Terrarium Creature Development

Developing Creatures For a Survival-of-The-Fittest Competition In the Microsoft .NET Terrarium Environment

Brian R. Richardson

Abstract

The Terrarium game from Microsoft was created as a tool to demonstrate the capabilities of .NET, while at the same time allowing developers to get familiar with programming in .NET. This game is designed as an ecosystem simulation where each developer places a program they have developed with .NET, called a "Creature," into the environment to pit their creature against other developers' creatures. So the question is, what strategies will create creatures that will be successful in the Terrarium game? Strategies for creature development will be discussed and compared in this paper. A set of performance results will also be produced based on the implementation of the strategies used for the final versions of each creature.

1. Introduction

1.1 What is the Terrarium Game?

The Terrarium game is a Survival-of-the-Fittest Competition that allows programs developed using the Microsoft .NET Framework to compete against each other. Terrarium is a simulation of an ecosystem where there are three types of organisms: plants, herbivores, and carnivores. The goal of this multi-player game is to design a creature that can live longer and build a larger population than any other developer's creature placed in the environment. This allows developers to compete against each other using the strategies they create for their creatures.

With most games a player's control of the character starts when the game begins. However, in Terrarium each player's control of the character ends when they upload their creature to the environment [2]. Each player develops and records their playing strategy in a program called a "Creature". When they launch their program into the environment they no longer have any control over it, it is now in the Terrarium game and must rely on the instructions the developer has given it for its survival. At this point all a developer can do is monitor the progress of their creature against other developers' creatures to see how successful their strategies are in the game.

1.2 Why Do People Participate?

The Terrarium game allows developers to compete based on their strategies and programming abilities to see who is the top developer. One way Microsoft encourages participation is by posting the names of the top three developers of herbivores and carnivores on the Terrarium web page. This ranking allows for competing participants to see who can create the creature with the longest life span and largest population.

Microsoft makes available basic code to define plants, herbivores, and carnivores to allow programmers, who are unfamiliar with programming in the .NET Framework, to have a Visual Basic or C# program to start with as a basis for creating their creatures. Also, the Terrarium web site allows developers to share the source code of their creatures on an online bulletin board. This gives developers the chance to learn .NET in a fun competitive environment.

1.3 How Does Terrarium Prevent Cheating and Malicious Code?

One of the concerns of developers participating in the game is about others that may try and cheat [2]. When the Terrarium game was developed this was an obvious concern since a game being played by developers writing code to define their creatures' interaction in the game could find ways to give their creatures unfair advantages by breaking the rules defined for creature behaviour. In the Terrarium game, any creature whose properties or methods go outside the acceptable boundaries for a creature will automatically be removed from the game [2].

Malicious code is a concern since each creature multiplies and spreads to other machines on the network, and behaviour that appears similar to that of a virus could be used as a tool to spread viruses [2]. Microsoft insists that this is not a possibility since there is no part of the code for a creature that could allow it to exit the Terrarium game and cause damage. When a creature's code is scanned by the Terrarium game, if there is any code that does not fit within the definition of what is allowed for a creature, it is prevented from entering the game [2].

1.4 Organization of Work

This paper is organized as follows. Section 2 gives an introduction to the design of the Terrarium game. Section 3 outlines the basic features of a plant, herbivore, and carnivore. Section 4 discusses some examples of strategies for creature design. Section 5 shows the performance results for both the original and finished versions of each creature. This is followed by a discussion on the performance results and a look at which changes to each creature resulted in a performance improvement. Section 6 provides some insights into the Terrarium game as a tool for teaching .NET. Finally, section 7 discusses conclusions.

2. Design of Terrarium

2.1 Modes for Running Terrarium

The Terrarium game has two modes of operation: Terrarium mode and Ecosystem mode. The Terrarium mode, used for debugging of creature programs, allows you to run a small local environment that you can control by adding any number of a creature you have developed and monitoring their behaviour [4]. The Ecosystem mode is the game that connects you to other peers and allows you to submit your creature into the game to compete against other developers' creatures [4]. The Terrarium mode is needed since once a creature is entered in Ecosystem mode it cannot be reentered until all instances of that creature have disappeared from the network. In Ecosystem mode, if an error occurs in the creature, the developer receives no indication from the Ecosystem stating why their creature failed [4].

2.2 Peer-to-Peer - The Ecosystem "Teleporter"

When a Terrarium peer is run in Ecosystem mode it connects to the Terrarium central server that returns a list of 20-30 well-connected peers. Each Terrarium peer application has a blue sphere called a teleporter in the Terrarium display that randomly moves around in the environment. If a creature you have entered comes into contact with the teleporter it is automatically transported to another peer in the network that was randomly selected from the stored list of peers [4].

2.3 Terrarium Architecture

The basic architecture of the Terrarium system is similar to Napster in that it is a peer-to-peer system that relies on a central server for peer discovery and reporting [4]. When a peer logs onto the Terrarium system it first connects to the central Terrarium server run by Microsoft. The new peer tells the server its location and makes itself available as part of the networked ecosystem. When a peer wants to send a creature to other peers it receives a list of randomly selected peers from the central server and then sends its creature to one of those peers.

