**Министерство образования и науки Российской Федерации**

**федеральное государственное бюджетное образовательное учреждение высшего профессионального образования**

**«Российский экономический университет имени Г.В. Плеханова»**

Институт цифровой экономики и информационных технологий

Кафедра Управления информационными системами и программирования

**КУРСОВАЯ РАБОТА**

по дисциплине «Технологии и методы программирования»

на тему «Разработка приложения “Аквариум”»

Выполнил(а)

студент(ка) группы 291Д-07ИБ/16

очной формы обучения

Направления:

Информационная безопасность

Зайцев Никита Владимирович

Научный руководитель:

Внукова Тамара Васильевна

|  |  |
| --- | --- |
| Дата: |  |
| Оценка: |  |
| Подпись: |  |

Москва

2019

**Содержание**

[Введение 3](#_Toc534820718)

[1 Проектирование приложения «Аквариум» 6](#_Toc534820719)

[1.1 Модель интерфейса 6](#_Toc534820720)

[1.2 Модель приложения 7](#_Toc534820721)

[2 Разработка приложения «Аквариум» 8](#_Toc534820722)

[2.1 Выбор языка и среды разработки для создания приложения 8](#_Toc534820724)

[2.2 Выбор графического интерфейса 9](#_Toc534820725)

[3 Реализация приложения «Аквариум» 10](#_Toc534820726)

[3.1 Графический интерфейс 10](#_Toc534820728)

[3.2 Общая схема работы приложения 11](#_Toc534820729)

[3.3 Создание окна и основных структур 13](#_Toc534820730)

[3.4 Двойной буфер и тред анимации 15](#_Toc534820731)

[3.5 Выполнение Технических Заданий 16](#_Toc534820732)

[3.6 Дополнительный функционал 17](#_Toc534820733)

[Вывод 19](#_Toc534820734)

[Приложения 20](#_Toc534820735)

# Введение

Компьютерные приложения развлекательного или научно-познавательного характера играют важную роль в современном обществе. Их использование – популярный способ времяпровождения, а их разработка – надёжный источник заработка. Данная работа описывает создание подобного приложения: аквариума с рыбами, симулирующего экосистему.

Целью данной работы является Разработка оконного приложения Windows Desktop Application, симулирующее аквариум с рыбами и обладающее пользовательским интерфейсом, дающим пользователю возможность управления аквариумом.

Чтобы достичь цели, необходимо выполнить следующие задачи:

1. Использовать ООП (Объектно-Ориентированное Программирование) и методы из библиотеки STL (Standard Template Library).

* Объектно-ориентированное программирование (ООП) — методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определённого класса, а классы образуют иерархию наследования[1] .
* Библиотека стандартных шаблонов (STL) (англ. Standard Template Library) — набор согласованных обобщённых алгоритмов, контейнеров, средств доступа к их содержимому и различных вспомогательных функций в C++ [2].

1. Определить объект TFish - аквариумная рыбка. Рыбка имеет координаты, скорость, размер, цвет, направление движения. Методами объекта являются:

* Init - устанавливает значения полей объекта и рисует рыбу на экране методом Draw.
* Draw - рисует рыбу в виде уголка, с острием в точке Coord и направленного острием по ходу движения рыбы.
* Look - проверяет несколько точек на линии движения рыбы. Если хоть одна из них отличается по цвету от воды, возвращается её цвет и расстояние до рыбы. Run - перемещает рыбу в текущем направлении на расстояние, зависящее от текущей скорости рыбы. Иногда случайным образом меняет направление движения рыбы. Если рыба видит препятствие, направление движения меняется, пока препятствие не исчезнет из поля зрения рыбы.

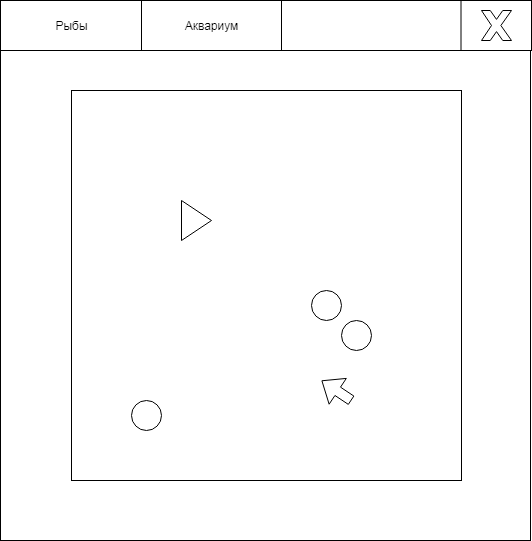
1. Определить объект Taquarium, который является местом обитания рыб. Он представляет собой область экрана, наполненную водой. Рыбы живут в аквариуме, поэтому экземпляры объекта Tfish должны быть полями объекта Taquarium. Методы:
   * Init - включает графический режим, заполняетаквариум водой, скалами и рыбами.
   * Run - организует бесконечный цикл, в котором выполняется метод Run всех обитателей аквариума.
   * Done - выключает графический режим.
2. Определить два объекта Tpike и Tkarp , которые наследуют объект Tfish. Оба они отличаются от Tfish тем, что по разному изображают себя на экране: Tpike - в виде зеленой стрелки, а Tkarp – в виде красного треугольника. Воспользуйтесь виртуальными методами. Для этого вернитесь к определению Tfish и откорректируйте его, сделав Draw пустым и виртуальным. Объедините карпов и щук в две стаи. Стая - это связанный список рыб в динамической памяти. Для связи добавить в объекты Tpike и Tkarp поле Next - указатель на следующую рыбу в стае. Сделайте аквариум не владельцем рыб, а двух стай и позвольте пользователю пополнять стаи, вводя рыб с клавиатуры.
3. Позволить щукам проявить свой дурной характер и поедать карпов, как только они их увидят. Здесь возникнет проблема - установить, какого именно карпа видит щука. Она решается путем просмотра всей стаи карпов и поиска того, чьи координаты близки к координатам данной щуки. Найденный карп удаляется из стаи.

Курсовая работа включает в себя ввeдeние, три глaвы, вывод, cпиcок иcпoльзoвaннoй литeрaтуры и прилoжeния. Пeрвaя глaвa пocвящeнa проектированию приложения и разработке его основных концепций. Вторая глава посвящена проблемам, возникшим в ходе разработки. Третья глава посвящена разбору разработанной программы. В зaключeнии пoдвoдятcя итoги прoвeдeннoй работы.

# Проектирование приложения «Аквариум»

## Модель интерфейса

Концептуальная модель интерфейса изображена на рисунке 1.

  
**Рис. 1 – концептуальная модель интерфейса программы**

Модели интерфейса были разработаны с помощью сервиса draw.io[[1]](#footnote-1).

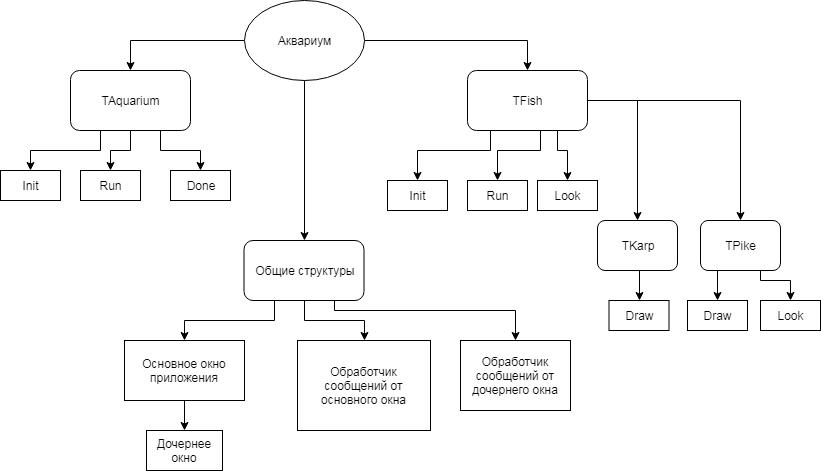
Рассмотрим пользовательский интерфейс.

