High Level Design 1.0

Coop Simulator

#### Tolga NAKIBOGLU

[1. Introduction 3](#_Toc520281971)

[2. Scope 3](#_Toc520281972)

[3. Goals and Objectives 3](#_Toc520281973)

[4. Design Details 3](#_Toc520281974)

[5. Class Diagram 6](#_Toc520281975)

[6. Future Scope 6](#_Toc520281976)

# Introduction

**This document describes the high level architecture for the Coop Simulator project. The requirement is to simulate population growth process of any breed of animal in a coop environment. It will provide configuration parameters to customize the dynamics of the population growth, particularly based on the target breed specifications. It will get input as the simulation duration in months and will provide output as total counts of animals, male, female and the dead for that duration.**

# Scope

**Only “Rabbit” as Animal breed is supported in current release, though the design supports implementation of new animal breeds for future scope.**

**Coop Simulator targets to simulate population growth in more generic than specific approach.**

**Coop Simulator focuses on the basic population growth aspects such as:**

**- Average life time in coop environment**

**(Animal life expectancy may vary base on wild life and coop environment)**

- Average sexual maturity age

- Female pregnancy length

- Average new born count

**Coop Simulator does not try to simulate:**

**- Environmental and genetic Health factors**

**- Predator interaction**

**- Micro details of specifications of the target breed**

**(Sexual behaviours based on environment/age/sex etc...)**

**- Breed specific Social interactions**

# Goals and Objectives

**Population growth simulations naturally require optimized performance management as they are much resource dependant (both memory and CPU); Coop Simulator targets to provide high performance by using multi-threading, concurrent collections and a proper intended design.**

# Design **Details**

* **Main operation is handled in** Simulation **class.** Simulation **class has three concurrent collections which keep all animal instances during entire simulation process.**
  1. ConcurrentBag<IMale> availableMales:

**This collection keeps sexually available male animals. ConcurrentBag is used as order is not important when trying to fetch a male for mating.**

* 1. ConcurrentBag<IFemale> availableFemales:

**This collection keeps sexually available female animals. ConcurrentBag is used as order is not important when trying to fetch a female for mating.**

* 1. ConcurrentDictionary<int, ConcurrentBag<ProcessedAnimal>> processedAnimals:

**This collection keeps the Animals which are stored with a process type for future execution. The “key” of the dictionary represents the next simulation cycle (in days) that the stored process will execute. There are three defined simulation process types:**

* + 1. ***SexuallyImmature***

**Animals which are sexually immature, unable to breed (from birth till it becomes sexually mature), waiting to get sexually mature at target execution cycle.**

* + 1. ***Unavailable***

**Animals which are sexually mature but unavailable for sexual interaction, waiting to get available at target execution cycle.**

* + 1. ***Pregnant***

**Female Animals which are pregnant, waiting to give birth at target execution cycle.**

* SimulationHelper **class handles the simulation operations that are invoked from** Simulation **class.**
* Simulation **class iterates the simulation cycle daily till the end of the simulation duration which was provided by the user. There are three main operations that are handled at** SimulationHelper**:**

1. ***HandleCurrentProcessedAnimals***

SimulationHelper **iterates given** ConcurrentBag<ProcessedAnimal> **collection:**

**For “SexuallyImmature” process type:**

SimulationHelper **calls** OnSexuallyMature **method of** Simulation **class informing that the time has come for the target animal to become sexually mature. When an animal becomes sexually mature, it is removed from** processedAnimals **and added to** availableMales/availableFemales.

**For “Unavailable” process type:**

SimulationHelper **calls** OnAvailable **method of** Simulation **class informing that the time has come for the target animal to become available for sexual interaction. When an animal becomes available, it is removed from** processedAnimals **and added to** availableMales/availableFemales.

**For “Pregnant” process type:**

SimulationHelper **calls** OnBirth **method of** Simulation **class informing that the time has come for the target female to give birth. When a female gives birth, the new-born animals are added to** processedAnimals **with the process type of “SexuallyImmature”**

**All call-backs to** Simulation **class are handled as asynchronous operation (Tasks) and HandleCurrentProcessedAnimals method waits till all the Tasks are completed in order to synchronize the simulation state.**

**In order to maximize the performance and reduce the overhead of running a new Task per operation, a predefined number of operations are added to lists and given to Tasks. The number comes from configuration parameter simulationOperationCountPerTask)**

1. ***HandleAvailableAnimals***

SimulationHelper **iterates given** ConcurrentBag<IFemale> availableFemales **collection:**

SimulationHelper **tries to fetch a female from** availableFemales. **If a female is found then** SimulationHelper **tries to fetch a male from** availableMales. **If couple is found for sexual interaction,** SimulationHelper **calls** OnMatingPartnersFound **method of** Simulation **class informing that a male and a female is ready for a sexual interaction.** Simulation **class calls** Mate **method of** SimulationHelper, **if mating succeeds female gets pregnant and gets added to** processedAnimals **with the process type of “Pregnant”**, **otherwise it gets added to processedAnimals with the process type of “Unavailable”.** **In either case male gets added to** processedAnimals **with the process type of “Unavailable”.**

