```
{   //[Program.cs]
    internal static class Program
    {
        /// <summary>
        ///  The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            // To customize application configuration such as set high DPI settings or default font,
            // see https://aka.ms/applicationconfiguration.
            Application.EnableVisualStyles();
            ApplicationConfiguration.Initialize();
            Application.Run(new SelectScreen());
        }
    }
    /// <summary>
    /// Klasse die, die Farben für die jeweiligen Elemente des Programms enthält.
    /// </summary>
    public static class Theme
    {
        /// <summary>
        /// Die Farben für den DarkMode
        /// Anfang ist von: https://www.color-hex.com/color-palette/98179
        /// </summary>
        static Color[] DarkModeColors =
        {
            //------------Farbe-----------    ----------Beschreibung----------
            -----Systemfarbe-----
            Color.FromArgb(255, 255, 255),  // Text
            Control
            Color.FromArgb(62, 62, 66),     // Hintergund 4
            ControlDark
            Color.FromArgb(45, 45, 48),     // Hintergund 3
            ControlDarkDark
            Color.FromArgb(37, 37, 38),     // Hintergund 2
            ControlLight
            Color.FromArgb(30, 30, 30),     // Hintergund
            ControlLightLight
            Color.FromArgb(0, 122, 204),    // Kontur
            ControlText
            Color.FromArgb(22, 22, 50),     // Zu viele Schiffe
            Desktop
            Color.FromArgb(21, 55, 18),     // Richtige Anzahl Schiffe
            GrayText
            Color.FromArgb(128, 30, 0),     // Zu wenig Schiffe
            Highlight
            Color.FromArgb(128, 128, 128),  // Schiff
            HighlightText
            Color.FromArgb(128, 0, 0),      // Versenkt
            HotTrack
        };
        /// <summary>
        /// Die Farben für den DarkMode
        /// Anfang ist von: https://www.color-hex.com/color-palette/106748
        /// </summary>
        static Color[] LightModeColors =
        {
            //------------Farbe-----------    ----------Beschreibung----------
            -----Systemfarbe-----
            Color.FromArgb(0, 0, 0),        // Text
            Control
            Color.FromArgb(147,148,165),    // Hintergund 4
            ControlDark
            Color.FromArgb(210,211,219),    // Hintergund 3
            ControlDarkDark
            Color.FromArgb(228,229,241),    // Hintergund 2
            ControlLight
            Color.FromArgb(250,250,250),    // Hintergund
            ControlLightLight
```

```
53              Color.FromArgb(72,75,106),      // Kontur
                Desktop
54              Color.FromArgb(173, 216, 230),  // Zu viele Schiffe
                GrayText
55              Color.FromArgb(144, 238, 144),  // Richtige Anzahl Schiffe
                Highlight
56              Color.FromArgb(255, 182, 193),  // Zu wenig Schiffe
                HighlightText
57              Color.FromArgb(128, 128, 128),  // Schiff
                HotTrack
58              Color.FromArgb(255, 127, 80),   // Versenkt
                InactiveCaption
59          };
60          static bool DarkMode = true;
61
62          /// <summary>
63          /// Wechselt zwischen DarkMode und LightMode
64          /// </summary>
65          public static void ChangeMode()
66          {
67              DarkMode = !DarkMode;
68          }
69
70          /// <summary>
71          ///
72          /// </summary>
73          /// <returns>true falls das Theme gerade Dunkel ist.</returns>
74          public static bool IsDarkMode()
75          {
76              return DarkMode;
77          }
78
79          /// <summary>
80          /// Gibt die jeweilige Theme Farbe für die aktuelle Farbe zurück
81          /// </summary>
82          /// <param name="type"></param>
83          /// <returns></returns>
84          public static Color GetThemeColor(Color type)
85          {
86              // Es wird überprüft ob die Farbe eine Systemfarbe ist (also im
                Fromdesigner gesetzte) Farbe ist
87              if (type.IsNamedColor)
88              {
89                  // Wenn sie eine Systemfarbe ist, dann wird durch den Wert der Farbe
                    in dem KnownColor Enum die Farbe zurückgegeben
90                  if (((int)type.ToKnownColor()- 5) >= LightModeColors.Length)
91                      return type;
92                  return DarkMode ? DarkModeColors[(int)type.ToKnownColor() - 5] :
                    LightModeColors[(int)type.ToKnownColor() - 5];
93              }
94              else
95              {
96                  // Wenn nicht, dann wird der Index der Farbe in dem alten Colorarray
                    gesucht und die jeweilige Farbe, des Indexes im neuen zurückgegeben
97                  int index = DarkMode ? Array.IndexOf(LightModeColors, type) : Array.
                    IndexOf(DarkModeColors, type);
98                  if (index >= LightModeColors.Length || index < 0)
99                      return type;
100                 return DarkMode ? DarkModeColors[index] : LightModeColors[index];
101             }
102         }
103         /// <summary>
104         /// Aktualisiert das Theme für das aktuelle Form/Control
105         /// </summary>
106         /// <param name="control"></param>
107         public static void RefreshTheme(Control control)
108         {
109             // Für jedes Control wird die Funktion erneut aufgerufen
110             foreach (Control c in control.Controls)
111             {
112                 RefreshTheme(c);
```

```
113             }
114             // Wenn das Control ein DataGridView ist, dann werden die Farben der
                einzelnen Elemente gesetzt
115             if (control.GetType() == typeof(DataGridView))
116             {
117                 DataGridView dg = (DataGridView)control;
118                 dg.DefaultCellStyle.BackColor = GetThemeColor(dg.DefaultCellStyle.
                    BackColor);
119                 dg.DefaultCellStyle.ForeColor = GetThemeColor(dg.DefaultCellStyle.
                    ForeColor);
120                 // Für jede Zelle wird die Hintergrundfarbe und die Vordergrundfarbe
                    gesetzt
121                 foreach (DataGridViewRow i in dg.Rows)
122                 {
123                     foreach (DataGridViewCell c in i.Cells)
124                     {
125                         c.Style.ForeColor = GetThemeColor(c.Style.ForeColor);
126                         c.Style.BackColor = GetThemeColor(c.Style.BackColor);
127                     }
128                 }
129 
130                 dg.ColumnHeadersDefaultCellStyle.BackColor = GetThemeColor(dg.
                    ColumnHeadersDefaultCellStyle.BackColor);
131                 dg.ColumnHeadersDefaultCellStyle.ForeColor = GetThemeColor(dg.
                    ColumnHeadersDefaultCellStyle.ForeColor);
132                 dg.GridColor = GetThemeColor(dg.GridColor);
133                 dg.BackgroundColor = GetThemeColor(dg.BackgroundColor);
134 
135                 // Für jede Spalte wird die Hintergrundfarbe und die Vordergrundfarbe
                    vom Header gesetzt
136                 foreach (DataGridViewColumn i in dg.Columns)
137                 {
138                     i.HeaderCell.Style.BackColor = GetThemeColor(i.HeaderCell.Style.
                        BackColor);
139                     i.HeaderCell.Style.ForeColor = GetThemeColor(i.HeaderCell.Style.
                        ForeColor);
140                 }
141             }
142             // Wenn das Control ein Button ist, dann wird die Randfarbe des Buttons
                gesetzt
143             else if (control.GetType() == typeof(Button))
144             {
145                 ((Button)control).FlatAppearance.BorderColor = GetThemeColor(((Button)
                    control).FlatAppearance.BorderColor);
146             }
147             // Von jedem Control wird die Hintergrundfarbe und die Vordergrundfarbe
                gesetzt
148             control.BackColor = GetThemeColor(control.BackColor);
149             control.ForeColor = GetThemeColor(control.ForeColor);
150         }
151     }
152 }
153 {   //[SelectScreen.cs]
154     public partial class SelectScreen : Form
155     {
156         public SelectScreen()
157         {
158             InitializeComponent();
159 
160             // Ändert das Design erstmal damit der ChangeDesignBT
161             // das gegenteilige Design hat um ihn hervorzuheben
162             Theme.ChangeMode();
163             ChangeDesignBT_Click(null, null);
164             //-----------------------------------------------------
165         }
166 
167         /// <summary>
168         /// Wenn der JoinBT geklickt wird, dann wird ein <see cref="LobbyScreen"/>
            als <c>nicht Host</c> erstellt und angezeigt
169         /// </summary>
170         /// <param name="sender"></param>
```

```
171         /// <param name="e"></param>
172         private void JoinBT_Click(object sender, EventArgs e)
173         {
174             LobbyScreen ls = new LobbyScreen(false);
175             Hide();
176             ls.ShowDialog();
177             Show();
178         }
179         /// <summary>
180         /// Wenn der JoinBT geklickt wird, dann wird ein <see cref="LobbyScreen"/>
            als <c>Host</c> erstellt und angezeigt
181         /// </summary>
182         /// <param name="sender"></param>
183         /// <param name="e"></param>
184         private void HostBT_Click(object sender, EventArgs e)
185         {
186             LobbyScreen ls = new LobbyScreen(true);
187             Hide();
188             ls.ShowDialog();
189             Show();
190         }
191 
192         /// <summary>
193         /// Ändert das Design des aktuellen Forms
194         /// </summary>
195         /// <param name="sender"></param>
196         /// <param name="e"></param>
197         private void ChangeDesignBT_Click(object? sender, EventArgs? e)
198         {
199             // Speichert die Farben des Buttons im aktuellen Design
200             Color btBackColor = Theme.GetThemeColor(ChangeDesignBT.BackColor);
201             Color btForeColor = Theme.GetThemeColor(ChangeDesignBT.ForeColor);
202             // Wechselt das Design
203             Theme.ChangeMode();
204             Theme.RefreshTheme(this);
205             ChangeDesignBT.Text = Theme.IsDarkMode() ? "Light Mode" : "Dark Mode";
206             // Setzt die Farben des Buttons auf die Farben des alten Designs um das
                andere Design zu zeigen
207             ChangeDesignBT.BackColor = btBackColor;
208             ChangeDesignBT.ForeColor = btForeColor;
209         }
210     }
211 }
212 {   //[LobbyScreen.cs]
213     public partial class LobbyScreen : Form
214     {
215         // Icons:
216         // <a href="https://www.flaticon.com/free-icons/user" title="user icons">User
            icons created by Freepik - Flaticon</a>
217         // <a href="https://www.flaticon.com/free-icons/ai" title="ai icons">Ai icons
            created by Icongeek26 - Flaticon</a>
218 
219 
220         private ShipGrid shipGrid;
221         private bool isHost;
222 
223         /// <summary>
224         /// Der Socket der für die Verbindung genutzt wird
225         /// </summary>
226         private TcpSocket Socket { get; set; }
227         /// <summary>
228         /// Der Client Sobald eine Verbindung über den Socket hergestellt wurde
229         /// </summary>
230         private TcpClient? Client { get; set; }
231 
232         /// <summary>
233         /// CancellationTokenSource um den Verbindungsaufbau abzubrechen falls der
            Benutzer sich mit einem anderen Code verbinden möchte
234         /// </summary>
235         private CancellationTokenSource ConnectCancel = new();
236 
```

```
237         public LobbyScreen(bool _isHost)
238         {
239             InitializeComponent();
240 
241             Socket = new();
242 
243             isHost = _isHost;
244             // Erstellt ein neues ShipGrid wo nur der Host Anzahl der Schiffe ändern
                kann
245             shipGrid = new ShipGrid(ShipsInfoGrid, isHost);
246 
247             // Ändert das Design erstmal damit der ChangeDesignBT
248             // das gegenteilige Design hat um ihn hervorzuheben
249             Theme.ChangeMode();
250             ChangeDesignBT_Click(null, null);
251             //-------------------------------------------------
252 
253             if (!isHost)
254             {
255                 OponentIMG.Image = Properties.Resources.user;
256                 OponentNameLB.Text = "Du";
257                 PlayerNameLB.Text = "Gegner";
258                 StartBT.Visible = false;
259             }
260         }
261 
262         /// <summary>
263         /// Sobald das Form geladen hat wird der Code vom Internet abgerufen und
            angezeigt
264         /// </summary>
265         /// <param name="sender"></param>
266         /// <param name="e"></param>
267         private async void LobbyScreen_Load(object sender, EventArgs e)
268         {
269             await Socket.UpdateCode();
270             OnCodeChange();
271         }
272 
273         /// <summary>
274         /// Versucht eine Verbindung mit dem Client mit dem Code aus dem CodeInputTB
            herzustellen
275         /// </summary>
276         /// <param name="sender"></param>
277         /// <param name="e"></param>
278         private async void ConnectBT_Click(object sender, EventArgs e)
279         {
280             // Wenn bereits einmal versucht wurde eine Verbindung herzustellen, wird
                diese abgebrochen
281             // Falls bereits eine Verbindung besteht, wird abgebrochen
282             if (Client is not null)
283             {
284                 if (Client.Connected) return;
285                 ConnectCancel.Cancel();
286                 Client.Close();
287             }
288             string code = CodeInputTB.Text;
289 
290             Invoke(() => StatusLB.Text = "Wird Verbunden");
291             ConnectCancel = new();
292 
293             // Versucht eine Verbindung mit dem Code herzustellen bis es dur einen
                neuen Versuch abgebrochen wird oder eine Verbindung hergestellt wurde
294             try { Client = await Socket.Connenct(Compressor.UnZip(code)).WaitAsync(
                ConnectCancel.Token); }
295             catch (TaskCanceledException) { Invoke(() => StatusLB.Text = "Verbindung
                fehlgeschlagen"); return; }
296 
297             // Der host muss das gegner Bild und den Namen ändern -> da davor KI
298             if (isHost)
299             {
300                 OponentIMG.Image = Properties.Resources.user;
```

```
301                 OponentNameLB.Text = "Gegner";
302             }
303 
304             Invoke(() => StatusLB.Text = "Verbunden");
305 
306 
307             if (isHost)
308             {
309                 // Der Host sendet seine Schifftypen (ShipGridEntry) an den Client
                    sobald sich die Anzahl der Schiffe ändert
310                 await Client!.SendData(shipGrid.Ships.Values.ToArray());
311                 shipGrid.AmountChangedEvent += async (a, b) => await Client!.SendData(
                    b);
312             }
313             else
314             {
315                 // Der Client wartet auf Nachrichten vom Host
316                 _ = Client?.Receive().ContinueWith((message) => callback(message.
