Programmieren 1 - Praktikum 5

Benjamin Meyer, Michael Roth

Lernziele

- Verwenden von Klassen
- Implementieren einfacher Programmlogik
- Umgang mit `std::vector` und `std::array`

1 Vorbereitung

Ihnen steht im Repository ein Vorgabeprojekt zur Verfügung, **welches zu benutzen ist**. Falls Sie eine andere IDE als QtCreator verwenden wollen, dann benutzen Sie dennoch die dort beigefügten `.h` und `.cpp` Dateien.

1.1 Erweiterungen der Header

Sie dürfen die mitgelieferten Headerdateien des Vorgabeprojekts beliebig **erweitern**. Das bedeutet, Sie können zu den Klassen beliebig viele Attribute und Methoden **hinzufügen**, so lange diese **`private`** sind.

Sie dürfen keine der bestehenden Attribute oder Methoden **löschen** oder **verändern**, oder Methoden bzw. Attribute als `public` hinzufügen.

Kurz: Alles was `public` ist, bleibt unverändert!

1.2 Dokumentation

Ebenfalls im Repository enthalten ist eine Dokumentation des Vorgabequellcodes als `.pdf` Datei. Diese Dokumentation ist automatisch aus den Kommentaren in den Header Dateien durch das Tool `doxygen` erzeugt.

⇒ Die Dokumentation soll dazu dienen, dass Sie noch mal in Ruhe nachlesen können, was die einzelnen Funktionen und Methoden tun sollen. Weiterhin wollten wir Ihnen zeigen, wie eine automatisch generierte Dokumentation aussehen kann.

2 Das Spiel „Schiffe versenken“ (engl. „Battleships“)

In diesem Praktikum implementieren Sie das (hoffentlich) bekannte Spiel „Schiffe versenken“[1] bzw. „Battleships“[2] (englisch). Die Wikipedia Seiten zu den Regeln sind in den Fußnoten verlinkt, sollten Sie damit nicht vertraut sein.

Das Spiel an sich ist für zwei Spielerinnen gedacht, die eigentlich nicht das Spielbrett der anderen Spielerin sehen sollten. In dieser Implementierung jedoch werden die Spielbretter beider Spielerinnen auf der gleichen Konsole ausgegeben (Es geht ja nicht um die Erschaffung des perfekten Spiels, sondern um die eigentliche Programmierung dahinter!).

Grundsätzlich geht es darum, dass jede Spielerin auf einem Spielfeld von 10×10 Feldern insgesamt zehn Schiffe verschiedener Länge (zwei bis fünf Felder) verteilen. Die jeweilige Gegnerin „beschießt“ das Spielfeld der Spielerin durch Angabe von Reihe und Spalte eines Feldes. Die Spielerin sagt dann entweder „Wasser“ wenn kein Schiff getroffen wurde bzw. „Treffer“ wenn ein Schiff durch den Schuss getroffen wurde.

In der Regel besitzt jede Spielerin noch einen zusätzlichen „Schmierzettel“, auf dem dieser notiert welche Felder der Gegnerin bereits beschossen wurden.

[1] `https://de.wikipedia.org/wiki/Schiffe_versenken`
[2] `https://en.wikipedia.org/wiki/Battleship_(game)`

Für die Implementierung des Spiels werden insgesamt fünf Klassen entworfen:

`Player` stellt eine Spielerin dar und speichert den Namen sowie jeweils die Anzahl der gewonnenen und verlorenen Spiele.

`Part` stellt ein Schiffsteil dar, welches genau ein Feld belegt. Ein Schiff besteht damit aus zwei bis fünf Objekten von `Part`. Ein Schiffsteil kann entweder unbeschädigt oder beschädigt sein (Wenn es getroffen wurde).

`Ship` stellt ein Schiff dar, welches eben aus verschieden vielen Teilen und damit aus Objekten von `Part` besteht. Das Schiff als solches ist entweder:

- unbeschädigt, wenn alle Schiffsteile unbeschädigt sind,
- beschädigt, wenn mindestens ein (aber nicht alle!) Schiffsteil beschädigt ist,
- oder aber versenkt, wenn alle Schiffsteile beschädigt sind

`GameBoard` verfügt über ein `std::array` mit den Schiffen eines Spielers sowie dem „Schmierzettel“ zur Markierung welches gegenerische Feld bereits beschossen wurde. Die Klasse ist ebenfalls zuständig für die Ausgabe des Spielfelds bzw. des „Schmierzettels“.

`Battleship` ist die „Hauptklasse“ des Spiels und verwaltet zwei `Player` Objekte sowie die jeweils dazugehörigen `GameBoard` Objekte.

Hinweis: Lesen Sie unbedingt auch die mitgelieferte Dokumentation des Quellcodes, vor allen Dingen bei Unklarheiten!

