­Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Компьютерные системы и сети (КСиС)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту на тему

Игровое программное средство «Target Strike»

БГУИР КП 1-40 01 01 010 ПЗ

Выполнил

студент: гр. 151004 Глушаченко Н.С.

Проверил: Красковский П.Н.

Минск 2023

содержание

[Введение 3](#_Toc135409883)

[1 Аналитический обзор литературы 4](#_Toc135409884)

[1.1 Обзор существующих аналогов 4](#_Toc135409885)

[1.2 Перечень функциональных требований 7](#_Toc135409886)

[2 Моделирование предметной области 8](#_Toc135409887)

[2.1 Архитектура «клиент-сервер» 8](#_Toc135409888)

[2.2 Протоколы TCP и UDP 9](#_Toc135409889)

[2.3 Отрисовка 10](#_Toc135409890)

[2.4 Представление клиента на сервере 10](#_Toc135409891)

[3 Проектирование программного средства 11](#_Toc135409892)

[3.1 Структура программы 11](#_Toc135409893)

[3.2 Проектирование интерфейса программного средства 12](#_Toc135409894)

[3.3 Проектирование функционала программного средства 15](#_Toc135409895)

[4 Разработка программного средства 23](#_Toc135409896)

[4.1 Проверка попадания 23](#_Toc135409897)

[4.2 Обработка нажатий кнопок 25](#_Toc135409898)

[4.3 Логирование 27](#_Toc135409899)

[5 Тестирование программного средства 29](#_Toc135409900)

[5.1 Запуск, подключение, соединение 29](#_Toc135409901)

[5.2 Игровой процесс 31](#_Toc135409902)

[5.3 Конец игры 32](#_Toc135409903)

[6 Руководство пользователя 33](#_Toc135409904)

[6.1 Минимальные системные требования 33](#_Toc135409905)

[6.2 Главное меню 33](#_Toc135409906)

[6.3 Игровой процесс 34](#_Toc135409907)

[6.4 Настройки 35](#_Toc135409908)

[Заключение 37](#_Toc135409909)

[Список использованной литературы 38](#_Toc135409910)

[Приложение А 39](#_Toc135409911)

[Приложение Б 45](#_Toc135409912)

[Приложение В 48](#_Toc135409913)

# Введение

Целью данного курсового проекта по предмету является разработка игрового программного средства под названием "Target Strike". "Target Strike" – это игровое приложение, предназначенное для тренировки меткости и реакции игрока. В ходе игры необходимо привести прицел к мишени, при этом при перемещении координаты прицела сохраняются в файл. Оно создано для предоставления пользователю увлекательного и интерактивного опыта стрельбы по двухмерным мишеням.

Суть игры заключается в следующем: игрок настроивает параметры мишени и прицела. При начале игры на игровом поле в случайном месте повлется мишень. Используя стрелки на клавиатуре игрок должен привети прицел в мишень. Чем ближе центр прицела к центру мишени, тем больше очков получает игрок. При этом, все движения мишени записываются в виде в виде координат и в конце игры отрисовывается полный путь игрока. Загружаемые данные представляют собой координаты левого верхнего угла мишени и время ее присутствия на игровом поле. Таким образом, игроки получат возможность самостоятельно создавать сценарии и соревноваться между собой. Передавать сценарии можно по сети.

Для игры по сети используются специальные программы и протоколы, которые обеспечивают обмен данными между серверами и клиентами.

Одним из таких протоколов является TCP (Transmission Control Protocol), который гарантирует доставку и правильность передачи данных по сети. TCP работает на транспортном уровне модели OSI и использует порты для установления соединения между отправителем и получателем данных.

Другим протоколом является UDP, который не гарнтирует доставку и правильность передачи данных по сети, но работает быстрее и может быть применен для больших объемов данных. Соединение при этом не устанавливается, что облегчает отправку данных игроку с разных устройств.

Целью данной курсовой работы является разработка игрового приложения «Target Strike», представляющего собой реализацию TCP и UDP клиентов, позволяющих нескольким пользователю получить данные с других устройств, а не из файла на текущем устройстве:

В результате выполнения курсовой работы будет создано приложение, позволяющее сыграть в шахматы по сети, используя сокеты для обмена данными.

# Аналитический обзор литературы

## Обзор существующих аналогов

### AimTrainer.io

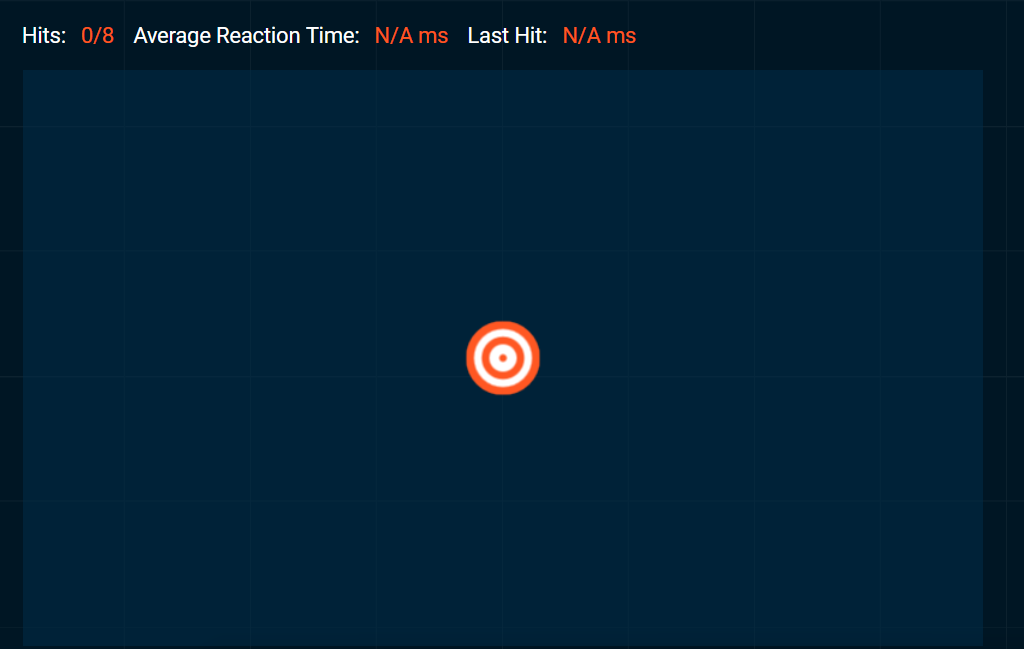


Рисунок 1.1 – AimTrainer.io

Список выполняемых функций:

* расчет точности;
* определение попадания;
* определение части мишени, в которую произошло попадание.

