Technische Universität Braunschweig

ISF

Battleships

Volle Kraft voraus

Software-Entwicklungspraktikum (SEP)
Sommersemester 2016

Pflichtenheft

Auftraggeber
Technische Universität Braunschweig
Institut für Datentechnik und Kommunikationsnetze
Prof. Dr.-Ing Rolf Ernst
Hans-Sommer-Straße 66
D-38106 Braunschweig

Betreuer: Johannes Schlatow, Mischa Möstl

Auftragnehmer:

Name	E-Mail-Adresse
Majid Dashtiepielehroud	m.dashtiepielehroud@tu-braunschweig.de
Jana Luscher	jana.luscher@tu-braunschweig.de
Alexander Manegold	a.manegold@tu-braunschweig.de
Alexander Benedikt Ohnesorge	a.ohnesorge@tu-braunschweig.de
Jerome Rank	jerome.rank@tu-braunschweig.de
Mirjam Schulze	mirjam.schulze@tu-braunschweig.de

Braunschweig, 30. Mai 2016

BearbeiterInnenübersicht

Das vorliegende Dokument wurde von der gesamten Arbeitsgruppe bearbeitet. Die Arbeitsgruppe besteht aus Majid Dashtiepielehroud, Jana Luscher, Alexander Manegold, Alexander Benedikt Ohnesorge, Jerome Rank und Mirjam Schulze. Im Rahmen dieser Teamarbeit wurden Texte gemeinsam verfasst und korrigiert, so dass jedes Gruppenmitglied an jedem Kapitel mitgearbeitet hat. Somit fungiert jedes Gruppenmitglied als AnsprechpartnerIn.

Kapitel	**Autoren**	**Kommentare**
1	gesamtes Team	-
1.1	gesamtes Team	-
1.2	gesamtes Team	-
1.3	gesamtes Team	-
1.4	gesamtes Team	-
2	gesamtes Team	-
2.1	gesamtes Team	-
2.2	gesamtes Team	-
2.3	gesamtes Team	-
3	gesamtes Team	-
4	gesamtes Team	-
5	gesamtes Team	-
6	gesamtes Team	-
6.1	gesamtes Team	-
6.2	gesamtes Team	-
6.3	gesamtes Team	-
6.4	gesamtes Team	-
6.5	gesamtes Team	-
7	gesamtes Team	-
8	gesamtes Team	-
8.1	gesamtes Team	-
8.2	gesamtes Team	-
8.2.1	gesamtes Team	-
8.3	gesamtes Team	-
9	gesamtes Team	-

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Zielbestimmung

Das Projekt *Battleships - Volle Kraft voraus* findet im Rahmen des ▷ **Softwareentwicklungspraktikums (SEP)** 2016 statt. Der Auftraggeber des Projektes ist das *Institut für Datentechnik und Kommunikationsnetze* (*IDA*).

Bei *Battleships - Volle Kraft voraus* treten zwei Teams in einem an den Spieleklassiker ▷ ***Schiffe versenken*** angelehnten Spiel gegeneinander an. *Battleships - Volle Kraft voraus* folgt im Großen und Ganzen den allgemeinen Spielregeln des herkömmlichen Spiels *Schiffe versenken*. Somit verläuft das Spiel rundenbasiert. Außerdem ist das Spielziel unverändert und beinhaltet folglich alle Schiffe der Gegenpartei zu versenken und dabei in der eigenen Flotte noch mindestens ein nicht versenktes Schiff übrig zu haben.

Das Spielfeld beider Teams ist 8×8 Felder groß (*siehe* Abbildung 7.2 „LED Matrix: Spielstart ohne platzierte Schiffe“) und besteht damit aus 64 sogenannten Wasserfeldern. Jedem Team stehen als Flotte fünf Schiffe zur freien Platzierung auf dem eigenen Spielfeld zur Verfügung. Dies sind drei Schiffe der Länge zwei und zwei Schiffe der Länge drei. Die Länge der Schiffe ist in der Anzahl der zu belegenden Felder auf dem Spielfeld angegeben. Die Schiffe der Länge drei bilden eine gerade Linie und besitzen um den Mittelpunkt keinen Knick.

Nach dem Spielstart beginnt die sogenannte Initialisierungsphase. Innerhalb dieser Initialisierungsphase platziert jedes Team die eigenen Schiffe auf dem eigenen Spielfeld. Zuerst werden die Schiffe der Länge zwei und anschließend die Schiffe der Länge drei platziert. Die Schiffe dürfen sich dabei berühren, jedoch nicht überlappen und auch nicht diagonal platziert werden. Darüber hinaus müssen sie sich komplett im Spielfeld befinden und dürfen nicht über den Spielfeldrand hinaus ragen. Nachdem jedes Team die eigenen Schiffe gesetzt hat, beginnt die sogenannte Spielphase.

In der Spielphase ist jedes Team mit der Auswahl einer Aktion abwechselnd an der Reihe, wobei alle 10 Runden einem Team obendrein eine Bonusrunde mit einer weiteren Aktion zur Verfügung steht. Eine Aktion besteht dabei entweder aus der Option Schuss oder der Option Bewegung, welche in dieser Version von *Schiffe versenken* zusätzlich zur Verfügung steht. Die Aktion Schuss bezieht sich auf eine Koordinate des gegnerischen Spielfeldes. Bei einem Treffer auf ein gegnerisches Schiff ist das schießende Team erneut an der Reihe. Ein Schuss auf ein

Im Text glossarisierte Wörter sind bei Mauslick direkt mit dem Glossar (Kapitel 9) verlinkt. Bei der erstmaligen Verwendung sind diese glossarisierten Wörter zudem mit ▷ und fett markiert.

bereits vorher getroffenes Schiffsteil gilt nicht mehr als Treffer. Solange ein Schiff nicht getroffen wurde, gilt es als intakt. Wenn alle Felder eines Schiffs getroffen sind, gilt es als versenkt.

Für das Bewegen eines Schiffes stehen dem Team zwei Auswahlmöglichkeiten zur Verfügung: Fahren oder Drehung. Bei der Fahrt wird ein Schiff um ein Feld auf dem Spielfeld nach vorne oder nach hinten bewegt. Die Drehung eines Schiffes erfolgt um 90°entweder um die Achse des Schiffes (bei Schiffen der Länge drei) oder um einen der sogenannten Endpunkte (bei Schiffen der Länge zwei). Es können nur intakte Schiffe bewegt oder gedreht werden. Sowohl bei der Bewegung als auch bei der Drehung ist es nur erlaubt, Wasserfelder des Spielfeldes auszuwählen. Näher erklärt, bedeutet dies, dass keine Felder benutzt werden dürfen, auf denen intakte, getroffene oder versenkte Schiffe stehen.

Mit dem Versenken der gesamten gegnerischen Flotte bei gleichzeitigem Verbleib von mindestens einem nicht versenkten Schiff in der eigenen Flotte hat dieses Team einen Sieg errungen und beendet damit das Spiel.

Die Auftragnehmer realisieren eines dieser oben genannten Teams als einen autonom spielenden sogenannten ▷ **Spieleclient** und entwickeln hierzu das Softwareprodukt *Admiral Client*. Das Softwareprodukt *Admiral Client* wird als Modul innerhalb eines ▷ **Microkernel-Betriebssystems** auf einem ▷ **eingebetteten System** erstellt. Das eingebettete System besteht konkret aus einem vom Auftraggeber zur Verfügung gestelltem ▷ **Parallella Board**.

Das eigene Spielfeld wird über eine 8×8 große LED-Matrix visualisiert, welche an das Parallella Board angeschlossen ist. Hierfür wird innerhalb des *Admiral Client* folglich eine Schnittstelle zum Ansprechen der LED-Matrix implementiert.

Im Rahmen dieses Projekts werden als Bestandteil des *Admiral Client* mindestens zwei eigenständig spielende ▷ **Künstliche Intelligenzen (KIs)** entworfen. *Admiral Client* benötigt diese KIs, um autonome Entscheidungen bezüglich der Schiffsplatzierungen in der Initialisierungsphase und bezüglich der Aktionen Schuss oder Bewegung in der Spielphase zu treffen.

Der Auftraggeber stellt einen ▷ **Gameserver** bereit, welcher das Spielgeschehen sowie die Einhaltung der Spielregeln verwaltet. Die exakte Spezifikation des Gameservers inklusive der dortig implementierten Spiel- und Kommunikationsregeln ist in der Spezifikationsdatei des Auftraggebers zu finden[1]. Darüber hinaus stellt der Gameserver zwei Spielmodi zur Verfügung; den interaktiven Modus und den autonomen Modus. Welcher Modus aktiv ist, teilt der Gameserver dem *Admiral Client* nicht mit und ist damit auch der eingesetzten KI unbekannt. Im autonomen Modus spielt der *Admiral Client* beispielsweise gegen einen anderen Client, welcher von einer weiteren SEP-Gruppe beim IDA erstellt wird, und ebenfalls mit dem Gameserver via Netzwerk verbunden ist. Im interaktiven Modus spielt der *Admiral Client* direkt gegen einen Menschen.

[1] Link zur Gameserver-Spezifikation des IDA vom 26.06.2016

Die gesamte Kommunikation während des Spielverlaufs läuft über den bereitgestellten Gameserver nach einem ▷ **Request-Response-Prinzip** mittels des ▷ **User Datagram Protocols (UDP)** ab. Für die Umsetzung dieser Kommunikation wird innerhalb von *Admiral Client* eine Netzwerkschnittstelle implementiert. Wichtig ist hierbei zu erwähnen, dass der Begriff Gameserver und das Request-Response-Prinzip vom Auftraggeber verändert spezifiziert wurde. Dies bedeutet, dass der Gameserver nicht, wie sonst übliche Server, eine passive Rolle übernimmt und auf Anforderungen wartet, sondern der Gameserver versendet aktiv Nachrichten mit Anfragen. Im vorliegenden Fall versendet der Gameserver Requestnachrichten und *Admiral Client* antwortet darauf mit Responsenachrichten. Konkret bedeutet dies, dass der *Admiral Client* ausschließlich auf Anweisungen und Informationen vom Gameserver reagiert. Des Weiteren versendet der Gameserver Statusnachrichten, welche vom Spielclient unbeantwortet bleiben.

Den Spielstart erkennt *Admiral Client* anhand einer Requestnachricht zum Platzieren des ersten Schiffes innerhalb der Initialisierungsphase. Die vom *Admiral Client* eingesetzte KI berechnet eine günstige Lage zur Platzierung und sendet die Koordinaten der Endpunkte des Schiffes als Responsenachricht an den Gameserver zurück. Die Grundlage der Entscheidung für die KI zur Auswahl der Platzierung sind dabei zum einen zufällige Felder zum Verwirren des gegnerischen Teams und zur taktischen Verschleierung, als auch Felder mit größtmöglicher Bewegungsfreiheit. Bei einer fehlerhaften Platzierung weist der Gameserver per Statusnachricht auf diesen Fehler hin. *Admiral Client* kann die Platzierung eines Schiffs solange wiederholen, bis eine gültige Platzierung erfolgt ist. Bis alle Schiffe der Flotte platziert sind, erfolgt nach gültiger Platzierung eines Schiffes die Requestnachricht vom Gameserver für ein weiteres Schiff.

