Funktionale und objektorientierte Programmierkonzepte
Projekt

TECHNISCHE UNIVERSITÄT DARMSTADT

Entwurf

Achtung: Dieses Dokument ist ein Entwurf und ist noch nicht zur Bearbeitung/Abgabe freigegeben. Es kann zu Änderungen kommen, die für die Abgabe relevant sind. Es ist möglich, dass sich **alle** Aufgaben noch grundlegend ändern. Es gibt keine Garantie, dass die Aufgaben auch in der endgültigen Version überhaupt noch vorkommen und es wird keine Rücksicht auf bereits abgegebene Lösungen genommen, die nicht die Vorgaben der endgültigen Version erfüllen.

FOP Projekt **Gesamt: -4 Punkte**

Die Siedler von Catan

Beachten Sie die Seite *Verbindliche Anforderungen für alle Abgaben* im Moodle-Kurs.

Verstöße gegen verbindliche Anforderungen führen zu Punktabzügen und können die korrekte Bewertung Ihrer Abgabe beeinflussen. Sofern vorhanden, müssen die in der Vorlage mit `TODO` markierten `crash`-Aufrufe entfernt werden. Andernfalls wird die jeweilige Aufgabe nicht bewertet.

Die für diese Hausübung relevanten Verzeichnisse sind `src/main/java/hProjekt` und ggf. `src/test/java/hProjekt`.

Organisation und Information

Grundlegende Informationen und Bonus

Das FOP-Projekt ist *nicht* verpflichtend. Sie benötigen es *nicht* um die formale Prüfungszulassung (Studienleistung) zu erreichen. Sie können jedoch mit dem FOP-Projekt zusätzliche Bonuspunkte erreichen. Wir empfehlen jedoch dringend am FOP-Projekt teilzunehmen, da Sie hier richtig programmieren lernen. Veranstaltungen in weiterführenden Semestern setzen das Programmieren mehr oder weniger stillschweigend voraus.

Voraussetzung für den Bonus ist die bestandene Studienleistung. Sie können sich - neben den Bonuspunkten der Hausübungen - im Projekt weitere Bonuspunkte für die Klausur erarbeiten. Im Projekt können Sie zusätzlich XX Punkte erreichen. Ihre erreichte Punktzahl wird am Ende durch X.X geteilt und auf die nächstkleinere, ganze Zahl abgerundet werden. Zusammenfassend können Sie also bis zu XX Punkte zusätzlich für Ihr Punktekonto erarbeiten.

Sie werden das Abschlussprojekt in Teams von genau vier Personen bearbeiten.

Zeitplan im Überblick

- Date - Time: Deadline zur Anmeldung
- Date - Time: Verbindliche Teamtrainings der Hochschuldidaktischen Arbeitsstelle (HDA)
- Date - Time: Poolsprechstunden
- 15.03.2023 - 23:50 Uhr: Abgabe des Projekts

Poolsprechstunden

Die Projektutor*nnen bieten für Fragen vom **Datum-von - Datum-bis** Sprechstunden an. Sie finden eine Übersicht über die Sprechstundentermine im Projektabschnitt des zugehörigen moodle-Kurses.

Moodle - Forum

Sie finden zwei Foren im Projektabschnitt des Moodle-Kurses. In einem Forum können Sie Fragen hinsichtlich der Organisation und des Zeitplans stellen. Das andere Forum ist für Fragen rund um die Projektaufgaben gedacht. Fragen im Forum werden auch außerhalb der Poolsprechstunden beantwortet.

Anmeldung (bis Datum)

Um sich anzumelden, bilden Sie Gruppen aus **genau** 4 Studierenden. Sollten Sie noch keine Gruppe haben, so können Sie im Projektabschnitt des **moodle**-Kurses ein Forum zur Gruppenfindung nutzen. Sie wählen gleichzeitig bei der Eintragung in die Gruppen Ihren Termin für das Teamtraining der HDA aus. Alle Gruppenmitglieder müssen sich manuell in die Gruppen eintragen.

Abgabe des Projekts

Das Projekt ist bis zum **15.03.2023 um 23:50 Uhr Serverzeit** auf moodle abzugeben. Sie exportieren das Projekt genauso wie die Hausübungsabgaben als ZIP-Archiv. Dabei gelten auch die gleichen Konventionen wie bei den Hausübungen. Die Benennung erfolgt folgendermaßen: Projektgruppe xxx, wobei der Suffix xxx durch Ihre Gruppennummer zu ersetzen ist. Eine Person aus Ihrer Gruppe gibt das gesamte Projekt ab.

Sie geben neben dem Code zusätzlich eine PDF-Datei ab, die sich ebenfalls im ZIP-Archiv befindet. Diese PDF umfasst die Dokumentationen der weiterführenden Aufgaben und die Lösungen der Theorieaufgaben. Die Dokumentationen sollen es uns als Veranstaltende ermöglichen, Ihre Gedanken zu den Aufgaben zu verstehen.

