Creating a System Dynamics API for Developers

A Major Qualifying Project

Submitted to the Faculty of

Worcester Polytechnic Institute in partial

fulfillment of the requirements for the Degree

in Bachelor of Science in

System Dynamics

By

_____

Jacob Freise

Date: 8/15/19

Sponsoring Organization:

Worcester Polytechnic Institute

Project Advisors:

_____

_____

Professor Khalid Saeed, Advisor

Professor Jeffrey Kesselman, Advisor

review. For more information about the projects program at WPI, see http://www.wpi.edu/Academics/Projects

Abstract

The goal of this project is to create an Application Programming Interface (API) that allows developers to make use of system dynamics models to help drive realistic systems, and then implement a simple example of such a system.

Acknowledgments

Without help from certain individuals the completion of this project would not have been possible. Some of these people helped us by providing guidance while others helped by supplying the resources necessary to proper testing.

I would like thank Jeffrey Kesselmanand Khalid Saeed, the advisors to this project, for overseeing the project, providing much needed guidance, and giving me a sense of direction.

I would like to thank James P. Houghton for offering a high-level overview of how the project could work from the unique perspective of someone who is proficient with both programming and System Dynamics.

Table of Contents

Chapter 1: Introduction

The definition of a complex system is something that is more than the sum of its parts. The implementation of complex systems is sought after by game developers looking to add a sense of depth and complexity to their games. This process often tries to mimic the complex system by replicating its real-life mechanics, with many small individual factors lead to an emergent system. However, this can have unintended consequences as implementing a system in this manner requires a detailed understanding of the system. Lack of such knowledge will lead to the emergent system not matching the expected system, and sometimes in wildly unexpected ways due to the nature of chaotic systems.

1.1 Motivation

Games often have social systems such as economies, ecosystems, populations, ect. These social systems can be difficult to implement realistically, as developer would need to consider the effect of each individual relationship between the agents in the system. When the developer starts, they are simply trying to get a working model off of the ground, starting values and coefficients are usually arbitrarily picked values that get readjusted based off of trial and error. As they are developing each functionality of the system their perspective becomes tunneled. This can lead to problems that the developers do not expect as they are looking at the individual pieces of the system as they do not have the means to view it holistically.

Whereas a system dynamics modeler has the luxury of not needing to implement the functionality of the interactions between agents, they only need to consider the aggregate behavior of the total number of agents over time. This allows them the freedom to capture

nothing but the behavior of the complex system. The modeler can not tell you what any given agent is doing, but can tell you what all of them are doing. At the end of the day the game developer and the modeler are trying to achieve the same system results, but the game developer just has a much harder job.

Just as games often use a physics engine to drive realistic physics, a model could be used to drive realistic complex systems. The model could be developed and tested independently from the game to ensure proper dynamics, and then use those dynamics to provide information to the game.

1.2 Report Organization

This report is broken into four distinct chapters. Chapter 1: Introduction, introduces the intent and purpose of the report, providing the motivation for the development of the project. Chapter 2: Literature review investigates previous work done on this topic, discussing their goal, methods and conclusions. Chapter 3: Project Design talks about how the project works from a high level. Chapter 4: Implementation discusses how the API was implemented and provides the more technical details. Chapter 5: Post-Implementation talks about a demo project built within Unity to show the API in use. Chapter 6: Conclusions and future work summarizes the overall finding of this project and the next steps that can be taken to further development.

Chapter 2: Background

2.1 Literature Review

This is not the first time a program interface has been developed that accepts System Dynamics models. A program called pysd was developed by MIT doctoral student James P Houghton. Pysd

is a library that runs models in python with the purpose of improving integration of Big Data and Machine Learning into the System Dynamic workflow.[1] Pysd translates a SD model into python and then creates an object that can run. However, this model runs to completion, instead of simulating a single time step at a time. Our project goal requires the simulation pause after each timestep.

Chapter 3: Project Goals and Objectives

3.1 - Main Goal

The goal of this project is to develop a simple way for developers to implement a system dynamics model into their programs that help drive dynamics. The developers will be able to access any variable the model uses, and use the value of that data in their main program. Since programs involve human interaction, which models do not normally account for, the program must also be able to change the values within the model during simulation.

This project was intended to be used within the game development environment called Unity which uses C# or javascripts to drive game mechanics. C# was the language chosen as it is the more supported language by Unity.