Достоинства:

* случайная генерация мишеней;
* управление курсором.

Недостатки:

* невозможность задать координаты и время жизни мишени;
* невозможность получить путь курсора по экрану;
* невозможность получить координаты мишени по сети;
* мишень не исчезает с течением времени;

### Aimbooster

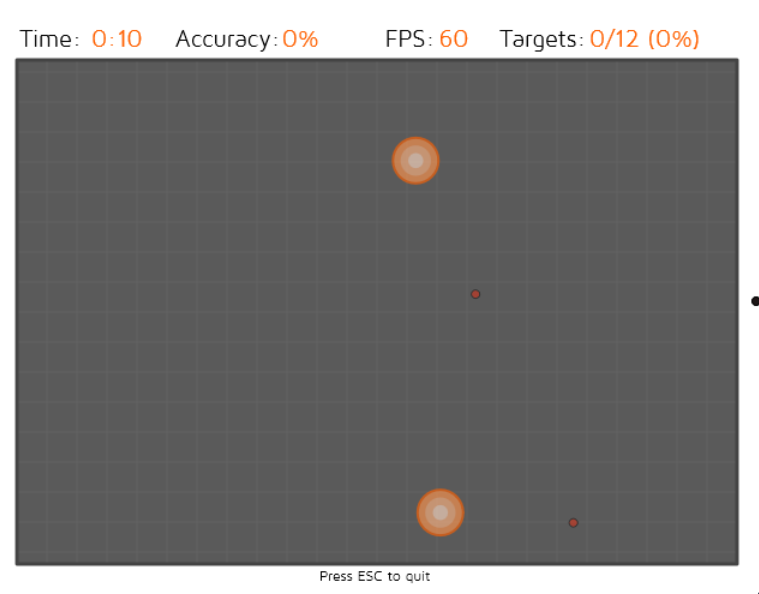


Рисунок 1.2 – Aimbooster

Список выполняемых функций:

* расчет точности;
* определение попадания;
* определение части мишени, в которую произошло попадание.

Достоинства:

* случайная генерация мишеней;
* управление курсором.

Недостатки:

* невозможность задать координаты мишени;
* невозможность получить путь курсора по экрану;
* невозможность получить координаты мишени по сети.

### Duck Hunt



Рисунок 1.3 – Шахматы «ChessPartner»

Список выполняемых функций:

* определение попадания.

Достоинства:

* случайная генерация мишеней.

Недостатки:

* невозможность задать координаты мишени;
* невозможность получить путь курсора по экрану;
* невозможность получить координаты мишени по сети.

## Перечень функциональных требований

После анализа аналогов был составлен следующий список выполняемых функций:

* приложение, которое позволяет сыграть в игру с генерацией мишеней из файла или при их получении по сети;
* поддержка UDP и TCP;
* логирование перемещений прицела по экрану;
* восстановление пути прицела по экрану по завершении игры;
* анализ места на мишени, куда произошло попадание;
* вычисление игрового счета;
* возможность настройки мишени, прицела, способа получения координат мишени и ее времени жизни, выбор скорости перемещения прицела, частоты обновления экрана, частоты выстрелов по мишени;
* игра прекращается после завершения потока данных(конец файла или конец передачи данных по сети).

Для разработки программного средства будет использоваться язык программирования C# и среда разработки Microsoft Visual Studio.

# Моделирование предметной области

## Архитектура «клиент-сервер»

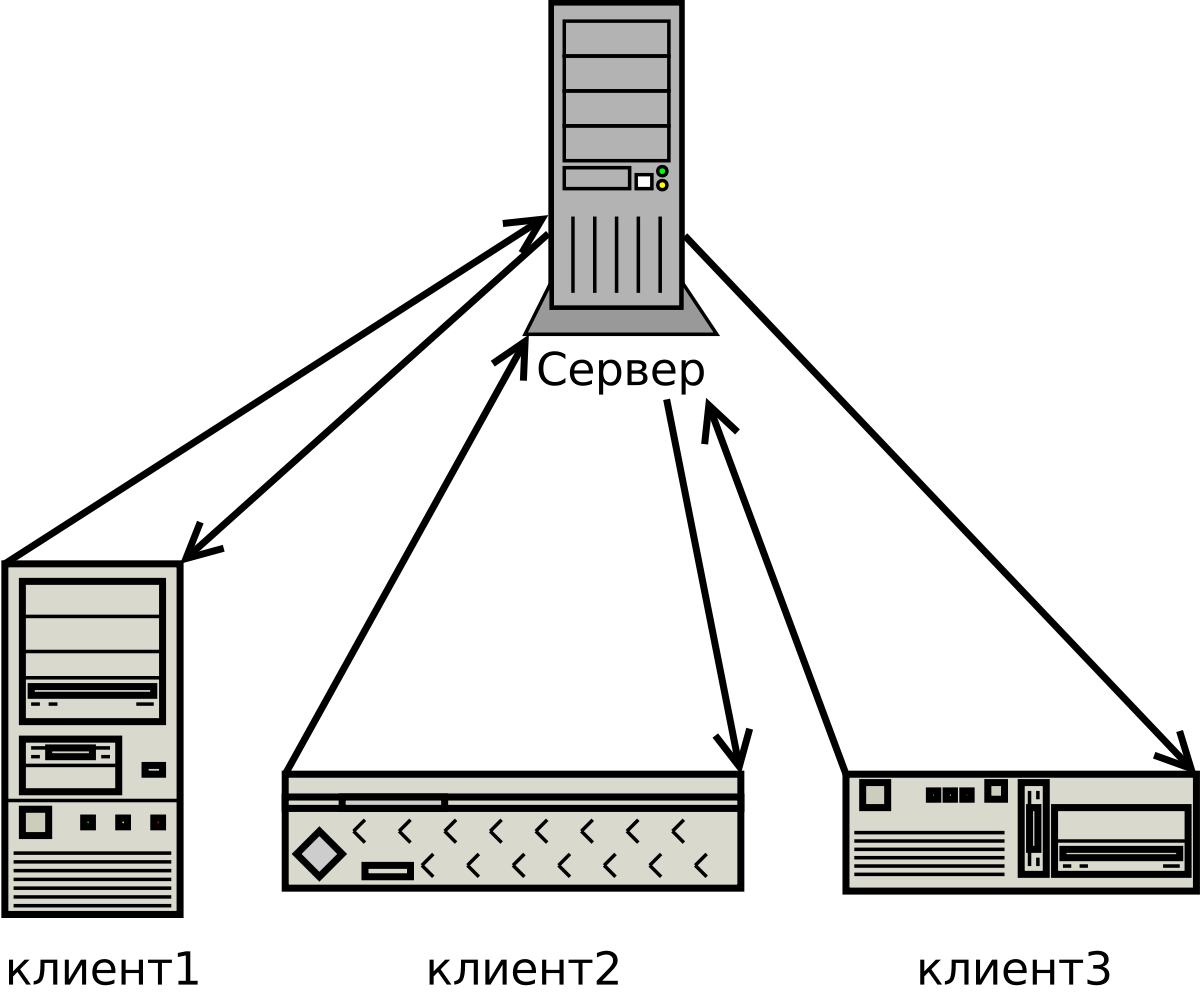


Рисунок 2.1 – Архитектура «клиент-сервер»