Verbindliche Anforderung (Für die Abgabe des Projekts**):**

Nachdem eines Ihrer Teammitglieder das Projekt auf moodle hochgeladen hat, müssen alle anderen Teammitglieder diese Aufgabe im entsprechenden Modul bestätigen. Andernfalls wird die Aufgabe nur im Entwurfsmodus gespeichert und nicht als Abgabe gewertet. **Es werden keine Abgaben im Entwurfsmodus akzeptiert. Diese werden nicht bewertet und unabhängig vom Inhalt der Abgabe mit 0 Punkte bewertet.** Geben Sie daher Ihr Projekt nicht kurz vor der Deadline ab. Ihre Teammitglieder müssen genügend Zeit haben, um die Abgabe zu bestätigen.

Plagiarismus und die Nutzung von KI

Selbstverständlich gelten die gleichen Regelungen zum Plagiarismus und zur Nutzung von KI, wie auch bei den Hausübungsabgaben. Daher hier nochmal der Hinweis, dass Ihr gemeinsames Repository für die Gruppenarbeit privat sein muss. Beachten Sie: Tritt der Fall ein, dass Ihre Dokumentation der Lösungswege unvollständig ist oder uns den Anlass für einen Verdacht gibt, dass Sie nicht nur innerhalb der eigenen Gruppe gearbeitet haben. In diesem Fall müssen Sie damit rechnen, dass Sie Ihre Ergebnisse bei einem privaten Testat bei Prof. Weihe persönlich vorstellen und erläutern müssen.

Inhaltliche Information zum Projekt

Im Laufe des Projekts werden Sie eine Implementation des beliebten Brettspiels „Die Siedler von Catan“ erarbeiten und diese ergänzen. Sie erhalten eine Vorlage, in welcher Sie - je nach Aufgabenstellung - Code ergänzen oder erweitern müssen, damit eine lauffähige Version des Brettspiels entsteht. Sie werden viele bekannte Themen aus der Vorlesung im Projekt behandeln und anwenden müssen. Damit stellt das Projekt auch eine gute Vorbereitung auf die Klausur dar. Selbstverständlich wünschen wir Ihnen viel Spaß bei der Implementierung des Brettspiels und wir freuen uns auf die, hoffentlich gelungenen, Implementationen!

Spielekonzept Die Siedler von Catan

Im Brettspiel „Die Siedler von Catan“ erschaffen die Spieler gemeinsam eine Inselwelt. In der Inselwert bauen sie Siedlungen, errichten Straßen und sammeln Rohstoffe. Das Ziel ist es, 10 oder mehr Siegpunkte zu erreichen, indem man Siedlungen zu Städten erweitert und Entwicklungskarten erwirbt.

Spielematerial

- Hexagonfeldern mit unterschiedlichen Geländeformen: Wälder, Hügel, Felder, Berge und Wüsten
- Siedlungen und Städte in verschiedenen Farben für jeden Spieler
- Straßen für den Bau von Wegen zwischen den Siedlungen
- Rohstoffkarten für Ressourcen wie Holz, Lehm, Getreide, Erz und Wolle
- Zwei Würfel zur Generierung von Ressourcen
- Entwicklungskarten mit verschiedenen Funktionen
- Ein Räuber

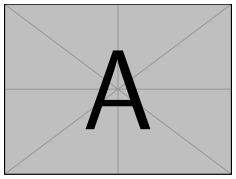

Abbildung 1: Ausschnitt eines Spielfeldes mit bebauten Siedlungen, Straßen und einem Räuber

Spieleablauf

Start

Die Spieler platzieren abwechselnd ihre Siedlungen und Straßen auf der Insel. Jeder Spieler beginnt mit zwei Siedlungen und zwei Straßen.

Aktionen

Die Spieler würfeln nacheinander zu Beginn ihres Zuges. Die gewürfelte Zahl bestimmt, welche Felder Rohstoffe produzieren. Spieler erhalten Ressourcen, wenn die Zahl auf den Feldern gewürfelt wird, die an ihre Siedlungen angrenzen. Danach können die Spieler handeln, Straßen und Siedlungen bauen oder Entwicklungskarten erwerben. Ein Sonderfall tritt auf, wenn eine „7“ gewürfelt wird. Dann gibt jeder Spieler, der mehr als 7 Rohstoffe besitzt die Hälfte davon ab, aber vorher wird jedoch abgerundet. Zusätzlich setzt der Spieler, der gerade am Zug ist, den Räuber auf eine andere Landschaft. Abschließend zieht dieser Spieler bei einem anderen Spieler, der eine Siedlung oder Stadt an dieser Landschaft besitzt, eine Rohstoffkarte aus der verdeckten Hand.

Weiteres müssen Sie außerdem auf bestimmte Bedingungen beim Bau von Strukturen achten, wie Siedlungen oder Straßen. Siedlungen müssen auf einer Kreuzung gebaut werden, zu der mindestens eine eigene Straße führt. Dabei muss die Abstandsregelung beachtet werden. Die Abstandsregelung besagt, dass eine Siedlung nur dann auf eine Kreuzung bebaut werden darf, wenn die 3 angerenzenden Kreuzungen nicht von Siedlungen oder Städten besetzt sind, unabhängig davon wem sie gehören.

Spieler dürfen Straßen nur auf Wegen bauen und auf jedem Weg darf maximal nur eine Straße gebaut werden. Außerdem dürfen Straßen nur an einer Kreuzung anliegen, an der eine eigene Straße, Siedlung oder eine Stadt angrenzt und auf der keine fremde Siedlung oder Stadt steht.

