Lucerne University of
Applied Sciences and Arts

HOCHSCHULE
LUZERN

Informatik

Modul Programmieren & Datenstrukturen

Birds'N'Cars - Projektdokumentation

Felix Businger, Lucien Egloff, Adrian Kohler, Cyrill von Uslar

Inhaltsverzeichnis

Abbildungen

1. Einleitung

Geschätzte Studierende

in der zweiten Hälfte des Semesters bearbeiten Sie in einer kleinen Gruppe ein kleineres Softwareprojekt. Mit diesem Projekt sind die folgenden Ziele verbunden:

- Sie wenden im Unterricht gelernte Konzepte der Sprache Java in einem grösseren Kontext an.
- Sie wiederholen wesentliche Elemente der Programmiersprache Java.
- Sie implementieren eine Softwarelösung im Team.
- Sie können Kontextdiagramme lesen und interpretieren.
- Sie können Komponentendiagramme lesen und interpretieren.
- Sie können Sequenzdiagramme lesen und interpretieren.
- Sie können Zustandsdiagramme lesen und interpretieren.
- Sie können Sourcecode mit Hilfe von Klassendiagrammen dokumentieren.
- Sie können in einem grösseren Programm die Übersicht wahren.

Das gesamte Projekt ist als Lernprojekt zu verstehen, bei dem Sie Schritt für Schritt Kenntnisse erwerben und anwenden.

Die Dozierenden und Assistierende begleiten Sie während des Projekts und ermöglichen Ihnen, wesentliche Erfahrungen zu reflektieren. Zur Unterstützung dieses Prozesses erstellen Sie jede Woche für die Dozierenden einen kurzen Projekt-Statusrapport mit folgendem Inhalt:

- Welche Arbeiten wurden in der letzten Woche ausgeführt. Was hat gut geklappt, wo hatten oder haben Sie Probleme?
- Welche Tätigkeiten sind für die nächste Woche vorgesehen?
- Welche Knackpunkte (Herausforderungen oder Risiken) bestehen noch? Was gedenken Sie dagegen zu unternehmen?

Eine Vorlage für diesen Projekt-Statusrapport [2] finden Sie im ILIAS.

Viel Erfolg sowie spannende und wertvolle Projekterfahrungen wünschen Ihnen

Ihr Dozierendenteam

1.1. Begriffe, Abkürzungen

Die Anwendung wird in Englisch erstellt. Gewisse Begriffe werden aber in dieser SysSpec auf Deutsch benutzt. In der folgenden Tabelle sind die Übersetzungen.

Engl. Begriff	*Dt. Begriff*
Battleship	Schiffe versenken
Grid	Spielfeld
Engine	Spielsteuerung
Opponent	Gegenspieler

1.2. Referenzen

[1] Projektauftrag Schiffe versenken (Projektauftrag.pdf)

[2] Projekt-Statusrapport (PRG2_Projekt-Statusrapport.docx)

2. Anforderungen

Aus der Aufgabenstellung [1] lassen sich zwei User Storys ableiten:

User Story 1:

Als Spieler möchte ich über das Netzwerk gegen einen Gegenspieler Schiffe versenken spielen um Spass zu haben.

User Story 2:

Als Spieler möchte ich gegen einen Computergegner Schiffe versenken spielen um Spass zu haben.

2.1. Systemübersicht

Aus diesen User Stories lässt sich das System visualisieren.