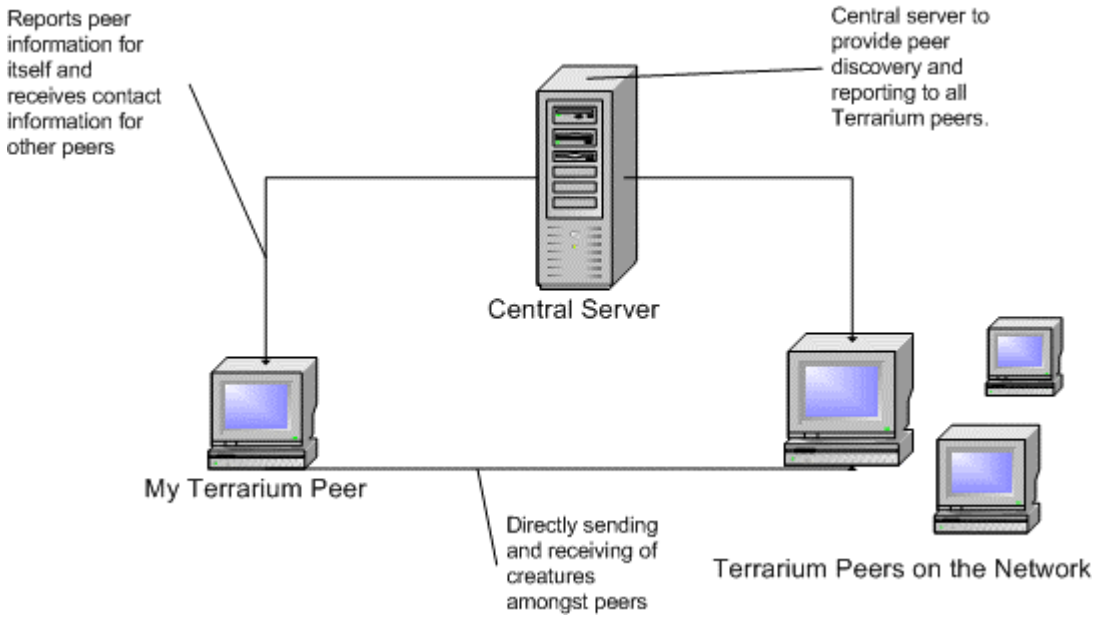

Figure 1: Terrarium System Architecture

2.4 XML Web Services

While each user's copy of the Terrarium game communicates with other nodes by the use of a peer-to-peer network, communication of peers to the central server is conducted through the use of web services provided by the central server that each peer calls [4].

When a new Terrarium game is started it calls a web service provided by the central server to send the server the IP address of the new peer [4]. As a response from this service the peer will either receive notification that it has successfully connected to the central server, or that its connection has been rejected for some reason (i.e., firewall, proxy, etc.) [4].

The next step for a Terrarium peer is to register as available to connect with other peers. The peer is able to do this by calling a registration web service provided by the central server. The response from this web service is a list of 20-30 well-connected peers. This is done to ensure your creatures will not end up being sent to a dead end or an area of the network that is only aware of a small number of peers [4].

When a user uploads a new creature the local peer scans the code to check to make sure the creature does not violate any rules of the game. The peer then calls a web service provided by the central server to send a copy of the creature to the central server for storage, only then will ten instances of the creature be created on the local peer [4].

For reporting of creatures currently alive on the local peer to all other peers, a reporting web service provided by the central server is called [4]. Each peer provides reporting information on the creatures it has to the central server so that developers can be kept up to date on how their creatures are performing.

3. Creatures

3.1 Creature Design

An ecosystem consists of living organisms. The .NET Terrarium is made up of three categories of organisms: plants, herbivores, and carnivores [1]. Plants in the terrarium environment are the most basic in terms of implementation of each of the three organisms. Herbivores and carnivores are much more complex since they both must move to explore their environment and locate food sources to be able to survive and reproduce.

Each herbivore and carnivore in the Terrarium environment is defined by a set of seven attributes called properties that state its strengths and weaknesses, as well as ten event handlers that define how the creature explores its surroundings, finds food sources, and reacts to the presence of other creatures. It is these attributes and event handlers that are used to define a developer's strategy for the design of their creature. In this section each creature's basic properties and event handlers will be discussed. (Strategies will be discussed later on in this paper).

3.2 Plants

The definition of a plant is relatively easy since plants do not have to have any sort of movement defined to allow them to locate sources of food. Each plant has three basic functions:

1. Grow
2. Reproduce by spreading seeds
3. Be eaten by herbivores

A plant organism does not need to contain any methods, it just has three properties that define how a plant functions.

1. The “MaximumEnergyPoints” property allows the developer to state how many energy points a herbivore will gain by consuming this plant. The maximum allowed is 10.
2. The “MatureSize” property which must be a number between 24 and 48 states the maximum size a plant can grow to when it has survived for some amount of time without being consumed. The smaller a plant is, the faster it will be able to reproduce. As a plant gets larger it will become easier for herbivores to see and will more likely be consumed.
3. The “SeedSpreadDistanceAttribute” states the maximum distance a plant can spread seeds from its current location. The maximum allowed is 100. These three properties alone are enough to define a functioning plant for the Terrarium game.

3.3 Herbivores

A herbivore in Terrarium is a creature that consumes plants and is consumed by carnivores. The three basic actions a herbivore must perform in order to survive are:

1. Locate and consume plants
2. Reproduce
3. Avoid carnivores

A herbivore has a “MatureSize” property that has the same settings as this property in a plant. The smaller the size, the faster they are able to move and reproduce. The larger the size the more ability they have to defend themselves, but they reproduce at a slower rate.

There are seven properties that a developer must distribute a total of 100 points amongst to define the basic strengths and weaknesses of their herbivore. Strategy is required since every point you add to one property results in a point being removed from one of the other six properties. The following is a description of the basic seven properties that both herbivores and carnivores possess.

1. “MaximumEnergyPoints” states how much energy the creature can store to allow it to survive longer in between food sources.
2. “EatingSpeedPoints” states how fast the creature can consume a food. This allows it to more quickly move to the next food source.
3. “AttackDamagePoints” states how much damage will occur when this creature attacks another. With a herbivore any points given here are useless.
4. “DefendDamagePoints” states how much damage will be caused to a creature that attacks. This is only useful if the creature is a herbivore.
5. “MaximumSpeedPoints” states how fast the creature is. This is important for a herbivore to avoid being caught, and to a carnivore to catch its prey.
6. “CamouflagePoints” states how well the creature blends into its environment making it difficult for others to see.
7. “EyesightPoints” states how far away a food source or predator needs to be in order for the creature to be able to see it.

The next stage in developing your herbivore is to write the methods that define your herbivore’s behaviour. Each event handler that is initialized in the constructor of the herbivore defines the actions that it will take each time this program is called by the

Terrarium environment. There are ten event handlers that developers can use in their creature to call methods they define; however, only two of those event handlers are mandatory. They are “LoadEvent” and “IdleEvent”.

The LoadEvent method is run first whenever the herbivore is called. This allows the herbivore to perform basic actions the developer can set; such as search for a plant in the immediate area and target that plant, if one is found. This is just one example of behaviour a developer can implement each time this event occurs.

