

Software Engineering 1

Abgabedokument

Teilaufgabe 1

(Anforderungsanalyse und Planungsphase)

Persönliche Daten, bitte vollständig ausfüllen:

Nachname, Vorname:	Boghean, Adrian
Matrikelnummer:	11742914
E-Mail-Adresse:	a11742914@unet.univie.ac.at
Datum:	29.10.2022

Aufgabe 1: Anforderungsanalyse (2 Punkte)

Typ der Anforderung: funktional

Anforderung 1

- **Beschreibung:** Registrierung des Clients – Nachdem ein Spiel erstellt wurde, kann der Client seine KI registrieren, damit es an einem Spiel teilnehmen kann.
- **Bezugsquelle:** Netzwerkprotokoll, „**Sobald ein Spiel erstellt wurde und beiden Clients die eindeutige SpiellID (siehe oben) bekannt ist können sich die Clients für das neue Spiel registrieren.**“
- **Bezugsquelle:** Spielidee, „**Nach Start des Clients registrieren sich die KIs für das Spiel am Server.**“

Anforderung 2

- **Beschreibung:** Kartenerstellung – Nach der Registrierung muss der Client eine zufällig generierte Kartenhälfte erstellen, damit der Server eine komplette Karte erstellen kann.
- **Bezugsquelle:** Spielidee, „**Hierzu erstellt jede der beiden KIs zufällig eine Hälfte der finalen Spielkarte (mit je 5 x 10 Feldern).**“
- **Bezugsquelle:** Netzwerkprotokoll, „**Im Anschluss an die Registrierung eines Spielers muss der Client eine Kartenhälfte generieren und an den Server übertragen.**“

Anforderung 3

- **Beschreibung:** Platzierung des Schatzes – Nachdem die Clients die Kartenhälften gesendet haben, muss der Server den Schatz auf der Kartenhälfte jedes Clients "verstecken", damit die KIs nach dem Schatz suchen können.
- **Bezugsquelle:** Spielidee, „**Hierzu wird je ein Schatz vom Server auf jeder der beiden Kartenhälften versteckt. Ein Schatz ist hierbei jeweils der KI zugeordnet, die auf dieser Kartenhälfte startet und kann nur von dieser "aufgenommen/gesehen" werden.**“

Anforderung 4

- **Beschreibung:** Bewegung der KI – Während des Spiels muss sich der Client auf der Karte bewegen, damit die KI das Spiel nicht verliert.
- **Bezugsquelle:** Spielidee, „**Die Karten sind in Felder aufgeteilt, zwischen denen sich die Spielfiguren, auf Anweisung der KIs, schrittweise waagrecht und senkrecht bewegen.**“
- **Bezugsquelle:** Spielidee, „**Eine Spielfigur kann sich nur horizontal und vertikal zu direkt benachbarten Feldern bewegen, das Überspringen von Feldern ist nicht möglich.**“

Typ der Anforderung: nicht funktional

Anforderung 5

- **Beschreibung:** Erforderliche Felder auf der Kartenhälfte – Nach dem der Client sich registriert hat, muss der Client eine Kartenhälfte erstellen die mindestens 5 Bergfelder, 24 Wiesenfelder und 7 Wasserfelder, damit wird sie vom Server akzeptiert.
- **Bezugsquelle:** Spielidee, „Jede Kartenhälfte muss mindestens 5 Bergfelder, 24 Wiesenfelder, 7 Wasserfelder und eine korrekt platzierte Burg beinhalten.“
- **Bezugsquelle:** Spielidee, „Diese und andere in diesem Dokument genannte Bedingungen/Regeln sind vom Client einzuhalten und vom Server zu überprüfen.“

Anforderung 6

- **Beschreibung:** Maximale Spielrunden – Wenn das Spiel mehr als 320 Runden hat, wird der Server das Spiel beenden, damit die Spiele für die Zuschauer spannend bleiben.
- **Bezugsquelle:** Spielidee, „Um die Spiele für die Zuschauer spannend zu gestalten, wurde festgelegt, dass ein Spiel insgesamt nicht länger als 320 Spielaktionen (und damit 320 Runden) dauern darf.“

