Санкт-Петербургский государственный институт точной механики и оптики (технический университет)

Кафедра «Компьютерные технологии»

С.М. Марков, А.А. Шалыто

Система управления травоядным существом для игры «Terrarium»

Объектно-ориентированное программирование с явным выделением состояний

Проект создан в рамках
«Движения за открытую проектную документацию»
http://is.ifmo.ru

Санкт Петербург
2003

Содержание

1. Введение

В данной работе рассматривается применение технологии автоматного программирования [1] для моделирования травоядного существа в игре *Terrarium* [2]. Применение автоматной модели, основанной на состояниях, позволило разработать простое для понимания, но в то же время достаточно продвинутое существо.

Анализ ссылок в форуме на сайте [2] позволил выбрать соперников для разрабатываемого существа. При этом оказалось, что для многих существ приведены открытые коды, максимум с небольшими комментариями. Проектная документация на эти существа отсутствует.

Цель настоящей работы состоит в разработке существа и проектной документации к нему, что позволит при необходимости значительно проще изменить его функциональность, чем для других известных существ.

2. Что такое 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. После этого 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. Подробно с этой игрой можно познакомиться в работе [2], а о самой оболочке и установке игры - в работе [3].

Для установки игры необходимо установить платформу *.NET* и среду *Terrarium*. Данные продукты можно скачать с сайта [2]. Это требует около 30 мб места на диске и

установки операционной системы *Windows 98/Me/NT/2000/XP*. Подробно об установке среды *Terrarium* изложено на сайте [2].

Внешний вид игры *Terrarium* приведен на рис.1. На этом рисунке есть растения *davplant 1* (названия не приводятся), травоядные *hm4, hm5, hm7* и плотоядные существа *asgard 1.7*. В ходе выполнения работы создано несколько версий травоядного существа с названием *smm*. На рисунке используются разработанные нами травоядные существа версии 2.15 (*smm 2.15*).

Обратим внимание, что на этом рисунке есть как маленькие, так и выросшие хищники.

Рис. 1. Внешний вид игры *Terrarium*

3. Жизненный круговорот

В игре представлен сбалансированный рост различных существ на одной территории под названием "террариум". Это означает, что множество различных типов существ должны сосуществовать, бороться за территорию или пищу и размножаться для поддержания популяции. Этот принцип известен как круговорот жизни.

Для того, чтобы обеспечить равновесие экосистемы, проектировщики оболочки *Terrarium* выбрали, как отмечено выше, три различных типа живых существ.

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

Травоядные - малоагрессивные животные, способные к мирному совместному проживанию вблизи растений, используемых ими для питания.

Существа, которых следует бояться, - это хищники, которые могут быстро уничтожить всю популяцию травоядных и использовать их при питании, чтобы вскормить новое поколение.

Для того, чтобы экосистема успешно существовала, все эти создания должны жить вместе, иначе система станет неуравновешенной, и в конечном счете существа умрут от болезни и 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 существ. С момента рождения существа хотят есть, и они должны есть, чтобы выживать и расти. С ростом в размере увеличивается и их жизненный опыт, а также способность бороться и защищать себя и свою территорию. Чаще всего эти сражения будут кончаться смертью по крайней мере одного животного. 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.

Но даже в смерти есть жизнь - смерть выросших и старых существ приводит к увеличению доступного свободного пространства и продовольствия для молодых поколений, которые появляются в результате размножения зрелых особей и продолжают жизненный круговорот.

Выше перечислены основные особенности среды. Со всеми ее свойствами можно ознакомиться в работе [4].

4. Варианты игры Terrarium

Данная игра может происходить как в распределенном варианте, когда одновременно в сети находится большое количество компьютеров (обычно около 100), либо только на локальном компьютере.

Распределенный вариант игры интересен тем, что внеся существо на свой локальный компьютер, оно может быть перенесено на другие компьютеры подключенные к сети *Terrarium.*. Данная сеть называется экосистемой (ecosystem). Существа переносятся на другие компьютеры при помощи телепортации.

Телепортация осуществляется при помощи фиолетового шара (рис.2), который наезжая на существо, переносит его либо на другой компьютер с запущенной средой *Terrarium (*в распределенном варианте), либо просто в другую часть экрана (когда игра происходит на одном компьютере).