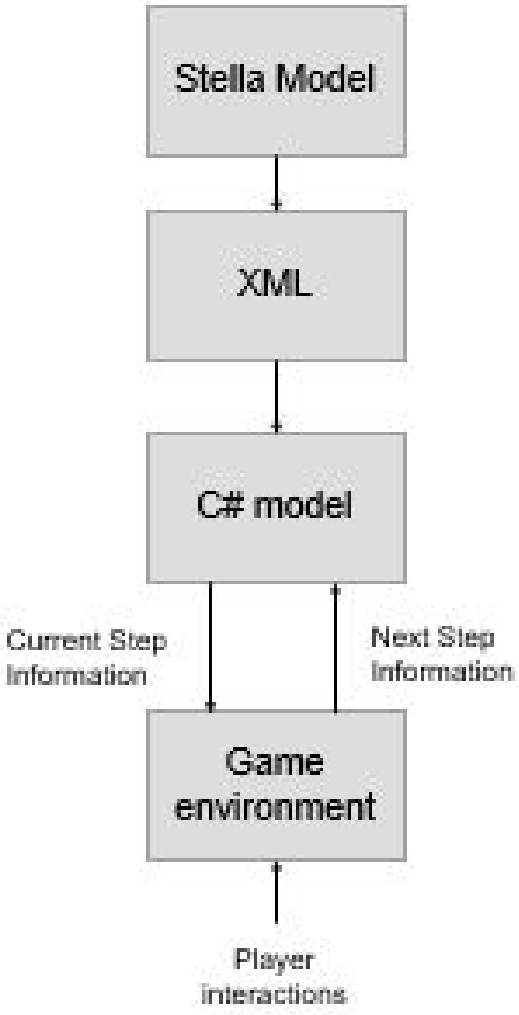
3.2 - Project Objectives

The primary object is to take an existing system dynamics model and implement into C#, allowing a developer to push and pull data to and from it. In order to do this the following things must be achieved

- Research and understand how system dynamics models are saved.
- A way to take the saved model and take its relevant data and relationships into C#

- A program that takes this saved data and rebuild the model in code.
- An interface that allows the developer to access the data as the model is being simulated

3.3 - Design Decisions



The Stella model is saved as an XML file. This XML file is parsed into C# and the model is rebuilt in code. The program has a list of all of the variables within the model and their current value. The program goes through each variable and its children, to ensure everything a variable

relies on is updated before the variables new value is calculated. The first variables that are updated are the stocks of the model as these need to be calculated first to use in other calculations, the stocks take their variable values from the previous timestep. All of the variable values for that current timestep are available to the developer to use or change.

Chapter 4: Implementation

4.1 - Parser

C# has a library called XDocument that allows easy parsing of XML files. The XML file is loaded into an XDocument object and then searched for an element defined by the System Dynamics program named "variables." This element hold all of the variables in the model, which are then further broken down into stocks, flows, and converters. Each of these are saved into a C object, named 'Variable'.

The Variable class consists of the name of the variable, the current value, its equation, and optionally a graphical function. The name and equation are stored as strings, the value as a float, and the graphical function as a new graphical object. The variable class simply serves as a way to store the data of the model, it makes very limited use of private functions.

The graphical class is used to store graphical functions defined by the model. In system dynamics graphical functions are drawn which substitute for a mathematical equations in order to save time and effort. Instead of a continuous curve defined by an equation, it consists of many points connected with straight lines. The graphical class has a public function called getOutput that can take an x value and returns a y value. This is done by finding which straight line the point would fall onto, and then using that lines linear equation to determine the y value.

Essentially many different piecewise linear interpolation functions create a non-linear interpolation function often used by modelers.

4.2 - Model Builder

The model class takes an xml file location and returns a model object. This model creates a list of variable objects from parsing the xml file. These variable objects are used to fill a hashtable with the name of a variable as the key, and the respective variable object as the value of the hashtable. Each value in the hashtables is put through a recursive function to check if it is updated, if the value is not updated, it checks its children. If a variable's function includes another variable, that other variable is considered a child. If the children are updated then the variable value is calculated based off of the child values and an updated flag is set within the variable object.

4.3 - Interface

Since the model is stored in a model object, there are a couple public functions that allow developers to interface with the model. The first is ".simulate()", this simply performs one timestep of calculation.

The second function is ".getVariable()", this takes a string of a variable name and returns the current value from the hashtable. This allows developers to access any given value of the model at the current timestep, to be used in their application.