Will oder kann der Spieler in seinem Zug keine Aktion mehr durchführen, erklärt der Spielen seinen Zug für beendet. Der nächste Spieler im Uhrzeigersinn ist dann an der Reihe.

Wertung während des Spiels

- **Siedlungen:** Jede Siedlung bringt 1 Siegpunkt.
- **Städte:** Verbesserte Versionen von Siedlungen, die 2 Siegpunkte bringen.
- **Längste Handelsstraße:** Bonuspunkte für den Spieler mit der längsten durchgehenden Straße (2 Siegpunkte).
- **Größte Rittermacht:** Bonuspunkte für den Spieler mit den meisten Ritterkarten (2 Siegpunkte).

Spielende

Das Spiel endet, wenn ein Spieler mindestens 10 Siegpunkte erreicht. Dann erfolgt die Endwertung:

- **Siegpunkte für Siedlungen und Städte:** 1 Siegpunkt für jede Siedlung, 2 Siegpunkte für jede Stadt.
- **Längste Handelsstraße:** Bonuspunkte (2 Siegpunkte) für den Spieler mit der längsten durchgehenden Straße.
- **Größte Rittermacht:** Bonuspunkte (2 Siegpunkte) für den Spieler mit den meisten Ritterkarten.

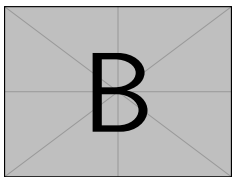

Abbildung 2: Das Beispiel zeigt die Punktezählung für den gelben Spieler. Durch den Besitz der längsten Handelsstraße und drei Siedlungen hat der gelbe Spieler insgesamt 5 Siegpunkte erreicht.

Zusätzliche Informationen

Weitere Informationen sowie die offizielle Spielanleitung erhalten Sie unter den nachfolgenden Links:

- Offiziele Anleitung
- Wikipedia Artikel

- Youtube Video, welches die Spielregeln erläutert

Implementationsstruktur

HexGrid - Das Spielfeld

Das HexGrid repräsentiert das Spielfeld als hexagonale Struktur. Es besteht aus einzelnen Tiles oder Landschaften, welche die Spielumgebung definieren.

Das `HexGrid` an sich ist ein Interface und wird in `HexGridImpl` implementiert.

Das Spielfeld verwendet ein **axiales Koordinatensystem** mit den Achsen q, r und s. Die Koordinaten müssen die Bedingung $q+r+s=0$ erfüllen. Durch dieses System können Tiles auf dem Spielfeld eindeutig lokalisiert werden. Die s-Koordinate wird berechnet als $s=-q-r$.

Die Verwendung von **Spiralringen** ist eine effektive Methode, um systematisch durch die Hexagon-Kacheln zu navigieren. Unser Algorithmus teilt das Spielfeld in Ringe auf und füllt dann die Hexagon-Kacheln in jedem Ring nacheinander. Dies ermöglicht eine geordnete Durchquerung des HexGrids, was für die Implementierung des Setzen der einzelnen Tiles auf dem Spielfeld entscheidend ist. Spiralringe bieten auch die Möglichkeit, die Anzahl der Hexagon-Kacheln in einem größeren Hexagon zu berechnen und können für die Bestimmung von Bewegungsbereichen in Spielen verwendet werden.

Eine äußerst **informative Erklärung** mit anschaulichen Grafiken, die das axiale Koordinatensystem veranschaulichen, steht für eine vertiefte Einsicht unter dem folgenden Link zur Verfügung: Hexagonal Grids. Bei der Umsetzung des Spielfeldes haben wir uns stark von dieser Erklärung inspirieren lassen, um die grundlegenden Konzepte klar und verständlich zu integrieren.

TilePosition

Die Klasse `TilePosition` ist zentral für die Positionierung auf dem Spielbrett im axialen Koordinatensystem. Sie repräsentiert eine Position durch die axialen Koordinaten q und r, wobei die s-Koordinate berechnet wird, um die Bedingung $q+r+s=0$ zu erfüllen.

Mit Funktionen wie `add`, `subtract` und `scale` ermöglicht die Klasse grundlegende Rechenoperationen zwischen Positionen. Eine wichtige Funktion ist `neighbours`, die alle benachbarten Positionen einer gegebenen Position zurückgibt.

Die Richtungen, um eine Position werden durch `EdgeDirection` definiert, wobei Funktionen wie `left` und `right` die Navigation zwischen benachbarten Richtungen erleichtern. Diese Richtungen sind relevant für den Spielaufbau, insbesondere für den Bau von Straßen und Siedlungen. Zusätzlich werden Richtungen in Bezug auf Kreuzungen `IntersectionDirection` definiert, wobei jede Richtung „linke“ und „rechte“ Nachbarrichtungen hat, was beim Aufbau von Straßen und Siedlungen eine Rolle spielt.

Tile

`Tile` ist ein Interface, welches in der Klasse `TileImpl` implementiert wird. Diese fungiert als einer der zentralen Elemente in der Implementierung des Spiels „Die Siedler von Catan“. Jedes Tile repräsentiert einen Baustein des Spielfelds und bietet Zugriff auf verschiedene Eigenschaften und Operationen. Diese umfassen Höhe und Breite als

`ObservableDoubleValues`, den Tile-Typ (`WOODLAND`, `MEADOW`, `FARMLAND`, `HILL`, `MOUNTAIN` und `DESERT`), eine mit dem Tile verbundene Würfelzahl und die Position im axialen Koordinatensystem.

