Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение   
высшего образования   
«Алтайский государственный технический университет им. И.И. Ползунова»

Факультет информационных технологий

Кафедра прикладной математики

Отчет защищен с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель от вуза\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.Ю. Андреева

(подпись) (и.о., фамилия)

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_2022 г.

(дата)

Отчет об учебной (технологической) практике

«Имитационное моделирование популяции волков и кроликов»

УП 09.03.04. 1. 1.25 О

Студент группы ПИ-02 И.П. Замятин

и.о., фамилия

Руководитель практики доцент, к.ф.-м.н. А.Ю. Андреева

должность, ученое звание и.о., фамилия

Барнаул 2022

ФГБОУ ВО «Алтайский государственный технический университет   
им. И. И. Ползунова»

Кафедра «Прикладная математика»

**Индивидуальное задание**

**На учебную (технологическую (проектно-технологическую) практику)**

(вид и тип практики по УП)

студенту Замятин Иван Павлович группы ПИ-02

(Ф.И.О.)

**График проведения практики**

|  |  |  |
| --- | --- | --- |
| **№**  **п/п** | **Наименование работ, выполняемых на практике** | **Сроки выполнения** |
| 1 | Разработка техническое задание |  |
| 2 | Проектирование объектной модели |  |
| 3 | Реализация модели в виде программы с графическим интерфейсом |  |
| 4 | Написание отчета и защита практики | 14.07.2022 |

Руководитель практики от университета АндрееваА.Ю., доцент каф. ПМ

(подпись) (Ф.И.О., должность)

Задание принял к исполнению \_\_\_\_\_\_\_

(подпись) (Ф.И.О.)

**Инструктаж по ОТ, ТБ, ПБ, ПВТР**

Инструктаж обучающегося по ознакомлению с требованиями охраны труда, техники безопасности, пожарной безопасности, а также правилами внутреннего трудового распорядка проведен «20» июня 2022 г.

Руководитель практики от

Университета Андреева А.Ю. доцент каф. ПМ

(подпись) (Ф.И.О., должность)

**Аннотация**

Отчет об учебной практике содержит описание программы, реализующей моделирование популяции волков и кроликов России: техническое задание, структуру данных, структуру файлов, описание программного продукта. Код программы на языке Visual C# .NET размещен в репозитории на Github и в приложении Б. В приложении А приведены снимки экранных форм программы.

Отчет содержит 36 страниц, 6 рисунков, 3 источника литературы.

**Оглавление**

[Введение 5](#_Toc112878951)

[1. Техническое задание 6](#_Toc112878952)

[1.1. Терминология 6](#_Toc112878953)

[1.2. Описание процесса функционирования модели 7](#_Toc112878954)

[1.3. Требования к функциональности программы 8](#_Toc112878955)

[2. Проект программного продукта 9](#_Toc112878956)

[2.1. Математическая модель 9](#_Toc112878957)

[2.2. Диаграмма классов 11](#_Toc112878958)

[2.3. Жизненный цикл объектов модели 12](#_Toc112878959)

[3.1. Выбор средств реализации 14](#_Toc112878960)

[3.2. Описание классов 15](#_Toc112878961)

[Заключение 18](#_Toc112878962)

[Список использованных источников 19](#_Toc112878963)

[Приложение A. Снимки экранных форм пользовательского интерфейса 20](#_Toc112878964)

[Приложение Б. Исходный код 22](#_Toc112878965)

Введение

Моделирование – основной метод исследований объектов, процессов или явлений, с целью получения объяснений этих явлений, а также для предсказания развития явлений и процессов, интересующих исследователей. Задача моделирования - выявить главные, характерные черты явления или процесса, его особенности, его поведение. Моделирование применяется в разных областях человеческой жизни: в медицине, демографии, страховании, социологии, в научно-исследовательской деятельности, в повседневной жизни и даже в компьютерных играх.

Модель (объекта – оригинала) – вспомогательный объект, отражающий наиболее существенные для исследования закономерности, суть, свойства, особенности строения и функционирования объекта-оригинала. Задачи моделирования:

* Понять сущность изучаемого объекта
* Научиться управлять объектом и определять наилучшие способы управления
* Решать прикладные задачи

Также моделирование позволяет прогнозировать прямые или косвенные последствия того или иного процесса или явления. Это одна из самых главных задач моделирования в случае с распространением вирусов.

1. Техническое задание
   1. Терминология

**Поле**– это модель мира, которая включает в себя решетку ячеек, размером 20\*20 отображающих животных разных групп: волк, кролик, волчонок, крольчонок.

**Кролик-** объектжертва в возрасте от 1 до 7

**Волчонок-** объект хищник в возрасте от 1 до 20

**Крольчонок-** объект жертва в возрасте от 0 до 1

**Инициатива-** режим, при котором у хищников начинается активная фаза, например поиск партнёра для размножение или охота на кролика

**Сытость-** показатель уровня сытости у хищников(при низком уровне сытости включается инициатива)

* 1. Описание процесса функционирования модели

Карта размером 20х20 заселен дикими кроликами, волками и волчицами. Имеется по нескольку представителей каждого вида. Кролики довольно глупы: в каждый момент времени они с одинаковой вероятностью 1/9 передвигаются в один из восьми соседних квадратов (за исключением участков, ограниченных береговой линией) или просто сидят неподвижно. Каждый кролик с вероятностью 0,2 превращается в двух кроликов. Каждая волчица передвигается случайным образом, пока в одном из соседних восьми квадратов не окажется кролик, за которым она охотится. Если волчица и кролик оказываются в одном квадрате, волчица съедает кролика и получает одно очко. В противном случае она теряет 0,1 очка. Волки и волчицы с нулевым количеством очков умирают.

В начальный момент времени все волки и волчицы имеют 1 очко. Волк ведет себя подобно волчице до тех пор, пока в соседних квадратах не исчезнут все кролики; тогда, если волчица находится в одном из восьми близлежащих квадратов, волк гонится за ней. Если волк и волчица окажутся в одном квадрате и там нет кролика, которого нужно съесть, они производят потомство случайного пола.

Запрограммировать предполагаемую экологическую модель и понаблюдать за изменением популяции в течение некоторого периода времени.

