*Todo list

Wahlpflichtfach KI

Projekt: Schiffe Versenken

Autoren: Victor Apostel, Stefan Bogdanski, Sibille Ritter
Studiengang: (Internationaler) Studiengang Technische Informatik B.Sc.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Hier Aufgabenbeschreibung

2 Spielregeln

3 Kommunikationsprotokoll

Die Kommunikation zweier Spieler erfolgt auf Basis von Textnachrichten, den sogenannten Kommandos, die einer fest definierten Codierung genügen. Das Ziel dieses Kapitels ist die Erläuterung des Nachrichtenaufbaus und deren Interpretation.

Jedes Kommando, das zu interpretieren gilt, ist aus Gründen der Prologkompatibilität mit runden Klammern umgeben und endet mit einem ".". Da zudem stets ganze Zeilen verarbeitet werden sollen, wird der Nachricht das nicht druckbare Zeichen "\n" angehängt. Damit wird signalisiert, dass die gesamte Nachricht übermittelt wurde.

Innerhalb der umschließenden runden Klammern werden zwei Angaben erwartet. Die erste Angabe wird als *Opcode* bezeichnet und dient der eindeutigen Bestimmung des Kommandotyps. Die zweite Angabe des Kommandos ist eine in eckigen Klammerun umgebene Parameterliste und kann null bis drei durch Kommata getrennte nummerische Elemente beinhalten. Anhand des Opcodes richtet sich die Anzahl der zu erwartenden Parameter.

Die formale Beschreibung des Kommandoaufbaus liegt im Folgenden in Form von regulären Ausdrücken vor.

$$
\begin{aligned}
COMMAND &:= \backslash(OPCODE, LIST\backslash)\backslash.NEWLINE \\
LIST &:= \backslash[[PARAMS]?\backslash] \\
NEWLINE &:= \backslash\backslash r \\
OPCODE &:= [1-5] \\
PARAMS &:= [0-9] + [, [0-9]+]\{0,2\}
\end{aligned}
$$

Der Spielablauf durchläuft mehrere Zustände, die durch das Kommunikationsprotokoll

abgedeckt werden müssen. So startet jeder Spieler in einem Initialisierungszustand. In diesem Zustand wird festgelegt, welcher der beiden Spieler den ersten Angriff ausführt.

Stellt der Server eine gültige Anzahl an verbundenen Spielteilnehmern fest, sendet dieser ein Startsignal an alle Teilnehmer und die Clients wechseln ihrerseits vom Initialisierungszustand in den Angriffs-, bzw. Verteidigungszustand. Weitere Nachrichten die es zu übertragen gilt, sind zum Einen der Angriff auf eine Feldkoordinate, sowie zum Anderen das Ergebnis des Angriffs.

Die Abbildung der genannten Aktionen in Opcodes, sowie die erwarteten Parameter können der Tabelle 1 entnommen werden.

Opcode	**Bedeutung**	**Param 1**	**Param 2**	**Param 3**
1	Angriff	X	Y	-
2	Ergebnis des Angriffs	X	Y	Ergebnis
3	Spieler startet im Verteidigungszustand	-	-	-
4	Spieler startet im Angriffszustand	-	-	-
5	Startsignal	-	-	-

Tabelle 1: Kommunikationsprotokoll

Das Ergebnis eines Angriffs wird ebenfalls kodiert übertragen und kann den Zustand des Clients beeinflussen. Im Regelfall sendet bzw. empfängt dieser die Ereignisse “Wasser“, “Schiff wurde getroffen“, oder “Schiff wurde versenkt“. Des Weiteren kann als Reaktion auf einen Angriff die Nachricht eintreffen, dass das letzte Schiff versenkt wurde. In diesem Fall wechseln die Clientzustände in Abhängigkeit vom Sender und Empfänder dieser Nachricht in “gewonnen“ bzw. “verloren“.

Eine Auflistung der möglichen Ergebnisse eines Angriffs werden in der Tabelle 2 präsentiert.