                    Result));
317             }
318 
319         }
320         /// <summary>
321         /// Wenn eine eue Nachricht vom Host kommt, wird diese Funktion aufgerufen
322         /// </summary>
323         /// <param name="message"></param>
324         private void callback(StunTools.Message message)
325         {
326             // Wenn die Nachricht ein String ist und der String "START" ist, wird das
                Spiel gestartet
327             if (typeof(string) == message.ObjectType)
328             {
329                 if (message.GetData<string>() == "START")
330                 {
331                     startGame();
332                     return;
333                 }
334             }
335             else
336             {
337                 // Sonst ist die Nachricht ein IEnumerable<ShipGridEntry>, also
                    werden die Schifftypen aktualisiert
338                 shipGrid.SetEntries(message.GetData<IEnumerable<ShipGridEntry>>()!);
339             }
340             // starte eine neue Task die auf die nächste Nachricht wartet
341             _ = Client?.Receive().ContinueWith((message) => callback(message.Result));
342         }
343 
344         /// <summary>
345         /// Sobald der Code sich ändert wird diese Funktion aufgerufen um den Code
            anzuzeigen
346         /// </summary>
347         internal void OnCodeChange()
348         {
349             Invoke(() => CodeLB.Text = Socket.Code);
350             Invoke(() => PortLB.Text = Socket.LocalEndPoint?.Port.ToString());
351         }
352 
353         /// <summary>
354         /// Kopiert den Code in die Zwischenablage
355         /// </summary>
356         /// <param name="sender"></param>
357         /// <param name="e"></param>
358         private void CopyCodeBT_Click(object sender, EventArgs e)
359         {
360             if (Socket.Code != null)
361                 Clipboard.SetText(Socket.Code);
362         }
363 
364         /// <summary>
365         /// Startet das eigentliche Spiel
```

```
366         /// </summary>
367         private void startGame()
368         {
369             BeginInvoke(() =>
370             {
371                 // Wenn das Spiel gestartet wird, wird das LobbyScreen versteckt und
                    das MainGame geöffnet
372                 Hide();
373                 new MainGame(Client!, shipGrid.Ships.Values.ToArray()).ShowDialog();
374                 // Nachdem das Maingame geschlossen wurde, wird das LobbyScreen
                    geschlossen
375                 Close();
376             });
377         }
378         /// <summary>
379         /// Startet das Spiel wenn der StartBT geklickt wird
380         /// </summary>
381         /// <param name="sender"></param>
382         /// <param name="e"></param>
383         private async void StartBT_Click(object sender, EventArgs e)
384         {
385             if (Client is not null && Client.Connected)
386                 await Client.SendData("START");
387             startGame();
388         }
389 
390         /// <summary>
391         /// Ändert das Design des aktuellen Forms
392         /// </summary>
393         /// <param name="sender"></param>
394         /// <param name="e"></param>
395         private void ChangeDesignBT_Click(object? sender, EventArgs? e)
396         {
397             // Speichert die Farben des Buttons im aktuellen Design
398             Color btBackColor = Theme.GetThemeColor(ChangeDesignBT.BackColor);
399             Color btForeColor = Theme.GetThemeColor(ChangeDesignBT.ForeColor);
400             // Wechselt das Design
401             Theme.ChangeMode();
402             Theme.RefreshTheme(this);
403             ChangeDesignBT.Text = Theme.IsDarkMode() ? "Light Mode" : "Dark Mode";
404             // Setzt die Farben des Buttons auf die Farben des alten Designs um das
                andere Design zu zeigen
405             ChangeDesignBT.BackColor = btBackColor;
406             ChangeDesignBT.ForeColor = btForeColor;
407         }
408     }
409 }
410 {   //[ShipGrid.cs]
411     public class ShipGrid
412     {
413         // ---------Für Multiplayer---------
414         public delegate void AmountChangedEventHandler(object? sender, IEnumerable<
            ShipGridEntry> entries);
415         public event AmountChangedEventHandler? AmountChangedEvent;
416         // ---------Für Multiplayer---------
417 
418         /// <summary>
419         /// Ob die Anzahl der fehlenden Schiffe bearbeitet werden kann
420         /// </summary>
421         public readonly bool Editable;
422 
423         /// <summary>
424         /// Ships als <see cref="ShipGridEntry"/> mit der länge des Schifftyps als key
425         /// </summary>
426         public readonly Dictionary<int, ShipGridEntry> Ships;
427 
428         /// <summary>
429         /// Das <see cref="DataGridView"/> was dir Table in der UI darstellt
430         /// </summary>
431         private readonly DataGridView shipsGrid;
432 
```

```
433
434     #pragma warning disable CS8618 // Weil nicht alle Felder direkt in dem Konstruktor
        gesetzt werden, sondern auch in der Methode genShips
435             public ShipGrid(DataGridView _shipsGrid, bool _editable = false)
436     #pragma warning restore CS8618
437             {
438                 shipsGrid = _shipsGrid;
439                 Editable = _editable;
440
441                 Ships = new Dictionary<int, ShipGridEntry>();
442
443                 shipsGrid.CellMouseEnter += mouseEnter;
444                 shipsGrid.MouseLeave += mouseLeave;
445
446                 if (Editable)
447                     shipsGrid.CellEndEdit += editEnd;
448
449                 genShips();
450             }
451
452             /// <summary>
453             /// Gibt zurück ob die Anzahl der fehlenden Schiffe 0 ist
454             /// </summary>
455             /// <returns><see cref="true"/> wenn nichts fehlend</returns>
456             public bool ShipAmountEmpty()
457             {
458                 foreach (ShipGridEntry s in Ships.Values)
459                 {
460                     if (s.Amount != 0)
461                         return false;
462                 }
463
464                 return true;
465             }
466
467             /// <summary>
468             /// Setzt die Anzahl der fehlenden Schiffe des jeweiligen Schifftypes
469             /// </summary>
470             /// <param name="shipGridEntries"></param>
471             public void SetEntries(IEnumerable<ShipGridEntry> shipGridEntries)
472             {
473                 foreach (ShipGridEntry ship in shipGridEntries)
474                 {
475                     Ships[ship.Length].Update(ship);
476                 }
477             }
478
479             /// <summary>
480             /// Setzt die Anzahl der fehlenden Schiffe des jeweiligen Schifftypes durch
                die Länge der Links
481             /// </summary>
482             /// <param name="links"></param>
483             public void UpdateAmount(Link[] links)
484             {
485                 ResetAmount();
486                 foreach (Link link in links)
487                 {
488                     // Falls es ein Schifftyp gibt, das die Länge des Links hat, dann
                        wird die Anzahl der fehlenden Schiffe des Typs um 1 verringert
489                     if (Ships.ContainsKey(link.Length))
490                     {
491                         Ships[link.Length].Amount--;
492                     }
493                 }
494             }
495             /// <summary>
496             /// Setzt den Wert der fehlenden Schiffe auf den amount Wert
497             /// </summary>
498             /// <param name="amount">Wenn null dann auf den Maximalwert</param>
499             public void ResetAmount(int? amount = null)
500             {
```

```
501             foreach (ShipGridEntry ship in Ships.Values)
502             {
503                 ship.ResetAmount(amount);
504             }
505         }
506 
507         /// <summary>
508         /// Hover effect für die Zellen
509         /// Sobald die Maus über eine Zelle ist wird es als ausgewählt markiert
510         /// </summary>
511         /// <param name="sender"></param>
512         /// <param name="e"></param>
513         private void mouseEnter(object? sender, DataGridViewCellEventArgs e)
514         {
515             if (e.ColumnIndex != -1 && e.RowIndex != -1)
516                 shipsGrid[e.ColumnIndex, e.RowIndex].Selected = true;
517         }
518 
519         /// <summary>
520         /// Ändert die Anzahl der fehlenden Schiffe des jeweiligen Schifftypes
521         /// </summary>
522         /// <param name="sender"></param>
523         /// <param name="e"></param>
524         private void editEnd(object? sender, DataGridViewCellEventArgs e)
525         {
526             // Wenn die Eingabe eine Zahl ist und zwischen 0 und 10 liegt, dann wird
                die Anzahl der fehlenden Schiffe des jeweiligen Schifftypes geändert
527             if (int.TryParse((string)shipsGrid[e.ColumnIndex, e.RowIndex].Value, out
                int res) && res <= 10 && res >= 0)
528             {
529                 Ships.Values.ElementAt(e.RowIndex).MaxAmount = res;
530                 AmountChangedEvent?.Invoke(sender, Ships.Values);
531                 return;
532             }
533             // Sonst wird die Anzahl in der UI auf den vorherigen Wert gesetzt
534             Ships.Values.ElementAt(e.RowIndex).ResetAmount();
535         }
536 
537         /// <summary>
538         /// Hover effect für die Zellen
539         /// Löscht die Auswahl sobald die Maus die Zelle verlässt
540         /// </summary>
541         /// <param name="sender"></param>
542         /// <param name="e"></param>
543         private void mouseLeave(object? sender, EventArgs e)
544         {
545             ResetSelection();
546         }
547 
548         /// <summary>
549         /// Löscht die Auswahl
550         /// </summary>
551         public void ResetSelection()
552         {
553             shipsGrid.ClearSelection();
554         }
555 
556         /// <summary>
557         /// Gibt die Schiffe als <see cref="int"/> zurück indem die Anzahl der
            fehlenden Schiffe durch jeweiligen int der Länge des Schiffs dargestellt wird
558         /// Also ein Schiff mit der Länge 3 und 2 Schiffen wird als {3, 3} dargestellt
559         /// </summary>
560         /// <param name="getMissingOnly">Ob nur die Anzahl die noch nicht plaziert
            wurde zurückgegeben werden soll</param>
561         /// <returns></returns>
562         public IEnumerable<int> GetShipsAsInt(bool getMissingOnly = false)
563         {
564             foreach (ShipGridEntry ship in Ships.Values)
565             {
566                 // Entweder wird die Anzahl der fehlenden Schiffe oder die Anzahl der
                    maximalen Schiffe zurückgegeben
```

```
567                 int amount = getMissingOnly ? ship.Amount : ship.MaxAmount;
568                 for (int i = 0; i < amount; i++)
569                 {
570                     yield return ship.Length;
571                 }
572             }
573         }
574 
575         /// <summary>
576         /// Erstellt eine neue Zeile in der Tabelle und fügt den Schifftype in das <see cref="Ships"/> dict hinzu
577         /// </summary>
578         /// <param name="name"></param>
579         /// <param name="length"></param>
580         /// <param name="amount"></param>
581         /// <returns></returns>
582         private ShipGridEntry createShiptypeCell(string name, int length, int amount)
583         {
584             DataGridViewRow row = shipsGrid.Rows[shipsGrid.Rows.Add(name, length, amount)];
585             ShipGridEntry ship = new ShipGridEntry(name, length, amount, row);
586             Ships.Add(length, ship);
587             return ship;
588         }
589 
590         /// <summary>
591         /// Generiert/Formatiert die Tabelle und erstellt alle <see cref="ShipGridEntry"/>s
592         /// </summary>
593         private void genShips()
594         {
595             // Erstelle die Spalten
596             shipsGrid.Columns.Add("Column0", "Schiff");
597             shipsGrid.Columns.Add("Column1", "Länge");
598             shipsGrid.Columns.Add("Column2", "Fehlend");
599 
600             // Macht das man nicht sortieren kann
601             shipsGrid.Columns[0].SortMode = DataGridViewColumnSortMode.NotSortable;
602             shipsGrid.Columns[1].SortMode = DataGridViewColumnSortMode.NotSortable;
603             shipsGrid.Columns[2].SortMode = DataGridViewColumnSortMode.NotSortable;
604 
605             // Erlaubt nur die Bearbeitung der Anzahl der fehlenden Schiffe wenn Editable true ist
606             shipsGrid.Columns[0].ReadOnly = true;
607             shipsGrid.Columns[1].ReadOnly = true;
608             shipsGrid.Columns[2].ReadOnly = !Editable;
609 
610             #region Ships
611             createShiptypeCell("Atomschiff", 6, 0);
612             createShiptypeCell("Schlachtschiff", 5, 1);
613             createShiptypeCell("Kreuzer", 4, 2);
614             createShiptypeCell("Zerstörer", 3, 3);
615             createShiptypeCell("U-Boot", 2, 4);
