

****

Klasse: E2FI

Team 17

Alexander Zimmermann,   
Simon Hodler,   
Thorsten Kaufmann

Inhaltsverzeichnis

[1. Ausgangssituation 3](#_Toc266960962)

[2. Anforderungen 3](#_Toc266960963)

[2.1 A1 Projektverwaltung 3](#_Toc266960964)

[2.2 A2 Netzwerkfähigkeit 3](#_Toc266960965)

[2.3 A3 Intuitive Benutzeroberfläche 3](#_Toc266960966)

[2.4 A4 Piratendesign 3](#_Toc266960967)

[3. Projektumfeld 4](#_Toc266960968)

[3.1 Auftraggeber 4](#_Toc266960969)

[3.2 Projektleiter 4](#_Toc266960970)

[3.3 Projektteam 4](#_Toc266960971)

[4. Systemumfeld 5](#_Toc266960972)

[4.1 Komponenten 5](#_Toc266960973)

[4.2 Umfeld 5](#_Toc266960974)

[4.2.1 Ansicht der Log-Messages. 6](#_Toc266960975)

[4.2.2 Ansicht des Tortoise Kontext- Menüs 7](#_Toc266960976)

[5. Entwurf 8](#_Toc266960977)

[6. Umsetzung 9](#_Toc266960978)

[5.1 Implementierung der Oberfläche 9](#_Toc266960979)

[5.2 Implementierung der Anforderungen 9](#_Toc266960980)

[5.2.1 Anlegen der Spielfelder 9](#_Toc266960981)

[5.2.2 Darstellung und Setzen der Schiffe 11](#_Toc266960982)

[5.2.3 Netzwerkkommunikation 12](#_Toc266960983)

[5.2.4 Spiellogik 12](#_Toc266960984)

[5.3 Layout und Design des Spielfelds 13](#_Toc266960985)

[7. Abbildungsverzeichnis 16](#_Toc266960986)

[8. Quellcodeverzeichnis 16](#_Toc266960987)

# Ausgangssituation

# Anforderungen

## 2.1 A1 Projektverwaltung

Da das Projekt an mehreren PC’s und von mehreren Personen bearbeitet wird, muss eine Lösung gefunden werden, wie alle Teammitglieder an jedem PC stets mit der aktuellsten Version arbeiten können.  
  
Um Änderungen an dem Projekt zu sichern und die aktuellsten Version den Mitgliedern bereitstellen zu können muss eine Projektverwaltung eingerichtet werden.  
Somit kann jedes Teammitglied auf die Ressourcen des Projektes zugreifen.   
  
Dazu sollte ein SVN Repository eingerichtet werden und bei jedem Projektteilnehmer ein lokaler Client zur Dateiverwaltung installiert werden.

## 2.2 A2 Netzwerkfähigkeit

Das Spiel muss Netzwerkfähig sein um über das lokales Netzwerk und dem Internet spielen zu können.

## 2.3 A3 Intuitive Benutzeroberfläche

Des Weiteren soll das Spiel eine einfache und selbsterklärende Benutzeroberfläche besitzen, die eine Verwendung des Spiels ohne große Einarbeitungszeit ermöglicht.

## 2.4 A4 Piratendesign

Das Design des Spiels soll einen Piraten / Karibik look bekommen.   
Es soll von dem typischen Weltkriegsszenario abweichen.

# Projektumfeld

## 3.1 Auftraggeber

Ferdinand von Steinbeis – Schule Reutlingen.

## 3.2 Projektleiter

Alexander Zimmermann

## 3.3 Projektteam

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Rolle / Rollen** | **Name** | | **Telefon** | **E-Mail** |
| Entwickler | | Simon Hodler | 0163 / 7641337 | [simon.hodler@hs-albsig.de](mailto:hodlersi@hs-albsig.de) |
| Entwickler | | Thorsten Kaufmann | 0151 / 24069734 | [thorsten.kaufmann@hs-albsig.de](mailto:kaufmath@hs-albsig.de) |
| Projektleiter | | Alexander Zimmermann | 0160 / 90559017 | [alexander.zimmermann@hs-albsig.de](mailto:zimmeral@hs-albsig.de) |
| Auftraggeber | Ferdinand von Steinbeis – Schule Reutlingen | | 07121 / 485-111 |  |

