Software Engineering 1
Abgabedokument
Teilaufgabe 1
(Anforderungsanalyse und Planungsphase)

Persönliche Daten, bitte vollständig ausfüllen:

Nachname, Vorname:	Nikita Antonov
Matrikelnummer:	a01348746
E-Mail-Adresse:	a01348746@unet.univie.ac.at
Datum:	26.03.19

Aufgabe 1: Anforderungsanalyse (2 Punkte)

Analysieren der Spielidee und des Netzwerkprotokolls um 10 Anforderungen (bestehend zumindest aus 3 funktionalen, 3 nichtfunktionalen und einer Designbedingung) nach den folgenden Kriterien zu dokumentieren.

Anforderung 1

- **Beschreibung:** Geschwindigkeit – jede Aktion muss schnell genug realisiert werden(weniger als 3 Sekunden)
- **Bezugsquelle:** eine KI für jede Aktion (Spielerbewegung, Kartengenerierung etc.) nicht mehr als 3 Sekunden Bedenkzeit zur Berechnung erhält.
- **Typ der Anforderung:** nicht funktional

Anforderung 2

- **Beschreibung:** Spielstatus – Client muss immer Spielstatus kennen um zu wissen, ob es möglich jetzt Aktion durchzuführen oder nicht.
- **Bezugsquelle:** Entsprechend müssen Sie immer mit den Statusnachrichten ermitteln ob Ihr Client überhaupt derzeit eine Aktion absetzen darf (wichtig für die Übertragung der Karten und der Bewegungen).
- **Typ der Anforderung:** funktional

Anforderung 3

- **Beschreibung:** GIT – als Entwickler muss ich Source-Code Manager verwenden.
- **Bezugsquelle:** Source-Code-Management Lösungen wie GIT oder SVN sind Standard im Alltag von Softwareentwicklungsprojekten und müssen daher auch von Ihnen hier eingesetzt werden.
- **Typ der Anforderung:** Designbedingung

Anforderung 4

- **Beschreibung:** Registrierung – Spielern müssen sich selbst registrieren.
- **Bezugsquelle:** Die Clients können sich für das neue Spiel registrieren. Daher es ist vorgesehen, dass die SpielID einmalig durch einen Menschen erzeugt wird (z.B. indem über einem Webbrowser auf den Endpoint zu Erstellung eines neuen Spiels zugegriffen wird) und dann beispielsweise die Clients mit der SpielID als Startparameter gestartet werden.
- **Typ der Anforderung:** funktional

Anforderung 5

- **Beschreibung:** Nachrichtenaustausch – Nachrichten müssen mittels HTTP-Protokoll ausgetauscht werden.
- **Bezugsquelle:** Die technologische Basis des Nachrichtenaustauschs stellt eine Restschnittstelle dar, daher es wird das HTTP Protokoll verwendet sowie die zugehörigen Operationen GET und POST.
- **Typ der Anforderung:** designbedingung

Anforderung 6

- **Beschreibung:** Zuverlässigkeit – ich möchte in meinem Programm die Fehler sehen und beheben.
- **Bezugsquelle:** Daher im Fehlerfall sind `exceptionName` und `exceptionMessage` mit Details zum Fehler befüllt sowie das `state` Element mit dem Text **Error** definiert. Sollte kein Fehler vorliegen ist `state` mit dem Wert **Okay** definiert.
- **Typ der Anforderung:** funktional

Anforderung 7

- **Beschreibung:** Server überlastung verhindern – ich muss der Zeitspanne zwischen Abfragen erhalten.
- **Bezugsquelle:** Um zu verhindern, dass der Server überlastet wird sollte zwischen zwei vom gleichen Client durchgeführten Abfragen zum Spielstatus mindestens eine Zeitspanne von 0,4 Sekunden vergehen.
- **Typ der Anforderung:** nicht funktional

