# Lastenheft für *Marvelous Mashup*

**Version 2020-12-12**

## Abstract

Dieses Lastenheft beschreibt die Anforderungen an das Spiel *Marvelous Mashup*, welches im Softwaregrundprojekt entwickelt werden soll. Dieses Dokument wird wahrscheinlich zu späteren Zeiten überarbeitet oder erweitert werden, um geänderte oder präzisierte Anforderungen widerzuspiegeln, die sich im Lauf des Projekts ergeben können.

**Kapitel 1** gibt eine Übersicht zur Architektur und den Komponenten des zu entwickelnden Produkts, und legt einige technische Randbedingungen fest.

**Kapitel 2** beschreibt die Spielregeln des eigentlichen Spiels *Marvelous Mashup*.

**Kapitel 3** listet verbindliche und optionale Anforderungen an die einzelnen Komponenten des verteilten Systems auf.

**Kapitel 4** macht Vorgaben zum Ablauf des Projekts und beschreibt den einzuhaltenden Entwicklungsprozesses.

**Kapitel 5** legt die Abnahmekriterien für das Produkt fest.

# Inhaltsverzeichnis

**1 Produkt und Einsatzszenarien** **3**
1.1 Komponenten und Architektur . . . 3
1.2 Anwendungssprache, Implementierungssprache und Dokumentationssprache . . . 4
1.3 Programmiersprachen und Technologien . . . 4
1.4 Plattformen . . . 4
1.5 Netzwerkkommunikation und Nachrichtenprotokoll . . . 4
1.6 Formate für Content- und Konfigurations-Dateien . . . 4

**2 Spielregeln von *Marvelous Mashup*** **5**
2.1 Szenarios . . . 5
2.2 Charaktere . . . 6
2.3 Bewegung . . . 6
2.4 Aktionen . . . 6
2.5 Die Infinity Stones . . . 7
2.6 Partie-Vorbereitung . . . 8
2.6.1 Wahlphase . . . 8
2.7 Beginn der Partie . . . 8
2.8 Runden und Züge . . . 8
2.8.1 Züge von Charakteren . . . 8
2.9 Goose . . . 9
2.10 Stan . . . 9
2.11 Ende der Partie und Bestimmung des Siegers . . . 9
2.11.1 Behandlung überlanger Partien . . . 9
2.11.2 Sieg . . . 10
2.12 Allgemeine Bemerkungen zur Interpretation der Regeln . . . 10

**3 Funktionalität von Komponenten** **11**
3.1 Server . . . 11
3.2 Benutzer-Client . . . 12
3.3 KI-Client . . . 13
3.4 Editor . . . 13

**4 Vorgaben zum Entwicklungsprozess** **14**
4.1 Begleitveranstaltung . . . 14
4.2 Moodle . . . 14
4.3 Git und Gitlab . . . 14
4.4 Qualitätssicherung und SonarQube . . . 14
4.5 Docker . . . 14
4.6 Tutorien und Rolle der Tutoren . . . 15
4.7 Agiler Entwicklungsprozess . . . 15
4.8 Dokumentation . . . 15
4.9 Standardisierungskomitee . . . 16
4.10 Messe . . . 16
4.11 Abschlussturnier . . . 17
4.12 Highscore und Preise . . . 17

**5 Kriterien für die Abnahme** **18**
5.1 Abnahmesitzung . . . 18

**A Beispiele für Charaktere aus der Charakter-Konfiguration** **19**

# 1 Produkt und Einsatzszenarien

Im Softwaregrundprojekt soll das **rundenbasierte Taktik-Spiel *Marvelous Mashup*** als Multiplayerspiel entwickelt werden. Dieses Kapitel beschreibt die Architektur des verteilten Systems und listet seine Komponenten auf. Danach werden technische Vorgaben gemacht.

## 1.1 Komponenten und Architektur

Das zu entwickelnde Produkt ist eine verteilte Anwendung mit **Client-Server-Architektur.** Die Clients kommunizieren dabei nur mit dem Server, aber nicht untereinander. Das System besteht aus folgenden Komponenten (Abb. 1.1):

- Ein **Server**, der eine Partie von *Marvelous Mashup* verwalten und abwickeln kann. An einer Partie nehmen genau zwei gegeneinander spielende Clients teil. Eine Partie kann von beliebig vielen anderen Clients als passive Zuschauer beobachtet werden. Der Server beinhaltet die Anwendungslogik zur Verwaltung von Clients und Partien sowie die Umsetzung der Spielregeln von *Marvelous Mashup*. Einstellungen für eine konkrete Partie werden über eine **Partie-Konfiguration** vorgegeben. Die Partie-Konfiguration bestimmt u.a., wieviel Zeit den Clients für bestimmte Phasen des Spiels gewährt wird und wieviele Runden vergehen dürfen, bis die Bedingung für überlange Partien eintritt. Der Server bekommt auch das **Szenario** übergeben, das gespielt werden soll, und eine **Charakter-Konfiguration** mit den spielbaren Superhelden.
- Ein **Benutzer-Client** mit graphischer Oberfläche, der es einem einzelnen menschlichen Benutzer ermöglicht, entweder aktiv als Mitspieler an einer Partie teilzunehmen, oder als passiver Zuschauer im Zuschauermodus eine Partie zwischen zwei anderen Clients zu verfolgen.
- Ein **KI-Client** um auch vereinsamten Spielern ein spannendes Multiplayer-Erlebnis bieten zu können. KI-Clients werden autonom von einer Künstlichen Intelligenz gesteuert und können als Mitspieler an einer Partie teilnehmen.
- Ein **Editor,** mit dem Content und Konfigurationen für das Spiel erstellt werden können, wie **Szenarios, Partie-Konfigurationen** und **Charakter-Konfigurationen.**

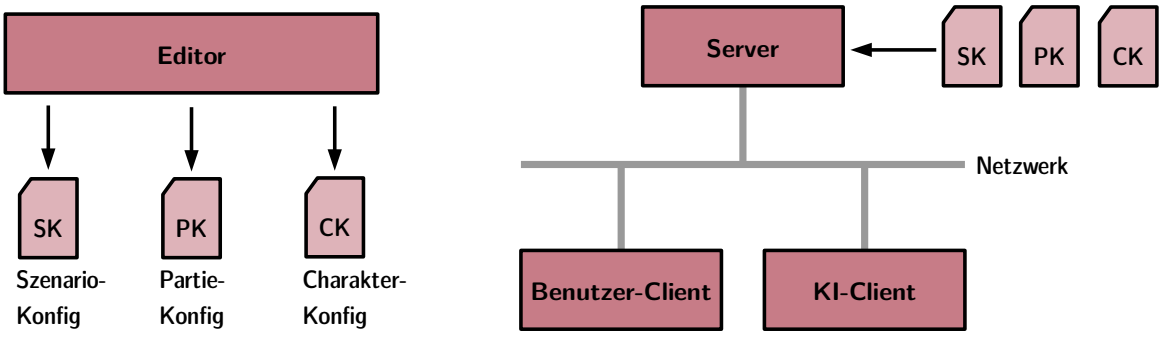


Abbildung 1: Komponenten und Architektur des Multiplayerspiels.

