SoPra - **Projektbeschreibung**

- Einzelphase -

October 4, 2021

Max Musterfrau **0000001**

Dieses Dokument erklärt oberflächlich den Aufbau des Servers der Referenzimplementierung aus der Gruppenphase. In Ihrer Abgabe müssen Sie die Funktionsweise des Servers nicht genauer erklären, solange Sie auf unserer Implementierung aufbauen. Sie sollten jedoch möglichst detailliert die eingeführten Änderungen und deren Funktionsweise erklären. Je ausführlicher Ihre Erklärung ist, desto besser können wir Ihren Entwurf bewerten. Dies kann im Zweifel entscheidend für das Bestehen des Software-Praktikums sein.

Funktionsweise

Hier beschreiben Sie, wie Ihr Server funktioniert und wie Sie die Änderungen umgesetzt haben.

Das Grundprinzip, nach dem wir unsere Implementierung entworfen haben, war *MVC*. Wir haben also versucht, Logik und Daten voneinander abzutrennen. Um eben dies zu erreichen, nutzen wir eine Command- sowie zwei Visitor-Strukturen, welche die Spiellogik kapseln. Davon abgegrenzt sind unsere Datenklassen. Diese enthalten (fast) keine Logik, sondern nur Daten. Eine Ausnahme hierzu sind erweiterte Getter und Setter, welche den Spielzustand intern konsistent halten sollen.
Falls die Spielerin nun ein Command an den Server sendet, so wird dieser direkt ausgeführt. Hierbei operiert der Command ausschließlich auf unserem Model und bekommt (zum Senden von Events) noch einen `ConnectionWrapper`, welcher die `ServerConnection` wrapped. Für den Fall, dass ein Command ausgeführt wird, welcher von einer Ability beeinflusst wird, so wird ein entsprechender `AbilityVisitor` erstellt und der Visitor übernimmt die entsprechende Logik des eingegangenen Commands. Es gibt außerdem eine separate Visitor-Struktur, um das Nutzen von Karten abzuhandeln. Unsere Abilities sind als *Decorator* modelliert.
Ebenfalls nutzen wir einen Builder und Factories, um das Einlesen und Validieren der Konfigurationsdatei möglichst gut von unserer eigentlichen Modelstruktur abzukapseln.

Designentscheidungen

Beschreiben Sie hier genau, wie Sie die bestehende Implementierung geändert haben und begründen Sie Ihre Entscheidungen.

Wir haben uns hauptsächlich für drei Patterns entschieden: *Builder*, *Visitor* und *Command*.

- **Builder**: Die Nutzung eines Builders erlaubt es uns, die Erstellung der Entities und der Welt zu kapseln. Ebenfalls haben wir uns dazu entschieden, das Parsen und Validieren so modular zu halten, dass es prinzipiell möglich wäre, eine andere Modelstruktur damit aufzubauen.
- **Visitor**: Die Überlebenden haben verschiedene Fähigkeiten und können durch das Ausrüsten bestimmter Karten weitere Fähigkeiten hinzugewinnen. Um auf die richtige Fähigkeit korrekt zu reagiern, müssen wir wissen um welche es sich handelt. Dies realisieren wir durch die entsprechenden Visitors.
- **Command**: Das Commandpattern bietet sich bei diesem Projekt gut an, da es die Interaktion mit der `ServerConnection` vorsieht, Commands zu verwenden. Wir nutzen die Commands, um die Befehle der Spieler auszuführen und die Logik der Befehle vom Rest des Spiels abzukapseln.

Herausforderungen

Welche Änderungen waren am schwierigsten umzusetzen? Wie haben Sie die daraus resultierenden Probleme gelöst?

Die größte Herausforderung bei diesem Projekt war es, das Handhaben der Fähigkeiten und Ausrüstungskarten zu modellieren ohne die Spiellogik überall zu verteilen, bzw. ohne mit `instanceof` o.Ä. herauszufinden mit welchen Abilities man gerade arbeitet. Die Nutzung der Visitors hilft hierbei.