Anforderung 7

- **Beschreibung:** Maximale Bedenkzeit – In jeder Runde muss der Client in weniger als 5 Sekunden eine Entscheidung treffen, damit er nicht automatisch verliert.
- **Bezugsquelle:** Spielidee, „Für jede dieser rundenbasierten Spielaktion hat die KI maximal 5 Sekunden Bedenkzeit (insgesamt dauert ein Spiel maximal 10 Minuten). Sollten diese Bedingungen nicht erfüllt werden, verliert die KI, welche gerade an der Reihe ist, das Spiel automatisch und der zugehörige menschliche Spieler bekommt dies vom Client mitgeteilt.“

Anforderung 8

- **Beschreibung:** Intervall zwischen Anfragen – Während des Spiels muss der Client 0,4 Sekunden warten, bevor er eine weitere Anfrage an den Server senden kann, damit der Server nicht überlastet wird.
- **Bezugsquelle:** Netzwerkprotokoll, „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: Designbedingung

Anforderung 9

- **Beschreibung:** Technologische Basis des Nachrichtenaustauschs – Um die Kommunikation zwischen Client und Server während des Spiels zu ermöglichen, wird das HTTP-Protokoll verwendet sowie die zugehörigen Operationen GET und POST. So sind sich der Client und der Server über ihre Aktionen bewusst.
- **Bezugsquelle:** Netzwerkprotokoll, „**Im Folgenden werden die Nachrichten und technischen Hintergründe des Protokolls beschrieben. Die technologische Basis des Nachrichtenaustauschs stellt eine Restschnittstelle dar, daher es wird das HTTP Protokoll verwendet sowie die zugehörigen Operationen GET und POST.**“

Aufgabe 2: Anforderungsdokumentation (2 Punkte)

Dokumentation Anforderung

- **Name:** Bewegung der KI
- **Beschreibung und Priorität:** Nachdem der Client eine Kartenhälfte gesendet hat und der Server die gesamte Karte zurückgeschickt hat, muss der Client warten, bis er an der Reihe ist. Wenn der Client an der Reihe ist, muss der Client eine Bewegung senden. Je nachdem, wohin die KI die Spielfigur bewegen will, muss der Client die gleiche Bewegung eine bestimmte Anzahl wiederholen. Zum Beispiel sind zwischen zwei Wiesenfeldern 2 Bewegungen in dieselbe Richtung erforderlich.
 - Priorität der Anforderung als: **Hoch**
- **Relevante Anforderungen:**
 - **9:** Technologische Basis des Nachrichtenaustauschs – Der Client muss die Bewegung durch Senden einer HTTP POST Request mitteilen.
 - **2:** Kartenerstellung – Zu Beginn muss sich die KI auf den Feldern bewegen, die von demselben Client generiert werden. Die Startposition ist seine eigene Burg, und um auf ein anderes Feld zu bewegen, muss der Client je nach Feld die richtige Anzahl von Bewegungen senden.
 - **7:** Maximale Bedenkzeit – Eine Bewegung muss in weniger als 5 Sekunden erfolgen.
 - **6:** Maximale Spielrunden – Es sind maximal 320 Bewegungen pro Spiel erlaubt.
- **Relevante Business Rules:**
 - Das Spiel ist rundenbasiert, in jeder Runde kann jede KI nur eine Bewegung senden.
 - Die Spielfigur kann sich nur horizontal oder vertikal bewegen.
 - Die KI kann sich nicht auf ein Wasserfeld bewegen. Wenn dies passiert, hat die KI automatisch verloren.
 - Die KI kann sich nicht außerhalb der Karte bewegen. Wenn die KI beschließt, sich über den Rand der Karte zu bewegen, verliert sie automatisch.
 - Die KI kann nicht über Felder springen und kann sich auch nicht diagonal bewegen.
 - Die Anzahl der Bewegungen, die erforderlich sind, um von einem Feld zu einem anderen zu gehen, hängt von der Art des Feldes ab.
 - Um ein Wiesenfeld zu verlassen, ist eine Bewegung erforderlich, und um ein Wiesenfeld zu betreten, ist eine weitere Bewegung erforderlich. Wiesenfeld nach Wiesenfeld 2 Bewegungen erforderlich.
 - Um ein Berg zu verlassen, sind 2 Bewegungen erforderlich. Um einen Berg zu betreten, sind weitere 2 Bewegungen erforderlich.
 - Um ins Wasserfeld zu gehen, ist nur eine Bewegung erforderlich.
 - Die Richtung einer Bewegung kann jederzeit geändert werden. Dadurch wird die Anzahl der Bewegungen, die zum Verlassen und Betreten eines Feldes erforderlich sind, zurückgesetzt.
 - Der Client kann nur dann eine Bewegung senden, wenn er selbst an der Reihe ist.