Jedes Team muss am Ende der Projektlaufzeit ein funktionierendes Spiel mit allen Komponenten abliefern. Ihr sollt dabei die Komponenten **Benutzer-Client** und **KI-Client** selbst entwickeln. Von den beiden Komponenten **Server** und **Editor** müsst ihr nach eigener Wahl nur *eine* implementieren und bekommt die jeweils andere von einem anderen Team auf einer Messe zur Verfügung gestellt (s. Abschnitt 4.10).

Die Idee ist hierbei, die verteilte Anwendung so zu implementieren, dass man die Komponenten über Teamgrenzen hinweg beliebig kombinieren kann, also bspw. Szenario und Charakter-Konfig mit Editor von Team A bauen, Partie-Konfig mit Editor von Team B, damit dann den Server von Team C starten, auf dem ein Benutzer-Client von Team D gegen den KI-Client von Team E spielt, während Benutzer-Clients von nochmal fünf anderen Teams als Beobachter zuschauen...

## 1.2 Anwendungssprache, Implementierungssprache und Dokumentationssprache

Die **Benutzerschnittstelle** der Anwendung (d.h. aller Komponenten) kann auf Deutsch oder Englisch gestaltet werden. Die **Implementierungssprache** (Bezeichner im Source Code und Kommentare) soll jedoch Englisch sein. Sonstige **Dokumente** wie Benutzerhandbuch, Testdokumentationen, Projekttagebuch, usw. können auf Deutsch oder Englisch verfasst werden.

## 1.3 Programmiersprachen und Technologien

Als Programmiersprachen empfehlen wir **Java, JavaScript, C# und C++.** Die hauptsächliche Lehrsprache in Grundlagenveranstaltungen ist Java. Jeder im Team sollte also zumindest Grundkenntnisse in Java haben. Teams dürfen auch andere Sprachen verwenden (z.B. ihre KI in Haskell schreiben), aber kein Teammitglied soll von der Arbeit an Komponenten ausgeschlossen werden, weil es mit einer "exotischen" Sprache nicht vertraut ist.

Verschiedene Komponenten können auch in unterschiedlichen Sprachen entwickelt werden. Jedes Team kann selbst entscheiden, welche Sprachen, Bibliotheken, Frameworks und Technologien es verwendet. Dies setzt allerdings voraus, dass innerhalb des Teams ein Konsens über diese Auswahl besteht, und der Tutor als Vertreter des Auftraggebers zustimmt. Da alle Teammitglieder an der Implementierung mitwirken sollen, sollten Sprachen gewählt werden, mit denen alle hinreichend vertraut sind.

## 1.4 Plattformen

Die Komponenten **Benutzer-Client** und **Editor** müssen jeweils auf mindestens einer der folgenden Plattformen laufen:

- eine der verbreiteten, aktuellen Linux-Distributionen
- eine aktuelle Version von Microsoft Windows
- basierend auf Webtechnologien, in einem aktuellen, standardkonformen Browser

Die Komponenten **KI-Client** und **Server** müssen inklusive aller Abhängigkeiten als Docker-Container lauffähig sein (s. Abschnitt 4.5). Nach pullen des jeweiligen Docker-Images können diese Komponenten deshalb ohne komplizierte lokale Build-Prozesse direkt gestartet werden.

## 1.5 Netzwerkkommunikation und Nachrichtenprotokoll

Als Netzwerkprotokoll für die Kommunikation der Komponenten untereinander ist das **WebSocket-Protokoll** [2] vorgegeben. Über die WebSocket-Verbindung werden **Textstrings im UTF-8 Encoding** ausgetauscht, die im Format **JSON** [3] repräsentierte Nachrichten des anwendungsspezifischen Spielprotokolls enthalten. Das Spielprotokoll wird vom Standardisierungskomitee definiert (s. Abschnitt 4.9).

## 1.6 Formate für Content- und Konfigurations-Dateien

**Szenarios, Charakter-Konfigurationen** und **Partie-Konfigurationen** sollen im Format **JSON** repräsentiert werden. Die genauen Schemata definiert das Standardisierungskomitee.

# 2 Spielregeln von *Marvelous Mashup*

*Marvelous Mashup* ist ein **rundenbasiertes Taktik-Spiel** für **zwei Spieler** in der Welt der Comic Superhelden und Superschurken.

In der Savanne von Wakanda haben sich zwei rivalisierende Gruppen von Superhelden versammelt. Jeder Spieler steuert eine **Heldengruppe. Thanos** hat gerade alle **sechs Infinity Stones** gesammelt, da kommt **Goose** daher und verschluckt sämtliche Steine. Thanos ist daraufhin so frustriert, dass er sich erstmal vom Acker macht. Goose ist ein Flerken. Sie sieht zwar aus wie eine kleine orange Katze, ist aber in Wirklichkeit ein Alien mit einem transdimensionalen Magen, in dem sie alles mögliche verstauen kann. Außerdem kann sie sich selbst verschlucken, und woanders wieder ausspucken. Diese Fähigkeit erlaubt Goose, sich beliebig zu teleportieren.[1]

Da die Infinity Stones nun aber Goose schwer im Magen liegen, teleportiert sie sich kreuz und quer in der Savanne hin und her, und spuckt nacheinander alle Steine wieder aus. Die Superhelden versuchen sich diese Steine zu schnappen, und kloppen sich darum. Das Ziel des Spieles ist, **einem Charakter der eigenen Heldengruppe alle sechs Steine zu geben,** womit dieser Spieler die Partie gewonnen hat. Falls die Helden sich aber zulange über die Savanne kloppen, taucht irgendwann Thanos wieder auf, holt sich alle Steine zurück, und fängt dann an, Leute zu zerstäuben, weil er richtig miese Laune hat.

*Marvelous Mashup* ist ein Spiel mit vollständiger Information. Das bedeutet, es gibt keinen Fog of War oder ähnliches. Alle Spieler können zu jeder Zeit das ganze Spielfeld und die Eigenschaften sowie den Zustand und das Inventar aller Charaktere sehen.

## 2.1 Szenarios

*Marvelous Mashup* spielt in der Savanne von Wakanda. Als **Szenario** für eine Partie ist ein Spielbrett gegeben, das ein rechteckiges kartesisches Raster aus $x * y$ **Feldern** (schachbrettartige Felder) ist. Mit dem Editor kann man solche Szenarios erstellen und in einer Szenario-Konfiguration speichern, die dann vom Server geladen wird.