Code	**Bedeutung**
1	Wasser
2	Schiff wurde getroffen
3	Schiff wurde getroffen und versenkt
4	Letztes Schiff wurde versenkt. Das Spiel ist beendet

Tabelle 2: Ergebniskodierung

4 Softwarearchitektur

Die Software des Spiels “Schiffe versenken“ wurde als Client-Server Anwendung gemäß Abbildung 1 entworfen. Die zentrale Kommunikationsschnittstelle stellt der Kommunikationsserver dar, der auf dem Port 54321 eingehende Verbindungen annimmt. Verbindungen können sowohl mittels eines Java-Clients hergestellt werden, die von einem Menschen bedient werden, als auch von Prolog-Clients, die vollständig autonom agieren. Es liegen hinsichtlich der Clientkombinationen keine Beschränkungen vor, sodass auch z.B. ein Prologclient gegen einen anderen Prologclient antreten kann.

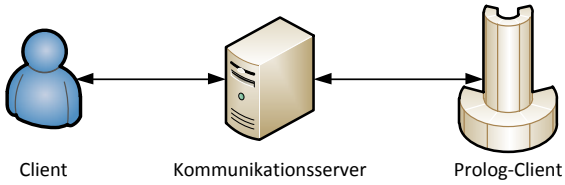

Abbildung 1: Darstellung der möglichen Kommunikationsteilnehmer

Der gesamte Spielverlauf lässt sich anhand der Statusdiagramme in den Abbildungen 2 und 3 beschreiben. Diese sind sowohl für den Javaclient, als auch für das Prologprogramm gültig.

Unmittelbar nach Start des Clients befindet sich dieser im Initialisierungszustand *INITIALIZATION*. Während dieser Phase obliegt es dem Client seine Schiffe gemäß den Regeln zu platzieren. Des Weiteren hat er eine Nachricht des Servers zu empfangen, die angibt, ob sein initialer Zustand *DEFENCE* oder *ATTACK* sein soll, wenn er in die *RUNNING*-Phase übergeht. Dieser Übergang erfolgt durch den Empfang des Startkommandos, mit dem Opcode = 5.

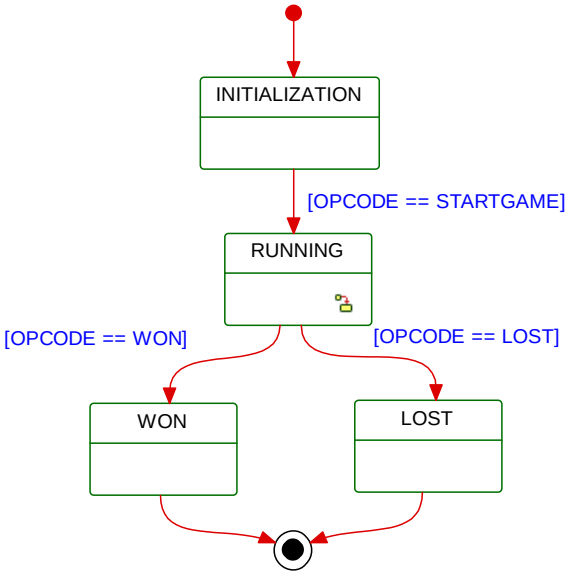

Abbildung 2: Zustandsübergänge des Clients

Befindet sich der Client im *RUNNING*-Status, so wechselt er zwischen seinen internen Zuständen *ATTACK* und *DEFENCE* hin und her. Dieser Wechsel erfolgt immer dann, wenn eine *ATTACKRESPONSE* Nachricht mit dem Opcode = 2 übertragen wurde. Der primäre Unterschied zwischen beiden Zuständen ist die Reihenfolge der erwarteten Nachrichten. Befindet sich der Client im Subzustand *DEFENCE*, so erwartet er von seinem Kontrahenten eine Nachricht mit dem Opcode = 1 und beantwortet diese seinerseits mit dem Opcode = 2. Sollte sich der Client im Subzustand *ATTACK* befinden, so sendet er zuerst die Nachricht mit dem Opcode = 1 und erwartet im Anschluss eine Nachricht seines Gegners.