Anforderung 8

- **Beschreibung:** XML – Nachrichtenaustausch muss in XML-Format erfolgen
- **Bezugsquelle:** Die ausgetauschten Daten bzw. Nachrichten werden im XML Format definiert bzw. erwartet.
- **Typ der Anforderung:** designbedingung

Anforderung 9

- **Beschreibung:** Karte – jede Client-System generiert die Kartenhälfte
- **Bezugsquelle:** Im Anschluss an die Registrierung eines Spielers muss der Client eine Kartenhälfte generieren und an den Server übertragen.
- **Typ der Anforderung:** funktional

Anforderung 10

- **Beschreibung** Zustand zu halten – weil HTTP ist zustandslos, müssen wir die SpielID und SpielerID explizit bestimmen.
- **Bezugsquelle:** Hierbei ist der Teil `<SpielID>` mit der während der Spielerzeugung erhaltenen eindeutigen `SpielID` zu ersetzen und `<SpielerID>` mit der während der Registrierung vom Server mitgeteilten eindeutigen `SpielerID`.
- **Typ der Anforderung:** funktional

Anforderung 11

- **Beschreibung:** Die Eigenschaften von Avatar-Bewegung – Avatar kann sich nur unter 180 oder 90 Grad bewegen und nicht diagonal
- **Bezugsquelle:** Daher der Avatar kann sich nach oben (**Up**), links (**Left**), rechts (**Right**), sowie unten (**Down**) bewegen.
- **Typ der Anforderung:** nicht funktional

Anforderung 12

- **Beschreibung:** Rundenbasiert – Aktionen passieren nacheinander
- **Bezugsquelle:** Das Spiel rundenbasiert abläuft (daher setzt jede KI abwechselnd eine Spielaktion).
- **Typ der Anforderung:** nicht funktional

Aufgabe 2: Anforderungsdokumentation (3 Punkte)

Dokumentation von drei in Teilaufgabe 1 identifizierten Anforderungen nach dem vorgegebenen Schema.

Dokumentation Anforderung 1

- **Name:** Zustand zu halten
- **Beschreibung und Priorität:** System muss die eindeutige ID für Spiel und Spieler erstellen um ihre Zustände zu speichern bei jeder Iteration. Andernfalls kann das System verschiedene Spielen und Spielern nicht unterscheiden, weil HTTP Zustandslos ist(ähnlich wie Cookies).
  Priorität: Hoch
- **Impuls/Ergebnis Listen:**
  Impuls: Client sendet an Server die Anfrage zum Erstellen eines neuen Spiels.
  Ergebnis: Der Server antwortet mit einer Nachricht, die eine eindeutige SpielID beinhaltet und speichert diese ID.
  Impuls: Client sendet an Server die Anfrage um sich zu registrieren mit Vorname, Nachname und Matrikelnummer im Body der Nachricht.
  Ergebnis: Der Server antwortet mit einer Nachricht, die eine eindeutige SpielerID beinhaltet und speichert diese ID.
  Impuls: Client verwendet dann diese Daten um Zustand abzufragen
  Ergebnis: Server antwortet mit der Information über Zustand
- **Benutzergeschichten:**
  Als Client möchte ich bei die Anfragen mich und aktuelles Spiel zu bezeichnen um die Möglichkeit Spielstatus anzufragen zu haben.
  Als Server möchte ich erstellte Spiel, Spielern und Spielstatus-Veränderungen eindeutig zu speichern um ermöglichen es zu identifizieren.
- **Benutzerschnittstelle:** Weil habe ich KI in meinem Spiel, so steuert Spieler das Spiel nicht manuell. Aber Mensch muss Spiel manuell erstellen(durch HTTP GET) und dann die Daten(Vorname, Nachname und Matrikelnummer) als Argumenten zu übergeben um sich als Spieler zu registrieren.
- **Externe Schnittstellen:** Als Schnittstelle um eine Verbindung zur Server zu schaffen und notwendige Ressourcen zu identifizieren, verwendet das System REST und deswegen URLs als Adressen:
  1. **http://<domain>:<port>/game/new** Um ein neues Spiel zu erstellen
  2. **http://<domain>:<port>/game/<SpielID>/register** Um Spieler zu registrieren
  3. **http://<domain>:<port>/game/<SpielID>/state/<SpielerID>** Um Zustand abzufragen