Он включает в себя основное поле, где изображён аквариум с камнями (круги) и двигающимися рыбами (треугольник в соответствии с п.4 Технического Задания (далее ТЗ) обозначает карпа, стрелка – щуку). Над полем располагается выпадающее меню, дающее пользователю возможность работы с рыбами и аквариумом.

В подменю “Рыбы” можно вводить в аквариум новых рыб с указанием типа и размера, удалять рыб из аквариума по клику курсора и включать режим тестирования (“Дебага”) ИИ рыб, отображая лучи зрения рыбы, обычно не видимые пользователю. В подменю “Аквариум” можно взаимодействовать с аквариумом в целом: повторно рисовать аквариум с камнями, включать и останавливать анимацию, и выключать графический режим для выхода из приложения.

## Модель приложения

Общая концептуальная схема приложения, его элементов и методов на рисунке 2.



**Рис. 2 – концептуальная модель приложения**

# Разработка приложения «Аквариум»



## Выбор языка и среды разработки для создания приложения

При выполнении данной работы возникла проблема выбора среды разработки для создания приложения.

Популярные среды разработки:

* Visual Studio;
* Eclipse;
* CodeBeans;
* Code::Blocks;
* CodeLite.

Исследуя доступные варианты, я внезапно обнаружил, что на моём ПК уже установлена и запущена среда разработки «Microsoft Visual Studio 2017».

Выбранная среда разработки: Microsoft Visual Studio 2017

Выбранный язык программирования: С++

Основные преимущества данной среды разработки:

1. быстрота разработки приложения (RAD);
2. высокая производительность разработанного приложения;
3. низкие требования разработанного приложения к ресурсам ПК;
4. бесплатна для некоммерческого использования;
5. имею опыт работы в ней;
6. совместима с рядом языков.

Используемые компоненты и библиотеки:

* STL (Standard Template Library);
* GDIPlus.lib (Оболочка Windows GDI)[3].

При выборе ПО я руководствовался собственным опытом работы с указанными программами и языками, так как вышеперечисленное ПО, библиотеки и языки изучалось мною на семинарах в РЭУ имени Г.В. Плеханова и самостоятельно в целях саморазвития.

## Выбор графического интерфейса

Так как целевое приложение включает в себя элементы графики и анимации, для реализации мне потребовался дополнительный программный интерфейс - API.

API(программный интерфейс приложения, интерфейс прикладного программирования) (англ. application programming interface, API [эй-пи-ай]) — описание способов (набор классов, процедур, функций, структур или констант), которыми одна компьютерная программа может взаимодействовать с другой программой. [4]

В частности, мне требовался графический интерфейс для создания и изменения изображений на экране. В число известных графических интерфейсов можно включить:

* DirectX;
* OpenGL;
* Vulkan;
* Windows GDI.

Большинство перечисленных интерфейсов имеют предназначены для продвинутой 3D-графики, соответственно имея громадный объём и высокую сложность освоения. Как начинающий, я выбрал интерфейс Windows GDI для своей работы, так как его довольно ограниченный функционал соответствует простоте целевой программы, а также быстрее, легче изучается.

**GDI** (Graphics Device Interface, Graphical Device Interface) — один из трёх основных компонентов или «подсистем», вместе с ядром и Windows API, составляющих пользовательский интерфейс (оконный менеджер GDI) Microsoft Windows.

**GDI** — это интерфейс Windows для представления графических объектов и передачи их на устройства отображения, такие, как мониторы и принтеры.

**GDI** отвечает за отрисовку линий и кривых, отображение шрифтов и обработку палитры. [5]

# Реализация приложения «Аквариум»



## Графический интерфейс

При запуске приложения перед пользователем возникает окно приложения, где изображена область аквариума и выпадающее меню как пользовательский интерфейс (Приложение А).

При нажатии на кнопку “Рыбы” выпадает меню с опциями для добавления, удаления и проверки ИИ рыб:

Опция “Добавить” показывает окно добавления рыбы, где можно выбрать тип и размер новой рыбы. При нажатии на кнопку “ОК” рыба возникает в аквариуме в следующем кадре (Приложение Б).

Опция “Удалить” выводит информационную панель с инструкцией для пользователя: “клик” мышью по рыбе делает её красной и удаляет из аквариума в следующем кадре.

Опция “Дебаг” отображает лучи зрения рыб, по которым осуществляется навигация рыбы по аквариуму, уклонение от препятствий и охота щук на карпов (Приложение В).

Далее разберём меню “Аквариум”:

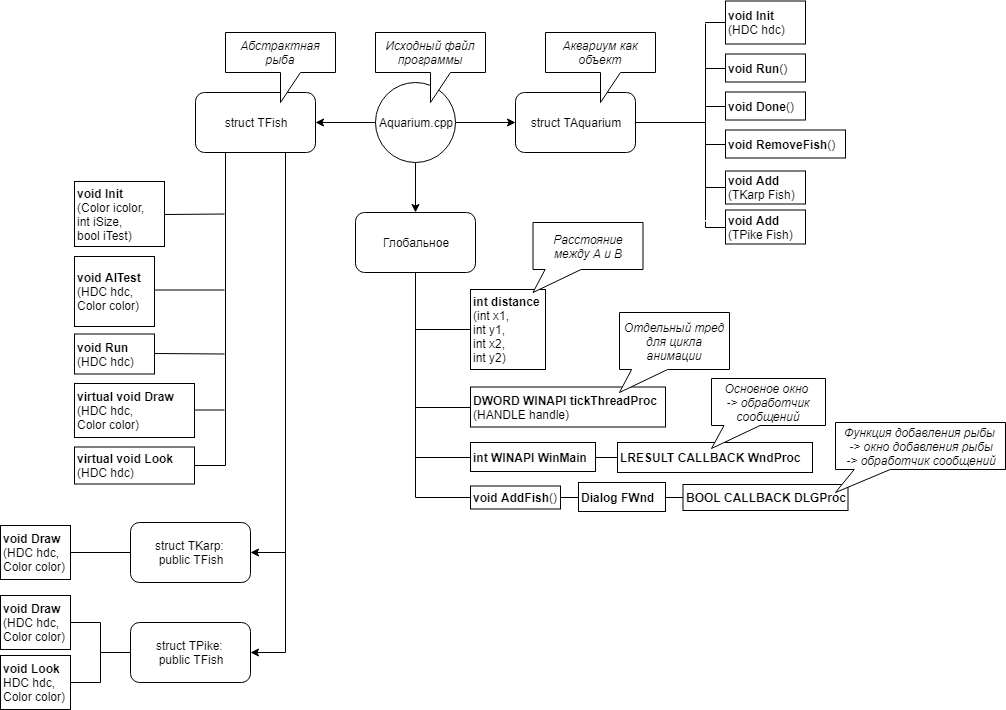
Кнопка “Переделать” пересоздаёт аквариум, случайно распределяя по нему новые камни. Рыбы не меняются.

Кнопка “Пауза” останавливает анимацию, давая пользователю возможность работы с неподвижными рыбами и изучению аквариума.