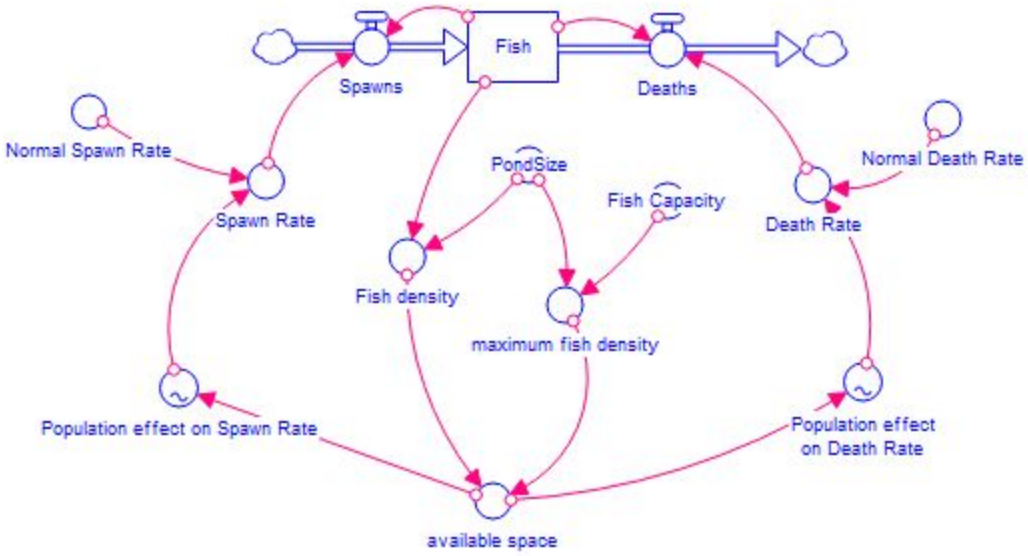
The third and final function is ".setVariable()", this takes a string of a variable name and a float value and sets the current value of the variable to the given float. This allows developers to update the model with data from outside the model, such as from user input.

Chapter 5: Post-Implementation

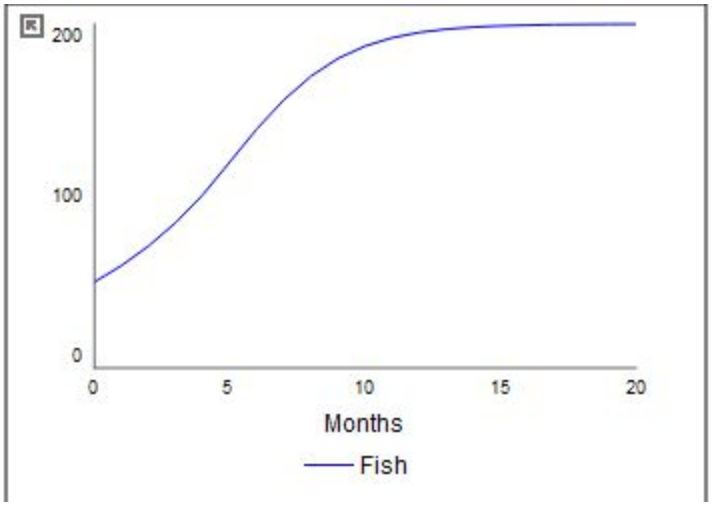
A hammer is never as impressive as the house it builds, this applies to an API as well. The API alone does not do very much, so to provide a better understanding of the utility this project provides I created a unity project that is driven by a simple model.

5.1 - Fish Model

When considering the type of model to represent the API I wanted something limited to one stock to ensure simplicity, I also wanted the model to use features common in many system dynamics models. I wanted graphical functions, variables with multiple levels of dependencies, and an interesting subject. I decided to choose fish because they are a common specimen used by modelers and they have compact environments that can fit on a computer screen.



The fish capacity determines the number of fish the pond can support. If the pond is below the fish capacity, fish will grow to meet it. If the fish are above the fish capacity the fish will die off until the environment can support their numbers. This dynamic is achieved by finding the difference between the maximum number of fish and the current number of fish to find the amount of available space the pond has left. The more available space the more breeding and the less amount of death occurs. The less available space the less breeding and more death occurs. This causes the fish to always be pressured towards having zero available space.