Das Alternieren der Zustände erfolgt solange, wie in der Antwortnachricht mit dem Opcode = 2 keine Meldung über den Verlust aller Schiffe transferiert wird. Dieses Ereignis wird gemäß Tabelle 2 auf Seite 4 mit dem Ergebniscode = 4 beschrieben. Empfängt der Client die Nachricht, so gilt das Spiel als gewonnen. Umgekehrt verliert der Client das Spiel, wenn er diese Nachricht verschickt.

Im Rahmen dieses Kontexts wechselt der Spielzustand in *WON* bzw. *LOST* und das Programm kann beendet werden.

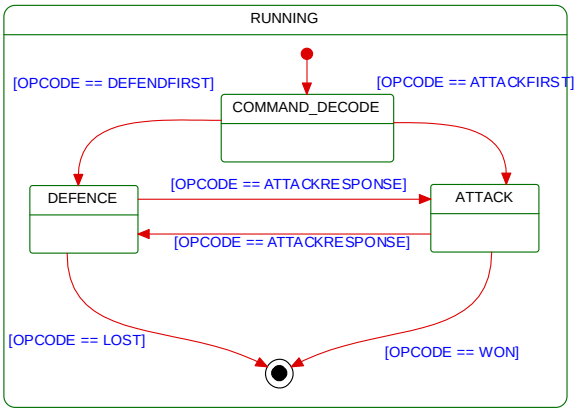

Abbildung 3: Interne Zustandsübergänge während des laufenden Spiels

Die vorgesehene Kommunikationsabfolge wird ebenfalls in Abbildung 4 in Form eines Sequenzdiagramms dargestellt. Der Client, der sich zuerst mit dem Server verbindet, beginnt per Konvention im Verteidungsmodus und der zweite verbundene Client startet im Angriffsmodus. Erst wenn sich zwei Teilnehmer am Server verbunden haben, sendet dieser das Startsignal an alle Clients.

Die Spielphase wird durch eine Schleife bestimmt, in der die Teilnehmer von den Angriffs- in den Verteidigungszustand wechseln und umgekehrt. Die dabei übertragenen Nachrichten werden stets an den Server übertragen, der diese an den zweiten Client weiterleitet. Dadurch kommt keine direkte Verbindung beider Spieler zustande.

Das Programmende aus Sicht der Kommunikation wird erreicht, wenn ein Spieler die Nachricht überträgt, dass er über keine Schiffe mehr verfügt. In diesem Fall werden die Verbindungen getrennt und der Server kann neue Verbindungen von Spielern zum Ausrichten eines neuen Spiels annehmen.

