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)

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_2023 г.

дата

Отчет об учебной (технологической) практике

«Моделирование популяции видов в различных условиях окружающей среды»

УП 09.03.04. 1.3 О

Студент группы ПИ-11 Е.А. Гречишкин

и.о., фамилия

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

должность, ученое звание и.о., фамилия

Барнаул 2023

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 Гречишкинку Егору Александровичу группы ПИ-11

(Ф.И.О.)

График проведения практики

|  |  |  |
| --- | --- | --- |
| **№**  **п/п** | **Наименование работ, выполняемых на практике** | **Сроки выполнения** |
| **1** | **Разработка технического задания** | **20.06.2023** |
| **2** | **Проектирование объектной модели** | **25.06.2023** |
| **3** | **Реализация модели в виде программы с графическим интерфейсом** | **07.07.2023** |
| **4** | **Написание отчета и защита практики** | **13.07.2023** |

Руководитель практики от университета Андреева А.Ю. доцент каф. ПМ

(подпись) (Ф. И. О., должность)

Задание принял к исполнению Гречишкин Е.А.

(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 «19» июня 2023 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 некоторого вида при различных условиях. Отчет включает в себя техническое задание, устанавливающее правила, по которым работают модели, сведения о проектировании: о применении математических понятий и методов, о необходимых классах и об алгоритмах, по которым действуют объекты внутри моделей - и описание полученного программного продукта. Код программы на языке C# размещен в репозитории на Github и в приложении Б. В приложении А приведены снимки экранных форм программы.

Оглавление

[Введение 5](#_Toc139331593)

[1. Техническое задание 6](#_Toc139331594)

[1.1. Терминология 6](#_Toc139331595)

[1.2. Описание процесса функционирования модели 6](#_Toc139331596)

[1.2.1. Элементарная симуляция 7](#_Toc139331597)

[1.2.1.1. Симуляция хищников 10](#_Toc139331598)

[1.2.1.2. Симуляция конкуренции 10](#_Toc139331599)

[1.3. Требования к функциональности программы 10](#_Toc139331600)

[2. Проект программного продукта 12](#_Toc139331601)

[2.1. Математическая модель 12](#_Toc139331602)

[2.2. Диаграмма классов 14](#_Toc139331603)

[2.3. Жизненный цикл объектов модели 15](#_Toc139331604)

[3. Описание программного продукта 16](#_Toc139331605)

[3.1. Выбор средств реализации 16](#_Toc139331606)

[3.2. Описание классов 16](#_Toc139331607)

[3.2.1. Список классов со всеми полями, свойствами и методами 16](#_Toc139331608)

[3.2.2. Список классов-форм 23](#_Toc139331609)

[3.2.3. Класс Program 24](#_Toc139331610)

[Заключение 25](#_Toc139331611)

[Список использованных источников 26](#_Toc139331612)

[Приложение А 27](#_Toc139331613)

[Приложение Б 28](#_Toc139331614)

# Введение

Моделирование — исследование объектов познания на их моделях; построение и изучение моделей реально существующих объектов, процессов или явлений с целью получения объяснений этих явлений, а также для предсказания явлений, интересующих исследователей.

Познавательные возможности модели обуславливаются тем, что модель отображает (воспроизводит, имитирует) какие-либо существенные черты объекта-оригинала. При этом исследователь абстрагируется от несущественных черт, которые модель может не повторять.

Модели обычно используются, когда невозможно или непрактично создавать экспериментальные условия, при которых учёные могут непосредственно измерять результаты. Хотя прямое измерение результатов в контролируемых условиях всегда будет более надёжным, чем смоделированные оценки результатов.

Помимо научных исследований модели имеют и другие применения. Одно из них – в целях обучениях. На школьных уроках можно встретить модели земного шара (глобус), солнечной системы, атома и других вещей.

Построение современных моделей практически не обходится без использования вычислительной техники. Основная цель данной работы – построение модели, имитирующей какие-либо реальные процессы, с использованием информационных технологий. В качестве темы были взяты процессы в живой природе: существование некоторого биологического вида в заданных условиях и его взаимодействие с другими видами.

1. **Техническое задание**
   1. **Терминология**

**Конкурент –** элементарное существо, участвующее в симуляции конкуренции, потребляющее еду и борющееся за нее с другим видом (см. пункт 1.2.1.2).

**Пища (еда) –** сущность, служащая источником энергии для других сущностей.

**Симуляционное поле** – таблица, каждая ячейка которой содержит одну сущность, участвующую в симуляции.

**Сущность (существо)** – объект, участвующий в симуляции, обладающий рядом свойств и набором возможных действий, называемых тактовыми.

**Такт** – единица измерения времени в рамках симуляций, за которую все сущности, участвующие в ней, совершают ровно одно из предусмотренных действий.

**Тактовое действие** – любое действие, которое может совершить сущность за один такт.

**Травоядное** – элементарное существо, участвующее в симуляции хищников, потребляющее еду (см. пункт 1.2.1.1).

**Хищник** – элементарное существо, участвующее в симуляции хищников, потребляющее травоядные сущности (см. пункт 1.2.1.1).

**Элементарное существо** – см. пункт 1.2.1.

**Энергия** – величина, за счет которой сущности поддерживают свою жизнедеятельность и могут совершать тактовые действия.

* 1. **Описание процесса функционирования модели**

Программа предполагает наличие нескольких разных симуляций. В каждой из симуляций предлагается масштабируемое прямоугольное поле (**симуляционное поле**), разбитое на квадратные ячейки (клетки). В каждой ячейке может быть расположено **существо** (**сущность**), участвующее в симуляции и обладающее рядом свойств. Это существо может передвигаться в соседние ячейки, потреблять пищу в окружающих клетках и совершать прочие действия, предусмотренные конкретной симуляцией. Одно действие, которое может совершить конкретная сущность, выполняется за единицу времени, в дальнейшем называемую **тактом** (**тактовое действие**). Пользователь может настраивать параметры симуляции – например, количество определенных существ или свойства самой среды (свойства клеток поля). Пользователь может просматривать изменения состояния поля по ходу симуляции (после того, как программа обработает очередной такт, то есть все существа сделают какое-то одно действие), вносить корректировки по своему усмотрению.

* + 1. *Элементарная симуляция*

В начале симуляции пользователь расставляет на поле несколько живых существ (**элементарных существ**), занимающих по одной ячейке. Элементарные существа обладают следующим набором тактовых действий:

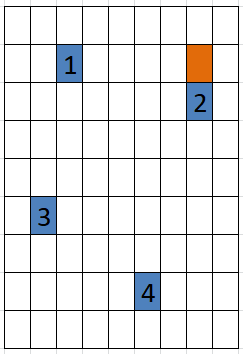
* умереть, если закончилась энергия или время жизни существа превысило максимально возможное число тактов;
* размножиться (потомок располагается в одной из соседних ячеек);
* съесть **пищу** в одной из соседних клеток;
* перейти на соседнюю клетку.

Соседней клеткой считается любая из восьми окружающих клеток (в том числе, расположенных по диагонали). Для совершения любого из вышеперечисленных действий соседняя клетка выбирается случайно. Например, если в двух соседних клетках располагается пища, то элементарное существо случайно выбирает одну из них для потребления.

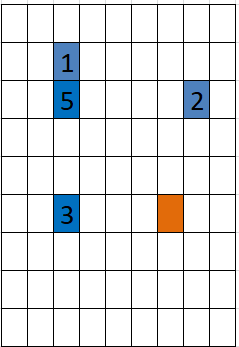
Возможное тактовое действие выбирается существом в том порядке, в котором они перечислены. То есть сначала проверяется, что существо все ещё живо, затем существо делает попытку размножиться, затем съесть пищу, затем перейти на соседнюю клетку. Если ни одно из действий не может быть совершено, то такт пропускается (существо бездействует). Это может произойти, если, например, все соседние клетки заняты другими элементарными сущностями. В этом случае невозможно размножиться (некуда размещать потомка), съесть пищу (пища отсутствует) или переместиться (в каждой клетке располагается только одна сущность).

Каждое живое существо обладает таким показателем как **энергия**. Энергия представляет собой целое неотрицательное число и измеряется в условных единицах (у.е.). Каждое из действий затрачивает определенное количество энергии. Кроме того, каждый такт, вне зависимости от выбранного действия, существо затрачивает энергию на поддержание жизнедеятельности. Например, у существа в начале симуляции было 100 у.е. энергии. За первый такт существо переместилось в соседнюю ячейку и потеряло 10 единиц энергии – пять на само перемещение и пять для продолжения своей жизни. За второй такт существо размножилось и потеряло в сумме 25 единиц энергии – 20 на размножение и 5 единиц теряет постоянно. За третий такт существо съедает пищу и получает 15 единиц энергии, но теряет из них 5. Итого после трех тактов существо имеет 75 у.е. энергии. При потере всей энергии существо погибает. Существо также гарантированно погибает после определенного количества тактов, проведенных на поле. После гибели существо убирается с поля.

Пища для существ появляется случайным образом на поле перед началом очередного такта и занимает одну клетку, так как представляет собой отдельную сущность. Пользователь настраивает скорость появления пищи и её энергетическую ценность (количество энергии, которое поглощает элементарное существо с потреблением одной единицы пищи).

Примерный вид симуляции можно посмотреть на рисунках 1 и 2. Рисунок 1 иллюстрирует начало симуляции с четырьмя существами (синие пронумерованное квадраты) и одной единицей пищи (оранжевый квадрат). Каждое из существ совершает одно из тактовых действий, список которых представлен выше. На рисунке 2 представлено возможное состояние поля перед вторым тактом. Видно, что существо 1 сделало потомка (квадрат под номером 5). Существо 2 съело пищу. Единица пищи восполнилась в случайном месте на поле. Существо 3 сдвинулось на клетку вправо. Существо 4 погибло, так как время жизни превысило максимально допустимое число тактов или закончилась энергия.

**Рисунок 1.** Начало элементарной симуляции

**Рисунок 2.** Элементарная симуляция перед вторым тактом

Пользователь настраивает следующие параметры симуляции:

* Максимальное время жизни существ – в тактах. Если время жизни существа превысит это значение, то существо умирает.
* Энергия для поддержания жизнедеятельности – в у.е. Отнимается за каждый такт жизни существа.
* Шанс размножения – в процентах с точностью до десятых долей (от 0.0% до 100.0%). Срабатывает, если у существа достаточно энергии для размножения. Существу должно хватать энергии как минимум на три такта после размножения (для поддержания жизнедеятельности и движений). Если у существа недостаточно энергии, то размножение не происходит (даже при шансе в 100%).
* Энергия, отдаваемая потомку при размножении – в у.е. При размножении эта энергия отнимается от текущей энергии существа и передается потомку. Потомок начинает свою жизнедеятельность с данным количеством энергии.
* Начальное количество энергии – в у.е. Существа, которые на поле размещены пользователем, будут иметь такой запас энергии.
* Энергия, необходимая для движения – в у.е. Отнимается от энергии существа, если оно перемещается в соседнюю клетку. Существо не сможет переместиться, если у него недостаточно энергии для этого.
* Скорость генерации пищи – в штук/такт. Каждый такт на поле появляется такое количество пищи, которое не превышает данное значение.
* Энергия пищи (энергетическая ценность) – в у.е. Эта энергия передается элементарному существу, которое поглотило одну единицу пищи.

**Элементарная симуляция служит основой для двух следующих симуляций** – **симуляции хищников и симуляции конкуренции**.

* + - 1. *Симуляция хищников*

Симуляция хищников работает по принципам элементарной симуляции. Ранее описанные элементарные существа в симуляции хищников называются **травоядными**. Помимо травоядных, в данной симуляции закономерно появляются существа, называемые **хищниками**. Их отличие заключается в том, что вместо потребления пищи они потребляют травоядных. То есть хищники ищут в соседних клетках травоядное существо, забирают энергию травоядного существа себе, а травоядное существо погибает. В остальном хищники не отличаются от травоядных существ.

Пользователь может отдельно настроить параметры для травоядных и для хищников (шанс размножения, максимальное время жизни и так далее).

* + - 1. *Симуляция конкуренции*

Симуляция конкуренции работает по принципам элементарной симуляции. В симуляции конкуренции пользователь может создать два разных вида элементарных существ, называемых **конкурентами**. Поведение обоих видов конкурентов ничем не отличается от поведения элементарных существ в элементарной симуляции. Конкуренция видов заключается в том, что они потребляют одну и ту же пищу.

Параметры обоих видов настраиваются отдельно. Например, можно задать один шанс размножения для первого вида и другой шанс для второго вида.

* 1. **Требования к функциональности программы**

Программа должна иметь графический интерфейс. При запуске программы пользователь видит приветственное окно, в котором ему предлагается выбрать одну из симуляций. После выбора симуляции и настроек размеров симуляционного поля должно отобразиться непосредственно окно с выбранной симуляцией. **Необходимо реализовать симуляцию конкуренции и симуляцию хищников**. Элементарная симуляция является составной частью этих симуляций, и её отдельная реализация не требуется.

Окно симуляции должно содержать панель управления симуляцией в левой части, а все остальное пространство отводится под графическое представление симуляции. На панели управления должны быть предусмотрены функции для начала симуляции, её непрерывного выполнения и приостановки, функции для настройки симуляции (настраиваемые параметры описаны в разделе 1.2), функции для добавления и удаления сущностей, предусмотренных симуляцией (добавления хищников, травоядных, конкурентов).

Графическое изображение симуляционного поля должно представлять собой таблицу с квадратными ячейками, в которых отображаются сущности. Примерный вид поля был показан на рисунках 1 и 2. Добавление и удаление сущностей лучше всего реализовать в интерактивном режиме – путем клика левой кнопки мыши по ячейкам поля. Необходимо, чтобы пользователь мог видеть результаты работы симуляции после каждого такта. Продолжительность одного такта при непрерывном выполнении должна составлять примерно 0,5 – 1 секунду реального времени.

Также на панели управления должна отображаться информация о текущем количестве сущностей каждого вида, которые располагаются на поле: сколько в данный момент есть хищников, травоядных, конкурентов одного типа, конкурентов второго типа.

По возможности, программу необходимо спроектировать таким образом, чтобы её функционал легко расширялся путем добавления новых симуляций, отличных от обозначенных.