616             createShiptypeCell("Jetski", 1, 0);
617             #endregion
618 
619             // Setze die Höhe der Zeilen auf die Höhe der Tabelle durch die Anzahl der Einträge
620             int headerHeight = shipsGrid.ColumnHeadersHeight;
621             for(int i = 0; i < Ships.Count; i++)
622             {
623                 shipsGrid.Rows[i].Height = (shipsGrid.Size.Height - headerHeight) / Ships.Count;
624             }
625         }
626     }
627     /// <summary>
628     /// Die Datenebene für eine Reihe in <see cref="DataGridView"/>
629     /// </summary>
630     public class ShipGridEntry
631     {
```

```
632         public string Name { get => name; set { name = value; updateVariable(name, 0);
             } }
633         public int Length { get => length; set { length = value; updateVariable(length
            , 1); } }
634
635         /// <summary>
636         /// Anzahl der fehlenden Schiffe
637         /// </summary>
638         public int Amount { get => amount; set { amount = value; updateVariable(amount
            , 2); updateColor(amount, 2); } }
639
640         /// <summary>
641         /// Anzahl der exestierenden Schiffe
642         /// </summary>
643         public int MaxAmount { get => maxAmount; set { maxAmount = value; Amount =
            value; } }
644
645         private string name;
646         private int length;
647         private int amount;
648         private int maxAmount;
649
650         /// <summary>
651         /// Die Zeile in der Tabelle die die Daten in der UI darstellt
652         /// Könnte null sein da es wenn das <see cref="ShipGridEntry"/> über
            JsonConvert.DeserializeObject (Netzwerk) erstellt wird, die Variable nicht
            gesetzt wird
653         /// </summary>
654         [JsonIgnore]
655         private readonly DataGridViewRow? entryRow;
656
657         public ShipGridEntry(string _name, int _length, int maxAmount, DataGridViewRow
             _entryRow)
658         {
659             entryRow = _entryRow;
660             name = _name;
661             length = _length;
662             MaxAmount = maxAmount;
663         }
664
665         /// <summary>
666         /// Aktualisiert die Daten auf die selben werte eines anderen <see
            cref="ShipGridEntry"/>
667         /// </summary>
668         /// <param name="shipGridEntry"></param>
669         public void Update(ShipGridEntry shipGridEntry)
670         {
671             Name = shipGridEntry.Name;
672             Length = shipGridEntry.Length;
673             Amount = shipGridEntry.Amount;
674             MaxAmount = shipGridEntry.MaxAmount;
675         }
676
677         /// <summary>
678         /// Setzt den Amount auf den value Wert
679         /// </summary>
680         /// <param name="amount">Wenn null dann auf den Maximalwert</param>
681         public void ResetAmount(int? value = null)
682         {
683             if (value == null) Amount = MaxAmount;
684             else Amount = value.Value;
685         }
686
687         /// <summary>
688         /// Aktualisiert die Variable in der UI
689         /// </summary>
690         /// <param name="value">Neuer wert</param>
691         /// <param name="cell">Reihe</param>
692         private void updateVariable(object value, int cell)
693         {
694             // Soll nicht aktualisiert werden wenn die Variable null ist (Es von
```

```
            einem Netzwerk kommt)
            if (entryRow is not null)
                entryRow.Cells[cell].Value = value.ToString();
        }

        /// <summary>
        /// Aktualisiert die Farbe in der UI basierend auf dem Wert
        /// </summary>
        /// <param name="value">amount</param>
        /// <param name="cell">Reihe</param>
        private void updateColor(int value, int cell)
        {
            // Soll nicht aktualisiert werden wenn die Variable null ist (Es von
            einem Netzwerk kommt)
            if (entryRow is null) return;
            // Holt die korrekte Farbe von Theme basierend auf dem Wert von value
            switch (value)
            {
                case 0:
                    entryRow.Cells[cell].Style.BackColor = Theme.GetThemeColor(
                    SystemColors.GrayText);
                    break;
                case < 0:
                    entryRow.Cells[cell].Style.BackColor = Theme.GetThemeColor(
                    SystemColors.Highlight);
                    break;
                case > 0:
                    entryRow.Cells[cell].Style.BackColor = Theme.GetThemeColor(
                    SystemColors.Desktop);
                    break;
            }

        }
    }
}
{   //[MainGame.cs]
    public partial class MainGame : Form
    {
        // Hier wird viel Invoke verwendet, da die Methoden von anderen Threads
        aufgerufen werden können dadurch das hier teilweise auf netzwerkdaten
        gewartet wird
        // Durch Invoke wird sichergestellt, dass die Methode auf dem UI Thread
        ausgeführt wird, denn nur der UI Thread kann das UI verändern

        /// <summary>
        /// Die Logik des Spiels
        /// </summary>
        public FieldHandler FieldHandler;

#pragma warning disable CS8618 // Weil nicht alle Felder direkt in dem Konstruktor
gesetzt werden, sondern auch in der Methode genShips
        public MainGame(TcpClient? connection, ShipGridEntry[] shipPlacementConfig)
#pragma warning restore CS8618
        {
            InitializeComponent();
            Start(shipPlacementConfig, connection);

            // Ändert das Design erstmal damit der ChangeDesignBT
            // das gegenteilige Design hat um ihn hervorzuheben
            Theme.ChangeMode();
            ChangeDesignBT_Click(null, null);
            //-------------------------------------------------
        }
        /// <summary>
        /// Erstellt die spielwichtigen Objekte
        /// </summary>
        /// <param name="shipPlacementConfig"></param>
        /// <param name="connection"></param>
        public void Start(ShipGridEntry[] shipPlacementConfig, TcpClient? connection)
        {
            // Erstelle die Felder
```

```
            Field playerField = new Field(PlayerFieldGrid);
            Field enemyField = new Field(EnemyFieldGrid);

            // Erstelle die Schiffgrids
            ShipGrid playerShipGrid = new ShipGrid(PlayerShipsInfoGrid);
            ShipGrid enemyShipGrid = new ShipGrid(EnemyShipsInfoGrid);

            playerShipGrid.SetEntries(shipPlacementConfig);
            enemyShipGrid.SetEntries(shipPlacementConfig);

            // Erstelle den FieldHandler
            FieldHandler = new FieldHandler(playerField, enemyField, playerShipGrid,
            enemyShipGrid, this, connection);
        }

        /// <summary>
        /// Zeigt an, dass der Gegner bereit ist
        /// </summary>
        public void EnemyReady()
        {
            Invoke(() => EnemyInfoLB.Text = "Bereit");
        }

        /// <summary>
        /// Zeigt an, ob man gewonnen oder verloren hat
        /// </summary>
        /// <param name="won">Wenn true dann gewonnen</param>
        public void EndGame(bool won)
        {
            BeginInvoke(() => MessageBox.Show($"Du hast dieses Match {(won ? "GEWONNEN
            " : "VERLOREN")}!\nKehre nun zum Startbildschirm zurück!", "END"));
        }

        /// <summary>
        /// Setzt den Text des TurnArrows und der Info Labels
        /// </summary>
        /// <param name="turn">Wenn true dann ist der Spieler am Zug</param>
        public void SetTurn(bool turn)
        {
            Action a;
            if (turn)
            {
                a = () =>
                {
                    TurnArrowTB.Text = "<-";
                    PlayerInfoLB.Text = "Am Zug";
                    EnemyInfoLB.Text = "Wartet auf Dich";
                };
            }
            else
            {
                a = () =>
                {
                    TurnArrowTB.Text = "->";
                    PlayerInfoLB.Text = "Wartet auf Gegner";
                    EnemyInfoLB.Text = "Am Zug";
                };
            }

            if (InvokeRequired)
                Invoke(a);
            else
                a();
        }

        /// <summary>
        /// Wenn das Form geladen hat, werden alle Auswahlen zurückgesetzt
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void Form1_Load(object sender, EventArgs e)
```

```
826             {
827                 FieldHandler.Loaded();
828             }
829             /// <summary>
830             /// Das Playerfeld wird angezeigt oder versteckt
831             /// </summary>
832             /// <param name="sender"></param>
833             /// <param name="e"></param>
834             private void hideBoard_CheckedChanged(object sender, EventArgs e)
835             {
836                 FieldHandler.HidePlayerField(((CheckBox)sender).Checked);
837             }
838 
839             /// <summary>
840             /// Überprüft ob alles passt und setzt dann den Spieler auf Bereit
841             /// </summary>
842             /// <param name="sender"></param>
843             /// <param name="e"></param>
844             private void ReadyBT_Click(object sender, EventArgs e)
845             {
846                 // Wenn der Spieler bereit ist, wird geprüft ob alle Schiffe platziert
                    sind
847                 if (!FieldHandler.PlayerShipAmountEmpty())
848                 {
849                     MessageBox.Show("Es sind noch nicht alle oder zu viele Schiffe
                        platziert!");
850                     return;
851                 }
852 
853                 FieldHandler.Ready();
854 
855                 // Der Button und das Controlcenter werden versteckt
856                 ReadyBT.Visible = false;
857                 FieldControllGB.Visible = false;
858 
859 
860                 PlayerInfoLB.Text = "Bereit";
861             }
862 
863             /// <summary>
864             /// Setzt die übrigen Schiffe zufällig
865             /// </summary>
866             /// <param name="sender"></param>
867             /// <param name="e"></param>
868             private void CompleteBT_Click(object sender, EventArgs e)
869             {
870                 FieldHandler.RandomShips();
871             }
872 
873             /// <summary>
874             /// Löscht das ganze Spielfeld
875             /// </summary>
876             /// <param name="sender"></param>
877             /// <param name="e"></param>
878             private void ClearBT_Click(object sender, EventArgs e)
879             {
880                 FieldHandler.ClearPlayer();
881             }
882 
883             /// <summary>
884             /// Ändert das Design des aktuellen Forms
885             /// </summary>
886             /// <param name="sender"></param>
887             /// <param name="e"></param>
888             private void ChangeDesignBT_Click(object? sender, EventArgs? e)
889             {
890                 // Speichert die Farben des Buttons im aktuellen Design
891                 Color btBackColor = Theme.GetThemeColor(ChangeDesignBT.BackColor);
892                 Color btForeColor = Theme.GetThemeColor(ChangeDesignBT.ForeColor);
893                 // Wechselt das Design
894                 Theme.ChangeMode();
```

```
895                 Theme.RefreshTheme(this);
896                 ChangeDesignBT.Text = Theme.IsDarkMode() ? "Light Mode" : "Dark Mode";
897                 // Setzt die Farben des Buttons auf die Farben des alten Designs um das
                    andere Design zu zeigen
898                 ChangeDesignBT.BackColor = btBackColor;
899                 ChangeDesignBT.ForeColor = btForeColor;
900             }
901         }
902     }
903     {   //[MainGameScripts/FieldHandler.cs]
904         public class FieldHandler
905         {
906             // Die Felder
907             private Field playerField;
908             private Field enemyField;
909 
910             // Die Schiffgrids
911             private ShipGrid playerShipGrid;
912             private ShipGrid enemyShipGrid;
913 
914             private MainGame mainGame;
915 
916             private EnemyPlayer enemyPlayer;
917 
918             public bool GameEnded { get; private set; } = false;
919             public bool IsReady { get; private set; } = false;
920 
921             private bool mouseDown = false;
922             private bool isPlayerTurn;
923 
924 
925             public Size FieldSize { get => playerField.FieldSize; }
926 
927             /// <summary>
928             /// Für die Zufallspalzierung
929             /// </summary>
930             public IEnumerable<int> FieldConfig { get => playerShipGrid.GetShipsAsInt(); }
931 
932             /// <summary>
933             /// Ändert alle nötigen Werte für den Spieler, auch in der UI
934             /// </summary>
935             public bool IsPlayerTurn { get => isPlayerTurn; set { if (GameEnded) return;
                    mainGame.SetTurn(value); enemyField.DoAllowUserinput(value); isPlayerTurn =
                    value;  }  }
936 
937 
938             public FieldHandler(Field _playerField, Field _enemyField, ShipGrid
                    _playerShipGrid, ShipGrid _enemyShipGrid, MainGame _mainGame, TcpClient?