* 1. Требования к функциональности программы

В программе имеется графический интерфейс: одна главная форма, на которой расположено поле для отображения карты, а также кнопки паузы и кнопка старта и рестарта, все животные обозначены соответствующими картинками.

На поле отображаются объекты пять типов, при этом по мере выполнения программы они осуществляют взаимодействие друг с другом. Все они при запуске располагаются в пределах карты случайным образом.

Предусмотрены следующие типы объектов:

* Хищники (волк, волчица);
* Травоядные (кролик);
* Волчонок
* Крольчонок

Движение объектов на карте осуществляется по таймеру и начинается после нажатия кнопки «START/STOP».

Для повторного запуска программы имеется кнопка «RESTART»

Логический конец симуляции наступает в тот момент, когда на карте не остается или травоядных животных, или хищников. При этом фактически симуляция не останавливается, что позволяет хищникам бесконечно долго перемещаться по карте до тех пор, пока программа не будет закрыта.

Хищники могут охотиться на травоядных животных. Поведение хищников задается их начальными характеристиками, показатели которых меняются с течение времени (Хищник начинает охоту по достижение зрелого возраста и при низком уровне сытости <6).

Программа не является интерактивной, то есть пользователь никак не может влиять на исход симуляции, а может выступать только в роли наблюдателя.

1. Проект программного продукта
   1. Математическая модель

Поле модели – матрица из *W* строк и *H* столбцов. В каждой точки матрицы может быть несколько сущностей.

**Передвижение**

Хищник и жертва перемещается последовательно – Восемь клеток за ход (*dW*≤8 *И dH*≤8).

С течение времени у хищников теряется сытость(Hunger);

**Сущности**

**Волк**

Характеристики Волка:

* Hunger – сытость. Если Hunger≤6, то волк ищет кролика.   
  0≤Hunger ≤ *10*
* *age* – возраст волка. *age* ≥0

Если Hunger==0|| *age==20* то волк погибает

**Волчица**

Характеристики травы:

* *food*– количествоеды.0 ≤*food*
* Hunger – сытость. Если Hunger≤6, то волк ищет кролика.   
  0≤Hunger ≤ *10*
* *age* – возраст волка. *age* ≥0
* Pregnancy – является маркером этапа беременности волчицы

0≤*Pregnancy*≤3

**Кролик**

Характеристики кролика:

* *age* – возраст кролика. *age* ≥0

*если age>7, то с вероятность 0,2 создается новый кролик*

**Взаимодействие объектов**

**Волк-Волчица**

Если возраст волка *wolf.age>=1* а его инициатива принимает значение «истина» *initiative = true ,* то ищется волчица в клетках доступных для перемещение (*dX*, *dY*), если расстояние *dX* ≤ 8 и *dY* ≤ 8 и в этих приделах есть не беременная волчица  *pragnacy==0*, то волк перемещается к ней. После чего волчица беременеет *pragnacy== 3,* а волк теряет инициативу *initiative = false.* Аналогично происходит и с волчицей

**Волк-кролик**

Если возраст волка *wolf.age>=1 и его сытость* hunger<6, то его инициатива принимает значение «истина» *initiative = true* и начинается поиск кроликов, кролики ищется в клетках доступных для перемещения(*dX*, *dY*), если расстояние *dX* ≤ 8 и *dY* ≤ 8, то волк наподдает на кролика, после чего увеличивается сытость волка *hunger+=10 ,* а его инициатива становиться «Ложной» *initiative = false.* Аналогично происходит и с волчицей