1. **Проект программного продукта**
   1. **Математическая модель**

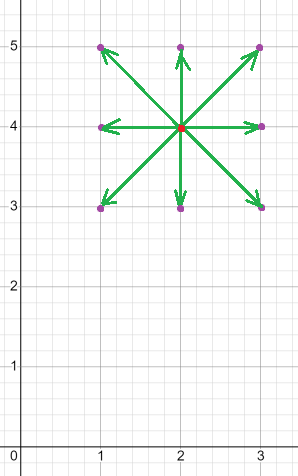
Представим симуляционное поле в виде фрагмента координатной плоскости. Сущности размещаются в точках с целыми координатами (*x; y*), где x ≥ 0, y ≥ 0, x < W, y < H, W – ширина симуляционного поля, H – высота симуляционного поля. Если какая-либо из обозначенных точек не занята сущностью, то будем говорить, что по этим координатам находится **пустая сущность**.

Сущность, добавляемая пользователем или полученная при размножении другой сущности, помещается в **тактовую очередь**. При обработке очередного такта из тактовой очереди последовательно извлекаются сущности, которые совершают тактовое действие. Если сущность после тактового действия остается живой, то она помещается обратно в очередь. Если сущность мертва, то она удаляется с поля, а по координатам размещается пустая сущность.

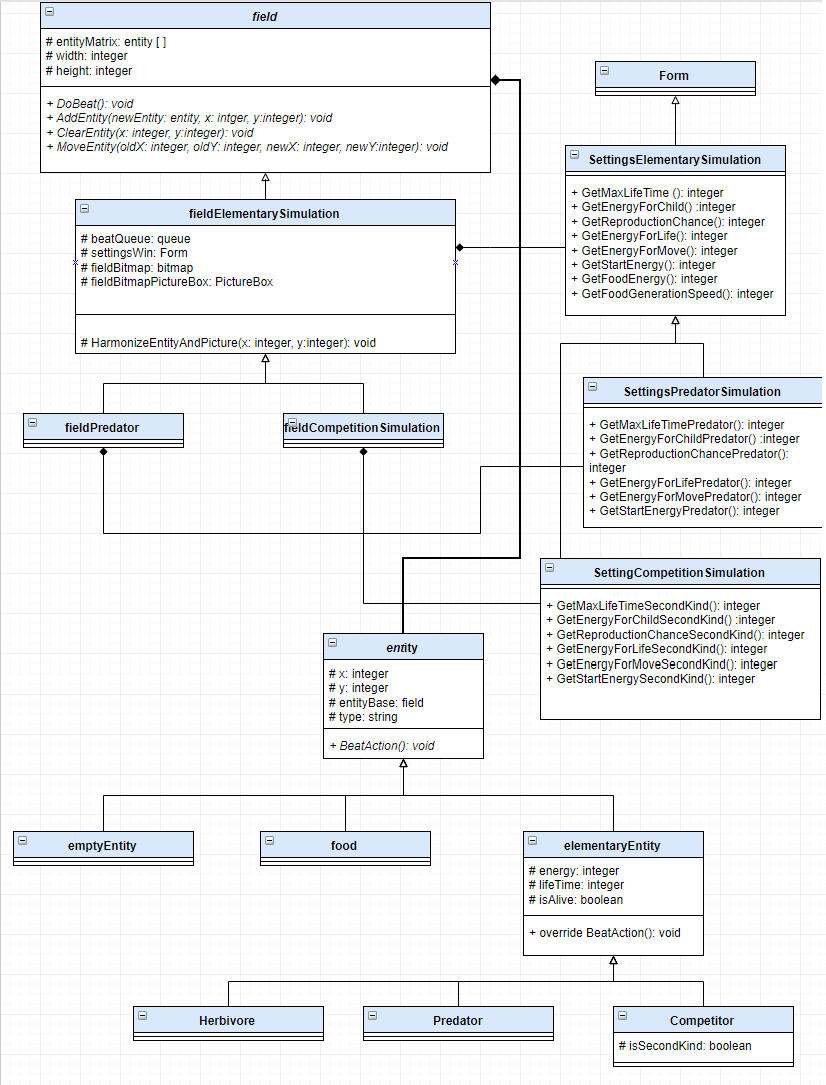
Сущность способна «видеть» другие сущности (определять их состояние, параметры), располагающиеся по координатам, отличным максимум на единицу от координат данной сущности. Например, сущность с координатами (2; 4) может «видеть» сущности по координатам (1; 5), (2; 3), (3; 4) и так далее.

При поиске пищи (или поиске травоядного для хищника), места для размещения потомка, места для перемещения генерируются два случайных целых числа *а* и *b* в диапазоне [-1; 1], являющие координатами вектора (*a*, *b*). После этого проверяется точка с координатами (*x + a*; *y + b*), где (*x; y*) – координаты сущности, совершающей тактовое действие. Если сущность по сгенерированным координатам удовлетворяет условиям тактового действия (например, является пищей и может быть поглощена сущностью, совершающей тактовое действие), то тактовое действие считается совершённым, иначе процесс повторяется, Всего может быть сделано до 16 попыток найти точку, удовлетворяющую условиям действия. Если точка не была найдена, то сущность переходит к следующему возможному действию или пропускает этот такт, если список действий исчерпан.

Наглядная иллюстрация поиска представлена на рисунке 3. Красной точкой обозначена сущность, совершающая в данный момент тактовое действие. Зеленым обозначены возможные векторы (*a; b*). Фиолетовые точки – это сущности, которые может «видеть» сущность в красной точке. Предположим, что сущность в красной точке – травоядное существо, ищущее еду. Тогда оно может поглотить сущность в одной из фиолетовых точек, если сущность является пищей. Если пища не была найдена, то травоядное существо переходит к поиску точки для перемещения, то есть ищет пустую сущность среди фиолетовых точек.

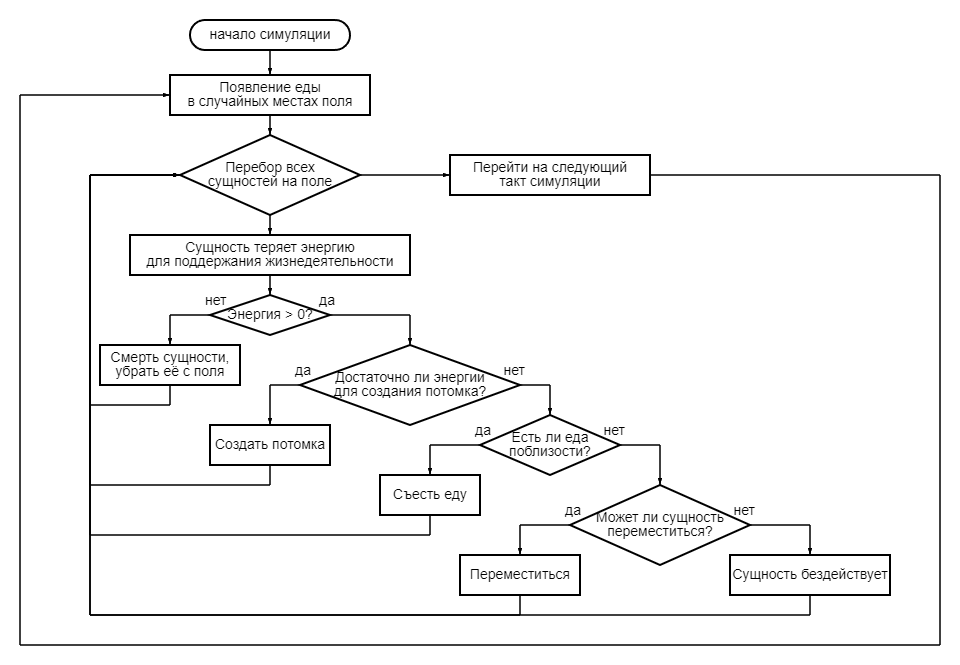
Для размещения пищи перед обработкой очередного такта применяется схожий принцип. Генерируются два числа, являющиеся координатами некоторой точки. Если в данной точке размещена пустая сущность, то она заменяется сущностью-пищей. Для каждой единицы пищи также делается до 16 попыток размещения. То есть при скорости генерации в 5 единиц процедура генерации в худшем случае может пройти до 80 раз.

**Рисунок 3.** Процесс поиска

* 1.  **Диаграмма классов**

**Рисунок 4.** Диаграмма классов

Основополагающими являются классы entity и field. Класс entity содержит всю необходимую информацию о сущности, а его потомки реализуют метод BeatAction(), определяющий поведение конкретной сущности. Класс field и его потомки содержат всю информацию о текущей симуляции и определяют способы отображения симуляционного поля. Для обработки одного такта необходимо вызвать метод DoBeat(), который вызывает метод BeatAction() для всех сущностей в очереди beatQueue. В beatQueue помещаются экземпляры классов Herbivore, Predator, Competitor, являющихся потомками класса elementaryEntity. Настройки симуляций хранятся в классах SettingsPredatorSimulation и SettingsCompetitionSimulation, которые являются потомками системного класса Form, то есть являются окнами.

* 1. **Жизненный цикл объектов модели**

**Рисунок 5**. Схема функционирования модели

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

Для реализации проекта была выбрана среда программирования Microsoft Visual Studio 2022, в составе которой имеется .NET Framework. .NET Framework, в свою очередь, содержит средство разработки Windows Form. Среди языков программирования, поддерживаемых в среде, был выбран С# ввиду доступной документации на сайте разработчика Microsoft и легкости нахождения информации по возникающим проблемам на интернет-форумах.

* 1. **Описание классов**

Некоторые из особенностей классов представлены в пункте 2.2. Там можно найти диаграмму классов и небольшое пояснение по их взаимодействию.

* + 1. **Список классов со всеми полями, свойствами и методами**

1. **internal abstract class entity**

Абстрактный класс, служащий основой любой сущности для любой симуляции. Наследники определяют характеристики конкретных сущностей и их поведение (в первую очередь, метод BeatAction(), отвечающий за тактовое действие).

Экземпляры классов-наследников хранятся в матрице EntityMatrix класса field. Метод DoBeat() класса field вызывает метод BeatAction() класса entity.

***Конструкторы класса:***

* *public entity()* – конструктор по умолчанию.

***Поля класса:***

* *protected int x* – координата х на симуляционном поле (индекс в матрице EntityMatrix класса field)
* *protected int y* – координата y на симуляционном поле (индекс в матрице EntityMatrix класса field)
* *field entityBase* – ссылка на объект класса field, в котором хранится данная сущность

***Свойства класса:***

* *field EntityBase {get; set}* – устанавливает и получает ссылку на объект класса field, в котором хранится данная сущность (связано с полем entityBase).
* *public string Type { get; protected set; }* – устанавливает и получает тип, к которому относится данная сущность (например, к типу хищников).

***Методы класса***

* *public abstract void BeatAction()* – задает поведение сущности (её тактовые действия). Наследники определяют конкретное поведение.

1. **internal class emptyEntity : entity**

Экземпляры этого класса сигнализируют об отсутствии сущности по заданным координатам (пустая сущность).

***Конструкторы класса:***

* *public emptyEntity()* – конструктор по умолчанию.

***Методы класса***

* *public override void BeatAction()* – переопределяет родительский абстрактный метод. Тело метода пустое, так как пустая сущность по определению не является объектом, участвующим в симуляции.

1. **internal class food : entity**

Экземпляры этого класса сигнализируют о наличии еды по заданным координатам.

***Конструкторы класса:***

* *public food()* – конструктор по умолчанию.

***Методы класса***

* *public override void BeatAction()* – переопределяет родительский абстрактный метод. Тело метода пустое, так как пища не совершает каких-либо действий.

1. **internal class elementaryEntity : entity**

Элементарные сущности из элементарной симуляции.

***Конструкторы класса:***

* *public elementaryEntity()*– конструктор по умолчанию.

***Поля класса:***

* *protected int energy* – количество энергии, которым сущность обладает в данный момент.
* *protected int lifeTime* – время жизни сущности с момента её появления на поле (в тактах).

***Свойства класса:***

* *public bool IsAlive { get; set; }* – определяет, живо ли существо.
* *public virtual int Energy { get; set; }* – связано с полем energy.
* *public virtual int LifeTime { get; set; }* – связано с полем lifeTime.

***Методы класса***

* *public override void BeatAction()*– задает поведение сущности (её тактовые действия).

1. **internal class Competitor : elementaryEntity**

Описывает сущностей-конкурентов.

***Конструкторы класса:***

* *public Competitor() –* конструктор по умолчанию.

***Свойства класса:***

* *public bool IsSecondKind { get; set; }* – определяет, принадлежит ли это существо ко второму биологическому виду или к первому.

***Методы класса***

* *public override void BeatAction()*– задает поведение сущности (её тактовые действия).

1. **internal class Predator : elementaryEntity**

Описывает сущностей-хищников

***Конструкторы класса:***

* *public Predator() –* конструктор по умолчанию.

***Методы класса***

* *public override void BeatAction()*– задает поведение сущности (её тактовые действия).

1. **internal class Herbivore : elementaryEntity**

Описывает травоядных сущностей

***Конструкторы класса:***

* *public Herbivore()–* конструктор по умолчанию.

***Методы класса***

* *public override void BeatAction()*– задает поведение сущности (её тактовые действия).

1. **internal abstract class field**

Класс и его наследники предназначены для хранения всей информации о симуляционном поле (о расположении каждой сущности – класс entity – и о характеристиках каждой ячейки поля) и отображения информации на экран.

***Конструкторы класса:***

* *public field (int width, int height )* – конструктор задает размеры симуляционного поля.

***Свойства класса:***

* *public entity[,] EntityMatrix { get; }* – матрица сущностей.
* *public int Width { get; }* – ширина матрицы сущностей.
* *public int Height { get; }* – высота матрицы сущностей.

***Методы класса***

* *public virtual void DoBeat()* – обрабатывает очередной такт симуляции, вызывая метод BeatAction() для сущностей из EntityMatrix.
* *public virtual void MoveEntity(int oldX, int oldY, int newX, int newY)* – перемещает сущность из координат (oldX, oldY) в координаты (newX, newY). При этом, по старым координатам создается пустая сущность emptyEntity.
* *public virtual void ClearEntity(int x, int y)* – удаляет сущность по координатам (x, y). При этом, по этим координатам создается пустая сущность.
* *public virtual void AddEntity(entity newEntity, int x, int y)* – добавляет сущность newEntity в матрицу EntityMatrix по индексам x, y. При этом, старая сущность удаляется.