                    connection)
939             {
940                 playerField = _playerField;
941                 enemyField = _enemyField;
942 
943                 playerShipGrid = _playerShipGrid;
944                 enemyShipGrid = _enemyShipGrid;
945 
946                 mainGame = _mainGame;
947 
948                 // Wenn eine Verbindung besteht, dann ist der Gegner ein Netzwerkspieler,
                    ansonsten ein Computerspieler
949                 if (connection != null && connection.Connected)
950                     enemyPlayer = new NetworkPlayer(this, connection);
951                 else
952                     enemyPlayer = new ComputerPlayer(this);
953             }
954 
955             /// <summary>
956             /// Setzt den Spieler auf Ready
957             /// Erlaubt dem Spieler nicht mehr seine Schiffe zu bewegen
958             /// Gibt dem Gegner bescheid, dass der Spieler bereit ist
959             /// </summary>
```

```
960         public void Ready()
961         {
962             playerField.DoAllowUserinput(false);
963             IsReady = true;
964             enemyPlayer.Ready();
965             playerShipGrid.ResetAmount();
966             enemyShipGrid.ResetAmount();
967         }
968 
969         /// <summary>
970         /// Überprüft ob alle Schiffe gesetzt wurden
971         /// </summary>
972         /// <returns>true wenn alle auf dem Feld sind/returns>
973         public bool PlayerShipAmountEmpty()
974         {
975             return playerShipGrid.ShipAmountEmpty();
976         }
977 
978         /// <summary>
979         /// Lösche alle Auswahlmarkierungen
980         /// </summary>
981         public void ClearFields()
982         {
983             playerField.ResetSelection();
984             enemyField.ResetSelection();
985 
986             playerShipGrid.ResetSelection();
987             enemyShipGrid.ResetSelection();
988         }
989 
990         /// <summary>
991         /// Setzt alle nötigen Eventlisteners
992         /// </summary>
993         private void start()
994         {
995             // Wenn der Spieler auf ein Feld klickt oder mit der Maus drüberfährt
996             playerField.FieldGrid.CellMouseDown += (sender, e) => {
                playerCellHoverPress(sender, new DataGridViewCellEventArgs(e.ColumnIndex,
                e.RowIndex)); };
997             playerField.FieldGrid.CellMouseEnter += playerCellHoverPress;
998 
999             // Wenn die Maus gedrückt oder losgelassen wird
1000            playerField.FieldGrid.MouseDown += (sender, e) => { mouseDown = true; };
1001            playerField.FieldGrid.MouseUp += (sender, e) => { mouseDown = false; };
1002
1003            // Wenn der Spieler auf ein Gegnderfeld klickt
1004            enemyField.FieldGrid.CellMouseClick += enemyCellPress;
1005            enemyField.HoverEvent += enemyPlayer.OnEnemyFieldHover;
1006            enemyField.UnHoverEvent += enemyPlayer.OnEnemyFieldUnHover;
1007
1008            // Damit nicht auf dem Gegnerfeld geklickt werden kann
1009            enemyField.DoAllowUserinput(false);
1010        }
1011
1012        /// <summary>
1013        /// Wenn das Fenster geladen hat
1014        /// </summary>
1015        public void Loaded()
1016        {
1017            start();
1018
1019            // Es werden alle Auswahlen zurückgesetzt
1020            ClearFields();
1021
1022            // Für Computer Gegner damit er direkt Ready gehen kann.
1023            enemyPlayer.OnLoad();
1024        }
1025        /// <summary>
1026        /// Setzt ein Feld auf dem Spielerfeld
1027        /// </summary>
1028        /// <param name="sender"></param>
```

```
1029            /// <param name="e"></param>
1030            private void playerCellHoverPress(object? sender, DataGridViewCellEventArgs e)
1031            {
1032                // Wenn der Spieler auf ein Feld klickt
1033                // Wenn die Maus gedrückt, das Feld nicht die Zahlen und Buchstaben ist
                    und das Feld geändert wurde
1034                if ( mouseDown && e.ColumnIndex > 0 && e.RowIndex > 0 && playerField.
                    Update(e.ColumnIndex - 1, e.RowIndex - 1))
1035                {
1036                    // Aktualisiere die Anzahl der Schiffe in ShipGrid
1037                    playerShipGrid.UpdateAmount(playerField.PlayingField.Links.ToArray());
1038                    playerField.ResetSelection();
1039                }
1040            }
1041            /// <summary>
1042            /// Setzt ein Feld auf dem Gegnerfeld
1043            /// </summary>
1044            /// <param name="sender"></param>
1045            /// <param name="e"></param>
1046            private void enemyCellPress(object? sender, DataGridViewCellMouseEventArgs e)
1047            {
1048                // Wenn der Spieler auf ein Gegnerfeld klickt
1049                // Wenn der Spieler am Zug ist, das Feld nicht die Zahlen und Buchstaben
                    ist und das Feld getroffen wurde
1050                if (IsPlayerTurn && e.ColumnIndex > 0 && e.RowIndex > 0 && enemyField.Hit(
                    e.ColumnIndex - 1, e.RowIndex - 1).HasValue)
1051                {
1052                    // OnChange ist eine Funktion die für die Gegner antworten zuständig
                        ist
1053                    IsPlayerTurn = false;
1054                    // Sende dem Gegner eine Nachricht das ein Feld getroffen wurde
1055                    _ = enemyPlayer.OnEnemyFieldHit(enemyField.PlayingField[e.ColumnIndex
                        - 1, e.RowIndex - 1]).ContinueWith(x =>
1056                    {
1057                        // Der Gegner antwortet mit einem HitType, was es für ein Treffer
                            war
1058                        switch (x.Result)
1059                        {
1060                            case HitType.HIT:
1061                                enemyField.Update(e.ColumnIndex - 1, e.RowIndex - 1);
1062                                break;
1063                            case HitType.SUNKEN:
1064                                enemyField.Update(e.ColumnIndex - 1, e.RowIndex - 1);
1065                                enemyField.ShipSunken(enemyField.PlayingField[e.
                                    ColumnIndex - 1, e.RowIndex - 1]);
1066                                EnemyShipSunken();
1067                                break;
1068                            case HitType.ENDGAME:
1069                                enemyField.Update(e.ColumnIndex - 1, e.RowIndex - 1);
1070                                enemyField.ShipSunken(enemyField.PlayingField[e.
                                    ColumnIndex - 1, e.RowIndex - 1]);
1071                                EndGame(true);
1072                                break;
1073                        }
1074                    });
1075                    enemyField.ResetSelection();
1076                }
1077            }
1078
1079            /// <summary>
1080            /// Beendet das Spiel und zeigt die übrigen Schiffe des Gegners an
1081            /// </summary>
1082            /// <param name="result"></param>
1083            private async void EndGame(bool result)
1084            {
1085                mainGame.EndGame(result);
1086                GameEnded = true;
1087
1088                // Holt sich alle Schiffe die noch nicht getroffen wurden
1089                IEnumerable<Point> res = await enemyPlayer.GetNonhitShips(playerField.
                    PlayingField.GetNonhitShipTiles().Select(t => t.Position));
```

```
1090
1091            // Blinkt die Schiffe, die noch nicht getroffen wurden bis das Fenster
                geschlossen wird
1092            while (true)
1093            {
1094                foreach (Point p in res)
1095                    enemyField.Update(p.X, p.Y);
1096                await Task.Delay(700);
1097            };
1098
1099        }
1100
1101        /// <summary>
1102        /// Plaziere alle Schiffe zufällig und aktualisiere die Anzahl der Schiffe in
            ShipGrid
1103        /// </summary>
1104        public void RandomShips()
1105        {
1106            playerField.Random(playerShipGrid.GetShipsAsInt(true));
1107            playerShipGrid.UpdateAmount(playerField.PlayingField.Links.ToArray());
1108        }
1109
1110        #region EnemyPlayer
1111
1112        /// <summary>
1113        /// Für den Gegner um einen Treffer zu setzen
1114        /// </summary>
1115        /// <param name="columnIndex"></param>
1116        /// <param name="rowIndex"></param>
1117        /// <returns>Was für ein Treffertyp der Treffer war</returns>
1118        public HitType PlayerHit(int columnIndex, int rowIndex)
1119        {
1120            // Wenn das Feld getroffen wurde
1121
1122            HitType hitType = playerField.Hit(columnIndex, rowIndex).GetValueOrDefault
                (HitType.HIT);
1123            if (hitType == HitType.SUNKEN || hitType == HitType.ENDGAME)
1124            {
1125                PlayerShipSunken();
1126            }
1127            if (hitType == HitType.ENDGAME)
1128            {
1129                EndGame(false);
1130                return hitType;
1131            }
1132            IsPlayerTurn = true;
1133
1134            return hitType;
1135        }
1136
1137        /// <summary>
1138        /// Setzt den Gegner auf Ready
1139        /// </summary>
1140        internal void EnemyReady()
1141        {
1142            mainGame.EnemyReady();
1143        }
1144
1145        /// <summary>
1146        /// Für Multiplayer
1147        /// Um anzuzeigen, wo der Gegner gerade mit der Maus ist
1148        /// </summary>
1149        /// <param name="columnIndex"></param>
1150        /// <param name="rowIndex"></param>
1151        internal void HoverCellPlayer(int columnIndex, int rowIndex)
1152        {
1153            playerField.ResetSelection();
1154            playerField.AddSelection(columnIndex, rowIndex);
1155        }
1156
1157        /// <summary>
```

```
1158            /// Für Multiplayer
1159            /// Um anzuzeigen, wo der Gegner gerade mit der Maus ist
1160            /// </summary>
1161            internal void HoverCellPlayerEnd()
1162            {
1163                playerField.ResetSelection();
1164            }
1165            #endregion
1166
1167            /// <summary>
1168            /// Ein Spielerschiff wurde versenkt
1169            /// Aktualisiere die Anzahl der Schiffe in PlayerShipGrid
1170            /// </summary>
1171            public void PlayerShipSunken()
1172            {
1173                // Filtert alle Tiles die getroffen wurden in ein Array
1174                playerShipGrid.UpdateAmount(playerField.PlayingField.Links.Where(l => l.
                    IsHit).ToArray());
1175            }
1176
1177            /// <summary>
1178            /// Ein Gegnerschiff wurde versenkt
1179            /// Aktualisiere die Anzahl der Schiffe in EnemyShipGrid
1180            /// </summary>
1181            public void EnemyShipSunken()
1182            {
1183                // Filtert alle Tiles die getroffen wurden in ein Array
1184                enemyShipGrid.UpdateAmount(enemyField.PlayingField.Links.Where(l => l.
