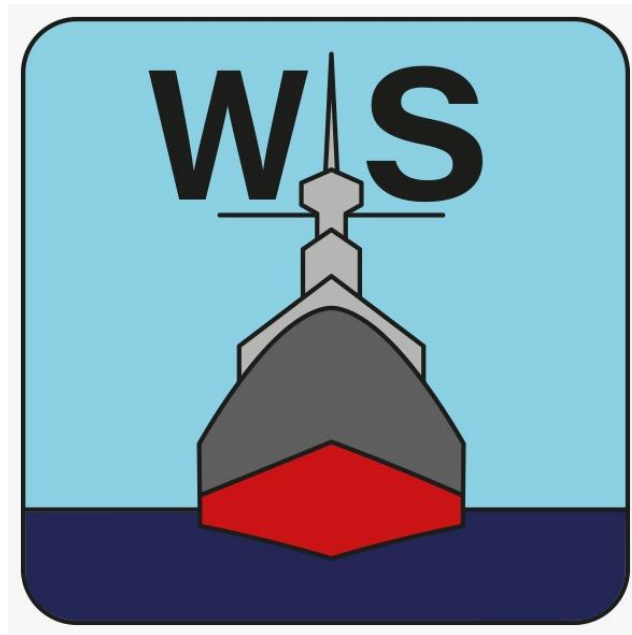


# Warships



Ein Spiel von Luca Thurm und Max Zimmermann für das Webtechnologie-Projekt

SoSe 2018

## Inhalt

1 Anforderungen .....	3
2 Spielkonzept.....	4
2.1 Komponenten des Spiels .....	4
2.2 Spielablauf .....	5
2.3 Regeln zum Bewegen der Schiffe.....	5
3 Architektur/Implementierung .....	5
3.1 Model.....	7
3.1.1 GameModel.....	7
3.1.2 PlayingField.....	8
3.1.3 Field.....	8
3.1.4 Enemy .....	9
3.1.5 Entity .....	9
3.1.6 Ship .....	9
3.1.7 Rock .....	10
3.1.8 PowerUp.....	10
3.2 View.....	10
3.2.1 HTML-Dokument.....	11
3.2.2 GameView-Klasse.....	11
3.3 Controller.....	13
3.3.1 Eventhandling .....	13
3.3.2 Laden der Level .....	13
4 Levelkonzept .....	14
4. 1 Aufbau der Level .....	14
4.2 Parametrisierung der Level.....	14
5 Nachweis der Anforderungen .....	15
6 Verantwortlichkeiten.....	16
7 Nutzung des Spiels .....	16

# 1 Anforderungen

Das Spiel sollte folgende Anforderungen, wie unter <https://lernraum.fh-luebeck.de/mod/page/view.php?id=86165> angegeben erfüllen.

ID	Kurztitel	Anforderung
AF-1	Single-Player-Game als Single-Page-App	<ul style="list-style-type: none"><li>• Das Spiel ist als Ein-Spieler-Game zu konzipieren.</li><li>• Das Spiel ist als HTML-Single Page App zu konzipieren.</li><li>• Das Spiel muss als <b>statische Webseite</b> von beliebigen Webservern (HTTP) oder Content Delivery Networks (CDN) bereitgestellt werden können (bspw. mittels GitHub Pages).</li><li>• Alle Spielressourcen (z.B. Level-Dateien, Bilder, CSS-Dateien, etc.) sind daher relativ und nicht absolut zueinander zu adressieren.</li></ul>
AF-2	Balance zwischen technischer Komplexität und Spielkonzept	<ul style="list-style-type: none"><li>• Sie sollen ein interessantes Spielkonzept entwickeln oder ein bestehendes Spielkonzept abwandeln. Spielkonzepte sind nicht immer technisch komplex (z.B. Memory).</li><li>• Sie sollen jedoch ein Spiel konzipieren, dass eine vergleichbare Komplexität mit den Spielen der Hall-of-Fame hat (Memory wäre bspw. zu einfach; Schach zu kompliziert, da sie für ein Ein-Spieler-Game eine Gegner-KI entwickeln müssten).</li><li>• Unabhängig von der inneren Komplexität soll das Spiel schnell und intuitiv erfassbar sein und angenehm auf einem SmartPhone zu spielen sein.</li></ul>
AF-3	DOM-Tree-basiert	<ul style="list-style-type: none"><li>• Das Spiel soll dem MVC-Prinzip folgen (Model, View, Controller).</li><li>• Das Spiel soll den DOM-Tree als View nutzen.</li><li>• Es sind keine Canvas-basierten Spiele erlaubt. <i>Hintergrund: Sie sollen Webtechnologien lernen und nicht wie man eine Grafikbibliothek programmiert.</i></li></ul>
AF-4	Target-Device: SmartPhone	<ul style="list-style-type: none"><li>• Das Spiel soll für eine SmartPhone Bedienung konzipiert werden.</li><li>• Entsprechende Limitierungen sind zu berücksichtigen.</li><li>• Als Target Devices sind die Plattformen Android und iOS zu berücksichtigen.</li><li>• Das Spiel soll mit HTML5 mobile Browsern auf den genannten Plattformen spielbar sein.</li></ul>
AF-5	Mobile First Prinzip	<ul style="list-style-type: none"><li>• Das Spiel soll bewusst für SmartPhones konzipiert werden.</li><li>• Das Spiel soll auch auf Tablets und Desktop PCs spielbar sein. Einschränkungen sind zu minimieren, werden aber billigend in Kauf genommen (z.B. fehlende 3D-Lage bei Desktop Browsern).</li><li>• Sie sollen typische mobile Interaktionen sinnvoll nutzen (z.B. Swipe, Wischen, 3D Lage im Raum, etc.).</li><li>• Übertragen sie nicht eine typische Desktop-Bedienung auf Mobile.</li></ul>