The fish population follows an S-curve because it starts off following the reinforcing feedback loop of population growth, but as it gets closer to the population limit, the balancing feedback loop of population capacity takes over as the dominant relationship

5.2 The Model - Simulation Interface

The first step to get the model working in Unity is to make sure you have a .dll of the API. After inserting the .dll into your Unity assets, the model can be accessed by providing its file location.

```
//make a model
private Model FishModel = new Model("Fishmodel.stmx");
```

Now all the information about the model can be accessed by referencing FishModel. The API allows for three calls, Simulate(), GetVariable(), and SetVariable(). I start off by using GetVariable() to pull information from the model that is independent of time, such as the size of the pond.

```
// Awake is called at the start of the program
Event function
void Awake()
{
    m_SpawnArea = new Vector3( x: FishModel.getVariable("PondSize"), y: 50f, z: FishModel.getVariable("PondSize"));
    GetWaypoints();
    CreateAIGroups();
    InvokeRepeating( methodName: "SpawnNPC", time: 0.5f, repeatRate: spawnTimer);
}
```

FishModel.getVariable("PondSize") is being used to get the area of the pond surface. Also note that InvokeRepeating() is used to call the function SpawnNPC at an interval rate determined by spawnTimer. Since a model normally completes instantly we need to find some way to spread the calls out over time. If we updated the model every frame we would step through it too quickly. The repeating function allows the user to control the speed of iterations in real time using a slider that sets the value of spawnTimer, which determines the speed of the model.

```
1 usage
void SpawnNPC()
{
    //get number of fish from the model
    int numberOfFish = (int) FishModel.getVariable("Fish");

    //for every species
    for (int i = 0; i < AIObject.Count(); i++)
    {
        //if species is set to spawn and has a model
        if (AIObject[i].enableSpawner && AIObject[i].objectPrefab != null)
        {
            //update species population information
            AIObject[i].setValues(numberOfFish);
        }
    }
    //Simulate model for one time step
    FishModel.Simulate();
}
```

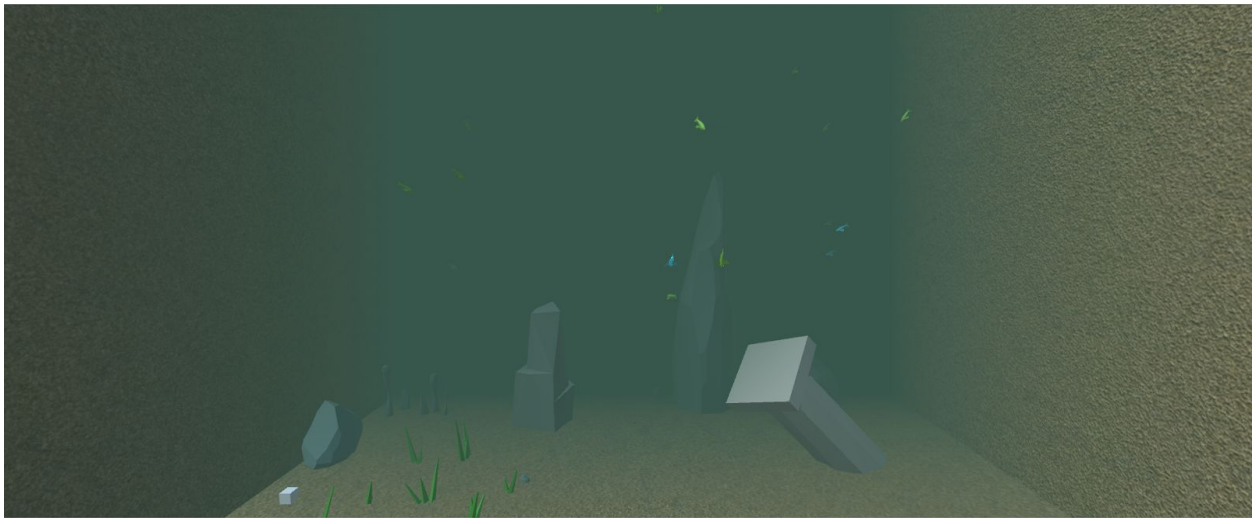
SpawnNPC is the function used to both get information from the model and tell it to go to the next time step. The stock of fish is accessed using FishModel.getVariable("Fish"), and then temporarily stored in numberOfFish where it is used to update the relevant part of the game with the new information. After the game is updated the model is then told to tick to the next timestep using FishModel.Simulate().