3 Klasse `Player`

Die Klasse `Player` dient einfach nur der Verwaltung einer Spielerin. Dort werden für jeden Spieler folgende Informationen gespeichert:

- Name der Spielerin
- Anzahl der gewonnen Spiele
- Anzahl der verlorenen Spiele

Der (einzige) Konstruktor von `Player` bekommt einen `string` übergeben, welcher als Name verwendet werden soll. Weiterhin soll der Konstruktor die Anzahl der gewonnen bzw. verlorenen Spiele mit dem Wert 0 initialisieren.

Der Name der Spielerin soll dabei ein `const` Attribut der Klasse `Player` sein.

Hinweis: Hier ist **explizit** eine Initialisierungsliste zu verwenden!

4 Klasse `Part`

Die Klasse `Part` stellt ein Schiffsteil dar, welches genau ein Feld des Spielfelds belegt. Daher „weiß“ ein `Part` Objekt, an welcher Position des Spielfelds es sich selbst befindet. Die Attribute von `Part` sind:

- Reihe und
- Spalte
- Status, also ob das Teil unbeschädigt oder beschädigt ist

Für die Position existieren jeweils die beiden Getter-Methoden `getRow` bzw. `getCol`. Für den Status existiert der Getter `isDamaged`, welcher `true` liefert falls dieses Teil beschädigt ist, sowie der Setter `setDamaged`, welcher den Status auf „beschädigt“ setzt. Eine „Reparatur“, also ein Zurücksetzen des Status auf „unbeschädigt“, ist nicht vorgesehen.

Der (einzige) Konstruktor von `Part` soll ein unbeschädigtes Schiffsteil an der in den Parametern übergebenen Position erstellen.

5 Klasse `Ship`

Die Klasse `Ship` stellt ein Schiff für das Spiel dar und besteht im Wesentlichen nur aus einer Sammlung an `Part` Objekten. Ein `Ship` Objekt selbst „weiß“ nicht an welcher Position es steht, daher sind die Positionen einzeln in den `Part` Objekten gespeichert. Als Datenstruktur für die `Ship` Objekte wird ein `std::vector` verwendet da Schiffe einer unterschiedliche Anzahl an Teilen besitzen (bzw. Schiffe unterschiedlich lang sind).

Der Konstruktor `Ship(int row, int col, int lengthOfShip, Direction direction)` wird verwendet, um ein neues Schiff zu erstellen. Dabei gilt:

- `row` und `col` geben die Position des ersten Schiffteils an
- Insgesamt sollen `lengthOfShip` Teile erstellt werden (Nach dem ersten also noch `lengthOfShip - 1` weitere Teile)
- Das Schiff soll in die Richtung `direction` ausgerichtet werden. Alle weiteren Teile nach dem ersten werden also wie folgt erstellt:
  - Nach oben, wenn `direction = Direction::north`
  - Nach rechts, wenn `direction = Direction::east`
  - Nach unten, wenn `direction = Direction::south`
  - Nach links, wenn `direction = Direction::west`

  Wobei die Reihe 0 die oberste Reihe und Spalte 0 die linkeste Spalte ist.

5.1 Methode `hasPartIn(int row, int col)`

Diese Methode soll `true` zurück liefern, falls ein Teil dieses Schiffes an der angegebenen Position liegt. Dazu müssen Sie den `std::vector` mit den `Part` Objekten durchsuchen, ob eben ein solches Teil existiert. Falls nicht, soll die Methode `false` zurück geben.

5.2 Methode `getPartIn(int row, int col)`

Diese Methode liefert eine **Referenz** auf das Schiffsteil an der angegebenen Position zurück. Ist kein solches Teil vorhanden, soll die Methode eine **Exception** vom Typ `std::invalid_argument` liefern.

Update: Sie können auch die Exception weg lassen und davon ausgehen, dass die übergebenen Werte korrekt sind.

5.3 Methoden `isSunk()` und `isDamaged`

Diese Methoden sollen ermitteln, ob das Schiff versenkt wurde bzw. ob es beschädigt ist. Ein Schiff gilt als versenkt, sobald **alle** seine Teile beschädigt sind. Ein Schiff ist beschädigt, wenn mindestens eines der Teile beschädigt ist. Damit ist ein versenktes Schiff natürlich auch beschädigt.

6 Klasse `GameBoard`

Die Klasse `GameBoard` stellt eine Aufzählung der Schiffe einer Spielerin zur Verfügung (als `std::array<Ship>`). Weiterhin verwaltet die Klasse den „Schmierzettel“ (als `std::array<std::array<char, 10>, 10>`auf dem notiert werden soll, welche Positionen der Spielerinnen bei der Gegnerin bereits beschossen hat.

6.1 Methoden `printBoard()` und `printEnemyBoard()`

Diese Methoden sollen das Spielfeld der Spielerin sowie ihren „Schmierzettel“ ausgeben.

