Population Dynamics Simulation

Software Design Document

Prepared by:
Simay Cural, Na'ama Nevo,
James Settles, Walt Jones

Release Date
September 12, 2022

Executive Summary **3**

Document Versioning **4**

Project Description **5**

Features **6**
Feature Matrix 7
Feature Discussion 8
P.1 - Language 8
P.2 - Bundling 8
D.1 - Encryption 8
L.1 - Simulation Settings GUI 8
L.2 - Simulation Execution GUI 8
L.3 - Creature Settings GUI 9
T.1 - Error Handling 9
C.1 - Creatures 9
C.2 - Interactions 9
C.3 - 2D World 9
C.4 - Statistic Tracking 9
C.5 - Population 9
C.6 - Food 10
C.7 - Creature States 10
C.8 - Real Time Graphs of Population Sizes 10
C.9 - Creature Shade Correlates to Health 10
C.10 - Genetic Changes 10
C.11 - Mutations 10

System Design **10**
Architecture Overview 10
High-level System Architecture 11

Major Components **11**
GUI Components 11
Population Components 11
Creature Components 11
Food Components 11
SimulationEngine Components 11

Detail Design **12**
GUI Components 12
MapGUI 12
SettingsGUI 12
Population Components 13

PopulationInstantiator 13
Population 13
Creature Components 13
CreatureBuilder 14
Creature 14
InteractionMediator 14
Interaction 14
AI 15
States 15
Food Components 15
FoodBuilder 15
Food 16
SimulationEngine Components 16
SimulationEngine 16
Data Formats **16**

Executive Summary

PDS is a customizable population simulation engine focused on the educational market. PDS is intended to be an easy-to-use alternative to more complex population simulating software like HexSim and SPLATCHE2. A freemium monetization model is to officially be supported by PDS. PDS will be distributed as a standalone application without network connectivity. This document provides nontechnical information regarding the purpose and behavior of PDS.

Document Versioning

Date	Owner	Comment
9/6/2022	Simay Cural	Created document outline, updated feature matrix and discussion, Detail Design descriptions
9/7/2022	Na’ama	Updated features discussion, Data Formats, Detail Design descriptions
9/7/2022	Walt	Added high level diagraming
9/7/2022	James	Expanding upon different GUI types
9/7/2022	James	Correcting and expanding upon descriptions of features relating to genetics
9/12/2022	Na’ama	Added design descriptions
9/12/2022	Walt	Added updated UML diagrams & descriptions

Project Description

PDS is simulation software that will allow the user to create various types of population simulations. The user will be able to create populations of modifiable creatures and see how they interact. The end user will not need to modify any code to run the simulation, and all inputs will be managed in a graphical user interface running on the user's local machine.

PDS will not require other programs or network connectivity to be fully functionable and will operate as an executable file. PDS will implement a full set of statistical features for educational purposes, such as multiple runs, saving and loading parameters, and visualized results of simulations in the form of graphs, charts, or images. PDS will be multiplatform, and will support different hardware configurations. The simulation runs will ideally be computationally fast and as efficient as possible. PDS will also implement a random parameter generator so that the user doesn't have to specify parameters if they do not want to. PDS will support further development of parameters for populations.

Features

The feature matrix enumerates the technical features required to support each of the business requirements. The discussion section provides details regarding the constraints and functionality of the feature. The ids are used for traceability. Features that can be removed should strike-through the feature id and have a comment added to identify why this feature can be removed without impacting the BRD requested functionality..