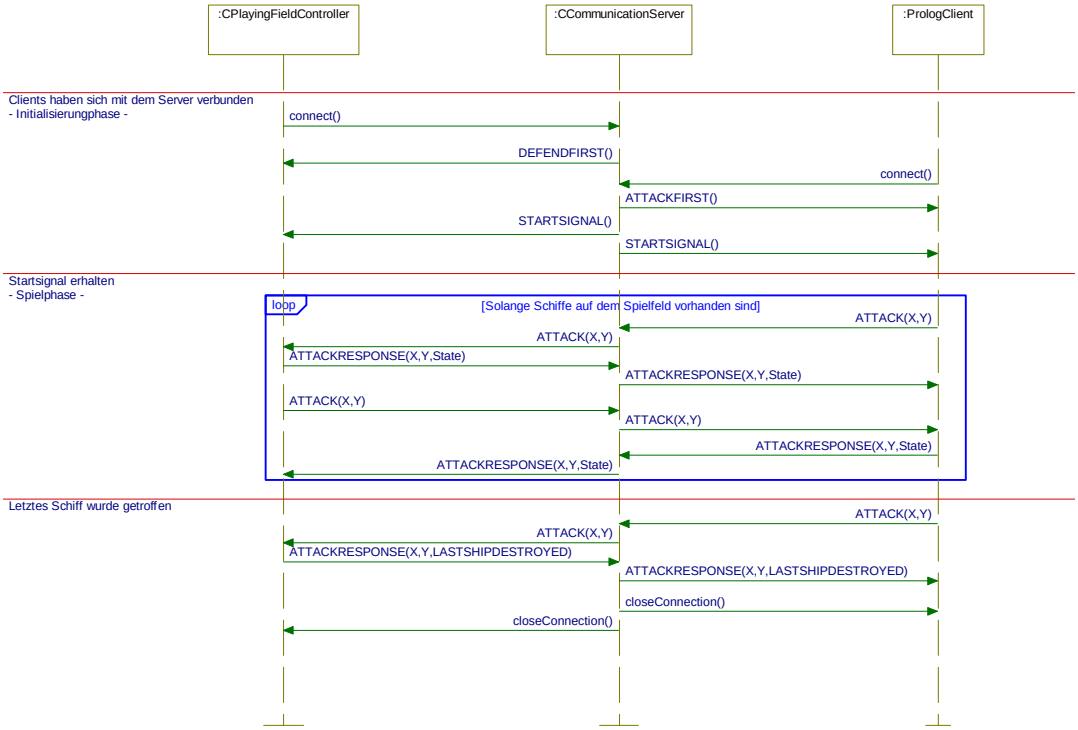

Abbildung 4: Sequenzdiagramm der Kommunikation

4.1 Spielserver

Der Server bildet die Schnittstelle zwischen den beiden Kommunikationspartnern, indem er eingehende Nachrichten eines Clients an jeden anderen verbundenen Teilnehmer weiterleitet. Des Weiteren limitiert er die maximale Spielerzahl und generiert das Startsignal, sobald sich zwei Teilnehmer mit ihm verbunden haben. Außerdem legt das Programm fest, welcher der Spieler im Verteidigungs- bzw. Angriffsmodus startet.

Die Serverlogik wird maßgeblich durch die beiden Klassen `CCommunicationServer` und `CClientHandler` gesteuert (vgl. Abbildung 5). Die Aufgabe der Klasse `CCommunicationServer` ist die Annahme eingehener Verbindungen. Für jeden verbundenen Client wird ein separater Thread vom Typ `CClientHandler` erzeugt und gestartet.

Diese Threadobjekte führen die eigentliche Serverlogik aus. Sie überwachen den eingehenden Datenstrom und sobald eine Nachricht eingegangen ist, wird diese kopiert und an jeden anderen Client weitergeleitet. Für die Überwachung des Eingangsstro-

mes ist die Methode `run()` verantwortlich und das Duplizieren der Nachricht wird in `notifyAllOtherClients(String line)` gehandhabt. Der eigentliche Sendevorgang wird durch den Aufruf der Methode `send(String msg)` ausgeführt.