                    IsHit).ToArray());
1185            }
1186
1187            /// <summary>
1188            /// Löscht das Spielerfeld und setzt die Anzahl der Schiffe in PlayerShipGrid
                zurück
1189            /// </summary>
1190            public void ClearPlayer()
1191            {
1192                playerField.Clear();
1193                playerShipGrid.ResetAmount();
1194            }
1195
1196            /// <summary>
1197            /// Versteckt das Spielerfeld
1198            /// </summary>
1199            /// <param name="isChecked">true bedeutet verstecken</param>
1200            internal void HidePlayerField(bool isChecked)
1201            {
1202                // Verstecke das PlayerField
1203                playerField.FieldGrid.Visible = !isChecked;
1204            }
1205
1206
1207            /// <summary>
1208            /// Muss vor Themenwechsel aufgerufen werden um das Blinken zu stoppen
1209            /// -> Somit verhindern das etwas nicht die richtige Farbe hat
1210            /// </summary>
1211            internal void ChangeTheme()
1212            {
1213                playerField.CancelBlink();
1214                enemyField.CancelBlink();
1215            }
1216        }
1217    }
1218    {   //[MainGameScripts/Field.cs]
1219        using System.Diagnostics;
1220        using System.Xml.Serialization;
1221
1222        namespace GFS_Spiel.MainGameScripts
1223        {
1224        public class Field
1225        {
```

```
1226            public PlayingField PlayingField { get; private set; }
1227            public DataGridView FieldGrid { get; }
1228            public Size FieldSize { get; }
1229
1230            private Tile? lastHitTile;
1231
1232            #region Events
1233            public delegate void HoverEventHandler(object? sender, Point e);
1234            public delegate void UnHoverEventHandler(object? sender, Point e);
1235
1236            public event HoverEventHandler? HoverEvent;
1237            public event UnHoverEventHandler? UnHoverEvent;
1238            #endregion
1239
1240            /// <summary>
1241            /// CancellationTokenSource um das Blinken des letzten getroffenen Feldes
                zwischenzustoppen
1242            /// </summary>
1243            private CancellationTokenSource cancellationTokenSource = new();
1244
1245            public Field(DataGridView dataGridView, Size? size = null)
1246            {
1247                if (size == null) FieldSize = new Size(10, 10);
1248
1249                FieldGrid = dataGridView;
1250                PlayingField = new(FieldSize);
1251
1252                //Für Hover
1253                FieldGrid.CellMouseEnter += mouseEnter;
1254                FieldGrid.CellMouseClick += mouseClick;
1255                FieldGrid.CellMouseLeave += mouseLeave;
1256
1257                genField(FieldSize.Width, FieldSize.Height);
1258                blinkLastHit();
1259            }
1260
1261            /// <summary>
1262            /// Stoppt das Blinken des letzten getroffenen Feldes
1263            /// </summary>
1264            public void CancelBlink()
1265            {
1266                cancellationTokenSource.Cancel();
1267            }
1268
1269            /// <summary>
1270            /// Blinkt das letzte getroffene Feld
1271            /// </summary>
1272            private async void blinkLastHit()
1273            {
1274                while (true)
1275                {
1276                    // Wenn das letzte getroffene Feld nicht null ist, dann blinkt es
1277                    if (lastHitTile is not null)
1278                    {
1279                        // Speichert das DataGridViewCell des letzten getroffenen Feldes
1280                        DataGridViewCell dc = FieldGrid[lastHitTile.Position.X + 1,
                            lastHitTile.Position.Y + 1];
1281                        // Speichert die Farbe des letzten getroffenen Feldes
1282                        Color prevColor = dc.Style.BackColor == Color.Empty ? FieldGrid.
                            DefaultCellStyle.BackColor : dc.Style.BackColor;
1283                        // Berechnet die neue Farbe
1284                        // Wenn die Farbe hell ist wird sie dunkler, wenn sie dunkel ist
                            wird sie heller
1285                        float f = prevColor.GetBrightness() > (Theme.IsDarkMode() ? 0.5f:
                            0.51f) ? 0.6f : 2f;
1286                        Color newColor = Theme.IsDarkMode() ? Color.FromArgb(255, Math.Min
                            ((int)(prevColor.R * f), 255), prevColor.G, Math.Min((int)(
                            prevColor.B * f), 255)) : Color.FromArgb(255, prevColor.R, Math.
                            Min((int)(prevColor.G * f), 255), prevColor.B);
1287                        // Setzt die neue Farbe
1288                        dc.Style.BackColor = newColor;
```

```
                    try
                    {
                        // Wartet 500ms oder bis das Blinken abgebrochen wird
                        await Task.Delay(500, cancellationTokenSource.Token);
                    }
                    catch (TaskCanceledException)
                    {
                        // Erstellt eine neue CancellationTokenSource
                        cancellationTokenSource = new();
                    }
                    // Setzt die Farbe zurück
                    if (dc.Style.BackColor == newColor)
                        // Setzt sie nur zurück wenn sie nach den 500ms nicht von was anderem geändert wurde
                        dc.Style.BackColor = prevColor;
                }
                await Task.Delay(500);
            }
        }

        /// <summary>
        /// Ob das Feld für den Spieler anklickbar ist
        /// </summary>
        /// <param name="value"></param>
        public void DoAllowUserinput(bool value)
        {
            FieldGrid.Enabled = value;
        }

        /// <summary>
        /// Wenn die Maus in ein Feld geht, dann wird das Feld als ausgewählt markiert
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void mouseEnter(object? sender, DataGridViewCellEventArgs e)
        {
            if (e.ColumnIndex > 0 && e.RowIndex > 0)
            {
                AddSelection(e.ColumnIndex - 1, e.RowIndex - 1);
            }

        }

        /// <summary>
        /// Wenn die Maus aus einem Feld geht, dann wird die Auswahl des Feldes aufgehoben
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void mouseClick(object? sender, DataGridViewCellMouseEventArgs e)
        {
            if (e.ColumnIndex > 0 && e.RowIndex > 0)
                RemoveSelection(e.ColumnIndex - 1, e.RowIndex - 1);
        }

        /// <summary>
        /// Wenn die Maus aus einem Feld geht, dann wird die Auswahl des Feldes aufgehoben
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void mouseLeave(object? sender, DataGridViewCellEventArgs e)
        {
            if (e.ColumnIndex > 0 && e.RowIndex > 0)
            {
                RemoveSelection(e.ColumnIndex - 1, e.RowIndex - 1);
            }

        }

        /// <summary>
```

```
1357            /// Fügt die Auswahl eines Feldes hinzu
1358            /// </summary>
1359            /// <param name="columnIndex"></param>
1360            /// <param name="rowIndex"></param>
1361            public void AddSelection(int columnIndex, int rowIndex)
1362            {
1363                FieldGrid[columnIndex + 1, rowIndex + 1].Selected = true;
1364                HoverEvent?.Invoke(this, PlayingField[columnIndex, rowIndex].Position);
1365            }
1366
1367            /// <summary>
1368            /// Löscht die Auswahl eines Feldes
1369            /// </summary>
1370            /// <param name="columnIndex"></param>
1371            /// <param name="rowIndex"></param>
1372            public void RemoveSelection(int columnIndex, int rowIndex)
1373            {
1374                FieldGrid[columnIndex + 1, rowIndex + 1].Selected = false;
1375                UnHoverEvent?.Invoke(this, PlayingField[columnIndex, rowIndex].Position);
1376            }
1377
1378            /// <summary>
1379            /// Löscht die Auswahl aller Felder
1380            /// </summary>
1381            public void ResetSelection()
1382            {
1383                FieldGrid.ClearSelection();
1384            }
1385
1386            /// <summary>
1387            /// Generiert ein Spielfeld mit der gegebenen größe
1388            /// </summary>
1389            /// <param name="sizex"></param>
1390            /// <param name="sizey"></param>
1391            private void genField(int sizex, int sizey)
1392            {
1393
1394                // +1 Wegen Zahlen und Buchstaben
1395                sizex++;
1396                sizey++;
1397
1398                // Generiere alle Spalten und passt die größe der Spalten an
1399                for (int i = 0; i < sizex; i++)
1400                {
1401                    FieldGrid.Columns.Add("Column" + i, i.ToString());
1402
1403                    FieldGrid.Columns[i].Width = FieldGrid.Size.Width / sizex;
1404                }
1405
1406                // Generiere alle Reihen und passt die größe der Reihen an
1407                for (int i = 0; i < sizey; i++)
1408                {
1409                    FieldGrid.Rows.Add();
1410
1411                    FieldGrid.Rows[i].Height = FieldGrid.Size.Height / sizey;
1412                }
1413
1414                // Benenne die Reihen und Zahlen
1415                for (int i = 1; i < sizex; i++)
1416                {
1417                    FieldGrid.Rows[0].Cells[i].Value = ((char)(i + 64)).ToString();
1418
1419                    FieldGrid.Rows[i].Cells[0].Value = i.ToString();
1420                }
1421
1422                // Passe die größe des Felds auf die größe der Spalten und Reihen an
1423                FieldGrid.Size = new Size((int)(Math.Floor((decimal)FieldGrid.Size.Height
                / sizex) * sizex + 3), (int)(Math.Floor((decimal)FieldGrid.Size.Width /
                sizey) * sizey + 3));
1424
1425                // Mache die 1 Spalte und Reihe Fett
```

```
1426            FieldGrid.Rows[0].DividerHeight = 2;
1427            FieldGrid.Columns[0].DividerWidth = 2;
1428        }
1429
1430        /// <summary>
1431        /// Wenn der zustand des Feldes sich wechselt (Schiff oder kein Schiff)
1432        /// Versuche das Feld in PlayerField zu aktualiesiern
1433        /// </summary>
1434        /// <param name="columnIndex"></param>
1435        /// <param name="rowIndex"></param>
1436        /// <returns>true wenn er sich geöndert hat false wenn nicht</returns>
1437        public bool Update(int columnIndex, int rowIndex)
1438        {
1439            Tile t = PlayingField[columnIndex, rowIndex];
1440            bool? isShip = PlayingField.Update(t);
1441
1442            if (isShip == null) return false;
1443
1444            // Wenn das Feld jetzt ein Schiff ist, dann ändere die Farbe
1445            updateColor(columnIndex, rowIndex, isShip);
1446
1447            return true;
1448        }
1449
1450        /// <summary>
1451        /// Markiert alle Felder des Schiffes als versenkt
1452        /// </summary>
1453        /// <param name="tile"></param>
1454        public void ShipSunken(Tile tile)
1455        {
1456            // Kennzeichne alle Felder des Schiffes als versenkt
1457            foreach (Tile t in tile.Link!.Tiles)
1458            {
1459                updateColor(t.Position.X, t.Position.Y, true, true);
1460                // Ändert auch die Farbe der Nachbarn, da hier keine Schiffe mehr
                    sein können
1461                foreach (Tile n in t.GetNeighbours())
1462                {
1463                    n.Hit();
1464                    updateText(n.Position.X, n.Position.Y, true);
1465                }
1466            }
1467        }
1468
1469        /// <summary>
1470        /// Aktualisiert die Farbe des Feldes je nachdem was das Feld ist
1471        /// </summary>
1472        /// <param name="columnIndex"></param>
1473        /// <param name="rowIndex"></param>
1474        /// <param name="isShip"></param>
1475        /// <param name="isSunken"></param>
1476        private void updateColor(int columnIndex, int rowIndex, bool? isShip, bool
            isSunken = false)
1477        {
1478            switch (isShip)
1479            {
1480                case true:
1481                    FieldGrid[columnIndex + 1, rowIndex + 1].Style.BackColor = Theme.
                        GetThemeColor(isSunken ? SystemColors.HotTrack : SystemColors.
                        HighlightText);
1482                    break;
1483
1484                case false:
1485                    FieldGrid[columnIndex + 1, rowIndex + 1].Style.BackColor =
                        FieldGrid.DefaultCellStyle.BackColor;
1486                    break;
1487            }
1488        }
1489        /// <summary>
1490        /// Aktualisiert den Text des Feldes je nachdem ob das Feld getroffen wurde
            oder nicht
```

```
1491            /// </summary>
1492            /// <param name="columnIndex"></param>
1493            /// <param name="rowIndex"></param>
1494            /// <param name="isShip"></param>
1495            /// <param name="isSunken"></param>
1496            private void updateText(int columnIndex, int rowIndex, bool isHit)
1497            {
1498                switch (isHit)
1499                {
1500                    case true:
1501                        FieldGrid[columnIndex + 1, rowIndex + 1].Value = "X";
1502                        break;
1503                    case false:
1504                        FieldGrid[columnIndex + 1, rowIndex + 1].Value = "";
1505                        break;
1506                }
1507            }
1508
1509            /// <summary>
1510            /// Versucht das Feld als getroffen zu markieren
1511            /// </summary>
1512            /// <param name="columnIndex"></param>
1513            /// <param name="rowIndex"></param>
1514            /// <returns>
1515            /// null -> Feld war schon getroffen.