The IdleEvent method is run while the herbivore is waiting to be called again. During the idle time the herbivore can perform actions such as eating a plant if one is found, reproduce whenever it is able to, and decide on new positions to move to in the environment to search for more plants.

Of course there is a lot more functionality that can be added to a herbivore. The above two events are just the base requirement to produce a functioning herbivore. A herbivore with no more functionality than the base requirement will not last long in the terrarium game. Besides finding food, a herbivore’s other major concern is avoiding carnivores.

3.4 Carnivores

A carnivore in Terrarium is a creature that eats herbivores. The three basic actions a carnivore must perform in order to survive are:

1. Locate and target herbivores
2. Consume herbivores
3. Reproduce

A carnivore has the same set of available properties as a herbivore. The only difference is in the strategies used to distribute the 100 points amongst the seven properties since a carnivore has to find and catch herbivores in order to be able to eat, and unlike a herbivore, a carnivore does not have to worry about being attacked and eaten itself.

A carnivore has as a base the same two events as a herbivore called “LoadEvent” and “IdleEvent”. The main difference in writing methods for a carnivore is that a carnivore must target and eat herbivores, rather than plants. What makes the development of a carnivore more difficult is that it has to locate and catch a moving herbivore for food, while a herbivore has a much easier job of locating and eating plants.

As with a herbivore, the carnivore will need additional functionality added to it if it is going to have a chance at surviving in the environment without starving to death first. This leaves developers with endless options on how to design the characteristics of how their carnivore explores its environment, identifies herbivores, and successfully attacks herbivores.

3.5 Building The Creature

Out of the 20+ languages that .NET supports only Visual Basic and C# are currently supported by Terrarium. All Terrarium code written for this project was in C#. The following instructions explain how to build your creature application using Visual Studio .NET, import the Terrarium creature libraries, and upload a creature to your Terrarium peer. The following is a general outline of steps to create a creature in Visual Studio summarized from the paper "Terrarium Introduction to Creature Development" [1]:

1. Start Visual Studio .NET and select *File -> New -> Project*
2. Select *Visual C# Projects*
3. Select *Class Library* and provide a name for the new class file
4. Enter the C# source code for your creature into the new class file
5. Select *Project -> Add Reference* to add the required Terrarium libraries
6. In the new window select the *.NET* tab then select the library *System.Drawing.dll* by clicking of the line stating this dll file then clicking on the *Select* button
7. Select Browse then locate the file *organismbase.dll* from the *bin* directory in the folder where your Terrarium game was installed (i.e., C:\Program Files\Terrarium\Bin\organismbase.dll) and select *OK*
8. Now build the application by selecting *Build -> Build Solution*
9. If no errors occur you should get a dll file. For example, if the name of your Visual Studio project was Carnivore then your Carnivore dll file will be located in *~myprojects\Carnivore\bin\debug\Carnivore.dll*
10. Start your Terrarium application in either Ecosystem mode or Terrarium mode
11. On the control panel of the Terrarium interface in the bottom left hand corner of the window select the *Terrarium* tab
12. Click on the button *Introduce Animal* then select *Browse*
13. Navigate to the location of a creature's dll file such as Carnivore.dll
14. Select OK, then ten instances of the creature will be loaded into Terrarium

3.6 Poorly Developed Creatures

In the Terrarium game developers must each plan out a strategy for how their creature will interact in the ecosystem. However, strategy alone is not enough, they must implement these strategies using a .NET programming language which is either Visual Basic or C#, but care must be taken to ensure the code is well written.

The ecosystem is not tolerant of flawed creatures. If at any time during the creature's lifespan an exception occurs, that creature is immediately removed from the ecosystem [3]. Efficiency is also important for a creature to be allowed to remain in the ecosystem. If any method in your creature's code ever takes more than two to five milliseconds to execute, it will also be removed from the ecosystem [3]. This time limit for a creature in Terrarium helps prevent any one creature from monopolizing processor time and can prevent an error such as an infinite loop from causing a Terrarium client to crash [4].

The automatic removal of flawed code from the ecosystem forces developers to debug their creature programs before entering them in the game. Since the main purpose of the game is to allow developers to learn to program with the .NET platform, ensuring that

each developer's code does not contain errors or run with little efficiency ensures this learning experience is not diminished.

Once a creature program is uploaded to the ecosystem it is out of the control of the developer and cannot be re-introduced into the environment until all instances of the creature are gone from the ecosystem [3]. To allow for debugging of creatures the Terrarium game provides the stand-alone Terrarium mode so that a developer can test their creatures before running the Ecosystem mode that allows the developer's creatures to be sent out on the network to other peers.

4. Creature Design Strategies

4.1 Design Basics

What is the perfect design for a creature in the Terrarium game? There is no perfect creature design. Just like any other game, much of your strategy depends on the play of the person you are competing against. It is true that you can improve specific skills in a game such as Half-Life by practicing moving around and using maps, but those skills alone do not directly result in success against an opponent.

In a game such as Half-Life a player can adjust their playing strategies throughout a competition based on how their opponent plays. However, in Terrarium players must define their playing strategies in a program called a "Creature" before they enter the game. Once their creature is entered into the game, the developer who created it has no control over the creature's behaviour from that point on. This is what makes Terrarium unique as a game and also makes planning a strategy for game play more difficult than a typical computer game.

In this section some basic strategies for creature design will be discussed. However, since there are potentially too many possible strategies in creature design to even list, only a few will be discussed as examples.

4.2 Properties

When deciding on the importance to assign to each property based on its numerical value, two strategies must be considered: what in general will make a herbivore/carnivore more able to survive and what strategies might the developers creating competing creatures be using. Deciding on a successful set of property values is a combination of these two strategies. You can organize your properties in a way that may seem perfect for the type of creature you are creating (i.e., herbivore), but if there are developers who are designing their creatures (i.e., carnivores) to specifically attack that strategy, then it may fail.

Deciding on each value to assign to a property is partially a process of outwitting your opponent, which is difficult to do when you do not know whom you are competing against or what their strategies will be. An excellent way to get started is to look at the Terrarium Animal Farm to get ideas about the types of strategies typically used for herbivores and carnivores.