Кнопка “Выход” останавливает работу приложения.

## Общая схема работы приложения

Для разбора принципов работы программы, сперва рассмотрим и разберём её схему (Рисунок 3).

****  
  
**Рис. 3 – подробная схема программы, её компонентов и методов**

На схеме представлены структуры и основные методы программы с комментариями.

Исходный код (см. Приложение Ж) включает в себя три основных компонента:

1. **Глобальные методы**
   1. **distance** – Функция нахождения расстояния между двумя точками – полезна для множественных вычислений в векторной графике.
   2. **tickThreadProc** – “тред” (поток команд), где находится вечный цикл анимации. Создан, чтобы цикл анимации не грузил основное окно, тем самым блокируя передачу сообщений к нему от пользователя.
   3. **WinMain** – основное окно программы, представленное на рис.1 и 3. Содержит аквариум и меню. Имеет собственный обработчик сообщений WndProc.
   4. **AddFish** – функция добавления новой рыбы. Создаёт дополнительное диалоговое окно FWnd с параметрами ввода, имеющее собственный обработчик сообщений DLGProc.
2. **TAquarium** – структура аквариума.
   1. **Init** – рисует аквариум и случайно заполняет его водой и камнями. Доступна в меню.
   2. **Run** – рисует один кадр анимации внутри аквариума и стирает предыдущий. Иными словами, производит метод Run для всех рыб TFish внутри себя. Этот метод находится в вечном цикле внутри треда tickThreadProc.
   3. **Done** – останавливает анимацию. Доступна в меню.
   4. **RemoveFish** – функция удаления рыбы. Определяет цвет выделенного курсором пикселя в окне программы. Если это цвет рыбы, удаляет её. Доступна в меню.
   5. **Add** – добавляет новую рыбу в соответствующую стаю (Карпов либо Щук) в аквариуме и рисует её в случайном положении.
3. **TFish** – структура случайной рыбы. Доступна в меню.
   1. **Init –** задаёт начальные значения рыбы, в частности, размер. Остальные характеристики генерируются процедурно.
   2. **Look –** ИИ рыбы. Проверяет несколько точек на пути движения, уворачивается от всего, кроме воды. Виртуальная, так как у Щук ИИ отличается от других рыб (Карпов).
   3. **Draw –** Рисует рыбу на экране выбранным цветом. Виртуальная, чтобы рисовать разные виды рыб.
   4. **AITest –** изображает на экране лучи зрения рыбы из Look. Для тестирования и демонстрации поведения рыб.
   5. **Run –** цикл жизни (анимации) рыбы.
      1. Закрашивает рыбу в предыдущим кадре цветом воды методом Draw.
      2. Двигает рыбу, т.е. просчитывает её координаты в новом кадре и случайно поворачивает.
      3. Проводит анализ окружения с помощью Look и соответственно поворачивает ещё раз при необходимости.
      4. Наконец, рисует рыбу в новом кадре.

У структуры рыбы TFish есть два ответвления:

* **TKarp** – Карп. Имеет собственный метод Draw, рисуется в виде оранжевой стрелки.
* **TPike –** Щука. Имеет собственный метод Draw, рисуется в виде зелёной ёлочки. Имеет собственный метод Look, что позволяет ей не уворачиваться, а преследовать карпов и поедать их, при этом жирея и замедляясь.

## Создание окна и основных структур

Перейдём к более подробному разбору работы аквариума. Полный исходный код приведён в Приложении Ж.

Как и в любом оконном приложении для Windows, работа программы начинается с, ожидаемо, создания окна. Наше основное окно называется WinMain и имеет собственый класс уникальный класс. Окно имеет фиксированный размер 800х800 – так удобнее рассчитывать координаты для векторных рисунков. Чтобы запретить пользователю менять этот размер, мы напрочь убираем подобный функционал из стиля окна:

/\*Стиль окна = \*/ WS\_OVERLAPPEDWINDOW ^ WS\_THICKFRAME,

Прямо во время создания окна мы добавляем в него пользовательское меню через функцию AppendMenu (рис. 4, 8).

Создав окно, мы можем инициализировать интерфейс GDI и организовать цикл сообщений. Для их чтения и обработки создаём обработчик WndProc. И первое сообщение, которое улавливает обработчик – WM\_CREATE – “Окно создаётся”. При получении этого первого же сообщения мы без лишних слов хотим нарисовать (пока не существующий) аквариум перед пользователем, дабы оправдать название приложения. Для этого проведём следующие действия:

Создадим объект аквариума TAquarium и заранее инициализируем его. Аквариум в моём подсознании – это стеклянный прямоугольник, внутри которого вода, камни и рыбы. Начнём с простого и добавим в аквариум метод Init, который посредством GDI будет рисовать в окне наш стеклянный (синий) прямоугольник, заполненный водой (бледно-голубой заливкой). Камни станут овалами случайной формы и размера, хаотично разбросанными внутри стеклянного прямоугольника (вид сверху).

Рыбы на данный момент не существуют, создадим класс рыбы TFish.   
Рыба имеет следующие переменные:

* координаты носа (Если у рыб есть нос);
* размер, т.е. длина туши в пикселях от носа до хвоста, от 15 до 50;
* скорость, равная 50/размер + 1 и за счёт этого ненулевая;
* направление;
* цвет.

Для начала все эти параметры нужно задать. Именно это делается в методе TFish::Init. Затем рыбу нужно нарисовать, что посредством инструментов GDI и геометрических формул делаем в методе Draw. Затем рыбу нужно двигать. Что означает двигать рыбу? Это классическая двухмерная анимация. Рисуем кадр, стираем его (то есть рисуем повторно, но цветом фона), и рисуем следующий.

Чтобы рассчитать положение рыбы в следующем кадре нужен метод Look, который выпускает перед рыбой три “луча”, анализирующих цвета. Если лучи обнаруживают только воду, рыба плывёт вперёд. Если нет, рыба поворачивает от препятствия.

И, конечно же, просчитав коодинаты рыбы в кадре, его нужно нарисовать методом Run. Всё это делается последовательно в бесконечном цикле с перерывами (Sleep) в 1000/30 миллисекунд.

## Двойной буфер[[2]](#footnote-2) и тред[[3]](#footnote-3) анимации

“Холстом” для рисования кадров служит контекст устройства DC. Здесь возникает первая проблема: “холст” один. Когда перед зрителем быстро рисуется и стирается картинка, она кажется мерцающей. Чтобы решить проблему, создаём второй, временный контекст устройства. В цикле анимации мы копируем изображение (bitmap) из первого, основного DC во второй, проводим все операции рисования там же, во втором DC, и возвращаем полученный результат на первый DC. Так как пользователь видит только основной DC, изображение больше не мерцает.

Мы получили рыбу со всеми методами, которая может “плавать” без мерцания и уворачиваться от камней. Теперь вернёмся к аквариуму.

Рыб необходимо хранить в стае, или списке std::list, принадлежащем аквариуму. Цикл анимации двигает всю стаю методом Aquarium::Run. Для добавления новых рыб в список делаем метод TAquarium::Add, который добавляет в список новую рыбу и помещает её в случайную область аквариума. Для удобства пользователя Add будет осуществляться посредством отдельного окна, созданного по специальному шаблону в файле ресурса Aquarium.rc. Всё, что нужно – “попросить” основное окно через сообщение создать дочернее.