# Systemumfeld

## 4.1 Komponenten

|  |  |
| --- | --- |
| **Betriebssystem** | Microsoft Windows 7 |
| **Programmiersprache** | C# |
| **Entwicklungsumgebung** | Microsoft Visual Studio 2010 |
| **Zusätzliche Software** | Adobe Photoshop  GIMP  Audacity  MS Projekt 2010 |

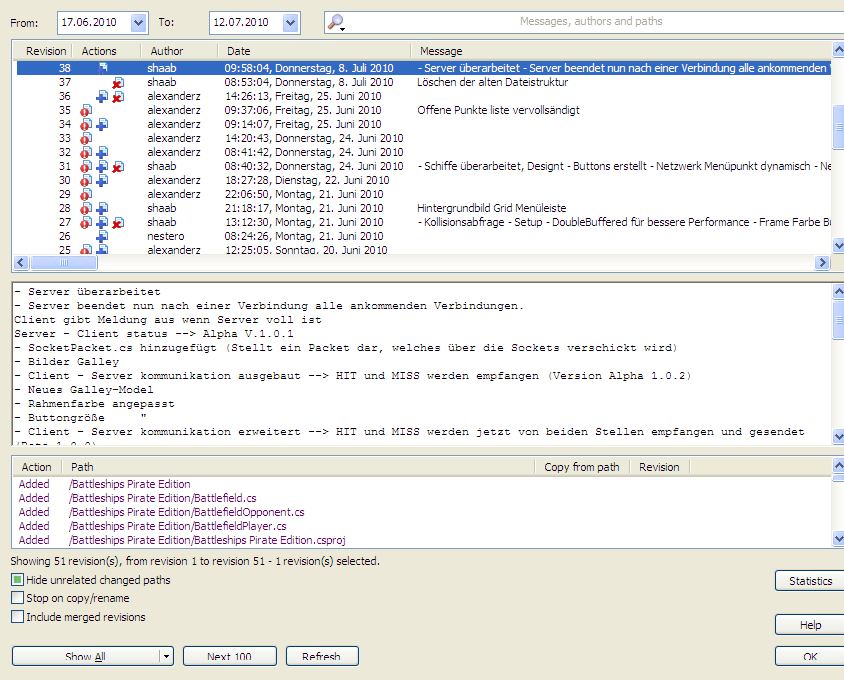
## 4.2 Umfeld

Der Projektspeicherort befindet sich auf der Projektverwaltungsseite <http://www.origo.ethz.ch>.  
Origo bietet eine Subversion (SVN) Repository (Ablage) auf das per Webinterface oder lokalem Client zugegriffen werden kann. Wenn ein Teilnehmer des Projekts eine Änderung vorgenommen hat, kann er diese mit dem lokal installierten Client „Tortoise“ auf das Repository hochladen. Dabei wird bei jedem neuen Upload eine neue Revision (Änderungsstand) erstellt.

Somit kann bei einem Verlust oder einem versehentlichen überschreiben die Vorgängerversion wiederhergestellt werden.

### 4.2.1 Ansicht der Log-Messages.

Abbildung – Ansicht der Log-Messages



**1..**

**2.**

**3.**

1. Ansicht der einzelnen Revisions, es kann auf alle vorherigen Versionen zugegriffen werden.  
Es wird angezeigt von wann und von wem die Version hochgeladen wurde.

2. Hier werden die Änderungen angezeigt die der jeweilige Benutzer seit der letzten Revision vorgenommen hat.

3. In diesem Fenster wird angezeigt welche Dateien geändert, hinzugefügt oder entfernt wurden.

### 4.2.2 Ansicht des Tortoise Kontext- Menüs

Tortoise bietet als lokale Versionsverwaltung die Möglichkeit über das Windows Kontextmenü die aktuellste Projektversion herunterzuladen (SVN Update) oder Änderungen am Projekt hochzuladen (SVN Commit).

Außerdem gelangt man über das Kontextmenü unter dem Punkt „TortoiseSVN“ auf weitere Einstellmöglichkeiten.

So lässt sich z.B. ein Verzeichnis sperren damit keine Änderungen mehr vorgenommen werden können (Get lock…) .

Falls Zugriff auch einer ältere Datei benötigt wird kann eine bestimmte Version einer Datei oder des gesamten Projekts der jeweiligen Revision heruntergeladen werden (Update to revision…).