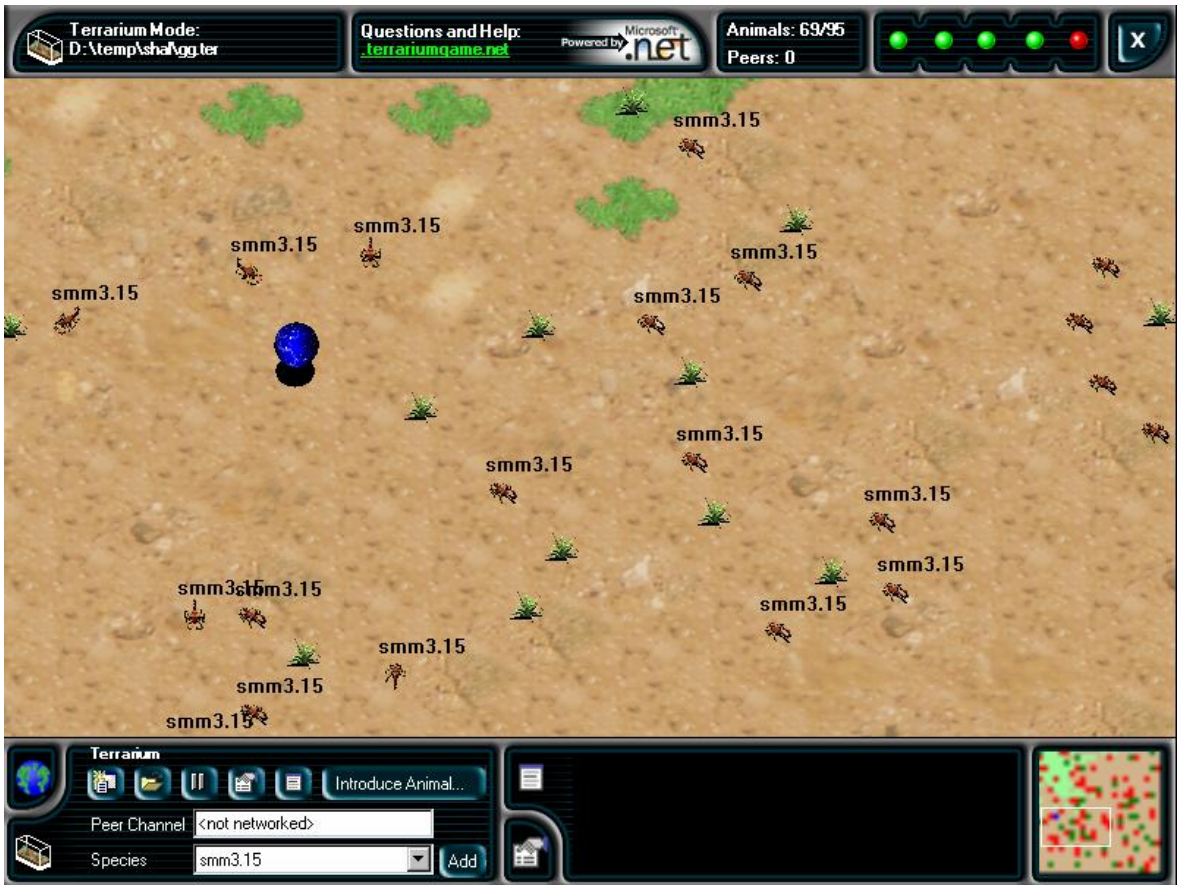

Рис. 2. Телепортация в игре *Terrarium*

5. Событийная модель Terrarium

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*. Существуют также события, которые происходят только один раз для каждого существа, например, событие *рождение* (*Born).*

Каждому существу должны быть назначены те события, на которые оно будет реагировать. Существу может быть назначено одно событие, например, *Idle*, или много

событий. 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.

Так как у нас не стояла задача привести полное описание модели *Terrarium*, то с ней можно ознакомиться на сайтах [6, 7].

6. Разрабатываемое существо

В работе принято решение разработать одно травоядное существо. Данный выбор не влияет на подход к решению задачи. Его можно применить для растений и хищников, хотя состояния, входные и выходные параметры, естественно, будут другими.

Цель игры состоит в разработке такого существа, популяция которых может наилучшим образом противостоять окружающему миру. Поэтому в начале игры среда вводит в систему сразу по несколько существ каждого из выбранных типов. Среда ведет подсчет очков, отражающих успешность жизни популяции. Количество типов существ одновременно находящихся в игре может быть от двух до нескольких десятков.