Felder des Spielbretts können freier Raum sein, oder durch Hindernisse oder Objekte besetzt sein. Sie können betretbar sein oder nicht, und die Sichtlinie durchlassen, oder blockieren.

| *Feldart* | *Beschreibung* | *frei?* | *betretbar?* | *blockiert Sichtlinie?* |
|---|---|---|---|---|
| **Gras** | Ein freies Feld, auf dem nichts platziert ist. Charaktere können auf ihm stehen, oder darüber hinweglaufen. | ja | ja | nein |
| **Fels** | Felsen haben anfangs 100 HP, und können durch Angriffe beschädigt werden. Sinken die HPs eines Felsen auf 0, so zerbröckelt der Felsen und das Feld wird zu einem Grasfeld. | nein | nein | ja |
| **Grasfeld mit Charakter** | Ein besetztes Grasfeld auf dem ein Charakter steht. Ein anderer Charakter kann auf dieses Feld laufen, wobei der dort stehende Charakter (egal ob lebendig oder ausgeknockt) auf das Feld gesetzt wird, vom dem der andere kam. Die beiden tauschen also ihre Plätze. | nein | ja | ja |
| **Grasfeld mit Infinity Stone** | Ein besetztes Grasfeld, auf dem ein Infinity Stone liegt. Wenn ein Charakter auf dieses Feld läuft, nimmt er den Stein in sein Inventar auf. Auf einem Feld kann höchstens ein Stein auf dem Boden liegen. | nein | ja | nein |

Die **Entfernung** zwischen zwei Feldern $A$ und $B$ ist definiert als die minimale Anzahl von aufeinanderfolgenden Schritten auf Nachbarfelder (in alle acht Richtungen), um vom $A$ nach $B$ zu gelangen (d.h. wie viele Züge ein König im Schach mindestens machen müsste).

Zwischen zwei Feldern $A$ und $B$ besteht eine **Sichtlinie**, wenn folgendes gilt: Wir betrachten zunächst die Verbindungs-

[1] Wie das technisch genau funktioniert ist nicht ganz klar, die Quellen sind da eher vage...

linie vom Mittelpunkt von $A$ zum Mittelpunkt von $B$. Alle Felder außer $A$ und $B$ selbst (also diejenigen dazwischen), die von der Verbindungslinie *geschnitten* werden, müssen Felder sein, die die Sichtlinie nicht blockieren, damit $B$ von $A$ aus sichtbar ist. Undurchsichtige Felder, die von der Verbindungslinie lediglich an einer Ecke tangiert (also nicht geschnitten) werden, unterbrechen diese nicht (das kann bei diagonalen Linien im Raster der Felder vorkommen).

## 2.2 Charaktere

Charaktere haben bestimmte Werte, die in der Charakter-Konfiguration definiert sind.

- **Name:** Superhelden haben einen coolen Superhelden-Namen.
- **Movement Points (MP) und Action Points (AP):** Jeder Charakter bekommt pro Zug eine bestimmte Anzahl an Punkten, die er jeweils für Bewegungsschritte und Aktionen nutzen kann.
- **Health Points (HP):** Stellen den Gesundheitszustand des Charakters dar. Wenn die HPs auf 0 sinken, ist der Charakter ausgeknockt, und liegt einfach passiv am Boden. Er kann keine Züge mehr machen, bis er ggf. wiederbelebt wird. Falls ein Charakter Infinity Stones in seinem Inventar hat, wenn er ausgeknockt wird, so werden diese Steine in zufälliger Reihenfolge auf zufällige freie Nachbarfelder des Charakters platziert. Sie fliegen also aus seiner Tasche und landen um ihn herum verstreut auf dem Boden, jeder Stein auf einem eigenen Feld.
- **Inventar:** Ein Charakter kann Infinity Stones in seinem Inventar tragen.

Die Werte und das Inventar aller Charaktere sind immer für alle Spieler sichtbar.

Anhang A enthält eine beispielhafte Liste von Charakteren.

## 2.3 Bewegung

Zu Beginn seines Zuges bekommt jeder Charakter so viele **Movement Points (MP)**, wie in der Charakter-Konfiguration für ihn festgelegt ist. Für einen Movement Point kann ein Charakter sich auf ein betretbares Nachbarfeld bewegen. Er kann dabei im Raster der Felder horizontal, vertikal oder diagonal ziehen (wie ein König im Schachspiel).

Wenn ein Charakter sich auf ein Feld bewegt, auf dem bereits eine andere Person steht, so drängelt sich der Charakter knallhart vorbei und tauscht den Platz mit der Person. Der aktive Charakter steht also auf dem Feld, auf dem die Person davor stand, und diese wird auf das Feld zurückgeschubst, von dem der Charakter kam. Das gilt für lebendige und ausgeknockte Charaktere.

## 2.4 Aktionen

Zu Beginn seines Zuges bekommt jeder Charakter so viele **Action Points (AP)**, wie in der Charakter-Konfiguration für ihn festgelegt ist. Für einen Aktionspunkt kann der Charakter eine Aktion machen. Es gibt mehrere Arten von Aktionen:

- **Nahkampf-Angriff:** Mit einer Aktion kann ein Charakter einen Gegner auf einem benachbarten Feld hauen. Der Gegner bekommt den Nahkampf-Schaden des Angreifers (steht in der Charakter-Konfiguration) von seinen Health Points abgezogen.
- **Fernkampf-Angriff:** In der Charakter-Konfiguration steht die Reichweite in Feldern (mindestens 2) und der Schaden für Fernangriffe. Mit einer Aktion kann ein Charakter einen Fernkampf-Angriff auf einen Gegner, der nicht auf einem benachbarten Feld steht, aber sich in Reichweite und Sichtlinie befindet, durchführen. Der Gegner bekommt den Schaden von seinen HPs abgezogen. Auf Gegner, die direkt neben dem Charakter stehen, kann dieser also nur einen Nahkampf-Angriff machen.

- **Verwendung eines Infinity Stones:** Mit einer Aktion kann ein Charakter die spezielle Aktion durchführen, die ihm ein Infinity Stone in seinem Inventar ermöglicht. Die Steine haben aber individuelle Cooldown-Zeiten, sodass diese Aktionen nur alle *n* Runden angewendet werden können.
- **Übergabe eines Infinity Stones:** Mit einer Aktion kann ein Charakter einem anderen auf einem benachbarten Feld einen Infinity Stone aus seinem Inventar in dessen Inventar übergeben. Da das Spielziel ist, dass am Ende ein einziger Charakter der eigenen Gruppe alle sechs Steine hält, genügt es nicht nur, mit mehreren Charakteren alle Steine zu sammeln, sondern man muss sie am Ende alle im Inventar eines Charakters zusammen haben.

## 2.5 Die Infinity Stones

Es gibt sechs Infinity Stones. Jeder Stein hat eine **Farbe**[2] und erlaubt dem Charakter, der ihn im Inventar hat, eine **spezielle Aktion** auszuführen. Nachdem diese Aktion ausgeführt wurde, kann der jeweilige Stein *n* Runden lang nicht mehr benutzt werden, um diese Aktion auszuführen. Diese **Cooldown** Rundenzahl ist spezifisch für jeden Stein, und ist an den Stein gebunden, nicht an den Charakter der ihn besitzt. Auch wenn der Stein von jemand anderem übernommen wird, bleibt sie wirksam. Die Cooldown-Zeiten für die einzelnen Steine werden in der Partie-Konfiguration festgelegt.