Здесь возникает вторая проблема: мы не можем отправлять сообщения основному окну. Бесконечный цикл внутри окна означает, что программа либо рисует следующий кадр, либо спит. Для решения этой проблемы мы создаём новый процесс tickThreadProc, не зависящий от окна, и помещаем туда цикл. Процесс привязывается к основному окну и запускается прямо при создании окна, т.е. при сообщении WM\_CREATE. Теперь у нас есть процесс с окном, обрабатывающий сообщения, и процесс с анимацией, обрабтывающий, соответственно, анимацию. Проблема №2 решена.

## Выполнение Технических Заданий

Получив “рабочий” “аквариум”, можно свериться с поставленным во введении списком задач. Требуется два вида рыб – карпы и щуки, поедающие карпов, поэтому делаем частные вариации объекта TFish: TPike и TKarp – щука и карп. Чтобы дать щукам и карпам уникальные внешность и искуственный интеллект, делаем методы Draw и Look в родительском классе TFish виртуальными, и добавляем их в дочерние классы TPike и TKarp. Частные методы Draw обеспечивают уникальную внешность рыб, а модифицированный метод Look у щук (TPike) заставляет их не уклоняться от карпов, а наоборот, преследовать их. Оригинальный метод Draw сохраняется, т.к. соответствует п.1 Технического Задания (далее ТЗ) – рыба как уголок. Список рыб в аквариуме разделяется на два, Karps и Pikes.

По ТЗ, Щуки должны есть карпов. Для этого (опять по ТЗ) нужно перебрать список карпов и удалить ближайшего к щуке. Список карпов есть объект аквариума (ТЗ). Следовательно, для работы со списком нужен указатель на аквариум. Здесь проблема: мы не можем хранить указатель на аквариум в объекте щуки - ведь щука сама есть принадлежность аквариума (ТЗ), а ребёнок не может создать собственного родителя без перемещений во времени. Чтобы обойти проблему, мы даём щуке сигнал "Хочу есть" (Boolean Kill), а её родитель каждый кадр ищет этот сигнал и “кормит” щуку.

Чтобы непосредственно “убить” карпа, мы перебираем весь список карпов и ищем в нём ближайшего к щуке, после чего убираем его из списка и, следовательно, анимации.

ТЗ создаёт очередное препятствие: для перебора должен использоваться итератор Next – указатель на следующую рыбу. Полагаю, автор ТЗ хотел увидеть подобную структуру:

for (std::list<TKarp>::iterator Next = Karps.begin(); Next != Karps.end(); ++Next) {}

Однако C++17 позволяет перебирать списки намного компактнее:

for (auto& Next : Karps) {}

Здесь в роли итератора выступает автоматически подобранная переменная. Я использовал в своём коде именно последний способ, потому что технически это всё-таки итерация.

Следующий пункт ТЗ: в аквариуме должна быть кнопка выключения графического режима. Для этого в меню добавляю кнопку выхода из приложения. Графический режим выключается при закрытии окна. Вставьте мем Roll Safe.

## Дополнительный функционал

В ходе разработки для собственного удобства или из интереса было введено несколько полезных, но не требуемых заданием функций приложения:

* Возможность тестирования ИИ рыб.   
  Рыбам был добавлен метод AITest, рисующий их (по умолчанию не видимое пользователю) поле зрения, и переключатель test, значение которого пользователь может менять через меню для выполения метода AITest в цикле анимации.
* Возможность удаления рыб.   
  Рыба удаляется при выполнении трёх условий:
  + Курсор в области окна
  + Курсор наведён на рыбу (определяется по цвету)
  + Нажата левая кнопка мыши

Эта функция может показать пользователю не интуитивной, поэтому в меню доступна подсказка (рис. 6).

* Возможность остановки анимации.  
  В меню доступен переключатель pause, который в значении 1 останавливает цикл в треде tickThreadProc.
* Динамическое изменение размера рыбы.  
  Когда щука ест карпа, её размер увеличивается пропорционально добыче, а скорость снижается.

# Вывод

Результатом моей работы является оконное приложение Windows, симулирующее аквариум с рыбами как экосистему. Благодаря использованию интерфейсов WinAPI и GDI+, встроенных в ОС Windows, приложение работает быстро и безотказно, а простой интерфейс знаком и легко понятен пользователю. Благодаря своим функциям и возможностям, “Аквариум” может применяться как в развлекательных целях, так и для изучения жизни существ в закрытой экосистеме. Программа имеет широкие перспективы улучшения и развития:

* добавление новых видов рыб и других существ с уникальными моделями поведения, например, улиток;
* расширение методов контроля над рыбами, например, перетаскивание рыб курсором;
* создание новых объектов аквариума, таких, как водросли или ракушки, и, наконец,
* улучшение качества графики и анимации, в частности, увелчиение частоты кадров, сглаживание анимации, использование текстур (изображений) для рыб, камней и дна аквариума.

Таким образом, поставленная задача была выполнена, а полученный результат работы расширен и улучшен за пределами требований технического задания.

# Приложения

Список источников

[1] ООП – Википедия, свободная энциклопедия <https://ru.wikipedia.org/wiki/Объектно-ориентированное_программирование>

[2] STL – Википедия, свободная энциклопедия  
<https://ru.wikipedia.org/wiki/Стандартная_библиотека_шаблонов>

[3] (переведено с англ.) Графический интерфейс GDI+ - Центр разработки для Windows  
<https://docs.microsoft.com/en-us/windows/desktop/gdiplus/-gdiplus-gdi-start>

[4] Программные интерфейсы – Википедия, свободная энциклопедия  
<https://ru.wikipedia.org/wiki/API>

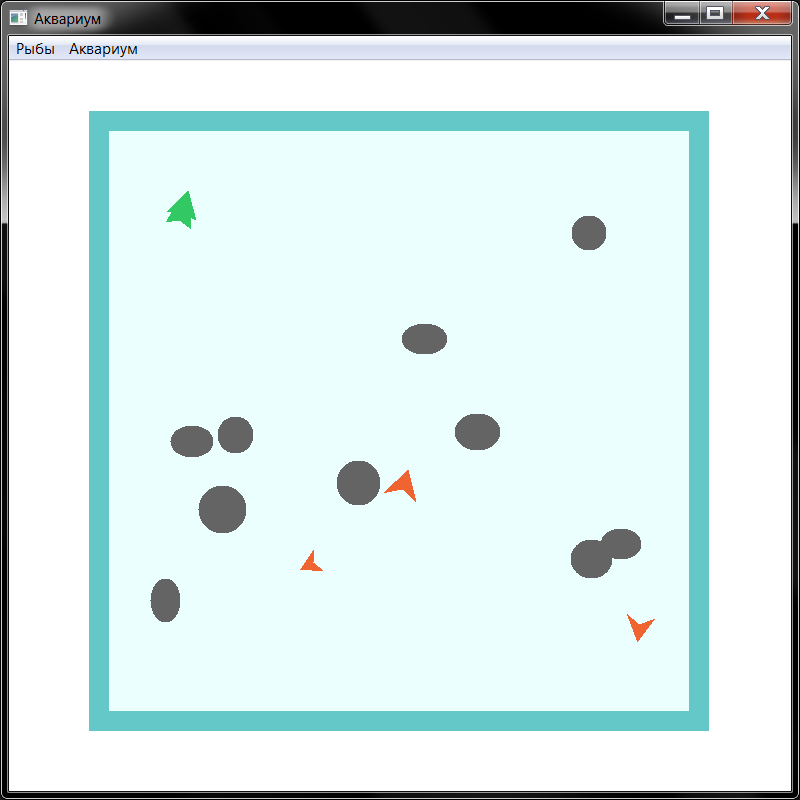
[5] GDI (определение) – Википедия, свободная энциклопедия  
<https://ru.wikipedia.org/wiki/GDI>

