ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

УХТИНСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Кафедра ИСТ

Курсовой проект

Дисциплина: «Информационные технологии»

Тема:

«Разработка модели анализа и объектно-ориентированного компонента для игры Terrarium 2.0»

Выполнил

студент группы ИСТ-2-04

Зверев Ю.В.

Проверил

доцент кафедры ИСТ, д. т. н.

Истомин М.А.

Ухта 2008

**Содержание**

Введение

Постановка задачи

Задание на курсовой проект

Описание игры Terrarium

Событийная модель игры Terrarium 2.0

Разработка модели поведения животного

Описание поведения животного

Сбор и анализ статистики животного

Заключение

Список литературы

Приложение

Листинг

**Введение**

В рамках данного курсового проекта ставится задача разработки модели анализа и разработки объектно-ориентированного компонента для игры Terrarium2.0. В соответствии с заданием разработанный компонент должен моделировать поведение травоядного животного. При этом основным моментом является возможность собираться в стада с выделением вожака на основе опыта.

Основной целью данного курсового проекта является разработка объектно-ориентированной модели животного, которая объясняется построением модели игры Terrarium. Она построена на базе концепций объектно-ориентированного программирования.

Разработка данной модели подразумевает детальную проработку структуры классов и связей между ними. При этом предполагается использование так называемых шаблонов проектирования, которые наиболее полно отображают структуру разрабатываемой модели, а также обеспечивают распределение обязанностей между объектами.

Разработка компонента, моделирующего поведение животного, производится с использованием технологии .NET, функционирующей на базе платформы Microsoft .NET Framework версии 2.1, посредством среды разработки Microsoft Visual Studio 2008 с использованием языка программирования C#.

Тестирование разработанной модели поведения животного выполняется в системе Microsoft Terrarium версии 2.0, так как именно эта версия системы предназначена для тестирования компонентов, разработанных с использованием технологии .NET, функционирующей на базе платформы Microsoft .NET Framework версии 2.1.

# Постановка задачи

## Задание на курсовой проект

Разработать объектно-ориентированный компонент для игры Terrarium 2.0: «травоядное» с низким уровнем энергии, средним уровнем камуфляжа, высоким зрением и средней скорости, а также с возможностью особей собираться в стада с выделением вожака на основе опыта.

Решение поставленной задачи выполнить в следующих этапах:

1. Изучить документацию на игру, проанализировать поведение травоядного существа, поставляемого вместе с игрой, выделить его задачи (прецеденты) для выживания в экосистеме и успешные и неуспешные сценарии выполнения этих задач.

2. Реализовать модуль с животным на Visual C#

3. Собрать статистику по итогам жизни животного.

Существо определенного вида в игре представляет собой динамически загружаемую библиотеку (Dynamic Load Library – DLL), в которой определена объектная модель, описывающая основные свойства и поведение животного. Библиотека работает на основе платформы .NET Framework. Для различных версий платформы существуют различные версии игры: для платформы .NET Framework 1.0 - версия игры Terrarium 1.0, для платформы .NET Framework 1.1 - Terrarium 1.2, для платформы .NET Framework 2.0 - Terrarium 2.0. Для разработки существа в рамках данного курсового проекта была использована версия платформы 2.1, тестируемая соответственно в игре Terrarium 2.0.

Объектная модель, описывающая основные свойства и поведение животного, реализуется в виде одного или нескольких классов посредством одного из языков, входящих в комплект поставки платформы .NET Framework 2.1 и поддерживающих основные концепции объектно-ориентированного подхода: Visual C#, Visual Basic.

## Описание игры Terrarium

Terrarium – это разработанное фирмой Microsoft приложение, которое представляет собой оболочку для игры, описание которой приведено ниже. Эта оболочка разработана для демонстрации возможностей среды .NET. Смысл игры состоит в разработке системы управления травоядным или плотоядным существом, а так же растением. Существа и растения заселяются (загружаются) в экосистему, которая может быть построена как на одном компьютере, так и на основе взаимодействия большого числа компьютеров.

Игра предоставляет конкурентоспособную среду для испытания различных вариантов существ. Среда является весьма реалистической эволюционной моделью искусственного интеллекта, в которой можно оценить роль различных черт поведения и свойств животных в процессе борьбы за выживание.

Для того, чтобы обеспечить равновесие экосистемы, проектировщики оболочки Terrarium выбрали три различных типа живых существ:

– Растения

– Травоядные

– Хищники (Плотоядные)

Растения питаются естественным солнечным светом террариума. Они служат единственным источником пищи для травоядных.

Травоядные – мало агрессивные животные, способные к мирному совместному проживанию вблизи растений, используемых для питания.

Хищники – это животные которые уничтожают популяцию травоядных и используют их при питании.

Для того, чтобы экосистема успешно существовала, все животные террариума должны жить вместе, иначе система станет потеряет равновесие, и в конечном счете существа умрут от голода.