Den Beginn der Spielphase und damit das erfolgreiche Platzieren aller Schiffe erfährt *Admiral Client* durch eine Requestnachricht bezüglich der Auswahl einer Aktion oder durch eine Statusnachricht bezüglich eines Schusses des gegnerischen Teams auf das eigene Spielfeld. Ein Treffer ist dabei ein sogenannter *Hit* und ein Wasserschuss ein sogenannter *Miss*. Jedes Team wird nun abwechselnd vom Gameserver nach einer Aktion gefragt, welche innerhalb einer beschränkten Zeit beantwortet werden müssen. Ein Zug verfällt, wenn ein Team nicht innerhalb dieses Zeitkorridors eine gültige Responsenachricht versendet. Bezüglich fehlerhafter Schüsse oder Bewegungen versendet der Gameserver zwar Statusnachrichten, ansonsten werden fehlerhafte Responsenachrichten vom Gameserver jedoch ignoriert.

Gründe zur Entscheidungsfindung der KI bezüglich Schuss oder Bewegung werden im folgenden aufgezeigt:

Die KI wählt die Option Schuss aus, um die Bewegungsfreiheit gegnerischer Schiffe einzuschränken und ein Raster zu setzen. Grundlage sind sowohl bereits getroffene gegnerische Schiffe, als auch vorhergehende Bewegungsaktionen des gegnerischen Teams. Zur Verschleierung der Taktik und Verwirrung werden in wahlloser Wiederholung zufällige Schüsse gesetzt.

Die Grundlage der Entscheidung für die KI zur Auswahl einer Bewegung ist die Analyse gegnerischer Schüsse, sowie das Ziel, intakten Schiffen einen möglichst hohen Bewegungsgrad zu erhalten. Zur Verschleierung der Taktik und Verwirrung werden in wahlloser Wiederholung zufällige Bewegungen gesetzt.

Sobald alle Schiffe eines Teams versenkt wurden, ist das aktuelle Spiel beendet. *Admiral Client* muss dies allerdings eigenständig erkennen, da der Gameserver dies nicht aktiv mitteilt.

Für das verwendete eingebettete System wird das angepasste Microkernel-Betriebssystem konkret generiert.[2] Dies geschieht mittels des offenen ▷ **Genode OS** ▷ **Frameworks**.

Zusammenfassend ist das Ziel ein Softwarepaket, eingebettet in Genode OS, für ein Parallella Board, bestehend aus einem autonom spielenden *Admiral Client* mit mindestens zwei KIs zur Steuerung der Spielaktionen, einem Treiber für die 8×8 LED-Matrix und einem Netzwerkprotokoll für die UDP-Kommunikation mit dem Gameserver.

1.1 Musskriterien

In diesem Abschnitt wird erläutert, welche Kriterien das Spiel auf jeden Fall erfüllen muss. Diese sind nötig, damit das Softwareprodukt nutzbar ist.

⟨$RM1$⟩ Die Software muss über Genode OS auf dem Parallella Board lauffähig sein.
⟨$RM2$⟩ Der *Admiral Client* startet nach dem Einschalten des Boards und dem Hochfahren von Genode OS selbstständig.
⟨$RM3$⟩ Der *Admiral Client* reagiert eigenständig als autonomes System auf Anfragen, Informationen und Anweisungen des Gameservers.
⟨$RM4$⟩ Für die Kommunikation des *Admiral Client* mit dem Gameservers muss ein fehlerfrei arbeitendes UDP/IP-Netzwerkschnittstellenprotokoll implementiert werden.
⟨$RM5$⟩ Für die Visualisierung des Spielgeschehens muss ein fehlerfrei arbeitender Treiber eine 8×8 LED-Matrix ansteuern.
⟨$RM6$⟩ Die Ausgabe auf der LED-Matrix muss in mindestens drei verschiedenen Farben erfolgen (blau für Wasser, grün für intaktes Schiffsteil, rot für zerstörtes Schiffsteil).
⟨$RM7$⟩ Die KI des *Admiral Client* muss sämtliche Spielregeln des Spiels implementiert haben.
⟨$RM8$⟩ Die KI des *Admiral Client* muss nach Aufforderung des Gameservers im Rahmen ihres Spielzuges über die Schiffsplatzierung auf dem Spielfeld im Sinne der Spiellogik entscheiden können.
⟨$RM9$⟩ Die KI muss nach Aufforderung des Gameservers im Rahmen ihres Spielzuges im Sinne der Spiellogik zwischen den Aktionen Schuss oder Bewegung entscheiden.
⟨$RM10$⟩ Für die Entscheidung der KI über Schuss oder Bewegung muss die KI die Informationen

[2] Das Microkernel-Betriebssystem wird im Folgenden kurz *Genode OS* genannt.

des Gameservers über vorangegangene eigene und gegnerische Aktionen verarbeiten können.
⟨*RM*11⟩ Die KI muss aufgrund der erhaltenen Informationen des Gameservers selbstständig erkennen, wann sie gewonnen oder verloren hat.

1.2 Sollkriterien

Hier wird erläutert, welche Leistungen das Softwareprodukt nicht unbedingt erfüllen muss. Sie sollten jedoch realisiert werden, um die Projektziele zu erreichen.

⟨*RS*1⟩ Die KI soll lernfähig sein und ihre Spieltaktik aufgrund der gegnerischen Spielaktionen anpassen.
⟨*RS*2⟩ Die LED-Matrix soll den Start des Spiels und das Spielende (Sieg oder Niederlage) mit einem kurzen Aufleuchten aller LEDs darstellen. Blaues Aufleuchten für Start, grünes Aufleuchten für Sieg und rotes Aufleuchten für Niederlage.

1.3 Kannkriterien

In diesem Unterkapitel wird dargelegt, welche Leistungen zwar im Produkt enthalten sein können, jedoch für die Zielerreichung nicht notwendig sind. Dies sind Leistungen, die nur bearbeitet werden, wenn am Ende der Bearbeitungszeit noch Kapazitäten übrig bleiben.

⟨*RC*1⟩ Es ist möglich, die LED-Matrix in anderen oder mehreren Farben darstellen zu lassen, z.B. zur Unterscheidung der Schiffe.
⟨*RC*2⟩ Die LED-Matrix kann bei einem erhaltenen Treffer blinken.
⟨*RC*3⟩ Die Visualisierung auf der LED-Matrix kann den letzten erhaltenen Schuss in passender Farbe blinkend darstellen.
⟨*RC*4⟩ Da der *Admiral Client* mehrere KIs zur Verfügung stellt, kann bei mehreren Spieldurchläufen eine andere ausgewählt werden.
⟨*RC*5⟩ Die LED-Matrix reagiert auf eine Error-Meldung und blinkt auf.
⟨*RC*6⟩ Die LED-Matrix zeigt einen erfolgreichen und fehlerfreien Systemstart an.

1.4 Abgrenzungskriterien

In diesem Abschnitt wird erläutert, welche Leistungen das Produkt auf keinen Fall abdecken wird.

⟨*RW*1⟩ Der *Admiral Client* ist für keine andere Umgebung als für das ausgewählte Parallella Board mit dem spezifizierten Gameserver vorgesehen.
⟨*RW*2⟩ Der Gameserver darf die Spielregeln nicht verändern.
⟨*RW*3⟩ Das Spiel findet ausschließlich über ein lokales Netzwerk mit dem Gameserver statt (nicht über externe Netzwerke oder externe Server).
⟨*RW*4⟩ Es ist nicht möglich, mit mehr als zwei Parteien zu spielen.
⟨*RW*5⟩ Es gibt auf dem Parallella Board keinen Zugriff auf die Software; Fehler können somit nicht selbstständig nach dem Kompilieren behoben werden.
⟨*RW*6⟩ Der *Admiral Client* kann nicht modifiziert werden und wählt eigenständig die passend zu scheinende Spieltaktik.
⟨*RW*7⟩ Während *Admiral Client* ausgefeilte Algorithmen für seine Entscheidungen verwendet, ist es nicht möglich einen Sieg der KI zu garantieren.
⟨*RW*8⟩ Da das Produkt ein autonomes System darstellt, wird auf ein User-Handbuch verzichtet.

2 Produkteinsatz

Dieses Kapitel dient dazu, den Einsatzbereich der Software näher zu spezifizieren. Des Weiteren werden die Zielgruppen beschrieben, für die das Produkt entwickelt wird. Abschließend werden die Rahmenbedingungen erläutert, die erfüllt sein müssen, um das Softwareprodukt zu nutzen.

2.1 Anwendungsbereiche

Das Softwareprodukt *Battleships - Volle Kraft voraus* dient auf der einen Seite als Anschauungsobjekt der Implementierung von Modulen innerhalb des Genode OS Frameworks. Auf der anderen Seite dient es auch zur praktischen Umsetzung von KI-Algorithmen. Als Einsatzort für das Produkt sind zum einen Präsentationen im Rahmen des SEP 2016 zu nennen, zum anderen aber auch Werbezwecke für Genode OS. Des Weiteren kann es auch zur allgemeine Unterhaltung eingesetzt werden.

2.2 Zielgruppen

Zu der Zielgruppe des Produktes zählen Personen, die sich für die betriebssystemnahe Softwareentwicklung auf einem eingebetteten System interessieren und eine Anschauung der Leistungsfähigkeit des Parallella Boards erhalten wollen. Darüber hinaus spricht das Produkt Personen an, die sich für KI-Entwicklung begeistern und einen Eindruck von deren Möglichkeiten und Herausforderungen erhalten wollen. Eine konkrete Zielgruppe besteht aus der weiteren SEP-Gruppe, die unabhängig einen eigenen Client entwickelt. Zudem besteht eine nicht zu vernachlässigende Zielgruppe aus jenen spielbegeisterten Personen, welche mittels interaktivem Modus des Gameservers ein Duell *Schiffe versenken* gegen den Client wagen wollen.

2.3 Betriebsbedingungen

- Das Parallella Board, sowie die LED-Matrix müssen zum Spielen des Spiels vorhanden sein.
- Das Genode OS mit dem Client Modul muss als SD-▷ **Image** vorliegen.

- Die Stromversorgung für das Board und die LED-Matrix muss gewährleistet sein.
- Der Spielclient benötigt einen permanenten Zugang zum Gameserver, welcher das Spiel koordiniert. Folglich muss das Parallela Board mit dem Gameserver mittels Netzwerkkabel verbunden sein.
- Es muss am Gameserver eine Gegenpartei zum Client vorhanden sein, entweder ein weiterer, per Netzwerk angeschlossener Client oder eine Person.
- Für den interaktiven Modus werden eine Computermaus, eine Tastatur und ein Monitor benötigt.
- Eine Beobachtung des Systems ist im autonomen Betrieb nicht nötig.
- Prinzipiell ist der unbeaufsichtigte Betrieb möglich. Da das Spiel jedoch mit seinen physikalischen Bestandteilen zum Spielen vor Ort sein muss, muss sichergestellt werden, dass diese nicht entwendet werden.
- Da die Komponenten empfindliche, elektrische Systeme beinhalten, dürfen diese keiner Feuchtigkeit oder anderen extremen Witterungsbedingungen ausgesetzt werden. Wie die Hardware, und damit die darauf laufende Software auf Veränderungen der Druckverhältnisse oder elektromagnetische Schwankungen reagiert, ist nicht gewährleistend vorhersagbar.

3 Produktübersicht

In diesem Kapitel werden die Produktfunktionen beschrieben und in einem Use-Case-Diagramm visualisiert. Es wird skizziert *was* die Software im Wesentlichen tut. Dies sind sogenannte Use-Cases (grafisch als *Ellipsen* dargestellt). Mittels Verbindungslinien wird gezeigt, in welcher Beziehung diese Use-Cases zueinander stehen. Zwei Use-Cases und deren Zusammenspiel werden mit Hilfe von Aktivitätsdiagrammen detaillierter abgebildet.