* 1. Диаграмма классов

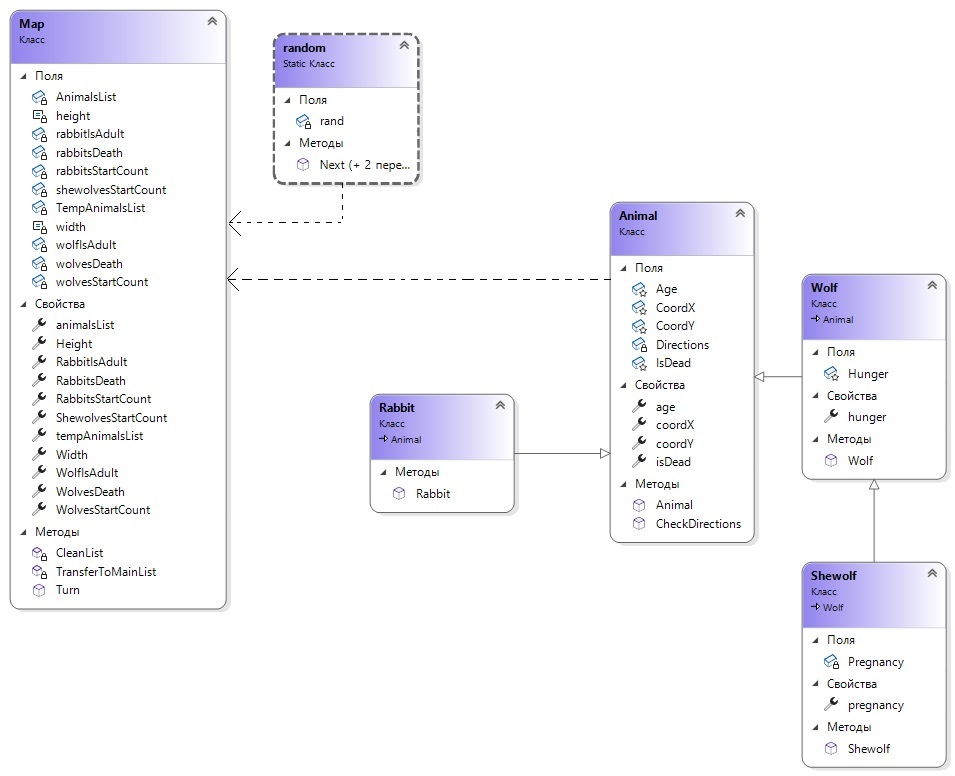
****

Рисунок 2.1 Диаграмма классов программы

* 1. Жизненный цикл объектов модели

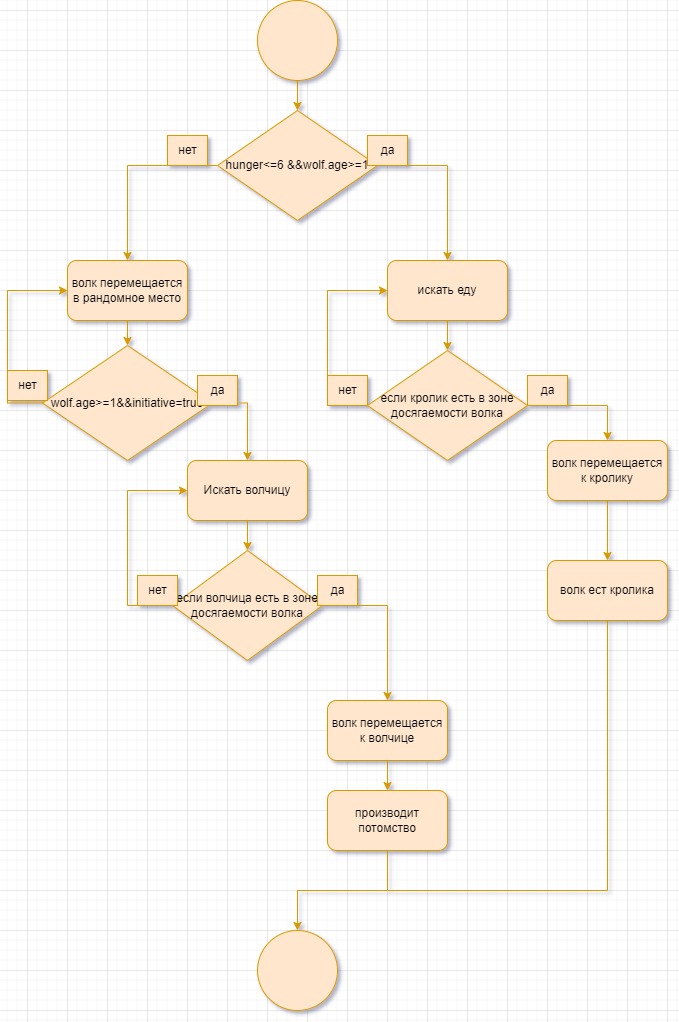


Рисунок 2.2 Диаграмма деятельности для объектов-хищников

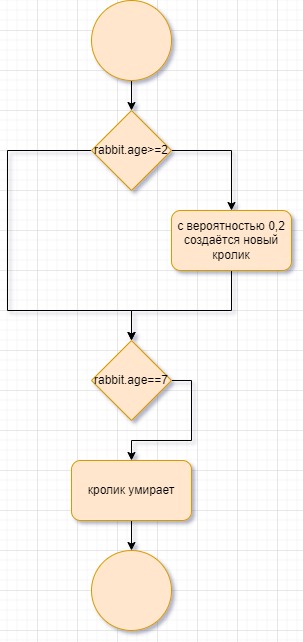


Рисунок 2.3 Диаграмма деятельности для объектов-травоядных

1. **Описание программного продукта**
   1. Выбор средств реализации

Данная модель была реализована на языке С# с использованием среды разработки программного обеспечения Microsoft Visual Studio 2019.

Язык С# - компилируемый статически типизированный язык программирования общего назначения.

Программа соответствует всем парадигмам объектно-ориентированного программирования.

* 1. Описание классов

**Классы:**

**Animal –** Родительский класс, в котором реализованыобщие данные всех животных.

Поля:

* CoordX- Координата местоположение животного по оси X
* CoordY- Координата местоположение животного по оси Y
* Age-Возраст
* IsDead- информация о состоянии животного
* Directions-направление хода животного

Методы:

* public Animal(int x, int y) – инициализация данных о животных
* public List<int[]> CheckDirections()-определение возможных направлений хода животного

Методы доступа:

* public int coordX – получить значение местоположение животного по оси X
* public int coordY- получить значение местоположение животного по оси Y
* public int age – получить значение возраста животного
* public bool isDead – получить значение о состоянии животного

**Map –** Класс, в котором реализованы основные функции животных.

Поля:

* height- высота поля
* width - ширина поля
* rabbitsStartCount- стартовое количество кроликов
* wolvesStartCount- стартовое количество волков
* shewolvesStartCount-стартовое количество волчиц
* rabbitIsAdult- возраст, при котором кролик считается взрослым
* wolfIsAdult- возраст, при котором волк/волчица считается взросл(ым/ ой)
* rabbitsDeath- возраст, при котом кролик умирает
* wolvesDeath- возраст, при котом волк/волчица умирает
* AnimalsList- список всех животных
* TempAnimalsList- дополнительный список животных

Методы:

* public void Turn()-основная функция хода всех животных
* void CleanList()-исключение мёртвых животных из списка
* void TransferToMainList()-перенос волчат из временного списка в основной

Методы доступа:

* public List<Animal> animalsList- получить список всех животных
* public List<Animal>tempAnimalsList- получить список временных животных
* public int RabbitsStartCount- получение значение стартового количества кроликов
* public int WolvesStartCount- получение значение стартового количества волков
* public int ShewolvesStartCount- получение значение стартового количества волчиц
* public int RabbitIsAdult – получение значение возраста, при котором кролик считается взрослым
* public int WolfIsAdult- получение значение возраста, при котором волк/волчица считается взрослым
* public int RabbitsDeath- получение значение возраста, при котором кролик умрет
* public int WolvesDeath- получение значение возраста, при котором волк/волчица умрет
* public static int Height- получение значения высоты
* public static int Width- получение значение ширины

**Rabbit –**  Класс кролика.

Поля:

* CoordX- Координата местоположение животного по оси X
* CoordY- Координата местоположение животного по оси Y

Методы:

* public Rabbit(int x, int y):base(x, y)-функция инициализация данных кролика

**Rand –** Класс для случайного перемещения животных (травоядных) на карте и пол хищников .

Поля:

* rand – получение случайного значение

Методы:

* public static int Next(int x)- функция генерации случайного пола у животных
* public static int Next(int x, int y)- функция генерации случайных координат у животных

**SheWolf–** Класс Волчиха.

Поля:

* Pregnancy- волчица беременна на этот ход

Методы:

* public Shewolf(int x, int y):base(x, y)-функция инициализация данных волчицы

**Wolf–** Класс волка.

Поля:

* Hunger-сытость

Методы:

* public Wolf(int x, int y):base(x, y)- функция инициализация данных волка

Методы доступа:

* public int hunger-получение значение сытости

Заключение

Была разработана программа, которая обладает следующим функционалом:

* Визуализация взаимодействия хищников и травоядных.