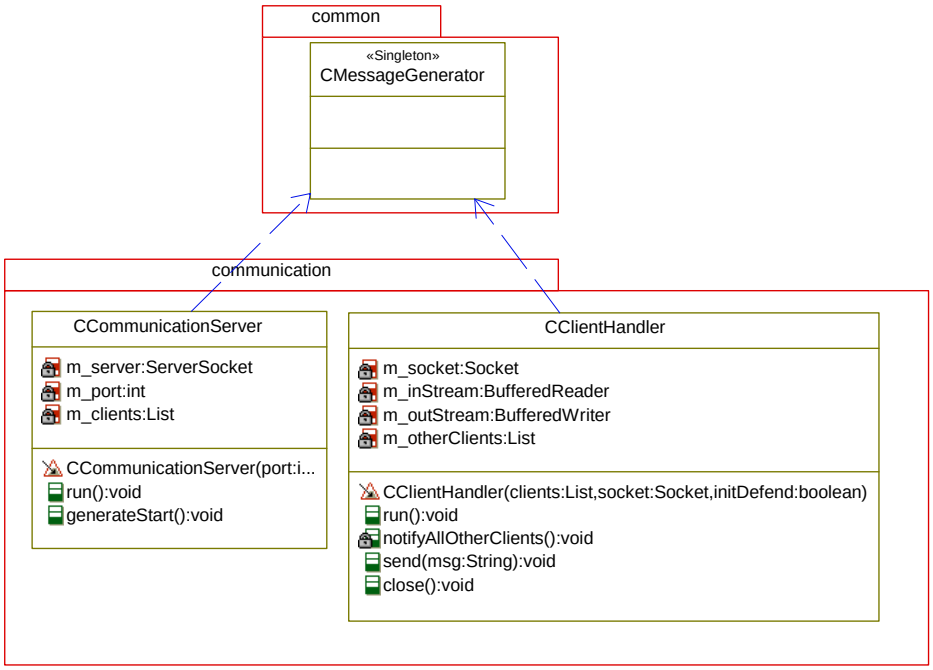

Abbildung 5: Klassendiagramm des Servers

4.2 Client - Java

Der hier beschriebene Javaclient ist ein möglicher Teilnehmer, der sich mit den Kommunikationsserver verbinden und ein Spiel austragen kann. Sein Design orientiert sich am Model-View-Controller (MVC) Prinzip, wobei das Spielbrettmodell auf Grund seines geringen Umfangs im Controller eingebettet wurde.

Eine Übersicht des Entwurfs wird in Abbildung 6 dargestellt. Die beiden Klassen `CBattleShipGUI` und `CPlayingFieldPanel` bilden den *View* Bestandteil des MVC Prinzips ab.