7. Стратегия игры

Разрабатываемое существо решает несколько задач, которые оно должен выполнять, для того, что бы выжить.

Первое и самое важное – это поиск пиши. Поэтому каждый ход существо осматривает окрестность для поиска пиши. В случае, если пища (растения) находится, то разрабатываемое существо идет к ней и начинает есть. Причем для того, что бы жить в балансе с окружающим миром существо не должно есть растение до конца, так как иначе растение умрет. Разрабатываемое существо отслеживает степень своего голода и уровень повреждения растения, и на основе этого принимает решение о продолжении еды. Отметим, что в данной игре нет необходимости в контакте при еде.

В то же время, если рядом есть другие травоядные, существо съедает растение полностью. Такая стратегия себя оправдывает, так как ее применение часто приводит к экологическому дисбалансу, в результате которого все растения и животные, кроме нескольких растений и нескольких экземпляров разработанного существа, вымирают.

Вторая по важности задача состоит в необходимости убегать от плотоядных существ. Как только травоядное замечает плотоядное существо, оно сразу принимает защитные меры и делает попытку убежать. При разработке существа было принято решение убегать от хищника в противоположном направлении.

Третьим фактором является возможность размножения, и как только появляется такая возможность (например, если существо не убегает), оно сразу начинает размножаться. Существо размножается так, что в результате размножения рядом с ним появляется его клон.

8. Описание подхода

1. На основе документации на игру и анализа возможного поведения плотоядного существа выделяются состояния, в которых оно может находиться в течение своей жизни.
2. Для каждого состояния выбирается название.
3. Рассматриваются входные переменные для автомата, управляющего существом. Разрабатывается их словесные описания.

4. Определяются выходные действия, формируемые в состояниях и на переходах между состояниями. Разрабатывается их словесные описания.
5. Строится граф переходов.
6. На основе графа переходов разрабатывается программа.

9. Автомат управления существом

9.1. Входные переменные

Таблица 3

Обозначение	**Название**	**Описание**
x1	Видно растение	Есть ли растение в зоне видимости? Если есть, то занести его в *targetPlant*
x2	Растение подходит для еды	Проверяется, не нанесен ли большой урон растению
x3	Растение близко, его можно есть	Находиться ли растение достаточно близко, для того чтобы его есть?
x4	Плотоядное в области видимости	Есть ли плотоядное в зоне видимости? Если да, то занести ближайшего в *attackerAnimal*
x5	Можно размножаться	Может ли существо размножаться и достаточно ли энергии для этого?
x6	Плотоядного не видно	Нет ли плотоядного в зоне видимости?
x7	Рядом другое травоядное	Есть ли рядом травоядное другого типа?
x8	Существует ли растение, к которому существо идет	Существует ли растение указанное в *targetPlant?*
x9	Остановка в результате блокировки	Если x15=1, то было ли причиной остановки блокировка?
x10	BornEventHandler	Устанавливается, если это первый ход
x11	AttackCompletedEventHandler	Устанавливается после окончания атаки
x12	EatCompletedEventHanlder	Устанавливается, когда существо закончило есть
x13	ReproduceCompletedEventHandler	Вызывается, когда воспроизводство потомства было закончено
x14	AttackedEventHandler	Устанавливается при атаке разрабатываемого существа другим существом. Занести информацию об атакующем существе в *attackerAnimal*
x15	MoveCompletedEventHandler	Вызывается при окончании движения
x16	DefendCompletedEventHandler	Вызывается при окончании подготовки к защите

9.2. Выходные переменные

Таблица 4

Обозначение	**Название**	**Описание**
z1	Защищаться	Защищаться от существа, информация о котором хранится в переменной ***attackerAnimal***
z2	Гулять	Идти в случайном направлении. Присвоить координаты точки, к которой идем, переменной ***currentDestionation***
z3	Идти к еде	Идти к еде (***targetPlant)***. Присвоить координаты точки, к которой идем, переменной ***currentDestionation***
z4	Есть	Есть растение (***targetPlant)***
z5	Воспроизводиться	Воспроизводиться
z6	Бежать	Убегать от существа (***attackerAnimal).*** Присвоить координаты точки, к которой идем, переменной ***currentDestionation***
z7	Стоять	Ничего не делать
z8	Сменить направление	Повернуться на 90 градусов влево от точки (***currentDestination)***
z9	Вернуть направление	Идти к точке (***currentDestination)***