Возможны дальнейшие усовершенствования программы:

* Добавление параметра скорости и хищникам, и жертвам.
* Улучшение функционала кроликов.
* Усовершенствование экосистемы поля (добавление травы для кроликов).

Список использованных источников

1. Пахомов Б. И. П12 C# для начинающих. — СПб.: БХВ-Петербург, 2014. —432 с.: ил.
2. Сайт CyberForum [Электронный ресурс]. Режим доступа: https://www.cyberforum.ru/, свободный.
3. Сайт Habr [Электронный ресурс]. Режим доступа: https://habr.com/ru/post/336330/

Приложение A. Снимки экранных форм   
пользовательского интерфейса

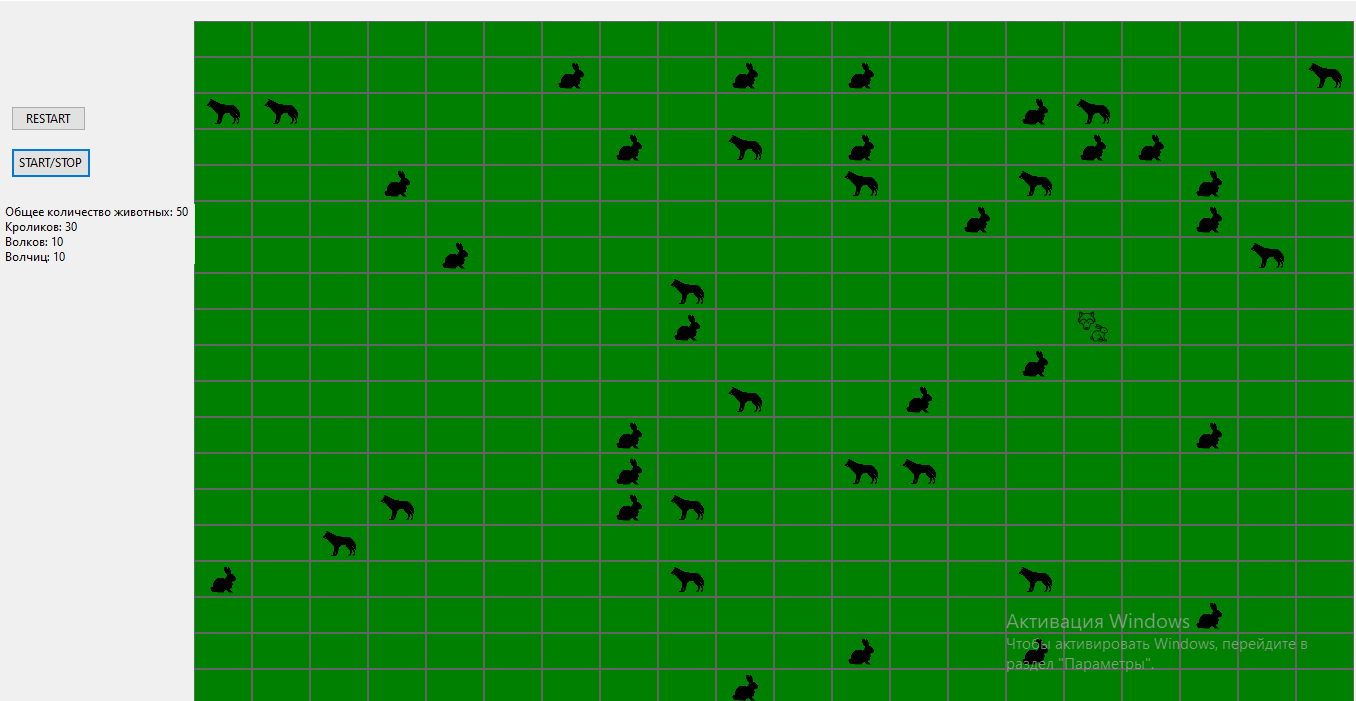


Рисунок A.

квадрат размером 20х20 заселен дикими кроликами, волками и волчицами.