		<ul style="list-style-type: none"> <li>Sie müssen allerdings keine mobile Interaktionen sklavisch nutzen - wenn dies nicht sinnvoll für das Spielkonzept ist.</li> </ul>
AF-6	Das Spiel muss schnell und intuitiv erfassbar sein und Spielfreude erzeugen.	<ul style="list-style-type: none"> <li>Das Spiel muss schnell und intuitiv erfassbar sein.</li> <li>Das Spiel muss Spielfreude erzeugen.</li> <li>Hintergrund: <i>Ihre Spiele sollen ggf. als Anschauungsbeispiele für Studieninteressierte Schüler auf Messen oder ähnlichen Veranstaltungen gezeigt werden. Sie arbeiten also nicht für "die Tonne" - andere sollen ihre Spiele tatsächlich spielen.</i></li> <li><i>Tipp: Vermeintlich einfache Spielprinzipien haben ihre Stärken und sind dennoch komplex genug um die technischen Anforderungen dieses Projekts abzudecken.</i></li> </ul>
AF-7	Das Spiel muss ein Levelkonzept vorsehen	<ul style="list-style-type: none"> <li>Es ist ein steigender Schwierigkeitsgrad über mindestens 7 Level vorzusehen.</li> <li>Level sollen deklarativ in Form von Textdateien beschrieben werden können (z.B. JSON, XML, CSV, etc.).</li> <li>Diese Level sollen durch das Spiel nachgeladen werden können, so dass nachträglich Level ergänzt und abgeändert werden können, ohne die Programmierung des Spiels anpassen zu müssen.</li> </ul>
AF-8	Ggf. erforderliche Speicherkonzepte sind Client-seitig zu realisieren	<ul style="list-style-type: none"> <li>Aufgrund der Zielgruppe sollen durch das Spiel gesammelte Daten auf den Geräten bleiben.</li> <li>Aufgrund des Demonstrationscharakters auf Messen dürfen keine zentrale Server für Highscores, etc. erforderlich sein oder angebunden werden.</li> <li>Ggf. erforderliche Stateful solutions sind mittels client-seitiger Storage-Konzepte zu lösen. D.h. z.B. mittels <u>local storage</u>.</li> </ul>
AF-9	Dokumentation	<ul style="list-style-type: none"> <li>Das Spiel muss nachvollziehbar dokumentiert sein.</li> <li>Das Spiel muss analog dem SnakeGame dokumentiert sein. <i>Hinweis: Nutzen sie die SnakeGame-Dokumentation als Template.</i></li> </ul>

Tabelle 1: Anforderungen

## 2 Spielkonzept

### 2.1 Komponenten des Spiels

Das Spiel Warships basiert auf dem Spielkonzept von klassischem „Schiffe Versenken“. Es wird auf einem Spielfeld der Größe 9 x 16 gespielt, welches in der Mitte horizontal in das Feld des Spielers und das des Gegners aufgeteilt wird. Das Gegnerische Spielfeld wird vom Nebel verdeckt, so dass der Spieler keine Informationen über die Position der gegnerischen Schiffe oder anderer Entitäten hat. Auf jeder Kachel des Spielfelds kann sich höchstens eine Entität befinden. Die Entitäten des Spiels sind:

- Schiffe

- Inseln/Felsen
- Power-Ups


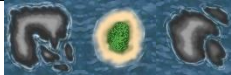

Bild	Name	Beschreibung
	Schiff	Schiffe gehören einer der spielenden Parteien. Sind alle Schiffe einer Partei zerstört, so verliert diese. Es existieren Schiffe in den Größen zwei, drei, vier und Fünf. Schiffe können sich, sofern sie nicht beschädigt sind, entsprechend ihrer Ausrichtung bewegen, sofern sich keine Hindernisse wie Inseln oder andere Schiffe im Weg befinden. Sind alle Kacheln eines Schiffes getroffen, wird dieses zerstört und geht unter.
	Inseln / Felsen	Inseln/Felsen dienen zur Verwirrung der Spielenden. Werden sie getroffen, sieht es für den Spieler aus, als hätte er ein Schiff des Gegners entdeckt. Des Weiteren können sie den Weg für Schiffe blockieren.
	Power-Ups	Power-Ups sind temporäre Verstärkungen, die sich nur auf der Seite des Gegners befinden. Werden sie getroffen, aktiviert sich ihr Effekt. Es existieren zwei Arten von Power-Ups: <ul style="list-style-type: none"> <li>• das Sicht-PowerUp deckt ein Schiff des Gegners auf</li> <li>• das Schuss-Power-Up vergrößert den Radius, der von Schüssen getroffen wird</li> </ul>

Tabelle 2: SpielKomponenten

## 2.2 Spielablauf

Der Spielablauf sieht wie folgt aus:

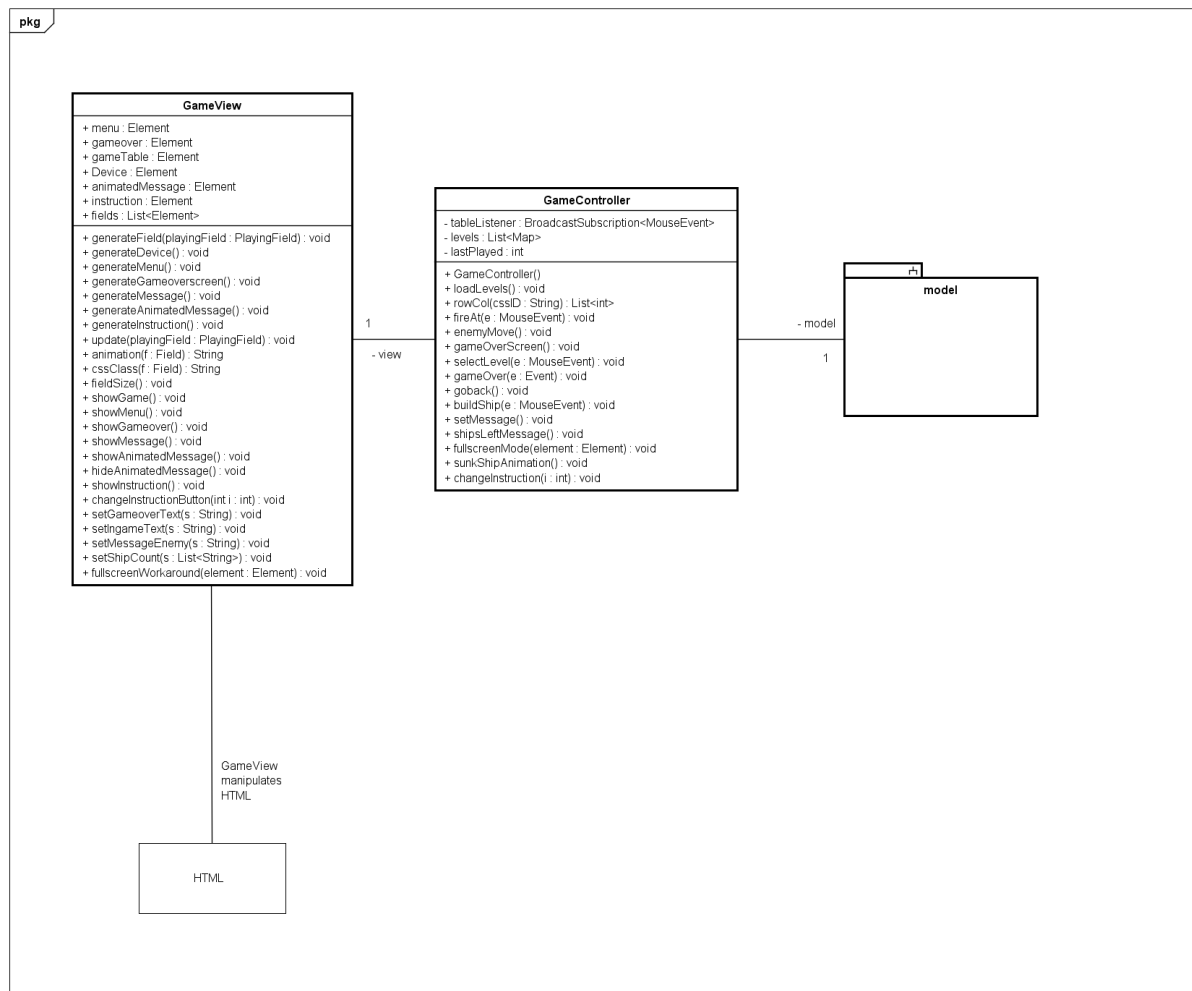
1. Die Inseln/Felsen und die Power-Ups werden zufällig auf dem Spielfeld verteilt
2. Die Parteien verteilen ihre Schiffe auf das eigene Spielfeld, wobei diese nur auf freie Kacheln gesetzt werden können
3. Parteien können nun rundenweise entscheiden, ob sie auf ein Feld des Gegners schießen möchten, oder ob sie eines ihrer Schiffe bewegen wollen, sofern dies möglich ist (siehe Regeln zum Bewegen der Schiffe). Wenn auf ein Feld geschossen wurde, wird danach signalisiert, ob es sich bei der Attacke um einen Treffer handelt (signalisiert durch rotes Kreuz) oder nicht (signalisiert durch weißes Kreuz). Treffer werden angezeigt, wenn sich auf dem Feld ein Schiff oder ein Felsen befindet.
4. Nachdem eine Partei alle Schiffe der anderen Partei zerstört hat, gewinnt diese das Spiel

## 2.3 Regeln zum Bewegen der Schiffe

1. Schiffe können sich nicht bewegen, wenn sie getroffen sind
2. Schiffe können sich nicht ins gegnerische Feld bewegen
3. Schiffe können sich nicht bewegen, wenn sich ein Hindernis im Weg befindet

## 3 Architektur/Implementierung

Das Programm ist gemäß der Aufgabenstellung nach dem MVC-Muster aufgebaut, weswegen sich die Funktionalität auf mehrere Klassen aufteilt.



powered by Astah

Abbildung 1: Architektur

Das Modell setzt sich aus mehreren Klassen zusammen, die das Spiel konzeptionell abbilden.

Die View verwaltet den DOM-Tree der HTML-Seite und bietet Methoden, um diesen zu manipulieren.

Der Controller reagiert auf die Eingaben des Nutzers und leitet aufgrund dieser Änderungen im Model ein. Des Weiteren liefert der Controller dem View Die Daten, die es aus dem Model braucht, um sich upzudaten.