9.3. Схема связей

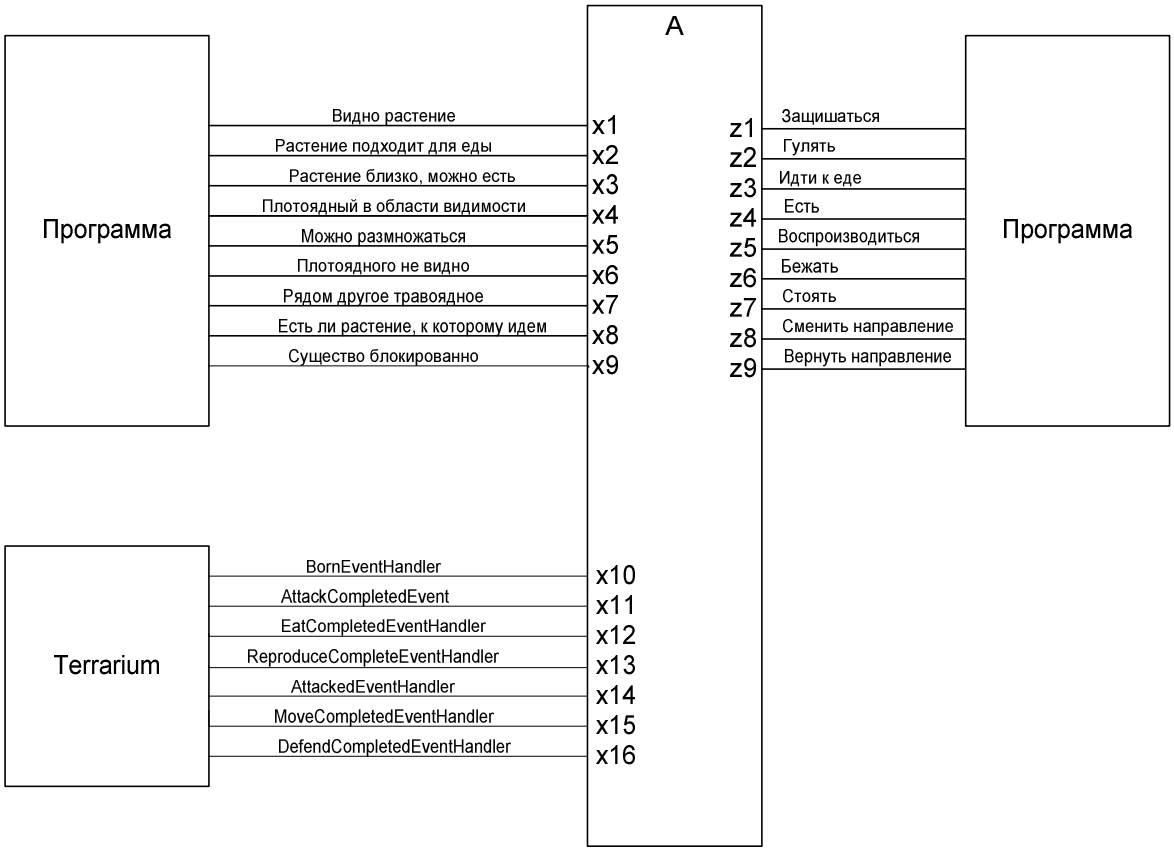

Рис. 3. Схема связей

9.4. Граф переходов

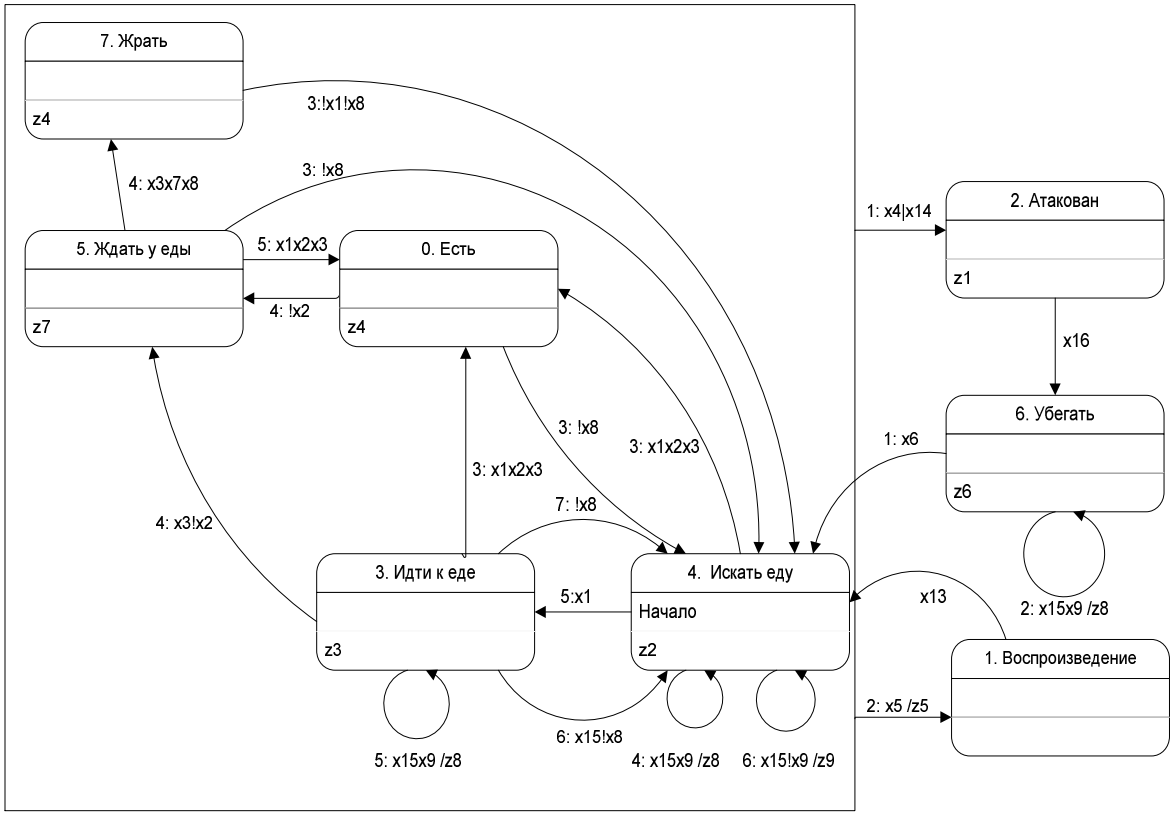

Рис. 4. Граф переходов, управляющий травоядным существом

10. Построение программы

Программа реализована на языке C#. Она представляет собой один класс (*SMMCRITTER*). Автомат является одним из методов этого класса. Он реализован формально и изоморфно по графу переходов. Автомат вызывается (запускается) по событию *Idle*, которое приходит из среды последним в каждом ходе. Отметим, что эволюция проектирования программы привела к тому, что начальным состоянием является не нулевое состояние, а четвертое. Отметим также, что действия в вершине выполняются после перехода в нее.

Установка значений входных переменных *x1 – x8* происходит в функциях *fx1() – fx8()*, которые практически не имеют внутренней логики. Данные функции вызываются программой перед запуском автомата.

Событийная модель игры описана в разд.5. Выбранные в функции *Initialize()* события, которые среда посылает программе каждый ход (тик), преобразуются в соответствующих обработчиках во входные переменные *x9 — x16,* опрашиваемые автоматом после его запуска. Отметим, что обработчики событий практически не имеют собственной логики.

11. Результаты игр

После разработки данной программы были проведены тестовые запуски с различными существами. Ниже приведены примеры запуска разработанного существа *smm 3.15* с хищником *asgard 1.7,* растением *davplant 1* и другим плотоядным *hm7*. Вначале среда разместила в экосистеме по 10 особей каждого вида. В табл. 5 можно увидеть, как развивались события с течением времени.

Таблица 5

	Время, мин			
Название	10	30	90	120
smm 3.15	19	24	27	22
asgard 1.7	12	13	12	14
davplant 1	18	16	22	21
hm7	15	20	19	23

Отметим, что в таблице, которую выдает игра, значительно больше характеристик существ по сравнению с табл. 5, например, сколько особей умерло, сколько родилось и т.д.

12. Заключение

Использование автоматного подхода позволило централизовать логику весьма сложной игры, тем самым, упростив программирование. Данный подход значительно упростил также отладку (поиск ошибок в логике программы) благодаря тому, что все несоответствия между предполагаемым и реальным поведением сразу удавалось обнаружить. Это связано с тем, что граф переходов является паттерном предполагаемого поведения, а структура программы полностью соответствует графу.