1. **internal class fieldElementarySimulation : field**

Симуляционное поле для элементарной симуляции.

***Конструкторы класса:***

* *public fieldElementarySimulation(int width, int height)* – конструктор задает размеры симуляционного поля.

***Поля класса:***

* *protected bool enabledDrawing* – Определяет, работает ли отрисовка сущностей на экране.
* *protected Queue<elementaryEntity> beatQueue* – тактовая очередь.
* *protected bool isAdditionalMood* – при установке флага изображение поля реагирует на нажатия левой кнопки мыши и добавляет на поле объект класса elementaryEntity.
* *protected bool isDeletingMood* – при установке флага изображение поля реагирует на нажатия левой кнопки мыши и убирает сущность с поля.
* *protected Bitmap fieldBitmap* – изображение симуляционного поля.
* *protected PictureBox fieldBitmapPictureBox* – элемент управления, отображаемый в окне, с которым можно взаимодействовать при установке флагов isDeletingMood, isAdditionalMood.

***Свойства класса:***

* *public int ElemEntCount { get; }* – возвращает количество сущностей, находящихся в тактовой очереди beatQueue.
* *public SettingsElementarySimulation SettingsWin { get; }* – окно параметров симуляции
* *public PictureBox GetPicture { get; }* – возвращает элемент управления fieldBitmapPictureBox для добавления в окно.
* *public bool IsDeletingMood { get; set; }* – связано с полем isDeletingMood.
* *public bool IsAdditionalMood { get; set; }* – связано с полем isAdditionalMood.
* *public bool EnabledDrawing { get; set; }* – связано с полем enabledDrawing.

***Методы класса***

* *public override void DoBeat()* – переопределяет родительский метод.
* *public override void AddEntity(entity newEntity, int x, int y)* – переопределяет родительский метод.
* *public override void ClearEntity(int x, int y)* – переопределяет родительский метод.
* *public override void MoveEntity(int oldX, int oldY, int newX, int newY)* – переопределяет родительский метод.
* *protected virtual void HarmonizeEntityAndPicture(int x, int y)* – согласовывает изображение fieldBitmap с матрицей EntityMatrix. Перерисовывает сущность по координатам (x, y).
* *protected virtual void EntityClick(object sender, MouseEventArgs e)* – обработчик события нажатия левой кнопки мыши по элементу управления fieldBitmapPictureBox. Добавляет и удаляет сущности при установке флажков isDeletingMood, isAdditionalMood.

1. **internal class fieldPredator : fieldElementarySimulation**

Симуляционное поле симуляции хищников

***Конструкторы класса:***

* *public fieldPredator(int width, int height)* – конструктор задает размеры симуляционного поля.

***Свойства класса:***

* *new public SettingsPredatorSimulation SettingsWin { get; }* – заменяет родительское окно параметров симуляции новым.
* *public bool IsPredatorAdded { get; set; }* – при установке флага совместно с isAdditionalMood изображение поля реагирует на нажатия левой кнопки мыши и добавляет на поле объект класса Predator, иначе – Herbivore.
* *public int PredatorsCount { get; protected set; }* – счетчик хищников на поле.

***Методы класса***

* *public override void DoBeat()* – переопределяет родительский метод.
* *public override void AddEntity(entity newEntity, int x, int y)* – переопределяет родительский метод.
* *public override void ClearEntity(int x, int y)* – переопределяет родительский метод.
* *protected override void HarmonizeEntityAndPicture(int x, int y)* – переопределяет родительский метод.
* *protected override void EntityClick(object sender, MouseEventArgs e)*– переопределяет родительский метод.

1. **internal class fieldCompetitionSimulation : fieldElementarySimulation**

Симуляционное поле симуляции конкуренции

***Конструкторы класса:***

* *public fieldCompetitionSimulation(int width, int height)* – конструктор задает размеры симуляционного поля.

***Поля класса:***

* *protected int firstKindCount* – счетчик конкурентов первого биологического вида.
* *protected int secondKindCount* – счетчик конкурентов второго биологического вида

***Свойства класса:***

* *new public SettingsCompetitionSimulation SettingsWin { get; }* – заменяет родительское окно параметров симуляции новым.
* *public bool IsSecondKindAdded { get; set;}* – при установке флага совместно с isAdditionalMood изображение поля реагирует на нажатия левой кнопки мыши и добавляет на поле объект класса Competitor и устанавливает в нем флаг IsSecondKind.
* *public int FirstKindCount { get; }* – связано с полем firstKindCount.
* *public int SecondKindCount { get; } –* связано с полем secondKindCount*.*

***Методы класса***

* *public override void DoBeat()* – переопределяет родительский метод.
* *public override void AddEntity(entity newEntity, int x, int y)* – переопределяет родительский метод.
* *public override void ClearEntity(int x, int y)* – переопределяет родительский метод.
* *protected override void HarmonizeEntityAndPicture(int x, int y)* – переопределяет родительский метод.
* *protected override void EntityClick(object sender, MouseEventArgs e)*– переопределяет родительский метод.
  + 1. **Список классов-форм**

1. HelloForm

Приветственное окно, открывающееся при старте приложения, на котором происходит выбор нужной симуляции

1. WidthAndHeightForm

В окне задаются размеры симуляционного поля. Окно отображается при вызове конструкторов форм, отображающих симуляции: ElementarySimulationForm. PredatorSimulationForm, CompetitionSimulationForm. В случае успешного задания размеров происходит переход на соответствующее окно. В другом случае идет возврат на приветственное окно.

1. ElementarySimulationForm. PredatorSimulationForm, CompetitionSimulationForm.

Окна, отображающие симуляцию. Имеют поле simField, которое хранит потомка класса field.

1. SettingsElementarySimulation

Является неотъемлемой частью класса fieldElementarySimulation. Содержит следующие свойства:

* *public int MaxLifeTime { get; private set; }* – максимальное время жизни элементарной сущности в тактах.
* *public int EnergyForChild { get; private set; }* – энергия, которую элементарная сущность передает потомку.
* *public int ReproductionChance { get; private set; }* – шанс, с которым элементарная сущность размножится при условии, что есть свободная клетка поблизости и для размножения достаточно энергии.
* *public int EnergyForLife { get; private set; } –* энергия, которая отнимается за каждый такт существования у элементарной сущности (для поддержания жизнедеятельности).
* *public int EnergyForMove { get; private set; }* – энергия, необходимая для перемещения сущности в соседнюю клетку
* *public int StartEnergy { get; private set; }* – энергия, с которой пользователь создает элементарную сущность.
* *public int FoodEnergy { get; private set; }* – энергетическая ценность пищи.
* *public int FoodGenerationSpeed { get; private set; }* – количество единиц пищи, не больше которого может появиться на поле за один такт.

1. SettingsPredatorSimulation

Является неотъемлемой частью класса fieldPredator. Является наследником класса SettingsElementarySimulation. Свойства родительского класса относятся к травоядным. Сделаны аналогичные свойства для хищников с добавлением слова Predator в конец идентификаторов свойств.

1. SettingsCompetitionSimulation

Является неотъемлемой частью класса fieldCompetitionSimulation. Является наследником класса SettingsElementarySimulation. Свойства родительского класса относятся к первому биологическому виду конкурентов. Сделаны аналогичные свойства для второго вида с добавлением слов SecondKind в конец идентификаторов свойств.

* + 1. **Класс Program**

Является системным классом, с которого начинается выполнение программы.

Дополнен несколькими статическими полями и методами:

* *private static HelloForm helloForm* – содержит приветственное окно. Для запуска приложения используется системный метод Application.Run(), в который передается данная форма. При этом приложение завершает свою работу, когда для данной формы будет вызван метод Close().
* *public static string ChosenSimulation { get; set; }* – содержит строку с названием выбранной симуляции
* *public static void CloseApplication()* – закрывает приложение, вызывая метод Close() для приветственного окна.
* *public static void OpenSimulation()* – создает окно с выбранной симуляцией.
* *public static void OpenHelloForm()* – показывает приветственное окно.

# Заключение

С помощью созданной программы можно наглядно проиллюстрировать то, как условия окружающей среды влияют на численность особей некоторого биологического вида, а также как на популяцию влияет взаимодействие с другими видами. Разработанное ПО можно использовать в учебных целях, например, на школьных занятиях по биологии.

Программа имеет следующие достоинства:

* Наглядное изображение существ на поле, добавление и удаление существ в интерактивном режиме;
* Возможность настройки параметров симуляции;
* Возможность выбора симуляций, которые демонстрируют определенные ситуации.

Программа имеет следующие недостатки:

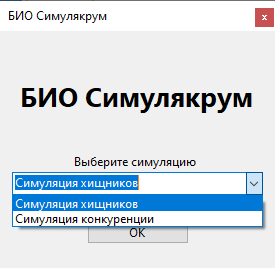
* Малое количество симуляций;
* Невозможность создания больше двух биологических видов в рамках одной симуляции;
* Невозможность настройки скорости симуляции;
* Отсутствие какой-либо инфографики по численности видов.

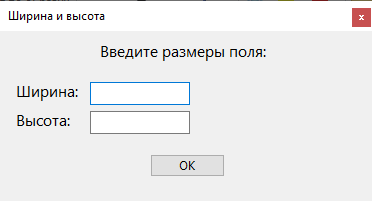
Дальнейшую модификацию программы можно производить исходя из выявленных недостатков. Для добавления новых симуляций можно использовать наследование от готовых классов entity и field. Также возможно добавление симуляций, основанных на функциональности вспомогательной элементарной симуляции, реализацию которой содержат классы fieldElementarySimulation и elementaryEntity. Скорость симуляции можно регулировать путем изменения интервала таймера, но сокращение интервала приведет к потере производительности из-за отрисовки сущностей. Можно настроить отключение отрисовки при небольших интервалах (в готовых симуляциях такая возможность в некоторой степени реализована, но скрыта от пользователя).

# Список использованных источников

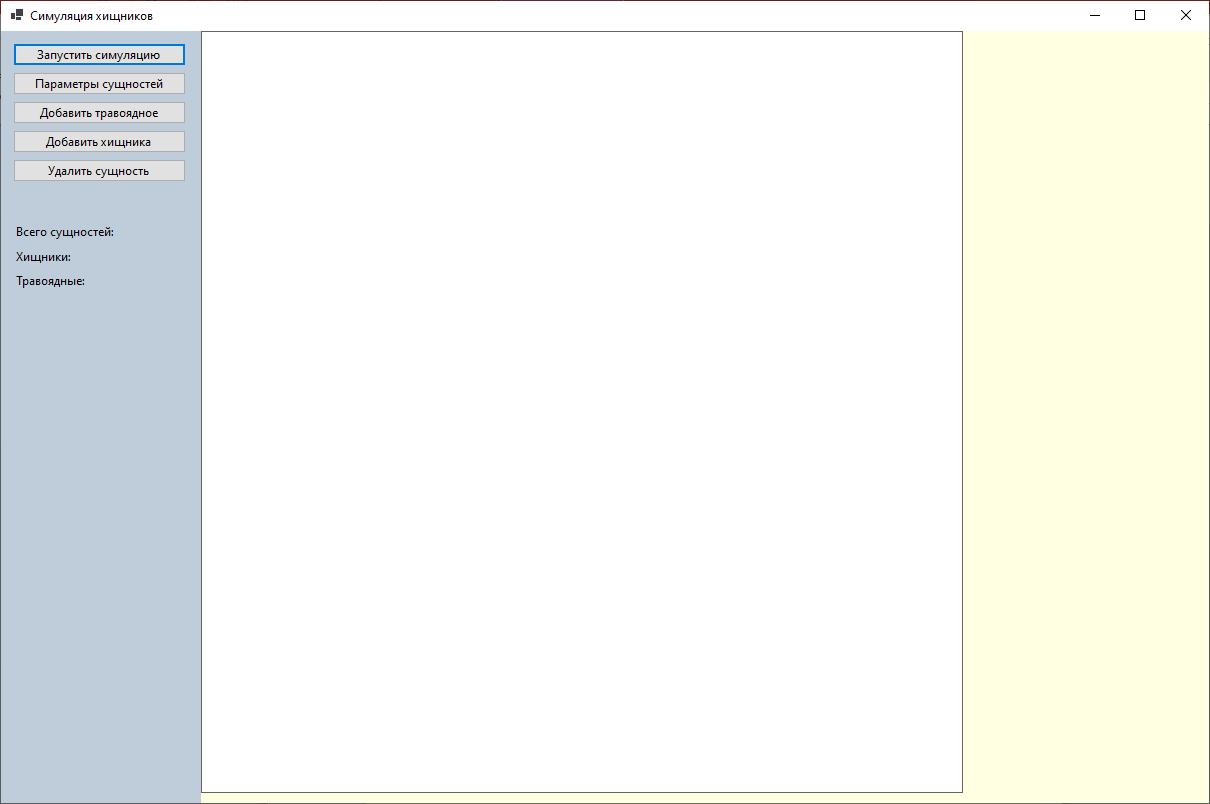
1. Документация по .NET / Microsoft. – [Б. м.], 2023. – URL: https://learn.microsoft.com/ru-ru/dotnet/ (дата обращения: 04.07.2023).
2. Полное руководство по языку программирования С# 11 и платформе .NET 7 // METANIT.COM Сайт о программировании : сайт – [Б. м.], 2023. – URL: <https://metanit.com/sharp/tutorial/> (дата обращения: 04.07.2023).
3. Использование диаграммы классов UML при проектировании и документировании программного обеспечения // Хабр : [сайт] – [Б. м.], 2006-2023. – URL: <https://habr.com/ru/articles/572234/> (дата обращения: 04.07.2023).
4. Моделирование // Википедия Свободная энциклопедия : [сайт] – [Б. м. и г.]. – URL: <https://ru.wikipedia.org/wiki/Моделирование> (дата обращения 04.07.2023).