## 3.1 Model

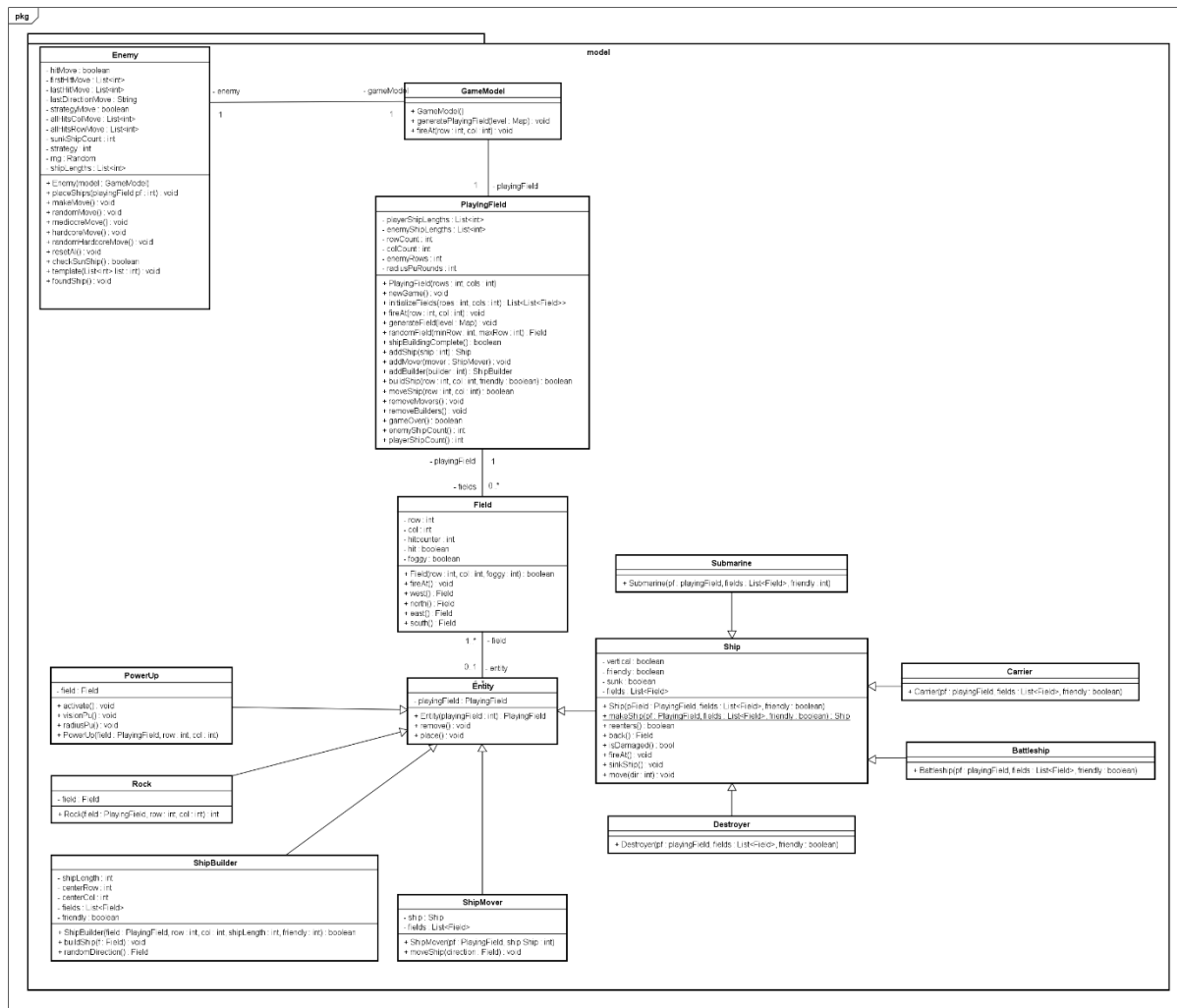


Abbildung 2: Model

Aus dem Spielkonzept haben wurden ein Schiff (Ship), ein Felsen (Rock), ein Power-Up (PowerUp) und ein Spielfeld (PlayingField) abgeleitet. Das Spielfeld enthält die Schiffe beider Parteien, die Felsen und die Power-Ups. Des Weiteren existieren im PlayingField Objekte, die dazu dienen Schiffe zu setzen (Shipuider) und zu bewegen (ShipMover). Damit der Gegner abgebildet werden kann gibt es die Klasse Enemy, welche die KIs für Spielzüge des Gegners sowie das Setzen der gegnerischen Schiffe beinhaltet. Sowohl das Spielfeld als auch der Gegner werden von der Klasse GameModel verwaltet, welche das Model zusammenfasst und dem Controller als Zugriffspunkt dient.

### 3.1.1 GameModel

Die GameModel-Klasse kapselt das Model gegenüber dem Controller ab und ist somit die Klasse aus dem Model, mit dem der Controller hauptsächlich interagiert.

Attribute:

- Der enemy ist der Gegner des Spiels.
- Das playingField ist das Spielfeld.

Methoden:

- GameModel() erzeugt eine neue GameModel-Instanz
- generatePlayingField() generiert das Spielfeld, wenn eine neue Runde gestartet wird

- fireAt() attackiert ein Feld des Spielfelds

### 3.1.2 PlayingField

Die PlayingField-Klasse modelliert das Spielfeld des Schiffe-Versenken-Spiels. Dafür hat die PlayingField-Klasse eine zweidimensionale Liste zum Attribut, die die einzelnen Fields des Spielfelds enthält. Alle Objekte, mit denen der Spieler während des Spiels interagiert befinden sich in einem Field in dieser Liste.