Обратим внимание, что авторами был пройден путь от версии 1.1 до версии 3.15. При этом, если в первой версии вся логика была выполнена на флагах, то в последней - их практически полностью заменил автомат.

В заключение отметим, что хотя в Интернете размещены различные варианты существ, они все не имеют внятного описания, а тем более, проектной документации [9]. Данный проект, возможно, является первым, в котором разработано существо с формально специфицированным поведением и полной документацией.

Литература

1. *Сайт кафедры* "Информационные системы" СПбИТМО (ТУ) http://is.ifmo.ru
2. ***Terrarium*** Home http://www.windowsforms.net/***Terrarium***
3. ***Terrarium*** GUI Walk-Through http://www.windowsforms.net/Terrarium/docs/GUI/
4. ***Terrarium*** Advanced Developer Guide http://www.windowsforms.net/Terrarium/docs/OrganismSDK/default.aspx
5. *Туккель Н.И., Шалыто А.А.* Система управления танком для игры ***Robocode***. Объектно-ориентированное программирование с явным выделением состояний http://is.ifmo.ru/?i0=projects&i1=tanks
6. *Дистель А.А., Кобак Д.А., Шалыто А.А.* Система управления дорожным светофором. Программирование с явным выделением состояний.
7. *Головешин А.* Использование конвертора Visio2SWITCH http://www.softcraft.ru/auto/switch/v2s.shtml
8. ***Terrarium*** Object Model http://www.windowsforms.net/Terrarium/docs/ObjectModel/default.aspx
9. Борушевский Д. Награды за «офис» и «террариум». Computerworld, #15-16/2002 http://www.osp.ru/cw/2002/15-16/015.htm