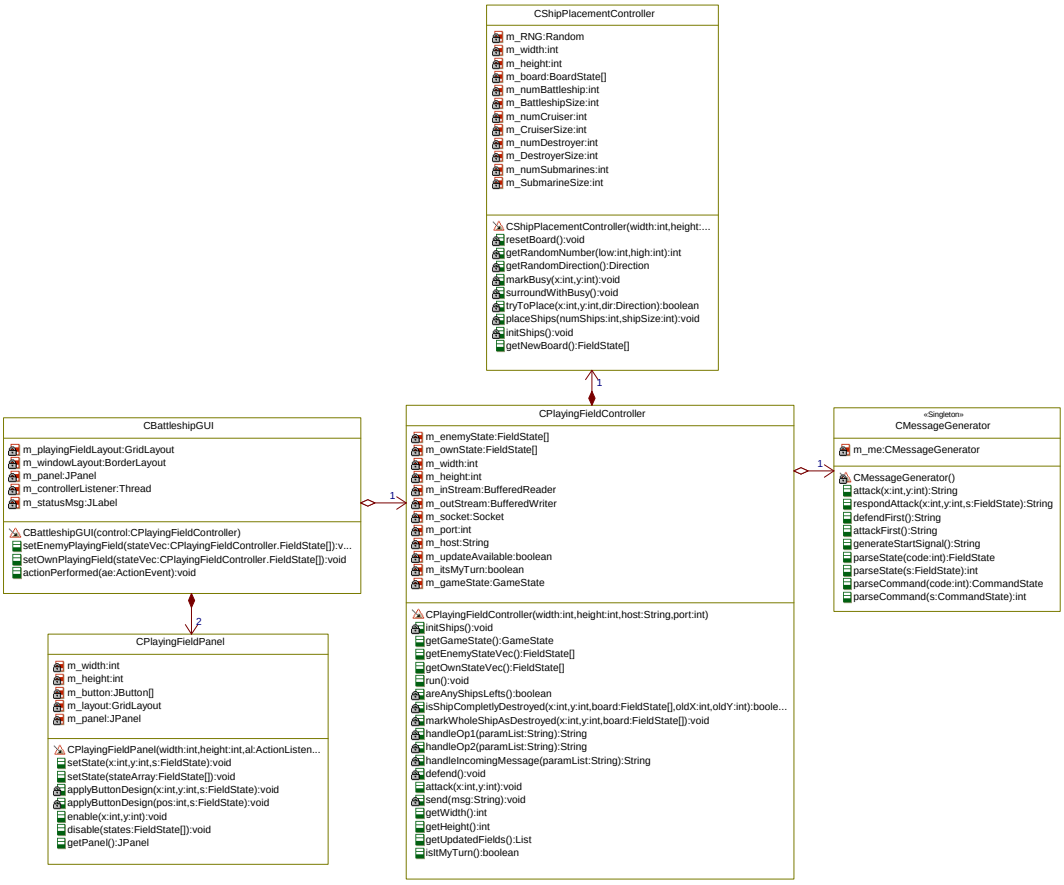

Abbildung 6: Klassendiagramm des Java Clients

4.2.1 View

Das eigene, sowie das gegnerische Spielfeld werden durch jeweils eine Instanz der Klasse `CPlayingFieldPanel` visualisiert, wie es in Abbildung 7 zu sehen ist. Dieses besteht primär aus einem gitterförmigen Spielfeld, dessen Spielfeldzustände durch Farben codiert sind. Das hier gezeigte Spielfeld stellt das eigene Wissen des gegnerischen Spielfelds dar. Die grauen Felder symbolisieren noch unbekanntest Terrain. Blau bedeutet, dass bei einem vorangegangenen Angriff ein Feld mit Wasser getroffen wurde. Rot symbolisiert ein getroffenes, jedoch noch nicht versenktes Schiff. Versenkte Schiffsfelder sind schwarz gehalten.

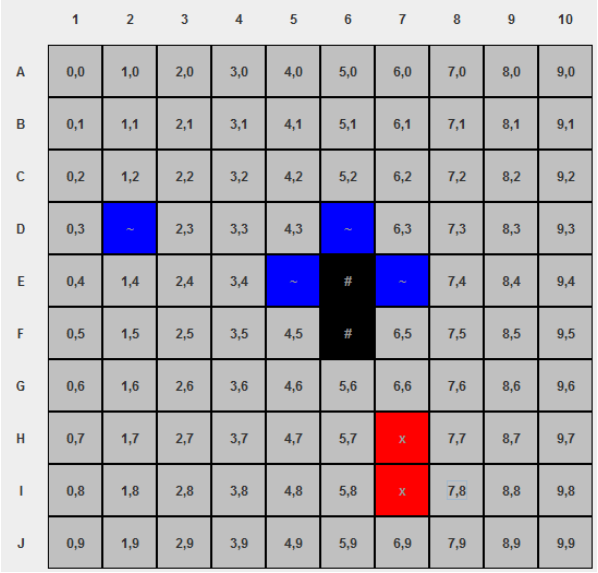

Abbildung 7: Gegnerisches Spielfeld mit einem zerstörten (schwarz) und einem beschädigten Schiff (rot)

Die visuelle Codierung des eigenen Spielfeldes wird analog zum gegnerischen Feld vorgenommen. Hier sind standardmäßig alle Felder aufgedeckt und wie in Abbildung 8 zu sehen ist, blau eingefärbt. Die eigenen Schiffe sind ebenfalls sichtbar und mit dunkelgrauer Farbe hervorgehoben. Angriffe des Gegners, die das Wasser getroffen haben, sind hellblau dargestellt. Analog zum gegnerischen Spielfeld sind Treffer rot und versenkte Schiffe schwarz eingefärbt.