- **Space Stone (blau):** Der Träger kann sich mit einer Aktion auf ein beliebiges freies Feld teleportieren.
- **Mind Stone (gelb):** Der Träger kann seine mentale Energie in einen Strahl fokussieren, und einen Fernkampf-Angriff (Schaden wird in Partie-Konfiguration definiert) auf jedes Ziel in Sichtlinie (also mit unbeschränkter Reichweite) ausführen.
- **Reality Stone (rot):** Der Träger kann mit einer Aktion auf einem benachbarten Feld einen Felsen verschwinden lassen, oder auf einem benachbarten freien Feld einen Felsen (mit 100 HP) erscheinen lassen.
- **Power Stone (violett):** Der Träger kann mit einer Aktion einen Nahkampf-Angriff ausführen, der den doppelten Schaden seines normalen Nahkampf-Angriffs macht. Weil das Verwenden des Power Stones anstrengend ist, werden die HPs des Trägers bei der Aktion um 10% seiner maximalen HPs reduziert, wobei der Wert nicht unter 1 fallen kann.
- **Time Stone (grün):** Der Träger kann mit einer Aktion die "Zeit zurückspulen", und seine MPs und APs auf den Anfangswert zurücksetzen. Bspw. kann ein Charakter mit 2 MP und 2 AP sich zwei mal bewegen, mit einer Aktion jemanden hauen, und mit seinem letzten AP den Time Stone benutzen, und hat dann wieder 2 MP und 2 AP mit denen er seinen Zug fortsetzen kann.
- **Soul Stone (orange):** Der Träger kann mit einer Aktion einen ausgeknockten Charakter auf einem benachbarten Feld "wiederbeleben". Die HPs dieses Charakters werden auf den vollen Wert aus der Charakter-Konfiguration gesetzt.

Wenn Charaktere mehrere Infinity Stones im Inventar haben, können sie also interessante Combos machen, bspw. jemanden kloppen, dann mit dem Space Stone woanders hin teleportieren, sich mit dem Time Stone zusätzliche APs holen, mit dem Soul Stone einen ausgeknockten Freund wiederbeleben, und dann mit dem Reality Stone einen Felsen platzieren, hinter dem sie Deckung vor dem Fernangriff eines Gegners haben.

[2] Wir verwenden hier die Farben, die die Steine im Universum #199999 des Multiversums haben.

## 2.6 Partie-Vorbereitung

Bevor die Partie startet, wählen die Spieler Charaktere für ihre Heldengruppe aus.

### 2.6.1 Wahlphase

Zu Beginn muss der Spieler eine **Heldengruppe aus 6 Charakteren zusammenstellen.** Die Herausforderung hier ist, die Charaktere aus dem Vorrat der in der Charakter-Konfiguration definierten Figuren so zu wählen, dass eine möglichst gute Kombination entsteht, was die Eigenschaften der Superhelden angeht.

Der Ablauf der **Wahlphase aus Sicht eines Spielers** ist so:

1. Der Server nimmt aus der Liste aller Charaktere der geladenen Charakter-Konfiguration 12 zufällige Charaktere, die dem Spieler angezeigt werden.
2. Der Spieler wählt 6 davon aus.

Die Charakter-Konfiguration muss also mindestens 24 Charaktere umfassen, da jedem der beiden Spieler nebenläufig eine Menge von 12 Charakteren (jeweils eigene) zur Auswahl gegeben wird.

## 2.7 Beginn der Partie

Vor der ersten Runde platziert der Server alle Charaktere auf zufälligen freien Felder auf dem Spielbrett.

## 2.8 Runden und Züge

*Marvelous Mashup* läuft in **Runden** ab, in denen Ereignisse passieren, und die einzelnen Charaktere ihre Züge nacheinander machen. Zu Beginn jeder Runde handelt der Server einige Ereignisse ab, wie etwa Aktionen von Goose, Stan oder Thanos.

Dann werden alle Charaktere vom Server für diese Runde in eine zufällige Reihenfolge gebracht. Die Charaktere kommen nun der Reihe nach dran, und können ihren **Zug** machen (wenn sie in diesem Moment nicht ausgeknockt sind), der aus mehreren **Zugphasen** besteht. Nach jeder Zugphase wird vom Server ausgewertet, ob die Bedingung für ein Ende der Partie eingetreten ist.

### 2.8.1 Züge von Charakteren

**Züge** von Charakteren bestehen aus **Zugphasen,** in denen jeweils entweder ein **Bewegungsschritt** oder eine **Aktion** gemacht wird. Wenn ein Charakter mit seinem Zug an der Reihe ist, werden zuerst seine Movement Points und Action Points auf die Werte aus der Charakter-Konfiguration gesetzt. Der Spieler kann dann für den aktiven Charakter diese Punkte in beliebiger Reihenfolge, Zugphase für Zugphase, einen nach dem anderen ausgeben (die Punkte müssen aber nicht alle ausgegeben werden).

Bsp.: Rocket Raccoon ist am Zug. Er hat z.B. laut der Charakter-Konfiguration 2 MP und 2 AP. Als erstes benutzt der Spieler einen AP um Rocket einen Fernkampf-Angriff auf Nebula machen zu lassen, die in Reichweite und Sichtlinie steht. Dann benutzt er einen MP, um den Space Stone auf einem Nachbarfeld aufzusammeln. Mit dem zweiten AP benutzt Rocket den Space Stone um sich neben den ausgeknockten Starlord zu teleportieren, damit er ihn im nächsten Zug (wenn er wieder APs hat) mit dem Soul Stone in seinem Inventar wiederbeleben kann. Den zweiten MP lässt der Spieler verfallen, da Rocket nach dem Teleport bereits ideal für den nächsten Zug steht. Rocket beendet seinen Zug.

Die Abhandlung des Zuges eines Charakters, d.h. seiner Bewegungsschritte und Aktionen, findet also in aufeinanderfolgenden Zugphasen, Punkt für Punkt statt.

## 2.9 Goose

Zu Beginn der Runden 1 bis 6 (also bevor die Charaktere ihre Züge machen) erscheint Goose auf einem zufälligen freien Feld, kotzt einen zufälligen der sich noch in ihrem Magen befindenden Infinity Stones auf diesem Feld aus, und verschwindet dann wieder.