4.3 Events

Out of the ten events that a creature can register for, only the LoadEvent and IdleEvent are required to build a creature. All methods that a developer creates to define the functionality of their creature can only be called by an event. A creature has ten events that they can choose to be notified of. Table 1 lists all available events along with descriptions of when they get called.

Event	Function Performed
BornEvent	Called when the creature is first born, then never again
LoadEvent	First event called during a creature's turn
IdleEvent	Called after all other events during a turn have been executed
TeleportedEvent	Called after having been teleported from one peer to another in the Ecosystem
ReproduceCompletedEvent	Called when finished reproducing
EatCompletedEvent	Called when finished eating
DefendCompletedEvent	Called when finished defending from an attack
AttackedEvent	Called when being attacked
AttackCompletedEvent	Called when an attack on another creature has finished
MoveCompletedEvent	Called when arrive at a desired location (i.e., if a herbivore, could be called when arrive at a plant)

Table 1: Events available in Herbivores and Carnivores

4.4 Movement

I. Basics

How the creature moves and explores its environment to either find food or avoid becoming another creature's food is one of the most important aspects of a creatures design. In the Terrarium game, the environment is organized as a grid with an X-axis and Y-axis. The creature can move to any spot on the grid by specifying a coordinate it would like to move to and deciding on the best path to get there based on its current position. Creature movement algorithms can be incredibly complex and result in thousands of lines of code being required, but can be worth the effort if it results in improving the creature's performance.

II. Searching For Food

What a herbivore and carnivore have in common is their constant need to locate food sources. A herbivore has an easier task in finding food since a herbivore eats plants that remain stationary and do not put up a fight when they are eaten. Carnivores have the more difficult task of locating herbivores and having to kill a creature that is capable of fighting back.

There are two common methods for searching the Terrarium environment for food: randomly chosen paths and grid searching. With a randomly chosen path a creature selects a direction to move in at random and then follows that path until it either reaches a source of food or comes to the edge of the Terrarium environment and then randomly

selects another path. The grid searching method allows a creature move from square to square in the Terrarium environment until all sections of the Terrarium have been covered, and then starts over again. These are just two basic strategies.

III. Attacking and Defending

Each creature must also define a strategy for how to move when either attacking another creature or defending from an attack. Since a carnivore must attack a herbivore in order to eat, it must plan how it will move when carrying out the attack. Without a strategy for attacking a creature the carnivore may end up either scaring the herbivore away or just injuring the herbivore before it escapes. A herbivore has to constantly be on guard watching for carnivores and using movement as a way of avoiding being attacked. In the event of an attack one of the best ways a herbivore can defend itself is to move in a way that allows it to escape from the carnivore.

4.5 Creature Property Manipulation

A creature can be designed to modify its own property values while it is in the Terrarium game. However, the catch is that the value for the property stated by the developer is always treated as the maximum value that can be assigned. One common use of this strategy is to adjust the speed that a creature moves at based on its current situation. For example, if a herbivore is searching for food, but is not being threatened by any carnivores in its immediate surroundings, it could move at a slower speed to conserve energy. However, if a carnivore approaches and attacks then the herbivore could set its speed back to the maximum allowed.

4.6 DNA

When a creature lives long enough and has stored enough energy it is then able to reproduce. The offspring inherits from the parent all properties and methods. However, there is a way to adjust the properties of the offspring to make them different from the parent and allow each new offspring to adapt to its environment. Creating evolving DNA for a creature is just one example of the many ways that developers have gone beyond the basics of creature development to be more competitive.

In Terrarium, changing the DNA of each offspring is made possible by the way a creature's information is passed to the new generation. Whenever a creature is mature enough and has enough energy stored to be able to reproduce, the Boolean attribute "canReproduce" is set to true. It is the job of the creature to monitor this attribute; most creatures check this attribute each time the "IdleEvent" is triggered.

When the creature attribute "canReproduce" is true, the method "BeginReproduction" can be called. If the method receives as an input parameter "null" it will take the values of the creature's properties and apply them to the offspring. However, if the method "BeginReproduction" is passed as an input parameter, a byte array, containing the name of each of the seven properties with each property name followed by the new value of the property, will be assigned to the new instance of the creature (offspring).

To implement a creature with evolving DNA that changes over time, a class called DNA must be created. Create a method in the DNA class that allows the name of each property to be entered along with the initial value, lower bound, and upper bound of the property. Add any other required methods to the DNA class as well as a method to return a properly formatted byte array containing each property's new value. The next step is to override the "SerializeAnimal" and "DeserializeAnimal" methods to ensure that the byte array storing the creature properties is used when the creature is serialized when sent and deserialized when received.

How the creature DNA evolves has any number of different strategies. One of the easiest is to have the DNA class randomly select a new value for each property between the defined bounds. Another is to have a set of different properties and alternate which set gets used each time the creature reproduces. Also, a creature can modify the properties that will be assigned based on the creature's interaction with its outside environment. For example, if a herbivore is constantly being threatened by carnivores getting too close or attacking, but has an abundance of food available, the properties may have points removed from energy stored and added to eyesight and speed to improve the offspring's performance. These are just a few examples, all of which require a fair amount of extra code, even if just implementing a simple evolving DNA.

4.7 Cooperative Creatures

Some of the more advanced Terrarium developers design their creatures to cooperate to improve their chances of survival. This strategy is mostly used in the development of carnivores. Since carnivores must find and kill herbivores, a food source that is able to move, carnivores can improve their chances of catching herbivores by hunting in a pack.

One example of a cooperative carnivore is one that identifies another carnivore and patterns its movement based on the movements of that carnivore. When a carnivore finds another carnivore and only if that carnivore is a "friend" (creature of the same species), they can then join together to explore the Terrarium. This allows a group of carnivores of the same species to be able to identify each other and work as a pack in searching the Terrarium for herbivores. This type of cooperation can improve a carnivore's chance of survival by improving its ability to find and kill herbivores. This is just one possible way a developer may create a cooperative creature, however, this approach does require a far more complex creature design.

4.8 Antenna

A creature has one basic way of directly communicating with a "friend"; this is through the use of the creature's antenna. The antenna every creature possesses is used as a visual way for creatures to signal each other. The way creatures are able to communicate is by the setting of the position of their antenna. A developer can define each of the nine possible positions of a creature's antenna to have a specific meaning. This allows all instances of a specific creature that are within visual range to use the position of the antenna of a "friend" to determine the information being conveyed.