Архитектура «клиент-сервер» – это вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг, называемыми серверами, и заказчиками услуг, называемыми клиентами. Серверы предоставляют различные функции или сервисы клиентам, которые инициируют запросы на такие сервисы посредством сетевых протоколов. Клиенты могут быть веб-браузерами, мобильными приложениями или другими программами, которые используют ресурсы или данные, предоставляемые серверами. Например, когда вы заходите на веб-сайт, ваш браузер отправляет запрос на веб-сервер, который отвечает на этот запрос, отправляя вам данные. Архитектура «клиент-сервер» имеет ряд преимуществ:

* серверы могут обслуживать множество клиентов одновременно;
* серверы могут хранить и обрабатывать данные в одном месте, что упрощает управление и обеспечение безопасности данных;
* сервер может обеспечивать взаимосвязь между клиентами.

Однако архитектура «клиент-сервер» также имеет некоторые недостатки:

* клиенты не могут работать без подключения к серверам и зависят от их доступности и производительности;
* серверы могут быть перегружены запросами от большого числа клиентов и стать узким местом в системе;
* серверы могут быть атакованы злоумышленниками, которые пытаются получить доступ к данным или нарушить работу системы.

В данном проекте пользователь будет запускать сервер при начале игры. На сервер будут поступать координаты мишени и время ее жизни. При этом, когда время жизни последней мишени истечет, будет отрисован весь путь прицела. Для TCP подключения сервер при закрытии формы будет отправлять координаты пути клиенту обратно.

## Протоколы TCP и UDP

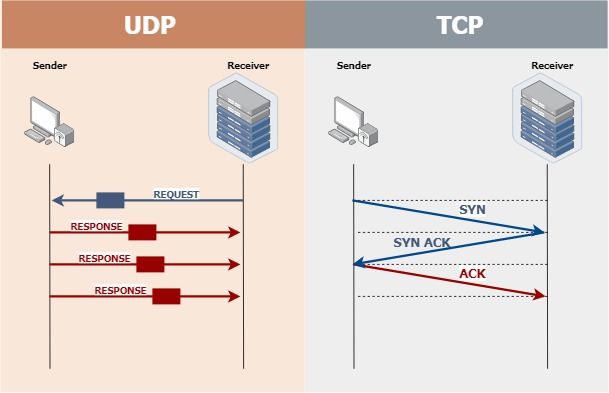


Рисунок 2.2 – Протоколы TCP и UDP

В играх чаще применяется UDP, так как он обеспечивает более высокую скорость и меньшую задержку передачи данных, чем TCP. UDP не гарантирует доставку и порядок пакетов, но это не критично для реального времени, где важнее актуальность информации. TCP же может приводить к замедлению и сбоям в игре из-за повторной отправки потерянных пакетов и автоматического контроля пропускной способности. Однако UDP требует дополнительной работы по реализации некоторых функций TCP, таких как надежность, упорядоченность и шифрование данных. Но поскольку шахматы – пошаговая игра, не требующая высокой актуальности данных, то протокол TCP будет более целесообразен.

В случае с TCP порт клиента по умолчанию – любой свободный (не менее 2000, не более 65535), работает на транспортном уровне модели OSI, что обеспечивает надежность и целостность передаваемой информации.

Протокол TCP предусматривает работу в соединительном режиме, т.е. в режиме “установка соединения – передача данных – разрыв соединения”. Является протоколом точка-точка, что означает, что взаимодействие происходит между двумя конечными устройствами.

Данный протокол хорошо подходит для передачи информации, доставка которой требует подтверждения получения и контроля ошибок.

## Отрисовка

Для отрисовки игры хорошим вариантом будет использование компонента Graphics – компонента System.Drawing. Рисование будет выполняться на форме. Графический интерфейс будет реализован с использованием WinForms.

## Представление клиента на сервере

Клиент – это не просто абстрактный сокет, который подключается к серверу. Клиент также имеет свои атрибуты и своё поведение: он может находиться в ожидании игры, уже иметь определённый клиент в качестве пары и так далее.

Поэтому сервер не работает напрямую с сокетами, а создаёт для каждого из них специальный объект-представление, в котором хранится сам сокет и другая сопутствующая информация о состоянии клиента. Сервер взаимодействует с этими объектами, чтобы управлять подключениями и передачей данных между клиентами.

# Проектирование программного средства

## Структура программы

При разработке приложения будут использованы следующие модули:

1. Figures – содержит классы и интерфейсы для фигур.

* IFigure – интерфейс для всех фигур;
* Figure – абстрактный класс, реализующий общие для всех фигур функции;
* Line – линия. Используется при отрисовке пути прицела по игровому полю;
* Ellipse – эллипс. Одно и представлений прицела и мишени;
* Rect – прямоугольник. Одно из представлений прицела и мишени.

1. Elements – игровой прицел и мишень:

* Cursor – игровой прицел. Может быть представлен фигурами, реализующими интерфейс IFigure;
* Target – игровая мишень. Представляет собой массив фигур. Каждая из фигур реализует поле мишени.

1. Forms – графический интерфейс. Все окна приложения:

* MainMenu – главное меню;
* PaintForm – игровое поле.
* SettingsForm – настройки приложения;

1. Network – сетевое взаимодействие:

* TCPReceiver – получение данных по TPC;
* UDPReceiver – получение даных по UDP.

## Проектирование интерфейса программного средства

### Главное меню

Главное меню должно предоставлять возможность перейти в настройки игры начать игру.



Рисунок 3.1 – Главное меню

### Настройки

Окно настроек должно позволить настроить основные параметры игры.



Рисунок 3.2 – Настройки

### Игровое поле

Окно игры должно представлять из себя область, содержащую внутри себя игровое поле, а также информационный блок о счете.



Рисунок 3.3 – Игровое поле

## Проектирование функционала программного средства

В игре должны быть использованы следующие методы:

* обработка подключения клиентов по TCP;
* отправка данных клиентам по TCP;
* обработка подключения клиентов по UDP;
* загрузка данных из файла;
* завершение игры.