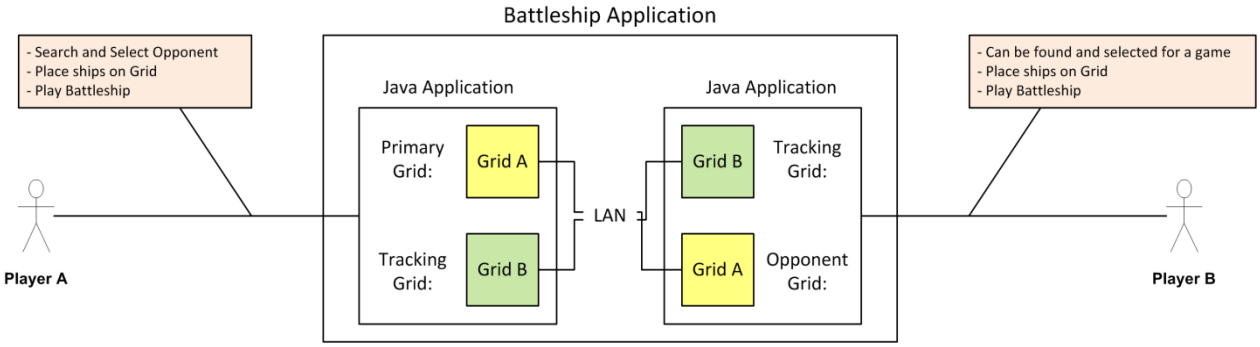

Abbildung 1: Kontextdiagramm der Battleship Application bei einem Spiel gegen einen Gegenspieler über das Netzwerk

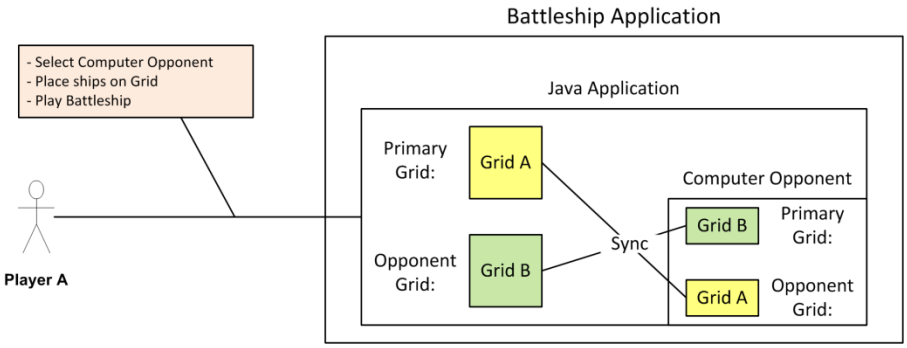

Abbildung 2: Kontextdiagramm der Battleship Application bei einem Spiel gegen einen Computergegner

Bei beiden Anwendungen wird deutlich, dass es vier Grids gibt, wobei jeweils zwei miteinander synchronisiert werden müssen. Diese Synchronisation ist die einzige Netzwerkaktivität während des Spiels.

2.2. Anwendungsfall

Hier ist der Anwendungsfall, welcher aus der User Story 1 abgeleitet wurde.

Name	Spiel gegen Netzwerkgegenspieler
Beschreibung	Netzwerkgegenspieler werden gesucht und danach wird gegen einen dieser Gegenspieler Battleship gespielt.
Beteiligte Akteure	User (Player A) Netzwerkgegenspieler (Player B)
Auslöser	Zwei Spieler wollen gegeneinander Battleship spielen.
Vorbedingungen	Beide Spieler haben die Applikation gestartet Die beiden Spieler befinden sich im selben Netzwerk
Ergebnis	Einer der beiden Spieler gewinnt das Spiel.
Standardablauf	1. Mögliche Gegenspieler werden im Netzwerk gesucht 2. Ein möglicher Gegenspieler wird selektiert, damit Battleship gespielt werden kann. 3. Wenn der Gegenspieler akzeptiert, wird ein neues Spiel erzeugt. 4. Beide Spieler setzen ihre Schiffe auf dem eigenen Spielfeld. Dabei gelten folgende Regeln: - Jedes Schiff besetzt eine bestimmte Anzahl von horizontal oder vertikal aufeinanderfolgenden Feldern. - Die Schiffe können sich nicht überlappen - Beide Spieler haben dieselben Schiffe 5. Sobald alle Schiffe gesetzt sind kann der Spieler sein eigenes Spielfeld als bereit markieren. Ab diesem Zeitpunkt können die Schiffe nicht mehr bewegt werden. 6. Sobald beide Spieler bereit sind beginnt das eigentliche Spiel. 7. Der Spieler der an der Reihe ist, attackiert ein Feld des Gegners. Bei einem Angriff gelten folgende Regeln: - Der Gegner signalisiert, ob sich auf diesem Feld ein Schiff befindet (Treffer) oder ob es Meer ist (Fehlschuss). - Bei einem Treffer darf der attackierende Spieler nochmals schiessen. - Bei einem Fehlschuss ist der andere Spieler dran. - Sobald alle Felder eines Schiffs getroffen worden sind, gilt dieses als versenkt. - Der attackierte Spieler teilt bei einem Treffer mit, ob sein Schiff versenkt ist. 8. Punkt 7 wird solange wiederholt, bis bei einem Spieler alle Schiffe versenkt wurden. Dann ist das Spiel vorbei und der Spieler, welcher noch Schiffe besitzt, hat gewonnen.
Exceptions (Spielabruch)	Beide Spieler selektieren sich gleichzeitig für ein Spiel. Die Verbindung zwischen zwei Spielern wird beendet. Einer der Spieler beendet die Applikation.