#### Attribute

- fields enthält alle Kacheln des Spielfelds und ist eine zweidimensionale Liste mit Fields, über die auf die Entitäten zugegriffen werden kann.
- playerShipLengths und enemyShipLengths enthalten die Längen der Schiffe des Spielers bzw. des Gegners.
- ayerBuilder und enemyBuilder sind ShipBuilder die zum Konstruieren der Schiffe für die Parteien gebraucht werden.
- rowCount und colCount geben die anzahl an Zeilen und Spalten des Spielfelds an.

#### Methoden

- PlayingField() erzeugt ein neues Spielfeld, dabei muss die Spielfeldgröße übergeben werden.
- gameOver() wird vom Controller genutzt um zu überprüfen, ob das Spiel abgeschlossen ist.
- generateField() wird genutzt um ein zufälliges Spielfeld mit einer bestimmten Anzahl von Felsen, Schiffen und Power-Ups, die in Form einer Map übergeben werden, zu generieren.
- newGame() setzt das Spielfeld zurück, löscht also zum Beispiel alle Entitäten auf dem Feld, um so ein neues Spiel vorbereiten zu können.
- randomField() liefert eine zufällige Kachel des Spielfeldes zurück und wird von Methoden innerhalb des Spielfeldes sowie einiger anderer Klassen im Model genutzt. Übergeben werden die minimale und maximale Zeile zwischen denen sich das Feld befinden soll.
- removeMovers() und removeBuilders entfernen die Objekte zum Bewegen und Bauen der Schiffe.
- addBuilder() und addMover() werden verwendet um die Objekte zum Bewegen und Bauen der Schiffe auf das Spielfeld zu setzen.
- buildShip() werden Schiffe für den Spieler und den Gegner konstruiert, diese Schiffe werden dann über addShip() aufs Spielfeld gesetzt.
- moveShip() dient dazu ein Schiff zu bewegen.
- Mit fireAt() wird auf ein Feld geschossen und so mit der Entität des Feldes interagiert
- playerShipCount() und enemyShipCount() liefern die Anzahl der übrigen Schiffe des Spielers bzw. des Gegners.

### 3.1.3 Field

Diese Klasse bildet die Kacheln des Spielfelds ab und kann höchstens eine Entität (z.B. Schiff, Felsen) enthalten. Alle Elemente die sich auf dem Spielfeld befinden sind in einer instanz der Field-Klasse eingelagert.

#### Attribute

- row und col sind die Zeile und die Spalte, in der sich die Kachel auf dem Spielfeld befindet.
- hit gibt an, ob auf das Feld schon geschossen wurde.
- foggy gibt an, ob das Feld neblig ist, also sich auf der gegnerischen Spielfeldhälfte befindet.
- entity ist die Entität, die dieses Spielfeld beinhaltet. Ist sie null, so ist das Field frei.

#### Methoden



- Field() erzeugt eine neue Kachel, es wird das Playing-Field, sowie Informationen darüber, ob das Feld neblig ist übergeben.
- fireAt() bildet einen Angriff auf die Kachel ab.
- north(), east(), west(), south() dienen dazu das Feld nördlich, östlich, südlich oder westlich des zu finden.

### 3.1.4 Enemy

Die Enemy-Klasse ist dafür zuständig, die Schiffe des Gegners zu platzieren und die Schiffe auf dem Territorium des Spielers anzugreifen. Dazu enthält KIs, welche Strategien in unterschiedlichen Schwierigkeitsstufen ausführen.

#### Attribute

- strategy gibt an, welche Strategie der Gegner aktuell verwendet.
- model ist das Modell, zu dem der Gegner gehört.
- shipLengths sind die Längen der Schiffe des Gegners.

#### Methoden

- Enemy() erzeugt einen neuen Gegner, dem Konstruktor wird das Spielfeld übergeben, auf dem der Gegner spielen wird
- placeShips() lässt den Gegner seine Schiffe platzieren
- makeMove() entscheidet, anhand der aktuell verwendeten Strategie, über die KI, die zum Einsatz kommen soll und führt dann eine der move()-Methoden aus
- randomMove() lässt den Gegner auf ein zufälliges Feld in der Hälfte des Spielers schießen
- mediocreMove() lässt den Gegner immer auf jede zweite Kachel des Spielers schießen, es sei denn ein Treffer wird angezeigt, was dazu führt, dass der Gegner erst alle Felder in der Umgebung des Treffers beschießt
- hardcoreMove(), sowie randomHardcoreMove() lassen den Gegner eine fortgeschrittene Strategie verwenden, bei der, bei einem Treffer auf ein Schiff zunächst versucht wird, dieses zu versenken. randomHardcoreMove() folgt dabei keinem Muster, sondern beschießt die Felder des Gegners zufällig.
- resetAI() setzt die KI zurück, damit ein neues Spiel begonnen werden kann

### 3.1.5 Entity

Entities sind Objekte, die auf dem Spielfeld in einer Kachel existieren können. Folglich erweitern folgende Klassen die Entity-Klasse:

- Ship
- PowerUp
- Rock
- ShipBuilder
- ShipMover

Entitäten werden über den Konstruktor Entity() erzeugt, welcher das Spielfeld auf dem sich die Entität befinden wird als Argument nimmt. Sie verfügen außerdem über die Methoden place() und remove() welche die Entität zum Spielfeld hinzufügen und entfernen.

### 3.1.6 Ship

Ships sind Entitäten die Schiffe abbilden. Sie enthalten unter anderem Informationen über den Zustand des Schiffes und die Partei zu der es gehört, sowie die Felder, die es belegt.

