ENTWICKLUNGSPORTFOLIO TEILKOMPONENTE JAVASCRIPT CLIENT

Modul 326 Objektorientiert entwerfen

Joel Blaser
Inf2019d

19.06.2021
Version 1

1 Inhaltsverzeichnis

2 Abbildungsverzeichnis

3 Lernjournal

3.1 Woche 1 / 27.5.2021

Heute haben wir mit dem Programmieren der Applikation begonnen. Ich habe zusammen mit Linus Ackermann eine Angular Applikation innerhalb des Repository aufgesetzt. Zusätzlich haben wir den Store für NGRX aufgesetzt, um später das Labyrinth darin speichern zu können. Dabei hatten wir keine Probleme. Die Applikation kann gestartet werden.

Wir hatten heute die Möglichkeit von zuhause zu arbeiten und haben diese genutzt. Für Schultage, an welchen man hauptsächlich am Programmieren ist, finde ich es viel praktische Zuhause arbeiten zu können, da ich zuhause zwei Monitore habe.

3.2 Woche 2 / 3.6.2021

Heute sind wir leider mit dem Code nicht soweit gekommen, wie wir es vorhatten. Wir konnten zumindest das Grundgerüst des Layouts des Designs machen, inklusive beriets funktionierendem Inputfeld für den Spielernamen.
Nebenbei haben wir noch nach einer Lösung gesucht, wie wir mit unserem Client eine TCP-Socket Verbindung herstellen können. Da die Browser zurzeit noch keine TCP-Verbindungen unterstützen, haben wir entschieden, dass wir einen Node JS Server für die Verbindung zwischen unserem Client und dem eigentlichen Server benutzen werden müssen. Diesen wollen wir das nächste Mal umsetzen.

3.3 Woche 3 / 10.6.2021

Heute konnten wir den Node JS Server zum laufen bringen und über ein Web Socket von unserem Client eine Verbindung herstellen. Vom Node JS Server aus, konnten wir dann die Verbindung weiterleiten auf den Bombermanserver. Zudem konnten wir die Logik für das Zeichnen des Labyrinths implementieren.
Der Grund, warum wir heute so gut vorangekommen sind ist, dass wir eine sehr gute Anleitung für Web Sockets gefunden haben und ich selbst bereits schon ein wenig Erfahrung mit Node JS hatte.

3.4 Woche 4 / 17.6.2021

Heute konnten wir die Abhandlung der Nachrichten zwischen Client und Server fertigstellen, so dass das Labyrinth je nach Nachricht entsprechend angepasst wird. Da wir beide schon Erfahrung mit Angular haben, lief dies sehr gut. Leider sind wir noch auf ein serverseitiges Problem gestossen, wodurch die Inhalte der Nachrichten nicht beim Server ankommen. Wir müssen dazu noch auf das Serverteam warten, um zu sehen, ob das Problem noch gelöst werden kann.
Auch unsere Zusammenarbeit im Team läuft sehr gut, da ich und Linus im selben Betrieb sind und auch schon oft bei Projekten im Team gearbeitet haben.

4 Entwurfsmuster – Observer

4.1 Problemstellung

Wir haben in unserem Client eine Komponente, welche das Labyrinth, inklusive der Spieler und Bomben, grafisch darstellt. Der Aktuelle Zustand des Labyrinths wird in einem Objekt gespeichert. Jedes Mal, wenn sich der Zustand des Labyrinths ändert, soll die Komponente das Labyrinth neu grafisch darstellen. Die Klasse mit dem Labyrinth sollte aber nichts von der Logik der Komponente wissen.

4.2 Auswahl des passenden Entwurfsmusters

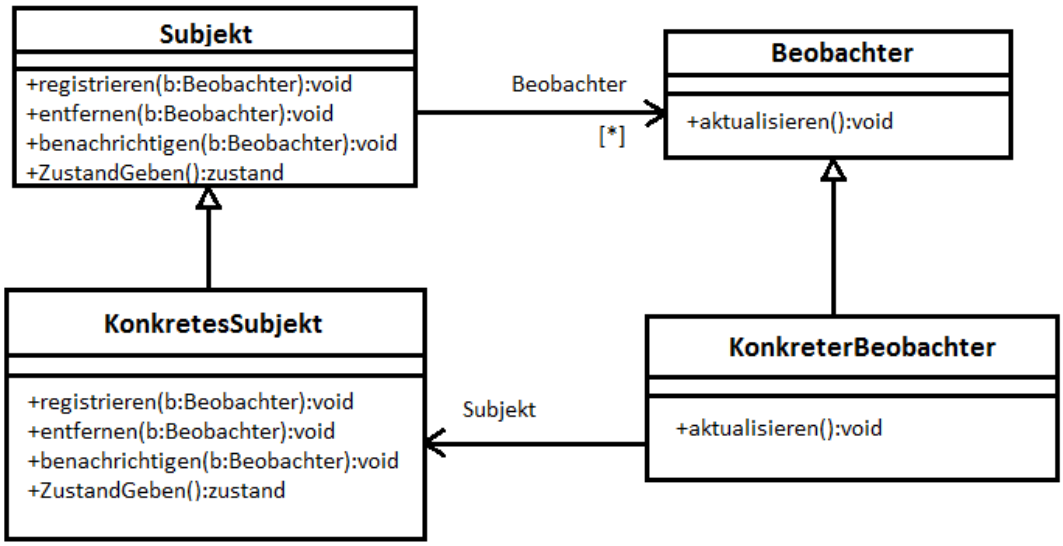

Abbildung 1 Observer Design Patterns (Quelle: https://de.wikipedia.org/w/index.php?curid=8536475)

Das Observer Entwurfsmuster wird verwendet, um Änderungen bei einem Objekt an dessen Beobachter weiterzugeben. Es gehört in die Kategorie der Verhaltensmuster.
Der Beobachter (Abonnent) muss die Struktur des Subjekts (beobachtetes Objekt) nicht kennen, da der Beobachter benachrichtigt wird, sobald beim Subjekt eine Änderung vorgenommen wurde. Der Beobachter wird aber nur benachrichtigt, wenn er dies möchte. Er kann sein «Abonnement» jederzeit künden.