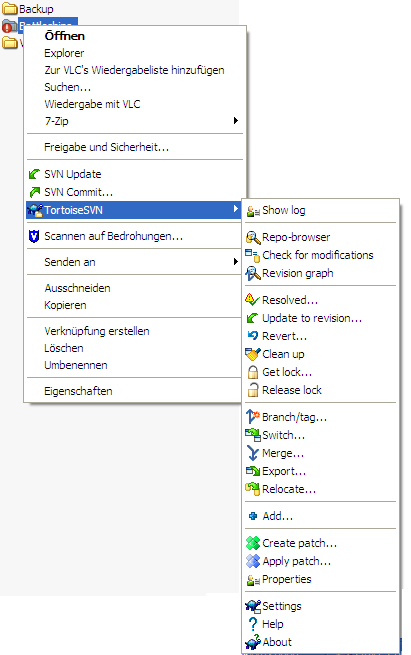


Abbildung - Tortoise Kontextmenü

# Entwurf

Noch vor Beginn der Implementierungsphase wurde der Entwurf des Spiels erstellt. Dabei wurden Konzepte über das Oberflächendesign, den Spielverlauf sowie dem Klassenentwurf erstellt.

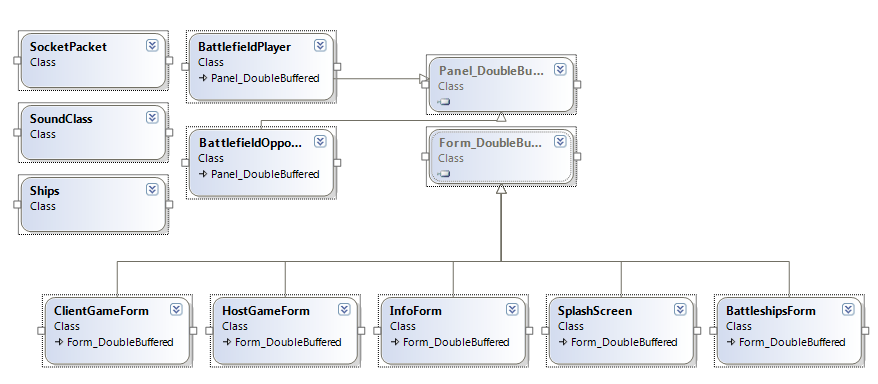


Abbildung 3 - UML Klassendiagramm

Abbildung 3 zeigt den Klassenentwurf des Spiels. Hier lassen sich alle benötigten Klassen, Attribute sowie Methoden bereits vor einer Implementierung feststellen. Dies erleichtert und beschleunigt den Entwicklungsvorgang erheblich.

# Umsetzung

## 5.1 Implementierung der Oberfläche

Zu Beginn der Implementierungsphase wurde zunächst der Entwurf der Oberfläche festgelegt. Dieser Entwurf wurde dann mit Hilfe von Adobe Photoshop entsprechend umgesetzt und in Visual Studio eingebunden. Die Oberfläche sollte vor allem ansprechend und klar strukturiert sein. Des Weiteren sollte sie benutzerfreundlich und selbsterklärend sein.