Das Softwareprodukt besteht aus einem Client, *Admiral Client*, der als Akteur mit dem Gameserver per Netzwerkschnittstelle verbunden ist. Wie in Abbildung 3.1 dargestellt, wartet *Admiral Client* auf den Empfang eines Datagramms (per *UDP*). Dieses Datagramm ist eine Request- oder eine Statusnachricht vom Gameserver, welche *Admiral Client* verarbeiten muss. Requestnachrichten beantwortet *Admiral Client* mit dazu passenden Responsenachrichten, während Statusnachrichten lediglich zur Kenntnis genommen werden.

Die zuvor erwähnten Nachrichten werden in Form von Datagrammen mittels UDP versendet und sind alle wie folgt aufgebaut:

ID	Type	Data
4 Byte	2 Byte	variable

Der Gameserver nummeriert jedes Datagramm mit einer Sequenz-ID, die mit jedem neuen Datagramm inkrementiert wird. Die Sequenz-ID steht am Anfang des Datagramms und ist 4 Byte lang. Wenn der Gameserver also eine Nachricht mit einer bestimmten Sequenz-ID an den *Admiral Client* sendet, enthält die darauf folgende Response-Nachricht die gleiche Sequenz-ID. Das Datagramm, das in einem Spiel als erstes versendet wird, enthält die Sequenz-ID 1. Der Nachrichtentyp wird durch das Type-Feld identifiziert und gibt somit an, wie das Data-Feld formatiert ist. Hierbei kann es sich um Request- und Status-Nachrichten des Gameservers oder um Response-Nachrichten des Admiral Client handeln.

Da *Admiral Client* autonom ohne Userinteraktion reagieren muss, ist diesbezüglich eine KI implementiert.

Sobald *Admiral Client* per Requestnachricht dazu aufgefordert wird, ein Schiff zu platzieren (während der sogenannten Initialisierungsphase), berechnet die KI eine geeignete Platzierung und sendet die Information als Datagramm zurück. Infolgedessen wird die LED-Matrix mit dieser Platzierung aktualisiert.

Fordert die Requestnachricht dazu auf, eine Aktion zu wählen (während der sogenannten Spielphase), muss die KI sich entscheiden, einen Schuss auf das gegnerische Spielfeld abzugeben oder eines der eigenen (intakten) Schiffe zu bewegen. Bei der Bewegungsaktion gibt es die Optionen Fahren (um ein Feld vor oder zurück) oder Drehen (als 90°-Drehung[3]). Die Aktionsentscheidung wird per Datagramm zurückgesendet und bei einer Schiffsbewegung wird die LED-Matrix aktualisiert.

Enthält das vom Admiral Client empfangene Datagramm eine Statusnachricht, wird diese ohne Responsenachricht verarbeitet. Die Statusnachricht enthält unter anderem Fehlermeldungen, welche in der Initialisierungsphase dazu auffordern, ein fehlerhaft platziertes Schiff erneut und dieses Mal fehlerfrei zu setzen. Des Weiteren informieren Fehlermeldungen innerhalb der Spielphase über falsch gesetzte Schüsse oder ungültige Bewegungszüge. Infolgedessen muss gegebenenfalls die LED-Matrix auf den vorherigen Stand zurückgesetzt werden. Zur Berücksichtigung kommender Entscheidungen müssen auch die KIs über die fehlerhaften Entscheidungen informiert werden. Außerdem werden Treffer der gegnerischen Partei gemeldet, welche ebenso von der KI-Spiellogik verarbeitet werden müssen und gegebenenfalls zu LED-Matrix-Aktualisierungen führen. Zusätzlich wird vom Gameserver das Resultat der eigenen Schüsse mitgeteilt.

[3] Eine 90°-Drehung erfolgt um einen der Endpunkte (Schiffe der Länge 2) oder um den Mittelpunkt (Schiffe der Länge 3).

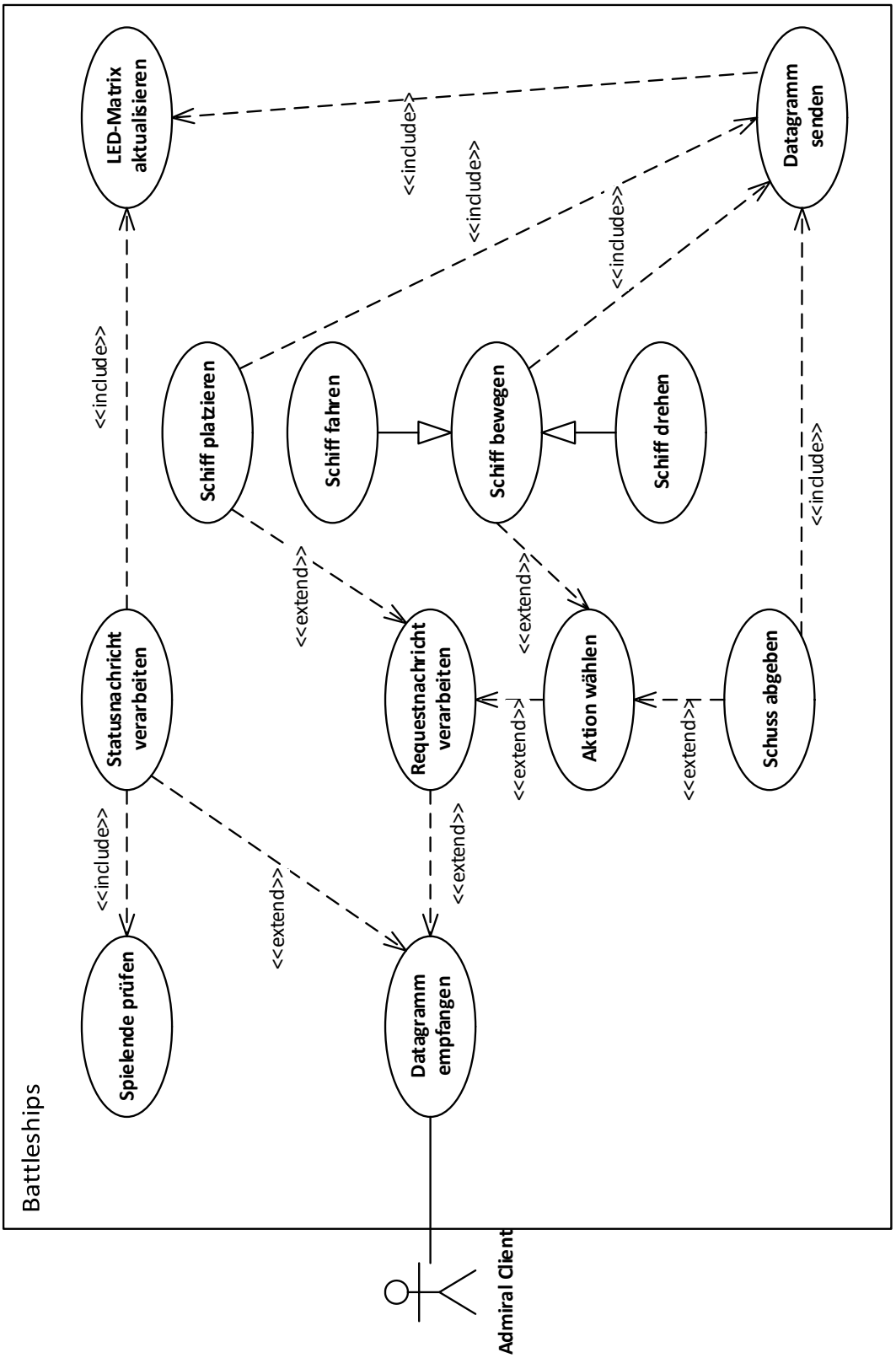

Abbildung 3.1: Use-Case-Diagramm Battleships

Im Anschluss an die Darstellung des Verhaltens mit allen auftretenden Anwendungsfällen in Abbildung 3.1 wird der konkrete nicht-triviale Use-Case *Schiffe platzieren* in einem Aktivitätsdiagramm, siehe Abbildung 3.2, dargestellt.

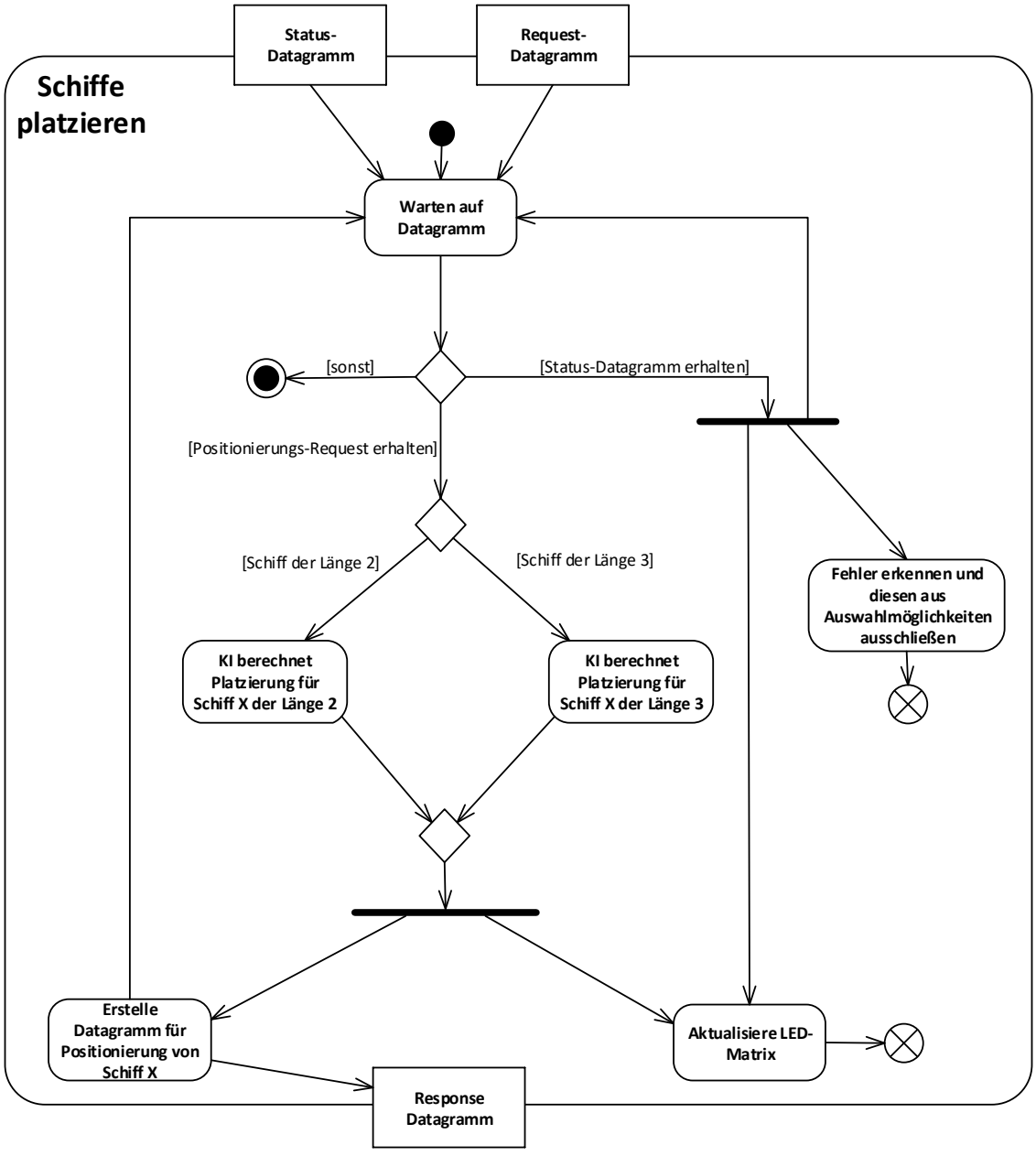

Abbildung 3.2: Aktivitätsdiagramm „Schiffe platzieren“

Die Darstellung in Abbildung 3.3 beschreibt den nicht-trivialen Use-Case *Aktion wählen* in einem weiteren Aktivitätsdiagramm.