Приложение. Листинг программы

```
using System;
using System.Drawing;
using System.Collections;
using System.IO;


[assembly: OrganismClass("SMMCRITTER")]    // Название класса

// Данные о разработчике
[assembly: AuthorInformation("Sergey Markov", "sergey@etherscan.com")]

[CarnivoreAttribute(false)]       // Разрабатываемое травоядное существо

[AnimalSkin("Beetle")]
[MarkingColor(KnownColor.Green)]

// Разрабатываемое существо среднего размера
[MatureSize(26)]

// Point Based Attributes

// Среда по умолчанию присваивает некоторые значения перечисленым ниже
// параметрам и предоставляет возможность дополнительно распределить
// 100 очков
[MaximumEnergyPoints(20)]   // Максимальное количество энергии
[EatingSpeedPoints(0)]      // Скорость еды
[AttackDamagePoints(0)]     // Ущерб,наносимый противнику при атаке
[DefendDamagePoints(0)]     // Эффективность защиты
[MaximumSpeedPoints(40)]    // Максимальная скорость
[CamouflagePoints(0)]       // Умение прятаться
[EyesightPoints(40)]        // Дальнозоркость


public class SMMCRITTER : Animal
{

      // Глобальные параметры

      // Растение, к которому приближаемся
      PlantState targetPlant = null;
      AnimalState attackerAnimal = null;      // Кто атакует
      MovementVector currentDestination=null; // Направление движения
      // Временное направление движения
      MovementVector tempDestination=null;
      const int cruisingSpeed = 5;            // Скорость нормального хода
      const int fleeingSpeed = 40;            // Скорость при беге

      // Параметр блокировки движения
      MoveCompletedEventArgs e;

      // Входные воздействия автомата
      bool x10, x11, x12, x13, x14, x15, x16;
      bool x1, x2, x3, x4, x5, x6, x7, x8, x9; // Входные переменные

      int y0;     // Переменная, кодирующая состояния автомата
```

```
//Выходные воздействия автомата

void z1()   //Защищаться
{
      BeginDefending(attackerAnimal); // Начать защищаться
}

void z2() //Гулять
{
      if(currentDestination==null)
      {
            int RandomX= OrganismRandom.Next(0, WorldWidth - 1);
            int RandomY= OrganismRandom.Next(0, WorldHeight - 1);
            currentDestination=new MovementVector(new
                  Point(RandomX,RandomY), cruisingSpeed);
            BeginMoving(currentDestination);
      }
      else
            BeginMoving(currentDestination);

}

void z3() //Идти к еде
{
      BeginMoving(new MovementVector(targetPlant.Position, 2));
}

void z4() //Есть
{
      BeginEating(targetPlant);
}

void z5() //Воспроизводиться
{
      BeginReproduction(null);
}

void z6() //Бежать
{
      Vector newVector = Vector.Subtract(attackerAnimal.Position,
            Position);
      Vector newPositionVector = newVector.Scale(10);
      Point newPosition = Vector.Add(Position, newPositionVector);
      BeginMoving(new MovementVector(newPosition, fleeingSpeed));
}

void z7() //Стоять
{

}

void z8() //Сменить направление
{
      //currentDestination=null;
      BeginMoving(getDestination());
}

void z9() //Вернуть направление
{
      if(currentDestination!=null)
            BeginMoving(currentDestination);
}
```

```
// Установить какие события обрабатываются
protected override void Initialize()
{
      Load += new LoadEventHandler(LoadEvent);
      Idle += new IdleEventHandler(IdleEvent);
      Attacked += new AttackedEventHandler(AttackedEvent);
      Born += new BornEventHandler(BornEvent);
      MoveCompleted +=
            new MoveCompletedEventHandler(MoveCompletedEvent);
      AttackCompleted += new
            AttackCompletedEventHandler(AttackCompletedEvent);
      ReproduceCompleted += new
            ReproduceCompletedEventHandler(ReproduceCompletedEvent);
}

// Вызывается, если существо было блокировано

void MoveCompletedEvent(object sender, MoveCompletedEventArgs e)
{
      x15=true;
      if(e.Reason == ReasonForStop.Blocked)
            x9=true;
      else
            x9=false;
}

// Вызывается при рождении существа

private void BornEvent(object sender, BornEventArgs e)
{
      y0=4; // Начальное состояние
}

// Сброс переменных

void InitVar()
{

      x1=false;
      x2=false;
      x3=false;
      x4=false;
      x5=false;
      x6=false;
      x7=false;
      x8=false;
      x9=false;
      x10=false;
      x11=false;
      x12=false;
      x13=false;
      x14=false;
      x15=false;
      x16=false;

}

// Инициализация перед каждым ходом
void LoadEvent(object sender, LoadEventArgs e)
{
      InitVar();
      FillVars();
}
```

```
// Вызывается, когда существо было атаковано
void AttackedEvent(object sender, AttackedEventArgs e)
{
      if(e.Attacker.IsAlive)
      {
            x14=true;
      }
}

public void AttackCompletedEvent(object sender,
      AttackCompletedEventArgs e)
{
}

// Воспроизведение прошло успешно
public void ReproduceCompletedEvent(object sender,
      ReproduceCompletedEventArgs e)
{
      x13=true;
}

//Видно растение
void fx1()
{
      ArrayList foundCreatures = Scan();
      if(foundCreatures.Count > 0)
      {
            foreach(OrganismState organismState in foundCreatures)
            {
                  if(organismState is PlantState)
                  {
                        targetPlant = (PlantState) organismState;
                        x1=true;

                        return;
                  }
            }
      }
}

//Растение подходит для еды
void fx2()
{
      if(x1&&targetPlant.PercentInjured < 0.7
            || State.EnergyState == EnergyState.Deterioration)
                  x2=true;
}

//Растение близко, можно есть
void fx3()
{