1516            /// <see cref="HitType"/> gibt an was getroffen wurde
1517            /// </returns>
1518            public HitType? Hit(int columnIndex, int rowIndex)
1519            {
1520                // Wenn das Feld Getroffen wurde, dann ändere Text
1521
1522                Tile tile = PlayingField[columnIndex, rowIndex];
1523
1524                if (tile.IsHit) return null;
1525                lastHitTile = tile;
1526                tile.Hit();
1527
1528                updateText(columnIndex, rowIndex, true);
1529                //hitAnimation();
1530
1531                // Falls nun alle Felder des Schiffes getroffen wurden, dann kennzeichne
                    das Schiff als versenkt
1532                if (tile.Link != null)
1533                {
1534                    if (tile.Link.IsHit)
1535                    {
1536                        ShipSunken(tile);
1537                        if (PlayingField.AllShipsHit())
1538                            return HitType.ENDGAME;
1539                        return HitType.SUNKEN;
1540                    }
1541                    return HitType.HIT;
1542                }
1543                return HitType.MISS;
1544            }
1545
1546            /// <summary>
1547            /// Versucht ein zufälliges Feld zu generieren
1548            /// </summary>
1549            /// <param name="ships">Schiffe wie in <see cref="ShipGrid.GetShipsAsInt"/></
                param>
1550            /// <param name="useCurrent">Ob er versuchen soll die schiffe auf das
                vorhandene Feld zu plazieren</param>
1551            /// <returns>Ob es Funktioniert hat</returns>
1552            public bool Random(IEnumerable<int> ships, bool useCurrent = true)
1553            {
1554                // Generiere ein neues Spielfeld mit den gegebenen Schiffen
1555                PlayingField? result = useCurrent ? PlayingField.Random(ships) :
                    PlayingField.Random(FieldSize, ships);
1556
1557                // Wenn das Spielfeld nicht generiert werden konnte, dann breche ab
```

```
            if (result == null) return false;

            PlayingField = result;

            // Aktualisiere die Farbe der Felder
            for (int i = 0; i < FieldSize.Width; i++)
            {
                for (int j = 0; j < FieldSize.Height; j++)
                {
                    updateColor(i, j, PlayingField[i, j].Link != null);
                }
            }
            return true;
        }

        /// <summary>
        /// Setzt das Spielfeld auf leer zurück
        /// </summary>
        public void Clear()
        {
            // Lösche das Spielfeld
            PlayingField = new PlayingField(FieldSize);

            for (int i = 0; i < FieldSize.Width; i++)
            {
                for (int j = 0; j < FieldSize.Height; j++)
                {
                    updateColor(i, j, false);
                }
            }
        }

        /// <summary>
        /// Gibt an Was getroffen wurde
        /// </summary>
        public enum HitType
        {
            /// <summary>
            /// Ein Schiffteil wurde getroffen
            /// </summary>
            HIT,
            /// <summary>
            /// Das Wasser wurde getroffen
            /// </summary>
            MISS,
            /// <summary>
            /// Ein Schiff wurde versenkt
            /// </summary>
            SUNKEN,
            /// <summary>
            /// Alle Schiffe wurden versenkt
            /// </summary>
            ENDGAME
        }
    }
}

}
{   //[MainGameScripts/PlayingField.cs]
    public class PlayingField
    {
        public readonly Size Size;
        public Tile[,] Tiles { get; }

        public readonly List<Link> Links;

        public PlayingField(Size _size)
        {
            Size = _size;
            Tiles = new Tile[Size.Width, Size.Height];
            Links = new();
```

```
1629            genTiles(Size);
1630        }
1631
1632        /// <summary>
1633        /// Erstellt ein zufälliges Spielfeld
1634        /// </summary>
1635        /// <param name="size">Die größe des Feldes</param>
1636        /// <param name="ships">Die anzahl von den schiffen, wie in ShiipGrid
            erklärt</param>
1637        /// <param name="maxAttempts">Wie viele Versuche er machen soll ein Feld
            zufällig erstellen</param>
1638        /// <returns></returns>
1639        public static PlayingField? Random(Size size, IEnumerable<int> ships, int
            maxAttempts = 10)
1640        {
1641            // Erstelle ein neues Spielfeld und versuche zufällig Schiffe zu
                platzieren
1642            return new PlayingField(size).Random(ships, maxAttempts);
1643        }
1644
1645        /// <summary>
1646        /// Gibt das <see cref="Tile"/> an der Position zurück
1647        /// </summary>
1648        /// <param name="x"></param>
1649        /// <param name="y"></param>
1650        /// <returns></returns>
1651        public Tile this[int x, int y]
1652        {
1653            get => Tiles[x, y];
1654        }
1655
1656        /// <summary>
1657        /// Gibt das <see cref="Tile"/> an der Position zurück
1658        /// </summary>
1659        /// <param name="pos"></param>
1660        /// <returns></returns>
1661        public Tile this[Point pos]
1662        {
1663            get => Tiles[pos.X, pos.Y];
1664        }
1665
1666        /// <summary>
1667        /// {Left, Bottom, Right, Top} array of distances from the point to the next
            hit
1668        /// </summary>
1669        /// <param name="p"></param>
1670        /// <returns></returns>
1671        public int[] DistancesToNext(Point p)
1672        {
1673            List<int> distances = new();
1674            for (int i = -1; i <= 1; i += 2)
1675            {
1676                int y;
1677                for (y = p.Y; y >= 0 && y < Size.Height; y += 1 * i)
1678                {
1679                    if (Tiles[p.X, y].IsHit)
1680                    {
1681                        distances.Add(Math.Abs(y - p.Y) - 1);
1682                        break;
1683                    }
1684                }
1685                if (!(y >= 0 && y < Size.Height)) distances.Add(Math.Abs(y - p.Y) - 1
                    );
1686                int x;
1687                for (x = p.X; x >= 0 && x < Size.Width; x += 1 * i)
1688                {
1689                    if (Tiles[x, p.Y].IsHit)
1690                    {
1691                        distances.Add(Math.Abs(x - p.X)-1);
1692                        break;
1693                    }
```

```
                }
                if (!(x >= 0 && x < Size.Width)) distances.Add(Math.Abs(x - p.X) - 1);
            }
            return distances.ToArray();
        }

        /// <summary>
        /// Gibt alle Tiles zurück, die nicht getroffen wurden und ein Schiff
        enthalten
        /// </summary>
        /// <returns></returns>
        public IEnumerable<Tile> GetNonhitShipTiles()
        {
            return Tiles.Cast<Tile>().Where(t => !t.IsHit && t.Link != null);
        }

        /// <summary>
        /// Gibt ein zufälliges leeres Feld zurück
        /// </summary>
        /// <param name="random"></param>
        /// <returns></returns>
        private Tile? randomEmptyTile(Random random)
        {
            // Gehe alle Felder durch und suche ein leeres Feld ohne Nachbarn
            int rx = random.Next(0, Size.Width);
            int ry = random.Next(0, Size.Height);

            for (int i = 0; i < Size.Width; i++)
            {
                for (int j = 0; j < Size.Height; j++)
                {
                    int x = (rx + i) % Size.Width;
                    int y = (ry + j) % Size.Height;

                    if (Tiles[x, y].CountNeighbours() == 0)
                    {
                        return Tiles[x, y];
                    }
                }
            }

            return null;
        }

        /// <summary>
        /// Generiert alle Tiles im Spielfeld
        /// </summary>
        /// <param name="size"></param>
        private void genTiles(Size size)
        {
            // Erstelle ein neues Spielfeld
            for (int i = 0; i < size.Width; i++)
            {
                for (int j = 0; j < size.Height; j++)
                {
                    Tiles[i, j] = new Tile(i, j, this);
                }
            }
        }

        /// <summary>
        /// Versucht das Tile zu einem Link hinzuzufügen/entfernen und somit als
        ein/kein "schiff" zu setzen
        /// </summary>
        /// <param name="columnIndex"></param>
        /// <param name="rowIndex"></param>
        /// <returns></returns>
        public bool? Update(int columnIndex, int rowIndex)
        {
            // Gibt null zurück, wenn das Feld nicht geLinkt werden kann
            Tile tile = Tiles[columnIndex, rowIndex];
```

```
1763            return Update(tile);
1764        }
1765
1766        /// <summary>
1767        /// Versucht das Tile zu einem Link hinzuzufügen/entfernen und somit als
            ein/kein "schiff" zu setzen
1768        /// </summary>
1769        /// <param name="tile"></param>
1770        /// <returns>
1771        /// false   -> wenn es geUnLinkt wurde
1772        /// true    -> wenn es geLinkt wurde
1773        /// null    -> wenn es nicht aktualisiert werden kann
1774        /// </returns>
1775        public bool? Update(Tile tile)
1776        {
1777            Link[]? links = tile.NeighbourLinks().ToArray();
1778
1779            // Wenn das Feld geLinked ist
1780            if (tile.Link != null)
1781            {
1782                // Entferne das Feld aus dem Link
1783                Link link = tile.Link;
1784                tile.Link.Remove(tile);
1785                tile.Link = null;
1786
1787                // Wenn der Link leer ist, entferne ihn
1788                if (link.Length == 0)
1789                {
1790                    Links.Remove(link);
1791                }
1792                // Wenn das Tile in der Mitte von einem Link ist
1793                if (links.Length == 2)
1794                {
1795                    // Lösche den link für alle Tiles
1796                    link.Tiles.ForEach((t) =>
1797                    {
1798                        t.Link = null;
1799                    });
1800                    // Update alle Tiles in dem alten Link
1801                    link.Tiles.ToList().ForEach((t) => Update(t));
1802
1803                    // Entferne den alten Link
1804                    Links.Remove(link);
1805                }
1806
1807                tile.Link = null;
1808
1809                return false;
1810            }
1811
1812            // Wenn Feld nicht geLinkt ist
1813            switch (links.Length)
1814            {
1815                //Erstelle neuen Link
1816                case 0:
1817                    // Wenn die Nachbarn leer sind, kann das Feld geLinkt werden
1818                    if (tile.CountNeighbours() != 0) return null;
1819                    // Erstelle neuen Link und füge hinzu
1820                    tile.Link = new Link();
1821                    tile.Link.Add(tile);
1822                    Links.Add(tile.Link);
1823                    break;
1824
1825                //Füge neues Tile zum vorhanden Link hinzu
1826                case 1:
1827                    // Wenn es nur ein Nachbar gibt und weniger als 6 im Link sind,
                        kann das Feld geLinkt werden
1828                    // Und wenn er die gleiche Richtung hat
1829                    if (tile.CountNeighbours() != 1) return null;
1830                    if (links[0].Length == 6) return null;
1831                    // Füge hinzu
```

```
                    tile.Link = links[0];
                    links[0].Add(tile);
                    break;

                //Verbinde die Links und füge hinzu
                case 2:
                    // Wenn es nur 2 NachbarTiles gibt
                    if (tile.CountNeighbours() != 2) return null;
                    // Und die Links zusammen nicht länger als 5 sind
                    if (links[0].Length + links[1].Length >= 6) return null;
                    // Und wenn sie die gleiche Richtung haben
                    if (!(links[0].Vertical == null || links[1].Vertical == null || (
                    links[1].Vertical == links[0].Vertical))) return null;
                    // Verbinde die Links
                    links[1].Tiles.ToList().ForEach((t) => { t.Link = links[0]; links[
                    0].Add(t); });
                    Links.Remove(links[1]);
                    // Füge hinzu
                    tile.Link = links[0];
                    tile.Link.Add(tile);
                    break;
            }
            return true;

        }

        /// <summary>
        /// Erstellt eine komplett neue Kopie des Spielfeldes
        /// </summary>
        /// <returns></returns>
        public PlayingField Clone()
        {
            // Clones the PlayingField
            PlayingField clone = new PlayingField(Size);
            // Clone all Tiles
            for (int i = 0; i < Size.Width; i++)
            {
                for (int j = 0; j < Size.Height; j++)
                {
                    if (Tiles[i, j].IsHit)
                        clone.Tiles[i, j].Hit();
                }
            }
            // Clone all Links
            foreach (Link link in Links)
            {
                Link copyLink = new Link();
                foreach (Tile tile in link.Tiles)
                {
                    clone.Tiles[tile.Position.X, tile.Position.Y].Link = copyLink;
                    copyLink.Add(clone.Tiles[tile.Position.X, tile.Position.Y]);
                }
                clone.Links.Add(copyLink);
            }
            return clone;
        }

        /// <summary>
        /// Versucht die Schiffe auf dem Spielfeld zufällig zu platzieren
        /// </summary>
        /// <param name="ships"></param>
        /// <param name="maxAttempts"></param>
        /// <returns></returns>
        public PlayingField? Random(IEnumerable<int> ships, int maxAttempts = 10)
        {
            // Zufällige Reihenfolge der Schiffe
            return TryPlaceShips(ships, this, maxAttempts, new Random());
        }

        /// <summary>
        /// Versucht die Schiffe durch Rekursion auf dem Spielfeld zufällig zu
```

```