## 2.10 Stan

Natürlich gibt es in *Marvelous Mashup* auch einen Cameo-Auftritt von Stan Lee, weil sich das so gehört. Zu Beginn der Runde 7 erscheint Stan auf einem zufälligen freien Nachbarfeld eines zufälligen ausgeknockten Charakters. Falls keiner der Charaktere ausgeknockt ist, erscheint Stan stattdessen auf einem zufälligen freien Nachbarfeld des Charakters, der aktuell die wenigsten HPs hat. Alle Charaktere, die eine Sichtlinie zu Stan haben, freuen sich, ihn zu sehen, und bekommen ihre HPs auf den vollen Wert gesetzt. Das gilt auch für alle Charaktere, die ausgeknockt sind, und in Sichtlinie liegen. Sie werden entsprechend wiederbelebt. Stan ruft "Excelsior!"[3] und verschwindet dann wieder.

## 2.11 Ende der Partie und Bestimmung des Siegers

Sobald ein Charakter am Ende einer Zugphase (nicht unbedingt seiner eigenen, da Steine übergeben werden können) alle sechs Infinitiy Stones in seinem Inventar hat, endet die Partie sofort, und der Spieler dieses Superhelden hat gewonnen.

### 2.11.1 Behandlung überlanger Partien

Auch die längste Partie von *Marvelous Mashup* sollte irgendwann einmal zuende gehen. Um zu vermeiden, dass Partien übertrieben lange dauern, brauchen wir einen Mechanismus, der die maximal mögliche Dauer einer Partie begrenzt. Wenn die Partie über mehr Runden läuft, als ein in der Partie-Konfiguration festgelegter Höchstwert (**Überlänge-Rundenzahl** $r_{max}$), wird sie durch einen speziellen **Überlängenmechanismus** einem beschleunigten Ende zugeführt. Sobald diese Überlängenbedingung eintritt, also zu Beginn der $r_{max}$-ten Runde, passiert folgendes:

- Thanos der wahnsinnige Titan erscheint auf einem zufälligen freien Feld. Er ist ein spezieller Non-Player Character, wird aber in der Zugreihenfolge wie die anderen irgendwo zufällig eingeordnet. Wenn Thanos mit seinem Zug dran ist, setzt er sich das nächstgelegene Feld zum Ziel, auf dem sich Infinity Stones befinden (entweder im Inventar eines Charakters oder am Boden liegend) und bewegt sich auf dieses Feld zu. Er hat dabei in der Runde, in der er erscheint, so viele MPs wie der Charakter im Spiel, der die meisten MPs hat. Thanos kann dabei auch durch Felsen laufen, die sobald er sie betritt, zerbröseln und zu Grasfeldern werden. Seine Wegfindung ignoriert also Hindernisse, er stürmt einfach geradezu auf den nächststehenden Steinträger oder den nächstgelegenen Stein am Boden zu. Wenn er auf ein Feld mit einem liegenden Stein läuft, nimmt er diesen in sein Inventar auf und beendet seinen Zug. Wenn Thanos auf das Feld eines Charakters mit Steinen im Inventar läuft, wird dieser Charakter auf das Feld gesetzt, von dem Thanos kam, ausgeknockt, und alle Steine im Inventar dieses armen Charakters kommen in das Inventar von Thanos und dieser beendet seinen Zug. Dieses Verhalten wiederholt Thanos bei allen seinen folgenden Zügen, bis er alle sechs Infinity Stones gesammelt hat, wobei seine MPs in jedem folgenden Zug um 1 erhöht werden. Das bedeutet, dass auch der Träger des Space Stones nicht ewig vor Thanos fliehen kann, indem er ständig herumteleportiert. Charaktere können nicht auf das Feld laufen, auf dem Thanos steht. Außerdem ist Thanos immun gegen alle Angriffe, und kann selbst nicht ausgeknockt werden.
- Wenn Thanos am Zug ist, und zu Beginn seines Zugs alle sechs Infinity Stones in seinem Inventar hat, bleibt er auf seinem gegenwärtigen Feld stehen und schnippst seine Finger. Jeder Charakter der Spieler wird daraufhin mit einer Wahrscheinlichkeit von 50% zerstäubt und aus dem Spiel entfernt. Damit ist dieser Zug für Thanos beendet. Das wiederholt er bei jedem folgenden Zug, solange noch Charaktere übrig sind. Sobald der letzte Charakter zerstäubt wurde, endet die Partie.

[3] Entweder eine aufpoppende Comic-Sprechblase oder ein cooler Soundeffekt ;)

### 2.11.2 Sieg

Wenn die Partie dadurch endet, dass ein Charakter alle sechs Infinity Stones in seinem Inventar hat, gewinnt der Spieler dem dieser Charakter gehört.

Wenn die Partie aber dadurch endet, dass Thanos alle Charaktere zerstäubt hat, wird der Sieger dadurch bestimmt, dass folgende **Sieg-Metriken** in Reihenfolge betrachtet werden (wenn nach einem Vergleich Gleichstand herrscht, wird die jeweils nächste Metrik als Tie-Breaker herangezogen):

1. Welche Gruppe hat zu einem Zeitpunkt die größte Anzahl an Infinity Stones besessen? (Also der maximale Wert der Summe aller gleichzeitig von allen Helden einer Gruppe im Inventar gehaltenen Steine.)
2. Welche Gruppe hat mehr gegnerische Charaktere ausgeknockt?
3. Welche Gruppe hat in Summe mehr Schaden an Health Points bei der gegnerischen Gruppe verursacht?
4. Wenn alles nichts hilft, wird der Sieger durch das Los bestimmt.

## 2.12 Allgemeine Bemerkungen zur Interpretation der Regeln

**Zufall:** Es wird an einigen Stellen gesagt, dass etwas zufällig gewählt werden soll. Damit ist immer eine gleichverteilt zufällige Auswahl unter den entsprechenden Möglichkeiten gemeint.

**Alternativen:** Manchmal soll etwas auf einem "zufälligen freien Nachbarfeld" oder so ähnlich platziert werden. Es kann sein, dass es kein freies Nachbarfeld gibt. In einem solchen Fall wird ein zufälliges (nicht freies) Nachbarfeld gewählt, und dann rekursiv davon ein zufälliges freies Nachbarfeld gesucht, usw., bis man irgendwann ein freies findet (man macht also im Prinzip einen Random Walk bis man auf einem freien Feld landet). Das Objekt wird also ggf. etwas weiter weg von dort platziert, wo es eigentlich hin sollte. Die Idee ist hier einfach, das zu tun, was dem beabsichtigten Sinn der Regeln möglichst am nähesten kommt und leicht zu implementieren ist. In seltenen Fällen können komische Dinge passieren, etwa das Objekte durch Hindernisse "hindurchtunneln" oder ähnliches. Das ist dann halt so. Wenn es mehrere gleichwertige Alternativen gibt (z.B. man sucht einen Charakter, der am nähesten zu einem Feld steht, und mehrere Charaktere sind gleich weit von dort entfernt), so soll eine der gleichwertigen Alternativen zufällig ausgewählt werden.