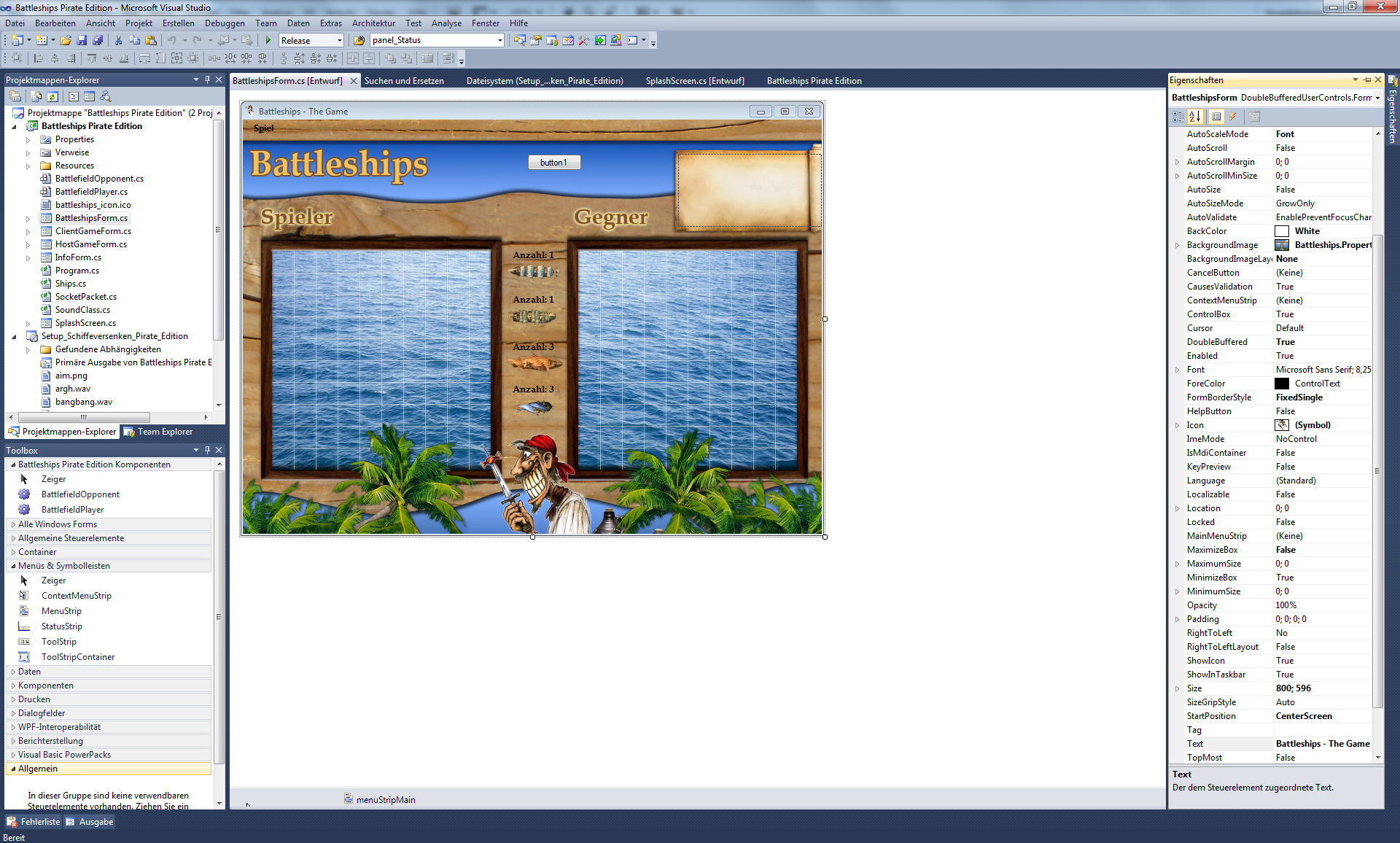


Abbildung 4 - Darstellung der Oberfläche in VS2010

## 5.2 Implementierung der Anforderungen

### 5.2.1 Anlegen der Spielfelder

Die Spieloberfläche ist im Wesentlichen in zwei Spielfelder aufgeteilt. Das „Spieler“- sowie „Gegner“-Spielfeld. Die beiden Felder unterscheiden sich in ihrem Aufbau sowie der Erstellung nicht. Dennoch müssen diese beiden Spielfelder voneinander separiert werden.  
Auf dem „Spieler“-Spielfeld kann der Spieler seine eigenen Schiffe setzen und bekommt im Laufe des Spiels angezeigt, ob der Gegenspieler Schiffe getroffen hat oder nicht.  
Auf dem „Gegner“-Spielfeld kann der Spieler das gegnerische Feld unter Beschuss nehmen und bekommt ebenfalls angezeigt ob er ein Schiff getroffen hat oder nicht.

Für die Spielfeldgenerierung wurden daher zwei Klassen entwickelt.

public class BattlefieldPlayer : Panel\_DoubleBuffered

Quellcode 1 - BattlefieldPlayer Klasse

public class BattlefieldOpponent : Panel\_DoubleBuffered

Quellcode 2 - BattlefieldOpponent Klasse

Die Klasse BattlefieldPlayer repräsentiert dabei das „Spieler“-Spielfeld, während die Klasse BattlefieldOpponent das „Gegner“-Spielfeld darstellt. Die beiden Klassen erben jeweils von einem eigen entwickelten Steuerelement Panel\_DoubleBuffered. Dieses Steuerelement stellt ein .Net Panel mit erweiterter DoubleBuffered-Eigenschaft dar. Dadurch wird das Zeichnen der Panels sowie deren Inhaltsbilder beschleunigt und ein mögliches Flimmern unterdrückt.