3. Systemspezifikation

Für die Anwendung wird das **Model-View-Controller Muster** verwendet:

- View: GUI
- Controller: Engine und Opponent
- Model: Grid

Die Komponente Opponent ist ein Stellvertreter (Proxy) für einen Gegenspieler. Alle Kommandos die an diese Komponente gesendet werden, werden weitergeleitet und via Netzwerk von einem Netzwerkgegenspieler oder von dem Computergegenspieler ausgeführt.

Die einzelnen Komponenten können auf der folgenden Abbildung betrachtet werden:

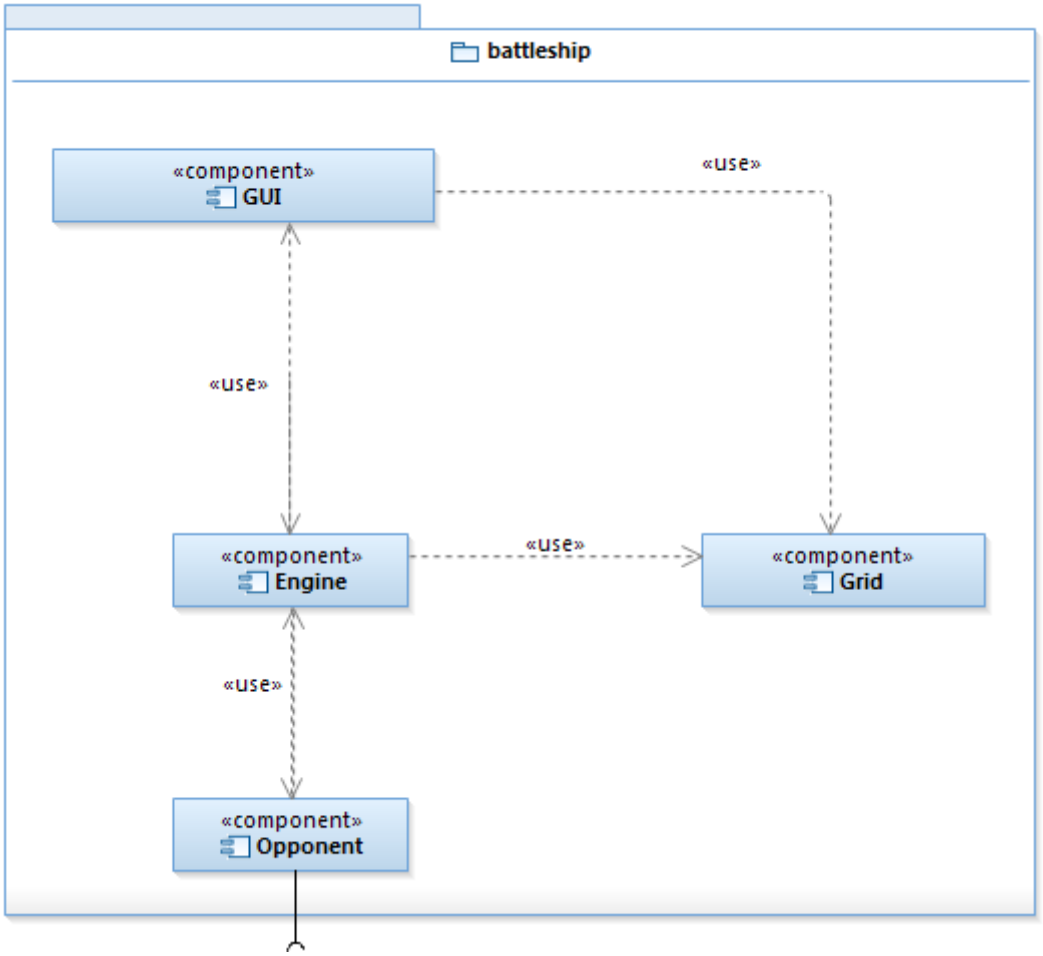

Abbildung 3: Die vier Komponenten der Battleship Application

3.1. Zustände

Währen der Laufzeit der Applikation treten vier Zustände ein:

- Gegenspieler auswählen (SelectingOpponent)
- Spielfeld vorbereiten (PreparingGrid)
- Spielen (Play)
- Spiel beendet (Finished)

Diese 4 Zustände sind im folgenden Diagramm dargestellt. Was in den einzelnen Zuständen passiert, ist im Folgenden beschrieben. Weiter existiert für jeden Zustand auch noch ein Sequenzdiagram.

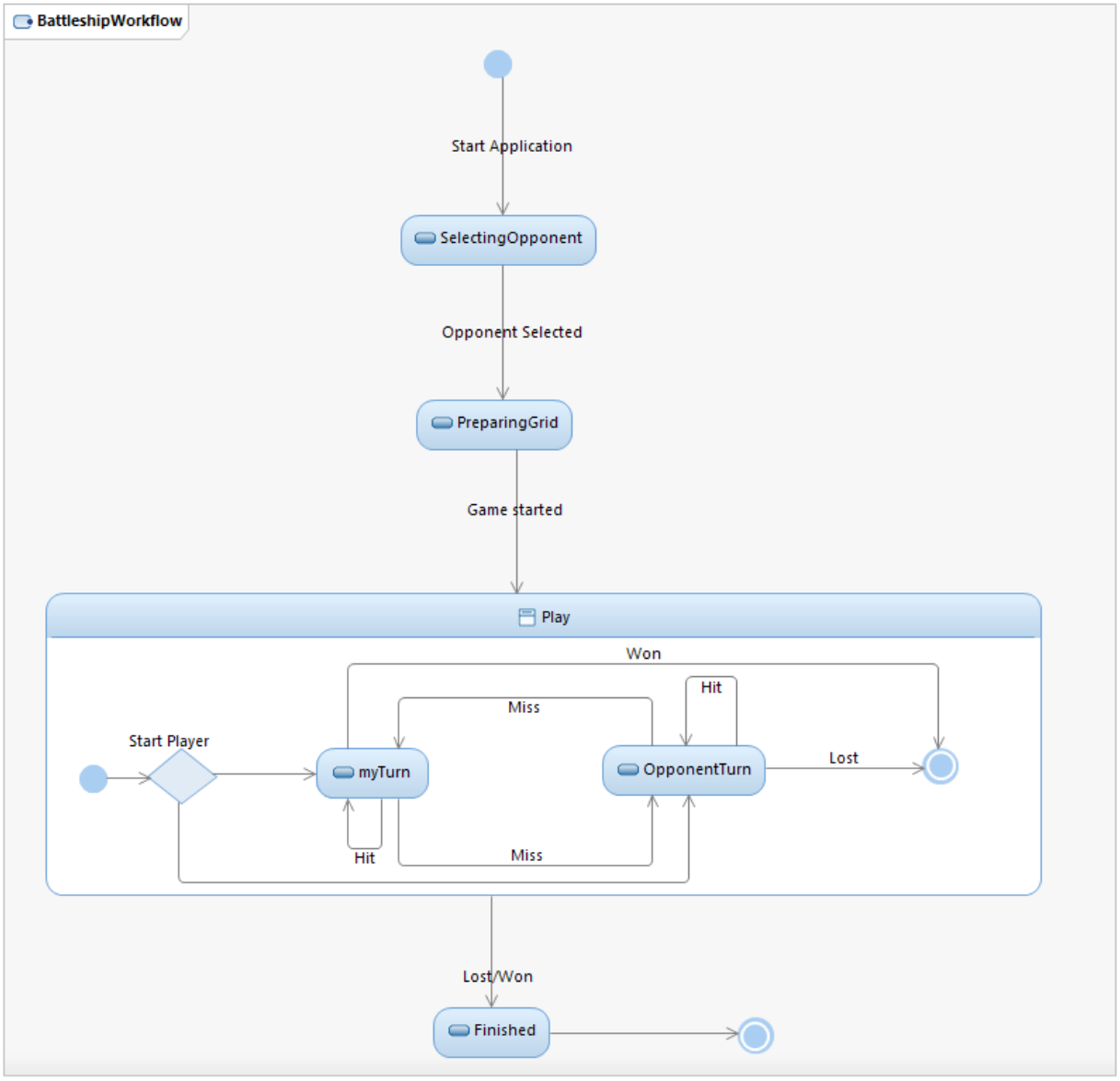

Abbildung 4: Statemaschine der Anwendung

3.1.1. SelectingOpponent

Im Multiplayermenü kann man eine IP-Adresse eingeben und „Connect“ klicken. Der User am PC mit der gegebenen IP bekommt, sofern er das Spiel auch gestartet hat, eine Spielanfrage und kann diese entweder annehmen oder ablehnen.

3.1.2. PreparingGrid

Wenn der Gegenspieler die Anfrage angenommen hat, wird das Grid geöffnet. Nun kann man im eigenen Grid einzelne Felder anklicken. Die Statusanzeige in der Mitte zeigt an, welche Figuren bereits gesetzt wurden. Sind alle Figuren gesetzt, kann man auf Ready klicken. Sobald beide Spieler bereit sind, beginnt das eigentliche Spiel.

3.1.3. Play

Das Spiel funktioniert wie ein normales Schiffeversenken. Klickt man auf das gegnerische Feld, wird ein Schuss abgegeben. Trifft man ins Leere, färbt sich das Feld blau. Trifft man eine gegnerische Figur, so wird das Feld rot.Erhält man ein Paket des Gegenspielers, wird das Spielfeld aktualisiert.

3.1.4. Sequenzdiagramme

Die Abläufe in den Zuständen werden in Sequenzdiagrammen dargestellt. Diese zeigen die Interaktionen zwischen den Komponenten.