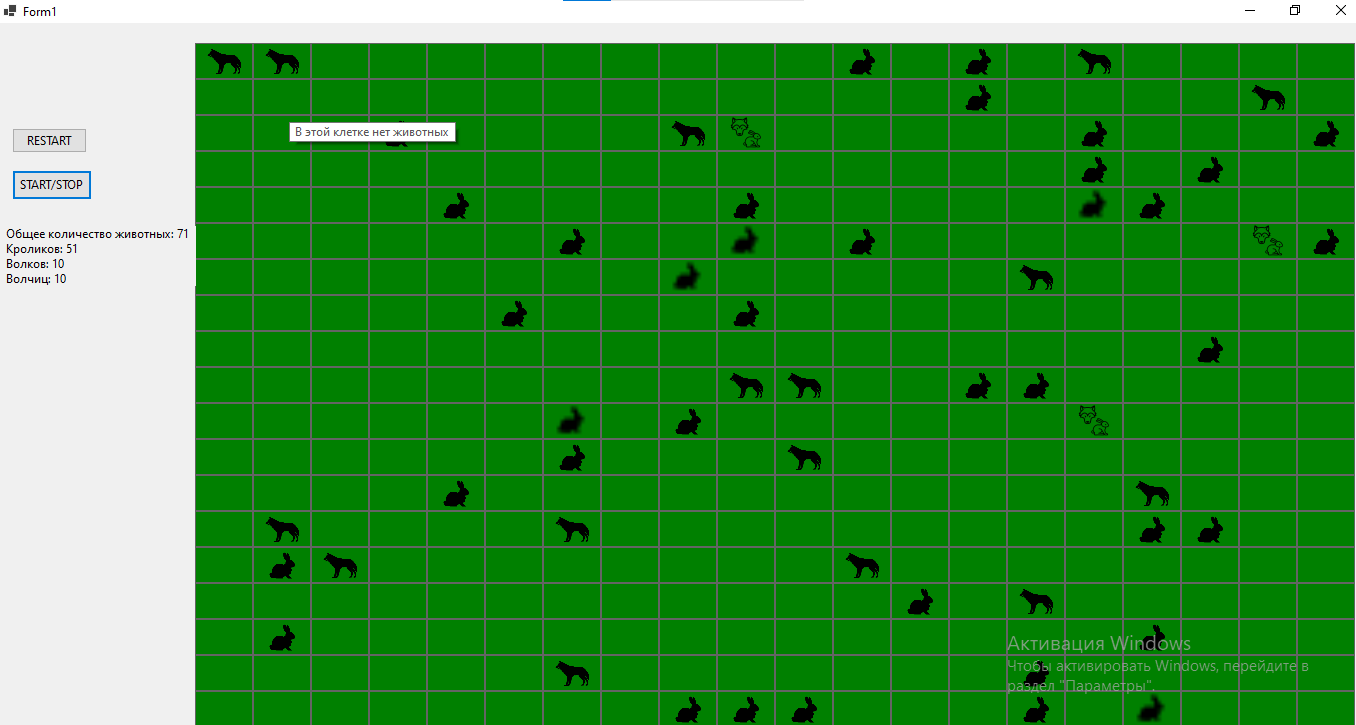


Рисунок A. 2

Кролики в каждый момент времени они передвигаются в один из восьми соседних квадратов (за исключением участков, ограниченных береговой линией) или просто сидят неподвижно.

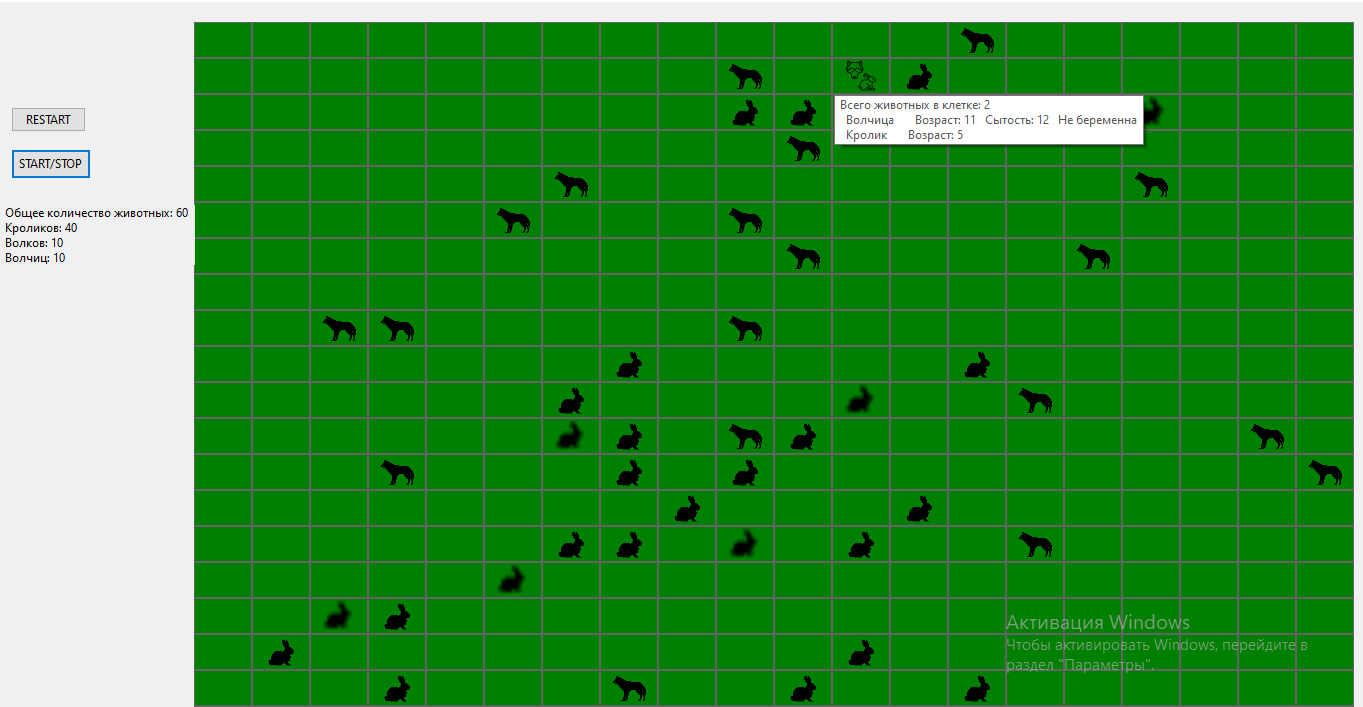


Рисунок A. 3

Каждый хищник передвигается случайным образом, пока в одном из соседних восьми квадратов не окажется кролик, за которым начинается охотиться. Когда хищник и жертва оказываются в одной клетке, волк ест кролика

Приложение Б. Исходный код

**Класс Map.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Summer\_Practice

{

class Map

{

const int height = 20; //ширина поля

const int width = 20; //высота поля

int rabbitsStartCount = 30;

int wolvesStartCount = 10;

int shewolvesStartCount = 10;

int rabbitIsAdult = 1;

int wolfIsAdult = 2;

int rabbitsDeath = 7;

int wolvesDeath = 20;

List<Animal> AnimalsList = new List<Animal>(); //список всех животных

List<Animal> TempAnimalsList = new List<Animal>(); //дополнительный список животных

public void Turn() //основная функция хода всех животных

{

//

//ходы кроликов

//

foreach (Animal animal in animalsList) //цикл по общему списку животных

{

if (animal.GetType() == typeof(Rabbit)) //если данное животное - это кролик

{

Rabbit rabbit = animal as Rabbit; //приведение объекта к типу "Кролик" для дальнейшей работы с ним

rabbit.age++; //взросление кролика

List<int[]> Directions = rabbit.CheckDirections(); //определение возможных направлений хода

int rand = random.Next(Directions.Count); //выбор случайной пары координат

rabbit.coordX = Directions[rand][0]; //перемещение кролика

rabbit.coordY = Directions[rand][1]; //

if (random.Next(5) == 1 && rabbit.age > RabbitIsAdult) //если кролик взрослый, то с вероятностью 0,2 создаётся новый кролик

tempAnimalsList.Add(new Rabbit(rabbit.coordX, rabbit.coordY));

if (rabbit.age == RabbitsDeath) //если кролик старый

rabbit.isDead = true; //смерть кролика

}

}

CleanList(); //исключение мёртвых животных из списка

TransferToMainList(); //перенос крольчат из временного списка в основной

//

//ходы волков и волчиц

//

foreach (Animal animal in animalsList) //цикл по общему списку животных

{

if (animal.GetType() == typeof(Wolf) || animal.GetType() == typeof(Shewolf)) //если данное животное - это волк или волчица

