optimization techniques inspired by Charles Darwin's theory of natural selection. They simulate the process of evolution by:

1. Generating an initial population with diverse characteristics
2. Evaluating individual fitness
3. Selecting individuals for reproduction
4. Applying crossover and mutation
5. Replacing the previous generation

2.2 State of the Art

WIP

- WIP

Chapter 3

System Design

3.1 Individual Representation

In this application each individual is represented as a unity game object with its own independent script, the genes used to define the traits of the individual are:

3.1.1 Genetic Traits

- **Speed:** Movement velocity (range: 0.1 - 60.0)
- **Size:** Physical dimensions (range: 0.5 - 6.0)
- **Vision Range:** Detection radius (range: 1.0 - 60.0)
- **Wander Time:** Duration of random movement cycles

3.1.2 Energy System

The energy management system includes:

- Base energy level
- Energy threshold for reproduction
- Energy stopping threshold for mating
- Metabolic costs based on traits

3.1.3 State Visualization

The application implements a color-coding system which makes it easy to determine the state that an individual is in:

- Red: Low energy
- Green: High energy
- Blue: Seeking mate prey
- Magenta: Hunting prey

3.1.4 Generational Tracking

Each individual displays its generation number, allowing for:

- Easily tracking how many generations the individuals stems from
- Analysis of survival rates

3.2 Behavioral Model

3.2.1 Action Priority Hierarchy

Individuals will attempt to fulfill the following, these actions are ordered based on priority from highest to lowest:

1. Flee from predator (if predator is detected)
2. Hunt prey (when energy levels are sufficient and prey is found)
3. Seek mates (when energy threshold is met)
4. Move towards food (if food is within range)
5. Wander randomly

3.2.2 Predator-Prey Dynamics

The system implements size-based predation:

- Predators can hunt individuals 1.5 times smaller
- Prey flees from individuals 1.5 times larger

3.2.3 Mating Behavior

Reproduction is governed by:

- Energy thresholds for initiating and stopping mating behavior
- Size compatibility checks between potential mates
- Energy distribution between parent and offspring

3.3 Fitness Function

The fitness of individuals is determined through multiple factors:

- Energy acquisition efficiency
- Predator avoidance success
- Hunting success rate
- Reproductive success

3.3.1 Energy Dynamics

Energy consumption code:

```
private void ConsumeEnergy()
{
  float energyUsage = baseEnergyUsage + (visionRange *
     visionMultiplier) + (speed * speedMultiplier) + (
     size * sizeMultiplier);
  energy -= energyUsage * Time.deltaTime;

  if (energy <= 0)
  {
    SpawnFood();
    Destroy(gameObject);
  }
}
```

Chapter 4

Implementation Details

4.1 State Management

4.1.1 Individual States

Each individual has multiple states that it can be in, the state is choosen based of factors such as proximity to prey and energy level:

```
private bool isLookingForMate = false;
private bool isHunting = false;
private GameObject currentTarget = null;
```

4.2 Reproduction System

4.2.1 Genetic Inheritance

The reproduction system implements controlled mutation:

```
private void Reproduce(Individual mate)
{
Individual child = Instantiate(gameObject)
.GetComponent<Individual>();
child.speed = Mathf.Clamp(speed + Random.Range(-
    mutationStrength, mutationStrength),0.1f,60f);
// Rest of traits
}
```

Chapter 5

Results and Analysis

HERE BE DRAGONS
MORE TO COME