Список литературы и интернет-ресурсов

1. Википедия – свободная энциклопедия  
   <https://ru.wikipedia.org/>
2. Microsoft Docs – хранилище документации Microsoft  
   <https://docs.microsoft.com/ru-ru/>
3. Stack Overflow – система вопросов и ответов о программировании  
   <https://stackoverflow.com/>
4. Исттом Ч. C++ Programming Fundamentals – свободно распространяемая, 2003
5. Страуструп Б. Программирование. Принципы и практика с использованием C++ – Вильямс, 2016
6. Лафоре Р. Объектно-ориентированное программирование в C++ – Питер, 2007

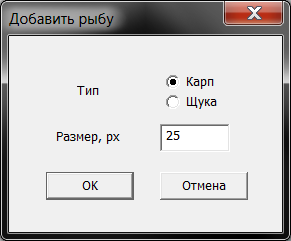
Приложение А

Основное окно программы



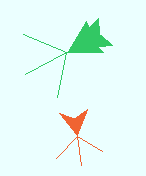
Приложение Б

Диалоговое окно добавления новой рыбы



Приложение В

Лучи зрения рыб



Приложение Г

Исходный код программы

#include "stdafx.h"

#include "Aquarium.h"

#include <windows.h>

#include <objidl.h>

#include <gdiplus.h>

#include <ctime> //Для рандома

#include <math.h> //Для геометрических формул

#include <list>

#include <iterator>

using namespace Gdiplus;

using namespace std;

#pragma comment (lib,"Gdiplus.lib")

#define PI 3.14159265 //Для перевода, т.к. math.h предпочитает радианы

//Инициализуем ID для меню

#define ID\_Add 1 //Добавить рыбу

#define ID\_Remove 2 //Убрать рыбу

#define ID\_Pause 3 //Остановить анимацию

#define ID\_Exit 4 //Закрыть окно

#define ID\_Restart 5 //Регенерация аквариума

#define ID\_Debug 6 //Отобразить ИИ рыб

HANDLE hTickThread; //Указатель на анимационный тред

HWND hWnd; //Основное окно приложения

HWND nWnd; //Второе окно для ввода данных

HDC aquarium; //КУ с пустым аквариумом

bool pause = 0; //Пауза анимации

bool test = 0; //Тест ИИ

const int fps = 30; //Скорость анимации

Color KarpC = { 255,240,100,50 };

Color PikeC = { 255,50,200,100 };

int distance(int x1, int y1, int x2, int y2) { //Просто для удобства: находит расстояние между двумя точками.

int d;

d = sqrt(pow((x2 - x1), 2) + pow((y2 - y1), 2)); //d = sqrt((x2-x1)^2+(y2-y1)^2)

return d;

}

struct TFish { //Абстрактная рыба

private: //Частное: стартовые переменные - задаются в Init и не меняются; HDC - задаётся через метод; методы Draw, Look и AITest - используются рекурсивно в методе Draw

HDC hdc; //Контекст устройства

virtual void Look(HDC hdc) //Зрение рыбы - три луча из носа. Обзор 100 градусов, дальность - два размера рыбы. Определяют цвет и дистанцию (в пикселях i) до объекта.

{

for (int i = 1; i < Size; i++) //Дальность - случайное значение, благо у нас нет рыб, пролетающих две собственные длины за кадр.

{

COLORREF CRF\_Left = GetPixel(hdc, Coord[0] - i \* cos((Dir - 50) \* PI / 180), Coord[1] - i \* sin((Dir - 50) \* PI / 180)); //Левый луч зрения

if (CRF\_Left != 0x00ffffeb && CRF\_Left != TColor.ToCOLORREF()) { //При обзоре игнорируем цвет фона (белый) и собственный цвет.

Dir += 45; //Иначе поворачиваем.

break; //При препятствии сразу забиваем на остаток вычислений

}

COLORREF CRF\_Mid = GetPixel(hdc, Coord[0] - i \* cos(Dir \* PI / 180), Coord[1] - i \* sin(Dir \* PI / 180)); //Центральный луч

if (CRF\_Mid != 0x00ffffeb && CRF\_Mid != TColor.ToCOLORREF()) {

Dir += 180;

break;

}

COLORREF CRF\_Right = GetPixel(hdc, Coord[0] - i \* cos((Dir + 50) \* PI / 180), Coord[1] - i \* sin((Dir + 50)\* PI / 180)); //Правый луч (Мб левый, не проверял))))

if (CRF\_Right != 0x00ffffeb && CRF\_Right != TColor.ToCOLORREF()) {

Dir -= 45;

break;

}

}

}

void AITest(HDC hdc, Color color) { //Дебаг - рисуем область зрения рыбы (TFish::Look).

Graphics graphics(hdc);

Pen red(color, 1);

int x = (Coord[0] - cos((Dir)\* PI / 180)); //DrawLine не способен решить, работать ему с int или float, из-за чего останавливает компиляцию. Поэтому насильно делаем один из аргументов int.

graphics.DrawLine(&red, //Формулы прямых прямо из Look

x,

Coord[1] - sin((Dir)\* PI / 180),

Coord[0] - Size \* cos((Dir)\* PI / 180),

Coord[1] - Size \* sin((Dir)\* PI / 180));

graphics.DrawLine(&red,

x,

Coord[1] - sin((Dir)\* PI / 180),

Coord[0] - Size \* cos((Dir - 50)\* PI / 180),

Coord[1] - Size \* sin((Dir - 50)\* PI / 180));

graphics.DrawLine(&red,

x,

Coord[1] - sin((Dir)\* PI / 180),

Coord[0] - Size \* cos((Dir + 50)\* PI / 180),

Coord[1] - Size \* sin((Dir + 50)\* PI / 180));

}

public:

bool Test = 0; //Переключатель режима разработчика

int Coord[2]; //Координаты носа (Если у рыб есть нос?). 0 - х, 1 - у

int Dir; //Направление в градусах

int Size; //Размер в пикселях

int Speed; //Скорость в пикселях, >1

Color TColor; //ARGB

void Init( //Устанавливает значения рыбы.

Color iColor, //Мы не можем привязать цвет рыбы к типу рыбы, потому что он динамически меняется (на белый)

int iSize,

bool iTest //По той же причине у Test не статическое значение

) {

TColor = iColor;

Size = iSize;

Test = iTest;

Speed = 50 / Size+1;//Скорость примерно определяется размером

}

virtual void Draw(HDC hdc, Color color) //Рисует рыбу. Выбор цвета ручки нужен для стирания (а точнее, закрашивания) предыдущего кадра.

{

Graphics graphics(hdc);

Pen pen(color, Size / 5);

SolidBrush fill(color);

graphics.DrawLine(&pen, Coord[0], Coord[1], Coord[0] + Size \* cos((Dir + 30)\*PI / 180), Coord[1] + Size \* sin((Dir + 30)\*PI / 180)); //Туша из двух линий (уголка) и глаза по формуле точки

graphics.DrawLine(&pen, Coord[0], Coord[1], Coord[0] + Size \* cos((Dir - 30)\*PI / 180), Coord[1] + Size \* sin((Dir - 30)\*PI / 180)); //на окружности x=rcosf, y=rsinf; угол между линиями 60.

graphics.DrawEllipse(&pen, Coord[0] + Size / 2 \* cos((Dir)\*PI / 180), Coord[1] + Size / 2 \* sin((Dir)\*PI / 180), Size / 8, Size / 8); //Глаз выглядит так, будто рыба готова тебе втащить. Фича.