For example, if my herbivore had assigned the signal of the antenna pointing straight upward as a sign that it was fleeing from a carnivore, this would allow all of my "friends" within visual range to be able to see this signal and have a chance to quickly move out of the way before the carnivore was able to get close enough to target them. This is just one example, but any meaning can be associated with each of the nine antenna positions.

4.9 Conservative Eating Habits

When a herbivore is eating, it is vulnerable to attacks by carnivores. The most common strategy used to reduce this risk is to increase the property that states the speed at which the creature is able to eat. Unfortunately, by adding points to this property, you must take points away from another property.

One strategy that can be used to reduce the herbivore's risk of attack while eating, without increasing the points assigned to the eating speed property, is to reduce the amount of plant the herbivore consumes in each bite. By default, once a herbivore starts eating it will continue until it has consumed the plant. By adjusting the bite size the herbivore will then only take a set amount of food during each cycle. This allows the herbivore to be able to perform functions such as scanning for carnivores or responding to attacks.

5. Performance Tests and Results

5.1 Test Structure

Testing of the herbivores and carnivores created was done using both the Terrarium mode and Ecosystem mode the game provides. The Terrarium mode allows for an environment where only creatures the local user inserts are in the game and does not allow any other creatures to enter. This allows for the behaviour of the creatures to be closely monitored and tested. Then each creature will be tested in Ecosystem mode to allow them to compete against other developers' creatures.

The base creatures tested were the plant, herbivore and carnivore that are provided by Microsoft as example programs to allow developers to easily get started with creature development. The initial performance results obtained were based on the performance of these base creatures. The second set of performance results was based on the performance of the final set of creatures after their properties and events had been modified to improve upon their original designs.

5.2 Lifespan and Population

In the Terrarium game, population determines the success of a creature. For a creature to be able to increase its population it has to live long enough to be able to reproduce, this is why having a longer lifespan is vital to a creature's success. For this reason the performance of the base and final versions of the creatures were evaluated based on population counts. Tests were run in both Terrarium mode and Ecosystem mode where the population count was taken at set intervals.

5.3 Performance of Base Creatures

I. Terrarium Mode

For the first test of the base creatures Terrarium mode was used. This allowed for the performance of these creatures to be tested in a closed environment. The performance results are shown in Figure 2 and Table 2.

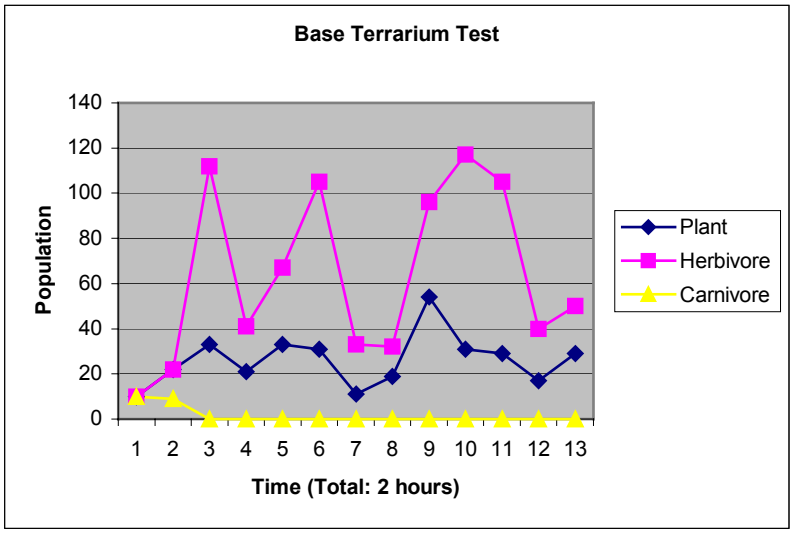

Figure 2: Performance Results of Base Creatures in Terrarium Mode.

Figure 2 shows the population numbers for the base creatures in a local Terrarium environment over a period of two hours. Carnivores did not fair too well mainly due to their inability to successfully locate and kill herbivores. The only reason herbivores were able to survive was due to the abundance of plants and little competition for food. The fluctuations in the herbivore population were due to overeating of plants. When herbivores grew in number they consumed too many plants. Plants dropped in number causing many herbivores to die. This allowed plants to multiply, herbivores to increase, and the cycle to begin again.

Creature	Births	Starved	Old Age	Killed	Sick
Plant	1633	0	25	0	1146
Herbivore	3376	0	44	9	3207
Carnivore	10	0	10	0	0

Table 2: Base Creature Population Statistics in Terrarium Mode.

Table 2 shows statistics on how many of each type of creature was born and how each creature met its demise during the two-hour test session. These statistics are useful in determining the strengths and weaknesses of a creature. When a creature is inserted into the Terrarium ten instances are created. Not a single carnivore was able to reproduce and carnivores were only able to successfully kill nine herbivores. This shows obvious weaknesses with the current carnivore design. Weaknesses were seen in the herbivore

design in that, without an abundance of plants available, herbivores quickly reduced in number. Both of these creatures need improvement.

II. Ecosystem Mode

For the second test of the base creatures Ecosystem mode was used. This allowed the base creatures to compete against all other creatures that are in the peer-to-peer Terrarium system. The performance results are shown in Figure 3 and Table 3.

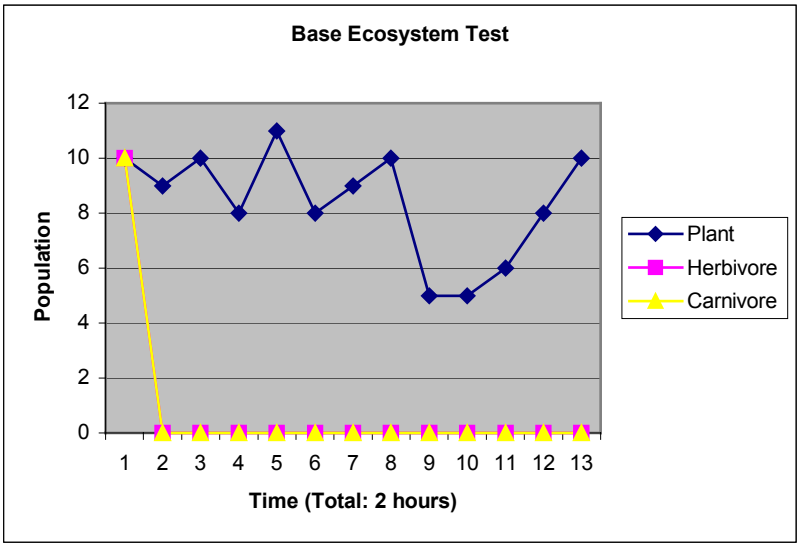

Figure 3: Performance Results of Base Creatures in Ecosystem Mode.