SelectingOpponent

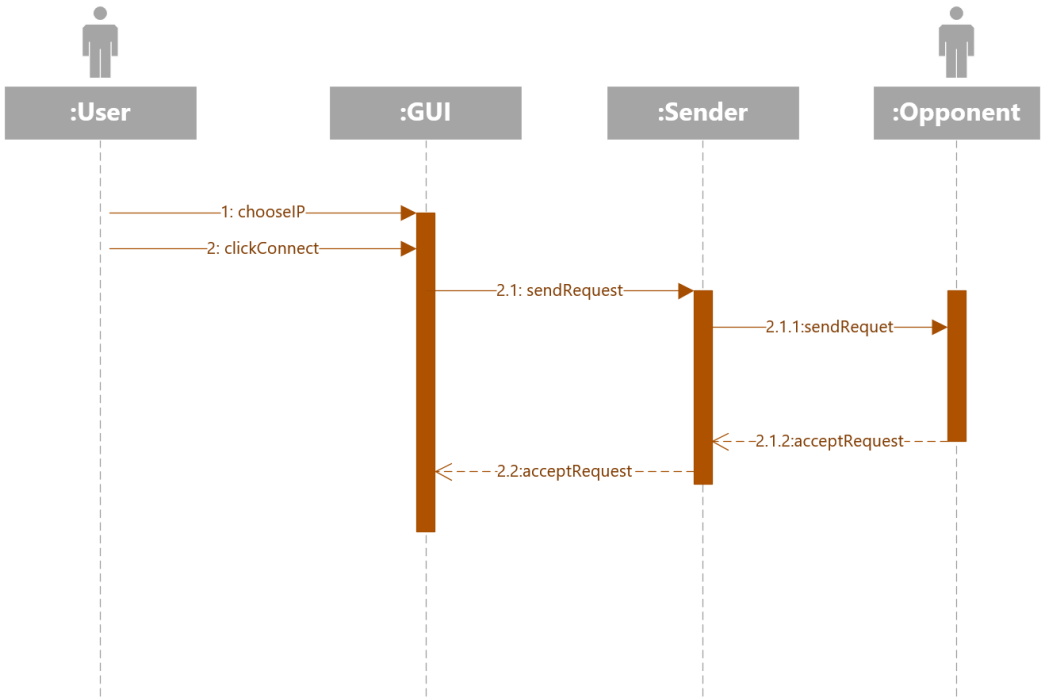

Abbildung 5: Sequenzdiagramm Spieleranfrage

PreparingGrid

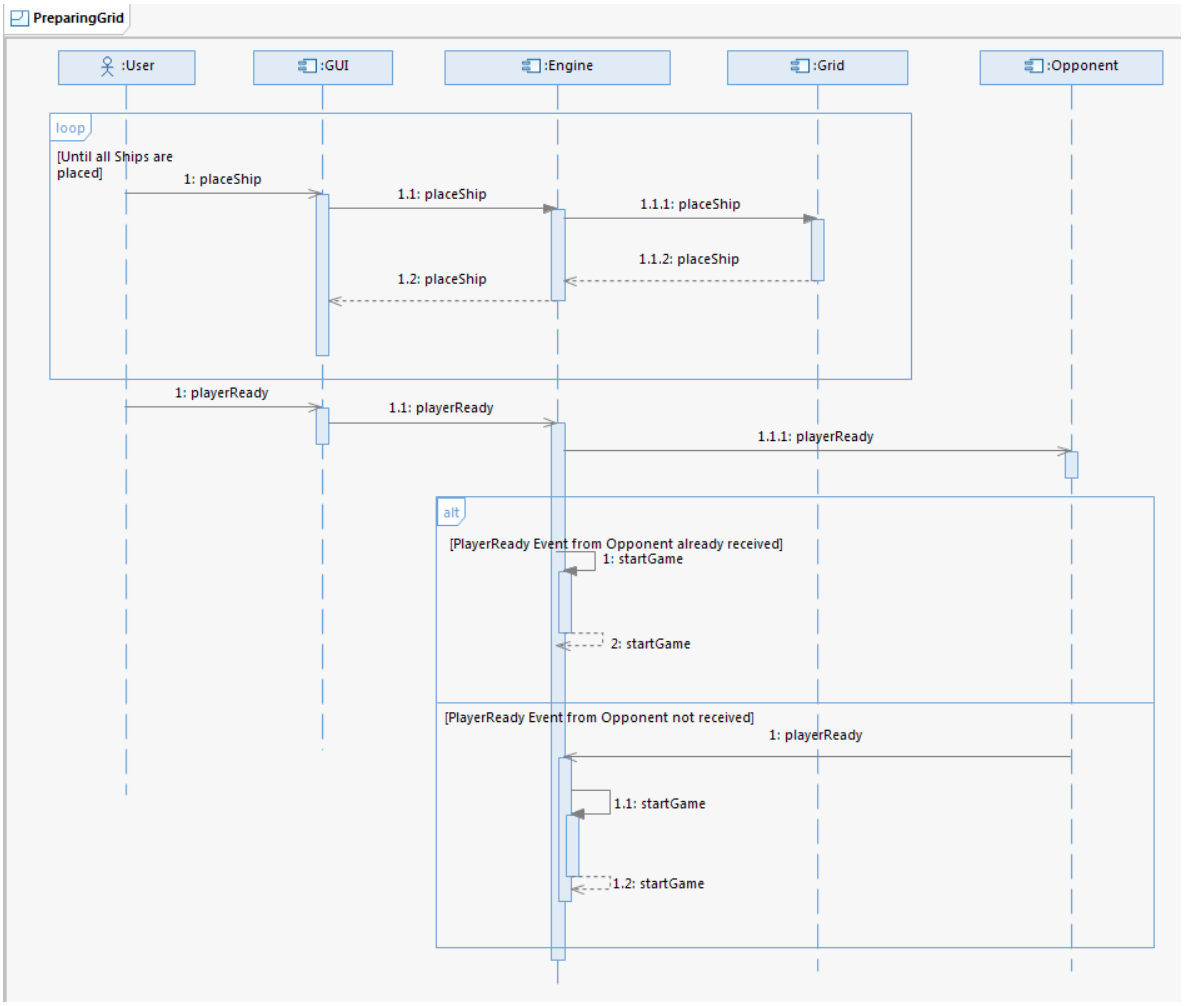

Abbildung 6: Sequenzdiagramm PreparingGrid

3.2. Funktionale Sicht

Das Programm besteht aus mehreren Modulen. Zu Grunde liegt eine Main-Klasse, welche alle anderen Klassen startet. In erster Linie ist das das GUI. Zudem wird das Modell initialisiert, also zwei „Parkplätze“, in denen der Zustand des gegnerischen und des eigenen Felds festgehalten wird.