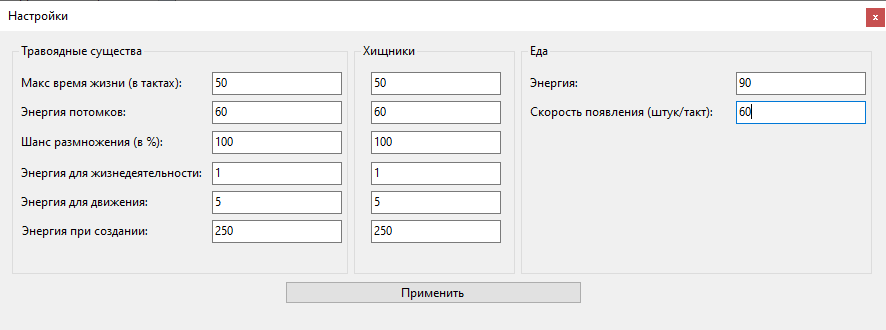
# Приложение А

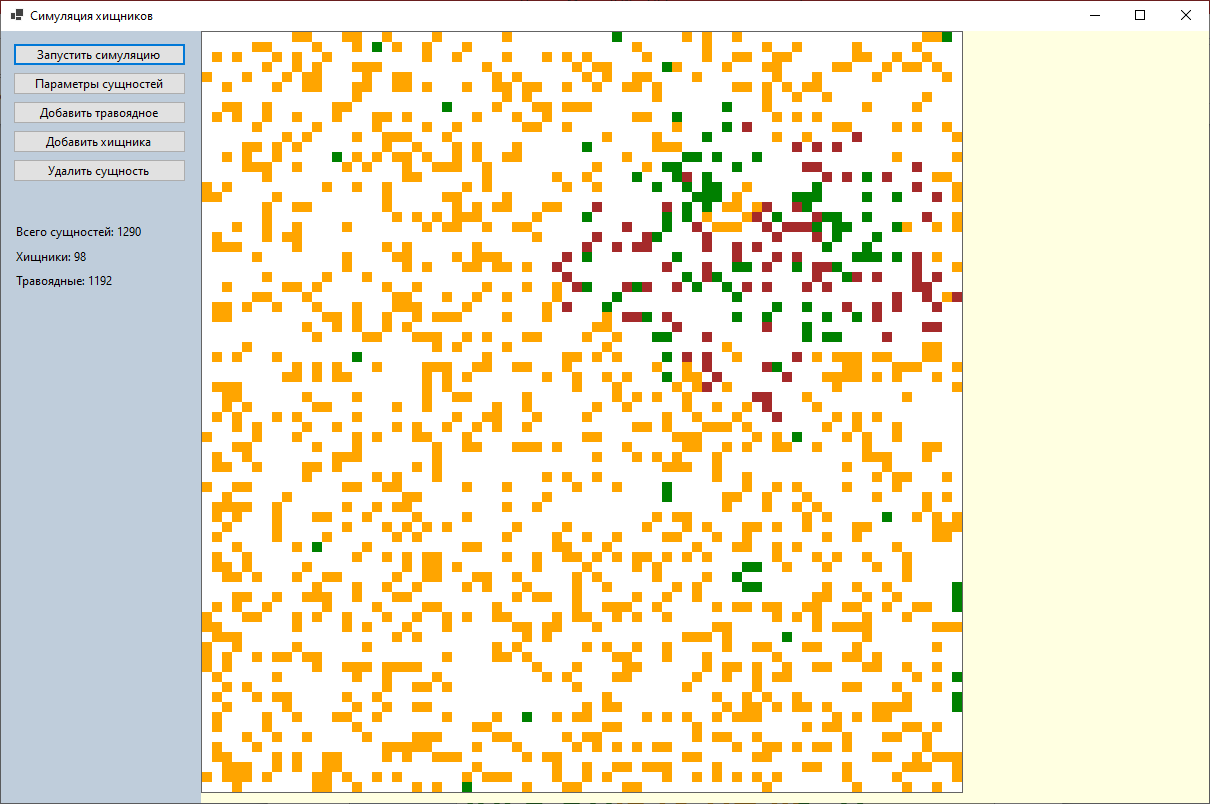
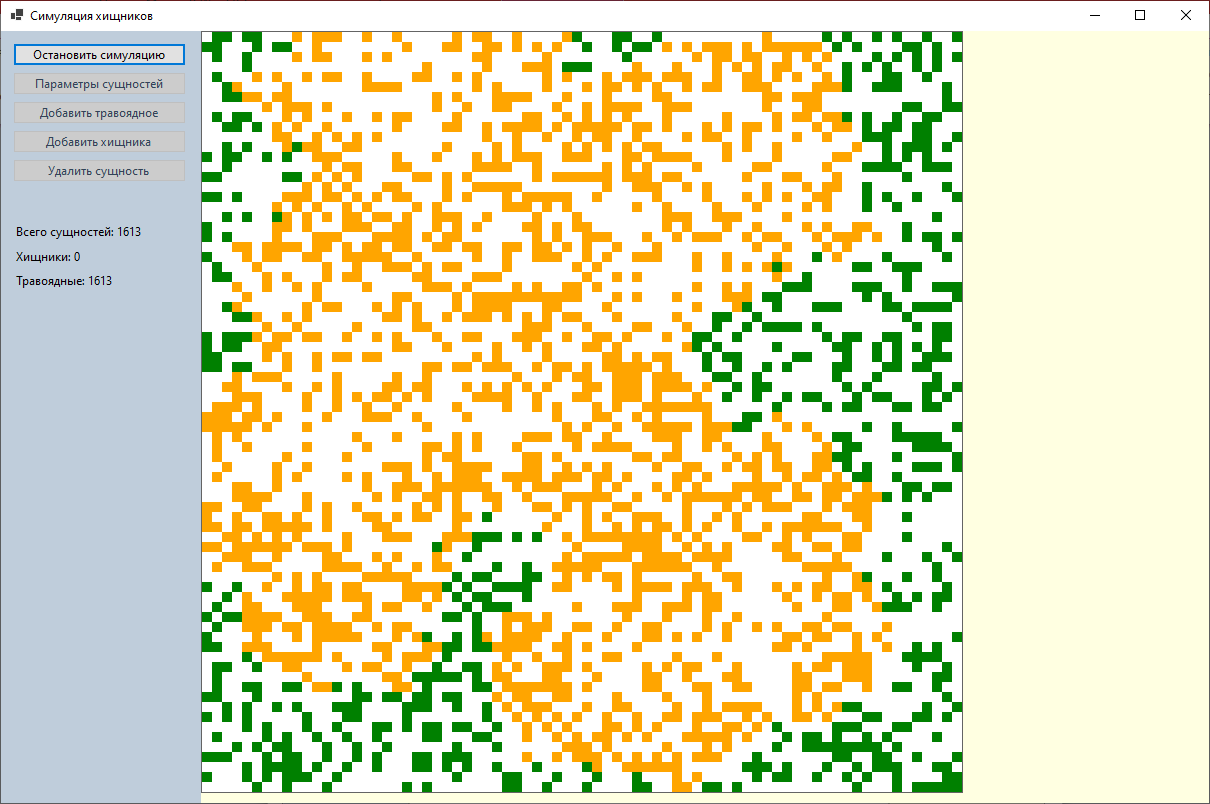
**Снимки экранных форм пользовательского интерфейса**

**Рисунок 6**. Приветственное окно и выбор симуляции

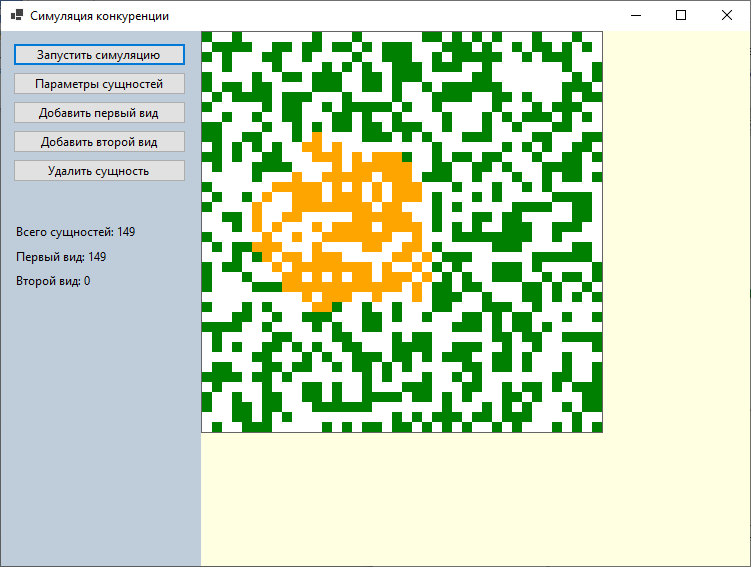
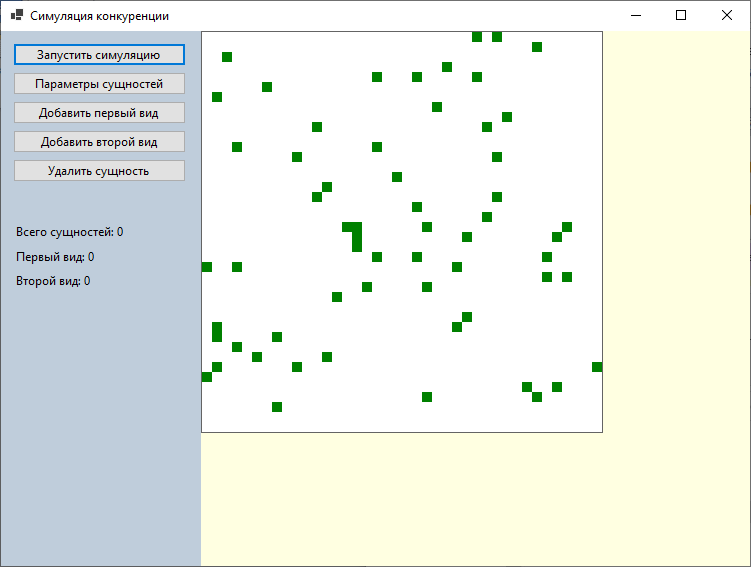
**Рисунок 7**. Размеры симуляционного поля

**Рисунок 8**. Окно симуляции хищников перед началом симуляции

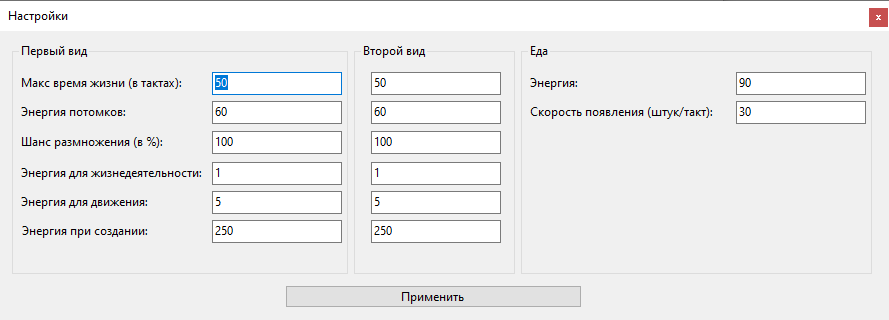
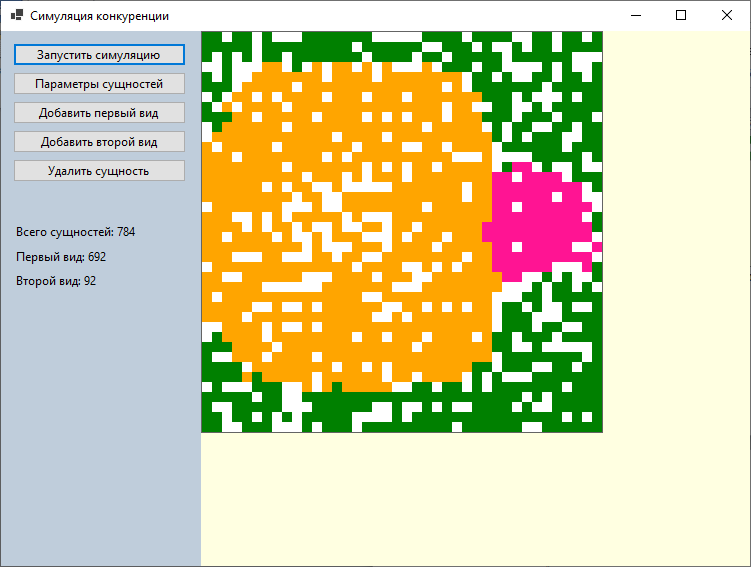
**Рисунок 9**. Окно параметров сущностей в симуляции хищников

**Рисунок 10**. Симуляция хищников, запущенная только с травоядными

**Рисунок 11**. Симуляция хищников, остановленная после добавления хищников

**Рисунок 12**. Симуляция конкуренции без добавления конкурентов

**Рисунок 13**. Симуляция конкуренции после добавления первого вида конкурентов

**Рисунок 14**. Симуляция конкуренции после добавления второго вида

**Рисунок 15**. Окно параметров симуляции конкуренции

# Приложение Б

**Исходный код программы**

Весь исходный код размещен на Github:

https://github.com/GrechishkinEgor/Practice.git

1. **CompetitionSimulationForm.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 Practice

{

public partial class CompetitionSimulationForm : Form

{

private fieldCompetitionSimulation simField;

public CompetitionSimulationForm()

{

InitializeComponent();

WidthAndHeightForm sizeWin = new WidthAndHeightForm();

if (sizeWin.ShowDialog() == DialogResult.OK)

{

simField = new fieldCompetitionSimulation(sizeWin.WidthField, sizeWin.HeightField);

this.Controls.Add(simField.GetPicture);

FieldPanel.SendToBack();

this.Show();

}

else

this.Close();

}

private void CompetitionSimulationForm\_FormClosed(object sender, FormClosedEventArgs e)

{

Program.CloseApplication();

}

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

{

simField.DoBeat();

ElemEntCountLabel.Text = "Всего сущностей: " + Convert.ToString(simField.FirstKindCount + simField.SecondKindCount);

FirstKindCountLabel.Text = "Первый вид: " + Convert.ToString(simField.FirstKindCount);

SecondKindCountLabel.Text = "Второй вид: " + Convert.ToString(simField.SecondKindCount);