Das Observer Entwurfsmuster wird noch einmal in drei Arten unterteilt.

4.2.1 Push Notification

Wenn sich das beobachtete Objekt ändert, werden die Beobachter benachrichtig, jedoch werden keine Daten mitgesendet. Das heisst, dass die Beobachter sich die Daten nach einer Nachricht selbst noch beschaffen müssen.

4.2.2 Push-Update Notification

Wenn sich das beobachtete Objekt ändert, werden die Beobachter benachrichtig und die geupdateten Daten werden mit der Benachrichtigung mitgesendet.

4.2.3 Pull Notification

Wenn sich das beobachtete Objekt ändert, werden die Beobachter nicht benachrichtigt. Die Beobachter müssen selbstständig nach dem Zustand des Objekts fragen.

4.3 Anwendung auf das ursprüngliche Problem

Da wir für unsere Teilkomponente Angular nutzen. greifen wir auf die Library RxJS zurück, welche ebenfalls von Angular selbst benutzt wird. Zusätzlich nutzen wir noch die Library NgRx, welche das von Redux inspirierte State Management, für Angular angepasst, zur Verfügung stellt. NgRx selbst greift auch wieder auf RxJS zurück.

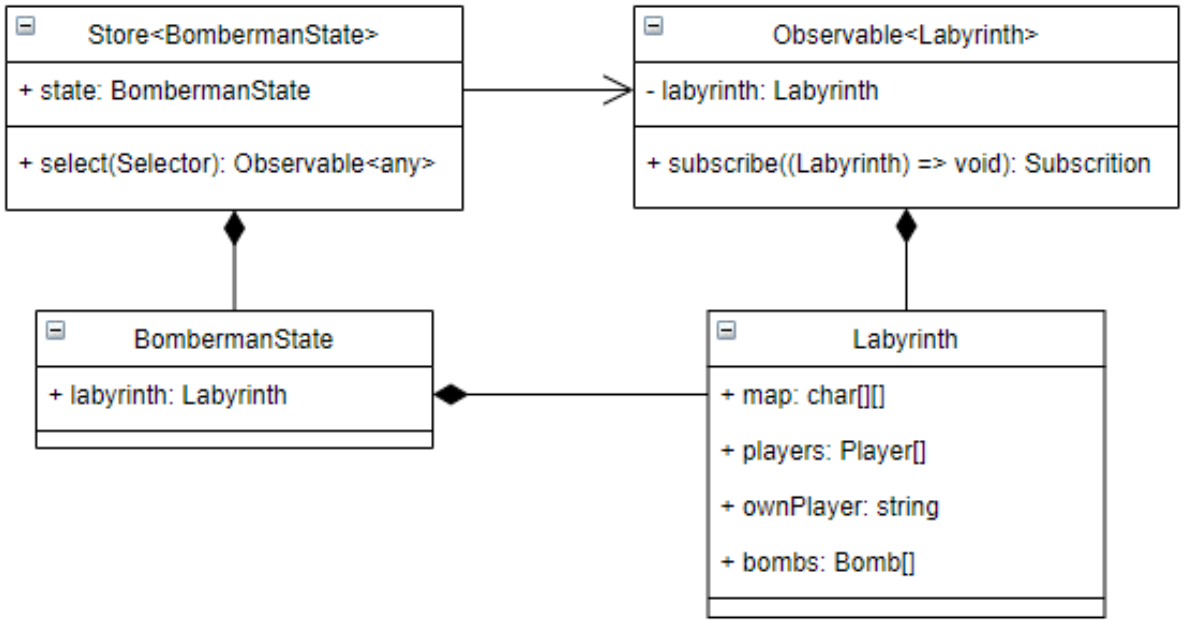

Abbildung 2 Observable and State Class Diagramm

Der Store hat den aktuellen State des Bomberman Spiels gespeichert. Dieser State beinhaltet das Labyrinth. Vom Store kann das Labyrinth selektiert werden. Dies gibt einem ein Observable (beobachtbares Objekt) mit dem Typen Labyrinth zurück. Dieses Objekt kann dann abonniert werden. Beim Abonnieren gibt man als Parameter eine Callback Methode mit, welche immer ausgeführt wird, wenn sich der Zustand des Labyrinths ändert.

Im folgenden Code Ausschnitt wird das Labyrinth vom State selektiert und anschliessend wird auf das erhaltenen Observable abonniert. Als Parameter wird eine Callback Methode übergeben, welche die Properties in der eigenen Klasse aktualisiert und die Methode "drawGame()" aufruft, welche das Labyrinth neu zeichnet. Die Callback Methode erhält jeweils als Parameter die neue Version des Labyrinths.

```
this.store.select(getLabyrinth).subscribe((labyrinth) => {
  this.map = labyrinth.map;
  this.players = labyrinth.players;
  this.ownPlayer = labyrinth.ownPlayer;
  this.bombs = labyrinth.bombs;

  if (this.size) {
    this.drawGame();
  }

  this.size = labyrinth.map.length;
});
```

Abbildung 3 Observable Subscription Code Example