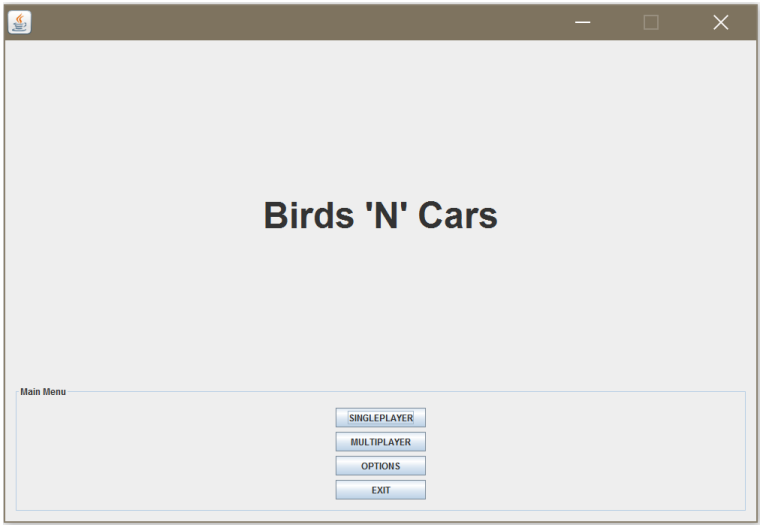

Abbildung 7: Hauptmenü

Das GUI hat ein CardLayout. Die Einzelnen Cards sind das Hauptmenü, das Multiplayermenü, sowie das Spielfeld. Dadurch ist die GUI-Klasse sehr umfangreich. Zudem sind auch viele Funktionen (v.a. ActionListener und Konsorten) dort implementiert.