}

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

{

if (SimRunButton.Text == "Запустить симуляцию")

{

SimRunButton.Text = "Остановить симуляцию";

Beat.Enabled = true;

EntitySettingsButton.Enabled = false;

AddFirstKindButton.Enabled = false;

AddSecondKindButton.Enabled = false;

DeleteEntityButton.Enabled = false;

}

else

{

Beat.Enabled = false;

SimRunButton.Text = "Запустить симуляцию";

EntitySettingsButton.Enabled = true;

AddFirstKindButton.Enabled = true;

AddSecondKindButton.Enabled = true;

DeleteEntityButton.Enabled = true;

}

}

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

{

simField.SettingsWin.ShowDialog();

}

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

{

if (AddFirstKindButton.Text == "Добавить первый вид")

{

AddFirstKindButton.Text = "Завершить добавление";

SimRunButton.Enabled = false;

EntitySettingsButton.Enabled = false;

AddSecondKindButton.Enabled = false;

DeleteEntityButton.Enabled = false;

simField.IsAdditionalMood = true;

simField.IsSecondKindAdded = false;

}

else

{

AddFirstKindButton.Text = "Добавить первый вид";

SimRunButton.Enabled = true;

EntitySettingsButton.Enabled = true;

AddSecondKindButton.Enabled = true;

DeleteEntityButton.Enabled = true;

simField.IsAdditionalMood = false;

}

}

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

{

if (AddSecondKindButton.Text == "Добавить второй вид")

{

AddSecondKindButton.Text = "Завершить добавление";

SimRunButton.Enabled = false;

EntitySettingsButton.Enabled = false;

AddFirstKindButton.Enabled = false;

DeleteEntityButton.Enabled = false;

simField.IsAdditionalMood = true;

simField.IsSecondKindAdded = true;

}

else

{

AddSecondKindButton.Text = "Добавить второй вид";

SimRunButton.Enabled = true;

EntitySettingsButton.Enabled = true;

AddFirstKindButton.Enabled = true;

DeleteEntityButton.Enabled = true;

simField.IsAdditionalMood = false;

}

}

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

{

if (DeleteEntityButton.Text == "Удалить сущность")

{

DeleteEntityButton.Text = "Завершить удаление";

SimRunButton.Enabled = false;

EntitySettingsButton.Enabled = false;

AddFirstKindButton.Enabled = false;

AddSecondKindButton.Enabled = false;

simField.IsDeletingMood = true;

}

else

{

DeleteEntityButton.Text = "Удалить сущность";

SimRunButton.Enabled = true;

EntitySettingsButton.Enabled = true;

AddFirstKindButton.Enabled = true;

AddSecondKindButton.Enabled = true;

simField.IsDeletingMood = false;

}

}

}

}

1. **Competitor.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Practice

{

internal class Competitor : elementaryEntity

{

public bool IsSecondKind { get; set; } = false;

public Competitor()

{

Type = "Competitor";

}

public override void BeatAction()

{

SettingsCompetitionSimulation settings = ((fieldCompetitionSimulation)entityBase).SettingsWin;

lifeTime++;

int vectorX, vectorY; //Координаты вектора

Random randomizer = new Random();

if (IsSecondKind)

{

energy = energy - settings.EnergyForLifeSecondKind;

if (energy <= 0)

{

energy = 0;

IsAlive = false;

}

if (lifeTime > settings.MaxLifeTimeSecondKind)

IsAlive = false;

if (!IsAlive)

return;

//Попытка размножения

int energyForReproduction = 3 \* (settings.EnergyForLifeSecondKind + settings.EnergyForMoveSecondKind) + settings.EnergyForChildSecondKind;

if (energy > energyForReproduction &&

randomizer.Next(1000) + 1 <= settings.ReproductionChanceSecondKind)

{

//Создание дочерней сущности

Competitor child = new Competitor();

child.energy = settings.EnergyForChildSecondKind;

child.IsSecondKind = true;

//Попытки разместить дочернюю сущность на симуляционном поле

for (int i = 0; i < 16; i++)

{

vectorX = randomizer.Next(-1, 2);

vectorY = randomizer.Next(-1, 2);

try

{

if (vectorX != 0 && vectorY != 0 &&

entityBase.EntityMatrix[this.x + vectorX, this.y + vectorY].Type == "emptyEntity")

{

//Успешная попытка. Завершить тактовое действие

entityBase.AddEntity(child, this.x + vectorX, this.y + vectorY);

this.energy -= settings.EnergyForChildSecondKind;

return;

}

}

catch (Exception) { }

}

}

//Попытки употребления еды

for (int i = 0; i < 16; i++)

{

vectorX = randomizer.Next(-1, 2);

vectorY = randomizer.Next(-1, 2);

try

{

if (entityBase.EntityMatrix[this.x + vectorX, this.y + vectorY].Type == "food")

{

//Успешная попытка. Завершить тактовое действие

this.energy += settings.FoodEnergy;

entityBase.ClearEntity(this.x + vectorX, this.y + vectorY);

return;

}

}

catch (Exception) { }

}

//Попытки найти подходящую для перемещения клетку

for (int i = 0; i < 16 && this.energy - settings.EnergyForMoveSecondKind > 0; i++)

{

vectorX = randomizer.Next(-1, 2);

vectorY = randomizer.Next(-1, 2);

try

{

if (entityBase.EntityMatrix[x + vectorX, y + vectorY].Type == "emptyEntity")

{

//Успешное перемещение и завершение тактового действия

entityBase.MoveEntity(x, y, x + vectorX, y + vectorY);

this.energy -= settings.EnergyForMoveSecondKind;

return;

}

}

catch (Exception) { }

}

}

else

{

energy = energy - settings.EnergyForLife;

if (energy <= 0)

{

energy = 0;

IsAlive = false;

}

if (lifeTime > settings.MaxLifeTime)

IsAlive = false;

if (!IsAlive)

return;

//Попытка размножения

int energyForReproduction = 3 \* (settings.EnergyForLife + settings.EnergyForMove) + settings.EnergyForChild;

if (energy > energyForReproduction &&

randomizer.Next(1000) + 1 <= settings.ReproductionChance)

{

//Создание дочерней сущности

Competitor child = new Competitor();

child.energy = settings.EnergyForChild;

child.IsSecondKind = false;

//Попытки разместить дочернюю сущность на симуляционном поле

for (int i = 0; i < 16; i++)

{

vectorX = randomizer.Next(-1, 2);

vectorY = randomizer.Next(-1, 2);

try

{

if (vectorX != 0 && vectorY != 0 &&

entityBase.EntityMatrix[this.x + vectorX, this.y + vectorY].Type == "emptyEntity")

{

//Успешная попытка. Завершить тактовое действие

entityBase.AddEntity(child, this.x + vectorX, this.y + vectorY);

this.energy -= settings.EnergyForChild;

return;

}

}

catch (Exception) { }

}

}

//Попытки употребления еды

for (int i = 0; i < 16; i++)

{

vectorX = randomizer.Next(-1, 2);

vectorY = randomizer.Next(-1, 2);

try

{

if (entityBase.EntityMatrix[this.x + vectorX, this.y + vectorY].Type == "food")

{

//Успешная попытка. Завершить тактовое действие

this.energy += settings.FoodEnergy;

entityBase.ClearEntity(this.x + vectorX, this.y + vectorY);

return;

}

}

catch (Exception) { }

}

//Попытки найти подходящую для перемещения клетку

for (int i = 0; i < 16 && this.energy - settings.EnergyForMove > 0; i++)

{

vectorX = randomizer.Next(-1, 2);

vectorY = randomizer.Next(-1, 2);

try

{

if (entityBase.EntityMatrix[x + vectorX, y + vectorY].Type == "emptyEntity")

{

//Успешное перемещение и завершение тактового действия

entityBase.MoveEntity(x, y, x + vectorX, y + vectorY);

this.energy -= settings.EnergyForMove;

return;

}

}

catch (Exception) { }

}

}

}

}

}

1. **elementaryEntity.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Reflection.Metadata.Ecma335;

using System.Text;

using System.Threading.Tasks;

namespace Practice

{

internal class elementaryEntity : entity

{

//Характеристики сущности

protected int energy = 100;

protected int lifeTime = 0;

public bool IsAlive { get; set; } = true;

//Свойства класса. Наследники могут задавать свои set'ы

public virtual int Energy

{

get => energy;

set { if (value >= 0) energy = value; }

}

public virtual int LifeTime

{

get => lifeTime;

set { if (value >= 0) lifeTime = value; }

}

//Переопредленное поведение существа

public override void BeatAction()

{

SettingsElementarySimulation settings = ((fieldElementarySimulation)entityBase).SettingsWin;

lifeTime++;

energy = energy - settings.EnergyForLife;

if (energy <= 0)

{

energy = 0;

IsAlive = false;

}

if (lifeTime > settings.MaxLifeTime)

{

IsAlive = false;

}

if (!IsAlive)

return;

int vectorX, vectorY; //Координаты вектора

Random randomizer = new Random();

//Попытка размножения

int energyForReproduction = 3 \* (settings.EnergyForLife + settings.EnergyForMove) + settings.EnergyForChild;

if (energy > energyForReproduction &&

randomizer.Next(1000) + 1 <= settings.ReproductionChance)

{

//Создание дочерней сущности

elementaryEntity child = new elementaryEntity();

child.energy = settings.EnergyForChild;

//Попытки разместить дочернюю сущность на симуляционном поле

for (int i = 0; i < 16; i++)

{

vectorX = randomizer.Next(-1, 2);

vectorY = randomizer.Next(-1, 2);

try

{

if (vectorX != 0 && vectorY != 0 &&

entityBase.EntityMatrix[this.x + vectorX, this.y + vectorY].Type == "emptyEntity")

{

//Успешная попытка. Завершить тактовое действие

entityBase.AddEntity(child, this.x + vectorX, this.y + vectorY);

this.energy -= settings.EnergyForChild;

return;

}

}

catch (Exception) { }

}

}

//Попытки употребления еды

for (int i = 0; i < 16; i++)

{

vectorX = randomizer.Next(-1, 2);

vectorY = randomizer.Next(-1, 2);

try

{

if (entityBase.EntityMatrix[this.x + vectorX, this.y + vectorY].Type == "food")

{

//Успешная попытка. Завершить тактовое действие

this.energy += settings.FoodEnergy;

entityBase.ClearEntity(this.x + vectorX, this.y + vectorY);

return;

}

}

catch (Exception) { }

}

//Попытки найти подходящую для перемещения клетку

for (int i = 0; i < 16 && this.energy - settings.EnergyForMove > 0; i++)

{

vectorX = randomizer.Next(-1, 2);

vectorY = randomizer.Next(-1, 2);

try

{

if (entityBase.EntityMatrix[x + vectorX, y + vectorY].Type == "emptyEntity")

{

//Успешное перемещение и завершение тактового действия

entityBase.MoveEntity(x, y, x + vectorX, y + vectorY);

this.energy -= settings.EnergyForMove;

return;

}

}

catch (Exception) { }

}

}

public elementaryEntity()

{

Type = "elementaryEntity";

}

//Создает полную копию сущности

public elementaryEntity(elementaryEntity obj)

{

Type = "elementaryEntity";

this.x = obj.x;

this.y = obj.y;

this.entityBase = obj.entityBase;

this.energy = obj.energy;

this.lifeTime = obj.lifeTime;

}

}

}

1. **ElementarySimulationForm.cs**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Diagnostics.Eventing.Reader;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Practice

{

public partial class ElementarySimulationForm : Form

{

private fieldElementarySimulation simField;

public ElementarySimulationForm()

{

InitializeComponent();

WidthAndHeightForm sizeWin = new WidthAndHeightForm();

if (sizeWin.ShowDialog() == DialogResult.OK)

{

simField = new fieldElementarySimulation(sizeWin.WidthField, sizeWin.HeightField);

this.Controls.Add(simField.GetPicture);

FieldPanel.SendToBack();

this.Show();

}

else

this.Close();

}

private void ElementarySimulationForm\_FormClosed(object sender, FormClosedEventArgs e)

{

Program.CloseApplication();

}

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

{

//simField.EnabledDrawing = false;

//Beat.Enabled = Beat.Enabled == true ? false : true;

//simField.DoBeat();

}

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

{

simField.DoBeat();

ElemEntCountLabel.Text = "Сущностей: " + Convert.ToString(simField.ElemEntCount);

}

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

{

if (SimRunButton.Text == "Запустить симуляцию")

{

SimRunButton.Text = "Остановить симуляцию";

Beat.Enabled = true;

EntitySettingsButton.Enabled = false;

AddEntityButton.Enabled = false;

DeleteEntityButton.Enabled = false;

DrawingButton.Enabled = false;

}

else

{

Beat.Enabled = false;

SimRunButton.Text = "Запустить симуляцию";

EntitySettingsButton.Enabled = true;

AddEntityButton.Enabled = true;

DeleteEntityButton.Enabled = true;

DrawingButton.Enabled = true;

}

}

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

{

if (AddEntityButton.Text == "Добавить сущность")

{

AddEntityButton.Text = "Завершить добавление";

SimRunButton.Enabled = false;

EntitySettingsButton.Enabled = false;

DeleteEntityButton.Enabled = false;

DrawingButton.Enabled = false;

simField.IsAdditionalMood = true;

}

else

{

AddEntityButton.Text = "Добавить сущность";

SimRunButton.Enabled = true;

EntitySettingsButton.Enabled = true;

DeleteEntityButton.Enabled = true;

DrawingButton.Enabled = true;

simField.IsAdditionalMood = false;

}

}

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

{

}

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

{

}

private void FieldPanel\_Paint(object sender, PaintEventArgs e)

{

}

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

{

}

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

{

if (DeleteEntityButton.Text == "Удалить сущность")

{

DeleteEntityButton.Text = "Завершить удаление";

SimRunButton.Enabled = false;

EntitySettingsButton.Enabled = false;

AddEntityButton.Enabled = false;

DrawingButton.Enabled = false;

simField.IsDeletingMood = true;

}

else

{

DeleteEntityButton.Text = "Удалить сущность";

SimRunButton.Enabled = true;

EntitySettingsButton.Enabled = true;

AddEntityButton.Enabled = true;

DrawingButton.Enabled = true;

simField.IsDeletingMood = false;