Priority Codes:

H - High, a must have feature for the product to be viable and must be present for launch
M - Medium, a strongly desirable feature but product could launch without
L - Low, a feature that could be dropped if needed

Feature Matrix

ID	Pri	Feature Name	Comment	BRD ID
P.1	H	Language	Code written in Java	s.1
P.2	H	Bundling	Bundling done in JAR file	s.4, s.5
D.1	L	~~Encryption~~		
L.1	H	Simulation Settings GUI		ux.1
L.2	H	Simulation Execution GUI		ux.1
L.3	M	Creature Settings GUI		ux.1
T.1	H	Error Handling		e.4
C.1	H	Creatures		e.1
C.2	H	Interactions		e.2
C.3	H	2D World		e.3
C.4	L	~~Statistic Tracking~~		e.5
C.5	H	Population		
C.6	H	Food		

ID	Pri	Feature Name	Comment	BRD ID
C.7	H	Creature States		
C.8	L	~~Real time graphs of the populations~~		e.5
C.9	M	~~Creature Shade Correlates to Health~~		
C.10	M	~~Genetic Changes~~		e.2
C.11	M	~~Mutations~~		

Feature Discussion

P.1 - Language

PDS must be multi-platform and run as a standalone application. Several programming languages could potentially be used; Python, JavaScript, JAVA, etc. While Python is cross-platform, there are challenges in bundling the code and getting consistent GUI behavior. JavaScript is widely available on many platforms but suffers from similar bundling issues and typically runs in a web browser (which is a problem for the target embedded market). JAVA has cross-platform support, includes capabilities for making executable bundles, has consistent GUI behavior across platforms

P.2 - Bundling

JAVA's built in JAR technology enables bundling of executable code with other assets. For many operating systems, JARs are executable by double clicking on the JAR file. Distributing games as JAR files also has the advantage of adding a step for those trying to improperly modify or reverse engineer game assets.

D.1 - Encryption

~~Game data (files describing the game as well as save files) should be encrypted to reduce the ability of end users to improperly modify or reverse engineer game assets. The specific encryption algorithm used does not need to be strong, however the key used for game files should be based on a file within the JAR rather than a fixed number used by all instances of the TAP engine. Save files should be encrypted using a key provided in the root AL file.~~

L.1 - Simulation Settings GUI

JAVA supports several UI toolkits; AWT, Swing, JavaFX. AWT is not sufficiently feature rich and has limited ability to produce a consistent interface across platforms. JavaFX adoption has been slow/uneven and the competing JavaFX versions have poor interoperability. Swing has been selected as the GUI framework since the behavior is consistent across platforms and the API is stable across versions. Using Swing, this GUI will allow users to change different settings for each simulation, affecting things like population sizes and map makeup.

L.2 - Simulation Execution GUI

Using the Swing UI toolkit, this GUI will display the simulation as it executes. This creates an animated representation of the creatures that updates overtime.

L.3 - Creature Settings GUI

Using the SWING UI toolkit, this GUI will be selected from the Simulation Settings GUI, allowing the user to change the makeup of the creatures in a selected population.

T.1 - Error Handling

If the simulation reaches an unsafe state, error information should be provided to inform the end user that an error occurred. The user should then be presented with an option to have PDS produce a more detailed error report that could be sent back to the game developer as part of a game bug report.

C.1 - Creatures

A fundamental part of PDS is to establish the creatures. The user should customize their creatures in the GUI prompts once the simulation is run. Customization should include what the creature should look like in the simulation (user should be able to pick shape, color, picture), their movement patterns, the aggression rate (towards food, prey, etc.).

C.2 - Interactions

Interactions should encapsulate the behavior between objects. There should be 3 possible interactions: between the same species, between different species, between a creature and food. When two of the same type of creature collide in the simulation, they may reproduce and create another creature. When two creatures of two different types collide, one may eat the other if they have a predator-prey relationship. The outcome of these interactions should change the state of the creature.

C.3 - 2D World

The 2D world should be the board where all of the populations conduct their behavior. The world should include all of the food, predator, and prey and should update the conditions in real time.

C.4 - Statistic Tracking

In the statistic tracking files, once a simulation is done running, PDS should output a save file that contains information on the creature parameters, run time, population sizes, etc. If the same parameters are run more than once, these informatics should be tracked to get a bigger picture of the simulation.

C.5 - Population

Populations consist of their respective species, or type of creature. Population features that are applicable to the whole species should be specified. Populations should have a population size and mutation rate.

C.6 - Food

Food is an object that has a shape, size, and color given by the user, and is stationary on the GUI simulation. A food object also has a regeneration rate, which determines how often a new food object will appear in the simulation. If the food is in the list of prey of a creature, then when the creature collides with the food, it will eat the food and gain health.