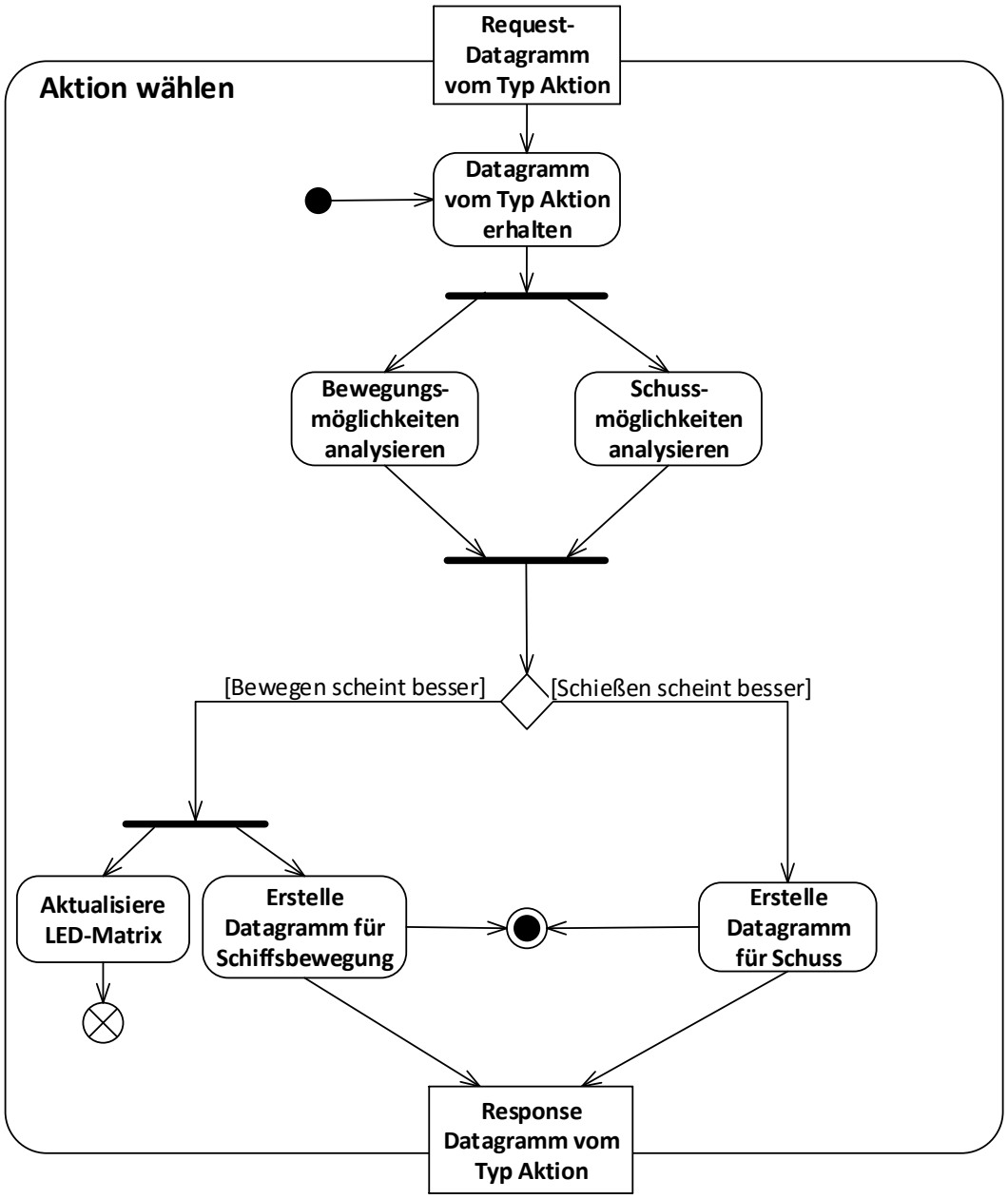

Abbildung 3.3: Aktivitätsdiagramm „Aktion wählen“

4 Produktfunktionen

In diesem Kapitel werden die Anwendungsfälle, die im Kapitel 3 Produktübersicht beschrieben wurden, konkretisiert. Hierbei ist nochmals besonders darauf hinzuweisen, dass nur die Funktionen des Clients im Rahmen des Projekts entwickelt werden.

Datagramm empfangen $\langle F10 \rangle$

Anwendungsfall: Empfangen eines Datagramms vom Gameserver.

Anforderung: $\langle RM1 \rangle$, $\langle RM2 \rangle$, $\langle RM4 \rangle$

Ziel: Erfolgreiches Empfangen von Nachrichten des Gameservers.

Vorbedingungen: Das Parallella Board muss eingeschaltet sein. Das Betriebssystem muss geladen sein. Die Verbindung zum Gameserver muss hergestellt sein.

Nachbedingung Erfolg: Datagramm liegt dem Client zur weiteren Verarbeitung vor und Datagrammtyp ist bekannt.

Nachbedingung Fehlschlag: Client reagiert nicht auf Anfrage des Servers.

Akteure: Client, Gameserver

Auslösendes Ereignis: Gameserver sendet eine Nachricht an den Client.

Beschreibung:

1. Datagramm wird über die Netzwerkschnittstelle empfangen.
2. Erkennung ob es sich bei dem Datagramm um eine Statusnachricht oder einen Request handelt.

Requestnachricht verarbeiten $\langle F20 \rangle$

Anwendungsfall: Bearbeitung eines Requests

Anforderung: $\langle RM3 \rangle$

Ziel: Klassifizierung des Requests und Weitergabe an zugehörige KI-Komponente

Vorbedingung: Datagramm muss empfangen worden sein. Datagramm wurde als Requestnachricht identifiziert.

Nachbedingung Erfolg: Request wird an eine andere KI-Komponente weitergegeben, die ihn bearbeiten kann.

Nachbedingung Fehlschlag: Keine Weitergabe findet statt.

Akteure: Client

Auslösendes Ereignis: Datagram vom Typ Request wurde empfangen.

Beschreibung:

1. Das Type-Feld des Datagramms wird ausgewertet.
2. Ansprechen der KI-Komponente mit entsprechendem Request-Typ.

Statusnachricht verarbeiten $\langle F30 \rangle$

Anwendungsfall: Bearbeitung einer Statusnachricht.

Anforderung: $\langle RM3 \rangle$, $\langle RM10 \rangle$

Ziel: Klassifizierung der Statusnachricht und Weitergabe der Information.

Vorbedingung: Datagramm muss empfangen worden sein. Datagramm wurde als Statusnachricht identifiziert.

Nachbedingung Erfolg: Client kann die Statusnachricht erkennen und die enthaltenen Informationen weitergeben.

Nachbedingung Fehlschlag: Statusnachricht wird nicht verarbeitet.

Akteure: Client

Auslösendes Ereignis: Datagramm vom Typ Statusnachricht wurde empfangen.

Beschreibung:

1. Die Art der Statusnachricht wird ermittelt.
2. Fehler bei eigenen Aktionen werden an die KI weitergegeben.
3. Schüsse des Gegners werden ebenfalls an die KI weitergegeben.

Aktion wählen $\langle F40 \rangle$

Anwendungsfall: KI wählt Aktion aus.

Anforderung: $\langle RM9 \rangle$, $\langle RM10 \rangle$

Ziel: Auswahl der Spielaktion

Vorbedingung: Ein Request für eine Aktion muss empfangen worden sein.

Nachbedingung Erfolg: Es wird entweder ein Schuss abgegeben oder ein Schiff bewegt.

Nachbedingung Fehlschlag: KI wählt keine Aktion aus und Request wird nicht weiter bearbeitet.

Akteure: Client

Auslösendes Ereignis: Requestnachricht vom Typ „Aktion“ wurde empfangen.

Beschreibung:

1. KI entscheidet die nächste Aktion.
2. Schuss abgeben und Schiff bewegen stehen der KI zur Auswahl.

Schiff platzieren $\langle F50 \rangle$

Anwendungsfall: Schiff wird platziert.

Anforderung: $\langle RM7 \rangle$, $\langle RM8 \rangle$

Ziel: Platzierung eines Schiffs.

Vorbedingung: Ein Request für eine Schiffsplatzierung muss empfangen worden sein.

Nachbedingung Erfolg: Schiffsplatzierung wird vorgenommen.

Nachbedingung Fehlschlag: Client kann Platzierungs-Request nicht beantworten.

Akteure: Client

Auslösendes Ereignis: Requestnachricht vom Typ „Positionierung Schiff“ wurde empfangen.

Beschreibung:

1. KI bestimmt aus ihrer Sicht optimale Platzierung des Schiffs.
2. Platzierung besteht aus Anfangs- und Endpunkt des Schiffs.
3. Punkte werden als X- und Y-Koordinate dargestellt.

Schiff bewegen $\langle F60 \rangle$

Anwendungsfall: Auswahl eines Schiffs und der Bewegungsart
Anforderung: $\langle RM7 \rangle$
Ziel: Bewegung eines Schiffs
Vorbedingung: KI muss sich für das Bewegen eines Schiffs entschieden haben.
Nachbedingung Erfolg: Schiff wird bewegt.
Nachbedingung Fehlschlag: Request wird nicht beantwortet.
Akteure: Client
Auslösendes Ereignis: KI entscheidet sich für Bewegung.
Beschreibung:

1. KI wählt ein Schiff aus.
2. KI entscheidet zwischen Fahren und Drehen.

Schiff fahren $\langle F70 \rangle$

Anwendungsfall: Schiff fährt
Anforderung: $\langle RM7 \rangle$
Ziel: Bestimmung der Koordinaten nach dem Fahren.
Vorbedingung: KI muss sich für Fahren entschieden haben.
Nachbedingung Erfolg: Schiff wird um ein Feld verschoben.
Nachbedingung Fehlschlag: Request wird nicht beantwortet.
Akteure: Client
Auslösendes Ereignis: KI hat sich für das Fahren entschieden.
Beschreibung:

1. Schiff fährt ein Feld vorwärts oder rückwärts.
2. Koordinaten nach dem Fahren werden bestimmt.

Alternativen: Schiff drehen

Schiff drehen $\langle F80 \rangle$

Anwendungsfall: Schiff dreht sich.
Anforderung: $\langle RM7 \rangle$
Ziel: Bestimmung der Koordinaten nach dem Drehen.
Vorbedingung: KI muss sich für Drehen entschieden haben.
Nachbedingung Erfolg: Schiff wird gedreht.
Nachbedingung Fehlschlag: Request wird nicht beantwortet.
Akteure: Client
Auslösendes Ereignis: KI hat sich für das Drehen entschieden.
Beschreibung:

1. Schiff dreht sich im Fall der Länge 3 um 90 Grad um seinen Mittelpunkt
2. Schiff dreht sich im Fall der Länge 2 um 90 Grad um einen Endpunkt
3. Koordinaten nach dem Drehen werden bestimmt.