}

}

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

{

}

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

{

if (DrawingButton.Text == "Выключить отрисовку")

{

DrawingButton.Text = "Включить отрисовку";

simField.EnabledDrawing = false;

}

else

{

DrawingButton.Text = "Выключить отрисовку";

simField.EnabledDrawing = true;

}

}

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

{

simField.SettingsWin.ShowDialog();

}

}

}

1. **emptyEntity.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Practice

{

internal class emptyEntity : entity

{

public override void BeatAction()

{

//Ничего не делает

}

public emptyEntity()

{

Type = "emptyEntity";

}

}

}

1. **entity.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Practice

{

internal abstract class entity

{

//Координаты сущности

protected int x = 0;

protected int y = 0;

//Свойства класса для изменения координат. Возможно переопределение свойств

public virtual int X

{

get => x;

set { if (value >= 0) x = value; }

}

public virtual int Y

{

get => y;

set { if (value >= 0) y = value; }

}

//Симуляционное поле, которому принадлежит сущность

protected field entityBase;

public virtual field EntityBase { get => entityBase; set { entityBase = value; } }

//Тип сущности. При наследовании необходимо изменить значение

public string Type { get; protected set; } = "entity";

//Тактовое действие сущности. Конкретное поведение реализуют наследники

public abstract void BeatAction();

public entity()

{

}

}

}

1. **field.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Practice

{

internal abstract class field

{

public entity[,] EntityMatrix { get; } = { };

public int Width { get; } = 0;

public int Height { get; } = 0;

public field(int width, int height)

{

EntityMatrix = new entity[width, height];

for (int i = 0; i < width; i++)

for (int j = 0; j < height; j++)

{

EntityMatrix[i, j] = new emptyEntity();

EntityMatrix[i, j].EntityBase = this;

}

this.Width = width;

this.Height = height;

}

public virtual void DoBeat()

{

foreach (entity someEntity in EntityMatrix)

someEntity.BeatAction();

}

public virtual void MoveEntity(int oldX, int oldY, int newX, int newY)

{

EntityMatrix[newX, newY] = EntityMatrix[oldX, oldY];

EntityMatrix[newX, newY].X = newX;

EntityMatrix[newX, newY].Y = newY;

EntityMatrix[oldX, oldY] = new emptyEntity();

EntityMatrix[oldX, oldY].EntityBase = this;

EntityMatrix[oldX, oldY].X = oldX;

EntityMatrix[oldX, oldY].Y = oldY;

}

public virtual void ClearEntity(int x, int y)

{

EntityMatrix[x, y] = new emptyEntity();

EntityMatrix[x, y].EntityBase = this;

EntityMatrix[x, y].X = x;

EntityMatrix[x, y].Y = y;

}

public virtual void AddEntity(entity newEntity, int x, int y)

{

EntityMatrix[x, y] = newEntity;

newEntity.EntityBase = this;

newEntity.X = x;

newEntity.Y = y;

}

}

}

1. **fieldCompetitionSimulation.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Practice

{

internal abstract class field

{

public entity[,] EntityMatrix { get; } = { };

public int Width { get; } = 0;

public int Height { get; } = 0;

public field(int width, int height)

{

EntityMatrix = new entity[width, height];

for (int i = 0; i < width; i++)

for (int j = 0; j < height; j++)

{

EntityMatrix[i, j] = new emptyEntity();

EntityMatrix[i, j].EntityBase = this;

}

this.Width = width;

this.Height = height;

}

public virtual void DoBeat()

{

foreach (entity someEntity in EntityMatrix)

someEntity.BeatAction();

}

public virtual void MoveEntity(int oldX, int oldY, int newX, int newY)

{

EntityMatrix[newX, newY] = EntityMatrix[oldX, oldY];

EntityMatrix[newX, newY].X = newX;

EntityMatrix[newX, newY].Y = newY;

EntityMatrix[oldX, oldY] = new emptyEntity();

EntityMatrix[oldX, oldY].EntityBase = this;

EntityMatrix[oldX, oldY].X = oldX;

EntityMatrix[oldX, oldY].Y = oldY;

}

public virtual void ClearEntity(int x, int y)

{

EntityMatrix[x, y] = new emptyEntity();

EntityMatrix[x, y].EntityBase = this;

EntityMatrix[x, y].X = x;

EntityMatrix[x, y].Y = y;

}

public virtual void AddEntity(entity newEntity, int x, int y)

{

EntityMatrix[x, y] = newEntity;

newEntity.EntityBase = this;

newEntity.X = x;

newEntity.Y = y;

}

}

}

1. **fieldElementarySimulation.cs**

using System;

using System.Collections.Generic;

using System.Drawing.Imaging;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Practice

{

internal class fieldElementarySimulation : field

{

//Определяет, работает ли отрисовка сущностей на экране

protected bool enabledDrawing = true;

//Очередь на обработку тактового действия

protected Queue<elementaryEntity> beatQueue = new Queue<elementaryEntity>();

//Количество элементарных сущностей на поле

public int ElemEntCount { get => beatQueue.Count; }

//Включен ли режим добавления новых сущностей

protected bool isAdditionalMood = false;

//Включен ли режим удаления сущностей

protected bool isDeletingMood = false;

//Окно параметров сущностей

public SettingsElementarySimulation SettingsWin { get; } = new SettingsElementarySimulation();

//Изображение симуляционного поля

protected Bitmap fieldBitmap;

protected PictureBox fieldBitmapPictureBox;

public PictureBox GetPicture { get => fieldBitmapPictureBox; }

public bool IsDeletingMood

{

get => isDeletingMood;

set => isDeletingMood = value;

}

public bool IsAdditionalMood

{

get => isAdditionalMood;

set => isAdditionalMood = value;

}

//Работает или нет отрисовка сущностей

public bool EnabledDrawing

{

get => enabledDrawing;

set

{

if (value == false)

fieldBitmapPictureBox.Visible = false;

else

{

fieldBitmapPictureBox.Invalidate();

fieldBitmapPictureBox.Visible = true;

}

enabledDrawing = value;

}

}

public override void DoBeat()

{

//Генерация еды

Random randomizer = new Random();

int x, y;

for (int i = 0; i < SettingsWin.FoodGenerationSpeed; i++)

{

//Попытка сгенерировать очередную еду

for (int j = 0; j < 16; j++)

{

x = randomizer.Next(Width);

y = randomizer.Next(Height);

if (EntityMatrix[x,y].Type == "emptyEntity")

{

food newFood = new food();

this.AddEntity(newFood, x, y);

break;

}

}

}

//Обработка тактовых действий сущностей

int queueLength = beatQueue.Count;

for (int i = 0; i < queueLength; i++)

{

elementaryEntity someEntity = beatQueue.Dequeue();

if (someEntity.IsAlive)

{

someEntity.BeatAction();

if (someEntity.IsAlive)

beatQueue.Enqueue(someEntity);

else

this.ClearEntity(someEntity.X, someEntity.Y);

}

}

if (enabledDrawing)

fieldBitmapPictureBox.Invalidate();

}

//Согласовать изображение c сущностью по координатам (х, у)

protected virtual void HarmonizeEntityAndPicture(int x, int y)

{

switch(EntityMatrix[x,y].Type)

{

case "elementaryEntity":

for (int i = x \* 10; i < x \* 10 + 10; i++)

for (int j = y \* 10; j < y \* 10 + 10; j++)

fieldBitmap.SetPixel(i, j, Color.Orange);

break;

case "food":

for (int i = x \* 10; i < x \* 10 + 10; i++)

for (int j = y \* 10; j < y \* 10 + 10; j++)

fieldBitmap.SetPixel(i, j, Color.Green);

break;

case "emptyEntity":

default:

for (int i = x \* 10; i < x \* 10 + 10; i++)

for (int j = y \* 10; j < y \* 10 + 10; j++)

fieldBitmap.SetPixel(i, j, Color.White);

break;

}

}

public fieldElementarySimulation(int width, int height) : base(width, height)

{

fieldBitmap = new Bitmap(width \* 10, height \* 10);

fieldBitmapPictureBox = new PictureBox();

fieldBitmapPictureBox.Image = fieldBitmap;

fieldBitmapPictureBox.BackColor = System.Drawing.SystemColors.Control;

fieldBitmapPictureBox.BorderStyle = System.Windows.Forms.BorderStyle.FixedSingle;

fieldBitmapPictureBox.Location = new System.Drawing.Point(200, 0);

fieldBitmapPictureBox.BackColor = Color.White;

fieldBitmapPictureBox.Name = "BitmapPictureBox";

fieldBitmapPictureBox.Size = new System.Drawing.Size(width \* 10, height \* 10);

fieldBitmapPictureBox.SizeMode = System.Windows.Forms.PictureBoxSizeMode.AutoSize;

fieldBitmapPictureBox.TabIndex = 0;

fieldBitmapPictureBox.TabStop = false;

fieldBitmapPictureBox.MouseClick += this.EntityClick;

}

//Обработчик события нажатия на сущность

protected virtual void EntityClick(object sender, MouseEventArgs e)

{

try

{

PictureBox currentBox = (PictureBox)sender;

entity currentEntity = EntityMatrix[e.X / 10, e.Y / 10];

if (isAdditionalMood)

{

elementaryEntity ent = new elementaryEntity();

ent.Energy = SettingsWin.StartEnergy;

this.AddEntity(ent, e.X / 10, e.Y / 10);

fieldBitmapPictureBox.Invalidate();

}

if (isDeletingMood)

{

emptyEntity ent = new emptyEntity();

this.AddEntity(ent, e.X / 10, e.Y / 10);

fieldBitmapPictureBox.Invalidate();

}

}

catch { }

}

public override void AddEntity(entity newEntity, int x, int y)

{

if (EntityMatrix[x, y].Type == "elementaryEntity")

((elementaryEntity)EntityMatrix[x, y]).IsAlive = false;

EntityMatrix[x, y] = newEntity;

newEntity.EntityBase = this;

newEntity.X = x;

newEntity.Y = y;

HarmonizeEntityAndPicture(x, y);

if (newEntity.Type == "elementaryEntity")

beatQueue.Enqueue((elementaryEntity)newEntity);

}

public override void ClearEntity(int x, int y)

{

if (EntityMatrix[x, y].Type == "elementaryEntity")

((elementaryEntity)EntityMatrix[x, y]).IsAlive = false;

EntityMatrix[x, y] = new emptyEntity();

EntityMatrix[x, y].EntityBase = this;

EntityMatrix[x, y].X = x;

EntityMatrix[x, y].Y = y;

HarmonizeEntityAndPicture(x, y);

}

public override void MoveEntity(int oldX, int oldY, int newX, int newY)

{

EntityMatrix[newX, newY] = EntityMatrix[oldX, oldY];

EntityMatrix[newX, newY].X = newX;

EntityMatrix[newX, newY].Y = newY;

EntityMatrix[oldX, oldY] = new emptyEntity();

EntityMatrix[oldX, oldY].EntityBase = this;

EntityMatrix[oldX, oldY].X = oldX;

EntityMatrix[oldX, oldY].Y = oldY;

HarmonizeEntityAndPicture(oldX, oldY);

HarmonizeEntityAndPicture(newX, newY);

}

}

}

1. **fieldPredator.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Practice

{

internal class fieldPredator : fieldElementarySimulation

{

new public SettingsPredatorSimulation SettingsWin { get; } = new SettingsPredatorSimulation();

public bool IsPredatorAdded { get; set; } = false;

public fieldPredator(int width, int height) : base(width, height)

{

}

public int PredatorsCount { get; protected set; } = 0;

public override void DoBeat()

{

//Генерация еды

Random randomizer = new Random();

int x, y;

for (int i = 0; i < SettingsWin.FoodGenerationSpeed; i++)

{

//Попытка сгенерировать очередную еду

for (int j = 0; j < 16; j++)

{

x = randomizer.Next(Width);

y = randomizer.Next(Height);

if (EntityMatrix[x, y].Type == "emptyEntity")

{

food newFood = new food();

this.AddEntity(newFood, x, y);

break;

}

}

}

//Обработка тактовых действий сущностей

int queueLength = beatQueue.Count;

for (int i = 0; i < queueLength; i++)

{

elementaryEntity someEntity = beatQueue.Dequeue();

if (someEntity.IsAlive)

{

someEntity.BeatAction();

if (someEntity.IsAlive)

beatQueue.Enqueue(someEntity);

else

this.ClearEntity(someEntity.X, someEntity.Y);

}

}

if (enabledDrawing)

fieldBitmapPictureBox.Invalidate();

}

public override void ClearEntity(int x, int y)

{

if (EntityMatrix[x, y].Type == "Predator")

{

PredatorsCount--;

((Predator)EntityMatrix[x, y]).IsAlive = false;

}

if (EntityMatrix[x, y].Type == "Herbivore")

((Herbivore)EntityMatrix[x, y]).IsAlive = false;

EntityMatrix[x, y] = new emptyEntity();

EntityMatrix[x, y].EntityBase = this;

EntityMatrix[x, y].X = x;

EntityMatrix[x, y].Y = y;

HarmonizeEntityAndPicture(x, y);

}

public override void AddEntity(entity newEntity, int x, int y)

{

if (EntityMatrix[x, y].Type == "Herbivore" || EntityMatrix[x, y].Type == "Predator")

((elementaryEntity)EntityMatrix[x, y]).IsAlive = false;

EntityMatrix[x, y] = newEntity;

newEntity.EntityBase = this;

newEntity.X = x;

newEntity.Y = y;

HarmonizeEntityAndPicture(x, y);

if (newEntity.Type == "Herbivore" || newEntity.Type == "Predator")

beatQueue.Enqueue((elementaryEntity)newEntity);

if (newEntity.Type == "Predator")

PredatorsCount++;

}

protected override void HarmonizeEntityAndPicture(int x, int y)

{

switch (EntityMatrix[x, y].Type)