C.7 - Creature States

A creature can be in one of three states depending on its health, represented by an integer between 0 and 100, at a certain time. If the health reaches 0, then the state of the creature is in the dead state. If the health is below 50, then the creature is in the sick state. If the creature has a health above 50, then it is in the healthy state.

C.8 - Real Time Graphs of Population Sizes

The value of the continuous simulation is observing how the population sizes interact and change over time. The population sizes at a given moment would not provide as much information as a graph of change over time, so the simulation will generate a line graph of each population size as the simulation progresses.

C.9 - Creature Shade Correlates to Health

As a creature loses health, its shade reduces. Therefore a creature with high health will look more vibrant and a creature with low health will look more dull.

C.10 - Genetic Changes

Each creature in a population has genes that affect its characteristics. When two creatures of the same type collide in the simulation, they may produce another creature whose genetic makeup is a mix of both parents.

C.11 - Mutations

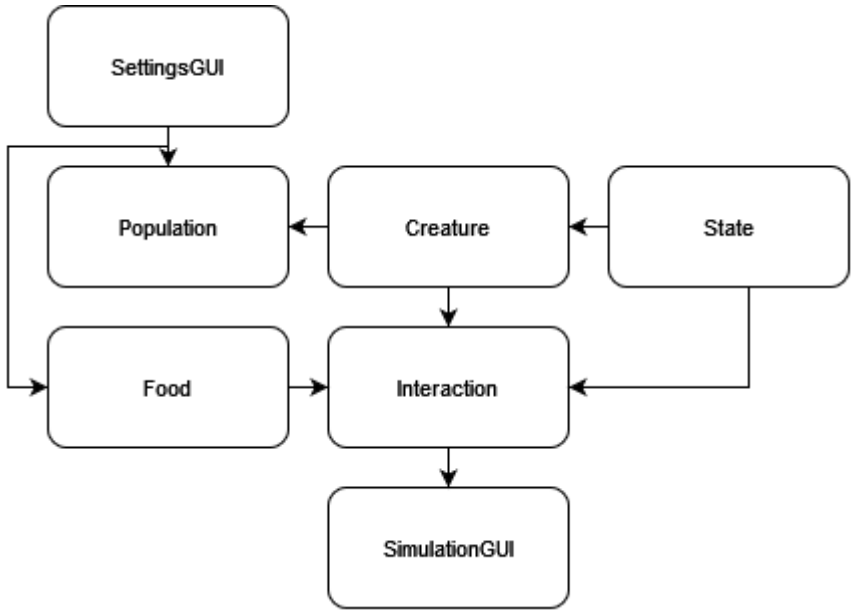
The offspring of two creatures may randomly have altered characteristics inherited from neither parent.

System Design

This section describes the system design in detail. An overview of the major system components and their roles will be provided, followed by more detailed discussion of components. Component discussion will reference the technical features the component helps satisfy. The design patterns we use are state, observer and builder.

Architecture Overview

High-level System Architecture



Major Components

GUI Components

L.1, L.2, L.3

The GUI components are responsible for showing the main menu, getting the parameters from the user, and portraying the simulation to the user in real time.

Population Components

C.5
The population components should be responsible for having features that collectively belong to all of one species.

Creature Components

C.1, C.7
Creature components should be used to customize certain attributes of a species including what they look like on the GUI map, the aggression rate towards food and other species, the movement patterns, etc.