Das Tile ist eng mit dem HexGrid verbunden und ermöglicht es, die Nachbarn eines Tiles abzurufen sowie Informationen zu Kreuzungen `Intersections` und den darauf platzierten Siedlungen und Straßen zu erhalten. Es gibt auch Funktionen zum Platzieren und Abrufen von Straßen und Siedlungen. Es kann auch überprüft werden, ob sich der Räuber derzeit auf dem Tile befindet. Die verschiedenen Tile-Typen sind Farben und Ressourcenarten zugeordnet. Insgesamt bildet die Tile-Klasse die Grundlage für die Implementierung der Spielmechanik von „Die Siedler von Catan“ auf dem HexGrid. Die Tile-Klasse definiert nämlich die Struktur und Beziehungen zwischen den Bausteinen des Spielfelds.

Intersection

`Intersection` ist ein Interface, welches von `IntersectionImpl` implementiert wird, diese repräsentiert die Schnittpunkte auf dem Spielfeld.

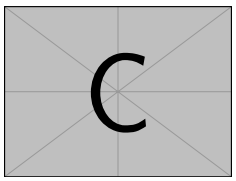

Abbildung 3: Die Abbildung zeigt das Spielfeld. Die roten Punkte stellen die einzelnen Schnittpunkte dar.

Es bietet verschiedene Funktionen zur Interaktion mit den Spielmechaniken. Durch die Klasse können Spieler Siedlungen auf den Kreuzungen platzieren, vorhandene Siedlungen zu Städten verbessern und Handelsplätze (Ports) nutzen. Es bietet auch den Zugriff auf verbundene Straßen, angrenzende Kreuzungen und benachbarte Tiles. Die Funktionen umfassen das Abrufen von Informationen, wie platzierten Siedlungen, Upgrades zu Städten, zugehörigen Häfen und verbundenen Straßen. Das Klasse erleichtert auch den Zugriff auf benachbarte Kreuzungen und Tiles sowie das Prüfen der Verbindung zu bestimmten Positionen.

Road

Die `RoadImpl` ist die Klasse, welches die Methoden aus dem Interface `Road` implementiert, diese fungiert als Bindeglied im `HexGrid` und repräsentiert die Straßen. Zur Erinnerung: Straßen verbinden die verschiedenen Siedlungen und Städte der Spieler. Diese Klasse ermöglicht die Definition von Straßenabschnitten zwischen zwei Positionen auf dem Spielfeld und hält Informationen über den Besitzer der Straße fest. Durch diese Klasse erhalten Spieler Zugriff auf wichtige Funktionen, die die Interaktionen mit benachbarten Straßen und Kreuzungen erleichtern.

Die Methode `getAdjacentTilePositions` gibt die Positionen der benachbarten Tiles der Straße zurück. Die Methode `getIntersections` liefert die Kreuzungen zurück, die mit der Straße verbunden sind. Besonders relevant ist die Fähigkeit, zu prüfen, ob eine Straße mit einer anderen verbunden ist `connectsTo`. Dies ermöglicht es Spielern zu überprüfen, ob alle ihre Straßen miteinander verbunden sind. Zusätzlich erlaubt `getConnectedRoads` den Zugriff auf alle benachbarten Straßen, die mit den anliegenden Kreuzungen verbunden sind.

Player

Die Klasse `PlayerImpl`, welches das Interface `Player` implementiert, bietet einen umfassenden Zugriff auf die verschiedenen Aspekte eines Spielers. Hier erhält man Einblicke in die Ressourcenverwaltung, den Status von Siedlungen und Straßen sowie die Entwicklungskarten des Spielers.

Über diese Klasse können Spieler ihre aktuellen Ressourcen und deren Mengen abfragen, sowie Ressourcen hinzufügen oder entfernen. Die Klasse bietet auch Informationen zu den von einem Spieler kontrollierten Straßen und Siedlungen. Sie bietet auch die Möglichkeit, die verbleibenden Optionen für den Bau von Straßen und Siedlungen zu ermitteln. Die Entwicklungskarten des Spielers, einschließlich ihrer Anzahl und Typen, können über die Schnittstelle verwaltet werden. Die Schnittstelle bietet auch Zugriff auf zusätzliche Informationen wie die Anzahl der gespielten Ritter und die Spielerfarbe.

Implementation des Models

Im Package `projekt.model` sind die oeben genannten Klassen implementiert. Um die Klassenstruktur und die verschiedenen Beziehungen visuell zu veranschaulichen, haben wir ein Diagram mit Hilfe von **Unified Modeling Language (UML)** modelliert. Hilfe zum Lesen eines UML-Diagramms finden Sie im Wikipedia-Artikel. Abbildung 4 zeigt das UML-Diagram der verschiedenen Klassen zur Implementierung des Models vom Spiel.