### Ожидание подключения клиентов по TCP

Ожидание подключений выполняется в отдельном потоке, чтобы не блокировать пользовательский интерфейс. Все подлючившиеся пользователи знасятся в список. При закрытии игровой формы всем клиентам высылается путь прицела за все время игры.



Рисунок 3.4 – Ожидание подключения клиентов

### Отправка данных клиентам по TCP

Отправка происходит при закрытии пользователем игрового окна. Каждый пользователь, который не отключился, получит полный путь прицела за все время игры.



Рисунок 3.5 – Отправка данных клиентам по TCP

### Обработка подключения клиентов по UDP

Ожидание подключений выполняется в отдельном потоке, чтобы не блокировать пользовательский интерфейс



Рисунок 3.6 – Обработка подключения клиентов по UDP

### Загрузка данных из файла

Загрузка данных происходит, если пользователь не выбрал взаимодействие по сети. Ему также необходимо будет выбрать файл с координатами. В методе производится проверка корректности получаемых данных. Данными считается группа значений: координата X, координата Y, время жизни.



Рисунок 3.7 – Загрузка данных из файла. Часть 1



Рисунок .8 – Загрузка данных из файла. Часть 2

### Завершение игры

Вызывается, когда заканчиваются данные. Прекращает рисование мишени, рисует путь прицела на основе лога.



Рисунок 3.9 – Завершение игры. Часть 1



Рисунок .0 – Завершение игры. Часть 2

# Разработка программного средства

## Проверка попадания

### Проверка попадания в эллипс

public override bool Contains(Point point)

{

double center\_x = (p1.X + p2.X) / 2.0;

double center\_y = (p1.Y + p2.Y) / 2.0;

double a = (p2.X - p1.X) / 2.0;

double b = (p2.Y - p1.Y) / 2.0;

double distance = Math.Sqrt(Math.Pow((point.X - center\_x) / a, 2) + Math.Pow((point.Y - center\_y) / b, 2));

return distance <= 1;

}

### Проверка попадания в прямоугольник

public override bool Contains(Point point)

{

if (point.X >= p1.X && point.X <= p2.X && point.Y >= p1.Y && point.Y <= p2.Y)

{

return true;

}

return false;

}

### Проверка попадания в мишень

public int Shoot(Point p)

{

for (int i = FIELDS - 1; i >= 0; i--)

{

if (figures[i].Contains(p))

{

return i + 1;

}

}

return 0;

}

## Обработка нажатий кнопок

Игрок может нажать на несколько кнопок сразу. Это может привести как к тому, что он остановится на месте, так и к тому, что он начнет двигаться по диагонали. Для обработка нажатия на несколько кнопок сразу были созданы логические переменные, которые изменяют свое значение при нажатии или отпускании кнопки.

### Нажатие кнопки

private void PaintForm\_KeyDown(object sender, KeyEventArgs e)

{

switch (e.KeyCode)

{

case Keys.Left:

leftPressed = true;

break;

case Keys.Right:

rightPressed = true;

break;

case Keys.Up:

upPressed = true;

break;

case Keys.Down:

downPressed = true;

break;

}

}

}

### Отпускание кнопки

private void PaintForm\_KeyUp(object sender, KeyEventArgs e)

{

switch (e.KeyCode)

{

case Keys.Left:

leftPressed = false;

break;

case Keys.Right:

rightPressed = false;

break;

case Keys.Up:

upPressed = false;

break;

case Keys.Down:

downPressed = false;

break;

}

}

## Логирование

Основной целью данной программы является логирование траектории движения пользователя. Запись происходит с частотой, задаваемой в настройкой. Вместе с логированием происходит и перемещение прицела по игровому полю.

private void Timer\_LOG\_Tick(object? sender, EventArgs e)

{

int deltaX = 0;

int deltaY = 0;

double speed = SPEED;

if ((downPressed ^ upPressed) && (rightPressed ^ leftPressed))

{

speed /= DIAGONAL\_CORRECTION;

}

if (leftPressed)

{

deltaX -= (int)Math.Round(speed);

}

if (rightPressed)

{

deltaX += (int)Math.Round(speed);

}

if (upPressed)

{

deltaY -= (int)Math.Round(speed);

}

if (downPressed)

{

deltaY += (int)Math.Round(speed);

}

var prev = new Point(rect.X, rect.Y);

if (rect.Left < 0)

{

rect.X = 0;

}

else if (rect.Right > SCREEN\_WIDTH)

{

rect.X = SCREEN\_WIDTH - rect.Width;

}

else

{

rect.X += deltaX;

}

if (rect.Top < 0)

{

rect.Y = 0;

}

else if (rect.Bottom > SCREEN\_HEIGHT)

{

rect.Y = SCREEN\_HEIGHT - rect.Height;

}

else

{

rect.Y += deltaY;

}

using var sw = new StreamWriter(LOG\_FILE\_PATH, true);

if (WRITE\_UNIQUE)

{

if (!(rect.X == prev.X && rect.Y == prev.Y))

{

sw.WriteLine((cursor.Center().X - SCREEN\_CENTER\_X) + " " + (cursor.Center().Y - SCREEN\_CENTER\_Y));

}

}

else

{

sw.WriteLine((cursor.Center().X - SCREEN\_CENTER\_X) + " " + (cursor.Center().Y - SCREEN\_CENTER\_Y)); ;

}

Invalidate();

}

# Тестирование программного средства

## Запуск, подключение, соединение

### Запуск

Исходный набор данных: запуск программы, начало игры.

Ожидаемый результат: открытие главного меню, после нажатия кнопки начала игры показ игровго поля и сокрытие главного меню.