Alternativen: Schiff fahren

Schuss abgeben $\langle F90 \rangle$

Anwendungsfall: Schuss auf das gegnerische Spielfeld wird abgeben.
Anforderung: $\langle RM7 \rangle$
Ziel: Schuss auf das gegnerische Spielfeld.
Vorbedingung: KI muss sich für Schießen entschieden haben.
Nachbedingung Erfolg: Schuss wird abgegeben.
Nachbedingung Fehlschlag: Request wird nicht beantwortet.
Akteure: Client
Auslösendes Ereignis: KI entscheidet sich für den Beschuss.
Beschreibung:

1. KI wählt Zielkoordinaten des Schusses

LED-Matrix aktualisieren $\langle F100 \rangle$

Anwendungsfall: LED-Matrix wird aktualisiert.

Anforderung: $\langle RM5 \rangle$, $\langle RM6 \rangle$

Ziel: Darstellung des eigenen Spielfeldes.

Vorbedingung: Änderung am eigenen Spielfeld ist aufgetreten.

Nachbedingung Erfolg: LEDs werden dem aktuellen eigenen Spielfeld entsprechend angepasst.

Nachbedingung Fehlschlag: Anzeige der LEDs entspricht nicht dem aktuellen eigenen Spielfeld.

Akteure: Client

Auslösendes Ereignis: Schiff bewegen, Schiff platzieren, beschossen werden.

Beschreibung:

1. LED Matrix stellt das eigene Spielfeld dar.
2. Aktualisierung wird über Schnittstelle zur LED-Matrix vorgenommen.

Spielende prüfen $\langle F110 \rangle$

Anwendungsfall: Spielende wird geprüft.

Anforderung: $\langle RM11 \rangle$

Ziel: Anhalten der KI im Falle eines Spielendes

Vorbedingung: Statusnachricht muss empfangen worden sein.

Nachbedingung Erfolg: Client hält, wenn das Spiel zu Ende ist und läuft weiter wenn das Spielende noch nicht eingetreten ist.

Nachbedingung Fehlschlag: Client erkennt das Ende des Spiels nicht und läuft weiter.

Akteure: Client

Auslösendes Ereignis: Statusnachricht wurde empfangen.

Beschreibung:

1. Client bestimmt anhand der aktuellen Spielfelder ob das Spiel beendet ist.

Datagramm senden $\langle F120 \rangle$

Anwendungsfall: Datagramm wird an den Gameserver übermittelt.

Anforderung: $\langle RM4 \rangle$

Ziel: Datenübermittlung an den Gameserver

Vorbedingung: Verbindung zum Gameserver muss existieren und es muss eine Aktion oder Platzierung vorgenommen worden sein.

Nachbedingung Erfolg: Erfolgreiche Übermittlung eines Datagramms an den Gameserver.

Nachbedingung Fehlschlag: Gameserver sendet Error zurück.

Akteure: Client, Gameserver

Auslösendes Ereignis: Dem Client liegt ein Datagramm für den Gameserver vor.

Beschreibung:

1. Aktion oder Platzierung wird in Datagramm umgewandelt.
2. Datagramm wird an Gameserver übermittelt.

5 Produktdaten

Für die Arbeitsweise des Clients ist ▷ **persistente Datenspeicherung** weder vorgesehen noch erforderlich, vor allem aber auch nicht ohne Weiteres möglich, da keine Festplatte im System existiert. Jegliche Informationen, die der Client - beziehungsweise vor allem die jeweilige KI - verwendet, sind entweder kompilierter Code innerhalb der Software oder werden im ▷ **RAM** abgelegt und sind ▷ **flüchtig** konzipiert, also nach einem Neustart des Systems nicht mehr vorhanden.

Um den Stand des Spiels auf der 8×8 LED-Matrix auszugeben, muss das eigene Spielfeld mit den Ausgabeinformationen gespeichert werden. Des Weiteren sind Daten zur Auswahl der eigenen Schuss-Aktion sowie Daten zu vergangenen und zukünftigen, gegnerischen Schuss-Aktionen nötig. Zusätzlich muss die Platzierung der Schiffe gespeichert werden, damit die KI sinnvolle Spielzüge berechnen kann.

Während nur die wenigsten der folgenden Informationen notwendig wären, um einen reibungslosen Spielablauf zu garantieren, wird hier die Möglichkeit genutzt, die KI mit zusätzlichen Informationen zu versorgen, damit diese bessere Entscheidungen treffen kann. Bei den Daten ist zu bedenken, dass das eigene Feld deutlich einfacher aufgebaut ist, da dieses nur als Abbildung dient. Im gegnerischen Feld gibt es Prognosen, also Felder, über die keine sichere Auskunft gegeben werden kann, *was* auf diesem Feld genau ist.

eigenes Spielfeld $\langle D10 \rangle$
Daten des Spielfelds (max. 1 Spielfeld):

- Art des Spielfeldes (0 = unbekannt, 1 = Wasserfeld, 2 = unversehrtes Schiffsfeld, 3 = getroffenes Schiffsfeld)
- Rundennummern der gegnerischen Schüsse
- Position des letzten gegnerischen Schusses
- Position des letzten gegnerischen Treffers
- prognostizierte Auswahlmöglichkeiten der Platzierung gegnerischer Schüsse
- prognostizierte Platzierung des nächsten gegnerischen Schusses

gegnerisches Spielfeld $\langle D20 \rangle$
Daten des Spielfelds (max. 1 Spielfeld):

- Art des Spielfeldes (0 = unbekannt, 1 = Wasserfeld, 2 = unversehrtes Schiffsfeld, 3 = getroffenes Schiffsfeld)
- Rundennummern der eigenen Schüsse
- Position des letzten abgegebenen Schusses
- Position des letzten abgegebenen Treffers
- prognostizierte Auswahlmöglichkeiten der Bewegung gegnerischer Schiffe
- prognostizierte Position einer konkreten gegnerischen Schiffsbewegung
- Auswahlmöglichkeiten der Platzierung eigener Schüsse
- ausgewählte Platzierung des eigenen Schusses

Platzierung eigenes Schiff $\langle D30 \rangle$
Diese Daten speichern Informationen über die Position von Schiffen im Spiel. Bei der Platzierung eines Schiffes wird immer zuerst eine Koordinate genannt und danach die zweite. Hieraus lässt sich also ein „Anfang“ und ein „Ende“ des Schiffes (Bug und Heck) ableiten. Diese Unterscheidung ist für das Spiel selbst nicht weiter relevant, dient hier also nur zur Erläuterung, dass mindestens zwei verschiedene Koordinaten nötig sind, um die Lage eines Schiffes genau zu speichern. (max. 5 Schiffe in der eigenen Flotte):

- Bug des Schiffes („Anfangspunkt“)
- Heck des Schiffes („Endpunkt“)
- Mittelpunkt des Schiffes bei Schiffen ungerade Länge (für Drehungen wichtig)
- mögliche Bewegungsfelder-Endpunkte zur Berechnung für die KI

6 Nichtfunktionale Anforderungen

In diesem Kapitel werden die nichtfunktionalen Anforderungen des Softwareproduktes festgelegt, das heißt, in welcher Qualität das Softwareprodukt vorliegen soll. Die aufgezählten Anforderungen beziehen sich auf den ISO Standard 9126.[4]

6.1 Funktionalität

Produktqualität	sehr gut	gut	normal	nicht relevant
Angemessenheit		x		
Richtigkeit	x			
Interoperabilität	x			
Ordnungsmäßgkeit	x			

Angemessenheit

Für die einzelnen Komponenten sollte *Angemessenheit* mindestens *gut* zu werten sein, da jede Komponente für ihre Funktion geeignet sein muss.

Richtigkeit und Ordnungsmäßgkeit

Die *Richtigkeit* und *Ordnungsmäßgkeit* sollten *sehr gut* sein. Bei Verletzung der Spielregeln oder falschen Berechnungen kann die komplette KI nicht mehr vernünftig arbeiten. Fehlerhafte Response-Nachrichteninhalte können zum Verfall von Spielzügen führen. Fehler in der LED-Matrix-Treiberimplementierung verursachen Anzeigeprobleme und Probleme mit dem Netzwerkprotokoll stören die Kommunikation mit dem Gameserver.

Interoperabilität

Die *Interoperabilität* sollte *sehr gut* sein, da der Client mit den Schnittstellen ohne Probleme interagieren muss.

[4] http://www.iso.org/iso/catalogue_detail.htm?csnumber=22749

6.2 Sicherheit

Produktqualität	sehr gut	gut	normal	nicht relevant
Zuverlässigkeit		x		
Reife			x	
Fehlertoleranz				x
Wiederherstellbarkeit	x			

Zuverlässigkeit

Die *Zuverlässigkeit* ist *gut* bewertet, da das Leistungsniveau gehalten werden kann. Denn es erfolgt ausschließlich durch den Gameserver ein Zugriff auf den Client, beziehungsweise auf das System. Außerdem ist diese Netzwerkkommunikation auf Request- und Status-Nachrichten vom Gameserver beschnitten. Darüber hinaus erfolgt keine persistente Datenspeicherung, die das Leistungsniveau gefährden könnte.

Wiederherstellbarkeit

Da das Produkt als Eingebettetes System konzipiert ist, ist es bei jedem Neu-Einschalten ohne Weiteres im richtigen Zustand. Folglich ist eine *Wiederherstellbarkeit* des Systems mit *sehr gut* gegeben.

6.3 Benutzbarkeit

Produktqualität	sehr gut	gut	normal	nicht relevant
Verständlichkeit				x
Erlernbarkeit				x
Bedienbarkeit				x
Effizienz	x			
Zeitverhalten	x			
Verbrauchsverhalten	x			

Verständlichkeit, Erlernbarkeit, Bedienbarkeit

Das Parallella Board benötigt lediglich eine Stromversorgung und der Client ist per Netzwerkkabel automatisch mit den Gameserver verbunden. Da das Produkt keine weitere Userbeteiligung beinhaltet, sind die Punkte *Verständlichkeit*, *Erlernbarkeit* und *Bedienbarkeit* nicht relevant.

Effizienz, Zeitverhalten, Verbrauchsverhalten

Jedoch sind die *Effizienz* sowie das *Zeit-* und *Verbrauchsverhalten* mit *sehr gut* gekennzeichnet, da der KI für die Berechnungen der Aktionsauswahl nur begrenzte Rechenzeit und Ressourcen, siehe Unterabschnitt 8.2.1 *Parallella Board*, zur Verfügung stehen. Bei Zeitüberschreitung verfällt die angefragte Aktion.

6.4 Änderbarkeit

Produktqualität	sehr gut	gut	normal	nicht relevant
Analysierbarkeit		x		
Modifizierbarkeit		x		
Stabilität	x			
Prüfbarkeit				x
Übertragbarkeit			x	
Anpassbarkeit			x	
Installierbarkeit				x
Konformität			x	
Austauschbarkeit			x	

Analysierbarkeit

Die *Analysierbarkeit* ist mit *gut* gekennzeichnet, da der Programmquellcode verständlich kommentiert, sowie gut analysierbar sein sollte, um Änderungen ohne Probleme und mit geringem Zeitaufwand implementieren zu können.

Modifizierbarkeit

Da der Punkt *Angemessenheit* (6.1) mit *gut* bewertet wurde, ist auch die *Modifizierbarkeit*, also der Aufwand für potenzielle Änderungen, mit *gut* zu bewerten.

Stabilität

Mögliche Änderungen am Programm-Quellcode sollen keinen Einfluss auf die Stabilität der Software haben, sodass immer ein reibungsloser Spielverlauf sichergestellt werden kann. Deshalb ist die *Stabilität* mit *sehr gut* notiert. Beispielsweise wären unter Umständen fehlerhafte Schiffsplatzierungen für den Gameserver, sowie Mängel in der Netzwerkimplementierung oder des LED-Matrix-Treibers für die Hardware belastend. Weiterhin sind Stabilitätskomplikationen besonders für einen im interaktiven Modus spielenden User sehr ärgerlich.