Design Patterns

Die Struktur und die Implementation des Projekts orientieren sich am Entwurfsmuster Model-View-Controller. Dieses Design Pattern wird im Abschnitt Model-View-Controller des Foliensatzes Fehlersuche und fehlervermeidender Entwurf behandelt. Wie Sie anhand des Aufbaus des Projekts sehen können, unterteilt sich das Projekt in drei wichtige Komponenten. Die drei Komponenten entsprechen drei Packages, welche heißen:

- `projekt.model`
- `projekt.view`
- `projekt.controller`

Das Package `projekt.model` implementiert im Wesentlichen die grundlegenden Daten wie den Aufbau eines Spielers oder des Spielfeldes. Das zweite Package `projekt.view` stellt die Daten der Komponente Model dar und interagiert mit dem Benutzer. Bei Änderungen der View wird die Komponente Controller benachrichtigt. Es werden keine Daten in der Komponente View verarbeitet, sondern nur entgegengenommen. Das Package `projekt.controller` lenkt und verwaltet die beiden anderen Komponenten.

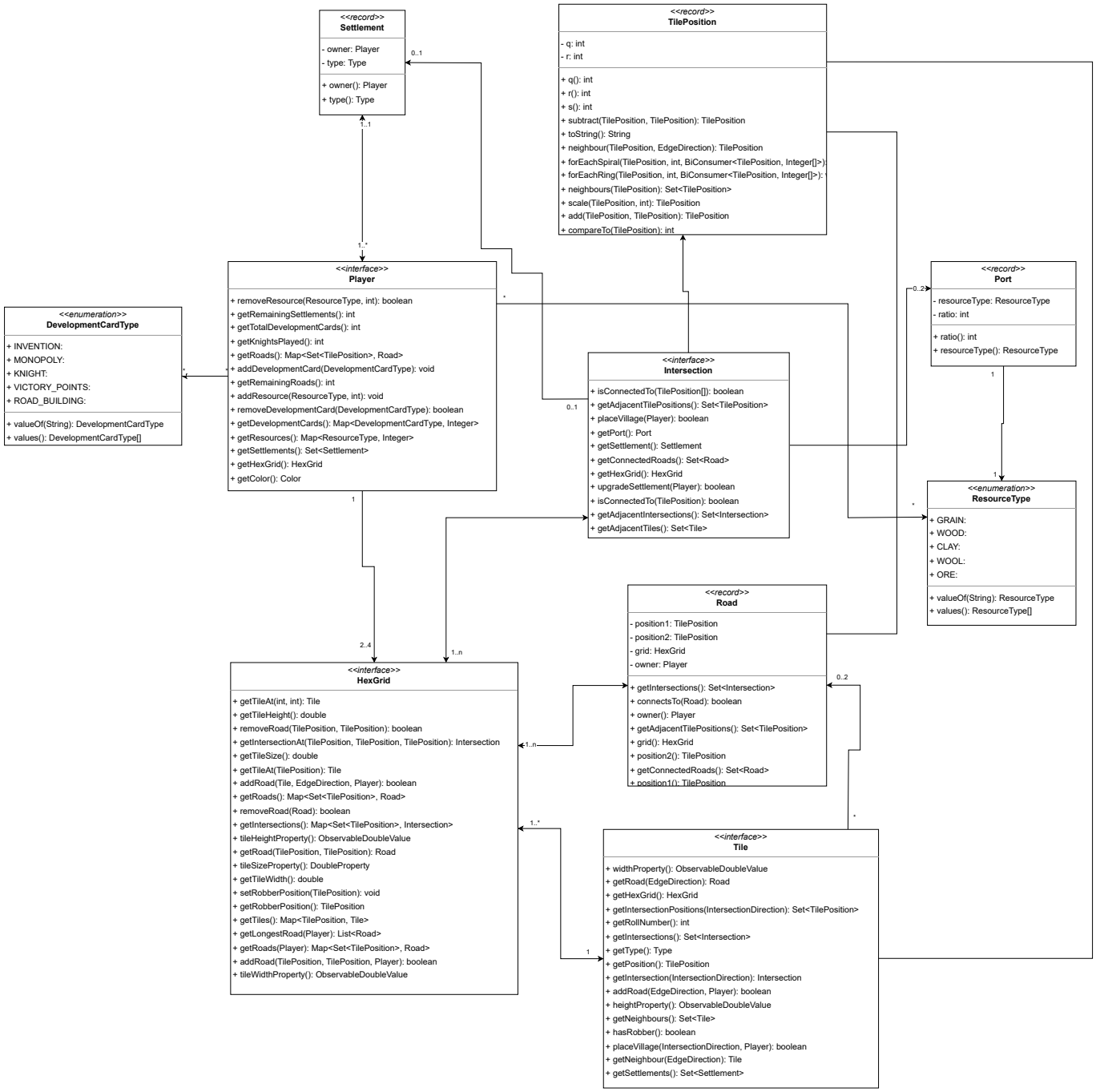

Abbildung 4: Die Abbildung zeigt die Beziehungen der einzelnen Klassen vom model.

Aufgaben

Das Projekt besteht aus zwei verschiedenen Aufgabentypen - Basisaufgaben und weiterführende Aufgaben. In den Basisaufgaben geben wir Ihnen genau vor, welche Funktionalität Sie zu implementieren haben.

In den weiterführenden Aufgaben erweitern Sie das Projekt nach Ihren Vorstellungen. Vergessen Sie nicht, diese gut zu dokumentieren, damit die Projekttutoren Ihre Implementationen nachvollziehen können. Die Aufgaben sind so konzipiert, dass diese Sie schrittweise an eine lauffähige Version des Spiels heranführen. Deshalb wird Ihnen empfohlen die Aufgaben in der nachfolgenden Reihenfolge abzuarbeiten.