{

Wolf wolf = animal as Wolf; //приведение объекта к типу "Волк" для дальнейшей работы с ним

wolf.age++; //взросление волка

bool initiative = true; //инициатива

List<int[]> Directions = wolf.CheckDirections(); //вычисление возможных направлений движения

if (animal.GetType() == typeof(Shewolf)) //если данное животное - это волчица

{

Shewolf shewolf = animal as Shewolf;

if (shewolf.pregnancy == 1) //если беременность подходит к концу

{

if (random.Next(2) == 0) //выбор пола волчонка

tempAnimalsList.Add(new Wolf(shewolf.coordX, shewolf.coordY)); //рождение

else

tempAnimalsList.Add(new Shewolf(shewolf.coordX, shewolf.coordY)); //рождение

}

if (shewolf.pregnancy > 0)

shewolf.pregnancy--; //увеличение срока

}

if (wolf.hunger <= 6 && wolf.age >= WolfIsAdult) //если волк взрослый и голоден, начинается поиск кроликов

{

for (int i = 0; i < Directions.Count; i++) //кролики ищутся в клетках, доступных для перемещения

{

foreach (Animal an in animalsList)

{

if (an.GetType() == typeof(Rabbit))

{

Rabbit rabbit = an as Rabbit;

if (rabbit.coordX == Directions[i][0] && rabbit.coordY == Directions[i][1]

&& !rabbit.isDead) //если кролик в нужной клетке и ещё не съеден

{

wolf.coordX = Directions[i][0]; // перемещение волка к кролику

wolf.coordY = Directions[i][1]; //

rabbit.isDead = true; //волк кушает кролика

wolf.hunger += 10; //повышение сытости

initiative = false; //инициатива отключается

}

if (!initiative)

break;

}

}

if (!initiative)

break;

}

}

if (!initiative && wolf.age >= WolfIsAdult) //поиск партнёра для размножения

{

if (animal.GetType() == typeof(Wolf)) //если животное - волк, то ищется волчица

{

foreach (Animal an in animalsList)

{

if (an.GetType() == typeof(Shewolf))

{

Shewolf s = an as Shewolf;

for (int i = 0; i < Directions.Count; i++) //поиск в клетках, доступных для перемещения

{

if (s.coordX == Directions[i][0] && s.coordY == Directions[i][1]

&& s.age >= 2 && s.pregnancy == 0 && !s.isDead) //если координаты и остальные условия совпадают

{

wolf.coordX = Directions[i][0]; // перемещение к волчице

wolf.coordY = Directions[i][1]; //

s.pregnancy = 3; //волчица забеременела

initiative = false; //инициатива отключается

}

if (!initiative)

break;

}

if (!initiative)

break;

}

}

}

else //если животное - волчица, то ищется волк

{

Shewolf shewolf = animal as Shewolf;

if (shewolf.pregnancy == 0) //если волчица сейчас не беременна

{

foreach (Animal an in animalsList)

{

if (an.GetType() == typeof(Wolf))

{

Wolf w = an as Wolf;

for (int i = 0; i < Directions.Count; i++) //поиск в клетках, доступных для перемещения

{

if (w.coordX == Directions[i][0] && w.coordY == Directions[i][1]

&& w.age >= 2 && !w.isDead) //если координаты и остальные условия совпадают

{

wolf.coordX = Directions[i][0]; // перемещение к волку

wolf.coordY = Directions[i][1]; //

shewolf.pregnancy = 3; //волчица забеременела

initiative = false; //инициатива отключается

}

if (!initiative)

break;

}

if (!initiative)

break;

}

}

}

}

}

wolf.hunger -= 1; //уменьшение сытости за ход

if (wolf.hunger == 0 || wolf.age == WolvesDeath) //смерть волка от голода или от старости

{

wolf.isDead = true;

continue;

}

if (initiative) //просто ход в случайном направлении (при отсутствии других вариантов)

{

int rand = random.Next(Directions.Count);

wolf.coordX = Directions[rand][0];

wolf.coordY = Directions[rand][1];

}

}

}

CleanList(); //исключение мёртвых животных из списка

TransferToMainList(); //перенос волчат из временного списка в основной

}

void CleanList() //исключение мёртвых животных из списка

{

for (int i = 0; i < animalsList.Count; i++)

{

if (animalsList[i].isDead)

{

animalsList.RemoveAt(i);

i--;

}

}

}

void TransferToMainList() //перенос животных из временного списка в основной

{

foreach (Animal animal in tempAnimalsList)

{

animalsList.Add(animal);

}

tempAnimalsList.Clear();

}

public List<Animal> animalsList

{

get { return AnimalsList; }

set { AnimalsList = value; }

}

public List<Animal> tempAnimalsList

{

get { return TempAnimalsList; }

set { TempAnimalsList = value; }

}

public int RabbitsStartCount

{

get { return rabbitsStartCount; }

set { rabbitsStartCount = value; }

}

public int WolvesStartCount

{

get { return wolvesStartCount; }

set { wolvesStartCount = value; }

}

public int ShewolvesStartCount

{

get { return shewolvesStartCount; }

set { shewolvesStartCount = value; }

}

public int RabbitIsAdult

{

get { return rabbitIsAdult; }

set { rabbitIsAdult = value; }

}

public int WolfIsAdult

{

get { return wolfIsAdult; }

set { wolfIsAdult = value; }

}

public int RabbitsDeath

{

get { return rabbitsDeath; }

set { rabbitsDeath = value; }

}

public int WolvesDeath

{

get { return wolvesDeath; }

set { wolvesDeath = value; }

}

public static int Height

{

get { return height; }

}

public static int Width

{

get { return width; }

}

}

}

**Класс Animal.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Summer\_Practice

{

class Animal

{

protected int CoordX;

protected int CoordY;

protected int Age; // возраст

protected bool IsDead; //

List<int[]> Directions = new List<int[]>();

public Animal(int x, int y)