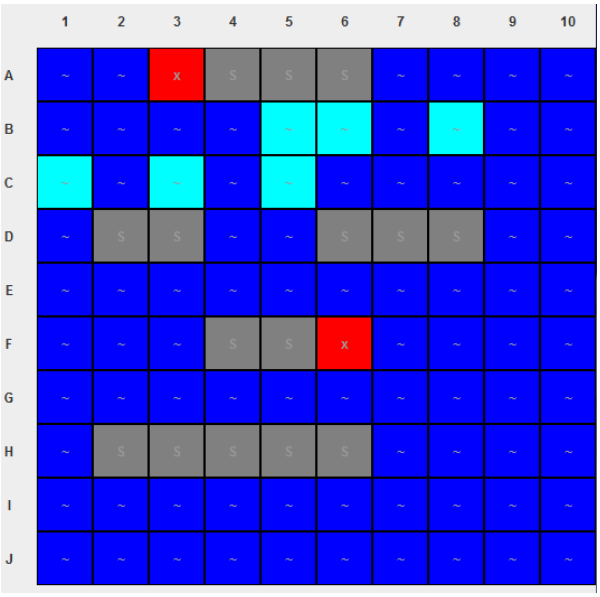

Abbildung 8: Eigenes Spielfeld mit beschädigten Schiffen (rot) und misglückten Angriffen des Gegners (hellblau)

Die GUI-Oberfläche besteht primär aus zwei Instanzen der Klasse `CPlayingFieldPanel`, die jeweils das eigene und gegnerische Spielfeld abbilden. Wie in Abbildung 9 zu erkennen ist befindet sich über jedem Spielfeld eine Überschrift, die die Zugehörigkeit signalisiert. Des Weiteren befindet sich in der unteren linken Ecke ein Statusfeld, das angibt, wer den nächsten Spielzug auszuführen hat bzw. ob man das Spiel verloren bzw. gewonnen hat. Sollte der Gegner am Zug sein, so wird gleichzeitig das gesamte Spielfeld deaktiviert, sodass keine Buttonaktionen ausgeführt werden können.

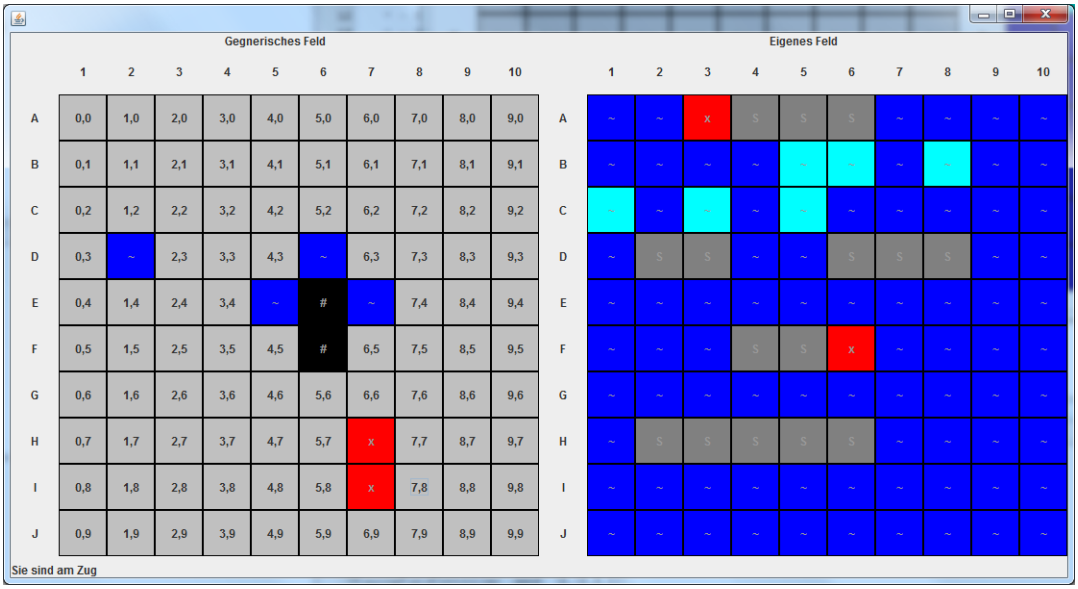

Abbildung 9: Gesamte Benutzeroberfläche

4.2.2 Controller / Model

Die Klasse `CPlayingFieldController` ist das zentrale Element der Spielsteuerung und wertet die eingehenden Nachrichten und Signale aus. Des Weiteren beinhaltet diese Klasse auch die Model-Komponente in Form zweier Arrays, die die Spielfelder repräsentieren. Der Abruf dieser Daten orientiert sich am Consumer-Producer Entwurfsmuster. Jede Änderung an diesen Arrays wird mittels der javaeigenen Methode `notifyAll()` den Beobachtern mitgeteilt.