Die Ship-Klasse wird von den folgenden Klassen erweitert:

- Carrier – ein Schiff der Länge fünf
- BattleShip – ein Schiff der Länge vier
- Submarine – ein Schiff der Länge drei
- Destroyer – ein Schiff der Länge zwei

Diese Klassen implementieren jeweils nur einen eigenen Konstruktor und dienen dazu die unterschiedlichen Schiffstypen zu unterscheiden

Attribute der Ship-Klasse:

- vertical gibt an, ob das Schiff aus der Sicht des Spielers vertikal oder horizontal ausgerichtet ist
- friendly gibt an, ob das Schiff dem Spieler oder dem Gegner gehört
- sunk gibt an, ob das Schiff versenkt ist
- fields enthält die Kacheln, die das Schiff besetzt

Methoden der Ship-Klasse:

- Ship() erzeugt ein neues Schiff und nimmt eine Liste mit den Feldern des Schiffs, sowie einen booleschen Wert, der Auskunft über den Besitzer des Schiffes gibt, als Argumente
- makeShip() ist eine statische Factory-Methode, die Schiffe zurückliefert
- reenters() gibt an, ob das Schiff das Spielfeld an einem Rand verlässt und am andereniedereintritt.
- back() liefert die Kachel welche das Heck des Schiffes enthält.
- fireAt() bildet einen Angriff auf das Schiff ab
- sinkShip() versenkt das Schiff.
- move() bewegt das Schiff entsprechend seiner Ausrichtung.

### 3.1.7 Rock

Felsen sind eine Entität, die genau eine Kachel in Anspruch nehmen, welches vom Attribut field repräsentiert wird. Außer dem Konstruktor Rock() hat die Klasse keine weiteren Methoden.

### 3.1.8 PowerUp

Genau wie die Rock-Klasse hat die PowerUp-Klasse nur ein Attribut, nämlich die Kachel, welche sie enthält.

Der Effekt eines Power-Ups wird zufällig aus einer Reihe von Möglichkeiten gewählt sobald auf das Power-Up geschossen wird.

Methoden der PowerUp-Klasse:

- PowerUp() erzeugt ein neues PowerUp-Objekt und nimmt das Feld , welches das Powerup enthält als Argumente.
- activate() aktiviert zufällig einen der PowerUp-Effekte.
- visionPu() deckt eines der gegnerischen Schiffe auf.
- radiusPu() vergrößert den Einschlagradius von Schüssen auf das gegnerische Feld.

## 3.2 View

Die View ist für die Darstellung des Spiels verantwortlich und setzt sich aus einem HTML-Dokument und einer Klasse welche den DOM-Tree dieses Dokuments manipuliert zusammen.

### 3.2.1 HTML-Dokument

Der Body des Dokuments beinhaltet anfangs nur divs für das Menü, das Spielfeld, den Game-Over-Screen und einen Informationsdialog. Der HTML-Code für diese wird, bei Start des Programms, in der zugehörigen View-Klasse erzeugt und in dem DOM-Tree eingefügt.

```
<!DOCTYPE html>

<html>
<head>
  <meta charset="utf-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta name="scaffolded-by"
content="https://github.com/google/stagehand">
  <title>warships</title>
  <link rel="stylesheet" href="styles.css">
  <link rel="icon" href="images/icon.png">
  <script defer src="main.dart" type="application/dart"></script>
  <script defer src="packages/browser/dart.js"></script>
</head>

<body id="body">

<div id="device"></div>

<div id="menu" align="center"></div>

<div id="gameover"></div>

<div id="message"></div>

<table id="gameTable"></table>

</body>
</html>
```

### 3.2.2 GameView-Klasse

Die GameView-Klasse dient als Schnittstelle zum HTML-Dokument und manipuliert dieses um das Spiel für den Nutzer darzustellen.

Wichtige Elemente für die GameView-Klasse sind:

- Das Element mit der id #menu wird genutzt um darin das Menü des Spiels anzuzeigen. Im Menü kann der Spieler das Level auswählen, welches er spielen möchte.
- Das Element mit der id #gameover wird genutzt um darin den Game-Over-Screen darzustellen, der erscheint, nachdem ein Spiel vorbei ist. Im Game-Over-Screen kann der Spieler sich entscheiden ob er das nächste Level spielen, oder zu Hauptmenü zurückkehren möchte
- Das Element mit der id #gameTable wird genutzt um die Tabelle für das eigentliche Spiel darzustellen. Zusätzlich zu den Kacheln fürs Spiel bietet die Tabelle eines um Informationen über die aktuelle Spielsituation darzustellen und einen Button, mit dem der Spieler ins Hauptmenü zurückkehren kann
- Das Element mit der id #message wird genutzt um dem Spieler während des Spielens Informationen zu geben. Es enthält einen Button um das Spiel fortzusetzen

Die wichtigen Methoden der GameView-Klasse sind:

- generate()-Methoden (z.B. generateMenu()) generieren den HTML-Code bei Aufruf der Webseite und fügen ihn in den DOM-Tree der HTML-Seite ein. Beinahe der gesamte HTML-Text wird auf diese Weise erzeugt, weswegen das HTML-Dokument so kurz ist.
- update() updatet die View entsprechend des aktuellen Zustands des Spiels, welcher als PlayingField übergeben wird.
- animation() dient zum darstellen von Explosionsanimationen, wenn auf ein Feld geschossen wurde.
- cssClass() ermittelt für ein Feld der Tabelle die CSS-Klasse. Übergeben wird eine Field-Instanz.
- showGame(), showMenu(), showGameOver() und showMessage() dienen dazu, das Spiel, das Menü, den Game-Over-Bildschirm, oder die Hilfsnachricht des Spiels wie in den Abbildungen 3, 4, 5 und 6 anzuzeigen.
- setIngameText() und setIngameLvl() verändern den Text der in den ingame-Textboxen angezeigt wird.
- setInstructions() ändert den Text der in der Spielanleitung angezeigt wird.