Dokumentation Anforderung 2

- **Name:** Karte – jede Client-System generiert die Kartenhälfte
- **Beschreibung und Priorität:** Zwei Clienten erstellen die Kartenhälfte selbst und verbinden diese Teile in einer Karte. Server generiert die Karte entsprechend nicht.
  Priorität: Hoch
- **Impuls/Ergebnis Listen:**
  Impuls: Client sendet die Nachricht mit Kartenhälfte als Body zum Server
  Ergebnis: Server erstellt die ganze Karte, speichert es und sendet state(Okay oder Error) als Antwort.

- **Benutzergeschichten:**
  Als Client möchte ich meine Kartenhälfte selbst generieren um das später zu Server zu senden und so die Belastung zu verteilen.
  Als Server möchte ich die schon generierte Kartenhälfte von Clienten bekommen um die ganze Karte zu schaffen und so die Belastung zu reduzieren.
- **Benutzerschnittstelle:** Das System generiert die Karte mittels Regeln automatisch und ohne Spieler Involvierung. Aber während des Spiels sieht er die Karte in CLI. Z.B.(Wenn G für Wiese, B für Berg, W für Wasser):
  GGGGGGGG
  GGGBGWWG
  GGGBGWWG
  WWWGBBGG
  WWWGBGGG
  GGGGBGGGG
  GWWGBBGG
  GGGGGGGG
- **Externe Schnittstellen:** Als Schnittstelle um eine Verbindung zur Server zu schaffen und notwendige Ressourcen zu identifizieren, verwendet das System REST und deswegen URLs als Adressen:
  1. **http://<domain>:<port>/game/<SpielID>/halfmap** Um Kartenhälfte zum Server zu tragen
  2. Dann bekommt er einfach XML-Antwort mit dem entsprechenden Status

Dokumentation Anforderung 3

- **Name:** Rundenbasiert – Aktionen passieren nacheinander
- **Beschreibung und Priorität:** Client muss immer Status anfragen(loop) um zu wissen, wenn kann er Bewegung übermitteln. Er kann nur eine Bewegung in eine Runde machen.
- **Impuls/Ergebnis Listen:**
  Impuls: Spielstatus ermitteln bis wäre es möglich die Bewegung durchzuführen.
  Ergebnis: Server antwortet mit Spielstatus(ist es möglich die Bewegung durchzuführen oder nicht)
  Impuls: Bewegung übermitteln
  Ergebnis: Server bestätigt der Empfang
- **Benutzergeschichten:**
  Als Client möchte ich meine Bewegung korrekt, nach einem anderen Spielern und möglichst schnell durchzuführen um das effizient zu machen.
  Als Server möchte ich die Bewegungen von Clienten nacheinander zu bekommen und bearbeiten um die Reihenfolge und Korrektheit zu halten.
- **Benutzerschnittstelle:** Auf die CLI Benutzeroberfläche gibt es kaum Einfluss, weil sind diese Bewegung von KI automatisch gesteuert.
- **Externe Schnittstellen:** Als Schnittstelle um eine Verbindung zur Server zu schaffen und notwendige Ressourcen zu identifizieren, verwendet das System REST und deswegen URLs als Adressen:
  1. **http://<domain>:<port>/game/<SpielID>/state/<SpielerID>** Um Spielstatus zu bekommen
  2. **http://<domain>:<port>/game/<SpielID>/move** Um die Spielbewegung zu Übertragen, wenn gibt es solche Möglichkeit.