Der Controller-Anteil wird durch die Erzeugung eines Threads realisiert, der eingehende Netzwerknachrichten annimmt und auswertet. Zur Generierung ausgehender Nachrichten, die einen Angriff signalisieren, dient die Methode `attack(int x, int y)` und wird durch die Interaktion mit der GUI aufgerufen. Sowohl der Thread, als auch die GUI-Interaktion blockieren sich auf Basis von Monitoren gegenseitig, die mit dem Schlüsselwort `synchronized` signalisiert werden. Mittels dieser Technik wird verhindert, dass die beiden Zustände *ATTACK* und *DEFENCE* unsachgemäß eingenommen werden.

Eingehende Angriffe werden durch den Vergleich mit dem eigenen Spielfeld behandelt und das Ergebnis per Nachricht bekannt gegeben. Gleichzeitig wird der Status des eigenen Spielfeldes an der entsprechenden Stelle aktualisiert. Im Fall von Wasser wird

an dieser Position der Status Verfehlt gesetzt. Bei einem Schiff wird der Status auf Getroffen geändert. Gleichzeitig wird eine Routine gestartet, die überprüft, ob das das ganze Schiff versenkt wurde. Ist dem so, so wird wiederum eine Routine gestartet, die überprüft, ob alle Schiffe versenkt wurden. Die entsprechenden Ergebnisse werden per Netzwerkantwort übermittelt. Sind keine eigenen Schiffe mehr vorhanden, oder ging die Nachricht ein, dass der Gegner über keine Schiffe mehr verfügt, so wird der Spielstatus auf verloren, bzw. gewonnen gesetzt.

Die Generierung der prologkompatiblen Nachrichten wird in der Klasse `CMessageGenerator` vorgenommen. Diese bietet für jede Aktion und Reaktion passende Methoden an und liefert den gewünschten zu übertragenen Text. Diese Klasse ist ebenfalls für die Dekodierung der Opcodes und Ergebnisse in interne Enumerationen zuständig.

In der Initialisierungphase ruft der Controller die Klasse `CShipPlacementController` auf. Diese Klasse ist für die regelkonforme Platzierung der Schiffe auf dem Spielbrett zuständig. Der Vorgang zum Platzieren eines Schiffes ist dabei von seinem Typ und somit seiner Größe unabhängig und kann wie folgt beschrieben werden:

1. Generiere zufällig die x/y Position, sowie die Orientierung, in der das Schiff platziert werden soll.

2. Prüfe ob diese Position belegt ist, oder ob sich in der direkten 8er Nachbarschaft ein Schiff befindet. Es gilt ebenfalls die Spielfeldgrenzen zu berücksichtigen.

3. Wiederholen den vorherigen Schritt rekursiv an der nächsten Position in angegebener Orientierung. Die Schiffsgröße wird mit jedem Rekursionsschritt um 1 verringert, sodass aus dieser Angabe die Abbruchbedingung erzeugt werden kann.

4. Wurde in keinem Feld rekursiv eine Behinderung festgestellt, werden die Felder mit der Schiffskennung versehen.

Diese vier Verarbeitungsschritte werden für jedes zu platzierende Schiff aufgerufen. Jedoch ist nicht gewährleistet, dass eine zufällig generierte Position auch frei ist. Deshalb steht pro Schiffstyp ein maximales Kontingent an Versuchen zur Verfügung, das sich nach jedem erfolgreich platzierten Schiff wieder füllt. Sollte das Kontingent für einen Schiffstypen aufgebraucht worden sein, so wird das Gesamtkontingent für das gesamte Spielbrett reduziert und das gesamte Spielbrett wird zurückgesetzt. Sollte auch dieses

Kontingent aufgebraucht worden sein, wird eine Fehlermeldung per Exception ausgegeben.

4.3 Client - Prolog

5 Evaluation

6 Literatur