Die GameView wird immer vom Controller geupdatet, wenn sich etwas auf dem Spielfeld verändert haben könnte. Wenn dies geschieht, ruft der Controller die update-Methode der View auf und übergibt ihr das PlayingField aus dem Model. Die update-Methode sucht nun für jede Zelle der Spieltabelle die geeignete CSS-Klasse für das Spiel. Die CSS-Klassen unterscheiden sich je nach Hintergrundbild der Zelle (z.B. die Sprites für Wasser, Nebel oder ein Schiff) und werden durch die cssClass-Methode ermittelt.

```
.water {
    background-size: 100%;
    background-image: url("images/water_fog/wasser.png");
}

.fog {
    background-size: 100%;
    background-image: url("images/water_fog/nebel.png");
}
```

Die View kann neben dem Spielfeld auch ein Menü mit der Levelauswahl, ein Tutorial anzeigen.



Abbildung 3: Menü

Abbildung 4: Spielfeld



Abbildung 5: Game-Over Bildschirm

### 3.3 Controller

Der Controller steuert den Spielablauf und reagiert auf Eingaben des Nutzers, ist also für das Eventhandling zuständig.

#### 3.3.1 Eventhandling

Der Controller reagiert auf Eingaben des Spielers auf die Spielfeldtabelle unterschiedlich, abhängig davon, ob alle Schiffe gesetzt wurden oder nicht. Sollten noch nicht alle Schiffe gesetzt worden sein, triggert ein Event auf die Spielfeldtabelle die `buildShip()`-Methode, welche das Model, bei einer Berührung eines leeren Felds, auffordert einen neuen ShipBuilder zu erstellen, und bei Berührung eines ShipBuilder-Pfeils das Schiff zu konstruieren. Danach wird die View geupdatet und der Text für das Textfeld des Spielfelds aktualisiert. Sollten alle Schiffe gebaut worden sein, wird die Methode für den Spielfeld-Listener zur `fireAt()`-Methode geändert.

Die `fireAt()`-Methode reagiert nur, wenn ein eigenes Schiff, oder ein Feld des gegnerischen Territoriums berührt wird. Bei Letzterem wird die `fireAt()`-Methode des GameModels aufgerufen, bei ersterem die `moveShip` des PlayingFields.

Das Menü, der Game-Over-Bildschirm und das Tutorial haben ebenfalls jeweils eigene Listener, die auf das Berühren ihrer Knöpfe reagieren. Für das Management der Menü-Buttons für die Level-Auswahl existieren im Controller die `selectLevel()`-Methode.

#### 3.3.2 Laden der Level

Der Controller ist auch für das Laden der Level zuständig. Beim Öffnen der Seite werden die Level aus der `levels.json` Datei geladen und im Controller als Liste von Maps, wobei jede Map ein Level darstellt, gespeichert.

Für das Laden der Level existiert im Controller die `loadLevels()`-Methode:

```
/**
 * loads the levels from the levels.json file
 */
void loadLevels() {
    HttpRequest.getString("levels.json").then((resp) => levels =
JSON.decode(resp));
}
```

## 4 Levelkonzept

### 4.1 Aufbau der Level

Die Level von Warships unterscheiden sich voneinander durch die Anzahl und Arten der eigenen und gegnerischen Schiffe, die Anzahl der Felsen und im eigenen und gegnerischen Territorium, die Anzahl der Power-Ups sowie die Strategie, welcher der Gegner folgt.

Die Felsen und Power-Ups werden in den beiden Territorien zufällig platziert, was den Wiederspielwert erhöht und die Schiffe werden von den beiden Parteien gesetzt. Beim Gegner ist dies auch zufallsbasiert.

Das Spiel verfügt über neun unterschiedliche Level mit ansteigendem Schwierigkeitsgrad. Im ersten Level verwendet der Gegner beispielsweise eine sehr primitive Strategie und hat genau so viele Hindernisse in seinem Territorium wie der Spieler, während er im letzten Level eine geschicktere Strategie nutzt und weniger Hindernisse in seinem Feld hat. Die neun Level sehen wie folgt aus:

Level	Schiffslängen des Spielers	Schiffslängen des Gegners	Felsen im Spielerterritorium	Felsen im Gegnerterritorium	Power-Ups	Strategie des Gegners (Schwierigkeitsgrad)
1	2, 2, 3, 3, 4	2, 2, 3, 3, 4	3	3	1	einfach
2	2, 2, 2, 3, 3	2, 2, 2, 3, 3	4	3	1	mittel
3	3, 4, 5	3, 4, 5	4	3	1	mittel
4	2, 3, 4, 5, 5	2, 3, 4, 5, 5	5	5	1	schwer
5	2, 2, 5, 5	2, 2, 5, 5	5	3	1	schwer
6	3, 3, 3, 3	3, 3, 3, 3	7	7	1	schwer
7	2, 2	2, 2	5	4	1	sehr schwer
8	2, 2	2, 2, 4, 5	5	3	1	sehr schwer
9	2, 2	2, 2, 2, 3, 4, 5	7	2	1	sehr schwer

Tabelle 3: Level

### 4.2 Parametrisierung der Level

Die Level werden als JSON-Objekte in der levels.json Datei gespeichert und sind folgendermaßen aufgebaut:

```
{
  "id": "1",
  "playerShips": [2, 2, 3, 3, 4],
  "enemyShips": [2, 2, 3, 3, 4],
  "playerRocks": 3,
  "enemyRocks": 3,
  "enemyPowUps": 1,
  "enemyStrategy": 0,
}
```

- id: eine Nummer um das Level zu identifizieren

- playerShips: die Längen aller Schiffe des Spielers und somit implizit ihre Anzahl
- enemyShips: die Längen aller Schiffe des Gegners und somit implizit ihre Anzahl
- playerRocks: die Anzahl der Felsen/Inseln im Territorium des Spielers
- playerRocks: die Anzahl der Felsen/Inseln im Territorium des Gegners
- enemyStrategy: ein Integer-Wert welcher die Strategie, welche der Gegner verwendet wird, angibt

Die levels.json-Datei wird beim Start des Spiels vom Controller geladen und als Liste von Maps mit den Levelinformationen gespeichert.