Die Konstruktoren der beiden Spielfeldklassen sind, beruhend auf ihrem gleichen Aufbau, identisch. Hier wird deshalb nur der Konstruktor der Klasse BattlefieldPlayer betrachtet.

public BattlefieldPlayer(int x, int y)

{

// Schiffe werden standardgemäß Horizontal gesetzt (mit einem Klick auf die rechte Maustaste kann das geändert werden)

horizontal = true;

…

this.Location = new Point(x, y);

this.Width = 300;

this.Height = 300;

this.Size = new Size(300, 300);

this.BackColor = Color.Transparent;

…

Quellcode 3 - Konstruktor der Klasse BattlefieldPlayer

Der Konstruktor bekommt bei seinem Aufruf zwei Integer-Werte übergeben. Diese Werte bestimmen die Position des Spielfeldes, bezogen auf die linke obere Ecke der Spieloberfläche. Mittels this.Location = new Point(x, y) wird die Position des Spielfeldes gesetzt. Da jedes der beiden Spielfelder 300x300px groß ist, wird über this.Width = 300 sowie this.Height = 300 die Breite bzw. Höhe des Spielfeldes gesetzt.

Die Spielfelder bestehen in sich aus einem Umfassenden Panel, welches wiederum 100 Panels der Größe 30x30px enthält (10x10 Panels). Dadurch lässt sich ein klassisches „Schiffe versenken“-Spielfeld darstellen.

…

//Unterpanel Spieler

for (int i = 0; i < pb.GetLength(0); i++)

{

for (int j = 0; j < pb.GetLength(1); j++)

{

Panel\_DoubleBuffered p = new Panel\_DoubleBuffered();

p.Location = new Point(i \* 30, j \* 30);

p.Tag = 0;

p.Margin = new Padding(0);

p.Padding = new Padding(0);

p.Name = "pb\_" + i.ToString() + ":" + j.ToString();

p.Size = new Size(30, 30);

p.MouseClick += new MouseEventHandler(p\_MouseClick);

p.MouseEnter += new EventHandler(p\_MouseEnter);

p.MouseLeave += new EventHandler(p\_MouseLeave);

p.BackColor = Color.Transparent;

p.BorderStyle = BorderStyle.None;

pb[i, j] = p;

this.Controls.Add(p);

pb\_Store[i, j] = new Panel\_DoubleBuffered();

}

}

}

Quellcode 4 – Schleifen zur Panel-Generierung

Die 100 Unterpanels werden in zwei verschachtelten Schleifen erstellt. Dabei läuft jede der beiden Schleifen von 0 bis 9 durch. In jedem Durchgang wird mittels Panel\_DoubleBuffered p = new Panel\_DoubleBuffered() ein neues Panel erstellt und letztlich über this.Controls.Add(p) zum übergeordneten Panel hinzugefügt.   
Da in jedem der einzelnen Unterpanels auf gewisse Ereignisse reagiert werden soll (z.B. MouseClick), wird in jedem Schleifendurchlauf ein entsprechender Eventhandler an jedes Unterpanel gehängt. p.MouseClick += new MouseEventHandler(p\_MouseClick) hängt einen Eventhandler an das MouseClick-Event eines Unterpanels. Das heißt, jedes Mal, wenn ein Spieler in ein Unterpanel klickt, wird die entsprechende Methode aufgerufen.

Dadurch, dass die Spielfelder dynamisch im Code und nicht im Designer erstellt werden, kann man beliebig viele Instanzen der Spielfelder generieren und an einer passenden Stelle der Oberfläche erzeugen.

### 5.2.2 Darstellung und Setzen der Schiffe

Wie bereits beschrieben besteht ein Spielfeld aus einem großen Panel welches wiederum 100 Unterpanels beinhaltet. Diese Unterpanels sind alle mit EventHandler-Methoden verknüpft. So zum Beispiel auch das MouseEnter-Event. Dieses Event wird dann ausgelöst, wenn der Spieler mit der Maus in ein solches Unterpanel eintritt.

Quellcode – MouseEnter Event

public void p\_MouseEnter(object sender, EventArgs e)

{

// Event wurde von einer Panel\_DoubleBuffered ausgelöst...