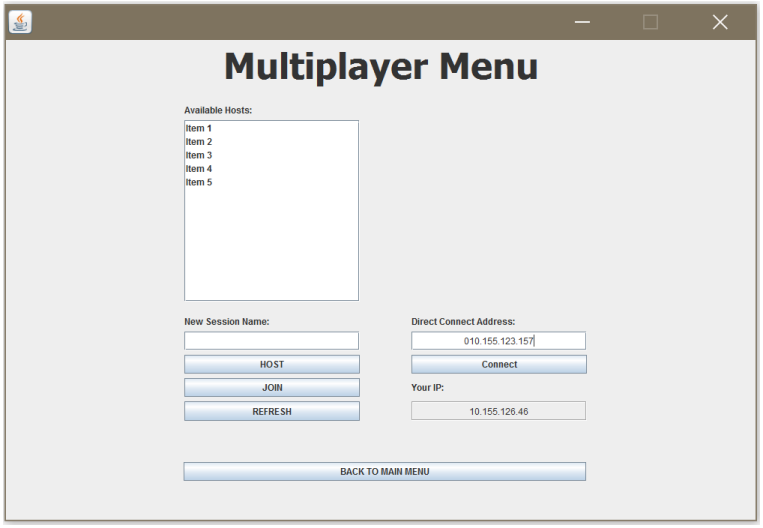

Abbildung 8: Multiplayer Menü

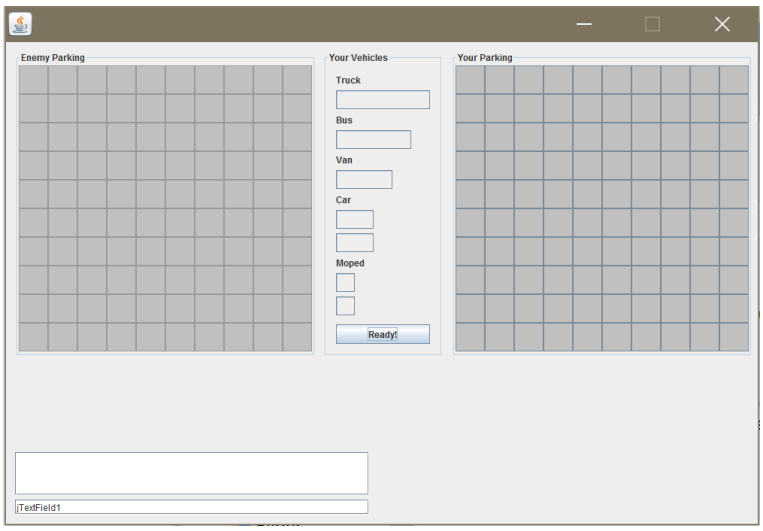

Abbildung 9: Leeres Spielfeld

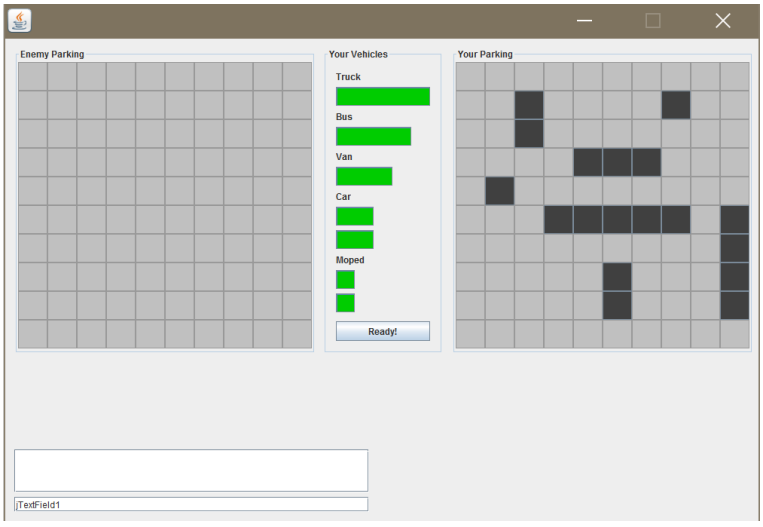

Abbildung 10: Schiffe gesetzt

Abbildung 11: Im Spiel

Für die Kommunikation gibt es weiter Klassen: einen Sender und einen Reciever. Diese senden und empfangen Daten.

Die Datenspeicherung (das Model im MVC-System) geschieht in der Klasse „ParkingLot“, welche ein zweidimensionales Array beinhaltet, sowie Methoden um die Inhalte dieses Arrays zu manipulieren.

Dazu gibt es weitere Helfer-Klassen, welche für die Funktion der Hauptklassen wichtig sind.

3.3. Verteilungs- und Betriebssicht

Die Applikation läuft sparsam und ressourcenschonend. Die Applikation wurde auf Windows 10 getestet. Für anderer Betriebssysteme wird keine Garantie übernommen.

3.4. Datensicht

Die Daten des Spiels werden in der ParkingLot-Klasse gespeichert. Für die Kommunikation werden mit dem Externalize-Interface die Daten serialisiert und gesendet bzw. eingelesen.

3.5. Kommunikation über Netzwerk

Die Netzwerkkommunikation geschieht via UDP-Verbindung mittels Datagram-Socket.

3.6. Erweiterungen

3.6.1. Nicht implementierte Funktionen

3.6.1.1. Singleplayer

Es wurde kein Computergegner implementiert. Die Schwierigkeit wird hier im sinnvollen Platzieren der Figuren ausgemacht. Ein zufälliges Schiessen auf das Feld des humanen Spielers könnte wahrscheinlich einfach implementiert werden.

3.6.1.2. Multiplayer

Alternativ zum direkten verbinden mittels IP wäre auch ein Sessionbrowser geplant – eine Liste mit Spielen, die noch einen Mitspieler brauchen. Man kann ein Spiel entweder hosten oder joinen. Die GUI-Komponenten dazu stehen bereits bereit.

3.6.1.3. Spielablauf

3.6.1.3.1. Spielzugsynchronisation

Es fehlt eine geeignete Spielzugsynchronisation. Momentan werden einfach Datenpakete hin und her geschickt, ohne einen geregelten Ablauf.

3.6.1.3.2. Spielabbruch

Noch zu implementieren sind Buttons zum Spielabbruch oder das Spiel zu speichern, sowie die korrekte Kommunikation dieser gegenüber dem Gegenspieler.

3.6.1.3.3. Siegbedingung

Eine Siegbedingung ist noch korrekt zu implementieren. Als Idee steht ein einfaches Auszählen der bereits getroffenen Felder.