{

case "Herbivore":

for (int i = x \* 10; i < x \* 10 + 10; i++)

for (int j = y \* 10; j < y \* 10 + 10; j++)

fieldBitmap.SetPixel(i, j, Color.Orange);

break;

case "food":

for (int i = x \* 10; i < x \* 10 + 10; i++)

for (int j = y \* 10; j < y \* 10 + 10; j++)

fieldBitmap.SetPixel(i, j, Color.Green);

break;

case "Predator":

for (int i = x \* 10; i < x \* 10 + 10; i++)

for (int j = y \* 10; j < y \* 10 + 10; j++)

fieldBitmap.SetPixel(i, j, Color.Brown);

break;

case "emptyEntity":

default:

for (int i = x \* 10; i < x \* 10 + 10; i++)

for (int j = y \* 10; j < y \* 10 + 10; j++)

fieldBitmap.SetPixel(i, j, Color.White);

break;

}

}

protected override void EntityClick(object sender, MouseEventArgs e)

{

try

{

PictureBox currentBox = (PictureBox)sender;

entity currentEntity = EntityMatrix[e.X / 10, e.Y / 10];

if (isAdditionalMood && currentEntity.Type == "emptyEntity")

{

elementaryEntity ent;

if (IsPredatorAdded)

{

ent = new Predator();

ent.Energy = SettingsWin.StartEnergyPredator;

}

else

{

ent = new Herbivore();

ent.Energy = SettingsWin.StartEnergy;

}

this.AddEntity(ent, e.X / 10, e.Y / 10);

fieldBitmapPictureBox.Invalidate();

}

if (isDeletingMood)

{

emptyEntity ent = new emptyEntity();

this.AddEntity(ent, e.X / 10, e.Y / 10);

fieldBitmapPictureBox.Invalidate();

}

}

catch { }

}

}

}

1. **food.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Practice

{

internal class food : entity

{

public override void BeatAction()

{

//Еда ничего не делает

}

public food()

{

Type = "food";

}

}

}

1. **HelloForm.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 Practice

{

public partial class HelloForm : Form

{

public HelloForm()

{

InitializeComponent();

}

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

{

if (ChosenSimulationBox.Text != null && ChosenSimulationBox.Text != "")

{

Program.ChosenSimulation = ChosenSimulationBox.Text;

this.Hide();

Program.OpenSimulation();

}

}

private void HelloForm\_FormClosed(object sender, FormClosedEventArgs e)

{

}

}

}

1. **Herbivore.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Practice

{

internal class Herbivore : elementaryEntity

{

public Herbivore()

{

Type = "Herbivore";

}

public override void BeatAction()

{

SettingsPredatorSimulation settings = ((fieldPredator)entityBase).SettingsWin;

lifeTime++;

energy = energy - settings.EnergyForLife;

if (energy <= 0)

{

energy = 0;

IsAlive = false;

}

if (lifeTime > settings.MaxLifeTime)

{

IsAlive = false;

}

if (!IsAlive)

return;

int vectorX, vectorY; //Координаты вектора

Random randomizer = new Random();

//Попытка размножения

int energyForReproduction = 3 \* (settings.EnergyForLife + settings.EnergyForMove) + settings.EnergyForChild;

if (energy > energyForReproduction &&

randomizer.Next(1000) + 1 <= settings.ReproductionChance)

{

//Создание дочерней сущности

Herbivore child = new Herbivore();

child.energy = settings.EnergyForChild;

//Попытки разместить дочернюю сущность на симуляционном поле

for (int i = 0; i < 16; i++)

{

vectorX = randomizer.Next(-1, 2);

vectorY = randomizer.Next(-1, 2);

try

{

if (vectorX != 0 && vectorY != 0 &&

entityBase.EntityMatrix[this.x + vectorX, this.y + vectorY].Type == "emptyEntity")

{

//Успешная попытка. Завершить тактовое действие

entityBase.AddEntity(child, this.x + vectorX, this.y + vectorY);

this.energy -= settings.EnergyForChild;

return;

}

}

catch (Exception) { }

}

}

//Попытки употребления еды

for (int i = 0; i < 16; i++)

{

vectorX = randomizer.Next(-1, 2);

vectorY = randomizer.Next(-1, 2);

try

{

if (entityBase.EntityMatrix[this.x + vectorX, this.y + vectorY].Type == "food")

{

//Успешная попытка. Завершить тактовое действие

this.energy += settings.FoodEnergy;

entityBase.ClearEntity(this.x + vectorX, this.y + vectorY);

return;

}

}

catch (Exception) { }

}

//Попытки найти подходящую для перемещения клетку

for (int i = 0; i < 16 && this.energy - settings.EnergyForMove > 0; i++)

{

vectorX = randomizer.Next(-1, 2);

vectorY = randomizer.Next(-1, 2);

try

{

if (entityBase.EntityMatrix[x + vectorX, y + vectorY].Type == "emptyEntity")

{

//Успешное перемещение и завершение тактового действия

entityBase.MoveEntity(x, y, x + vectorX, y + vectorY);

this.energy -= settings.EnergyForMove;

return;

}

}

catch (Exception) { }

}

}

}

}

1. **Predator.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Practice

{

internal class Predator : elementaryEntity

{

public Predator()

{

Type = "Predator";

}

public override void BeatAction()

{

SettingsPredatorSimulation settings = ((fieldPredator)entityBase).SettingsWin;

lifeTime++;

energy = energy - settings.EnergyForLifePredator;

if (energy <= 0)

{

energy = 0;

IsAlive = false;

}

if (lifeTime > settings.MaxLifeTimePredator)

{

IsAlive = false;

}

if (!IsAlive)

return;

int vectorX, vectorY; //Координаты вектора

Random randomizer = new Random();

//Попытка размножения

int energyForReproduction = 3 \* (settings.EnergyForLifePredator + settings.EnergyForMovePredator) + settings.EnergyForChildPredator;

if (energy > energyForReproduction &&

randomizer.Next(1000) + 1 <= settings.ReproductionChancePredator)

{

//Создание дочерней сущности

Predator child = new Predator();

child.energy = settings.EnergyForChildPredator;

//Попытки разместить дочернюю сущность на симуляционном поле

for (int i = 0; i < 16; i++)

{

vectorX = randomizer.Next(-1, 2);

vectorY = randomizer.Next(-1, 2);

try

{

if (vectorX != 0 && vectorY != 0 &&

entityBase.EntityMatrix[this.x + vectorX, this.y + vectorY].Type == "emptyEntity")

{

//Успешная попытка. Завершить тактовое действие

entityBase.AddEntity(child, this.x + vectorX, this.y + vectorY);

this.energy -= settings.EnergyForChildPredator;

return;

}

}

catch (Exception) { }

}

}

//Попытки съесть другое существо

for (int i = 0; i < 16; i++)

{

vectorX = randomizer.Next(-1, 2);

vectorY = randomizer.Next(-1, 2);

try

{

if (entityBase.EntityMatrix[this.x + vectorX, this.y + vectorY].Type == "Herbivore")

{

//Успешная попытка. Завершить тактовое действие

elementaryEntity eaten = (elementaryEntity)entityBase.EntityMatrix[this.x + vectorX, this.y + vectorY];

this.energy += eaten.Energy;

eaten.IsAlive = false;

entityBase.ClearEntity(this.x + vectorX, this.y + vectorY);

return;

}

}

catch (Exception) { }

}

//Попытки найти подходящую для перемещения клетку

for (int i = 0; i < 16 && this.energy - settings.EnergyForMovePredator > 0; i++)

{

vectorX = randomizer.Next(-1, 2);

vectorY = randomizer.Next(-1, 2);

try

{

if (entityBase.EntityMatrix[x + vectorX, y + vectorY].Type == "emptyEntity")

{

//Успешное перемещение и завершение тактового действия

entityBase.MoveEntity(x, y, x + vectorX, y + vectorY);

this.energy -= settings.EnergyForMovePredator;

return;

}

}

catch (Exception) { }

}

}

}

}

1. **PredatorSimulationForm.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 Practice

{

public partial class PredatorSimulationForm : Form

{

private fieldPredator simField;

public PredatorSimulationForm()

{

InitializeComponent();

WidthAndHeightForm sizeWin = new WidthAndHeightForm();

if (sizeWin.ShowDialog() == DialogResult.OK)

{

simField = new fieldPredator(sizeWin.WidthField, sizeWin.HeightField);

this.Controls.Add(simField.GetPicture);

FieldPanel.SendToBack();

this.Show();

}

else

this.Close();

}

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

{

if (SimRunButton.Text == "Запустить симуляцию")

{

SimRunButton.Text = "Остановить симуляцию";

Beat.Enabled = true;

EntitySettingsButton.Enabled = false;

AddHerbivoreButton.Enabled = false;

AddPredatorButton.Enabled = false;

DeleteEntityButton.Enabled = false;

DrawingButton.Enabled = false;

}

else

{

Beat.Enabled = false;

SimRunButton.Text = "Запустить симуляцию";

EntitySettingsButton.Enabled = true;

AddHerbivoreButton.Enabled = true;

AddPredatorButton.Enabled = true;

DeleteEntityButton.Enabled = true;

DrawingButton.Enabled = true;

}

}

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

{

simField.SettingsWin.ShowDialog();

}

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

{

if (AddHerbivoreButton.Text == "Добавить травоядное")

{

AddHerbivoreButton.Text = "Завершить добавление";

SimRunButton.Enabled = false;

EntitySettingsButton.Enabled = false;

AddPredatorButton.Enabled = false;

DeleteEntityButton.Enabled = false;

DrawingButton.Enabled = false;

simField.IsAdditionalMood = true;

}

else

{

AddHerbivoreButton.Text = "Добавить травоядное";

SimRunButton.Enabled = true;

EntitySettingsButton.Enabled = true;

AddPredatorButton.Enabled = true;

DeleteEntityButton.Enabled = true;

DrawingButton.Enabled = true;

simField.IsAdditionalMood = false;

}

}

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

{

if (DeleteEntityButton.Text == "Удалить сущность")

{

DeleteEntityButton.Text = "Завершить удаление";

SimRunButton.Enabled = false;

EntitySettingsButton.Enabled = false;

AddHerbivoreButton.Enabled = false;

AddPredatorButton.Enabled = false;

DrawingButton.Enabled = false;

simField.IsDeletingMood = true;

}

else

{

DeleteEntityButton.Text = "Удалить сущность";

SimRunButton.Enabled = true;

EntitySettingsButton.Enabled = true;

AddHerbivoreButton.Enabled = true;

AddPredatorButton.Enabled = true;

DrawingButton.Enabled = true;

simField.IsDeletingMood = false;

}

}

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

{

if (AddPredatorButton.Text == "Добавить хищника")

{

AddPredatorButton.Text = "Завершить добавление";

SimRunButton.Enabled = false;

EntitySettingsButton.Enabled = false;

AddHerbivoreButton.Enabled = false;

DeleteEntityButton.Enabled = false;

DrawingButton.Enabled = false;

simField.IsAdditionalMood = true;

simField.IsPredatorAdded = true;

}

else

{

AddPredatorButton.Text = "Добавить хищника";

SimRunButton.Enabled = true;

EntitySettingsButton.Enabled = true;

AddHerbivoreButton.Enabled = true;

DeleteEntityButton.Enabled = true;

DrawingButton.Enabled = true;

simField.IsAdditionalMood = false;

simField.IsPredatorAdded = false;

}

}

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

{

simField.DoBeat();

ElemEntCountLabel.Text = "Всего сущностей: " + Convert.ToString(simField.ElemEntCount);

HerbivoreCountLabel.Text = "Травоядные: " + Convert.ToString(simField.ElemEntCount - simField.PredatorsCount);

PredatorsCountLabel.Text = "Хищники: " + Convert.ToString(simField.PredatorsCount);

}

private void PredatorSimulationForm\_FormClosed(object sender, FormClosedEventArgs e)

{

Program.CloseApplication();

}

}

}

1. **Program.cs**

namespace Practice

{

internal static class Program

{

private static HelloForm helloForm = new HelloForm();

public static string ChosenSimulation { get; set; } = "";

public static void CloseApplication()

{

helloForm.Close();

}

public static void OpenHelloForm()

{

helloForm.Show();

}

public static void OpenSimulation()

{

switch (ChosenSimulation)

{

case "Элементарная симуляция":

ElementarySimulationForm simForm = new ElementarySimulationForm();

break;

case "Симуляция хищников":

PredatorSimulationForm simForm1 = new PredatorSimulationForm();

break;

case "Симуляция конкуренции":

CompetitionSimulationForm simForm2 = new CompetitionSimulationForm();

break;

case "":

case null:

default:

helloForm.Show();

break;

}

}

[STAThread]

static void Main()

{

ApplicationConfiguration.Initialize();

Application.Run(helloForm);

}

}

}

1. **SettingsCompetitionSimulation.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 Practice

{

public partial class SettingsCompetitionSimulation : SettingsElementarySimulation

{

public int MaxLifeTimeSecondKind { get; private set; } = 50;

public int EnergyForChildSecondKind { get; private set; } = 60;

public int ReproductionChanceSecondKind { get; private set; } = 1000;

public int EnergyForLifeSecondKind { get; private set; } = 1;

public int EnergyForMoveSecondKind { get; private set; } = 5;

public int StartEnergySecondKind { get; private set; } = 250;

public SettingsCompetitionSimulation()

{

InitializeComponent();

}

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

{

MaxLifeTimeSecondKindTextBox.Text = Convert.ToString(MaxLifeTimeSecondKind);

EnergyForChildSecondKindTextBox.Text = Convert.ToString(EnergyForChildSecondKind);

ReproductionChanceSecondKindTextBox.Text = Convert.ToString(ReproductionChanceSecondKind / 10.0);

EnergyForLifeSecondKindTextBox.Text = Convert.ToString(EnergyForLifeSecondKind);

EnergyForMoveSecondKindTextBox.Text = Convert.ToString(EnergyForMoveSecondKind);

StartEnergySecondKindTextBox.Text = Convert.ToString(StartEnergySecondKind);

}

private void MaxLifeTimeSecondKindTextBox\_Leave(object sender, EventArgs e)

{

try

{

int newLifeTime = Convert.ToInt32(MaxLifeTimeSecondKindTextBox.Text);

if (newLifeTime < 0)

throw new Exception();

}

catch

{

MaxLifeTimeSecondKindTextBox.Text = Convert.ToString(MaxLifeTimeSecondKind);

}

}

private void EnergyForChildSecondKindTextBox\_Leave(object sender, EventArgs e)

{

try

{

int newEnergy = Convert.ToInt32(EnergyForChildSecondKindTextBox.Text);

if (newEnergy < 0)

throw new Exception();