• **Impuls/Ergebnis - Typisches Szenario:**

Vorbedingungen:

- o Ein menschlicher Spieler hat ein Spiel erstellt.
- o Der Server teilt dem menschlichen Spieler eine SpielID mit.
- o Der menschliche Spieler fügt die SpielID in den Client ein.
- o Zwei Clients registrieren sich mit der gleichen SpielID.
- o Die Clients senden jeweils eine gültige Kartenhälfte.
- o Der Server sendet an jeden Client eine komplette Karte zurück.

Hauptsächlicher Ablauf:

- o Impuls: Ein Client fragt den Server nach dem Spielstatus.
- o Ergebnis: Der Server antwortet mit allen Informationen über den Spielstatus.
- o Impuls: Der Client findet heraus, dass er an der Reihe ist.
- o Ergebnis: Der Client denkt über die nächste Bewegung nach.
- o Impuls: Der Client hat entschieden, welches die nächste Bewegung ist.
- o Ergebnis: Der Client sendet die Bewegung an den Server.

Nachbedingungen:

- o Der Server empfängt eine gültige Bewegung.
- o Der Server hat die Position der KI auf der Karte aktualisiert.
- o Der Server prüft, ob die Gewinnbedingungen erfüllt sind.
- o Sie sind noch nicht erfüllt, das Spiel muss weitergehen.
- o Der Server sendet eine „Okay.“ Bestätigung an den Client zurück.
- o Jetzt ist der zweite Client an der Reihe.

• **Impuls/Ergebnis - Alternativszenario:** Client 1 ist eine Bewegung davon entfernt, das Spiel zu gewinnen.

Vorbedingungen:

- o Ein menschlicher Spieler hat ein Spiel erstellt.
- o Der Server teilt dem menschlichen Spieler eine SpielID mit.
- o Der menschliche Spieler fügt die SpielID in den Client ein.
- o Zwei Clients registrieren sich mit der gleichen SpielID.
- o Die Clients senden jeweils eine gültige Kartenhälfte.
- o Der Server sendet an jeden Client eine komplette Karte zurück.
- o Eine Anzahl von Spielrunden ist vergangen und Client 1 hat den Schatz gefunden.
- o Client 1 geht in Richtung des Clients 2 Burg.
- o Client 1 ist nur noch eine Bewegung von der Eroberung des Clients 2 Burg entfernt.
- o Jetzt ist der 1. Client an der Reihe.

Hauptsächlicher Ablauf:

- o Impuls: Client 1 fragt den Server nach dem Spielstatus.
- o Ergebnis: Der Server antwortet mit allen Informationen über den Spielstatus.
- o Impuls: Client 1 findet heraus, dass er an der Reihe ist.
- o Ergebnis: Client 1 denkt über die nächste Bewegung nach.
- o Impuls: Da Client 1 den Schatz schon hat, ist die nächste Bewegung in Richtung von Client 2 Burg.
- o Ergebnis: Der Client sendet die Bewegung an den Server.