// Senderobjekt erhalten --> Panel welches das Event ausgelöst hat

Panel\_DoubleBuffered tmp = (Panel\_DoubleBuffered)sender;

// Schiff zeichnen

drawShips(ref tmp);

}

Durch das Parsen (Konvertierung) des Senderobjektes (Panel\_DoubleBuffered)sender wird

das Panel ermittelt, welches das MouseEnter-Event ausgelöst hat. Damit kann explizit herausgefunden werden, an welcher Position der Spieler mit der Maus in das Spielfeld gefahren ist. Das ausgewählte Schiff wird dann über die Methode drawShips(ref tmp)

auf das Spielfeld gezeichnet

### 5.2.3 Netzwerkkommunikation

Die Netzwerkkommunikation wurde mittels Sockets realisiert.  
Sockets bilden eine standardisierte und plattformunabhängige Schnittstelle (API) zwischen der Netzwerkprotokoll-Implementierung und der eigentlichen Anwendungssoftware des Betriebssystems.

Sockets sind an IP Adressen sowie Ports gebunden abhängig von dem verwendeten Protokoll. Ein Server Socket horcht an einem gebundenen Port und wartet auf eingehende Verbindungsanfragen.  
Wird eine solche Verbindungsanfrage erkannt wird aus dem horchendem Host- Socket das in Verbindung stehende Host- Socket abgeleitet, d.h. der ursprüngliche Host- Socket bleibt und wartet weiterhin auf eine neue Verbindung, während ein neuer, auf den Client gerichteten Socket geöffnet wird.

Dieser wird nur für die Kommunikation mit diesem einem Client verwendet.  
Der Host bleibt solange aktiv bis die Verbindung zum Client auf der anderen Seite beendet wird.

string portStr = textBoxPort.Text;

int port = System.Convert.ToInt32(portStr);

// Create the listening socket...

m\_mainSocket = new Socket(AddressFamily.InterNetwork,

SocketType.Stream,

ProtocolType.Tcp);

IPEndPoint ipLocal = new IPEndPoint(IPAddress.Any, port);

// Bind to local IP Address...

m\_mainSocket.Bind(ipLocal);

// Start listening...

listboxMessage.Items.Add("Initialize Server...");

m\_mainSocket.Listen(1);

listboxMessage.Items.Add("Server initialized");

// Create the call back for any client connections...

m\_mainSocket.BeginAccept(new AsyncCallback(OnClientConnect), null);

public void OnClientConnect(IAsyncResult asyn)

{

try

{

if (m\_clientCount < 1)

{

// Here we complete/end the BeginAccept() asynchronous call

// by calling EndAccept() - which returns the reference to

// a new Socket object

m\_workerSocket = m\_mainSocket.EndAccept(asyn);

// Let the worker Socket do the further processing for the

// just connected client

WaitForData(m\_workerSocket);

### 5.2.4 Spiellogik

## 5.3 Layout und Design des Spielfelds

Das Layout des Spielfeldes wurde komplett in Adobe Photoshop erstellt.  
Die folgenden Bilder zeigen die Entwicklung des Layouts und Designs des Spielfeldes.

Abbildung 5 zeigt den ersten Entwurf des Spielfeldes in Visual Studio.   
Dabei wurde die Spielfeldgröße mit 10x10 Feldern festgelegt, sowie die quadratische Feldgröße mit 30px\*30px.   
Die Buttons für die Schiffe wurden zwischen den beiden Feldern platziert.

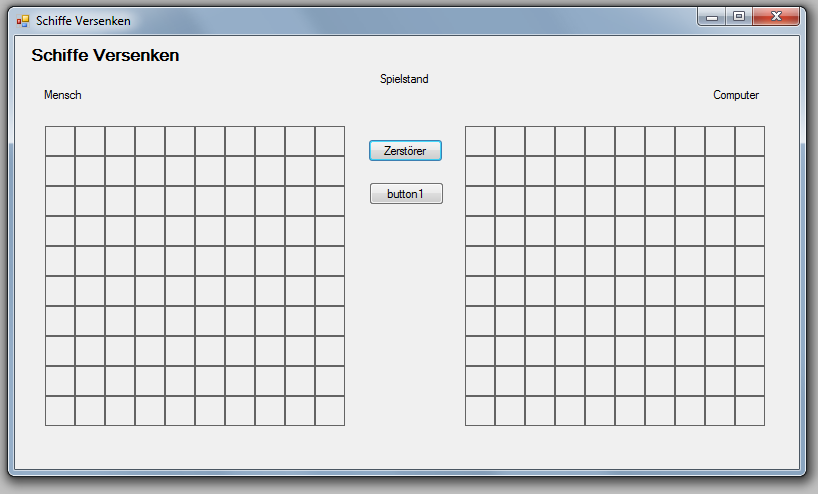


Abbildung Erster Entwurf des Spielfeldes noch in Visual Studio

In Abbildung 6 wurde das erste Konzept in Photoshop erstellt. Das Spielfeld besteht aus einer Grafik mit der Größe 800px\*600px.   
Für die Spielfelder des Spielers und des Gegners wurde ein Wasserbild mit Größe 300px\*300px eingesetzt.   
Diese werden dann hinter den einzelnen Panels in Visual Studio angezeigt.  
Es wurden die ersten Palmen und ein Pirat hinzugefügt die zum Thema „Pirate Edition“ passen.