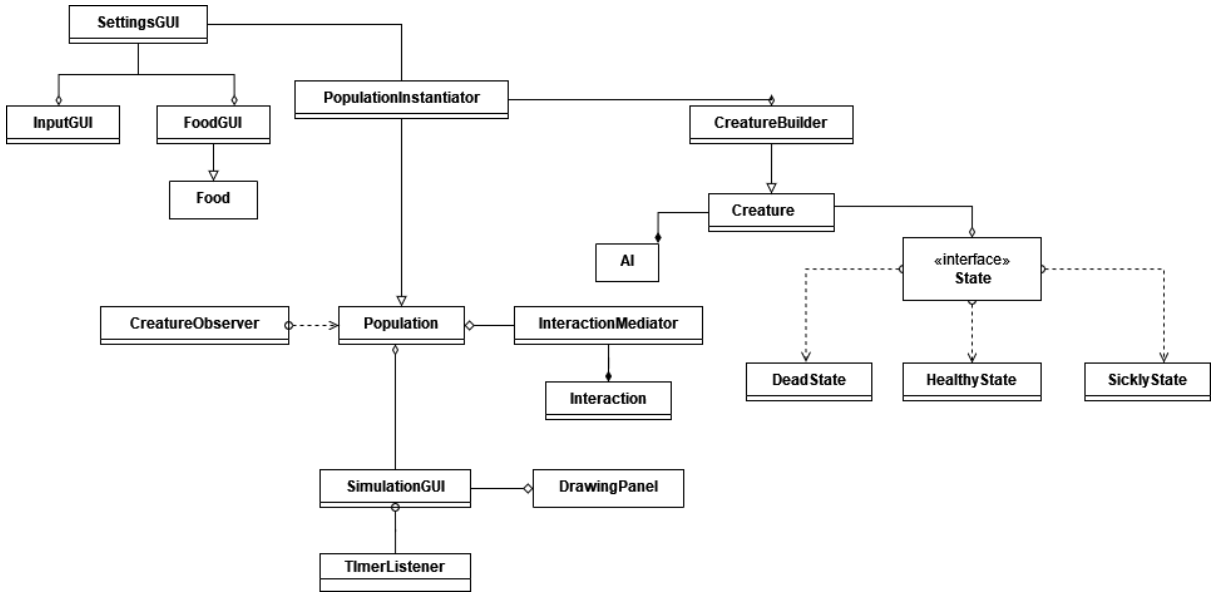
Food Components

C.6
Food components should be used by the creatures to increase their health. They should also be available to customize in the GUI.

SimulationEngine Components

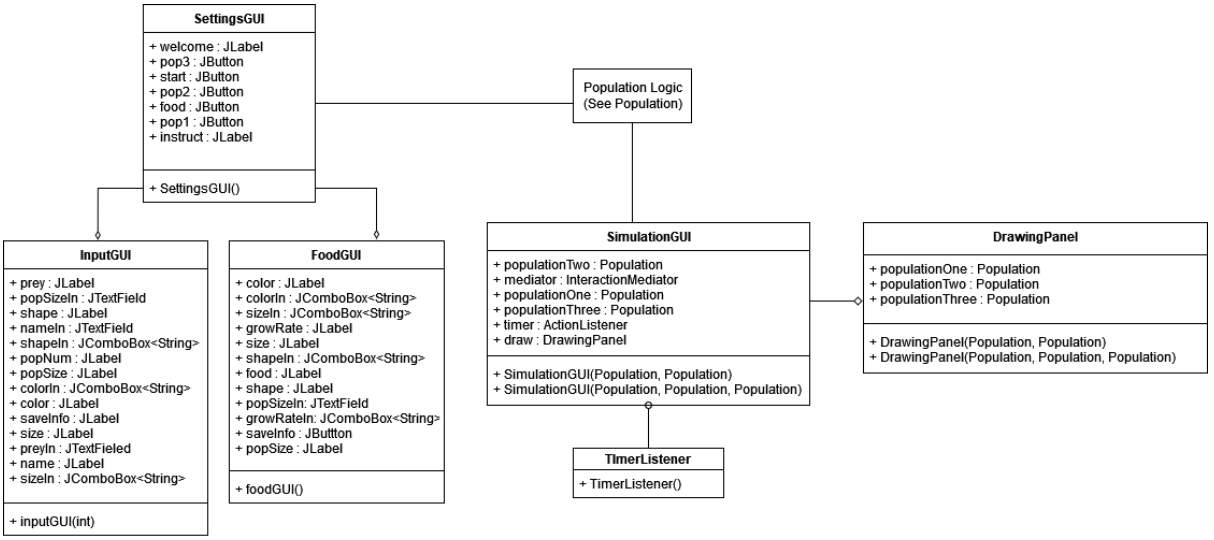
P.2, T.1
Simulation engine should be responsible for getting all of the individual parts of the simulation to come together and run finely.