Nachbedingungen:

- o Der Server empfängt eine gültige Bewegung.
- o Der Server hat die Position der KI auf der Karte aktualisiert.
- o Der Server prüft, ob die Gewinnbedingungen erfüllt sind.
- o Die Bedingungen sind erfüllt.
- o Der Server informiert den Client 1 über den Gewinn und den Client 2 über den Verlust des Spiels.
- o Das Spiel endet.

• **Impuls/Ergebnis - Fehlerfall:** Client 1 beschließt, ins Wasserfeld zu gehen.

Vorbedingungen:

- o Ein menschlicher Spieler hat ein Spiel erstellt.
- o Der Server teilt dem menschlichen Spieler eine SpielID mit.
- o Der menschliche Spieler fügt die SpielID in den Client ein.
- o Zwei Clients registrieren sich mit der gleichen SpielID.
- o Die Clients senden jeweils eine gültige Kartenhälfte.
- o Der Server sendet an jeden Client eine komplette Karte zurück.
- o Eine Anzahl von Spielrunden ist vergangen.
- o Client 1 befindet sich in der Nähe eines Wasserfeldes.
- o Client 1 ist auf einem Wiesenfeld.
- o Client 1 hat bereits eine Bewegung gesendet, um das Wiesenfeld zu verlassen.

Hauptsächlicher Ablauf:

- o Impuls: Client 1 fragt den Server nach dem Spielstatus.
- o Ergebnis: Der Server antwortet mit allen Informationen über den Spielstatus.
- o Impuls: Client 1 findet heraus, dass er an der Reihe ist.
- o Ergebnis: Client 1 denkt über die nächste Bewegung nach.
- o Impuls: Client 1 hat entschieden, welches die nächste Bewegung ist.
- o Ergebnis: Client 1 sendet die Bewegung an den Server.

Nachbedingungen:











- o Der Server empfängt eine Bewegung.
- o Der Server prüft die Bewegung und stellt fest, dass der Client 1 das Wasserfeld betritt.
- o Der Server informiert den Client 1 über den Gewinn und den Client 2 über den Verlust des Spiels.

• **Benutzergeschichten:**






















- o Als Menschlicher Anwender würde ich gerne eine SpielID erhalten, damit mein Client an einem Spiel teilnehmen kann.
- o Als Menschlicher Anwender möchte ich sehen können, wie das Spiel läuft. So kann ich sehen, ob mein Client gute Entscheidungen trifft.
- o Als Client möchte ich den Server kontaktieren können, damit ich meine KI auf dem Server registrieren kann
- o Als Client möchte ich eine vollständige Karte vom Server erhalten, damit die KI ihren Weg berechnen kann.
- o Als Client würde ich gerne vom Server den Spielstatus erhalten, damit ich weiß, wann ich an der Reihe bin.
- o Als Client würde ich gerne eine Bewegung an den Server senden können, damit ich gegen einen anderen Client spielen kann.
- o Als KI möchte ich gute Entscheidungen treffen, damit ich ein Spiel gewinnen kann.
- o Als KI möchte ich in der Lage sein, meine Aktionen zum richtigen Zeitpunkt zu berechnen, damit ich das Spiel nicht automatisch verliere.
- o Als KI würde ich gerne Aktionen durchführen, die nicht gegen die Regeln verstoßen, damit ich das Spiel nicht automatisch verliere.
- o Als Server möchte ich in der Lage sein, mit den Clients zu kommunizieren, damit ein Spiel stattfinden kann.
- o Als Server würde ich die Regeln überprüfen und sehen, ob die Clients die Regeln einhalten. Damit das Spiel fair ist.

• **Benutzerschnittstelle:** Während des Spiels sollten alle relevanten Spielinformationen über die CLI angezeigt werden. Kartenhälften, Spielfiguren, Positionen, Burgen, Felder, Terrains, Schätze, Bewegungen, Sieg/Niederlage und andere Aspekte sollten für den menschlichen Anwender in der command-line interface sichtbar sein. Auf diese Weise kann der menschliche Spieler beobachten, wie seine KI während des Spiels reagiert.