}

void Run(HDC hdc) //Двигает рыбу (+ ИИ и проч.)

{

Graphics graphics(hdc);

Color Water(255, 235, 255, 255);

Draw(hdc, Water); //Закрашиваем рыбу предыдущем кадре фоновым цветом

if (Test == 1) AITest(hdc, Water); //В тестовом режиме также закрашивает всё тестовое

Coord[0] -= Speed \* cos(Dir\*PI / 180); //Двигаем рыбу по направлению Dir со скоростью Speed.

Coord[1] -= Speed \* sin(Dir\*PI / 180);

if (rand() % 10 < 6) {

Dir += rand() % 31 - 15; //Двойной рандом: с вероятностью 60% поворачивает от 0 до 15 влево или вправо

}

Look(hdc);

Draw(hdc, TColor); //Рисуем новый кадр

if (Test == 1) AITest(hdc, TColor);

}

};

struct TKarp : public TFish { //Карп как тип рыбы.

bool operator ==(const TKarp& K) //Нам прийдётся сравнивать карпов (struct==struct), что C++ по умолчанию не может. Поэтому делаем свой оператор.

{

return Coord[0] == K.Coord[0] && Coord[1] == K.Coord[1]; //"Равными" считаются карпы с одинаковыми координатами. Не идеальное решение, но если два карпа чудом окажутся в одной точке перед щукой, они оба умрут.

}

void Draw(HDC hdc, Color color) { //У карпов свой стиль рисования, поэтому своя функция Draw

Graphics graphics(hdc);

SolidBrush fill(color);

Point K1(Coord[0], Coord[1]);

Point K2(Coord[0] + Size \* cos((Dir - 30)\*PI / 180), Coord[1] + Size \* sin((Dir - 30)\*PI / 180));

Point K3(Coord[0] + Size \* .6 \* cos((Dir)\*PI / 180), Coord[1] + Size \* .6 \* sin((Dir)\*PI / 180));

Point K4(Coord[0] + Size \* cos((Dir + 30)\*PI / 180), Coord[1] + Size \* sin((Dir + 30)\*PI / 180));

Point KPoints[] = { K1, K2, K3, K4 };

graphics.FillPolygon(&fill, KPoints, (sizeof(KPoints) / sizeof(\*KPoints)));

}

};

struct TPike : public TFish { //Щука как тип рыбы.

bool Kill; //Показывает, что щука кого-то убила в данном кадре.

/\*

Про Kill:

По ТЗ, щуки должны есть карпов. Для этого (опять по ТЗ) нужно перебрать список карпов и удалить ближайшего к щуке.

Список карпов есть объект аквариума (ТЗ). Следовательно, для работы со списком нужен указатель на аквариум.

Здесь проблема: мы не можем хранить указатель на аквариум в объекте щуки - ведь щука сама есть

принадлежность аквариума (ТЗ), а ребёнок не может создать собственного родителя без перемещений во времени.

Чтобы обойти проблему, мы даём щуке сигнал "Хочу есть", а её родитель каждый кадр ищет этот сигнал и кормит щуку.

\*/

bool operator ==(const TPike& K) //Нам прийдётся сравнивать щук (struct==struct), что C++ по умолчанию не может. Поэтому делаем свой оператор.

{

return Coord[0] == K.Coord[0] && Coord[1] == K.Coord[1]; //"Равными" считаются щуки с одинаковыми координатами.

}

void Draw(HDC hdc, Color color) { //У щук свой стиль рисования, поэтому своя функция Draw

Graphics graphics(hdc);

SolidBrush fill(color);

Point S1(Coord[0], Coord[1]);

Point S2(Coord[0] + Size \* .8 \* cos((Dir - 30)\*PI / 180), Coord[1] + Size \* .8 \* sin((Dir - 30)\*PI / 180));

Point S3(Coord[0] + Size \* .7 \* cos((Dir - 20)\*PI / 180), Coord[1] + Size \* .7 \* sin((Dir - 20)\*PI / 180));

Point S4(Coord[0] + Size \* cos((Dir - 20)\*PI / 180), Coord[1] + Size \* sin((Dir - 20)\*PI / 180));

Point S5(Coord[0] + Size \* .8 \* cos((Dir)\*PI / 180), Coord[1] + Size \* .8 \* sin((Dir)\*PI / 180));

Point S6(Coord[0] + Size \* cos((Dir + 20)\*PI / 180), Coord[1] + Size \* sin((Dir + 20)\*PI / 180));

Point S7(Coord[0] + Size \* .7 \* cos((Dir + 20)\*PI / 180), Coord[1] + Size \* .7 \* sin((Dir + 20)\*PI / 180));

Point S8(Coord[0] + Size \* .8 \* cos((Dir + 30)\*PI / 180), Coord[1] + Size \* .8 \* sin((Dir + 30)\*PI / 180));

Point SPoints[] = { S1, S2, S3, S4, S5, S6, S7, S8 };

graphics.FillPolygon(&fill, SPoints, (sizeof(SPoints) / sizeof(\*SPoints)));

}

void Look(HDC hdc) override //Зрение рыбы - три луча из носа. Обзор 100 градусов, дальность - два размера рыбы. Определяют цвет и дистанцию (в пикселях i) до объекта.

{

Kill = 0;

for (int i = 1; i < Size; i++) //Дальность - случайное значение, благо у нас нет рыб, пролетающих две собственные длины за кадр.

{

COLORREF CRF\_Left = GetPixel(hdc, Coord[0] - i \* cos((Dir - 50) \* PI / 180), Coord[1] - i \* sin((Dir - 50) \* PI / 180)); //Левый луч зрения

if (CRF\_Left == 0x003264f0) { //При виде карпа поворачиваем к нему

Dir -= 25;

}

else if (CRF\_Left != 0x00ffffeb && CRF\_Left != TColor.ToCOLORREF()) { //Если не карп, не фон и не другая щука, отворачиваемся

Dir += 45;

break; //При препятствии сразу забиваем на остаток вычислений

}

COLORREF CRF\_Mid = GetPixel(hdc, Coord[0] - i \* cos(Dir \* PI / 180), Coord[1] - i \* sin(Dir \* PI / 180)); //Центральный луч

if (CRF\_Mid == 0x003264f0) { //Подтверждаем убийство при виде карпа

Kill = 1;

}

else if (CRF\_Mid != 0x00ffffeb && CRF\_Mid != TColor.ToCOLORREF()) {

Dir += 180;

break;

}

COLORREF CRF\_Right = GetPixel(hdc, Coord[0] - i \* cos((Dir + 50) \* PI / 180), Coord[1] - i \* sin((Dir + 50)\* PI / 180)); //Правый луч (Мб левый, не проверял))))

if (CRF\_Right == 0x003264f0) { //При виде карпа поворачиваем к нему

Dir += 25;

}

else if (CRF\_Right != 0x00ffffeb && CRF\_Right != TColor.ToCOLORREF()) {

Dir -= 45;

break;

}

}

}

};

struct TAquarium { //Объект аквариума

private:

int x0 = 90; //Координаты аквариума: верхний левый угол + полстекла.

int y0 = 60; //Отступ сверху меньше, потому что часть окна занимает меню.

int aw = 600; //Размеры аквариума: ширина, высота.

int ah = 600;

HDC hdc;

public:

bool done = 0; //Переключатель: готов ли аквариум

std::list<TKarp> Karps; //Стая карпов

std::list<TPike> Pikes; //Стая щук

void Init(HDC Ahdc) { //Рисует аквариум и его содержимое

hdc = Ahdc;

Graphics graphics(hdc);

Pen Glass(Color(255, 100, 200, 200), 20); //Рассчётная толщина стекла, от края до края, 20 пикселей.