Жизненный круговорот – единственный путь от рождения до смерти, но и на этом пути есть много моментов, которые ведут к появлению интересных возможностей для разработчиков животных. С момента рождения животные хотят есть, и они должны есть, чтобы жить и расти. С ростом в размере увеличивается и их жизненный опыт, а также способность бороться и защищать себя и свою территорию.

Победившее животное получит необходимую ему еду или доступ к растениям. Без растений или других убитых животных травоядным и хищникам становится нечего есть, и в конечном счете, они будут голодать. Даже полностью здоровые животные все же уязвимы для болезни и рано или поздно умирают. Если болезнь, голод и атаки других особей не приведут к гибели существа, то в конечном счете, оно умрет от старости.

Таким образом, в игре максимально реально отражены условия существования настоящей экосистемы.

## Событийная модель игры Terrarium 2.0

Каждое действие существа или его поведение являются непосредственным результатом некоторого события, вызванного средой Terrarium. С помощью событий все существа оповещаются об изменениях в среде. Например, событие Idle порождается в конце каждого хода. Событие позволяет узнать, когда можно выполнять те или иные действия. События вызывают действия, которые происходят в каждом раунде игры. К событиям, которые порождаются после выполнения существом определенного действия, относятся, например, MoveCompleted, AttackCompleted и EatCompleted. Как правило, существо использует только подмножество событий, предоставляемых средой Terrarium.

Некоторые события, например MoveCompletedEventHanlder, имеют специальные параметры, которые могут применяться для выполнения более сложных действий. Эти специальные параметры существенны, так как они предоставляют информацию о процессе перемещения, о том, какая особь в настоящее время нападает на другую, или, возможно, какую-либо специальную информацию от родительского организма, которая может быть полезна.

Так как существа в игре должны действовать на основе модели поведения, полностью управляемой событиями, важно, чтобы каждое существо понимало, в каком порядке возникают события, как можно обработать несколько событий за один ход, и как избежать ситуаций, в которых менее важное действие выполняется вместо более важного. Каждому существу должны быть назначены те события, на которые оно будет реагировать

События обрабатываются посредством методов класса, описывающего свойства и поведение существа.

# Разработка модели поведения животного

## Описание поведения животного

# Сбор и анализ статистики животного

# Заключение

В рамках данного курсового проекта была поставлена задача разработки компонента для игры Microsoft Terrarium, который моделирует поведение животного–травоядного. При этом основным моментом являлось возможность собираться в стада с выделением вожака на основе опыта.

После разработки было проведено тестирование животного. Во время тестирования собиралась статистика по численности живых организмов на данный момент, количеству организмов, когда-либо существовавших в экосистеме, количеству убитых организмов, количеству умерших от старости.

# Список литературы

1. The Terrarium Ecosystem Overview – материалы страницы http://www.windowsforms.net/Terrarium
2. Фаулер М., Скотт К. UML. Основы. – Пер. с англ. – СПб: Символ-плюс, 2002. – 192 с., ил.
3. Марков С.М., Шалыто А.А.Система управления травоядным существом для игры «Terrarium».С.П.,2003: Санкт-Петербургский государственный институт точной механики и оптики, кафедра «Компьютерные технологии».
4. Нейгель, Кристиан, Ивьен, Билл, Глин, Джей, и др. C# для профессионалов. – М.: Издательский дом «Вильямс», 2006 – 1376 с.
5. Ларман, Крег. Применение UML и шаблонов проектирования. 2-е издание. – М.: Издательский дом «Вильямс», 2004. – 624 с.

# Приложение

##

## Листинг

using System;

using System.Collections;

using System.Drawing;

using System.IO;

using OrganismBase;

[assembly: OrganismClass("Exercise1.MyAnimal")]

[assembly: AuthorInformation("Юра", "YuG.Ermolaev@vorkuta.ru")]

namespace Exercise1

{

[Carnivore(false)]

 [MatureSize(28)]

 [AnimalSkin(AnimalSkinFamily.Beetle)]

 [MarkingColor(KnownColor.Yellow)]

 [MaximumEnergyPoints(14)] // Максимальное количество энергии

 [EatingSpeedPointsAttribute(0)] // Скорость еды

 [AttackDamagePointsAttribute(0)] // Ущерб,наносимый противнику при атаке

[DefendDamagePointsAttribute(0)] // Эффективность защиты

 [MaximumSpeedPointsAttribute(18)] // Максимальная скорость

 [CamouflagePointsAttribute(18)] // Умение прятаться

 [EyesightPointsAttribute(50)] // Дальнозоркость

 // CLASS

public class MyAnimal : Animal

{ private PlantState targetPlant = null; // Растение

// private AnimalState targetAnimal = null; // Животное

AnimalState attackerAnimal = null; // Кто атакует

const int cruisingSpeed = 5; // Скорость нормального хода

const int fleeingSpeed = 40; // Скорость при беге

protected override void Initialize()

{Idle += new IdleEventHandler(IdleEvent);