○ Um verschiedene Elemente des Spiels darzustellen, werden ASCII-Codes und Emojis in der CLI verwendet.

- ❖ Wiesenfelder:  (U+1F7E9)
- ❖ Wasserfelder:  (U+1F7E6)
- ❖ Bergfelder:  (U+1F7EB)
- ❖ Burg:  (U+1F3F0)
- ❖ Gegnerische Burg:  (U+1F3EF)
- ❖ Schatz:  (U+1F9F0)
- ❖ Meine KI:  (U+1F9B8)
- ❖ Gegnerische KI:  (U+1F9B9)
- ❖ Sieg: Unter der Karte wird eine Trophäe angezeigt:  (U+1F3C6)
- ❖ Niederlage: Unter der Karte wird ein Totenkopf angezeigt:  (U+1F480)

➤ Ein Beispiel für eine mögliche Anzeige in der CLI:

	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

Möglicher Text unter der Karte:

- No one won yet.
- You won! 
- You lost! 

• **Externe Schnittstellen:**

Wie bereits in der Anforderung 9 erwähnt, wird das HTTP-Protokoll verwendet sowie die zugehörigen Operationen GET und POST, damit der Client und der Server kommunizieren können. Im Netzwerkprotokoll wird auch erwähnt, dass die Dateien im XML Format ausgetauscht werden.

Für die Kommunikation stehen 5 Endpunkte zur Verfügung:

1. Erstellung eines neuen Spiels - Der menschliche Anwender sollte eine HTTP GET Operation an diesen Endpunkt senden:
 - `http(s)://<domain>:<port>/games`
 - i. Der Server antwortet mit einer XML Nachricht, die eine eindeutige SpielID beinhaltet.
2. Registrierung eines Clients - Der Client sendet einen HTTP POST Request mit einer XML Nachricht an diesen Endpunkt:
 - `http(s)://<domain>:<port>/games/<SpielID>/players`
 - i. Der Server antwortet mit einer eindeutigen SpielerID.
3. Übertragung einer der beiden Kartenhälften - Der Client sendet einen HTTP POST Request mit einer XML Nachricht im Body an folgenden Endpunkt:
 - `http(s)://<domain>:<port>/games/<SpielID>/halfmaps`
 - i. Der Server antwortet generisch mit einem `responseEnvelope`. Die Antwort informiert den Client über eventuelle Fehler bezüglich der Karte.
4. Abfrage des Spielstatus - Der Client sendet ein HTTP GET Request an folgenden Endpunkt:
 - `http(s)://<domain>:<port>/games/<SpielID>/states/<SpielerID>`
 - i. Der Server antwortet mit einem `responseEnvelope` die den Client informiert, wenn ein Fehler aufgetreten ist. Liegt kein Fehler vor, enthält der `responseEnvelope` alle aktuellen Informationen über das Spiel: `state`, `playerPositionState`, `terrain`, `treasureState`, `fortState`, `map` usw.
 - ii. Der Client muss 0,4 Sekunden warten, bevor er eine weitere Anfrage an den Server senden kann, damit der Server nicht überlastet wird.
5. Übertragung einer Bewegung - Der Client sendet einen HTTP POST Request an folgenden Endpunkt:
 - `http(s)://<domain>:<port>/games/<SpielID>/moves`
 - i. Der Server antwortet generisch mit einem `responseEnvelope`. Die Antwort informiert den Client über eventuelle Fehler bezüglich der Bewegung.

Aufgabe 3: Architektur entwerfen, modellieren und validieren (10 Punkte)

Client_klassendiagramm.svg

Main: Der Client wird von der Klasse Main gestartet. Dort muss zunächst eine PlayerData initialisiert werden. PlayerData enthält alle Informationen, damit sich ein Client und ein Spieler registrieren können. Dann wird innerhalb der Main Klasse eine Instanz von ClientController initialisiert.