The weaknesses in the current herbivore and carnivore became painfully obvious during this test. Neither creature survived their first ten minutes in Ecosystem mode. Many of the creatures in the Ecosystem have had more strategy and effort put into their design. This did not allow for the herbivore and carnivore to fair well and will result in both requiring significant changes if they are going to be at all competitive in the Terrarium game.

Creature	Births	Starved	Old Age	Killed	Sick
Plant	78	0	0	0	56
Herbivore	11	0	0	8	3
Carnivore	10	7	0	3	0

Table 3: Base Creature Population Statistics in Ecosystem Mode.

As you can see in Table 3, the statistics on each creature's population in the environment show how the herbivore and carnivore came to be extinct. When you release a creature into the Ecosystem ten instances of the creature are born. Table 3 shows that only one herbivore was born beyond the initial startup set of herbivores, and not a single additional carnivore was born. This test demonstrated that the herbivores were not developed enough to avoid carnivores and find food, and that the carnivores were not capable of successfully competing for food.

5.4 Improvements On Base Creatures

I. Properties

The easiest way to set the basis for a creature's strategy is to adjust the values assigned to the creature's properties. Each creature has 100 points that need to be distributed amongst the seven properties that define a creature's behaviour. It is important to plan the distribution of points carefully since any point assigned to one property results in a point being taken away from another. The property value changes for the herbivore are shown in Table 4 and for the carnivore in Table 5.

Property	Old	New	Reason for change
Max Energy	0	20	Need to store some energy between food sources (Max. energy a single plant can provide is 10 points)
Eating Speed	0	2	Need to eat fast and move, are vulnerable to attacks while eating
Attack Damage	0	12	Can fight back if threatened or attacked
Defend Damage	0	12	When an attacker strikes they take on damage
Max Speed	0	24	Need speed to escape from carnivores and get to plants
Camouflage	50	0	No need for camouflage, have speed, eyesight, and can inflict damage on any attacker
Eyesight	50	30	Eyesight is important to spot plants and carnivores, but useless if do not have enough speed to move fast

Table 4: Herbivore Property Changes.

Property	Old	New	Reason for change
Max Energy	0	10	Need to store some energy between food sources
Eating Speed	0	6	Want to get back to searching for food quickly
Attack Damage	52	20	Need to cause damage to a herbivore, but useless if do not have enough eyesight and speed to catch a herbivore
Defend Damage	0	0	No need to defend itself, it is the predator
Max Speed	28	24	Speed is important, but only if it can see a herbivore
Camouflage	0	0	No need to camouflage, have speed and eyesight
Eyesight	20	40	Must be able to see a herbivore, otherwise all other properties are useless

Table 5: Carnivore Property Changes.

II. Events and Methods

There are ten events that a creature can register to be notified of, but only two of these events are mandatory: LoadEvent and IdleEvent. The base creature only uses these two events. A developer is free to use any of the available events that are not mandatory. For the final creatures several other events will be used in both the herbivore and carnivore. This will be done to explore the features of each event and to see how they can each be used to the advantage of the creature.

5.5 Performance of Final Creatures

I. Terrarium Mode

The reason the base creatures were tested in Terrarium mode was to have a chance to observe their behaviour to get ideas on how to modify the creatures. For this reason it was not necessary to run the final creatures in Terrarium mode since the herbivore has been designed to defend against the strategies the carnivore uses and the carnivore has been designed to be able to effectively hunt these herbivores. The best way to test the improved creature designs is to test them in Ecosystem mode against other developers' creatures.

II. Ecosystem Mode

For the test of the final versions of the creatures in Ecosystem mode, both the base and final versions of the herbivore and carnivore were put into the Terrarium game to see if any performance advantage had been gained by the strategy changes made to the final creatures. For the final test there was no reason to include a plant to get the creature started since the local peer was already populated with hundreds of plants, herbivores, and carnivores. The performance results are shown in Figure 4 and Table 6.

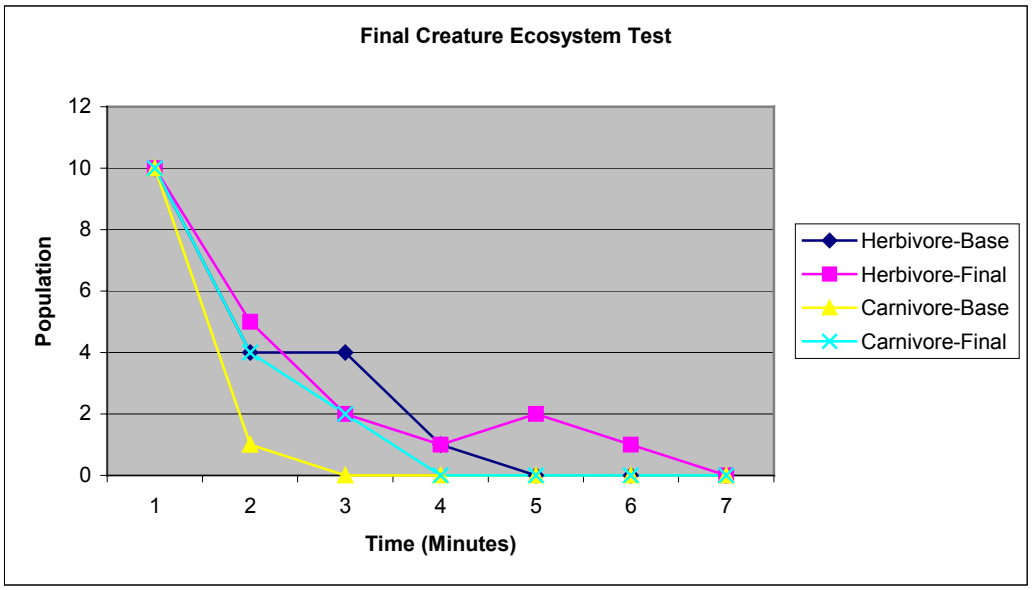

Figure 4: Performance Results for Base and Final Creatures in Ecosystem Mode.