Wenn bspw. Thanos zum "am nähesten gelegenen" Feld mit einem Stein laufen soll, und es liegt auf einem Feld mit Entfernung 5 ein Stein am Boden, während ebenso in Entfernung 5 Dr. Strange mit einem Stein im Inventar steht, und es gibt keine näheren Steine irgendwo, dann wählt Thanos zufällig ob er zu dem liegenden Stein oder Dr. Strange läuft, da beide Alternativen gleich weit entfernt sind.

Wenn Stan beim Charakter mit den wenigsten HPs erscheinen soll, und zwei haben beide den minimalen Wert von 42, dann sucht sich Stan eben zufällig aus, bei welchem der beiden er erscheint. Falls dieser Charakter aber kein "freies Nachbarfeld" hat, so sucht Stan wie oben beschrieben rekursiv nach irgendeinem freien Feld in der Nähe.

Wenn ein Charakter ausgeknockt wird, und er steht in der Ecke zwischen Felsen und hat nur zwei freie Nachbarfelder aber drei Steine fliegen aus seinem Inventar, so landen zwei auf diesen freien Nachbarfeldern, und der dritte sucht sich rekursiv ein freies Feld in der Nähe. Unter Umständen kann dieser Stein dann auf der anderen Seite der Felsmauer landen. Das sieht dann so aus, als sei er durch die Wand getunnelt, aber das ist okay.

Die Idee hier ist, die Regeln einfach und uniform zu halten, und dafür zu akzeptieren, das konsequenterweise ab und zu mal komische Dinge passieren.

**Qualität der Anforderungen in diesem Lastenheft:** Dieses Lastenheft enthält keine absichtlich eingebauten Fehler, Widersprüche, Auslassungen etc. Es kann aber durchaus sein, dass es Stellen gibt, die mehrdeutig sind und präzisiert oder ergänzt werden müssen, oder Regeln, die in Konflikt miteinander geraten. Das ist insofern realistisch, da Kunden, vor allem in agilen Projekten, anfangs nicht mit absoluter Genauigkeit wissen, was sie wollen, und wie sie es umgesetzt haben wollen. Über die Gitlab-Issue-Funktion im Repository des Standardisierungskomitees könnt ihr auf Probleme im Lastenheft hinweisen, oder Fragen stellen, wenn es Klärungsbedarf bei der Interpretation des Lastenhefts gibt.

# 3 Funktionalität von Komponenten

Die einzelnen Komponenten des Multiplayer-Spiels müssen bestimmte Funktionalitäten zur Verfügung stellen und Anforderungen erfüllen. Dabei wird zwischen **verbindlichen Anforderungen** und **optionalen Anforderungen** unterschieden. Optionale Anforderungen und selbst erdachte Features können nach Absprache mit dem Auftraggeber freiwillig implementiert werden, dürfen aber nicht zu Inkompatibilitäten mit anderen Komponenten führen, die diese nicht beinhalten. Komponenten, Protokolle und Formate sollten gegebenenfalls so gestaltet werden, dass optionale Features einfach ignoriert werden können, sofern Daten zu diesen in Nachrichten oder Dateien enthalten sind. Die Details regelt das Standardisierungskomitee.

## 3.1 Server

**Verbindlich:** Ein Server muss nicht-interaktiv auf einem Linux-System über die Kommandozeile in Form eines Docker-Containers gestartet werden können. Dabei können in einer vom Standardisierungskomitee definierten Form die nötigen Argumente (etwa Dateien mit Content und Konfigurationen) übergeben werden.

**Verbindlich:** Der Server lädt beim Start eine Partie-Konfiguration. Alle Einstellungen für eine Partie werden in der Partie-Konfiguration gespeichert. Enthaltene Konfigurationsdaten sind bspw. erlaubte Zeitspannen für Aktionen in den Rundenphasen, Rundenanzahl bis zum Eintritt der Überlängenbedingung für eine Partie, etc.

**Verbindlich:** Der Server lädt beim Start eine Szenario-Konfiguration. Diese Beschreibung definiert das Spielfeld, auf dem die Partie stattfindet.

**Verbindlich:** Der Server lädt beim Start eine Charakter-Konfiguration, die die für Spieler verfügbaren Charaktere beschreibt.

**Verbindlich:** Der Server erlaubt genau zwei Clients (Benutzer-Clients oder KI-Clients), sich über das Netzwerk bei ihm für eine Partie als Mitspieler anzumelden.

**Verbindlich:** Sobald sich zwei mitspielende Clients beim Server registriert haben, startet der Server eine Partie und wickelt sie gemäß der Spielregeln ab.

**Verbindlich:** Der Server erlaubt Benutzer-Clients sich als Zuschauer für eine (ggf. bereits laufende) Partie zu registrieren. Sie bekommen dann den aktuellen Spielzustand und zukünftige Updates geschickt.

**Verbindlich:** Falls von einem Client erwartet wird, innerhalb einer in der Partie-Konfiguration festgelegten Zeitspanne eine Nachricht an den Server zu schicken, und der Server keine Nachricht rechtzeitig empfängt, soll der Server die Verbindung zum Client abbrechen. Im Falle eines mitspielenden Clients wird dieser damit disqualifiziert.

**Verbindlich:** Verspätet eintreffende Nachrichten für eine Rundenphase, die in einer späteren Rundenphase beim Server eingehen, sollten entsprechend vom Server nicht als Protokollverletzung betrachtet, sondern als verspätet erkannt und einfach verworfen werden.

**Verbindlich:** Falls ein mitspielender Client eine Pausierung der Partie wünscht, unterbricht der Server diese, bis irgendeiner der Mitspieler anzeigt, dass er weiterspielen möchte. KI-Clients dürfen keine Pausen verlangen.

**Verbindlich:** Falls ein mitspielender Client seine Connection zum Server verliert, so bleibt seine Session zunächst bestehen. Der Client kann sich, innerhalb einer gewissen Zeitspanne, erneut mit dem Server verbinden, und seine Session fortsetzen. Der Client bekommt dazu vom Server den vollständigen aktuellen Spielzustand geschickt.

**Verbindlich:** Falls ein Client sich nicht an das Kommunikationsprotokoll hält, oder eine nach den Spielregeln unzulässige Aktion durchführen will, sendet der Server dem Client eine Nachricht über diesen Fehler, beendet die Verbindung zu ihm und schließt ihn damit vom weiteren Verlauf der Partie aus. Im Fall eines mitspielenden Clients gewinnt dadurch der gegnerische Mitspieler.

**Verbindlich:** Der Server informiert alle Clients (Mitspieler und Zuschauer) über die Aktionen der Spieler und die

Ereignisse, die sich daraus ergeben haben, und den daraus resultierenden Spielzustand.

**Verbindlich:** Der Server überprüft zu den relevanten Zeitpunkten, ob ein Spieler gemäß der Siegbedingungen gewonnen hat. Wenn dies der Fall ist, beendet er die Partie und benachrichtigt alle Clients entsprechend.

**Optional:** Der Server schreibt alle Informationen in eine Logdatei, die zum Nachvollziehen des Partieverlaufs nötig sind. Dadurch können Clients ein Replay der Partie abspielen. Das Standardisierungskomitee legt das Format für Logs bzw. Replay-Dateien fest.