Полученный результат:

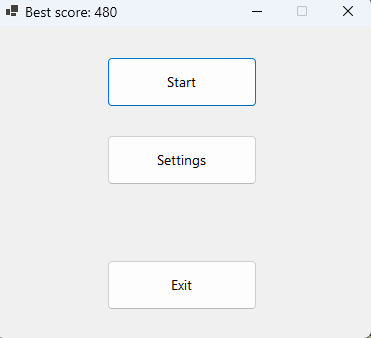


Рисунок 5.1 – Главное меню

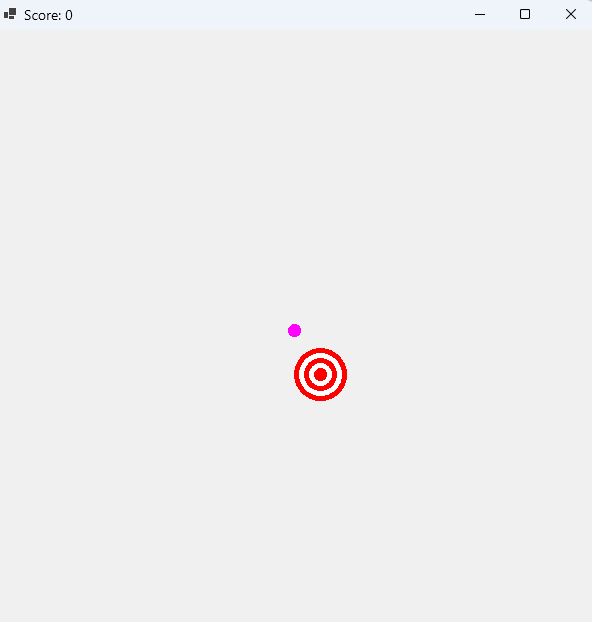


Рисунок .2 – Игровое поле

### Подключение по TCP

Исходный набор данных: запуск программы, нажатие кнопки начала игры, запуск TCP клиента.

Ожидаемый результат: установка соединения, получение данных.

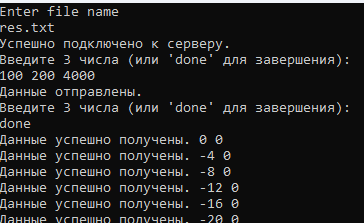


Рисунок .3 – Подключение по TCP

### Подключение по UDP

Исходный набор данных: запуск программы, нажатие кнопки начала игры, запуск TCP клиента.

Ожидаемый результат: отправка данных, получение подтверждения.

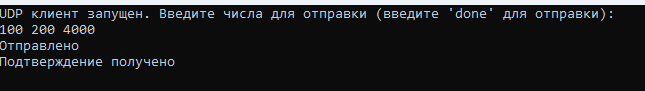


Рисунок 5.4 – Подключение по UDP

## Игровой процесс

Исходный набор данных: запрос на начало игры, установка первоначальных настроек.

Ожидаемый результат: в зависимости от полученных координат и времени жизни мишеней игрок увиди мишени в разных частях экрана. При наведении прицела на мишень, начнет увеличиваться счет.

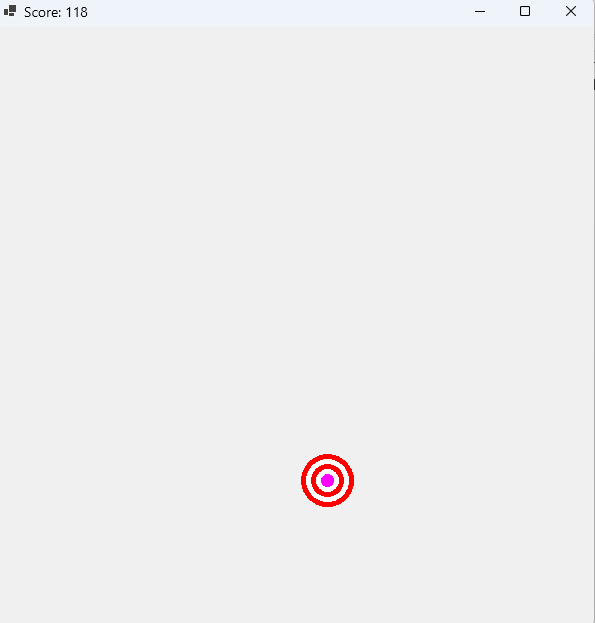


Рисунок 5.5 – Игровой процесс

## Конец игры

Исходный набор данных: данные для мишеней кончились, время жизни последней мишени кончилось

Ожидаемый результат: сообщение о завершении игры, вывод счета, отрисовка пути прицела в ходе игры.

Очевидно, что тест пройден успешно.

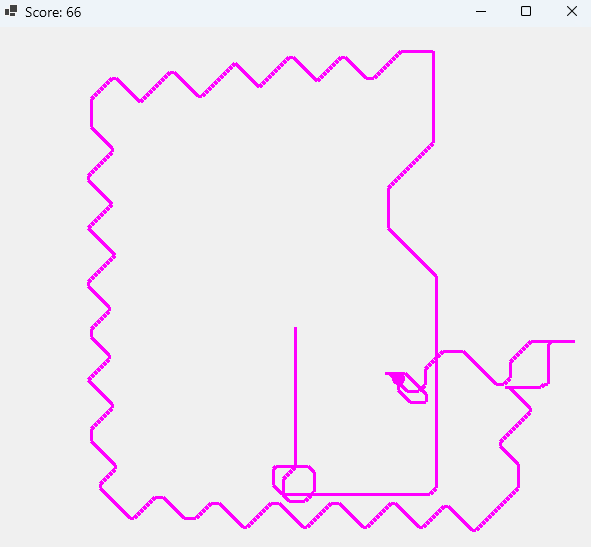


Рисунок 5.6 – Конец игры

# Руководство пользователя

## Минимальные системные требования

Для успешного запуска игрового приложения «Target Strike» и комфортной работы с ним необходимо соответствие минимальным системным требованиям:

* процессор Intel Pentium или лучше;
* оперативная память 1 GB или лучше;
* свободное место на диске 100 мегабайт или больше;
* операционная система Windows 10 или новее.

## Главное меню

Главное меню позволяет начать игру, открыть меню настроек и закрыть приложение. В левом верхнем углу вы можете видеть свой рекорд.

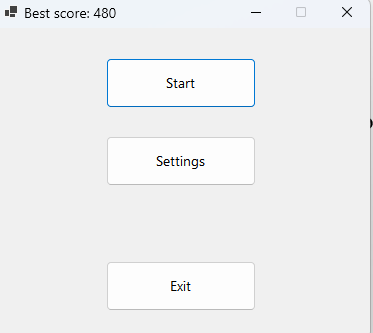


Рисунок 6.1 – Главное меню

## Игровой процесс

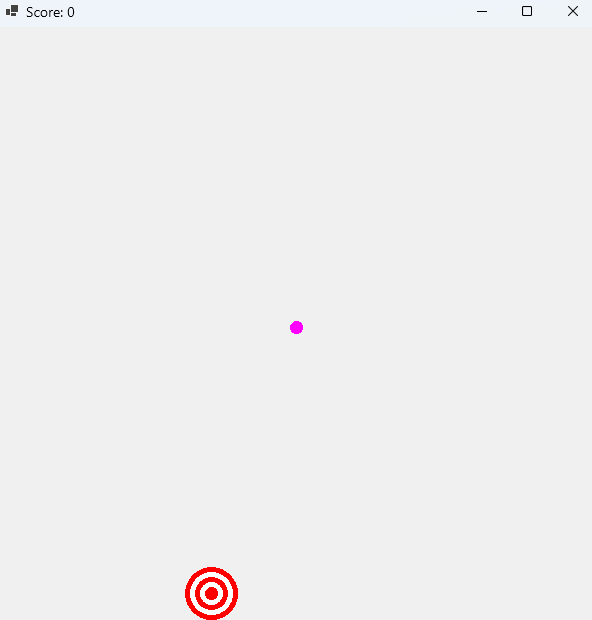


Рисунок 6.2 – Игровой процесс

После запуска игры перед пользователем предстанет игровая область.