Für die Ausgabe des Spielfelds muss ermittelt werden, an welchen Positionen sich die Schiffe bzw. deren Teile befinden. Geben Sie für jedes Feld des 10×10 Spielfeldes genau ein Zeichen aus, welches anzeigt was sich auf dem Feld befindet. Ein Feld kann dabei folgende „Zustände“ haben:

- Ein Wasserfeld, auf dem sich kein Schiff befindet. Benutzen Sie hierfür beispielsweise einen Punkt `'.'`

- Ein unbeschädigtes Schiffsteil, dargestellt durch die Nummer des zugehörigen Schiffes. Die Nummer ist identisch zum Index des Schiffes innerhalb des Arrays.
- Ein beschädigtes Schiffsteil kann durch ein `'X'` dargestellt werden
- Ein Teil eines versenkten Schiffes kann beispielsweise durch `'S'` oder auch `'#'` dargestellt werden.

Der Schmierzettel besteht aus lauter `char` Werten, wobei jedes Feld drei Zustände haben kann:

- Noch kein Schuss ausprobiert: `'.'`
- Treffer an dieser Stelle: `'X'`
- Wasser an dieser Stelle: `'O'`

6.2 Methoden `hit(int row, col)` und `mark(int row, col, bool wasHit)`

Die Methode `hit` wird vom Gegner aufgerufen und stellt den Beschuss des angegebenen Feldes dar. Falls sich dort ein Schiffsteil befindet, so wird dieses beschädigt und `true` zurück gegeben. Falls nicht, wird `false` zurück gegeben (Die gegnerische Spielerin muss wissen, ob sie etwas getroffen hat oder nicht).

Mit der Methode `mark` wird an der angegebenen Stelle des „Schmierzettels“ markiert, ob dort ein Schiff getroffen wurde oder nicht. Die Methode soll dann an die entsprechende Stelle ein `'X'` bzw. ein `'O'` schreiben.

6.3 Methode `randomPlaceShips()`

Diese Methode soll zufällig auf dem Spielfeld die folgenden Schiffe verteilen:

- Ein 'Schlachtschiff' bestehend aus fünf Teilen
- Zwei 'Kreuzer' bestehend aus vier Teilen
- Drei 'Zerstörer' bestehend aus drei Teilen
- Vier 'U-Boote' bestehend aus zwei Teilen

Die Schiffe sollen dabei so platziert werden, dass diese sich nicht gegenseitig überschneiden und alle Teile tatsächlich auf dem Spielfeld sind (Siehe auch den Konstruktor von `Ship` in Aufgabe 5 auf der vorherigen Seite).

Weiterhin soll jeder Aufruf tatsächlich eine neue Verteilung der Schiffe generieren. Insbesondere sollen beide Spieler verschiedene Verteilungen beim Start bekommen.[3]

6.4 Konstruktor

Der Konstruktor von `GameBoard` soll den „Schmierzettel“ mit dem Zeichen für „Noch kein Schuss ausprobiert“ befüllen und kann wahlweise ebenfalls `randomPlaceShips()` aufrufen.

6.5 Methode `allShipsSunk()`

Diese Methode soll `true` zurück liefern, falls **alle** Schiffe dieser GameBoard Instanz als „versenkt“ gelten.

7 Klasse `Battleship`

In der Klasse `Battleship` findet das eigentliche Spiel statt.

Dem Konstruktor der Klasse werden dabei zwei `string` Objekte übergeben, mit denen der Konstruktor die beiden `Player` Objekte **initialisieren** soll.

Ein Aufruf der Methode `play()` startet nun eine **Spielrunde**. Das bedeutet, dass zwei „frische“ GameBoard Objekte erzeugt werden müssen, auf denen die beiden Spieler spielen können. Die `Player` Objekte werden einmalig im Konstruktor erzeugt und initialisiert.

[3] Es gibt exakt 26.509.655.816.984 verschiedene Möglichkeiten die Schiffe aufzustellen. Wenn ihr Verfahren hinreichend zufällig ist, können Sie also davon ausgehen, dass beide Spieler unterschiedliche Startbedingungen haben.

Anschließend wird ein Startspieler ermittelt und jede Spielerin bekommt abwechselnd ihr Spielfeld sowie seinen „Schmierzettel“ ausgegeben und kann anschließend ein Feld angeben, welches sie bei der Gegnerin beschießen möchte. Die aktive Spielerin wird informiert, ob der Schuss getroffen hat oder nicht. Das Ergebnis wird automatisch auf dem „Schmierzettel“ der Spielerin vermerkt.

Nachdem die aktive Spielerin ihren Zug gemacht hat, ist die Gegnerin dran.

Die beiden Spielerinnen wechseln sich so lange ab, bis eine Spielerin gewonnen hat. Dies ist der Fall, wenn alle Schiffe der Gegnerin versenkt wurden. Beim Ende des Spiels sollen ebenfalls die Spielstatistiken in den Spielern aktualisiert werden.