H1: Implementierung des Modells -1 Punkte

Das erste Aufgabenset bezieht sich auf die Implementierung des Models. Hierfür fokusieren wir uns zuerst auf den Spieler und dann auf die Bauwerke.

Wie Sie im Einleitungstext bereits gelesen haben, modelliert das Interface `Player` einen Spieler. In der Klasse `PlayerImpl` werden Sie nun dieses Interface implementieren. Damit können Sie die Ressourcen, Siegespunkte und andere Informationen eines Spielers speichern.

H1.1: Inventarsystem

Das Ziel dieser Aufgabe ist es, dass Spieler Ressourcen sammeln, benutzen und handeln können. Sie implementieren im Folgenden die Methoden `getResources`, `addResource`, `removeResource`, `removeResources`, `getTradeRatio` und `hasResources`.
`PlayerImpl` verfügt über eine Objektkonstante *resources* vom Typ `Map<ResourceType, Integer>`. Die Schlüsselwerte dieser Map sind die jeweiligen Ressourcentypen (`WOOD`, `CLAY`, etc.). Einem Schlüsselwert ist die Anzahl (mindestens 0) der entsprechenden Ressource, die dieser Spieler besitzt, zugeordnet.

Die Methode `getResources` soll eine Sicht auf das Inventar des Spielers zurückgeben. Das heißt, es soll nicht möglich sein, anhand des von der Methode zurückgegeben Objekts das Inventar des Spielers zu modifizieren.

Die Methoden `addResource` und `removeResource` werden verwendet, um Ressourcen hinzuzufügen und zu entfernen.
`addResource` erhält als ersten Parameter den Typ der Ressource und als zweiten formalen Parameter die Anzahl, wie viel der Spieler von dieser Ressource zusätzlich erhält. Das Gegenstück, `removeResource`, entfernt entsprechend viele Ressourcen aus dem Inventar. Sollte ein Spieler weniger einer Ressource besitzen, als entfernt werden soll, gibt die Methode `false` zurück und verändert den Zustand des Spielers nicht. Ansonsten entfernt die Methode entsprechend viele Einheiten der Ressource und liefert `true` zurück.

Die Methode `removeResources` funktioniert analog zu `removeResource`, nur mit mehreren Ressourcen als Eingabe. Wichtig ist, dass die Methode den Zustand des Spielers nicht verändert, wenn irgendeine der im Parameter übergebenen Ressourcen nicht in ausreichender Menge vorhanden ist.

`hasResources` hat ebenfalls einen formalen Parameter vom Typ `Map<ResourceType, Integer>`. Diese Methode gibt `true` zurück, falls alle im Parameter übergebenen Ressourcen in ausreichender Menge vorhanden sind Sie gibt `false` zurück, falls das nicht der Fall ist.

Methode `getTradeRatio` gibt das beste Verhältnis für die übergebene Ressource zurück. Sollte der Spieler einen Spezial-Hafen für diese Ressource besitzen, wäre die Rückgabe 2, bei allgemeinen Häfen 3 und ansonsten 4. Eine

Rückgabe von n bedeutet, dass ein Spieler n Einheiten dieser Ressource tauschen kann, um 1 Einheit einer beliebigen anderen Ressource zu erhalten.

H1.2: Entwicklungskarten

Ähnlich der Verwaltung von Ressourcen, braucht ein Spieler Möglichkeiten, um Entwicklungskarten zu sammeln und zu verwenden. In dieser Aufgabe werden die Methoden `getDevelopmentCards`, `addDevelopmentCard`, `removeDevelopmentCard`, `getTotalDevelopmentCards` und `getKnightsPlayed` implementiert. Analog zu *resources*, speichert die Objektkonstante *developmentCards* die Entwicklungskarten des Spielers.

`getDevelopmentCards`, `addDevelopmentCard` und `removeDevelopmentCard` funktionieren analog zu ihren Gegenstücken in H1.1:
`getDevelopmentCards` gibt eine Sicht auf *developmentCards* zurück. `addDevelopmentCard` fügt eine Karte des übergebenen Typs hinzu. `removeDevelopmentCard` entfernt eine Karte des übergebenen Typs, sofern der Spieler mindestens eine Karte dieses Typs besitzt. Die Methode gibt `true` zurück, wenn die Karte entfernt wurde und `false` wenn nicht.

Die Methode `getTotalDevelopmentCards` gibt die Anzahl aller Entwicklungskarten im Besitz des Spielers zurück. Die Methode `getKnightsPlayed` gibt die Anzahl aller Karten vom Typ `KNIGHT` zurück.

H1.3: Alle Wege führen nach ...

Eine Straße hat drei Eigenschaften. Erstens, sie befindet sich immer zwischen zwei – ggf. gedachten – Tiles (siehe `TilePosition`). Zweitens, sie beginnt bei einer Kreuzung (siehe `Intersection`) und endet bei einer anderen Kreuzung. Drittens, sie hat einen Besitzer. Die Reihenfolge der Kreuzungen spielt keine Rolle; die „Startkreuzung“ kann ebenso das Ende sein wie umgekehrt. Sie werden im Folgenden die Methoden `getIntersections`, `connectsTo` und `getConnectedRoads` in `Road` implementieren.