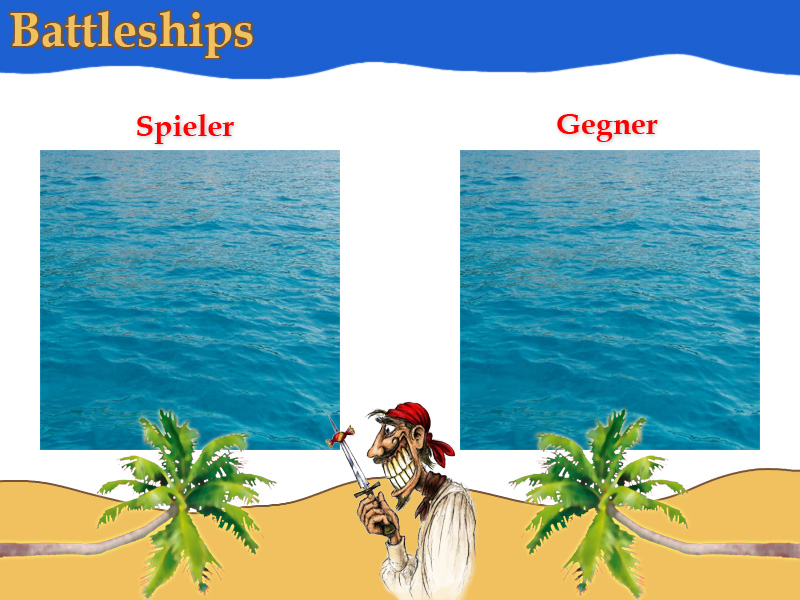


Abbildung Erstes Layout Design in Photoshop

In der nächsten Version in Abbildung 7 wurde schon der erste hölzerne Hintergrund des Spielfeldes eingefügt und hinter dem Schriftzug Battleships sieht man nun einen blauen Farbverlauf.



Abbildung Zweite Version des Spielfelds

In Abbildung 8 wurde Platze für die Menüleiste des Spieles gemach.  
Die Grafik wurde nach Oben etwas erweitert.   
An dieser Stelle ist nun ein Holzbalken auf dem später die Menüleiste angezeigt wird.  
Die Spielfelder wurden mit einem dunklen Holzrahmen eingefasst.



Abbildung Dritte Version des Spielfelds

In der finalen Version der Oberfläche (Abbildung 9) wurde in der rechten oberen Ecke ein Pergament hinzugefügt auf diesem dann der aktuelle Spielstatus angezeigt wird.  
Das Wasser für die jeweiligen Spielfelder wurde nochmal überarbeitet und mit einem Raster versehen. Nun ist man nichtmehr auf den Rand der Panels angewiesen und kann diese im Ruhezustand komplett Transparent schalten.



Abbildung Finale Version des Spielfeldes

# Abbildungsverzeichnis

[Abbildung 1 – Ansicht der Log-Messages 6](file:///C:\Users\itoa1\Desktop\zeugs\Dokumentation.docx#_Toc266798816)

[Abbildung 2 - Tortoise Kontextmenü 7](file:///C:\Users\itoa1\Desktop\zeugs\Dokumentation.docx#_Toc266798817)

[Abbildung 3 - UML Klassendiagramm 8](#_Toc266798818)

[Abbildung 4 - Darstellung der Oberfläche in VS2010 9](#_Toc266798819)

# Quellcodeverzeichnis

[Quellcode 1 - BattlefieldPlayer Klasse 10](#_Toc266798810)

[Quellcode 2 - BattlefieldOpponent Klasse 10](#_Toc266798811)

[Quellcode 3 - Konstruktor der Klasse BattlefieldPlayer 10](#_Toc266798812)

[Quellcode 4 – Schleifen zur Panel-Generierung 11](#_Toc266798813)