Detail Design



GUI Components

The user interacts only with the GUI screens when they run the program. The user can input the characteristics they want the populations to have on the first pop up window, and then the simulation runs in another frame after the user presses the start button.



SettingsGUI

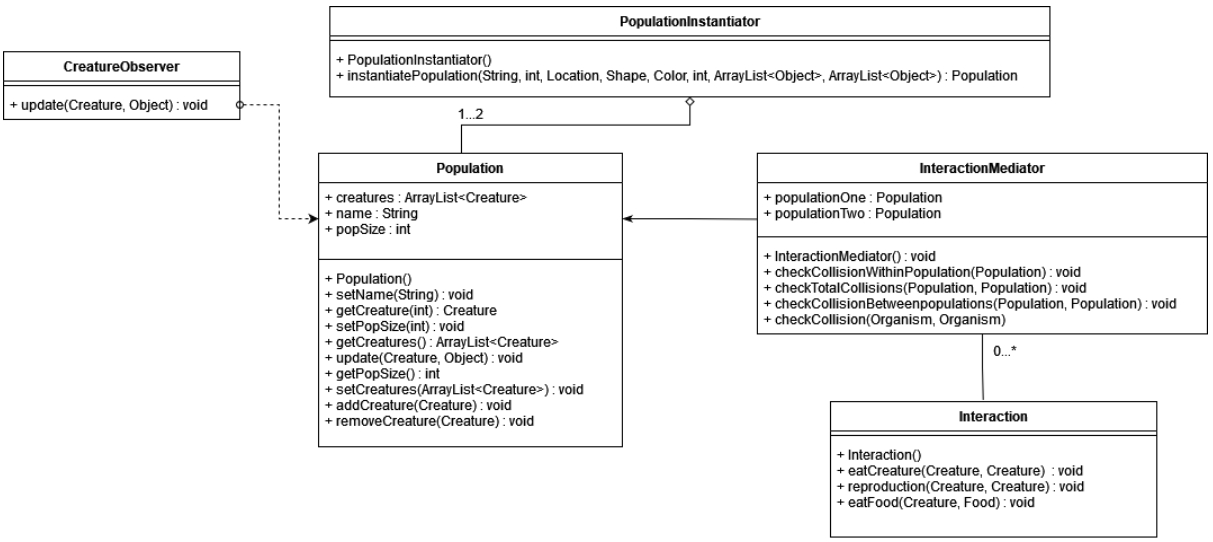
ux.1
The main function is contained in the SettingsGUI class. The SettingsGUI is where the user sets the parameters of the simulation and starts the simulation. When the program is run, the settings window appears and allows the user to click on different population buttons to edit the parameters in a new window that pops up. These parameters are encapsulated by the PopulationParameters and passed into the simulation when the user presses the save buttons. Once the populations are defined, the user can press the Run Simulation button, which calls the method to run the Simulation GUI.

SimulationGUI

ux.1
The simulation GUI is the visual component to the GUI Simulation. This is generated from the SettingsGUI when the user runs the simulation. The SimulationGUI is responsible for the visual board, live graphs, and a hard stop button.

Population Components

Populations are a list of a certain type of creature. The program allows for two creature populations to be created, along with one type of food.