Methode `connectsTo` erhält als Parameter eine andere Straße. Die Methode gibt genau dann `true` zurück, wenn die beiden Straßen über eine gemeinsame Kreuzung miteinander verbunden sind.

`getConnectedRoads` gibt die Menge von Straßen zurück, die an eine der beiden Kreuzungen anliegen.

Neben den Methoden in `Road` selbst, gibt es auch noch weitere Methoden in anderen Klassen, die auf Road aufbauen: Methode `getConnectedRoads` in `IntersectionImpl` gibt die Menge aller `Road`-Objekte zurück, die an die aktuelle Kreuzung angrenzen. `playerHasConnectedRoad` gibt genau dann `true` zurück, wenn der im Parameter übergebene Spieler eine Straße hat, die an diese Kreuzung angrenzt.

In `HexGridImpl` müssen noch die Methoden `getRoads(Player)`, `getLongestRoad` und `addRoad` implementiert werden. Die Map im Attribut *roads* in `HexGridImpl` bildet eine Menge von `TilePosition` auf ein `Road`-Objekt, welches zwischen diesen Positionen liegt, ab.
`getRoads(Player)` soll eine Sicht auf die Straßen des im Parameter spezifizierten Spielers zurückgeben. Wie der Name schon vermuten lässt, ist die Rückgabe von `getLongestRoad` eine Liste von `Road`-Segmenten, welche die längste zusammenhängende Straße des übergebenen Spielers darstellt. Die Methode `addRoad` fügt dem Spielbrett eine neue Straße zwischen den beiden angegebenen `TilePosition`s (erster und zweiter Parameter) hinzu. Die Straße darf nur hinzugefügt werden, wenn an dieser Position noch keine andere Straße gebaut wurde und wenn der platzierende Spieler / Besitzer (dritter Parameter) durch eine Siedlung oder andere Straße angrenzt. Zuletzt gibt die Methode `true` zurück wenn die Straße erfolgreich hinzugefügt wurde und `false` in allen anderen Fällen.

H1.4: Ein Dach über dem Kopf

Jetzt, da Spieler existieren, brauchen diese auch etwas zu tun, damit ihnen nicht langweilig wird. Aus diesem Grund wollen wir die Klassen für Straßen und Siedlungen, `Road` und `Settlement`, vervollständigen.

Siedlungen werden durch die Klasse `Settlement` modelliert. Sie werden unterschieden in Dörfer (`Settlement.Type.VILLAGE`) und Städte (`Settlement.Type.CITY`). Die Klasse `IntersectionImpl` besitzt die zwei Methoden `placeVillage` und `upgradeSettlement`, die Sie nun implementieren werden. Dafür verwenden Sie das Objektattribut *settlement*, welches ein Objekt vom Typ `Settlement` speichert, wenn auf dieser Kreuzung eine Siedlung gebaut wurde.

Die Methode `placeVillage` platziert ein Dorf auf der aktuellen Kreuzung, wenn diese nicht belegt ist und wenn der Spieler am Zug die Kreuzung durch seine Straßen erreichen kann. Sie gibt `true` zurück, wenn eine Siedlung erfolgreich platziert wurde und `false` in allen anderen Fällen. `upgradeVillage` wird verwendet, um existierende Siedlungen in Städte umzuwandeln. Sie ersetzt das gespeicherte `Settlement`-Objekt mit einem entsprechend initialisierten Objekt, wenn auf der aktuellen Kreuzung ein Dorf existiert. Des Weiteren gibt sie `true` zurück, wenn das Dorf erfolgreich ausgebaut wurde und `false` wenn nicht.

H2: Implementierung des Controllers **-1 Punkte**

In H1 haben wir uns mit der Implementation des Modells beschäftigt. Nun machen wir uns an die Implementierung des Controllers dran. Hierfür verwenden wir die von Ihnen vervollständigten Methoden aus dem Modell und nutzen diese für die Methoden aus dem Controller.

H2.1: Die Würfel sind gefallen…, wer bekommt nun Rohstoffe?

Wie im Spielekonzept beschrieben, werden Ressourcen verteilt, wenn der zu würfelnde Spieler keine „7“ geworfen hat. Dann werden im Spiel die Resourcen an diejenigen Spieler verteilt, die auf der jeweiligen Landschaft eine Siedlung oder eine Stadt besitzen.

Implementieren Sie die Methode `distributeResources` für die Verteilung von Ressourcen basierend auf der geworfenen Zahl. Die Methode erwartet den Parameter `int diceRoll`, um das `Tile` zu bestimmen. Ermitteln Sie das entsprechende `Tile` auf dem Spielfeld. Dann überprüfen Sie, ob auf den Kreuzungen `Intersections` dieser Landschaft Siedlungen oder Städte gebaut sind. Falls dies zutrifft, weisen Sie den Spielern, denen diese Strukturen gehören, die entsprechenden Ressourcen mit der Anzahl von 1 zu.

H2.2: Jetzt wird gehandelt…

In dieser Aufgabe soll der Handel zwischen Spielern ermöglicht werden. Sie implementieren dafür die Methode `tradeWithPlayer`.