{

CoordX = x;

CoordY = y;

Age = 0;

IsDead = false;

}

public int coordX

{

get { return CoordX; }

set { CoordX = value; }

}

public int coordY

{

get { return CoordY; }

set { CoordY = value; }

}

public int age

{

get { return Age; }

set { Age = value; }

}

public bool isDead

{

get { return IsDead; }

set { IsDead = value; }

}

public List<int[]> CheckDirections() //определение возможных направлений хода животного

{

Directions.Clear();

int x = coordX;

int y = coordY;

int[] coords1 = { x, y }; //создание пары координат

Directions.Add(coords1); //в случае, если волк остаётся на месте

if (coordX - 1 >= 0 && coordY - 1 >= 0) //влево вверх

{

x = coordX - 1;

y = coordY - 1;

int[] coords = { x, y }; //создание пары координат

Directions.Add(coords);

}

if (coordY - 1 >= 0) //вверх

{

x = coordX;

y = coordY - 1;

int[] coords = { x, y }; //создание пары координат

Directions.Add(coords);

}

if (coordX + 1 < Map.Width && coordY - 1 >= 0) //вправо вверх

{

x = coordX + 1;

y = coordY - 1;

int[] coords = { x, y }; //создание пары координат

Directions.Add(coords);

}

if (coordX + 1 < Map.Width) //вправо

{

x = coordX + 1;

y = coordY;

int[] coords = { x, y };

Directions.Add(coords);

}

if (coordX + 1 < Map.Width && coordY + 1 < Map.Height) //вправо вниз

{

x = coordX + 1;

y = coordY + 1;

int[] coords = { x, y };

Directions.Add(coords);

}

if (coordY + 1 < Map.Height) //вниз

{

x = coordX;

y = coordY + 1;

int[] coords = { x, y };

Directions.Add(coords);

}

if (coordX - 1 >= 0 && coordY + 1 < Map.Height) //влево вниз

{

x = coordX - 1;

y = coordY + 1;

int[] coords = { x, y };

Directions.Add(coords);

}

if (coordX - 1 >= 0) //влево

{

x = coordX - 1;

y = coordY;

int[] coords = { x, y };

Directions.Add(coords);

}

return Directions; //возвращение массива координат

}

}

}

**Класс Form1.cs**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Summer\_Practice

{

public partial class Form1 : Form

{

Map MAP = new Map();

PictureBox[,] space = new PictureBox[Map.Width, Map.Height];

public Form1()

{

InitializeComponent();

for (int i = 0; i < Map.Width; i++)

{

for (int j = 0; j < Map.Height; j++)

{

space[i, j] = new PictureBox();

space[i, j].Size = new System.Drawing.Size(58, 36);

space[i, j].Location = new System.Drawing.Point(i \* 58 + 195, j \* 36 + 20);

space[i, j].SizeMode = System.Windows.Forms.PictureBoxSizeMode.Zoom;

space[i, j].BorderStyle = System.Windows.Forms.BorderStyle.FixedSingle;

space[i, j].Tag = i \* 20 + j;

space[i, j].BackColor = System.Drawing.SystemColors.ControlLightLight;

space[i, j].MouseEnter += new System.EventHandler(this.pictureBox\_MouseEnter);

space[i, j].MouseClick += new System.Windows.Forms.MouseEventHandler(this.pictureBox\_MouseClick);

space[i, j].BackColor = Color.Green;

this.Controls.Add(space[i, j]);

}

}

SpawnAnimals();

Refresh();

}

public void Refresh() //обновление игрового поля и вывод его на экран

{

int count = 0; //общее кол-во зверей в клетке

int[] type = { 0, 0, 0, 0 }; //типы животных: волк, волчонок, кролик, крольчонок

for (int i = 0; i < Map.Width; i++)

{

for (int j = 0; j < Map.Height; j++)

{

for (int k = 0; k < 4; k++) //обнуление временной переменной

type[k] = 0;

count = 0;

foreach (Animal animal in MAP.animalsList) //цикл по списку животных

{

if (animal.coordX == i && animal.coordY == j) //если координаты животного и клетки совпадают

{

if (animal.GetType() == typeof(Rabbit)) //если животное - это кролик

{

if (animal.age <= MAP.RabbitIsAdult) //если это крольчонок

{

type[3]++;

count++;

continue;

}

type[2]++; //если это кролик

count++;

}

else if (animal.GetType() == typeof(Wolf) || animal.GetType() == typeof(Shewolf)) //если животное - это волк или волчица

{

if (animal.age <= MAP.WolfIsAdult) //если это волчонок

{

type[1]++;

count++;

continue;

}

type[0]++; //если это волк

count++;

}

}

}

if (count > 0) //если в клетке есть животные, выбирается картинка для отображения в клетке

{

if ((type[0] > 0 || type[1] > 0) && (type[2] > 0 || type[3] > 0)) //если в клетке разные животные

{

space[i, j].Image = global::Summer\_Practice.Properties.Resources.wolf\_and\_rabbit;

continue;

}

else if (type[0] > 0) //если в клетке волки и возможно волчата

{

space[i, j].Image = global::Summer\_Practice.Properties.Resources.wolf;

continue;

}

else if (type[1] > 0) //если в клетке только волчата

{

space[i, j].Image = global::Summer\_Practice.Properties.Resources.lil\_wolf;

continue;

}

else if (type[2] > 0) //если в клетке кролики и возможно крольчата

{

space[i, j].Image = global::Summer\_Practice.Properties.Resources.rabbit;

continue;

}

else //если в клетке только крольчата

{

space[i, j].Image = global::Summer\_Practice.Properties.Resources.lil\_rabbit;

continue;

}

}

else

space[i, j].Image = null; //если в клетке нет животных, она отображается пустой

}

}

//далее идёт вывод общего количества животных на игровом поле и количества каждого вида животных отдельно

int rabbits\_count = 0, wolves\_count = 0, shewolves\_count = 0; //создание переменных для подсчёта животных

count = 0;

foreach (Animal animal in MAP.animalsList)

{

count++; //считается общее количество животных

if (animal.GetType() == typeof(Rabbit)) //количество кроликов

rabbits\_count++;

if (animal.GetType() == typeof(Wolf)) //количество волков

wolves\_count++;

if (animal.GetType() == typeof(Shewolf)) //количество волчиц

shewolves\_count++;