**Optional:** Der standardmäßige Port für die WebSocket-Verbindung sollte 1218 sein. Erscheint irgendwie passend...

## 3.2 Benutzer-Client

**Verbindlich:** Der Benutzer-Client kann sich über das Netzwerk bei einem Server für eine dort angebotene Partie als Mitspieler registrieren.

**Verbindlich:** Der Benutzer-Client hat einen Zuschauermodus, bei dem er sich als passiver Zuschauer bei einem Server für eine Partie registrieren, oder einer bereits laufenden beitreten kann.

**Verbindlich:** Der Benutzer-Client teilt dem Server mit, dass er ein von einem Menschen gesteuerter Client ist.

**Verbindlich:** Benutzer-Clients und KI-Clients teilen dem Server beim Verbinden einen Namen mit (Name des menschlichen Spielers, oder "team42-ki", etc.), damit man Clients (sowohl mitspielende als auch Zuschauer) auf der Benutzeroberfläche klar identifizieren kann.

**Verbindlich:** Der Benutzer-Client visualisiert das Spielgeschehen mittels einer graphischen Oberfläche. Dabei sollten die Charaktere auf der Karte unterscheidbar dargestellt werden (etwa durch individuelle Avatare, oder eingeblendete Namen), damit man als Zuschauer erkennen kann, wer die Charaktere sind, und zu welchem Spieler sie gehören (etwa mit eigener Farbe je nach Spieler).

**Verbindlich:** Für das Verständnis des Spielverlaufs notwendige Informationen (vor allem die Werte bzw. den Zustand der Charaktere) sollen auf der graphischen Oberfläche dargestellt werden. Also etwa Health Bars für Charaktere, kleine Icons für die Steine im Inventar, oder z.B. ein seitliches Panel mit den Werten für MPs, APs, und den Schadenswerten für Angriffe etc.

**Verbindlich:** Nahkampf- und Fernkampf-Angriffe werden so visualisiert, dass man erkennen kann, wer wen angegriffen hat. Evtl. sollte auch noch der gemachte Schaden erkennbar gemacht werden, etwa durch eine Änderung der Health Bar oder durch Anzeige einer Zahl über dem Angegriffenen.

**Verbindlich:** Der mitspielende Benutzer kann über die graphische Oberfläche die in den jeweiligen Phasen einer Partie möglichen Aktionen vornehmen.

**Optional:** Der Benutzer-Client kann dem Benutzer Funktionen anbieten, die ihn beim Wählen und Durchführen von Aktionen unterstützen. Bspw. kann die Künstliche Intelligenz eingebunden werden, um Vorschläge für gute Aktionen anzubieten.

**Optional:** Der Benutzer-Client kann Funktionen anbieten, um dem Benutzer eine komfortablere Bedienung bei der Eingabe von Aktionen zu ermöglichen (z.B. Hotkeys).

**Verbindlich:** Die Abwicklung der einzelnen Rundenphasen und Zugphasen werden vom Benutzer-Client animiert dargestellt. Dabei wird darauf geachtet, dass die Dauer der Animationen an die für die jeweilige Phase in der Partie-Konfiguration gewährte Zeit angepasst ist.

**Verbindlich:** Der Benutzer-Client ermöglicht dem Benutzer einen Wunsch auf Pausierung der Partie anzugeben. Wenn eine Partie pausiert ist (egal ob von einem selbst oder dem Gegner), kann ein Benutzer angeben, wenn er eine Wiederaufnahme des Spiels wünscht. Nur Benutzer-Clients dürfen eine Pausierung beantragen, KI-Clients nicht.

**Verbindlich:** Sollte die TCP-Connection, die der Websocket-Connection zugrunde liegt, abbrechen, so sollte sowohl

ein Benutzer- als auch KI-Client versuchen, wieder eine Verbindung zum Server aufzubauen, und die beim Server noch anhängige Session fortzusetzen. Gerade im WLAN kann es vorkommen, dass ein System kurzzeitig offline ist. Das sollte nicht jedesmal zu einem Ausscheiden aus einer laufenden Partie führen (wichtig für Robustheit im Turnier). Die Lebensdauer einer Session kann sich also über mehrere Reconnections erstrecken.

**Verbindlich:** Am Ende einer Partie wird der Gewinner angezeigt.

**Optional:** Am Ende einer Partie werden interessante Statistiken über den Partieverlauf angezeigt, damit man einschätzen kann, wie gut die Spieler in den verschiedenen Aspekten des Spiels waren.

**Optional:** Der Benutzer-Client kann ein Replay einer gespielten Partie aus einer von einem Server erstellten Logdatei abspielen.

**Optional:** Auf der graphischen Oberfläche ist irgendwo ein Stan Lee Easter Egg versteckt.

## 3.3 KI-Client

**Verbindlich:** Ein KI-Client muss nicht-interaktiv auf einem Linux-System über die Kommandozeile in Form eines Docker-Containers gestartet werden können. Dabei können in einer vom Standardisierungskomitee definierten Form die nötigen Argumente (IP und Port des Servers, Schwierigkeitslevel, ...) übergeben werden, damit der KI-Client sich mit einem bestimmten Server verbinden und einer Partie beitreten kann. Nach dem Start findet keine Kommunikation zwischen der KI und deren Benutzer mehr statt. Die KI spielt völlig autonom.

**Verbindlich:** Die KI teilt dem Server mit, dass sie eine KI ist.

**Verbindlich:** KI-Clients dürfen selbst keinen Wunsch auf Pausierung einer Partie stellen, müssen aber beim Spiel gegen einen von Menschen gesteuerten Benutzer-Client mit Pausierungen umgehen können. KI-Clients erlauben ihrem menschlichen Gegner eine beliebig lange Pausierung, d.h. sie sprechen nie einen Wunsch nach Wiederaufnahme der Partie aus.

**Verbindlich:** Die KI bestimmt in allen Rundenphasen/Zugphasen regelkonforme und möglichst sinnvolle Aktionen innerhalb der in der Partie-Konfiguration vorgegebenen Zeit, und teilt sie dem Server gemäß dem Nachrichtenprotokoll mit.

**Optional:** Über eine Konfigurationsdatei oder Kommandozeilenargumente können verschiedene Intelligenzstufen oder Strategien für die KI eingestellt werden.

**Optional:** Der KI-Client kann eine Schnittstelle anbieten, um in den Benutzer-Client eingebunden zu werden. Die KI-Logik kann dann benutzt werden, um dem Benutzer beim manuellen Spielen Aktionen vorzuschlagen.

## 3.4 Editor

**Verbindlich:** Das Standardisierungskomitee definiert JSON-Schemata für Szenarios, Partie-Konfigurationen und Charakter-Konfigurationen. Die mit dem Editor erstellten Ressourcen werden in diesem Format in Dateien gespeichert, und können zur weiteren Bearbeitung aus Dateien geladen werden.