Цель игры набрать максимальное количество очков. Для просмотра координат откройте файл, в который вы записываете свои ходы(можно посмотреть в настройках). Частоту стрельбы и обновления экрана можно установить в настройках. В случае, если будет выбрана загрузка данных по сети, первая мишень появится в случайном месте, в дальнейшем ее место будет основано на присланных координатах. Игра завершается, когда закончится время жизни последней мишени. При использовании TCP соединения для отправки координат клиентам необходимо закрыть игровую форму.

## Настройки

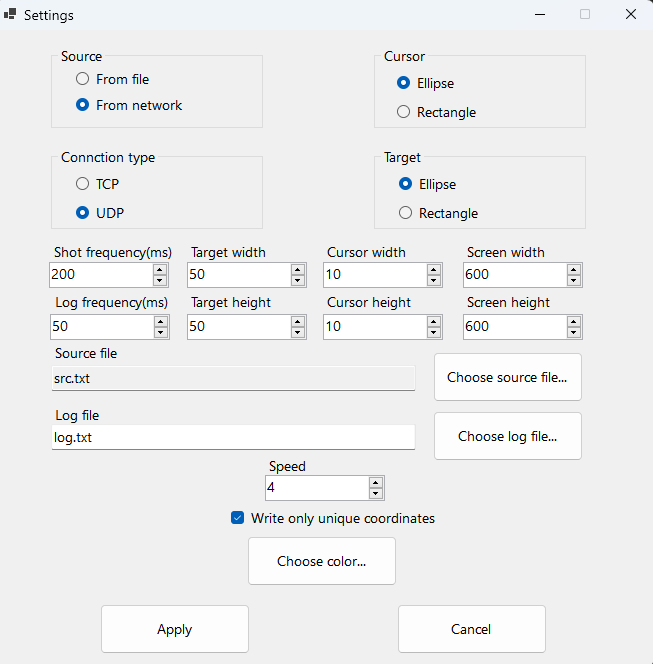


Рисунок .3 – Настройки

Программа предоставляет большое множество настроек.

Настройка источника данных: можно выбрать загрузки данных из файла или получение данных по одному из протоколов(TCP или UDP)

Выбор протокола: в случае выбора получания данных по сети, можно выбрать протокол, ко которому будут приходить данные. Протокол TCP также позволяет отправит всем подключенным клиентам результат перемещения прицела по игровому полю.

Выбор формы курсора: эллипс или прямоугольник.

Выбор размеров мишени: ширина и высота.

Выбор размера курсора: ширина и высота.

Выбор размера экрана: ширина и высота.

Выбор исходного файла: файл с координатами используется, если выбрано получени данных из файла. Формат данных: координата X, координата Y, время жизни мишени в миллисекундах. Каждая новая мишень должна быть описана с новой строки.

Выбор файла для логирования: можно выбрать существующий или указать имя нового файла.

Выбор скорости мишени: устанавливает максимальную скорость перемещения мишени по полю.

Выбор записи только уникальных координат: в случае, если прицел будет находиться на одном месе, координаты логироваться не будут.

Выбор цвета: откроется окно выбор цвета прицела.

Для применения настроек нажмите конопку Apply, для отмены внесенных изменений нажмите кнопку Cancel или закройте окно.

# Заключение

По итогу работы над курсовым проектом было разработано игровое приложение «Target Strike», обладающее графическим интерфейсом для взаимодействия с пользователем и позволяющее получать данные как по сети, так и из файла, используя как протокол TCP, так и протокол UDP.

Данное программное средство позволяет пользователю потренировать свою меткость и реакцию. Среди преимуществ программы можно отметить интуитивно понятный интерфейс, простоту реализации, современный дизайн, а также низкие системные требования.

При разработке данного средства были успешно выполнены следующие поставленные задачи:

* приложение, которое позволяет сыграть в игру с генерацией мишеней из файла или при их получении по сети;
* поддержка UDP и TCP;
* логирование перемещений прицела по экрану;
* восстановление пути прицела по экрану по завершении игры;
* анализ места на мишени, куда произошло попадание;
* вычисление игрового счета;
* возможность настройки мишени, прицела, способа получения координат мишени и ее времени жизни, выбор скорости перемещения прицела, частоты обновления экрана, частоты выстрелов по мишени;
* игра прекращается после завершения потока данных(конец файла или конец передачи данных по сети).

Приложение прошло все этапы тестирования и продемонстрировало корректную быструю работу.

Бесценным результатом курсового проектирования является полученный опыт работы с компьютерными системами и сетями, динамическим созданием компонентов, языком C#. Был получен опыт работы с графическим интерфейсом пользователя (изучение компонентов форм, свойств и методов объектов, событий), обобщены и применены все ранее полученные знания.

B перспективе данное программное средство может быть усовершенствовано (оптимизация кода, добавление новых функций и т.п.).

# Список использованной литературы

[1] Документация к языку C# [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/dotnet/csharp/ – Дата доступа: 18.05.2023.

[2] Полное руководство по языку программирования С# 11 и платформе .NET 7 [Электронный ресурс]. – Режим доступа: https://metanit.com/sharp/tutorial/. – Дата доступа: 18.05.2023.

[3] Протокол TCP[Электронный ресурс]. – Режим доступа: <https://metanit.com/sharp/net/3.4.php> – Дата доступа: 18.05.2023.