SolidBrush Water(Color(255, 235, 255, 255));

SolidBrush Rock(Color(255, 100, 100, 100));

graphics.FillRectangle(&Water, x0, y0, aw, ah); //Стенки аквариума

graphics.DrawRectangle(&Glass, x0, y0, aw, ah); //Стенки аквариума

for (int i = 0; i < 10; i++) { //Процедурно сгенерированные ландшафты!! Некст ген и пр.

int x1 = x0 + rand() % (aw - 70) + 10; //Чтобы не кидать камни в стекло, задаём для них область. Камни рисуются от верхнего левого угла.

int y1 = y0 + rand() % (ah - 70) + 10; //Удобная местность: от верхнего левого угла + стекло, до нижнего правого - (стекло + тощина камня).

graphics.FillEllipse(&Rock, x1, y1, rand() % 20 + 30, rand() % 20 + 30);

}

aquarium = CreateCompatibleDC(hdc);

HBITMAP initbmp = CreateCompatibleBitmap(hdc, 800, 800);

SelectObject(aquarium, initbmp);

BitBlt(aquarium, 0, 0, 800, 800, hdc, 0, 0, SRCCOPY); //Сохраняем пустой аквариум, чтобы случайно не стереть его позднее.

done = 1;

}

void Run() { //Анимация с тройным буфером: основной (видимый) DC; буферный DC, где проводится всё рисование; DC с аквариумом, чтобы не рисовать поверх него.

HDC mdc = CreateCompatibleDC(hdc); //Буферный DC (Memory DC) для операций за кадром

HBITMAP mbmp = CreateCompatibleBitmap(hdc, 800, 800); //Буферная bmp для MDC

SelectObject(mdc, mbmp);

BitBlt(mdc, 0, 0, 800, 800, hdc, 0, 0, SRCCOPY); //Делаем копию текущего кадра и отправляем её в MDC для обработки

BitBlt(mdc, 0, 0, 800, 800, aquarium, 0, 0, SRCCOPY); //Восстанавливаем аквариум на случай, если предыдущая операция его повредила.

for (auto& i : Karps) { //Двигаем всех карпов, найденных в аквариуме.

i.Run(mdc);

}

for (auto& i : Pikes) { //Двигаем всех щук, найденных в аквариуме.

i.Run(mdc); //Интересный факт: если двигать щук перед карпами, ИИ не работает.

if (i.Kill == 1) { //Если щука намерена убить, проверяем всех карпов. Ближайший карп к щуке-убийце должен быть удалён.

TKarp Dead = Karps.front(); //Целевой карп как переменная, по умолчанию первый карп в списке.

int dist = distance(i.Coord[0], i.Coord[1], Dead.Coord[0], Dead.Coord[1]); //Расстояние до цели. По умолчанию - до первого карпа в списке.

for (auto& k : Karps) {

if (distance(i.Coord[0], i.Coord[1], k.Coord[0], k.Coord[1]) < dist)

{

dist = distance(i.Coord[0], i.Coord[1], k.Coord[0], k.Coord[1]); //Ищем наименьшее расстояние

Dead = k;

}

}

i.Size += (Dead.Size / 5); //При питании щука жиреет пропорционально блюду

i.Speed = 50 / i.Size + 1;

Karps.remove(Dead); //Убираем из стаи карпа, равного целевому

}

}

BitBlt(hdc, 0, 0, 800, 800, mdc, 0, 0, SRCCOPY); //Переносим картинку из MDC в основной HDC.

DeleteObject(mbmp); //Очищаем буфер

DeleteDC(mdc);

}

void Add(TKarp Fish) { //Метод добавления рыбы в аквариум, с вариацией для карпов и щук

Fish.Coord[0] = x0 + rand() % (aw - 20 - Fish.Size \* 2) + 20 + Fish.Size; //Кидаем её куда попало. Отличие от генерации камней в том, что

Fish.Coord[1] = y0 + rand() % (ah - 20 - Fish.Size \* 2) + 20 + Fish.Size; //рыба рисуется от "центра", поэтому отступ размером с рыбу делаем со всех сторон.

Fish.Dir = rand() % 360;

Karps.push\_back(Fish);

}

void Add(TPike Fish) {

Fish.Coord[0] = x0 + rand() % (aw - 20 - Fish.Size \* 2) + 20 + Fish.Size;

Fish.Coord[1] = y0 + rand() % (ah - 20 - Fish.Size \* 2) + 20 + Fish.Size;

Fish.Dir = rand() % 360;

Pikes.push\_back(Fish);

}

void Done() { //Метод паузы анимации.

done = !done;

}

void RemoveFish() {

POINT p; //Хранит координаты курсора

if (GetCursorPos(&p)) { //Записывает координаты курсора

if (ScreenToClient(hWnd, &p)) //Переводит коорды в локальные на окне

{

if (GetPixel(hdc, p.x, p.y) == KarpC.ToCOLORREF()) { //Проверяем цвет рыбы. Если карп, копипаста кода пожирания карпов

TKarp Dead = Karps.front(); //Целевая рыба как переменная, по умолчанию первая в списке.

int dist = distance(p.x, p.y, Dead.Coord[0], Dead.Coord[1]); //Расстояние до цели. По умолчанию - до первой рыбы в списке.

for (auto& k : Karps) {

if (distance(p.x, p.y, k.Coord[0], k.Coord[1]) < dist)

{

dist = distance(p.x, p.y, k.Coord[0], k.Coord[1]); //Ищем наименьшее расстояние

Dead = k;

}

}

Dead.Draw(hdc,{ 255,255,0,0 }); //Окрашивает в красный рыбу перед удалением

Karps.remove(Dead); //Убираем из стаи рыбу, равную целевой

}

else if (GetPixel(hdc, p.x, p.y) == PikeC.ToCOLORREF()) { //То же для щук

TPike Dead = Pikes.front(); //Целевая рыба как переменная, по умолчанию первая в списке.

int dist = distance(p.x, p.y, Dead.Coord[0], Dead.Coord[1]); //Расстояние до цели. По умолчанию - до первой рыбы в списке.

for (auto& k : Pikes) {

if (distance(p.x, p.y, k.Coord[0], k.Coord[1]) < dist)

{

dist = distance(p.x, p.y, k.Coord[0], k.Coord[1]); //Ищем наименьшее расстояние

Dead = k;

}

}

Dead.Draw(hdc, { 255,255,0,0 }); //Окрашивает в красный рыбу перед удалением

Pikes.remove(Dead); //Убираем из стаи рыбу, равную целевой

}

}

}

}

};

TAquarium Aquarium; //Тот самый аквариум по умолчанию

TKarp Karp; //Заранее инициализируем объекты карпа и щуки. Позднее, все

TPike Pike; //рыбы в аквариуме будут базироваться на этих двух объектах.