        platzieren
1901    /// </summary>
1902    /// <param name="shipsRemaining"></param>
1903    /// <param name="board"></param>
1904    /// <param name="maxAttempts"></param>
1905    /// <param name="random"></param>
1906    /// <returns></returns>
1907    private PlayingField? TryPlaceShips(IEnumerable<int> shipsRemaining,
        PlayingField board, int maxAttempts, Random random)
1908    {
1909        int? shipToPlace = shipsRemaining.FirstOrDefault();
1910
1911        // Alle Schiffe platziert -> fertig
1912        if (shipToPlace == 0)
1913            return board;
1914
1915
1916        for (int attempts = 0; attempts < maxAttempts; attempts++)
1917        {
1918            PlayingField boardCopy = board.Clone();
1919
1920            Tile? pos = boardCopy.randomEmptyTile(random);
1921
1922            // Wenn kein Platz mehr frei ist -> abbrechen
1923            if (pos == null)
1924                return null;
1925
1926            // Zufällige Orientierung
1927            int orientation = random.Next(0, 2);
1928
1929            // Überprüfe ob Platz für Schiff
1930            bool b = false;
1931            for (int i = 0; i < shipToPlace; i++)
1932            {
1933                // weil größer werden könnte als sollte
1934                int x = pos.Position.X + i * orientation;
1935                int y = pos.Position.Y + i * (orientation ^ 1);
1936                if (x >= Size.Width || y >= Size.Height)
1937                {
1938                    b = true;
1939                    break;
1940                }
1941                Tile t = boardCopy[x, y];
1942                // Wenn ein Nachbar, außer das voherige existiert, dann ist kein
                    Platz
1943                if (t.CountNeighbours() > 1) { b = true; break; }
1944                boardCopy.Update(t);
1945            }
1946            // Wenn kein Platz -> nächster Versuch
1947            if (b) continue;
1948
1949            // Platz gefunden -> Schiff setzen
1950            // Nächtste Schiffe versuchen -> Rekursion
1951            PlayingField? nextBoard = TryPlaceShips(shipsRemaining.Skip(1),
                boardCopy, maxAttempts, random);
1952            if (nextBoard != null)
1953                return nextBoard;
1954        }
1955        return null;
1956    }
1957
1958    /// <summary>
1959    /// Gibt zurück ob alle Schiffe versenkt sind
1960    /// </summary>
1961    /// <returns>true wenn alle Schiffe versenkt</returns>
1962    public bool AllShipsHit()
1963    {
1964        // Wenn alle Schiffe versenkt sind
1965        return Links.All((l) => l.IsHit);
1966    }
1967}
```

```
1968
1969        public class Tile
1970        {
1971            public readonly Point Position;
1972
1973            public Link? Link { get; set; }
1974            public bool IsHit { get; private set; }
1975
1976            private PlayingField field;
1977
1978            public Tile(int x, int y, PlayingField pField, Link? link = null, bool isHit =
                 false)
1979            {
1980                Position = new(x, y);
1981
1982                Link = link;
1983                IsHit = isHit;
1984                field = pField;
1985            }
1986
1987            /// <summary>
1988            /// Setzt das Feld als getroffen
1989            /// </summary>
1990            public void Hit()
1991            {
1992                IsHit = true;
1993            }
1994            /// <summary>
1995            /// Gibt alle Nachbarn zurück, nach dem Schema:
1996            ///     + + +
1997            ///     + x +
1998            ///     + + +
1999            /// </summary>
2000            /// <param name="field"></param>
2001            /// <returns>Die benachbarten Tiles</returns>
2002            public IEnumerable<Tile> GetNeighbours()
2003            {
2004
2005                for (int i = -1; i <= 1; i++)
2006                {
2007                    for (int j = -1; j <= 1; j++)
2008                    {
2009                        // Wenn das Tile nicht außerhalb des Feldes liegt und nicht das
                            eigene Tile ist
2010                        if (Position.X + i >= 0 && Position.X + i < field.Size.Width &&
                            Position.Y + j >= 0 && Position.Y + j < field.Size.Height && !(j
                            == 0 && i == 0))
2011                        {
2012                            yield return field[Position.X + i, Position.Y + j];
2013                        }
2014                    }
2015                }
2016            }
2017            /// <summary>
2018            /// Gibt die anzahl der Nachbarn zurück, die ein Link haben
2019            /// </summary>
2020            /// <returns></returns>
2021            public int CountNeighbours()
2022            {
2023                return GetNeighbours().Count((t) => t.Link != null);
2024            }
2025            /// <summary>
2026            /// Gibt die Links der Nachbarn zurück
2027            ///       +
2028            ///     + x +
2029            ///       +
2030            /// Dies ist notwenig um zu überprüfen, wie ein feld geLinkt werden muss
                (siehe <see cref="PlayingField.Update"/>)
2031            /// </summary>
2032            /// <returns></returns>
2033            public IEnumerable<Link> NeighbourLinks()
```

```
2034            {
2035                foreach (Point p in new Point[] { new Point(0, -1), new Point(0, 1), new
                    Point(-1, 0 ), new Point(1, 0 ) })
2036                {
2037                    // Wenn das Tile nicht außerhalb des Feldes liegt und und ein Link
                        existiert
2038                    if (Position.X + p.X >= 0 && Position.X + p.X < field.Size.Width &&
                        Position.Y + p.Y >= 0 && Position.Y + p.Y < field.Size.Height && field
                        [Position.X + p.X, Position.Y + p.Y].Link != null)
2039                    {
2040                        yield return field[Position.X + p.X, Position.Y + p.Y].Link!;
2041                    }
2042                }
2043            }
2044        }
2045
2046        public class Link
2047        {
2048            public List<Tile> Tiles { get; private set; }
2049            public int Length => Tiles.Count;
2050
2051            /// <summary>
2052            /// true -> Vertikal
2053            /// false -> Horizontal
2054            /// null -> Länge 1
2055            /// </summary>
2056            public bool? Vertical => isVertical();
2057
2058            /// <summary>
2059            /// Alle Tiles sind getroffen
2060            /// </summary>
2061            public bool IsHit => Tiles.All((t) => t.IsHit);
2062
2063            /// <summary>
2064            /// Anzahl der getroffenen Tiles
2065            /// </summary>
2066            public int HitCount => Tiles.Count((t) => t.IsHit);
2067
2068            public Link()
2069            {
2070                Tiles = new List<Tile>();
2071            }
2072
2073            /// <summary>
2074            /// Tile zum Link hinzufügen
2075            /// </summary>
2076            /// <param name="tile"></param>
2077            public void Add(Tile tile)
2078            {
2079                Tiles.Add(tile);
2080            }
2081
2082            /// <summary>
2083            /// Tile aus dem Link entfernen
2084            /// </summary>
2085            /// <param name="tile"></param>
2086            public void Remove(Tile tile)
2087            {
2088                Tiles.Remove(tile);
2089            }
2090
2091            /// <summary>
2092            /// true -> Vertikal
2093            /// false -> Horizontal
2094            /// null -> Länge 1
2095            /// </summary>
2096            /// <returns></returns>
2097            private bool? isVertical()
2098            {
2099                // Wenn länge 1 -> weder noch
2100                if (Length == 1) return null;
```

```
2101
2102                // Wenn 2 X-Werte gleich sind -> vertikal
2103                return Tiles[0].Position.X == Tiles[1].Position.X;
2104            }
2105        }
2106    }
2107    {   //[MainGameScripts/EnemyPlayers/EnemyPlayer.cs]
2108        public abstract class EnemyPlayer
2109        {
2110            private readonly FieldHandler fieldHandler;
2111
2112            private protected Size FieldSize { get => fieldHandler.FieldSize; }
2113            /// <summary>
2114            /// Siehe <see cref="ShipGrid.GetShipsAsInt(bool)"/>
2115            /// </summary>
2116            private protected IEnumerable<int> FieldConfig { get => fieldHandler.
                FieldConfig; }
2117            private protected bool IsPlayerTurn { get => fieldHandler.IsPlayerTurn; set =>
                 fieldHandler.IsPlayerTurn = value; }
2118            private protected bool IsPlayerReady { get => fieldHandler.IsPlayerReady; }
2119
2120            public EnemyPlayer(FieldHandler _fieldHandler)
2121            {
2122                fieldHandler = _fieldHandler;
2123            }
2124            /// <summary>
2125            /// Siehe <see cref="FieldHandler.PlayerHit"/>
2126            /// </summary>
2127            /// <param name="point"></param>
2128            /// <returns>Siehe <see cref="FieldHandler.PlayerHit"/></returns>
2129            private protected HitType HitPlayer(Point point) => fieldHandler.PlayerHit(
                point.X, point.Y);
2130            /// <summary>
2131            /// Siehe <see cref="FieldHandler.PlayerHit"/>
2132            /// </summary>
2133            /// <param name="point"></param>
2134            /// <returns>Siehe <see cref="FieldHandler.PlayerHit"/></returns>
2135            private protected HitType HitPlayer(int columnIndex, int rowIndex) =>
                fieldHandler.PlayerHit(columnIndex, rowIndex);
2136
2137            /// <summary>
2138            /// Funktion die aufgerufen werden muss wenn der Gegner bereit ist
2139            /// </summary>
2140            private protected void EnemyReady() => fieldHandler.EnemyReady();
2141
2142            /// <summary>
2143            /// Markiert die Zelle auf dem Spielerfeld
2144            /// </summary>
2145            /// <param name="columnIndex"></param>
2146            /// <param name="rowIndex"></param>
2147            private protected void HoverCell(int columnIndex, int rowIndex) =>
                fieldHandler.HoverCellPlayer(columnIndex, rowIndex);
2148
2149            /// <summary>
2150            /// Löscht die Markierung auf dem Spielerfeld
2151            /// </summary>
2152            private protected void HoverCellEnd() => fieldHandler.HoverCellPlayerEnd();
2153
2154            /// <summary>
2155            /// Wird vom <see cref="FieldHandler"/> aufgerufen wenn der Spieler bereit ist
2156            /// </summary>
2157            public abstract void OnPlayerReady();
2158
2159            /// <summary>
2160            /// Wird vom <see cref="FieldHandler"/> aufgerufen wenn das Spiel geladen hat
2161            /// </summary>
2162            public abstract void OnLoad();
2163
2164            /// <summary>
2165            /// Wird vom <see cref="FieldHandler"/> aufgerufen wenn der Spieler ein Feld
                getroffen hat
```

```
2166            /// </summary>
2167            /// <param name="point">Das </param>
2168            /// <returns></returns>
2169            public abstract Task<HitType> OnEnemyFieldHit(Point point);
2170
2171            /// <summary>
2172            /// Gibt alle Schiffe zurück, die noch nicht vom Spieler getroffen wurden
2173            /// </summary>
2174            /// <param name="leftShips">Ein <c>IEnmuerable</c>, von allen übrigen Schiffe
                des Spielers als <see cref="Point"/></param>
2175            /// <returns></returns>
2176            public abstract Task<IEnumerable<Point>> GetNonhitShips(IEnumerable<Point>
                leftShips);
2177
2178            /// <summary>
2179            /// Optional: Wird vom <see cref="FieldHandler"/> aufgerufen wenn der Spieler
                mit der Maus über eine GegnerZelle geht
2180            /// </summary>
2181            /// <param name="sender"></param>
2182            /// <param name="point"></param>
2183            public virtual void OnEnemyFieldHover(object? sender, Point point) { return ;
                }
2184            /// <summary>
2185            /// Optional: Wird vom <see cref="FieldHandler"/> aufgerufen wenn der Spieler
                die GegnerZelle verlassen hat
2186            /// </summary>
2187            /// <param name="sender"></param>
2188            /// <param name="point"></param>
2189            public virtual void OnEnemyFieldUnHover(object? sender, Point point) { return
                ; }
2190        }
2191    }
2192    {   //[MainGameScripts/EnemyPlayers/ComputerPlayer.cs]
2193        internal class ComputerPlayer : EnemyPlayer
2194        {
2195            private PlayingField playerPlayingField;
2196            private PlayingField enemyPlayingField;
2197
2198            private Tile lastTile;
2199            private bool lastTileShipSunken;
2200
2201            private Random random = new Random();
2202
2203            #pragma warning disable CS8618
2204            public ComputerPlayer(FieldHandler _fieldHandler) : base(_fieldHandler)
2205            #pragma warning restore CS8618
2206            {
2207                playerPlayingField = new PlayingField(FieldSize);
2208                enemyPlayingField = PlayingField.Random(FieldSize, FieldConfig)!;
2209            }
2210
2211            /// <summary>
2212            /// Trifft den Spieler und aktualisiert das Spielfeld des playerPlayingFields
                damit der Computer weis, wo er schon getroffen hat
2213            /// </summary>
2214            /// <param name="t"></param>
2215            private void hitPlayer(Tile t)
2216            {
2217                t.Hit();
2218                switch (HitPlayer(t.Position))
2219                {
2220                    case HitType.HIT:
2221                        playerPlayingField.Update(t.Position.X, t.Position.Y);
2222                        break;
2223                    case HitType.SUNKEN:
2224                        lastTileShipSunken = true;
2225                        playerPlayingField.Update(t.Position.X, t.Position.Y);
2226                        // Alle Felder um das Schiff herum werden als getroffen markiert
                            weil hier kein Schiff mehr sein kann