[4] Класс TCPClient[Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/dotnet/api/system.net.sockets.tcpclient?view=net-6.0> – Дата доступа: 18.05.2023

[5] Класс TCPListener[Электронный ресурс]. – Режим доступа: [https://learn.microsoft.com/ru-ru/dotnet/api/system.net.sockets.tcplistener?view= net-8.0](https://learn.microsoft.com/ru-ru/dotnet/api/system.net.sockets.tcplistener?view=%20net-8.0) – Дата доступа: 18.05.2023

[6] Полное руководство по языку программирования С# 11 и платформе .NET 7 [Электронный ресурс]. – Режим доступа: <https://metanit.com/sharp/tutorial/> – Дата доступа: 18.05.2023

[7] Сетевое программирование на C# [Электронный ресурс]. – Режим доступа: <https://professorweb.ru/my/csharp/web/level1/web_index.php> – Дата доступа: 18.05.2023

[8] Сетевое программирование в .NET[Электронный ресурс]. – Режим доступа: [https://learn.microsoft.com/ru-ru/dotnet/fundamentals/networking/ overview](https://learn.microsoft.com/ru-ru/dotnet/fundamentals/networking/%20overview) – Дата доступа: 18.05.2023

Приложение А

(обязательное)

Исходный код программы

namespace NewTracker.Figures

{

internal interface IFigure

{

public Point p1 { get; set; }

public Point p2 { get; set; }

public void Draw(Graphics g);

public bool Contains(Point p);

public Point Center();

}

}

using Newtonsoft.Json;

using System.Text.Json;

namespace NewTracker.Figures

{

[Serializable]

public class Figure : IFigure

{

[JsonProperty("figures")]

private readonly List<Figure> figures = new();

public int Length => figures.Count;

protected float Width => Math.Abs(p1.X - p2.X);

protected float Height => Math.Abs(p1.Y - p2.Y);

public Point p1 { get; set; }

public Point p2 { get; set; }

public Color Color;

protected Pen Pen;

private const int PEN\_WIDTH = 3;

public Figure() : this(0, 0, 0, 0) { }

public Figure(int x1, int y1, int x2, int y2)

{

Pen = new(Color.FromArgb(new Random().Next(int.MaxValue)))

{

Width = PEN\_WIDTH

};

Color = Pen.Color;

p1 = new Point(Math.Min(x1, x2), Math.Min(y1, y2));

p2 = new Point(Math.Max(x1, x2), Math.Max(y1, y2));

}

public Figure(Point p1, Point p2)

{

Pen = new(Color.FromArgb(new Random().Next(int.MaxValue)))

{

Width = PEN\_WIDTH

};

Color = Pen.Color;

this.p1 = new Point(Math.Min(p1.X, p2.X), Math.Min(p1.Y, p2.Y));

this.p2 = new Point(Math.Max(p1.X, p2.X), Math.Max(p1.Y, p2.Y));

}

public Figure(Point p1, Point p2, Color color) : this(p1, p2)

{

Pen = new(color)

{

Width = PEN\_WIDTH

};

Color = color;

}

public Figure this[int index]

{

get => figures[index];

set => figures[index] = value;

}

public virtual void Draw(Graphics graphics) { }

public virtual Point Center() => new();

public virtual bool Contains(Point point) => false;

public void Remove(Figure figure)

{

if (figures.Contains(figure))

{

figures.Remove(figure);

}

}

public void RemoveAll(Predicate<Figure> match)

{

figures.RemoveAll(match);

}

public virtual bool Equals<T>(T figure) where T : Figure

{

if (GetType() == figure?.GetType() && figure.p1 == p1 && figure.p2 == p2)

{

return true;

}

return false;

}

public int AddUnique(Figure? val)

{

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

{

if (val != null && val.Equals(figures[i]))

{

figures[i].Pen = new(Color.FromArgb(new Random().Next(int.MaxValue)))

{

Width = PEN\_WIDTH

};

figures[i].Color = figures[i].Pen.Color;

return i;

}

}

if (val != null)

{

figures.Add(val);

}

return Length - 1;

}

public string Count()

{

return $"Total figures: {Length}";

}

}

}

namespace NewTracker.Figures

{

[Serializable]

public class Line : Figure

{

public Line() : this(0, 0, 0, 0) { }

public Line(int x1, int y1, int x2, int y2) : base(x1, y1, x2, y2) { }

public Line(Point p1, Point p2) : base(p1, p2) { }

public Line(Point p1, Point p2, Color color) : base(p1, p2, color) { }

public override bool Contains(Point point)

{

int dx = Math.Abs(p2.X - p1.X);

int dy = Math.Abs(p2.Y - p1.Y);

double expectedY;

try

{

expectedY = p1.Y + dy / dx \* (point.X - p1.X);

}

catch

{

return true;

}

return Math.Abs(point.Y - expectedY) < 5;

}

public override Point Center()

{

return new Point(p1.X + (p2.X - p1.X) / 2, p1.Y + (p2.Y - p1.Y) / 2);

}

public override void Draw(Graphics graphics)

{

Pen.Color = Color;

graphics.DrawLine(Pen, p1, p2);

}

}

}

namespace NewTracker.Figures

{

[Serializable]

public class Rect : Line

{

protected Brush Brush => new SolidBrush(Color);

public Rect() : base() { }

public Rect(Point p1, int w, int h) : base()

{

this.p1 = p1;

p2 = new(p1.X + w, p2.Y + h);

}

public Rect(Point p1, Point p2) : base(p1, p2) { }

public Rect(Point p1, Point p2, Color color) : base(p1, p2, color) { }

public Rect(Rectangle rect, Color color)

{

p1 = new(rect.Left, rect.Top);

p2 = new(rect.Right, rect.Bottom);

Color = color;

}

public override bool Contains(Point point)

{

if (point.X >= p1.X && point.X <= p2.X && point.Y >= p1.Y && point.Y <= p2.Y)

{

return true;

}

return false;

}

public override void Draw(Graphics graphics)

{

Pen.Color = Color;

graphics.DrawRectangle(Pen, p1.X, p1.Y, Width, Height);

graphics.FillRectangle(Brush, p1.X, p1.Y, Width, Height);

}

}

}

namespace NewTracker.Figures

{

[Serializable]

public class Ellipse : Rect

{

public Ellipse() : base() { }

public Ellipse(Color color) : base() { Color = color; }

public Ellipse(Point p1, Point p2, Color color) : base(p1, p2, color) { }

public Ellipse(Point p1, Point p2) : base(p1, p2) { }

public Ellipse(Point p1, int x, int y) : base(p1, x, y) { }

public Ellipse(Rectangle rect, Color color) : base(rect, color) { }

public override bool Contains(Point point)

{

double center\_x = (p1.X + p2.X) / 2.0;

double center\_y = (p1.Y + p2.Y) / 2.0;

double a = (p2.X - p1.X) / 2.0;

double b = (p2.Y - p1.Y) / 2.0;

double distance = Math.Sqrt(Math.Pow((point.X - center\_x) / a, 2) + Math.Pow((point.Y - center\_y) / b, 2));

return distance <= 1;

}

public override void Draw(Graphics graphics)

{

Pen.Color = Color;

graphics.DrawEllipse(Pen, p1.X, p1.Y, Width, Height);

graphics.FillEllipse(Brush, p1.X, p1.Y, Width, Height);