}

catch

{

EnergyForChildTextBox.Text = Convert.ToString(EnergyForChildSecondKind);

}

}

private void ReproductionChanceSecondKindTextBox\_Leave(object sender, EventArgs e)

{

try

{

int newChance = (int)(Convert.ToDouble(ReproductionChanceSecondKindTextBox.Text) \* 10);

if (newChance < 0 || newChance > 1000)

throw new Exception();

ReproductionChanceSecondKindTextBox.Text = Convert.ToString(newChance / 10.0);

}

catch

{

ReproductionChanceSecondKindTextBox.Text = Convert.ToString(ReproductionChanceSecondKind / 10);

}

}

private void EnergyForLifeSecondKindTextBox\_Leave(object sender, EventArgs e)

{

try

{

int newEnergy = Convert.ToInt32(EnergyForLifeSecondKindTextBox.Text);

if (newEnergy < 0)

throw new Exception();

}

catch

{

EnergyForLifeSecondKindTextBox.Text = Convert.ToString(EnergyForLifeSecondKind);

}

}

private void EnergyForMoveSecondKindTextBox\_Leave(object sender, EventArgs e)

{

try

{

int newEnergy = Convert.ToInt32(EnergyForMoveSecondKindTextBox.Text);

if (newEnergy < 0)

throw new Exception();

}

catch

{

EnergyForMoveSecondKindTextBox.Text = Convert.ToString(EnergyForMoveSecondKind);

}

}

private void StartEnergySecondKindTextBox\_Leave(object sender, EventArgs e)

{

try

{

int newEnergy = Convert.ToInt32(StartEnergySecondKindTextBox.Text);

if (newEnergy < 0)

throw new Exception();

}

catch

{

StartEnergySecondKindTextBox.Text = Convert.ToString(StartEnergySecondKind);

}

}

private void OKButton\_Click\_1(object sender, EventArgs e)

{

MaxLifeTimeSecondKind = Convert.ToInt32(MaxLifeTimeSecondKindTextBox.Text);

EnergyForChildSecondKind = Convert.ToInt32(EnergyForChildSecondKindTextBox.Text);

ReproductionChanceSecondKind = (int)(Convert.ToDouble(ReproductionChanceSecondKindTextBox.Text) \* 10);

EnergyForLifeSecondKind = Convert.ToInt32(EnergyForLifeSecondKindTextBox.Text);

EnergyForMoveSecondKind = Convert.ToInt32(EnergyForMoveSecondKindTextBox.Text);

StartEnergySecondKind = Convert.ToInt32(StartEnergySecondKindTextBox.Text);

base.OKButton\_Click(sender, e);

}

}

}

1. **SettingsElementarySimulation.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 Practice

{

public partial class SettingsElementarySimulation : Form

{

public SettingsElementarySimulation()

{

InitializeComponent();

MaxLifeTimeTextBox.Text = Convert.ToString(MaxLifeTime);

EnergyForChildTextBox.Text = Convert.ToString(EnergyForChild);

ReproductionChanceTextBox.Text = Convert.ToString(ReproductionChance / 10);

EnergyForLifeTextBox.Text = Convert.ToString(EnergyForLife);

EnergyForMoveTextBox.Text = Convert.ToString(EnergyForMove);

StartEnergyTextBox.Text = Convert.ToString(StartEnergy);

FoodEnergyTextBox.Text = Convert.ToString(FoodEnergy);

FoodGenerationSpeedTextBox.Text = Convert.ToString(FoodGenerationSpeed);

}

public int MaxLifeTime { get; private set; } = 50;

public int EnergyForChild { get; private set; } = 60;

public int ReproductionChance { get; private set; } = 1000;

public int EnergyForLife { get; private set; } = 1;

public int EnergyForMove { get; private set; } = 5;

public int StartEnergy { get; private set; } = 250;

public int FoodEnergy { get; private set; } = 90;

public int FoodGenerationSpeed { get; private set; } = 30;

private void MaxLifeTimeTextBox\_Leave(object sender, EventArgs e)

{

try

{

int newLifeTime = Convert.ToInt32(MaxLifeTimeTextBox.Text);

if (newLifeTime < 0)

throw new Exception();

//MaxLifeTime = newLifeTime;

}

catch

{

MaxLifeTimeTextBox.Text = Convert.ToString(MaxLifeTime);

}

}

private void SettingsElementarySimulation\_FormClosing(object sender, FormClosingEventArgs e)

{

this.ActiveControl = null;

}

private void EnergyForChildTextBox\_Leave(object sender, EventArgs e)

{

try

{

int newEnergy = Convert.ToInt32(EnergyForChildTextBox.Text);

if (newEnergy < 0)

throw new Exception();

//EnergyForChild = newEnergy;

}

catch

{

EnergyForChildTextBox.Text = Convert.ToString(EnergyForChild);

}

}

private void ReproductionChanceTextBox\_Leave(object sender, EventArgs e)

{

try

{

int newChance = (int)(Convert.ToDouble(ReproductionChanceTextBox.Text) \* 10);

if (newChance < 0 || newChance > 1000)

throw new Exception();

//ReproductionChance = newChance;

ReproductionChanceTextBox.Text = Convert.ToString(newChance / 10.0);

}

catch

{

ReproductionChanceTextBox.Text = Convert.ToString(ReproductionChance / 10);

}

}

private void EnergyForLifeTextBox\_Leave(object sender, EventArgs e)

{

try

{

int newEnergy = Convert.ToInt32(EnergyForLifeTextBox.Text);

if (newEnergy < 0)

throw new Exception();

//EnergyForLife = newEnergy;

}

catch

{

EnergyForLifeTextBox.Text = Convert.ToString(EnergyForLife);

}

}

private void EnergyForMoveTextBox\_Leave(object sender, EventArgs e)

{

try

{

int newEnergy = Convert.ToInt32(EnergyForMoveTextBox.Text);

if (newEnergy < 0)

throw new Exception();

//EnergyForMove = newEnergy;

}

catch

{

EnergyForMoveTextBox.Text = Convert.ToString(EnergyForMove);

}

}

private void StartEnergyTextBox\_Leave(object sender, EventArgs e)

{

try

{

int newEnergy = Convert.ToInt32(StartEnergyTextBox.Text);

if (newEnergy < 0)

throw new Exception();

//StartEnergy = newEnergy;

}

catch

{

StartEnergyTextBox.Text = Convert.ToString(StartEnergy);

}

}

private void FoodEnergyTextBox\_Leave(object sender, EventArgs e)

{

try

{

int newEnergy = Convert.ToInt32(FoodEnergyTextBox.Text);

if (newEnergy < 0)

throw new Exception();

//FoodEnergy = newEnergy;

}

catch

{

FoodEnergyTextBox.Text = Convert.ToString(FoodEnergy);

}

}

private void FoodGenerationSpeedTextBox\_Leave(object sender, EventArgs e)

{

try

{

int newSpeed = Convert.ToInt32(FoodGenerationSpeedTextBox.Text);

if (newSpeed < 0)

throw new Exception();

//FoodGenerationSpeed = newSpeed;

}

catch

{

FoodGenerationSpeedTextBox.Text = Convert.ToString(FoodGenerationSpeed);

}

}

protected void OKButton\_Click(object sender, EventArgs e)

{

this.DialogResult = DialogResult.OK;

MaxLifeTime = Convert.ToInt32(MaxLifeTimeTextBox.Text);

EnergyForChild = Convert.ToInt32(EnergyForChildTextBox.Text);

ReproductionChance = (int)(Convert.ToDouble(ReproductionChanceTextBox.Text) \* 10);

EnergyForLife = Convert.ToInt32(EnergyForLifeTextBox.Text);

EnergyForMove = Convert.ToInt32(EnergyForMoveTextBox.Text);

StartEnergy = Convert.ToInt32(StartEnergyTextBox.Text);

FoodEnergy = Convert.ToInt32(FoodEnergyTextBox.Text);

FoodGenerationSpeed = Convert.ToInt32(FoodGenerationSpeedTextBox.Text);

this.Close();

}

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

{

this.DialogResult = DialogResult.None;

MaxLifeTimeTextBox.Text = Convert.ToString(MaxLifeTime);

EnergyForChildTextBox.Text = Convert.ToString(EnergyForChild);

ReproductionChanceTextBox.Text = Convert.ToString(ReproductionChance / 10);

EnergyForLifeTextBox.Text = Convert.ToString(EnergyForLife);

EnergyForMoveTextBox.Text = Convert.ToString(EnergyForMove);

StartEnergyTextBox.Text = Convert.ToString(StartEnergy);

FoodEnergyTextBox.Text = Convert.ToString(FoodEnergy);

FoodGenerationSpeedTextBox.Text = Convert.ToString(FoodGenerationSpeed);

}

}

}

1. **SettingsPredatorSimulation.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 Practice

{

public partial class SettingsPredatorSimulation : SettingsElementarySimulation

{

public int MaxLifeTimePredator { get; private set; } = 50;

public int EnergyForChildPredator { get; private set; } = 60;

public int ReproductionChancePredator { get; private set; } = 1000;

public int EnergyForLifePredator { get; private set; } = 1;

public int EnergyForMovePredator { get; private set; } = 5;

public int StartEnergyPredator { get; private set; } = 250;

public SettingsPredatorSimulation()

{

InitializeComponent();

}

private void textBox3\_TextChanged(object sender, EventArgs e)

{

}

private void MaxLifeTimePredatorTextBox\_Leave(object sender, EventArgs e)

{

try

{

int newLifeTime = Convert.ToInt32(MaxLifeTimePredatorTextBox.Text);

if (newLifeTime < 0)

throw new Exception();

}

catch

{

MaxLifeTimePredatorTextBox.Text = Convert.ToString(MaxLifeTimePredator);

}

}

private void EnergyForChildPredatorTextBox\_Leave(object sender, EventArgs e)

{

try

{

int newEnergy = Convert.ToInt32(EnergyForChildPredatorTextBox.Text);

if (newEnergy < 0)

throw new Exception();

}

catch

{

EnergyForChildTextBox.Text = Convert.ToString(EnergyForChildPredator);

}

}

private void ReproductionChancePredatorTextBox\_Leave(object sender, EventArgs e)

{

try

{

int newChance = (int)(Convert.ToDouble(ReproductionChancePredatorTextBox.Text) \* 10);

if (newChance < 0 || newChance > 1000)

throw new Exception();

ReproductionChancePredatorTextBox.Text = Convert.ToString(newChance / 10.0);

}

catch

{

ReproductionChancePredatorTextBox.Text = Convert.ToString(ReproductionChancePredator / 10);

}

}

private void EnergyForLifePredatorTextBox\_Leave(object sender, EventArgs e)

{

try

{

int newEnergy = Convert.ToInt32(EnergyForLifePredatorTextBox.Text);

if (newEnergy < 0)

throw new Exception();

}

catch

{

EnergyForLifePredatorTextBox.Text = Convert.ToString(EnergyForLifePredator);

}

}

private void EnergyForMovePredatorTextBox\_Leave(object sender, EventArgs e)

{

try

{

int newEnergy = Convert.ToInt32(EnergyForMovePredatorTextBox.Text);

if (newEnergy < 0)

throw new Exception();

}

catch

{

EnergyForMovePredatorTextBox.Text = Convert.ToString(EnergyForMovePredator);

}

}

private void StartEnergyPredatorTextBox\_Leave(object sender, EventArgs e)

{

try

{

int newEnergy = Convert.ToInt32(StartEnergyPredatorTextBox.Text);

if (newEnergy < 0)

throw new Exception();

}

catch

{

StartEnergyPredatorTextBox.Text = Convert.ToString(StartEnergyPredator);

}

}

new private void OKButton\_Click(object sender, EventArgs e)

{

MaxLifeTimePredator = Convert.ToInt32(MaxLifeTimePredatorTextBox.Text);

EnergyForChildPredator = Convert.ToInt32(EnergyForChildPredatorTextBox.Text);

ReproductionChancePredator = (int)(Convert.ToDouble(ReproductionChancePredatorTextBox.Text) \* 10);

EnergyForLifePredator = Convert.ToInt32(EnergyForLifePredatorTextBox.Text);

EnergyForMovePredator = Convert.ToInt32(EnergyForMovePredatorTextBox.Text);

StartEnergyPredator = Convert.ToInt32(StartEnergyPredatorTextBox.Text);

base.OKButton\_Click(sender, e);

}

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

{

MaxLifeTimePredatorTextBox.Text = Convert.ToString(MaxLifeTimePredator);

EnergyForChildPredatorTextBox.Text = Convert.ToString(EnergyForChildPredator);

ReproductionChancePredatorTextBox.Text = Convert.ToString(ReproductionChancePredator / 10.0);

EnergyForLifePredatorTextBox.Text = Convert.ToString(EnergyForLifePredator);

EnergyForMovePredatorTextBox.Text = Convert.ToString(EnergyForMovePredator);

StartEnergyPredatorTextBox.Text = Convert.ToString(StartEnergyPredator);

}

}

}

1. **WidthAndHeightForm.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 Practice

{

public partial class WidthAndHeightForm : Form

{

public int WidthField { get; private set; } = 0;

public int HeightField { get; private set; } = 0;

private bool IsOKButtonPressed { get; set; } = false;

public WidthAndHeightForm()

{

InitializeComponent();

}

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

{

IsOKButtonPressed = true;

this.Close();

}

private void WidthAndHeightForm\_FormClosing(object sender, FormClosingEventArgs e)

{

if (IsOKButtonPressed)

try

{

WidthField = Convert.ToInt32(WidthTextBox.Text);

HeightField = Convert.ToInt32(HeightTextBox.Text);

if (WidthField <= 0 || HeightField <= 0 || WidthField \* HeightField > 10000)

throw new Exception();

this.DialogResult = DialogResult.OK;

}

catch (Exception)

{

IsOKButtonPressed = false;

WidthTextBox.Text = "";

HeightTextBox.Text = "";

WidthField = 0;

HeightField = 0;

MessageBox.Show("Значения ширины или высоты некорректны. \nВаше поле не должно быть больше 10000 клеток.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

e.Cancel = true;

}

else

{

Program.OpenHelloForm();

this.DialogResult = DialogResult.Cancel;

}

}

}

}