2227                        foreach (Tile ti in t.Link!.Tiles)
2228                        {
```

```
                    foreach (Tile n in ti.GetNeighbours())
                    {
                        n.Hit();
                    }
                }
                break;
            }
        }

        /// <summary>
        /// Führt einen Zug aus
        /// </summary>
        public async void Move()
        {
            await Task.Delay(random.Next(500, 1500));

            // wenn das letzte Feld ein Schiff war, dann wird das Schiff versucht zu versenken
            if (lastTile != null && lastTile.Link != null && !lastTileShipSunken)
            {
                if (lastTile.Link.HitCount == 1) // Der Computer weiß die Richtung noch nicht da nur ein Segement des Schiffes getroffen
                {
                    // Es wird eines der Felder um das Trefferfeld herum ausgewählt
                    foreach (Point p in new Point[] { new Point(0, -1), new Point(0, 1), new Point(1, 0), new Point(-1, 0) }.OrderBy(p => random.Next()))
                    {
                        // Wenn das Feld außerhalb des Spielfeldes liegt, wird es übersprungen
                        if (!(lastTile.Position.X + p.X >= 0 && lastTile.Position.X + p.X < enemyPlayingField.Size.Width && lastTile.Position.Y + p.Y >= 0 && lastTile.Position.Y + p.Y < enemyPlayingField.Size.Height))
                            continue;

                        // Wenn das Feld schon getroffen wurde, wird es übersprungen
                        Tile t = playerPlayingField[lastTile.Position.X + p.X, lastTile.Position.Y + p.Y];
                        if (t.IsHit) continue;

                        // Das Feld wird getroffen
                        hitPlayer(t);
                        break;
                    }

                }
                else
                {
                    // Es wird ein Feld ausgewählt, das noch nicht getroffen wurde
                    HashSet<Tile> possible = new HashSet<Tile>();
                    foreach (Tile t in lastTile.Link.Tiles.Where(t => t.IsHit)) // Für jedes getroffene Feld
                    {
                        // Es wird für jedes getroffene Feld ein Feld in der Richtung des Schiffes gesucht (+1 und -1)
                        for (int i = -1; i <= 1; i += 2)
                        {
                            int x = t.Position.X + (lastTile.Link.Vertical!.Value ? 0 : i);
                            int y = t.Position.Y + (lastTile.Link.Vertical.Value ? i : 0);

                            // Wenn das Feld außerhalb des Spielfeldes liegt, wird es übersprungen
                            if (!(x >= 0 && x < enemyPlayingField.Size.Width && y >= 0 && y < enemyPlayingField.Size.Height))
                                continue;
                            possible.Add(playerPlayingField[t.Position.X + ((bool)lastTile.Link.Vertical ? 0 : i), t.Position.Y + ((bool)lastTile.Link.Vertical ? i : 0)]);
```

```
                        }
                    }

                    // Es wird ein zufälliges Feld von den möglichen Feldern
                    ausgewählt
                    foreach (Tile t in possible.OrderBy(p => random.Next()))
                    {
                        if (t.IsHit) continue;

                        hitPlayer(t);
                        break;
                    }
                }
            }
            else
            {
                // Wenn das letzte Feld kein Schiff war, wird ein zufälliges Feld
                ausgewählt

                // Alle Felder die noch nicht getroffen wurden werden als mögliche
                Felder gespeichert
                List<Tile> possibleMoves = playerPlayingField.Tiles.Cast<Tile>().Where
                ((t) => !t.IsHit).ToList();

                // Die Felder werden nach der Entfernung zum nächsten getroffenen
                Feld sortiert und dann innerhalb der Entfernung versucht das
                mittigste auszuwählen und zufällig ausgewählt
                lastTile = possibleMoves.OrderBy(p =>
                {
                    int[] a = playerPlayingField.DistancesToNext(p.Position);
                    int vertical = a[0] + a[2];
                    int horizontal = a[1] + a[3];
                    return
                    FieldSize.Width - vertical +
                    FieldSize.Height - horizontal +
                    random.NextSingle() +
                    Math.Abs(a[0] - a[2]) / Math.Max(vertical, 1) +
                    Math.Abs(a[1] - a[3]) / Math.Max(horizontal, 1);
                }
                ).ElementAt(0);

                lastTileShipSunken = false;

                hitPlayer(lastTile);
            }
        }

        /// <summary>
        /// Entscheidet ob der Computer anfängt oder nicht
        /// </summary>
        public override void OnPlayerReady()
        {
            // Wenn beide Spieler bereit sind, wird ein zufälliger Spieler
            ausgewählt, der anfängt
            IsPlayerTurn = new Random().Next(0, 2) == 0;

            // Wenn der Computer anfängt, wird ein Zug gemacht
            if (!IsPlayerTurn)
                Move();
        }

        /// <summary>
        /// Setzt den Computer auf Bereit
        /// </summary>
        public override void OnLoad()
        {
            EnemyReady();
        }

        /// <summary>
        /// Gibt zurück was für ein Treffer der Spieler gemacht hat
```

```
2347            /// </summary>
2348            /// <param name="point"></param>
2349            /// <returns></returns>
2350            public override Task<HitType> OnEnemyFieldHit(Point point)
2351            {
2352                // Wenn der Computer an der Reihe ist, wird das Feld getroffen
2353                Tile t = enemyPlayingField[point];
2354                t.Hit();
2355
2356                // Wenn alle Schiffe versenkt wurden, wird das Spiel beendet
2357                if (enemyPlayingField.Links.All(l => l.IsHit))
2358                    return Task.FromResult(HitType.ENDGAME);
2359
2360                Move();
2361                // Wenn das Feld ein Schiff war, wird das Schiff versenkt
2362                if (t.Link != null)
2363                {
2364                    if (t.Link.IsHit)
2365                        return Task.FromResult(HitType.SUNKEN);
2366                    return Task.FromResult(HitType.HIT);
2367                }
2368                // Wenn das Feld kein Schiff war, wird es als verfehlt markiert
2369                return Task.FromResult(HitType.MISS);
2370            }
2371
2372            /// <summary>
2373            /// Gibt alle Positionen der Schiffe zurück, die noch nicht vom Spieler
                getroffen wurden
2374            /// </summary>
2375            /// <param name="leftShips"></param>
2376            /// <returns></returns>
2377            public override Task<IEnumerable<Point>> GetNonhitShips(IEnumerable<Point>
                leftShips)
2378            {
2379                return Task.FromResult(enemyPlayingField.GetNonhitShipTiles().Select(x =>
                    x.Position));
2380            }
2381        }
2382    }
2383    {   //[MainGameScripts/EnemyPlayers/NetworkPlayer.cs]
2384        public class NetworkPlayer : EnemyPlayer
2385        {
2386            private TcpClient connection;
2387            private bool isEnemyReady = false;
2388
2389            public NetworkPlayer(FieldHandler fieldHandler, TcpClient _connection) : base(
                fieldHandler)
2390            {
2391                connection = _connection;
2392            }
2393            /// <summary>
2394            /// Callback bevor das Spiel beginnt
2395            /// </summary>
2396            /// <param name="message"></param>
2397            private void callback(StunTools.Message message)
2398            {
2399                // Wenn Bool dann ist es der IsPlayerTurn Wert
2400                if (message.ObjectType == typeof(bool))
2401                {
2402                    IsPlayerTurn = message.GetData<bool>();
2403                    if (!IsPlayerTurn)
2404                        // Der nicht spielende Spieler muss auf einen Zug warten
2405                        _ = connection.Receive().ContinueWith(async (task) => await
                            onPointMessage(task.Result!));
2406                }
2407                else
2408                {
2409                    // Wenn String dann ist es "READY" und der Gegner ist bereit
2410                    isEnemyReady = true;
2411                    EnemyReady();
2412                }
```

```
2413            }
2414
2415            /// <summary>
2416            /// Das Callback für den wartenden Spieler
2417            /// </summary>
2418            /// <param name="message"></param>
2419            /// <returns></returns>
2420            private async Task onPointMessage(StunTools.Message message)
2421            {
2422                // Wenn die Nachricht ein Point ist, dann wurde der Spieler getroffen und
                    muss zurückmelden was es für ein Treffer war
2423                if (message.ObjectType == typeof(Point))
2424                {
2425                    Point point = message.GetData<Point>();
2426
2427                    // Gibt den Treffertyp zurück
2428                    await connection.SendData(HitPlayer(point));
2429                    HoverCellEnd();
2430                }
2431                else
2432                {
2433                    // Wenn die Nachricht ein string ist, dann ist es "Unhover"
2434                    if (message.ObjectType == typeof(string))
2435                    {
2436                        HoverCellEnd();
2437                    }
2438                    else
2439                    {
2440                        // Wenn die Nachricht ein Tuple aus String und Point ist, dann
                            ist es "Hover" und die Position der "Hover" Zelle
2441                        HoverCell(message.GetData<(string, Point)>().Item2.X, message.
                            GetData<(string, Point)>().Item2.Y);
2442                    }
2443
2444                    _ = connection.Receive().ContinueWith(async (task) => await
                        onPointMessage(task.Result!));
2445                }
2446            }
2447
2448            /// <summary>
2449            /// Wartet auf die nachricht vom Gegner, was für ein Treffer es war
2450            /// </summary>
2451            /// <returns></returns>
2452            private async Task<HitType?> WaitHitTypeMessage()
2453            {
2454                StunTools.Message? message = null;
2455                while (message?.ObjectType != typeof(HitType))
2456                    message = await connection.Receive();
2457                // Jetzt der Spieler wieder auf einen Zug warten
2458                _ = connection.Receive().ContinueWith(async (task) => await onPointMessage
                    (task.Result!));
2459                return message?.GetData<HitType>();
2460            }
2461
2462            /// <summary>
2463            /// Gibt zurück was für ein Treffer der Spieler gemacht hat
2464            /// </summary>
2465            /// <param name="point"></param>
2466            /// <returns></returns>
2467            public override async Task<HitType> OnEnemyFieldHit(Point point)
2468            {
2469                await connection.SendData(point);
2470                return (await WaitHitTypeMessage()).Value;
2471            }
2472
2473            /// <summary>
2474            /// Sendet dem Gegner das man bereit ist
2475            /// </summary>
2476            public override async void OnPlayerReady()
2477            {
2478                if (!isEnemyReady)
```

```
2479                    // Wenn der Gegner noch nicht bereit ist, wird "READY" gesendet
2480                    await connection.SendData("READY");
2481                else
2482                {
2483                    // Wenn der Gegner bereit ist, wird ein zufälliger Spieler
                        ausgewählt, der anfängt und dem Gegner mitgeteilt
2484                    IsPlayerTurn = new Random().Next(0, 2) == 0;
2485                    await connection.SendData(!IsPlayerTurn);
2486                    if (!IsPlayerTurn)
2487                        _ = connection.Receive().ContinueWith(async (task) => await
                            onPointMessage(task.Result!));
2488                }
2489            }
2490
2491            /// <summary>
2492            /// Startet den callback bis beide bereit sind
2493            /// </summary>
2494            public override void OnLoad()
2495            {
2496                _ = connection.Receive().ContinueWith(task => callback(task.Result));
2497            }
2498
2499            /// <summary>
2500            /// Sendet dem gegner die Position der Zelle aud der, der Mauszeiger ist
2501            /// </summary>
2502            /// <param name="sender"></param>
2503            /// <param name="point"></param>
2504            public override async void OnEnemyFieldHover(object? sender, Point point)
2505            {
2506                if (IsPlayerTurn)
2507                    await connection.SendData(("Hover", point));
2508            }
2509
2510            /// <summary>
2511            /// Sendet dem gegner das der Mauszeiger nicht mehr auf dem Feld ist
2512            /// </summary>
2513            /// <param name="sender"></param>
2514            /// <param name="point"></param>
2515            public override async void OnEnemyFieldUnHover(object? sender, Point point)
2516            {
2517                if (IsPlayerTurn)
2518                    await connection.SendData("Unhover");
2519            }
2520            /// <summary>
2521            /// Gibt alle Positionen der Schiffe zurück, die noch nicht vom Spieler
                getroffen wurden
2522            /// Und sendet dem Gegner die Positionen der Schiffe die von ihm noch nicht
                getroffen wurden
2523            /// </summary>
2524            /// <param name="l"></param>
2525            /// <returns></returns>
2526            public override async Task<IEnumerable<Point>> GetNonhitShips(IEnumerable<
                Point> l)
2527            {
2528                await connection.SendData(l);
2529                return (await connection.Receive())!.GetData<IEnumerable<Point>>()!;
2530            }
2531        }
2532    }
```