Beim Start eines Levels werden dem Model die Informationen für das Level durch die generatePlayingField-Methode des Models übermittelt (siehe 3.1.1).

## 5 Nachweis der Anforderungen

ID	Kurztitel	Erfüllt	Teilw. Erfüllt	Nicht Erfüllt	Erläuterung
AF-1	Single-Player-Game als Single-Page-App	X			Warships ist ein Singleplayer-Spiel welches eine einzige HTML-Seite lädt und diese manipuliert, wenn der Spieler mit der Anwendung interagiert (siehe 3.2). Warships kann als statische Webseite von beliebigen Webservern oder Content Delivery Networks bereitgestellt werden.
AF-2	Balance zwischen technischer Komplexität und Spielkonzept	X			Schiffe versenken ist ein interessantes klassisches Spiel, welches in unserer Interpretation, durch das Bewegen von Schiffen und die Einführung von Power-Ups und Hindernissen sowie die Implementierung der KI des Gegners an Komplexität gewinnt. Das Spielprinzip ist weithin bekannt, weswegen Warships intuitiv erfassbar und einfach zu bedienen ist. Sollten die neuen Features dennoch für Verwirrung sorgen, enthält das Spiel eine Spielanleitung, welche vom Hauptmenü aus aufrufbar ist
AF-3	DOM-Tree-basiert	X			Warships View basiert ausschließlich auf der Manipulation des DOM-Trees und verwendet keine Canvas-Elemente. Auch ist es nach dem MVC-Muster strukturiert (siehe Abbildung 1)
AF-4	Target-Device: SmartPhone	X			Warships ist auf Smartphones zugeschnitten, da es sehr bequem und intuitiv vom Touchscreen aus bedient werden kann, und die Darstellung des Spiels ebenfalls für Mobilgeräte optimiert ist. Warships kann sowohl auf iOS- als auch auf Android-Devices gespielt werden
AF-5	Mobile First Prinzip	X			Warships wurde bewusst für Smartphones konzipiert.
AF-6	Das Spiel muss schnell und intuitiv erfassbar sein und Spielfreude erzeugen.	X			Warships Spielprinzip ist durch „Schiffe Versenken“ vielen Menschen bekannt. Die neuen Features sind ebenfalls intuitiv erfassbar

AF-7	Das Spiel muss ein Levelkonzept vorsehen	X			Warships verfügt über neun Level mit steigendem Schwierigkeitsgrad, welche durch JSON-Objekte beschrieben werden (siehe 4.1).
AF-8	Ggf. erforderliche Speicherkonzepte sind Client-seitig zu realisieren	X			Das Spiel ist Client-seitig realisiert.
AF-9	Dokumentation	X			Die Dokumentation des Spiels ist vollständig und orientiert sich am Beispiel.

Tabelle 4: Anforderungen

## 6 Verantwortlichkeiten

V – Verantwortlich (hauptdurchführend, kann nur einmal pro Zeile vergeben werden)

U – Unterstützend (Übernahme von Teilaufgaben)

Komponente	Detail	Asset	Luca Thurm	Max Zimmermann	Anmerkungen
Model	GameModel	lib/src/model.dart	V	U	Enthält Inline-Dokumentation
	Enemy	lib/src/model.dart	U	V	Enthält Inline-Dokumentation
	PlayingField	lib/src/model.dart	V		Enthält Inline-Dokumentation
	Field	lib/src/model.dart	V		Enthält Inline-Dokumentation
	Ship	lib/src/model.dart	V		Enthält Inline-Dokumentation
	Carrier	lib/src/model.dart	V		Enthält Inline-Dokumentation
	Battleship	lib/src/model.dart	V		Enthält Inline-Dokumentation
	Submarine	lib/src/model.dart	V		Enthält Inline-Dokumentation
	Destroyer	lib/src/model.dart	V		Enthält Inline-Dokumentation
	Rock	lib/src/model.dart	V		Enthält Inline-Dokumentation
	PowerUp	lib/src/model.dart	V		Enthält Inline-Dokumentation
	ShipBuilder	lib/src/model.dart	V		Enthält Inline-Dokumentation
	ShipMover	lib/src/model.dart	V		Enthält Inline-Dokumentation
View	HTML-Dokument	index.html		V	
	CSS-Datei	styles.css	U	V	
		img/*		V	
	View-Logik	lib/src/view.dart	U	V	Enthält Inline-Doku
Controller	Eventhandling	lib/src/controller.dart	V	U	Enthält Inline-Doku
	Level	levels.json	V		
Dokumentation			V		

Tabelle 5: Verantwortlichkeiten

## 7 Nutzung des Spiels

Wir sind damit einverstanden, dass unser Spiel öffentlich zugänglich bereitgestellt wird. Durch unser Spiel werden keine Rechte an Bildern oder sonstigen eingesetzten Medien verletzt.