Installierbarkeit

Der Client ist ein Modul innerhalb von Genode OS, welches als Image auf einer SD-Karte vorliegt. Die SD-Karte muss nur fachgerecht in den SD-Karten-Slot des Parallella Board gesteckt werden. Der Client startet automatisch beim Einschalten des Boards. Die Installierbarkeit ist zwar damit *sehr gut* gegeben, aber im engeren Sinne mit *nicht relevant* markiert, da produkttechnisch keine installationsbedingten Anpassungen vorgenommen werden müssen.

6.5 Qualitätsanforderungen

Im Folgenden werden Qualitätsanforderungen an das Softwareprodukt in Bezug auf die zuvor angegebenen nichtfunktionalen Anforderungen formuliert.

- $\langle Q10 \rangle$ Die Software muss einen reibungslosen Spielablauf und eine hohe Interoperabilität gewährleisten können.

- $\langle Q20 \rangle$ Das Produkt muss voll ausgereift sein (um eine hohe Zuverlässigkeit und eine geringe Ausfallrate zu garantieren).

- $\langle Q30 \rangle$ Die KI soll hinreichend schnell Entscheidungen treffen, um das Zeitfenster für eine Antwort nicht zu verlassen.

- $\langle Q40 \rangle$ Die Software muss eine ordnungsgemäße Ansteuerung der Hardware-Komponenten bereitstellen, um Stabilität zu gewährleisten.

- $\langle Q50 \rangle$ Das Produkt muss im Quellcode Änderungen an der KI ermöglichen.

- $\langle Q60 \rangle$ Da die Hardware vom Auftraggeber gestellt wird, kann über deren Lebensdauer keine Aussage und Garantie gegeben werden.

- $\langle Q70 \rangle$ Es wird keine Garantie bezüglich der Stabilität oder Sicherheit von Genode OS gegeben; der Client nebst Netzwerkprotokoll und LED-Matrix-Treiber werden lediglich in das System eingebunden.

7 Benutzeroberfläche/Schnittstellen

In diesem Kapitel werden die Schnittstellen sowie die Benutzeroberfläche beschrieben. Im Rahmen des Projekts werden dies betreffend nur die Netzwerkschnitstellen zur Kommunikation mit dem Gameserver und die LED-Matrix als Optische Darstellung verwendet.

Netzwerkschnittstellen mit Gameserver:

Der Client ist bezüglich eingehender Request- und Statusnachrichten und ausgehender Responsenachrichten für die Spielablaufskommunikation mit einem Gameserver über ein UDP-Netzwerkschnittstellenprotokoll verbunden.

Benutzeroberfläche:

Die LED-Matrix stellt - im weiteren Sinne - für eine den Client beobachtende Person eine Art Benutzeroberfläche ohne Eingabemöglichkeit dar. Die LED-Matrix erlaubt die grafische Darstellung des Spielfelds. Blaue Felder sind hierbei Wasserfelder, die nicht von Schiffen belegt sind oder auf die noch nicht geschossen wurde. Rote Felder sind Felder, die von einem Schiff belegt sind und bereits getroffen wurden. Grüne Felder stellen mit Schiffen belegte Felder dar, die noch nicht vom Gegner getroffen worden sind. Nichtleuchtende LEDs werden in den folgenden Beispielen schwarz dargestellt. Die Darstellung des Spielfelds kann man sich, wie in Abbildung 7.8 gezeigt, vorstellen.

Wenn *Admiral Client* nicht läuft oder es anderweitige Probleme mit der Ansteuerung der LED-Matrix gibt, leuchten keine LEDs, *siehe* Abbildung 7.1. Damit lässt sich schnell ein generelles Problem erkennen.

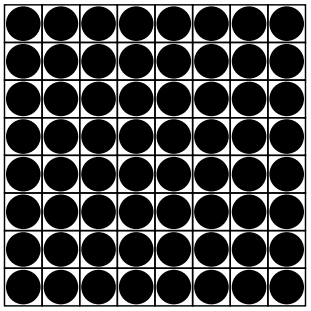

Abbildung 7.1: LED Matrix: Aus

Die nachfolgenden Abbildungen sind Beispiele, welche einen Überblick über diverse, auftretende Aktionen geben und zeigen deren grafische Darstellung auf dem Spielfeld. Um diese Beispielausgaben lebendiger zu gestalten, sind diese in den Kontext eines skizzierten Spielszenarios gesetzt.

Wenn das System mit *Admiral Client* vollständig gestartet ist und bis *Admiral Client* auf eine Anfrage zum Platzieren des ersten Schiffs wartet, leuchten alle LEDs blau. Die Abbildung 7.2 zeigt an, dass *Admiral Client* das Spielfeld aktiviert hat, aber noch keine Schiffe platziert hat.

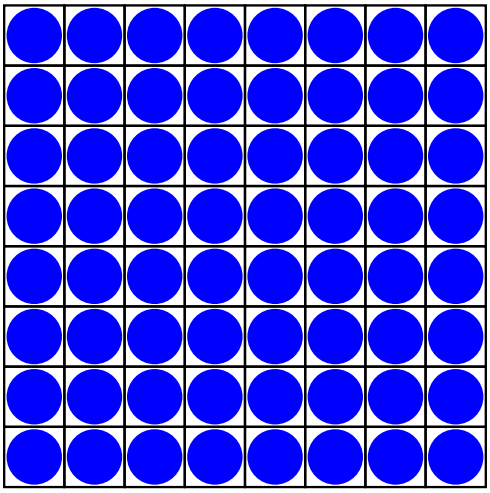

Abbildung 7.2: LED Matrix: Spielstart ohne platzierte Schiffe

In der Initialisierungsphase werden die Schiffe nacheinander platziert. Abbildung 7.3 zeigt ein Beispiel für die abgeschlossene Platzierung aller Schiffe der eigenen Flotte.

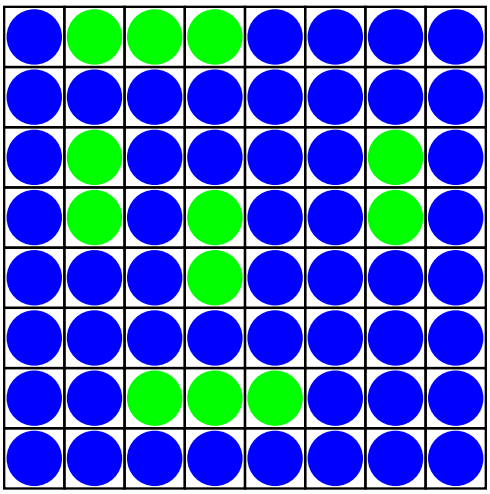

Abbildung 7.3: LED Matrix: Abgeschlossene Initialisierungsphase mit allen platzierten Schiffen der eigenen Flotte

Die erste Runde geht an das gegnerische Team, welches mit der Aktion Schuss sofort einen Treffer auf das Schiff der Länge zwei in der Spielfeldmitte landet, *siehe* Abbildung 7.4.

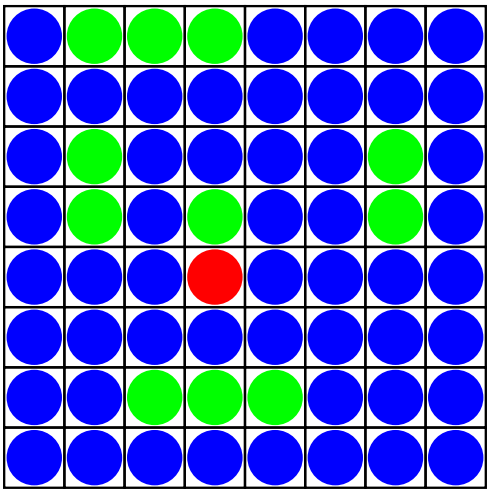

Abbildung 7.4: LED Matrix: Schuss des gegnerischen Teams in Runde 1 mit Treffer

Da das gegnerische Team in der vorherigen Runde getroffen hat, ist dieses nochmals an der Reihe. Unglücklicherweise trifft das gegnerische Team erneut und versenkt dabei das Schiff der Länge zwei in der Spielfeldmitte, *siehe* Abbildung 7.5.

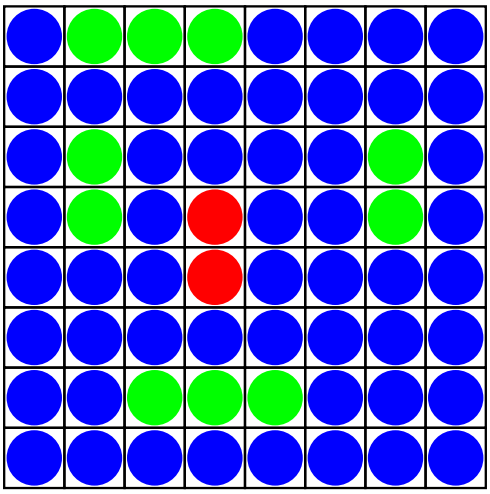

Abbildung 7.5: LED Matrix: Schuss des gegnerischen Teams in Runde 2 mit Treffer

Auch diesmal trifft das gegnerische Team. Diesmal wurde das rechte Schiff der Länge zwei getroffen, wie in der Abbildung 7.6 zu sehen ist.

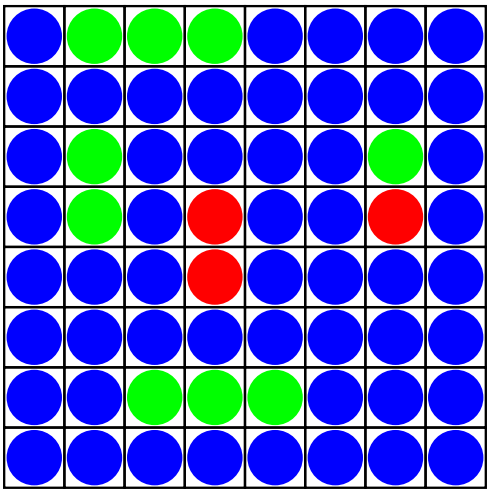

Abbildung 7.6: LED Matrix: dritter Schuss des gegnerischen Teams in Runde 3 mit Treffer

Das gegnerische Team ist erneut an der Reihe, aber diesmal erfolgt ein Fehlschuss. In Abbildung 7.7 ist dies mit dem Erlöschen der LED an der Fehlschussposition zu sehen.

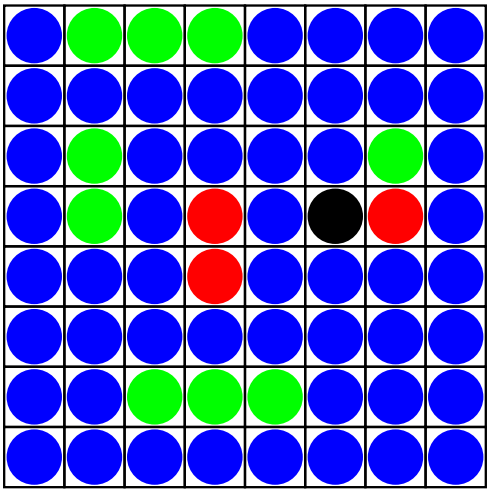

Abbildung 7.7: LED Matrix: vierter Schuss des gegnerischen Teams in Runde 4 mit Fehlschuss

Da das gegnerische Team einen Fehlschuss hatte, ist das eigene Team wieder an der Reihe. Die LED-Matrix wird wie in Abbildung 7.8 angezeigt.