}

}

}

Приложение Б

(обязательное)

Исходный код программы

using NewTracker.Figures;

namespace NewTracker.Elements

{

internal class Target

{

private const int FIELDS = 5;

private readonly IFigure[] figures = new IFigure[FIELDS];

private readonly Color[] colors = new Color[FIELDS] { Color.Red, Color.White, Color.Red, Color.White, Color.Red };

private readonly int dx, dy;

public Point p1

{

set

{

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

{

figures[i].p1 = new Point(value.X + dx / 2 \* i, value.Y + dy / 2 \* i);

}

}

}

public Point p2

{

set

{

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

{

figures[i].p2 = new Point(value.X - dx / 2 \* i, value.Y - dy / 2 \* i);

}

}

}

public Target(int width, int height)

{

dx = width / FIELDS;

dy = height / FIELDS;

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

{

figures[i] = new Ellipse(colors[i]);

}

}

public Target(Type TFigure, int width, int height) : this(TFigure, new Rectangle(0, 0, width, height)) { }

public Target(Type TFigure, Rectangle rect)

{

dx = rect.Width / FIELDS;

dy = rect.Height / FIELDS;

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

{

figures[i] = Activator.CreateInstance(TFigure, rect, colors[i]) as IFigure ?? new Ellipse(rect, colors[i]);

rect.X += dx / 2;

rect.Y += dy / 2;

rect.Width -= dx;

rect.Height -= dy;

}

}

public Target(Rectangle rect)

{

dx = rect.Width / FIELDS;

dy = rect.Height / FIELDS;

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

{

figures[i] = new Ellipse(rect, colors[i]);

rect.X += dx / 2;

rect.Y += dy / 2;

rect.Width -= dx;

rect.Height -= dy;

}

}

public Target(Point p1, Point p2) : this(new Rectangle(p1.X, p1.Y, p2.X - p1.X, p2.Y - p1.Y)) { }

public void Draw(Graphics g)

{

foreach (var figure in figures)

{

figure.Draw(g);

}

}

public int Shoot(Point p)

{

for (int i = FIELDS - 1; i >= 0; i--)

{

if (figures[i].Contains(p))

{

return i + 1;

}

}

return 0;

}

}

}

using NewTracker.Figures;

namespace NewTracker.Elements

{

internal static class MyCursor

{

public static IFigure CreateCursor()

{

return new Ellipse();

}

public static IFigure CreateCursor(Type TFigure, Rectangle rect, Color color)

{

return Activator.CreateInstance(TFigure, rect, color) as IFigure ?? new Ellipse(rect, color) as IFigure;

}

}

}

Приложение В

(обязательное)

Исходный код программы

using System.Net;

using System.Net.Sockets;

namespace NewTracker.Network

{

internal class TCPReceiver

{

private static List<int[]> arrays = new();

private static List<TcpClient> clients = new();

private static bool isRunning = false;

public static void Start()

{

int port = 12345;

var listener = new TcpListener(IPAddress.Any, port);

isRunning = true;

listener.Start();

while (isRunning)

{

if (listener.Pending())

{

TcpClient client = listener.AcceptTcpClient();

clients.Add(client);

var clientThread = new Thread(() => HandleClient(client));

clientThread.Start();

}

}

listener.Stop();

}

private static void HandleClient(TcpClient client)

{

try

{

NetworkStream stream = client.GetStream();

byte[] buffer = new byte[3 \* sizeof(int)];

while (isRunning)

{

if (stream.DataAvailable)

{

int bytesRead = stream.Read(buffer, 0, buffer.Length);

int[] numbers = new int[3];

Buffer.BlockCopy(buffer, 0, numbers, 0, buffer.Length);

arrays.Add(numbers);

}

}

}

catch

{

}

}

public static int[] GetNewTarget()

{

int[] res = Array.Empty<int>();

if (isRunning && arrays.Count != 0)

{

res = arrays[0];

arrays.RemoveAt(0);

return res;

}

return res;

}

public static void Stop(string logFile, int x\_center, int y\_center)

{

using var sr = new StreamReader(logFile);

string? str;

isRunning = false;

while ((str = sr.ReadLine()) != null)

{

string[] numbers = str.Split(' ');

int num1 = 0, num2 = 0;

if (numbers.Length == 2 && int.TryParse(numbers[0], out num1) && int.TryParse(numbers[1], out num2))

{

foreach (TcpClient client in clients)

{

try

{

NetworkStream stream = client.GetStream();

byte[] sendBytes = new byte[2 \* sizeof(int)];

Buffer.BlockCopy(BitConverter.GetBytes(num1), 0, sendBytes, 0, 4);

Buffer.BlockCopy(BitConverter.GetBytes(num2), 0, sendBytes, 4, 4);

stream.Write(sendBytes, 0, sendBytes.Length);

}

catch

{

}

}

}

}

foreach (TcpClient client in clients)

{

client.Close();

client.Dispose();

}

clients.Clear();

}

}

}

using System.Net;

using System.Net.Sockets;

using System.Text;

namespace NewTracker.Network

{

// remake

// Клиент отсылает результат всем пользователям

internal static class UDPReceiver

{

private static bool IsOn = false;

private static List<int[]> arrays = new();

public static int[] GetNewTarget()

{

int[] res = Array.Empty<int>();

if (IsOn && arrays.Count != 0)

{

res = arrays[0];

arrays.RemoveAt(0);

return res;

}

return res;

}

public static void Stop()

{

arrays.Clear();

IsOn = false;

}

public static void Start()

{

int port = 12345;

var udpServer = new UdpClient(port);

var clientEndpoint = new IPEndPoint(IPAddress.Any, port);

arrays = new();

IsOn = true;

try

{

while (IsOn)

{

if (udpServer.Available >= 3 \* sizeof(int))

{

byte[] receiveBytes = udpServer.Receive(ref clientEndpoint);

int[] numbers = new int[receiveBytes.Length / sizeof(int)];

Buffer.BlockCopy(receiveBytes, 0, numbers, 0, receiveBytes.Length);

int remainingNumbers = numbers.Length;

int currentIndex = 0;

while (remainingNumbers >= 3)

{

int[] array = new int[3];

Array.Copy(numbers, currentIndex, array, 0, 3);

arrays.Add(array);

currentIndex += 3;

remainingNumbers -= 3;

}

byte[] okBytes = Encoding.UTF8.GetBytes("OK");

udpServer.Send(okBytes, okBytes.Length, clientEndpoint);

}

}

}

catch (Exception e)

{

MessageBox.Show(e.Message);

}

finally

{

udpServer.Close();

}

}

}

}