// Load += MyAnimal\_Load;

Attacked += new AttackedEventHandler(AttackedEvent);

MoveCompleted += new Move Completed Event Handler(Move Completed Event);

}

// // Загрузка организма

// private void MyAnimal\_Load(object sender, LoadEventArgs e)

// {

// }

void IdleEvent(object sender, IdleEventArgs e)

{ try

{ MyWriteTrace();

if (CanReproduce)

{ WriteTrace("Я размножаюсь");

BeginReproduction(null);

}

// Если мы можем есть и имеем еду, то мы едим

if (CanEat)

{ WriteTrace("Я голодный.");

if (!IsEating)

{ WriteTrace("Нет еды: Ищем еду?");

if (targetPlant != null)

{ WriteTrace("Да, я вижу еду.");

if (WithinEatingRange(targetPlant))

{ WriteTrace("Имеем еду, старт поедания.");

BeginEating(targetPlant);

if (IsMoving)

{ WriteTrace("Остановимся, пока едим.");

StopMoving();

}

}

else

{ if (!IsMoving)

{ WriteTrace("Перемещаемся к еде");

BeginMoving(new MovementVector(targetPlant.Position, cruisingSpeed));

}

}

}

else

{WriteTrace("Не могу найти растение.");

if ((!ScanForTargetPlant())&&(!TraceScan()))

if (!IsMoving)

{if (!TraceScan())

{int RandomX = OrganismRandom.Next(0, WorldWidth - 1);

int RandomY = OrganismRandom.Next(0, WorldHeight - 1);

BeginMoving(new MovementVector(new Point(RandomX, RandomY), cruisingSpeed));

}

}

else

{WriteTrace("Перемещаемся и смотрим...");

}

}

}

else

{

WriteTrace("О! Еда.");

if (IsMoving)

{ WriteTrace("Остановимся пока кушаем.");

StopMoving();

}

}

}

else

{ WriteTrace("Воот: ничего не делаем.");

if ((IsMoving)&&(!TraceScan()))

StopMoving();

TraceScan();

}

}

catch (Exception exc)

{ WriteTrace(exc.ToString());

}

// if (!IsMoving)

// {

// Int32 RandomX = OrganismRandom.Next(0, WorldWidth - 1);

// Int32 RandomY = OrganismRandom.Next(0, WorldHeight - 1);

// BeginMoving(new MovementVector(new Point(RandomX, RandomY), 4));

// }

// else

// {

// };

}

void MyWriteTrace()

{ WriteTrace("Мой возраст = ", State.TickAge, " хода."); }

private bool ScanForTargetPlant()

{ try

{ ArrayList foundCreatures = Scan();

// TraceScan();

 if (foundCreatures.Count > 0)

{ // Всегда двигаемся подле самого близкого растения или ремовим его

foreach (OrganismState organismState in foundCreatures)

{ if (organismState is PlantState)

{ targetPlant = (PlantState)organismState;

BeginMoving(new MovementVector(organismState.Position, cruisingSpeed));

return true;

}

}

}

}

catch (Exception exc)

{ WriteTrace(exc.ToString());

}

return false;

}

// Вызывается, когда существо было атаковано

void AttackedEvent(object sender, AttackedEventArgs e)

{

if (e.Attacker.IsAlive)

{ Vector newVector = Vector.Subtract(attackerAnimal.Position,Position);

Vector newPositionVector = newVector.Scale(10);

Point newPosition = Vector.Add(Position, newPositionVector);

BeginMoving(new MovementVector(newPosition, fleeingSpeed));

}

}

// Запускается, когда закончили двигаться.

private void MoveCompletedEvent(object sender, MoveCompletedEventArgs e)

{ TraceScan();

}

private bool TraceScan()

{ WriteTrace("СКАНИРУЮ():");

ArrayList OrgArr = Scan();

Int32 ii=0;

Int32 Tick = State.TickAge;

for (Int32 I = 0; I < OrgArr.Count; I++)

{ OrganismState Org = OrgArr[I] as OrganismState;

String S;

if (Org is PlantState)

S = "Растение ";

else

{

if ((Org as AnimalState).AnimalSpecies.IsCarnivore)

S = "Хищник ";

else

{ WriteTrace("Смотрю кто старше");

if (((Org as AnimalState).TickAge > Tick) && (Org.Species.IsSameSpecies(this.Species)))

{ ii = I;

Tick = (Org as AnimalState).TickAge;

}

S = "Травоядный ";

}

}

WriteTrace(S);

}

if (Tick > State.TickAge)

{ WriteTrace("Вижу вожака на основе опыта. Иду следом");

OrganismState Org = OrgArr[ii] as OrganismState;

BeginMoving(new MovementVector((Org as AnimalState).Position, cruisingSpeed));

return true;

}

else

{ return false;

}

}

public override void SerializeAnimal(MemoryStream m)

{

}

public override void DeserializeAnimal(MemoryStream m)

{

}

}

}