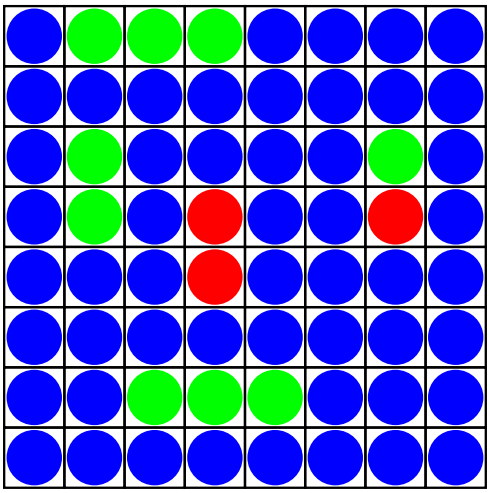

Abbildung 7.8: LED Matrix: Start der fünften Runde

Das untere Schiff der Länge drei bewegt sich ein Feld nach rechts in der fünften Runde, *siehe* Abbildung 7.9.

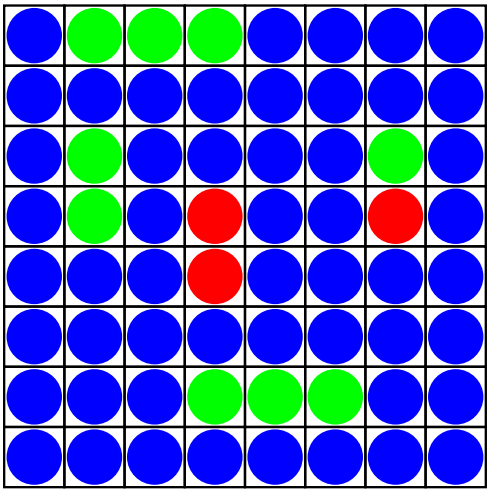

Abbildung 7.9: LED Matrix: unteres Schiff der Länge 3 bewegt sich ein Feld nach rechts in der fünften Runde

Der Schuss des gegnerischen Teams in Runde sechs geht erneut auf den unteren Endpunkt des rechten Schiffs der Länge zwei. Da dort schon ein getroffener Schiffsbereich vorliegt, ist der vorliegende Schuss ein Duplikat und gilt als Fehlschuss. In Abbildung 7.10 ist dies mit dem Erlöschen der LED an der Fehlschussposition zu sehen.

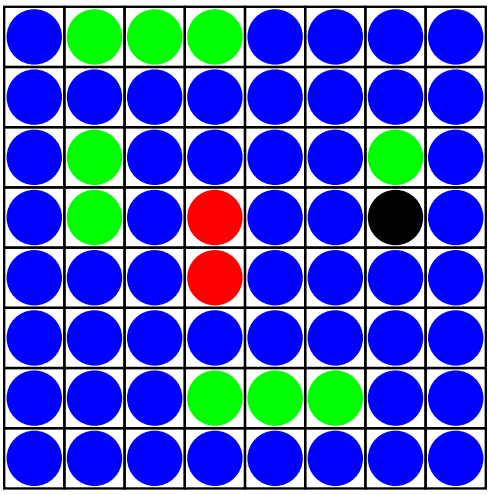

Abbildung 7.10: LED Matrix: Schuss des gegnerischen Teams in Runde 6 geht erneut auf den unteren Endpunkt des rechten Schiffs der Länge 2. Da dort schon ein getroffener Schiffsbereich vorliegt, ist der vorliegende Schuss ein Duplikat und gilt als Fehlschuss.

Das eigene Team ist in der siebten Runde wieder an der Reihe. Die LED-Matrix wird wie in Abbildung 7.11 angezeigt.

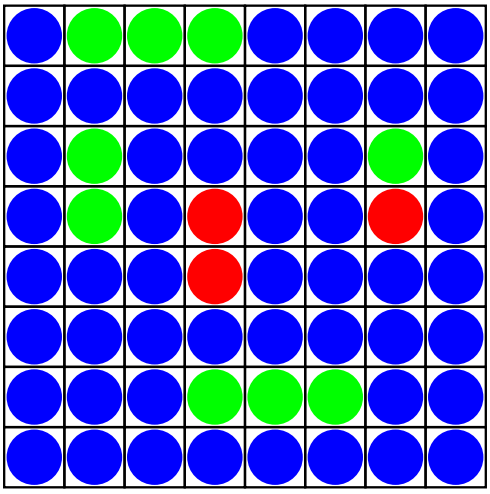

Abbildung 7.11: LED Matrix: Start der siebten Runde

Nun wird das untere Schiff der Länge drei um den Mittelpunkt gedreht. Abbildung 7.12 zeigt das Resultat dieser Bewegung.

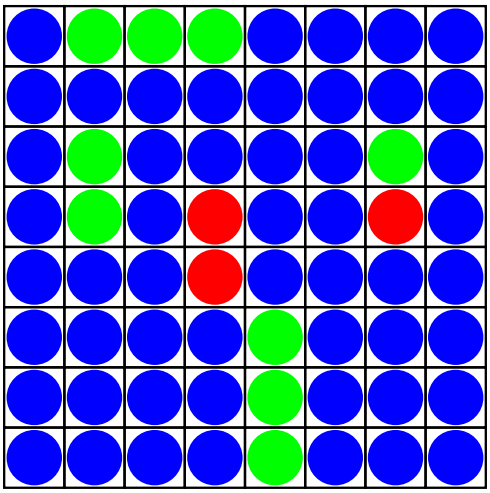

Abbildung 7.12: LED Matrix: unteres Schiff der Länge 3 dreht sich um den Mittelpunkt in der siebten Runde

Das gegnerische Team trifft diesmal das untere Schiff der Länge drei, wie in der Abbildung 7.13 zu sehen.

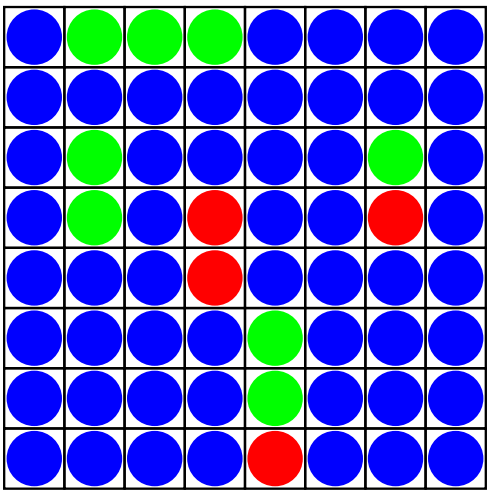

Abbildung 7.13: LED Matrix: Schuss des gegnerischen Teams in Runde 8 mit Treffer

Das gegnerische Team ist erneut an der Reihe, aber diesmal erfolgt ein Fehlschuss. In Abbildung 7.14 ist dies mit dem Erlöschen der LED an der Fehlschussposition zu sehen.

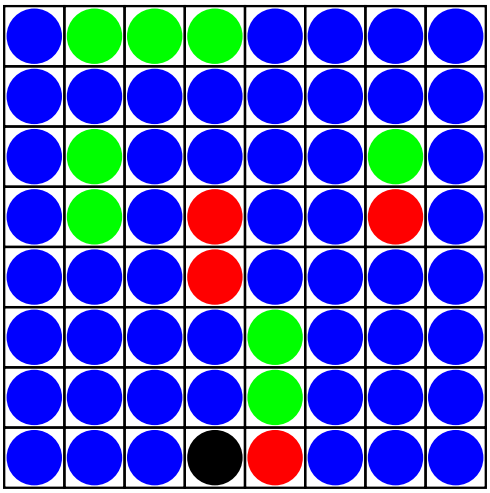

Abbildung 7.14: LED Matrix: Schuss des gegnerischen Teams in Runde 9 mit Fehlschuss

In der zehnten Runde gibt es einen frei verfügbaren Bonuszug vom Gameserver für das gegnerische Team. Da dieses nun ein Schiff bewegt, erfolgt keine Änderung bei der Ausgabe der eigenen LED-Matrix.

Das eigene Team ist in der elften Runde wieder an der Reihe. Die LED-Matrix wird wie in Abbildung 7.15 angezeigt.

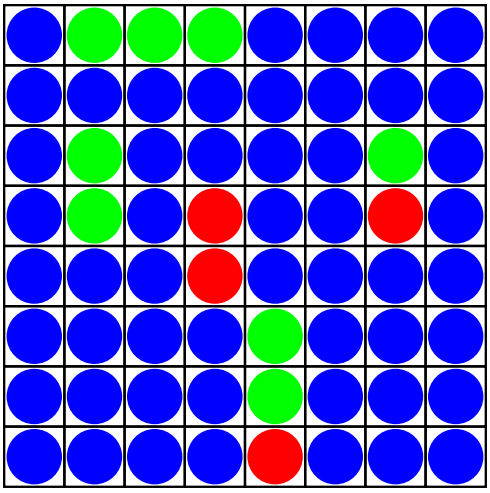

Abbildung 7.15: LED Matrix: Start der elften Runde

Jetzt wird das linke Schiff der Länge zwei um den oberen Endpunkt um 90° gegen den Uhrzeigersinn gedreht, *siehe* Abbildung 7.16.

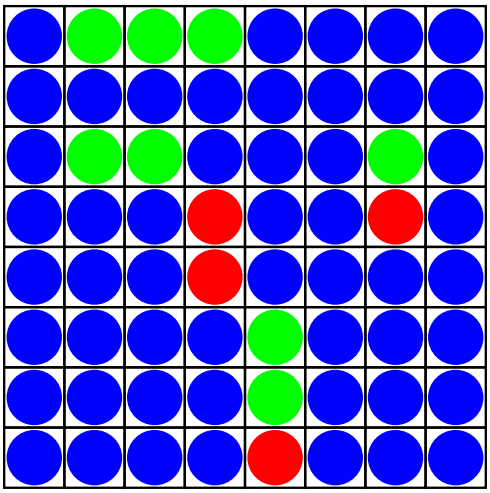

Abbildung 7.16: LED Matrix: linkes Schiff der Länge 2 dreht sich um den oberen Endpunkt um 90° gegen den Uhrzeigersinn in der elften Runde

Diese Manöver mit Schüssen und Bewegungen der beiden Teams gehen solange, bis dem ersten Team alle Schiffe versenkt wurden.

In der Runde X wurde das letzte Schiff der eigenen Flotte vom gegnerischen Team versenkt. Wie in der Abbildung 7.17 zu sehen, werden nun alle Schiffe als getroffen angezeigt.

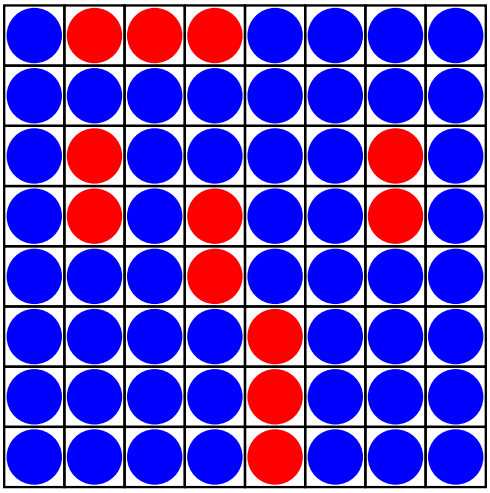

Abbildung 7.17: LED Matrix: in Runde X wurde das letzte Schiff der eigenen Flotte vom gegnerischen Team versenkt

Dies bedeutet, dass *Admiral Client* in der X-ten Runde verloren hat. Mit dem roten Aufleuchten aller LEDs siehe Abbildung 7.18, wird diese erhaltene Niederlage angezeigt.