Aufgabe 3: Definition von Meilensteinen und Aufwandsabschätzung (Projektplanung) (3 Punkte)

- **Meilenstein:** Grobe Rahmen implementieren
  - Arbeitspaket 1: Pakette in Eclipse für MVC erstellen
    - Aufwand: 1/2 Stunde
  - Arbeitspaket 2: Classenrahmen erstellen
    - Aufwand 1/2 Stunde
  - Arbeitspaket 3: Andere Grundlegende Patterns implementieren(wie z.B. observer)
    - Aufwand 1 Stunde
  - Arbeitspaket 4: Die Grundlegende Methoden innerhalb Classen schaffen(get/set, für Observer notwendige usw)
    - Aufwand ½ Stunde
  - Arbeitspaket 5: Die Grundelegende Bezihungen zwischen Classen schaffen und Testen
    - Aufwand ½ Stunde

- **Meilenstein:** Die Classen vollständig schaffen und lokale Verbindungen bauen
  - Arbeitspaket 1: Kartenhälfte Generierung machen und testen
    - Aufwand 2 Stunden
  - Arbeitspaket 2: Classen und Algorithem für Spielern(Avatar) machen
    - Aufwand ½ Stunde
  - Arbeitspaket 3: Bewegungsalgorithm und KI dazu schaffen
    - Aufwand 4 Stunden
  - Arbeitspaket 4: Lokale Verbindungen zwischen diese Classen vollständig bauen
    - Aufwand 2 Stunden
  - Arbeitspaket 5: Controller und View lokal einstimmen
    - Aufwand 1.5 Stunde

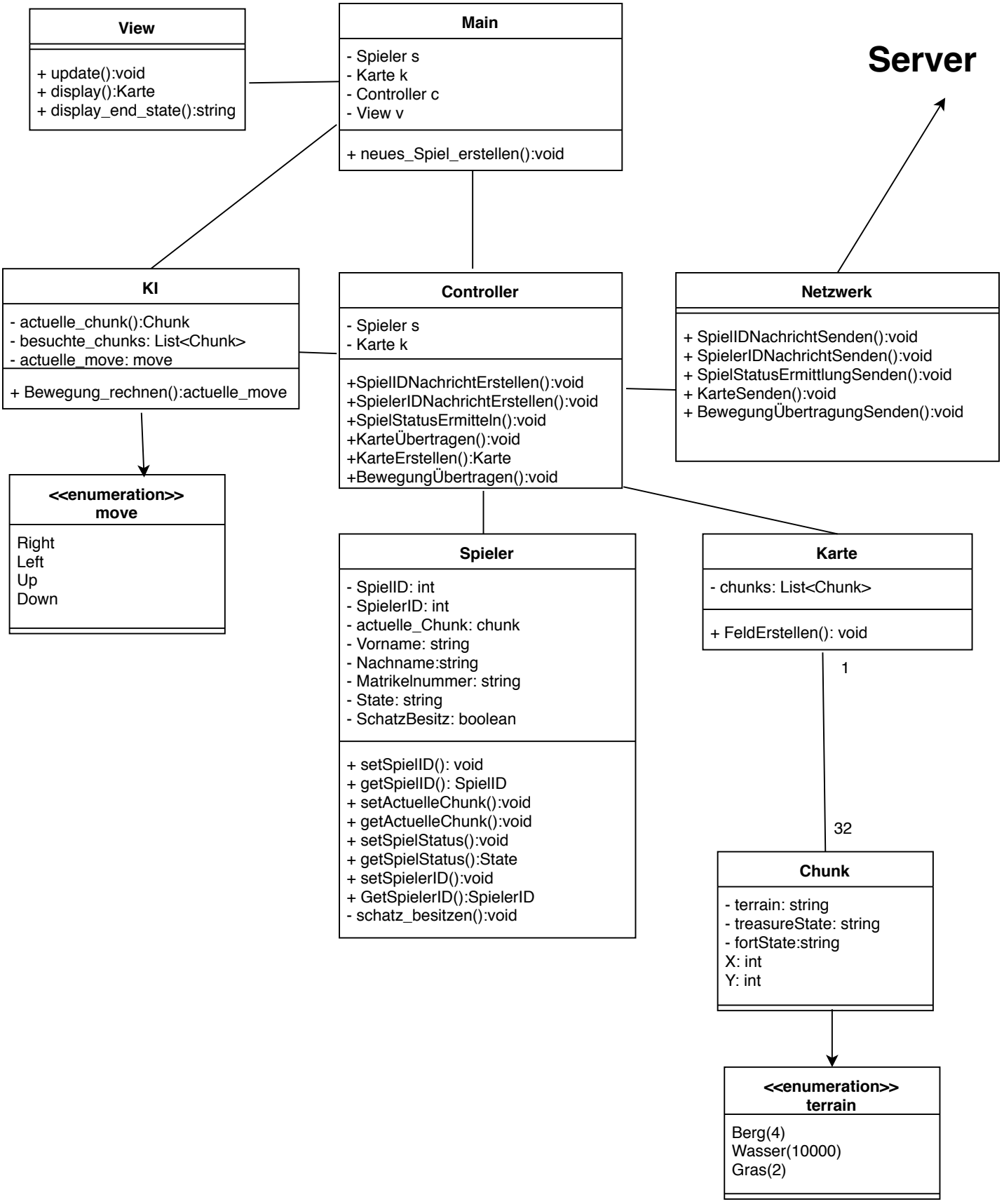
- **Meilenstein:** Netzwerkschnittstelle
  - Arbeitspaket 1: Netzwerkclasse bauen
    - Aufwand 2 Stunden
  - Arbeitspaket 2: Erstellung eines neuen Spiels konfigurieren
    - Aufwand 1 Stunde
  - Arbeitspaket 3: Registrierung konfigurieren
    - Aufwand 1 Stunde
  - Arbeitspaket 4: Übertragung einer Kartenhälfte konfigurieren
    - Aufwand 1.5 Stunde
  - Arbeitspaket 5: Abfrage des Spielstatus konfigurieren
    - Aufwand 1 Stunde
  - Arbeitspaket 6: Übertragung einer Spielbewegung konfigurieren
    - Aufwand 1 Stunde