PopulationInstantiator

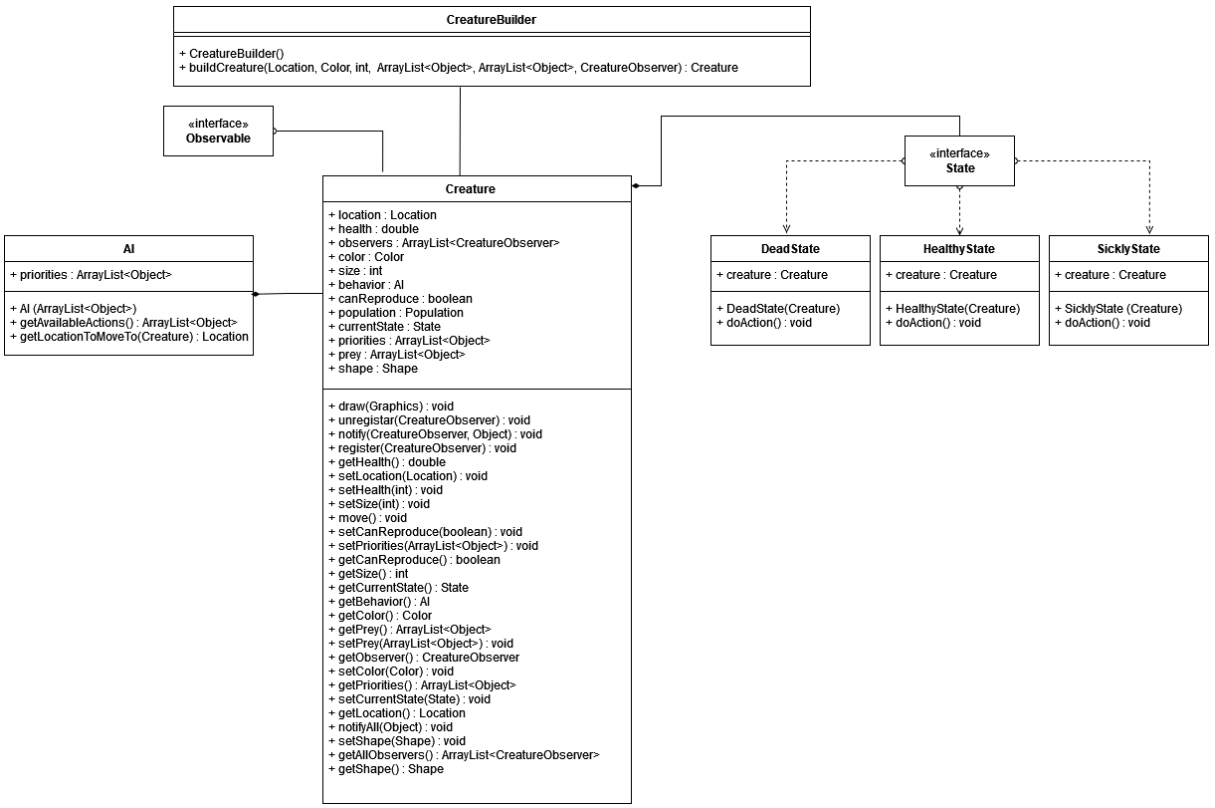
C.5
Population Instantiator should be used to get information from the user input and instantiate a population instance.
This is where the builder design pattern is implemented. The population instantiator is used to build populations instead of using the constructor in population.

Population

C.5
The Population is a class that is a list of many creatures of one kind.
The population class uses the observer design pattern. Each population implements the CreatureObserver interface and has the update(Creature, object) function. Update() removes or adds the creature to the population based on the object passed in.

Creature Components

Creatures are a class that describes a type of animal or other creature that participates in the simulation.



CreatureBuilder

C.1

CreatureBuilder is used to send information to the Creature and instantiate a Creature instance.

Creature

C.1

A creature has the fields size, shape, color, priorities, name, movement, and list of prey. The user can input these fields for a maximum of two different types of creatures per simulation. Creatures also implement the Observable interface, which updates all other creatures in a population when creatures are dead or born.

InteractionMediator

C.2

The interaction mediator class checks for collisions between objects in the GUI simulation. If two objects collide, then the interaction mediator determines which two objects the collision was between: food and creature, two of the same type of creature, or two different types of creatures. Based on the interaction, it calls the appropriate method from the interaction class.