5.3 The Simulation

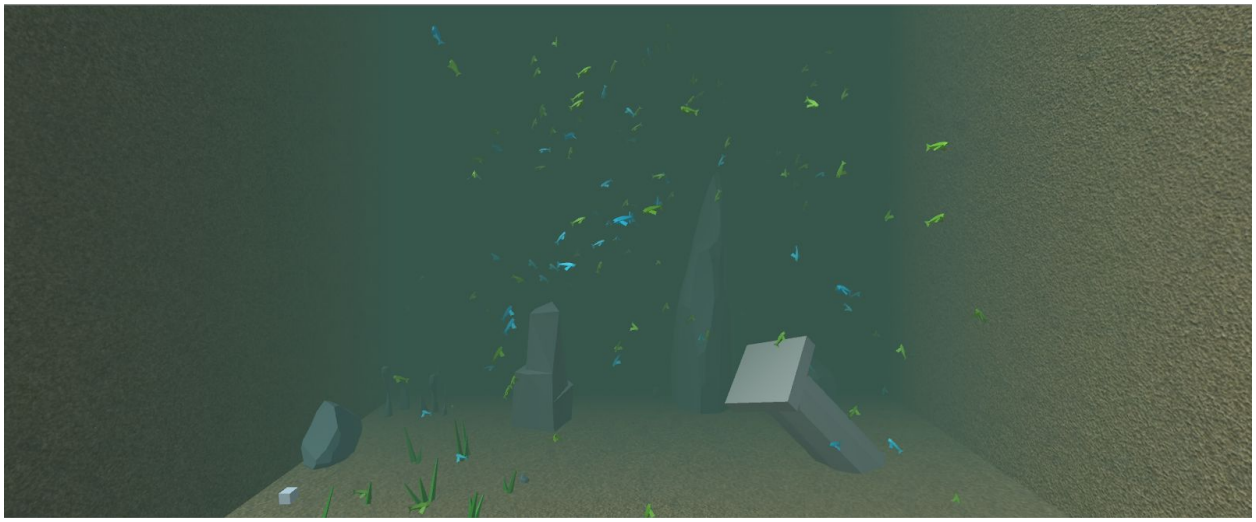
After the interface, the model has no influence the rest of the simulation is determined by the Unity project. In order to get semi-realistic behavior from the fish entities I implemented some

famous flocking behaviors first theorized by Craig Reynold's 'Boids'. Boids are creatures that have a limited perception range and follow three simple behaviors. Firstly, they want to group up with other boids near them, this is called cohesion. Secondly, they do not want to get too close, they keep separated at a distance smaller than their cohesion distance. Thirdly, they want to go in the direction of their neighbors. These three behaviors alone can give some very interesting and complex dynamics, especially when each behavior is given a weight value to determine their amount of influence.

My simulation uses some modified behaviors that allow me to create composite behaviors and filter out which boids are in the detection radius. I specifically wanted to them to avoid crashing into objects, to favor particular areas, and to be able to react differently to different types of boids. This allows me to create two different species, I created a bluegill in blue and a bass in green. The bluegills have a higher propensity to flock together by increasing their cohesion weight, and they run away from bass by increasing their avoidance weight against bass. The bass chase the bluegill by having an increased cohesion weight towards bluegill.



At the start of the simulation the fish population is very low.



However, further into the simulation the population has stabilized at capacity.

Chapter 6: Conclusions and Future Work

There are some limitations to this model simulator, system dynamics modeling software tends to include built-in functions that perform mathematical operations beyond simple multiplication and addition. They can perform things like averages and delays. In order for my model builder to handle these functions, they would each have to be included inside the model builder class. There are also some other more complex features of SD modeling the model builder cannot represent, such as conveyors.

This project only offers a bridge between a SD model and a C# program, the example demonstration does not come close to scratching the surface of potential as it does not use the .setVariable() function. This is the most powerful of the tools, but also the most dangerous. This is the function that allows you to override data in the model to replace it with information from the simulation. This allows the model to capture something otherwise impossible, real time feedback from a person's individual actions. The model could take into account the fact that a

player killed so many fish that their population could not recover. I see this important in a model that is more detailed and considers more factors of the system. Perhaps fish capacity is determined by the number of plants in the pond. So if a player kills some plants, the action inadvertently kills fish until the plants have time to recover.

Bibliography

1. Houghton, James; Siegel, Michael. "Advanced data analytics for system dynamics models using PySD." *Proceedings of the 33rd International Conference of the System Dynamics Society.* 2015.
2. Reynolds, Craig W. “Flocks, Herds and Schools: A Distributed Behavioral Model.” *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques - SIGGRAPH '87*, 1987, doi:10.1145/37401.37406.
3. ***Code can be found at:*** https://github.com/JakeFreise/StateMachine

Appendix

Model

Fish Model Equations

Fish(t) = Fish(t - dt) + (Spawns - Deaths) * dt {NON-NEGATIVE}

INIT Fish = 200

INFLOWS:

Spawns = Spawn_Rate*Fish {UNIFLOW}

OUTFLOWS:

Deaths = Fish*Death_Rate {UNIFLOW}

available_space = maximum_fish_density-Fish_density

Death_Rate = Normal_Death_Rate*Population_effect_on_Death_Rate

Fish_Capacity = 200

Fish_density = Fish/PondSize

maximum_fish_density = Fish_Capacity/PondSize

Normal_Death_Rate = .1

Normal_Spawn_Rate = .1

PondSize = 100

Population_effect_on_Death_Rate = GRAPH(available_space)

(-1.000, 2.000), (-0.833333333333, 1.83333333333), (-0.666666666667, 1.66666666667),

(-0.500, 1.500), (-0.333333333333, 1.33333333333), (-0.166666666667, 1.16666666667),

(0.000, 1.000), (0.166666666667, 0.833333333333), (0.333333333333, 0.666666666667),

(0.500, 0.500), (0.666666666667, 0.333333333333), (0.833333333333, 0.166666666667),

(1.000, 0.089)

Population_effect_on_Spawn_Rate = GRAPH(available_space)

(-1.000, 0.057), (-0.833333333333, 0.166666666667), (-0.666666666667, 0.333333333333),

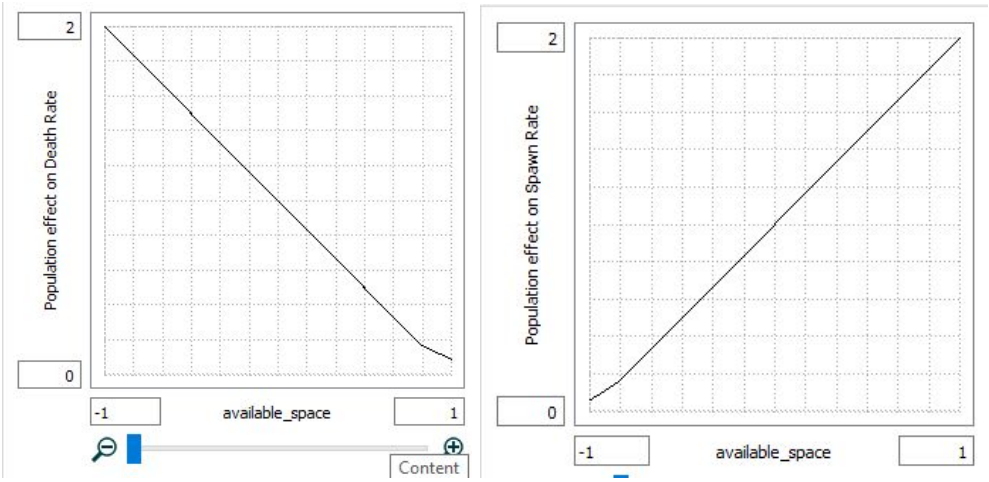
(-0.500, 0.500), (-0.333333333333, 0.666666666667), (-0.166666666667, 0.833333333333),

(0.000, 1.000), (0.166666666667, 1.16666666667), (0.333333333333, 1.33333333333), (0.500,

1.500), (0.666666666667, 1.66666666667), (0.833333333333, 1.83333333333), (1.000, 2.000)

Spawn_Rate = Normal_Spawn_Rate*Population_effect_on_Spawn_Rate

Graphical Functions



Program

Main Class

```
        public static void Main(string[] args)
        {
            string xml = "testModel.stmx";

            Model testModel = new Model(xml);

            for (int counter = 0; counter < 10; counter++)
            {
                testModel.Simulate();

Console.WriteLine(testModel.getVariable("TestStock"));
            }
        }
```