As shown in Figure 4, the base version of the herbivore lasted 5 minutes, while the final version of the herbivore survived 7 minutes before becoming extinct. This may not seem like a big improvement but it is a 40% increase in performance. The base version of the carnivore lasted 3 minutes, while the final version of the carnivore survived 4 minutes, which was a gain of 33% in performance. Considering that mainly creatures from the two top developers participating in the game populate the Terrarium game's Ecosystem mode, this increase in performance is meaningful. The final herbivore and carnivore had to go up against the best creatures in the game even though they were written by a developer who was new to C# and the Terrarium game.

Creature	Births	Starved	Old Age	Killed	Sick
Herbivore - Base	13	0	0	13	0
Herbivore - Final	12	0	0	11	1
Carnivore - Base	10	5	0	5	0
Carnivore - Final	10	0	0	9	1

Table 6: Base Creature Population Statistics in Ecosystem Mode.

5.6 What Caused the Performance Increase?

I. Variable Speed

Variable speed was used to allow each creature to adjust its speed based on its current situation to help conserve energy, since the faster a creature moves, the more energy it uses. If a herbivore has no immediate threats and is just searching for food, it can move at a slower speed to conserve energy, but change its speed to the maximum setting when fleeing from an attacker. If a carnivore has no target herbivores within sight, it can move at a slower speed while it searches for food, then move faster when it has spotted a herbivore to target.

II. Maneuvering Around Objects

When a creature is moving toward a food source and an object such as a rock is in the way, the creature needs some strategy for maneuvering around the object. If the base creatures came across an obstacle blocking their path to a target food source, they would select a different target and start to move in another direction. In the final version of each creature this was changed to allow it to maintain the same target food source by slowly moving counter clockwise around the object.

III. DNA

Unfortunately, due to the competition provided by the existing top creatures in the Ecosystem, neither the herbivore nor carnivore created were able to live long enough to produce more than one generation of the species. This prevented any testing of the DNA code to see if it made any difference at all in the success of the creature's offspring. For this reason the DNA code in both the herbivore and carnivore was removed.

IV. Attacking

For the event that a creature is attacked, it responds with the basic strategy of defending against the attack as well as attempting to attack the attacker. This is the strategy used in the base versions of each creature. This strategy was modified to have the creature move in the opposite direction of the attacker as well as adjust its speed to the highest setting possible by using the code implemented to handle variable speed. The creature, when threatened, will now move a set distance at top speed in the opposite direction of the threat. Also, the creature that attacked is now added to a list of known attackers that is reviewed every time the creature's surroundings is scanned. If a known attacker from the list is present, then the creature can take actions to move away before an attack can occur.

5.7 Performance Discussion

After viewing the performance results you are probably wondering why the performance gain in the new herbivore and carnivore over the base versions was so minor. The reason for the small increase in performance even after the extensive amount of work done to improve the creatures is because of the competition from top developers in the Terrarium game.

Since the Ecosystem is never cleaned up to remove existing creatures and allowed to start over again, only the strongest of creatures reproduce and take over the Ecosystem. This makes it increasingly more difficult for new developers to enter the game since even their first creature must compete against mainly experienced developers who have created some of the game's best creatures.

5.8 Problems with Dominating Creatures

At the moment there are currently a few developers whose creatures are completely dominating the Terrarium game. One developer currently has the top three ranked herbivores in Terrarium. This developer has held this ranking for months and so far has had no challengers come close. As a result, the Ecosystem mode is flooded with mostly just the top creatures and not many others. Any time you log into the Ecosystem mode, within minutes your peer will receive a number of creatures and the vast majority of them will only be instances of the top creatures.

Since the point of the Terrarium game is to have the creature with the largest population, this is the goal that the top developers have been able to achieve. While this may be fun for the few top developers to compete against each other, it prevents any developers who are just starting to learn .NET and would like to improve their skills by creating creatures for the Terrarium game from doing so. Unfortunately, these developers do not have the chance to gradually improve their skills developing creatures over time while playing the game and testing each new creature design they develop. This is like insisting that players who are new to the game of chess are only allowed to play against Garry Kasparov, even though they have never seen the game played. All they know is the basic rules of the game but none of the strategies. This is the same with the Terrarium game. Each new developer is handed the basic rules for the game but absolutely no indication of how to build a successful creature.

Currently the top developer of herbivores in the Terrarium game is David Kocur who has been dominating the list of top three herbivores for months. Almost all herbivores in the Terrarium game belong to David Kocur. This kind of saturation of the system demonstrates David's ability as a herbivore developer, but makes it very difficult for any new developer to start competing in the game. For carnivores the top developer has been Mark Yang. David Kocur uses the names "hm1, hm2, …, hm7" for all of his creatures, while Mark Yang uses "asgard1.1, asgard1.2, …, asgard1.8" to identify his creatures. The screen shot shown in Figure 5 is just an example of the typical saturation of the Ecosystem mode of top herbivores and carnivores.

Figure 5: A typical population of creatures on a peer running Ecosystem mode.

5.9 Herbivore: Problems With Early Creature Designs

I. Maneuvering Around Obstacles

The initial herbivore failed miserably and performed far worse than the base herbivore. After some investigation the problem was found. Whenever the herbivore found an object such as a rock that was in between it and the plant it was targeting, the herbivore would get stuck behind the rock for an infinite amount of time until either the plant it was targeting died, or a carnivore came close enough to cause the herbivore to run away. The reason for this problem is that with all methods in a creature, Terrarium offers a default implementation for each method that you can override. I chose to override all ten methods that are called for the ten events. In the case of movement, my method to deal with objects in the way of a plant was not as robust as the one provided by Terrarium. After several revisions the maneuvering of the herbivore around objects in its path was improved significantly.