**All call-backs to** Simulation **class are handled as asynchronous operation (Tasks) and HandleAvailableAnimals method waits till all the Tasks are completed in order to synchronize the simulation state.**

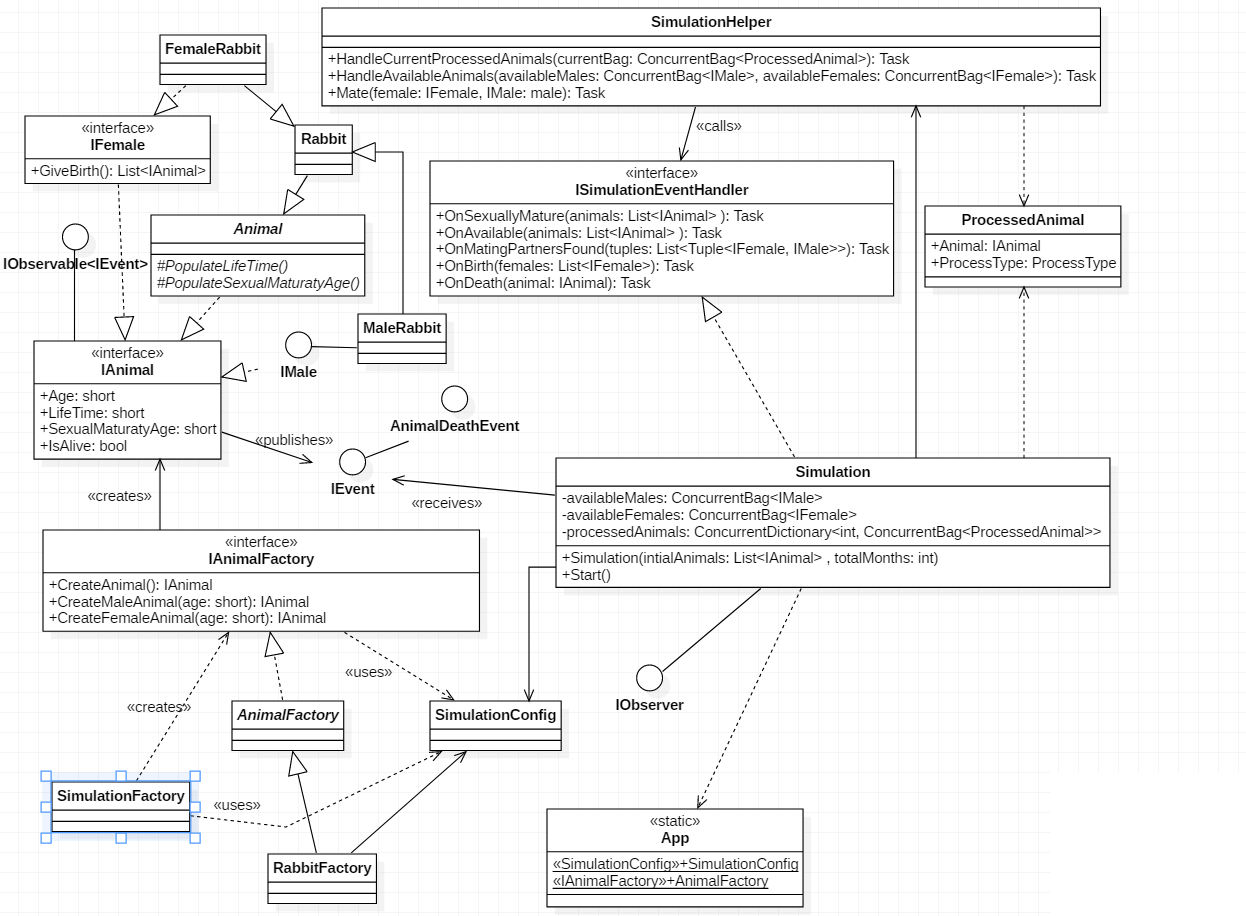
**In order to maximize the performance and reduce the overhead of running a new Task per operation, a predefined number of operations are added to lists and given to Tasks. The number comes from configuration parameter *simulationOperationCountPerTask*)**

1. ***Mate***

SimulationHelper **handles mating operation of a couple with a pregnancy success chance of 50% (this can be handled more realistic in future scope)**

* **Every Animal object** **(which implements** IAnimal **interface) publishes AnimalDeathEvent ,** Simulation **class subscribes for these events and tracks animal deaths. (for statistics purpose in current release)**

# Class **Diagram**



# **Future Scope**

* Logger implementation
* Exception handling
* Unit tests
* Integration testing
* Performance optimization
* More detailed simulation output can be provided as file.