Model Class

```
 public class Model
    {
```

```
        private List<Variable> variableList;

        private Hashtable H1 = new Hashtable();
        private Hashtable H2 = new Hashtable();

        public Model(string xmlFile)
        {
            XDocument xDoc = XDocument.Load(xmlFile);

            XNamespace ns =
xDoc.Root.Attributes("xmlns").First().Value;

            variableList = ParseXML(xDoc, ns);

            //fill hashtables with inital values
            foreach (var variable in variableList)
            {
                H1.Add(variable.Name, variable.Value);
                H2.Add(variable.Name, variable.Value);
            }
        }

        //returns current value
        public float getVariable(string variable)
        {
            return (float) H1[variable];
        }

        //sets next value
        public int setVariable(string variable, float value)
        {
            if (H1[variable] != null)
            {
                H1[variable] = value;
                return 1;
            }

            return 0;
```

```
        }

        public void Simulate()
        {
            //update H2 using H1
            foreach (var variable in variableList)
            {
                string explicitEquation =
MakeExplicitEquation(variable.Equation, H1);
                Expression e = new Expression(explicitEquation);
                float value = float.Parse(e.Evaluate().ToString());

                if (variable.Function == null)
                {
                    H2[variable.Name] = value;
                }
                else
                {
                    H2[variable.Name] = graphicalOutput(value,
variable.Function);
                }
            }

            //update H1 using H2
            foreach (var variable in variableList)
            {
                string explicitEquation =
MakeExplicitEquation(variable.Equation, H2);
                Expression e = new Expression(explicitEquation);
                float value = float.Parse(e.Evaluate().ToString());

                if (variable.Function == null)
                {
                    H1[variable.Name] = value;
                }
                else
                {
                    H1[variable.Name] = graphicalOutput(value,
```

```
variable.Function);
                }
            }
            //Console.WriteLine("Test Stock: "+H1["TestStock"]);
        }

        private string MakeExplicitEquation(string equation,
Hashtable table)
        {
            var operators = new List<string>{"+", "-", "*", "/"};
            equation = addSpacesToString(equation);
            //break equation into pieces
            string[] words = equation.Split(' ');

            //for each piece
            for (int counter = 0; counter < words.Length; counter++)
            {
                //that isnt a math operator
                if (!operators.Contains(words[counter]))
                {
                    //convert it to the value in table
                    var value =  table[words[counter]];
                    if (value != null)
                    {
                        words[counter] = value.ToString();
                    }
                }
            }

            //remake the string and return it
            string concat = (String.Join(" ", words));
            return concat;
        }

        private string addSpacesToString(string input)
        {
            string output = input;
```

```
            string[] words = input.Split('*');

            if (words.Length > 1)
            {
                output = words[0];

                for (int counter = 1; counter < words.Length;
counter++)
                {
                    output = output + " * " + words[counter];
                }
            }

            words = input.Split('/');

            if (words.Length > 1)
            {
                output = words[0];
                for (int counter = 1; counter < words.Length;
counter++)
                {
                    output = output + " / " + words[counter];
                }
            }
            return output;
        }

        private List<Variable> ParseXML(XDocument xDoc, XNamespace
ns)
        {
            var result = xDoc.Root.Descendants(ns + "variables");

            List<Variable> variableList = new List<Variable>();

            foreach (XElement variable in result)
            {
                //STOCKS
                var stockList = from q in variable.Descendants(ns +
```

```
"stock")
                            select new
                            {
                                name = q.FirstAttribute.Value,
                                initial = q.Element(ns +
"eqn").Value,
                                inflow = q.Element(ns +
"inflow").Value,
                                outflow = q.Element(ns +
"outflow").Value
                            };

                foreach (var stock in stockList)
                {
                    string equation = makeStockEquation(stock.name,
stock.inflow, stock.outflow);
                    variableList.Add(new Variable(stock.name,
float.Parse(stock.initial), equation));
                }

                //FLOWS
                var flowList = from q in variable.Descendants(ns +
"flow")
                               select new
                               {
                                   name = q.FirstAttribute.Value,
                                   equation = q.Element(ns +
"eqn").Value,
                               };
                foreach (var flow in flowList)
                {
                    variableList.Add(new Variable(flow.name, 0,
flow.equation));
                }

                //CONVERTERS
                var auxList = from q in variable.Descendants(ns +
"aux")
```

```
                                select new
                                {
                                    name = q.FirstAttribute.Value,
                                    equation = q.Element(ns +
"eqn").Value,
                                    graph = CreateTable(q, ns),
                                };
                foreach (var aux in auxList)
                {
                    Variable converter = new Variable(aux.name, 0,
aux.equation);
                    converter.SetFunction(aux.graph);
                    variableList.Add(converter);
                }
            }

            return variableList;
        }

        private static Graphical CreateTable(XElement element,
XNamespace ns)
        {
            var table = from q in element.Descendants(ns + "gf")
                select new
                {
                    list = q.Element(ns + "ypts").Value,
                    bounds = GetBounds(q)
                };

            foreach (var value in table)
            {
                Graphical newGraph = new Graphical(value.list,
value.bounds);
                return newGraph;
            }
            return null;
        }
```

```
        private static string[] GetBounds(XElement element)
        {
            string[] bounds = new string[4];
            var elements = element.Elements();

            var xLower = elements.First().Attribute("min").Value;
            var xUpper = elements.First().Attribute("max").Value;
            var yLower =
elements.ElementAt(1).Attribute("min").Value;
            var yUpper =
elements.ElementAt(1).Attribute("max").Value;

            bounds[0] = xLower;
            bounds[1] = xUpper;
            bounds[2] = yLower;
            bounds[3] = yUpper;

            return bounds;
        }

        private string makeStockEquation(string name, string inflow,
string outflow)
        {
            string plus = " + ";
            string minus = " - ";
            string equation = name;

            if (inflow != null)
            {
                equation = equation + plus + inflow;
            }
            if (outflow != null)
            {
                equation = equation + minus + outflow;
            }

            return equation;
        }
```

```
        private float graphicalOutput(float input, Graphical
function)
        {
            float output = input;
            if (input > function.Xmax)
            {
                output = function.Xmax;
            }
            else if (input < function.Xmin)
            {
                output = function.Xmin;
            }

            output = function.getOutput(output);
            return output;
        }
    }
```