Die Methode sollte ohne Erlaubnis des anderen Spielers handeln. Der Rückgabewert ist vom Typ `boolean` (entweder `true` oder `false`). Die Methode erwartet die Parameter `Player otherPlayer`, `Map<ResourceType, Integer> offer` für das eigene Angebot und `Map<ResourceType, Integer> request` für die angefragten Ressourcen.

Überprüfen Sie zuerst, ob beide Spieler genügend Ressourcen für den Handel haben. Falls nicht, geben Sie `false` zurück. Andernfalls übertragen Sie die entsprechenden Ressourcen zwischen den Spielern und geben Sie `true` zurück.

H2.3: Oder doch lieber handeln mit der Bank?

Abgesehen vom Handel zwischen Spieler, ist auch der Handel mit der Bank eine Möglichkeit, um an die gewünschten Resourcen zu kommen. Implementieren Sie dafür die Methode `tradeWithBank`. Die Methode erwartet die Parameter

`ResourceType offeredType`, `int offeredAmount` und `ResourceType requestedType`.

Überprüfen Sie zunächst das Handelsverhältnis des Spielers für den angebotenen Ressourcentyp. Falls dieses nicht der angeforderten Menge entspricht, geben Sie `false` zurück. Stellen Sie sicher, dass der Spieler genügend Ressourcen für den Handel hat, andernfalls geben Sie ebenfalls `false` zurück.

H2.4: Baumeisterische Eskapaden: Errichte Dörfer und Straßen

Das Ziel dieser Aufgabe ist es, das Bauen der jeweiligen Strukturen zu ermöglichen.

Sie implementieren dafür die Methoden `canBuildVillage` und `buildVillage` für das Bauen von Siedlungen und die Methoden `canBuildRoad` und `buildRoad` für das Bauen von Straßen.

Die Methode `canBuildVillage` dient als Prüfung, ob es einem Spieler möglich ist, eine Siedlung zu bauen. Diese gibt einen `boolean`-Wert zurück. Das bedeutet, dass der Rückgabewert `true` ist, wenn das Bauen möglich ist, und `false`, wenn nicht. Es gibt zwei Szenarien, in denen der Spieler eine Siedlung bauen kann: entweder das Spiel hat gerade begonnen, und der Spieler darf im Zuge dessen zwei Siedlungen platzieren, oder der Spieler verfügt über die notwendigen Ressourcen zum Bauen.

Die Methode `canBuildRoad` prüft, ob ein Spieler eine Straße bauen darf. Diese soll analog zu `canBuildVillage` implementiert werden.

Nachdem Sie die Methoden geschrieben haben, die prüfen, ob der Spieler im Allgemeinen bauen darf, implementieren Sie nun die Methoden, die für das eigentliche Bauen zuständig sind.

Implementieren Sie nun die Methode `buildVillage`. Diese nimmt als Parameter eine Kreuzung (`Intersection`) und hat den Rückgabetyp `boolean`. Zunächst prüfen Sie, ob der Spieler eine Siedlung bauen darf. Falls dies nicht der Fall ist, geben Sie `false` zurück. Setzen Sie dann eine Siedlung auf die gegebene Kreuzung. Die Bedingung, dass mindestens eine Straße zur jeweiligen Kreuzung führen muss, kann ignoriert werden, wenn der Spieler weniger als 2 Siedlungen besitzt. Die Methode soll `false` zurückgeben, wenn bereits eine Siedlung auf der Kreuzung gebaut wurde. Abschließend prüfen Sie, wie in `canBuildVillage`, ob der Spieler im Zuge des Spielstarts an der Reihe ist, zwei Siedlungen zu setzen. Wenn das nicht der Fall ist, ziehen Sie die benötigten Ressourcen für den Bau der Siedlung vom Spieler ab.

Jetzt fehlen nur noch die Straßen. Dafür implementieren Sie die Methode `buildRoad`. Diese nimmt als Parameter zwei `TilePosition` und hat ebenfalls den Rückgabetyp `boolean`. Zunächst prüfen Sie, ob der Spieler eine Straße bauen darf. Falls das nicht zutrifft, geben Sie `false` zurück. Beim Setzen der Straße achten Sie darauf, dass die Prüfung auf eine angrenzende Siedlung nur beim Start des Spiels gemacht werden soll, wenn der Spieler seine 2 Straßen auf dem Spielfeld platziert. Abschließend soll auch hier geprüft werden, ob das Spiel sich im Startzustand befindet, in dem der Spieler 2 Straßen platzieren soll. Falls dies nicht der Fall ist, sollen die benötigten Ressourcen für den Bau einer Straße vom Spieler entfernt werden.

H2.5: Urbanisierung: Vom beschaulichen Dorf zur aufstrebenden Stadt

Spieler haben die Möglichkeit, ihre Siedlung in eine Stadt zu entwickeln.

Implementieren Sie dafür die Methode `upgradeVillage`. Die Methode nimmt als Parameter eine Kreuzung (`Intersection`) an und ist vom Typ `boolean`. Geben Sie `false` zurück, wenn der Spieler nicht über die erforderlichen Ressourcen verfügt oder die Entwicklung zur Stadt nicht möglich ist.

Falls alle Bedingungen erfüllt sind, ziehen Sie die benötigten Ressourcen vom Spieler ab.

H3: Implementierung der View -1 Punkte

H4: Weiterführende Aufgaben -1 Punkte