- **Meilenstein:** User Interface(in CLI) konfigurieren
  - Arbeitspaket 1: Darstellung der Karte schaffen
    - Aufwand ½ Stunde
  - Arbeitspaket 2: Darstellung von Avataren schaffen
    - Aufwand 1 Stunde
  - Arbeitspaket 3: Darstellungen von Bewegungen schaffen
    - Aufwand 1.5 Stunde
  - Arbeitspaket 4: Diese Darstellungen miteinander bauen und testen
    - Aufwand 2 Stunden

Aufgabe 4: Architektur entwerfen, modellieren und validieren (7 Punkte)

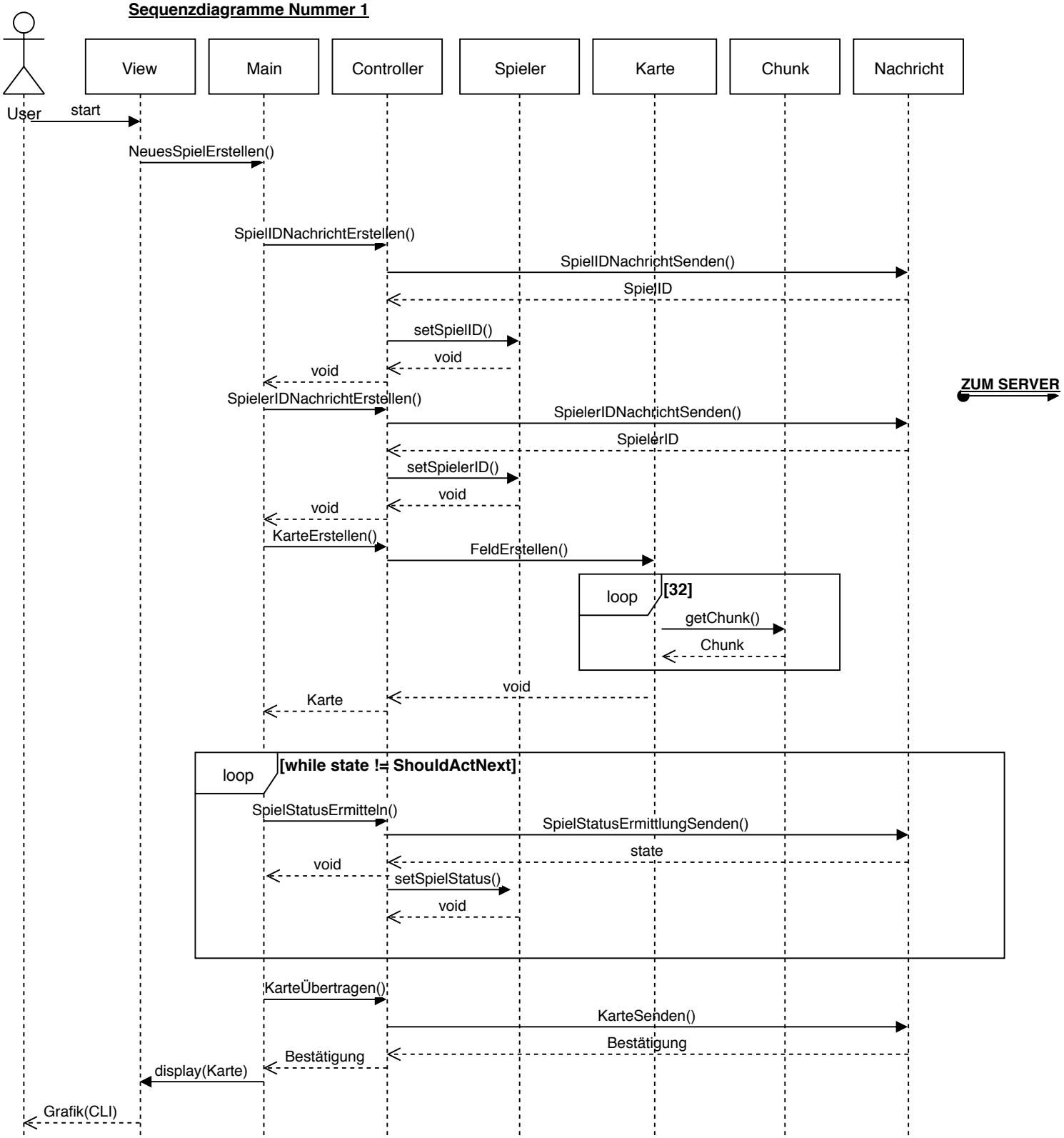
Hier habe ich meine Sequenzdiagramme und Classendiagramm. Meine Hauptaufgabe war die wichtigste Zusammenhänge zu zeigen. Vor allem habe ich der Architektur modelliert. Um das gut lesbar und verständlich zu machen, habe ich nicht die volle Beschreibung von alle Methoden mit allen passenden Argumenten. Zum Beispiel habe ich nicht immer get und set Methoden für alle Classenvariablen beschrieben, aber nur solche Methoden, die habe ich in meinem Sequenzdiagramm verwendet. Diese Diagramme dienen vor allem um die ganze Logik und Architektur zu verstehen und nicht als einer große Liste von alle mögliche und vielleicht sogar nicht notwendige Argumenten und Methoden.