DWORD WINAPI tickThreadProc(HANDLE handle) {

Sleep(500); //Даём основному треду прогрузиться

HDC hdc = GetDC(hWnd); //Получаем DC основного треда

for (;; ) { //Вечный цикл анимации

if (Aquarium.done == 0) Aquarium.Init(hdc); //Рисуем аквариум, если не нарисован

if (pause != 1)Aquarium.Run(); //Анимируем рыб, если не пауза

Sleep(1000/fps); //Соблюдаем частоту кадров

}

DeleteDC(hdc);

}

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM); //Обработчик сообщений от окна

INT WINAPI WinMain(HINSTANCE hInstance, HINSTANCE, PSTR, INT iCmdShow)

{

MSG msg;

WNDCLASS wndClass;

GdiplusStartupInput gdiplusStartupInput;

ULONG\_PTR gdiplusToken;

GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, NULL); // Инициализуем GDI+.

wndClass.style = CS\_HREDRAW | CS\_VREDRAW;

wndClass.lpfnWndProc = WndProc;

wndClass.cbClsExtra = 0;

wndClass.cbWndExtra = 0;

wndClass.hInstance = hInstance;

wndClass.hIcon = LoadIcon(NULL, IDI\_APPLICATION);

wndClass.hCursor = LoadCursor(NULL, IDC\_ARROW);

wndClass.hbrBackground = (HBRUSH)GetStockObject(WHITE\_BRUSH);

wndClass.lpszMenuName = NULL;

wndClass.lpszClassName = TEXT("Aquarium");

RegisterClass(&wndClass);

hWnd = CreateWindow( //Главное окно программы

TEXT("Aquarium"),

TEXT("Аквариум"),

WS\_OVERLAPPEDWINDOW ^ WS\_THICKFRAME, //Убираем стиль THICKFRAME, чтобы лишить юзера возможности менять размер окна.

CW\_USEDEFAULT,

CW\_USEDEFAULT,

800,

800,

NULL,

NULL,

hInstance,

NULL);

ShowWindow(hWnd, iCmdShow);

UpdateWindow(hWnd);

//Меню для главного окна

HMENU hMenubar = CreateMenu(); //Основная панель меню

HMENU hFishes = CreateMenu(); //Раздел меню "Добавить/Убрать рыбу, Дебаг"

HMENU hAquarium = CreateMenu(); //Раздел меню "Перезапуск, Пауза, Выход"

AppendMenu(hMenubar, MF\_POPUP, (UINT\_PTR)hFishes, L"Рыбы"); //Строки с L, чтобы перевести char\* в LPCWSTR

AppendMenu(hMenubar, MF\_POPUP, (UINT\_PTR)hAquarium, L"Аквариум");

AppendMenu(hFishes, MF\_STRING, ID\_Add, L"Добавить...");

AppendMenu(hFishes, MF\_STRING, ID\_Remove, L"Удалить...");

AppendMenu(hFishes, MF\_STRING, ID\_Debug, L"Дебаг");

AppendMenu(hAquarium, MF\_STRING, ID\_Restart, L"Переделать");

AppendMenu(hAquarium, MF\_STRING, ID\_Pause, L"Пауза");

AppendMenu(hAquarium, MF\_STRING, ID\_Exit, L"Выход");

SetMenu(hWnd, hMenubar);

while (GetMessage(&msg, NULL, 0, 0))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

GdiplusShutdown(gdiplusToken);

return msg.wParam;

}

HWND FWnd = NULL; //Дочернее окно приложения (Добавить рыбу).

BOOL CALLBACK DLGProc(HWND, UINT, WPARAM, LPARAM); //Обработчик сообщений от дочернего окна

void AddFish() //Создаёт окно добавления новой рыбы

{

FWnd = CreateDialog(

GetModuleHandle(NULL), //Возвращают текущую hinstance.

MAKEINTRESOURCE(IDD\_ADDFISH), //Окно делается по специально созданному ресурсу ADDFISH из файла Aquarium.rc

hWnd,

DLGProc

);

CheckRadioButton(FWnd, 699, 1007, 699); //Выбирает тип рыбы Карп по умолчанию (699 - кнопка Карп)

SendDlgItemMessage(FWnd, //Ограничиваем ввод до двух символов

IDC\_FSIZE,

EM\_LIMITTEXT,

(WPARAM)2,

(LPARAM)0);

ShowWindow(FWnd, SW\_SHOW);

}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, //Обработчик сообщений от основного окна

WPARAM wParam, LPARAM lParam)

{

srand(time(NULL)); //Делаем рандом рандомным

switch (message)

{

case WM\_CREATE: { //При создании окна отдельный тред для анимации.

hTickThread = CreateThread(NULL, NULL, &tickThreadProc, NULL, NULL, NULL);

break; }

case WM\_COMMAND: { //Читаем пользовательские команды из меню

if (LOWORD(wParam) == ID\_Exit) { //Закрывает окно.

exit(0);

}

if (LOWORD(wParam) == ID\_Remove) { //Удаляет рыб

MessageBox(hWnd, //Для начала даёт инструкцию пользователю

L"Кликните по рыбе, чтобы удалить",

L"Инфо",

MB\_OK | MB\_ICONINFORMATION);

break;

}

if (LOWORD(wParam) == ID\_Add) { //Добавляет рыбу в аквариум

AddFish();

}

if (LOWORD(wParam) == ID\_Restart) { //Пересоздаёт аквариум (рыбы не меняются).

Aquarium.Done();

}

if (LOWORD(wParam) == ID\_Pause) { //Останавливает анимацию.

pause = !pause;

}

if (LOWORD(wParam) == ID\_Debug) { //Дебаг ИИ.

test = !test;

for (auto& i : Aquarium.Karps)

i.Test = test;

for (auto& i : Aquarium.Pikes)

i.Test = test;

}

break; }

case WM\_LBUTTONDOWN:{ //Читает нажатия мыши для удаления рыб

Aquarium.RemoveFish();

break; }

case WM\_DESTROY:

PostQuitMessage(0);

return 0;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

}

} // WndProc

BOOL CALLBACK DLGProc(HWND hwndDlg, UINT message, WPARAM wParam, LPARAM lParam) //Обработчик сообщений от дочернего окна

{

TCHAR FishSize[2]; //Буфер, в котором хранится размер рыбы, передаваемый дочерним окном

switch (message)

{

case WM\_COMMAND:

switch (LOWORD(wParam))

{

case IDOK:

SendDlgItemMessage(FWnd, //Отправляем дочернему окну сообщение с просьбой передать содержимое строки "Размер"

IDC\_FSIZE,

EM\_GETLINE,

(WPARAM)0,

(LPARAM)FishSize); //Полученный ответ помещаем в буфер "FishSize", который позже переводим в int через \_ttoi

if (\_ttoi(FishSize) < 15 || \_ttoi(FishSize) > 50) //При некорректном размере просим переделать

{

MessageBox(FWnd,

L"Введите размер от 15 до 50 px",

L"Ошибка",

MB\_OK);

break;

}

if (IsDlgButtonChecked(FWnd, IDC\_KARP) == BST\_CHECKED) {//Если карп (699) выбран, добавляем карпа. Иначе щуку. Одна из опций выбрана по умолчанию

Karp.Init(

KarpC,

\_ttoi(FishSize),

test

);

Aquarium.Add(Karp);

EndDialog(hwndDlg, wParam);

return TRUE;

}

else {

Pike.Init(

PikeC,

\_ttoi(FishSize),

test

);

Aquarium.Add(Pike);

EndDialog(hwndDlg, wParam);

return TRUE;

}

case IDCANCEL:

EndDialog(hwndDlg, wParam);

return TRUE;

}

}

return FALSE; } //DLGProc

1. Draw.io — это бесплатный сервис, предназначенный для формирования диаграмм и схем. [↑](#footnote-ref-1)
2. Буфер в программировании – область памяти, используемая для временного хранения данных ввода-вывода [↑](#footnote-ref-2)
3. Тред (Thread) в программировании – поток команд [↑](#footnote-ref-3)