В начале игры создаются два игровых поля – левое, для своих кораблей, и правое – для кораблей противника. На левом поле изначально расположены 10 кораблей, которые можно перемещать мышкой или поворачивать нажатием клавиши CTRL. Для начала игры пользователь вводит свой порт, IP-адрес и порт противника в соответствующие поля, затем нажимает кнопки «Установить связь» и «Начать игру».

Перед началом игры создаются все визуальные компоненты в конструкторе MainForm (строка 47). Здесь же создаются основные объекты, используемые в игре:

//Делегат на метод отображения сообщения в чате

ShowMessageDelegate del = new ShowMessageDelegate(ShowMessage);

//делегат на метод выстрела соперника в поле игрока

ReceiveShotDelegate delShot = new ReceiveShotDelegate(Shot);

//делегат на метод выстрела игрока в поле соперника

ReceiveCellShotResultDelegate delShotRes = new ReceiveCellShotResultDelegate(ShotResults);

\_boardLeft = new Board(); //левое игровое поле

\_boardRight = new Board(false); //правое игровое поле. В конструкторе указываем, что не будем на нем рисовать корабли

\_scoreboard = new ScoreBoard(); //счёт

\_controller = new GameController(\_boardLeft, \_boardRight, \_scoreboard); //контроллер игры

network = new Network(del, delShot, delShotRes); //контроллер сетевого соединения

Делегаты на методы нужны для вызова этих методов из параллельных потоков. В нашем случае это класс network, который в параллельном потоке принимает запросы от противника.

**Network**

Этот класс предназначен для обеспечения клиент-серверного сетевого соединения. Имеются три команды – сообщение, выстрел и результат выстрела. Идентификаторы этих команд представлены строками, которые передаются в начале каждой датаграммы.

private const string cmdSendShot = "sht";

private const string cmdShotResult = "shr";

private const string cmdChatMsg = "msg";

Класс содержит в себе два udp-клиента: один на синхронную передачу датаграмм (в основном потоке), другой – на асинхронную (в отдельном потоке). Клиенты инициализируются в методе Connect()

public void Connect(string MyIP, string FriendIP, int MyPort, int FriendPort)

{

udpListener = new UdpClient(MyPort);

udpSender = new UdpClient();

udpSender.Connect(FriendIP, FriendPort);

\_thrListen = new Thread(new ThreadStart(ListenUdp));

\_thrListen.Start();

}

В методе ListenUdp() клиент, принимающий данные по udp, прослушивает порт, указанный в методе Connect. По приходу новой датаграммы сообщение разбирается, и в зависимости от идентификатора либо вызывается делегат на передачу сообщения, либо принимается выстрел от противника или результат выстрела в противника.

За правилами игры следит класс GameController, обращаясь к таблице результатов и объектам игровых полей (классы \_board1 для своего поля и \_board2 для поля соперника)

//выстрел от противника

public void shootResult(int X, int Y)

{

ShotResult res = \_board1.OpenentShotAt(X, Y);

if(res == ShotResult.Missed) //если противник промахнулся

{

\_board2.Mode = BoardMode.Game; //разрешаем себе стрелять в поле противника

\_scoreBoard.TakeControl(); //принимаем ход

}

else if(res == ShotResult.ShipDrowned) //иначе, если противник потопил корабль

{

\_scoreBoard.EnemyDrownedMe(); //выполняем обработчик потопления своего корабля

}

}

//результат выстрела в поле противника

public void friendShootResult(int X, int Y, ShotResult State)

{

switch(State)

{

case ShotResult.Missed: //если я промахнулся

\_board2.\_cells[X, Y].State = BoardCellState.MissedShot; //помечаю ячейку промахом

\_board2.Mode = BoardMode.Yeld; //запрещаем себе стрелять в поле противника

\_scoreBoard.YeldControl(); //передаем ход противнику

break;

case ShotResult.ShipHit: //если я попал в противника

\_board2.\_cells[X, Y].State = BoardCellState.ShotShip; //помечаю ячейку подстреленой

break;

case ShotResult.ShipDrowned: //если я потопил корабль

\_board2.\_cells[X, Y].State = BoardCellState.ShowDrowned; //помечаю ячейку потопленной

\_scoreBoard.IDrownedEnemy(); //вызываю обработчик потопления корабля противника

break;

}

\_board2.Invoke(new System.Action(() => \_board2.Refresh())); //перерисовываем поле противника

}

Класс ScoreBoard следит за счётом и оставшимися кораблями у каждого игрока (методы RefreshPlayerStats и RefreshScore). При запуске игры, когда каждый игрок имеет по 10 кораблей, отрисовка происходит синхронно, т.е. в основном потоке, и методы отрисовки вызываются напрямую. Во время игры обращение к классу ScoreBoard происходит в потоке приёма данных по UDP, поэтому приходится использовать дополнительные методы синхронизации, как и в методах отрисовки в других классах. Например,

\_playerStats.First.Invoke(new Action(() => \_playerStats.First.Text = string.Format(PlayerStatsTemplate, \_shipsLeft.X)));

Когда у одного из игроков не остается кораблей, игра считается завершенной.

Графические компоненты, такие как игровое поле и корабль, реализуются классами Board и Ship соответственно.