Klassendiagramm



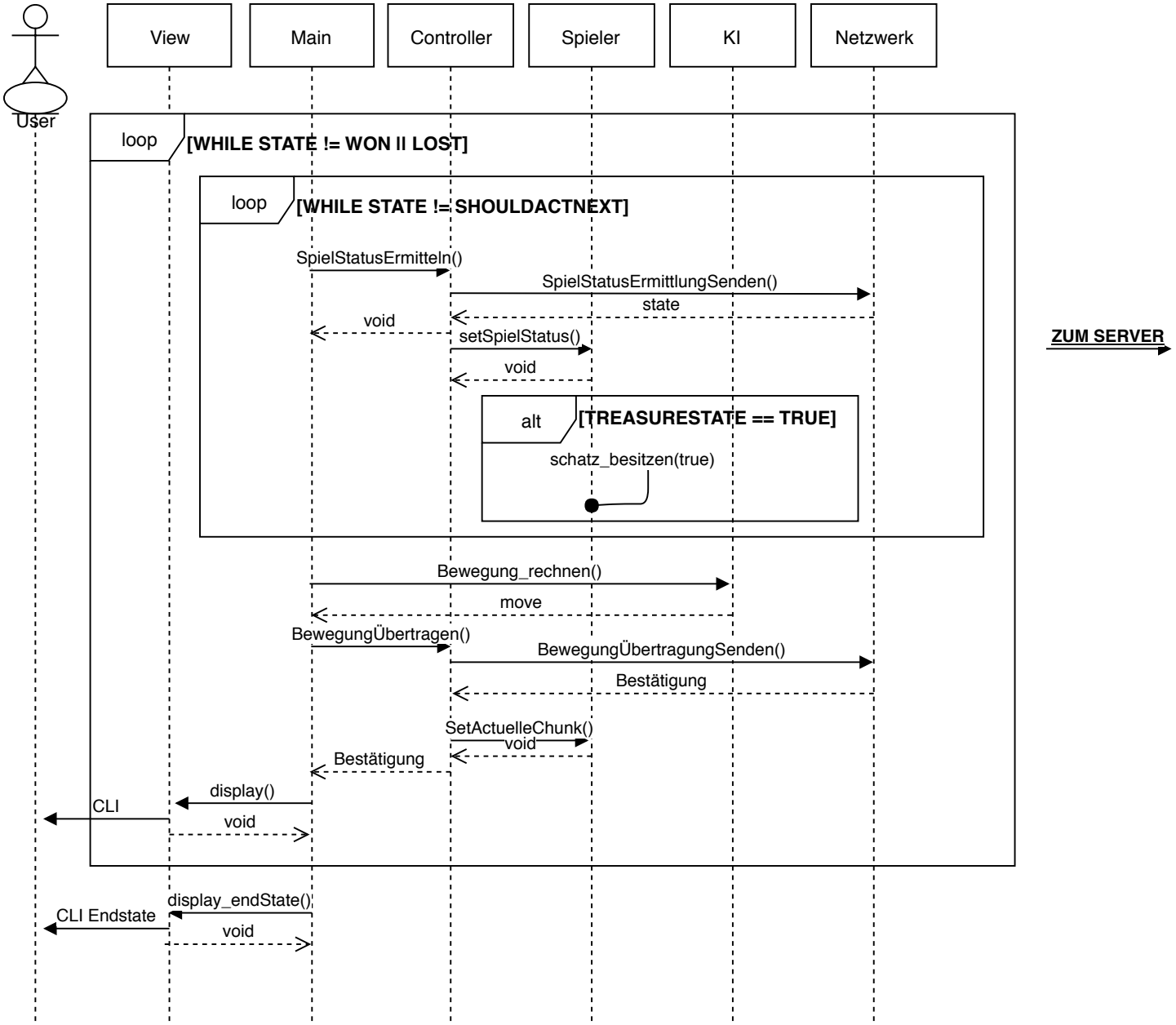
Ich habe hier nicht alle mögliche Methoden mit allen möglichen Argumenten, sondern nur die, die für gesamter Architektur und Logik wichtig sind

Sequenzdiagramme Nummer 1



Ich habe  **nicht verwendet, um das Modell gut lesbar zu machen. Für die meisten Klassen ist es aktiv während des gesamten Prozess.**

Sequenzdiagramm 2



Ich habe  **nicht verwendet, um das Modell gut lesbar zu machen. Für die meisten Klassen ist es aktiv während des gesamten Prozess.**