Interaction

C.2

After getting information from the InteractionMediator, this class should decide depending on which objects are interacting to call a specific creature state on each object. If two objects of the same population interact, the reproduction interaction

AI

C.2

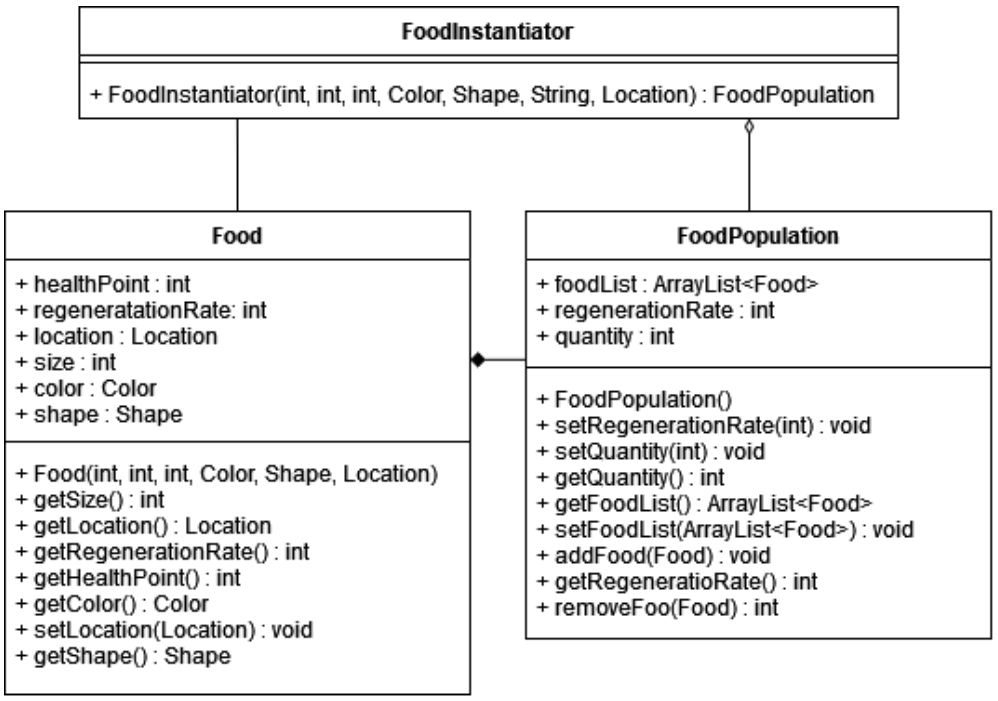
The AI should be responsible for the creature's movement patterns. Current implementation has random movement, but AI class could be extended with a strategy pattern to have multiple movement types.

States

C.2

States should be responsible for changing the current state of the creature. At any given time a creature can be in a healthy state, an injured/sick state, and a dead state. Interactions can affect the state. This is where the state design pattern is implemented. When the creature is in a sick state, it cannot reproduce. When it is in a dead state, then it is removed from the population list. The state class is responsible for carrying out these functions and changing states of the creatures.

Food Components



Every simulation has one type of food object, and the user can decide the characteristics of the food in the first window.

FoodInstantiator

C.6
Food instantiator should be the class that sends the information to the Food class to instantiate Food. The builder design pattern is also implemented here because the food constructor is not used to create new food instances.

Food

C.6
Food is a class that has fields of size, shape, color, and regeneration rate. The food objects do not move in the simulation, but can get eaten by a different creature and disappear or regenerate at a rate based on the regeneration rate given by the user. When the food regenerates, it appears in random new locations on the simulation window.

SimulationEngine Components

SimulationEngine

P.2 T.1
~~SimulartionEngine contains the main method of the code, and links together the GUI and the other classes.~~
The logic of the simulation engine lives in the SimulationGUI class.

Data Formats

The simulation does not take in any data files, and instead allows the user to enter in their decisions through a GUI. When the program runs, the user will be able to click on a population and another window will appear that allows the user to fill in all the details of the new creature population. Most of these decisions can be indicated by the user picking an option from a drop down menu, so there cannot be errors from an incorrect input. Some require the user to type in a number or string into a text field. The user can type the name of the population into a text field, which cannot produce an error because any string can be a valid name. The user types an integer into the text field for initial population size, which would cause an error if the user types in something that is not an integer.