II. Fleeing From Carnivores

In my first version of the herbivore, my strategy for dealing with a carnivore was to have the herbivore move as fast as possible in the opposite direction of the carnivore. This strategy worked fine until there were two carnivores, one on each side of the herbivore. This resulted in the herbivore moving back and forth between the two carnivores until it

either starved to death or was attacked and killed. To fix this problem the herbivore was modified to pick a direction that was not blocked by any object or creature and to move at its fastest speed in an attempt to get away.

Another problem with later versions of the herbivore occurred when a carnivore attacked. The herbivore would react by fleeing the location. However, if the herbivore encountered a food source (plant) while it was fleeing, it would immediately ignore the carnivore chasing it and stop to eat the plant. The code that handled the avoiding of carnivores worked well, but this was one weakness that was not foreseen. Since it is imperative that a herbivore be able to get food whenever possible, the code for finding food was set to have the highest priority, but the possibility of food becoming available during the process of avoiding an attack was not considered in the original design.

5.10 Carnivore: Problems With Early Creature Designs

I. Locating Herbivores

A carnivore is more difficult to develop than a herbivore since carnivores must find and catch a food source that can move, while a herbivore does not. This adds another layer of complexity on top of the already difficult task for a creature to identify and consume food sources. The initial version of the carnivore would locate a herbivore and then try and attack it. If the herbivore tried to run away, the carnivore would continue to target the herbivore until it was either out of site, or died. Often while it was chasing a fleeing herbivore it would pass by other herbivores that were closer, but would not switch to attacking a different herbivore after it had already selected a target. To improve this, the carnivore was modified to check its surroundings after each move to allow it to see if another herbivore may be a better choice.

II. Reproduction

In my first attempt at a carnivore, even though the carnivore was able to stay alive for a long enough period to reproduce, it was unable to do so because it was never able to store enough energy to reproduce. The carnivore was designed to constantly be moving and movement takes energy. Also, searching for herbivores and killing a herbivore uses energy. As a result, the carnivore was just able to get enough energy to move from one food source to the next, but without ever storing enough energy to be able to reproduce. This problem was reduced, but not solved, by having the speed that the carnivore moved at adjusted based on its current task, since the faster a creature moves, the more energy it wastes. For example, when the carnivore is just wandering it will move at a slower speed than when it is chasing a herbivore. This allows more energy to be conserved between food sources to help the carnivore build up enough energy to be able to reproduce.

6. Terrarium as a Teaching Tool

6.1 Does The Terrarium Game Teach .NET or Just Creature Strategies

Throughout the process of developing a herbivore and a carnivore the majority of time was spent defining strategies for each creature and then trying to find ways to implement these strategies in C#. At the time little consideration was given to what was being

learned about .NET and C#, but rather all time and effort was spent concentrating on just trying to find ways to implement the creatures to perform in specific ways. It was not until after the creatures were completed that it became apparent that learning the programming language C# and the Visual Studio development environment were a byproduct of the implementation of the creatures for the Terrarium game.

To help developers new to .NET get started, Microsoft provides a working base version of code for each creature that a developer can immediately use to start building their own creatures. Microsoft also provides a set of introductory tutorials on creature development that are helpful for developers new to .NET. One of the best features the Terrarium game provides for developers learning .NET is the zero tolerance for code that contains errors and also of inefficient code. If any creature violates either of these basic rules, the creature is automatically killed by the Terrarium game. This allows developers to learn a new programming language in a development environment, while providing them feedback on the quality of their code and forcing them to improve their code in order to be able to participate in the game.

6.2 Closed Terrarium Network vs. Microsoft's Ecosystem

As stated earlier is this paper, the Terrarium game's Ecosystem mode has been overwhelmed by two top developers. Now that the Terrarium game has been out for more than two years, developers have had the opportunity to perfect their creatures over a long period of time. Unfortunately, this makes it difficult for beginner Terrarium developers to see any meaningful results from using the Ecosystem that will allow them to observe their creature's behaviour and improve upon their design.

The Terrarium game can be used to set up a closed network of Terrarium peers where only those peers are allowed to participate and insert creatures into the game. This allows for a group of developers, such as a group of students who are all taking the same class and possesses similar levels of development experience, to be able to set up a closed game where they can compete against each other. This allows for developers to be able to improve their strategies and skills in an environment where they only have to compete against developers with similar levels of experience.

7. Conclusions

Designing creatures for the .NET Terrarium game is not just for serious computer game players who are also programmers, but is also for developers who are interested in learning .NET in a fun game environment, along with a programming language such as Visual Basic or C#. To some developers this is preferable to the traditional method of learning a new technology by working through the lessons provided in an introductory tutorial. When it comes to herbivore and carnivore development, there is no such thing as a perfect creature design. It is this aspect of the game that makes it so interesting.

After spending time developing strategies for herbivore and carnivore design, what was the most appealing about the game was that the complexity of the creature can be taken as far as the developer is willing to go if they have the talent as a programmer and the

time to spend implementing their design. It is this unbounded creature design that allows the game to increase in complexity as each developer's skill increases.

8. Acknowledgements
Thanks to Dr. Ralph Deters and Christopher A. Brooks for discussions and suggestions.

9. References

[1] Devhood.com (2002). Terrarium .NET: Introduction to Creature Development. Retrieved February 7, 2003, from the World Wide Web: http://www.devhood.com/tutorials/tutorial_details.aspx?tutorial_id=323&printer=t

[2] Loney, Matt (2002). Microsoft game is a code-eating battle. Retrieved February 7, 2003, from the World Wide Web: http://zdnet.com.com/2100-1104-830680.html

[3] Terrarium (2002). Terrarium Creature Templates. Retrieved February 7, from the World Wide Web: http://www.gotdotnet.com/terrarium/docs/Quick Start/fxquickstart.aspx

[4] Terrarium (2002). What Is Terrarium? Retrieved February 8, 2003, from the World Wide Web: http://www.gotdotnet.com/terrarium/whatis/

[5] Lamm, Ehud (2002). Terrarium. Retrieved February 8, 2003, from the World Wide Web: http://lambda.weblogs.com/discuss/msgReader$2969