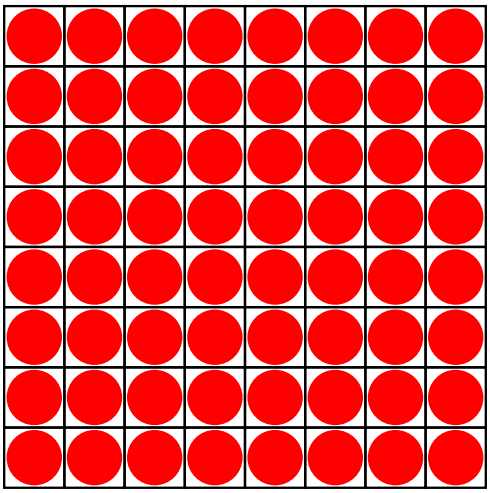

Abbildung 7.18: LED Matrix: Runde X ergibt eine Niederlage für *Admiral Client*

Sollte in Runde Y das eigene Team jedoch noch nicht versenkte Schiffe in der eigenen Flotte haben, aber die gegnerische Flotte komplett versenkt haben, hat *Admiral Client* in der Y-ten Runde einen Sieg errungen. Dies wird mit dem grünen Aufleuchten aller LEDs signalisiert und ist in Abbildung 7.19 dargestellt.

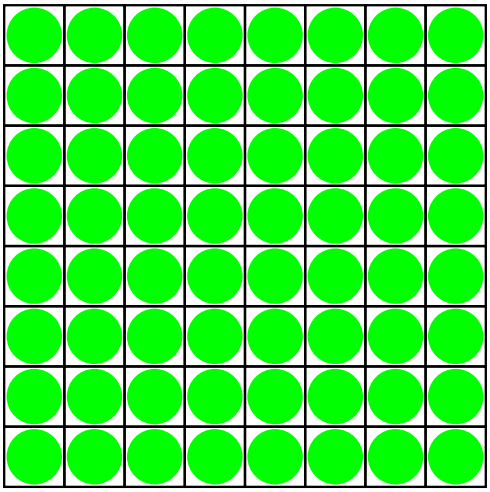

Abbildung 7.19: LED Matrix: Runde Y ergibt einen Sieg für *Admiral Client*

Nach einer Niederlage oder einem Sieg bereitet *Admiral Client* eine neu Partie vor und signalisiert die Bereitschaft mit dem blauen Aufleuchten aller LEDs, wie bereits vorher in Abbildung 7.2 beschrieben wurde.

8 Technische Produktumgebung

In diesem Kapitel wird die Umgebung beschrieben, in welche das Produkt letztendlich integriert werden soll. Es wird sowohl auf die Software, als auch auf die Hardware eingegangen und anschließend werden technische Daten, sowie die Produktschnittstellen dargestellt. Wichtig ist bei diesem Projekt, dass die technische Umgebung, in welcher das Produkt integriert werden soll, vom Auftraggeber detailliert vorgegeben ist und daher teilweise genaue Geräte beschrieben werden. Bei diesen ist die Verwendung und Optimierung auf die speziell dafür vorgesehenen Geräte notwendig.

8.1 Software

Als Betriebssystem für die Software des Clients wird ein speziell an das Parallella Board angepasstes Microkernel-Betriebssystem mittels des Genode OS Frameworks generiert. Innerhalb dieses Frameworks wird der *Admiral Client*komplett integriert, also werden die KI mit dem UDP-Netzwerkprotokoll zur Kommunikation mti dem Gameserver und der Treiber für die Ansteuerung der 8×8 LED-Matrix als Module implementiert. Die Programmiersprache zur Erstellung des Softwareproduktes ist dabei C++.

Der Gameserver wird dabei vom Auftraggeber (IDA) gestellt und wird auf einem Linux-Betriebssystem ausgeführt. Das Gerät, auf dem das Linux-Betriebssystem letztendlich läuft, ist variabel, muss aber über die nötigen Netzwerkanschlüsse verfügen, um mit zwei Clients kommunizieren zu können. Der Gameserver wurde mit der Programmiersprache *Python* erstellt, also muss das Gerät für den Gameserver sowohl Python (Version 2.7 oder neuer) als auch Pygame für Python2 installiert haben.

8.2 Hardware

Der Client läuft unter Genode OS auf einem Parallella Board mit angeschlossener LED-Matrix zur Spielfeldvisualisierung und ist per Netzwerk mit dem Gameserver verbunden.

8.2.1 Parallella Board

Die Produktspezifikationen[5] des Parallella Boards sind:

- 18-core kreditkartengroßer Computer (Dual-Core-Prozessor: Xilinx Zynq Dual-core ARM A9 7Z010 und 16-core-Co-Prozessor Epiphany RISC SOC) mit 667 MHz Prozessortakt
- 1 GB DDR3 SDRAM Arbeitsspeicher
- 128 MB, 3V, 108MHz Flashspeicher
- 1x USB 2.0
- 1x Gigabit Ethernet
- 1x Mikro-SD Card Slot
- 1x HDMI
- 48 GPIO Signal
- 5W Leistung
- $54mm \times 87mm$ Formfaktor

Um mit dem Client im interaktiven Modus zu spielen, muss am Gameserver eine Maus, eine Tastatur und ein Monitor angeschlossen sein. Ansonsten reicht ein weiterer, am Gameserver angeschlossener Spielclient. Dieser Client wird von einer weiteren und von uns unabhängigen SEP-Gruppe parallel realisiert.

8.3 Produktschnittstellen

Der Client ist über eine Netzwerkschnittstelle für die Kommunikation mit dem Gameserver verbunden. Hierfür kommt ein UDP-Netzwerkprotokoll zum Einsatz. Die Kommunikation läuft laut Gameserverspezifikation des Auftraggebers[6] über den UDP-Port 31337. Die IP-Adressen sind dabei statisch vergeben, wobei die IP-Adresse des Servers 192.168.12.34 und die Adressen der zwei Spielparteien 192.168.12.35 beziehungsweise 192.168.12.36 lauten.

[5] https://www.parallella.org/board
[6] Die Gameserverspezifikation wurde dabei vom Auftraggeber, dem IDA, zur Verfügung gestellt.

9 Glossar

Client

Ein Client ist ein Softwareprogramm, welches per Netzwerk mit einem Server verbunden ist und von diesem Informationen und Anweisungen entgegen nimmt und verarbeitet oder selbst Anfragen stellt. 6, 11–13, 18–25, 27–30

Eingebettetes System

Ein eingebettetes System ist ein Rechner, der in einem größeren Kontext eingebunden ist. Dem Rechner sind hierbei spezielle Aufgaben zugeordnet, die zur Funktion des Gesamtsystems beitragen. 6, 8, 11, 28

flüchtige Datenspeicherung

Flüchtige Datenspeicherung bezeichnet die Speicherung von Daten auf flüchtigen Datenspeichern, die Daten nur solange speichern können, wie sie mit Strom versorgt sind. Das Gegenteil der flüchtigen Datenspeicherung ist die persistente Datenspeicherung (*siehe* Persistenz). 25, 45

Framework

Ein Framework ist ein vorgefertigtes Programmiergerüst. 8, 11

Genode OS

Genode OS ist das Microkernel-Betriebssystem, auf dem später die KI laufen wird. *Siehe* Microkernel-Betriebssystem. 8, 11

Image

Ein Image ist ein identisches Speicherabbild von einem Speichermedium. Das Image enthält nicht nur die zu kopierenden Daten sondern auch deren Dateisystemstruktur. 11

KI

siehe Künstliche Intelligenz. 6–9, 11, 13, 14, 19–23, 25, 27, 44, 45

Künstliche Intelligenz (KI)

Die Künstliche Intelligenz (KI) ist ein Programm, welches nach bestimmten Kriterien Entscheidungen trifft. Diese imitiert das Verhalten von Menschen, im vorliegenden Fall das Spielen von Battleships. 44

Microkernel

Betriebssystemskerne, die sich von anderen Kernel-Typen in der Art unterscheiden, dass sie nur Funktionen zur Prozess- und Speicherverwaltung zur Verfügung stellen. 45

Microkernel-Betriebssystem

Ein Microkernel-Betriebssystem ist ein Betriebssystem, welches auf einem Microkernel aufbaut. *Siehe* Microkernel. 6, 8, 42, 44

Parallella Board

Das Parallella Board ist ein vollständig funktionierender Computer in Kreditkartengröße, auf welchem die KI später laufen wird. Das Board entspringt dem gleichnamigen Kickstarter-Projekt der Firma Adapteva. 6, 8, 10, 11, 18, 29, 30, 43

Persistenz

Persistenz, bzw. persistenze Datenspeicherung bezeichnet die Speicherung von Daten über einen längeren Zeitraum. Zum Beispiel die Speicherung von Daten auf einer Festplatte, welche auch nach dem Abschalten eines Computers noch vorhanden sind. Das Gegenteil ist die flüchtige Datenspeicherung (*siehe* flüchtige Datenspeicherung). 25, 44

RAM

RAM bedeutet *Random-Access Memory* und bezeichnet einen Speicher, in welchem die einzelnen Speicherzellen über die jeweiligen Speicheradressen direkt angesprochen werden können. Dieses Prinzip ermöglicht einen direkten Speicherzugriff, sowohl lesend als auch schreibend. Alle RAM-Speicher funktionieren als flüchtige Speicher (*siehe* flüchtige Datenspeicherung), das heißt, dass die gespeicherten Daten nach dem Wegfall der Stromversorung des RAM nicht gespeichert werden und verloren sind. 25

Request-Response-Prinzip

Das Request-Response-Prinzip entspricht der wechselseitigen Kommunikation in Client-Server-Modellen, sprich dem vorliegenden Kommunikationsmodell der Funktionsweise des Spielclients in Abhängigkeit zum Gameserver im vorliegenden Projekt. Requests sind hier Anforderungen des Gameservers an den Client und Responses entsprechen den Antworten des Clients an den Gameserver. 7

Schiffe versenken

Das Gesellschaftsspiel *Schiffe versenken* ist ein Spiel für zwei Personen bzw. für zwei Parteien, die je ein Spielfeld vor sich haben, welches nur sie selbst einsehen können. Das Spiel ist sehr alt und wurde traditionell mit Stift und Papier gespielt. Mittlerweile gibt es aber auch bereits digitale Versionen. Auf dem Spielfeld, welches ein Koordinatensystem besitzt, platziert jede Partei eine gleiche Anzahl Schiffe. Die gegnerische Partei beschießt das eigene Spielfeld durch Nennung einer Koordinate und versucht dabei, Schiffe zu treffen. Das Ziel ist es, alle Schiffe der Gegenpartei zu versenken und dabei in der eigenen Flotte noch mindestens ein nicht versenktes Schiffe übrig zu haben. 5, 11, 45

SEP

Softwareentwicklungspraktikum. 5, 11

Server

Ein Server ist ein Softwareprogramm, welches Dienste, Services oder Anweisungen für via Netzwerk ansgeschlossene Clients bereitstellt. Ein Beispiel wäre ein Gameserver wie im vorliegenden Projekt. 6–13, 18, 24, 27–30

User Datagram Protocol

Das User Datagram Protocol, kurz UDP, ist ein verbindungsloses, nicht zuverlässiges, ungesichertes und ungeschütztes Netzwerkprotokoll. Es versendet Datagramme in IP-basierten Rechnernetzen. 7