Variable Class

```
public class Variable
    {
```

```
        private string name;
        private float _ourValue;
        private string equation;
        private Graphical function = null;

        public Variable(string newName, float newValue, string
newEquation)
        {
            name = newName;
            _ourValue = newValue;
            equation = newEquation;
            //Console.WriteLine("NAME: {0}, EQN: {1}, INFLOW: {2},
OUTFLOW: {3}", name, _ourValue, _inflow, _outflow);
        }

        public string Name
        {
            get
            {
                return name;
            }
        }

        public float Value
        {
            get
            {
                return _ourValue;
            }
            set
            {
                _ourValue = value;
            }
        }

        public string Equation
        {
            get
```

```
            {
                return equation;
            }
        }

        public Graphical Function
        {
            get
            {
                return function;
            }
            set
            {
                function = value;
            }
        }

        public void SetFunction(Graphical value)
        {
            function = value;
        }
    }
```

Graphical Class

```
public class Graphical
    {
```

```
        private float[] xBounds = new float [2];
        private float[] yBounds = new float [2];
        private float[] yTable;
        private float[] xTable;
        private float[,] points;

        public Graphical(string stringTable, string[] bounds)
        {
            xBounds[0] = float.Parse(bounds[0]);
            xBounds[1] = float.Parse(bounds[1]);
            yBounds[0] = float.Parse(bounds[2]);
            yBounds[1] = float.Parse(bounds[3]);

            yTable = parseStringTable(stringTable);
            xTable = createXTable(xBounds, yTable.Length);
            points = new float[yTable.Length,2];

            for (int counter = 0; counter < yTable.Length; counter++)
            {
                points[counter,0] = xTable[counter];
                points[counter,1] = yTable[counter];

                //Console.WriteLine(points[counter, 0] + ", " +
points[counter, 1]);
            }
        }

        private float[] parseStringTable(string inputTable)
        {
            string[] words = inputTable.Split(',');

            float[] outputTable = new float[words.Length];

            for (int counter = 0; counter < words.Length; counter++)
            {
                //Console.WriteLine(words[counter]);
                outputTable[counter] = float.Parse(words[counter]);
            }
```

```
            return outputTable;
        }

        private float[] createXTable(float[] bounds, int divisions)
        {
            float difference = bounds[1] - bounds[0];

            float increment = difference / (divisions-1);

            float[] xTable = new float[divisions];

            for (int counter = 0; counter < divisions; counter++)
            {
                xTable[counter] = bounds[0] + increment * (counter);
                //Console.WriteLine(xTable[counter]);
            }

            return xTable;
        }

        public float getOutput(float x)
        {
            //Console.WriteLine(x);
            int target = 0;

            //find which line segment our x fits into
            for (int counter = 0; counter < points.Length; counter++)
            {
                if (x < points[counter, 0])
                {
                    target = counter;
                    break;
                }
                else if (x == points[counter, 0])
                {
                    return points[counter, 1];
                }
            }
```

```
            float a = (points[target, 1] - points[target - 1, 1]);
//y2-y1
            float b = (points[target, 0] - points[target - 1, 0]);
//x2-x1
            float m = a/b; //slope
            float intercept = points[target, 1] - m * points[target,
0]; //b

            float output = m*x + intercept;

            return output;
        }

        public float Xmin
        {
            get
            {
                return xBounds[0];
            }
        }

        public float Xmax
        {
            get
            {
                return xBounds[1];
            }
        }

        public float Ymin
        {
            get
            {
                return yBounds[0];
            }
        }
```

```
        public float Ymax
        {
            get
            {
                return yBounds[1];
            }
        }
    }
```