}

label1.Text = "Общее количество животных: " + count + " \nКроликов: " + rabbits\_count

+ " \nВолков: " + wolves\_count + " \nВолчиц: " + shewolves\_count; //вывод информации на экран

if (rabbits\_count==0|| wolves\_count==0|| shewolves\_count==0)

{

TurningTimer.Enabled = !TurningTimer.Enabled;

}

}

void SpawnAnimals() //генерация животных

{

for (int i = 0; i < MAP.RabbitsStartCount; i++)

{

Rabbit t = new Rabbit(random.Next(Map.Height), random.Next(Map.Width)); //создание объекта класса "Кролик" со случайными координатами

t.age = MAP.RabbitIsAdult + 1; //присвоение возраста

MAP.animalsList.Add(t); //добавление в общий список животных

}

for (int i = 0; i < MAP.WolvesStartCount; i++)

{

Wolf t = new Wolf(random.Next(Map.Height), random.Next(Map.Width)); //создание объекта класса "Волк" со случайными координатами

t.age = MAP.WolfIsAdult + 1; //присвоение возраста

MAP.animalsList.Add(t); //добавление в общий список животных

}

for (int i = 0; i < MAP.ShewolvesStartCount; i++)

{

Shewolf t = new Shewolf(random.Next(Map.Width), random.Next(Map.Height)); //создание объекта класса "Волчица" со случайными координатами

t.age = MAP.WolfIsAdult + 1; //присвоение возраста

MAP.animalsList.Add(t); //добавление в общий список животных

}

}

void Turn()

{

MAP.Turn();

Refresh();

}

private void TurnButton\_Click(object sender, EventArgs e)

{

if (TurningTimer.Enabled)

{

TurningTimer.Enabled = false;

}

else

{

Turn();

}

}

private void StartStopButton\_Click(object sender, EventArgs e)

{

TurningTimer.Enabled = !TurningTimer.Enabled;

}

private void RestartButton\_Click(object sender, EventArgs e)

{

TurningTimer.Enabled = false;

MAP.animalsList.Clear();

SpawnAnimals();

Refresh();

}

private void TurningTimer\_Tick(object sender, EventArgs e)

{

Turn();

}

private void pictureBox\_MouseEnter(object sender, EventArgs e)

{

if (!TurningTimer.Enabled)

{

PictureBox picturebox = sender as PictureBox;

int x = (int)picturebox.Tag / Map.Height;

int y = (int)picturebox.Tag % Map.Height;

string line = "";

int count = 0;

foreach (Animal animal in MAP.animalsList)

{

if (animal.coordX == x && animal.coordY == y)

{

count++;

if (animal.GetType() == typeof(Rabbit))

{

Rabbit? rabbit = animal as Rabbit;

line += " Кролик Возраст: " + rabbit.age + "\n";

}

if (animal.GetType() == typeof(Wolf))

{

Wolf wolf = animal as Wolf;

line += " Волк Возраст: " + wolf.age + " Сытость: " + wolf.hunger + "\n";

}

if (animal.GetType() == typeof(Shewolf))

{

Shewolf shewolf = animal as Shewolf;

if (shewolf.pregnancy > 0)

line += " Волчица Возраст: " + shewolf.age + " Сытость: " + shewolf.hunger + " Беременность: " + shewolf.pregnancy + "\n";

else

line += " Волчица Возраст: " + shewolf.age + " Сытость: " + shewolf.hunger + " Не беременна" + "\n";

}

}

}

if (count == 0)

toolTip1.SetToolTip(space[x, y], "В этой клетке нет животных");

else

toolTip1.SetToolTip(space[x, y], "Всего животных в клетке: " + count + " \n" + line);

}

}

private void pictureBox\_MouseClick(object sender, MouseEventArgs e)

{

PictureBox picturebox = sender as PictureBox;

int x = (int)picturebox.Tag / Map.Height;

int y = (int)picturebox.Tag % Map.Height;

if (e.Button == MouseButtons.Left)

{

Rabbit t = new Rabbit(x, y);

t.age = MAP.RabbitIsAdult + 1;

MAP.animalsList.Add(t);

Refresh();

}

else if (e.Button == MouseButtons.Right)

{

if (random.Next(2) == 0) //выбор пола волчонка

{

Wolf t = new Wolf(x, y);

t.age = MAP.WolfIsAdult + 1;

MAP.animalsList.Add(t);

Refresh();

}

else

{

Shewolf t = new Shewolf(x, y);

t.age = MAP.WolfIsAdult + 1;

MAP.animalsList.Add(t);

Refresh();

}

}

}

private void Form1\_Load(object sender, EventArgs e)

{

}

private void button1\_Click(object sender, EventArgs e)

{

TurningTimer.Enabled = !TurningTimer.Enabled;

}

**Класс Program.cs**

namespace Summer\_Practice

{

internal static class Program

{

/// <summary>

/// The main entry point for the application.

/// </summary>

[STAThread]

static void Main()

{

ApplicationConfiguration.Initialize();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new Form1());

}

}

}

**Класс Rabbit.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Summer\_Practice

{

class Rabbit : Animal

{

public Rabbit(int x, int y) : base(x, y)

{

CoordX = x;

CoordY = y;

}

}

}

**Класс rand.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Summer\_Practice

{

static class random

{

static Random rand = new Random();

public static int Next()

{

return rand.Next();

}

public static int Next(int x)

{

return rand.Next(x);

}

public static int Next(int x, int y)

{

return rand.Next(x, y);

}

}

}

**Класс Shewolf.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Summer\_Practice

{

class Shewolf : Wolf

{

int Pregnancy; //волчица беременна на этот ход

public Shewolf(int x, int y) : base(x, y)

{

CoordX = x;

CoordY = y;

Pregnancy = 0;

Hunger = 10;

}

public int pregnancy

{

get { return Pregnancy; }

set { Pregnancy = value; }

}

}

}

**Класс Wolf.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Summer\_Practice

{

class Wolf : Animal

{

protected int Hunger; //сытость

public Wolf(int x, int y) : base(x, y)

{

CoordX = x;

CoordY = y;

Hunger = 10;

}

public int hunger

{

get { return Hunger; }

set { Hunger = value; }

}

}

}