      if(x1&&WithinEatingRange(targetPlant))
            x3=true;
}
```

```
//Плотоядное существо в области видимости

void fx4()
{
      ArrayList foundAnimals = Scan();
      if(foundAnimals.Count > 0)
      {
            foreach(OrganismState organismState in foundAnimals)
            {
                  if(organismState is AnimalState)
                  {
                        IAnimalSpecies iAS =
                              (IAnimalSpecies) organismState.Species;
                        if (iAS.IsCarnivore)
                        {
                              attackerAnimal =
                                    (AnimalState) organismState;
                              x4=true;

                        }
                  }
            }
      }
}

//Можно размножаться
void fx5()
{
      if(CanReproduce)
            x5=true;
}

//Плотоядного не видно
void fx6()
{
      x6=!x4;
}

//Рядом другое травоядное
void fx7()
{
      ArrayList foundAnimals = Scan();
      if(foundAnimals.Count > 0)
      {
            foreach(OrganismState organismState in foundAnimals)
            {
                  if(organismState is AnimalState)
                  {
                        IAnimalSpecies iAS =
                              (IAnimalSpecies) organismState.Species;
                        if (!iAS.IsCarnivore&&
                                    !IsMySpecies(organismState))
                        {
                              x7=true;
                        }
                  }
            }
      }
}
```

```
//Есть ли растение, к которому идем
void fx8()
{
	if(targetPlant != null)
	{
		targetPlant = (PlantState) LookFor(targetPlant);
		if(targetPlant != null)
			x8=true;
	}
}

// Установка значений входных переменных
void FillVars()
{
	fx1();
	fx2();
	fx3();
	fx4();
	fx5();
	fx6();
	fx7();
	fx8();
}

//Автомат
void switchIdle()
{

	int yold=y0;

	switch(y0)
	{
		case 0:
			if(common())
			if(!x8)
				y0=4; else
			if(!x2)
				y0=5;
			break;

		case 1:
			if(x13)
				y0=4;
			break;

		case 2:
			if(x16)
				y0=6;
			break;

		case 3:
			if(common())
			if(x1&&x2&&x3)
				y0=0; else
			if(x3&&!x2)
				y0=5; else
			if(x15&&x9)
			{
				z8();
				y0=3;
			} else
			if(x15&&!x9||!x8)
				y0=4;
			break;
```

```
	case 4:
		if(common())
		if(x1&&x2&&x3)
			y0=0; else
		if(x15&&x9)
		{
			z8();
			y0=4;
		}	else
		if(x1)
			y0=3; else
		if(x15&&!x9)
				z9();
		break;

	case 5:
		if(common())
		if(!x8)
			y0=4; else
		if(x7&&x3&&x8)
			y0=7; else
		if(x1&&x2&&x3)
			y0=0;
		break;

	case 6:
		if(x6)
			y0=4; else
		if(x15&&x9)
			z8();
		break;

	case 7:
		if(common())
		if(!x1|!x8)
			y0=4;
		break;
}


printState();

switch(y0)
{
	case 0:
		z4();
		break;

	case 1:

		break;
	case 2:
		z1();
		break;
	case 3:
		z3();
		break;
	case 4:
		z2();
		break;
	case 5:
		z7();
		break;
```

```
            case 6:
                  z6();
                  break;
            case 7:
                  z4();
                  break;

      }

}

// Реализация групповых переходов
bool common()
{
      if(x4||x14)
            y0=2; else
      if(x5)
      {
            z5();
            y0=1;
      }
      else
            return true;

      return false;

}


// Вызов автомата, осуществляемый в конце хода
void IdleEvent(object sender, IdleEventArgs e)
{
      try
      {
            switchIdle(); // Вызов автомата
      }
      catch (Exception ex)
      {
            WriteTrace("Exception in switchIdle : {0}\n",ex.ToString());
      }
}


// Поворот на 90 градусов
MovementVector getDestination()
{
      Point originalDestination =  currentDestination.Destination;
      Vector originalVector =
            Vector.Subtract(originalDestination, this.Position);
      Vector newVector = originalVector.Rotate(Math.PI / 4);
      Vector unitVector = newVector.GetUnitVector();
      Vector newPositionVector = unitVector.Scale(20);
      Point newPosition =
            Vector.Add(Position, newPositionVector);
      MovementVector destination=
            new MovementVector(newPosition, cruisingSpeed);
      tempDestination=destination;

      return destination;
}
```

```
        // Выводит информацию для отладки
        void printState()
        {
              WriteTrace("State: "+y0+" ");
              if(x1)
                    WriteTrace("Plant is visible");

              if(x2)
                    WriteTrace(", Plant is eatable ");

              if(x3)
                    WriteTrace(", Plant can be eaten");
              if(x4)
                    WriteTrace(", Carnivore is visible");
              if(x5)
                    WriteTrace(", CanReproduce");
              if(x6)
                    WriteTrace(", Carnivore is not visible");
              if(x7)
                    WriteTrace(", Alien Herbivore is visible");
              if(x9)
                    WriteTrace(", Blocked");
              if(x8)
                    WriteTrace(", Plant Exist");

              if(x10)
                    WriteTrace(", x10");
              if(x11)
                    WriteTrace(", x11");
              if(x12)
                    WriteTrace(", x12");
              if(x13)
                    WriteTrace(", x13");
              if(x14)
                    WriteTrace(", x14");
              if(x15)
                    WriteTrace(", x15");
              if(x16)
                    WriteTrace(", x16");
        }

        // Служебные функции
        public override void SerializeAnimal(MemoryStream m)
        {
        }

        public override void DeserializeAnimal(MemoryStream m)
        {
        }
}
```