ClientNetworkHandler: Sendet und empfängt Informationen. Es verwendet auch einen Konverter, so dass lokale Klassen in Netzwerkklassen umgewandelt werden können. Diese Klasse prüft auch, ob neue Informationen vom Server verfügbar sind.

ClientController: Im ClientController wird ein ClientNetworkHandler initialisiert. Von hier aus werden auch alle Methoden des ClientNetworkHandlers aufgerufen. Der Controller ist dafür verantwortlich, wie sich der Client verhält. Der ClientController registriert den Spieler, fragt nach dem GameStatus, sendet die HalfMap, empfängt eine vollständige Karte, aktualisiert die Informationen und sendet die Bewegung.

Map Package: Enthält die vom Server gesendete FullMap und einen MapAnalyser. Der Analyser „scannt“ die vom Server erhaltenen neuen Informationen und extrahiert Spielerpositionen und Informationen über den Spielverlauf. MapAnalyser wird auch als Model in MVC verwendet.

Halfmap Package: Wenn der Controller eine neue HalfMap benötigt, wird sie dort mit dem ClientHalfMapGenerator und dem HalfMapValidator erzeugt. HalfMapValidator prüft die Regeln für eine Kartenhälfte. Wenn etwas falsch ist, muss eine neue HalfMap erstellt und erneut geprüft werden.

Movement Package: Hier wird der nächste Zug entschieden. Abhängig von der aktuellen Spielerposition und ob der Schatz eingesammelt wurde, wird ein Ziel ausgewählt. Dann wird der kürzeste Weg zwischen der aktuellen Position und dem Ziel berechnet.

Client_sequenzdiagramm.svg

Hier wird der Client darüber informiert, dass er den Schatz erfolgreich abgeholt hat. Dies führt zu einer Änderung der Strategie für die Erzeugung der nächsten Bewegungen.

Server_klassendiagramm.svg

Main: Ähnlich wie beim Client, startet hier der Server. Nun wird ein **ServerController** initialisiert.

ServerController: Ähnlich wie beim Client, hier findet die gesamte Interaktion mit dem **ServerNetworkHandler** statt. Alle Aktionen sind das Gegenteil von denen, die im Client durchgeführt werden. Statt einer **HalfMap** zu senden, wird hier beispielsweise eine **HalfMap** empfangen. **ServerController** prüft auch, ob die Nachrichten vom Server gültig sind: ob das Spiel existiert, ob das Spiel voll ist, ob ein Spieler registriert ist usw.

Hier sind auch alle aktiven Spiele gespeichert. Jedes Mal, wenn das Netzwerk etwas an den Controller sendet, wird die Liste der Spiele bereinigt. Das ist es, was der **ActiveGamesSanitizer** tut.

GameManager: Verantwortlich dafür, wie ein Spiel läuft. Nach Erhalt von 2 **HalfMaps** wird automatisch eine **Fullmap** erstellt. Hier wird auch geprüft, ob die **HalfMaps** die Anforderungen erfüllen. **GameStatus** wird dort unter Nutzung der aktuellen **GameStats** und **PlayerStats** erstellt.

MoveManager: Empfängt und bearbeitet einen Zug. Aktualisiert die **PlayerStats**, einschließlich der aktuellen Spielerposition. Es ist auch dafür verantwortlich, wie viele Züge erforderlich sind, damit der Spieler eine neue Position einnimmt.

Network Package: Nimmt Informationen vom Client entgegen. Wandelt auch die Klassen um, damit der Server mit den Clients kommunizieren kann.

Server_sequenzdiagramm.svg

Der Client sendet eine Karte, die eine Insel enthält. Wenn **GameManagers** das überprüft, stellt es fest, dass es eine Insel gibt und die Karte nicht akzeptiert werden kann. Der Spieler, der die **HalfMap** geschickt hat, verliert.