**Verbindlich:** Szenarios, Partie-Konfigurationen und Charakter-Konfigurationen können über eine graphische Oberfläche erzeugt und bearbeitet werden.

**Optional:** Der Editor kann zufällig generierte gültige Szenarios erzeugen.

**Optional:** Der Editor kann die vom Benutzer geladenen oder erstellten Szenarios, Partie-Konfigurationen und Charakter-Konfigurationen validieren. Offensichtlich unzulässige oder unsinnige Werte werden werden für den Benutzer hervorgehoben.

# 4 Vorgaben zum Entwicklungsprozess

In diesem Kapitel wird beschrieben, wie das Entwicklungsprojekt ablaufen soll, und welche Regeln dabei eingehalten werden sollen. Zusätzlich sind natürlich auch die Aufgabenstellungen und Anforderungen in den Team-Meilensteinen zu beachten.

## 4.1 Begleitveranstaltung

In der Begleitveranstaltung zum Softwaregrundprojekt werden regelmäßig neue **Meilensteine** und **Aufgaben** vorgestellt und besprochen. Neben diesem Lastenheft stellen auch diese Aufgabenblätter Anforderungen des Auftraggebers an die Projektteams dar.

## 4.2 Moodle

Alle für das Softwaregrundprojekt wichtigen Informationen und Dokumente werden über **Moodle** bereitgestellt. Dazu gehören Aufgabenblätter für Team-Meilensteine, zusätzliches Material, Referenzen, usw.

Die **Mailing-Funktion von Moodle** ist der hauptsächliche Kommunikationskanal in Richtung Projektteilnehmer, also sollten alle Teilnehmer sicherstellen, dass sie Moodle-Nachrichten zeitnah empfangen und lesen. Dazu kann ggf. eine Weiterleitung per E-Mail eingerichtet werden.

Im Moodle-Kurs gibt es auch ein Forum, das alle Teams benutzen können, um Themen von gemeinsamem Interesse zu diskutieren, Termine abzustimmen, etc. Projektteilnehmer können sich auch jederzeit mit Fragen oder Vorschlägen an uns (die "Auftraggeber") wenden.

## 4.3 Git und Gitlab

Jedes Team soll alle Artefakte in seinem **Git Repository** auf der `gitlab.informatik.uni-ulm.de` archivieren. Das gilt insbesondere für Source Code und jegliche Dokumentation. Die Abgabe von Team-Meilensteine erfolgt über Commits mit entsprechenden Tags.

Die Issue-Tracker Funktion von **Gitlab** dient in der Implementierungsphase als Scrum Board.

## 4.4 Qualitätssicherung und SonarQube

Wir verwenden das Tool **SonarQube** [4] zur Analyse der Qualität von Source Code. Jedes Teammitglied sollte den von ihm neu geschriebenen Code vor dem Push ins Repo damit analysieren. Teams sollten auch regelmäßig, etwa im Sprint Review, die Qualität der kompletten Codebase des Projekts begutachten, und die Behebung erkannter Probleme als Tasks für den nächsten Sprint einplanen. Erfahrungsgemäß amortisiert sich die Zeit, die *während* der Programmierphase in das Schreiben (und auch Aktualisieren) von Tests, in automatische Code-Analyse und manuelle Reviews gesteckt wird sehr schnell, weil man sonst am Ende sehr viel mehr Zeit investieren muss um eine Menge von spät erkannten Bugs in einem riesigen Haufen von Code mit schlechter Qualität zu finden und zu beheben.

## 4.5 Docker

Um das vollständige verteilte Spiel zu verwenden, muss man meist Komponenten von verschiedenen Teams auf mehreren Plattformen zum Laufen bringen (ein Editor für Content und Configs, ein Server, zwei spielende Clients (Benutzer-Clients oder KI-Clients), potentiell mehrere Beobachter-Clients). Diese Komponenten sind oft in unterschiedlichen Sprachen und unter verschiedenen Betriebssystemen entwickelt worden. Damit sie zuverlässig und ohne aufwendige

Build-Prozesse auf ggf. anderen Plattformen ausgeführt werden können, benötigen wir eine robuste und flexible Lösung zum Paketieren dieser Komponenten. Dazu verwenden wir das Containerization-Framework **Docker** [1]. Die Komponenten, die "headless" (also ohne graphische Oberfläche) betrieben werden (Server und KI-Clients) sollen mit allen Abhängigkeiten als Docker-Images paketiert werden. Dadurch können wir diese Komponenten bei teamübergreifenden Integrationstests[4] und Turnieren flexibel auf Linux- oder Windows-Rechnern starten.

## 4.6 Tutorien und Rolle der Tutoren

Jedes Team hat ein zugeordnetes **wöchentliches Tutorium** und einen **Tutor**. Der Tutor ist gegenüber dem Team ein **Vertreter des Auftraggebers.** In den Tutorien werden die vom Tutor korrigierten Abgaben von Einzel-Übungsblättern und Team-Meilensteinen besprochen. Die Tutorien sind auch dazu da, Verständnisprobleme zu beheben, als Einzelperson oder Team Feedback zu allen Aspekten des Projekts zu bekommen, oder jegliche Herausforderungen und Schwierigkeiten zu besprechen. Für die Tutorien besteht Anwesenheitspflicht.

In der Implementierungsphase, während der ein an Scrum orientierter Entwicklungsprozess durchgeführt werden soll, übernehmen die Tutoren gegenüber dem Team auch die Rolle des **Product Owners.** Die Tutorien erfüllen dann auch die Rolle von **Sprint Meetings.** Auch dafür besteht Anwesenheitspflicht.

## 4.7 Agiler Entwicklungsprozess

Im Softwaregrundprojekt wird während der Implementierungsphase ein an **Scrum** orientierter agiler Prozess durchgeführt. Der Tutor übernimmt die Rolle des **Product Owners.** Es finden **Sprints** mit Dauer von zwei Wochen statt. Entsprechend gibt es **Sprint Planning, Sprint Review und Sprint Retrospective Meetings.**

Typischerweise findet an einem Tutoriumstermin zuerst Review und Retrospective für den vergangenen Sprint statt, und dann Planning für den kommenden. Dennoch sollte man diese Arten von Meetings mit ihrem jeweils eigenen Inhalt und Ziel nicht vermischen, sondern gedanklich klar trennen, und sauber der Reihe nach durchführen. Wir legen dabei Wert auf eine ordentliche Einhaltung des Scrum-Prozesses. Dazu gehört auch ein sinnvoller Umgang mit zentralen Artefakten wie z.B. dem Scrum-Board.

"Agil" heißt nicht, dass man unstrukturiert von sich hinprogrammiert und einfach irgendwas macht. Bei diesem Projekt soll am Ende nicht nur ein irgendwie zusammengehacktes und vielleicht sogar einigermaßen funktionierendes Produkt herauskommen. Es soll stattdessen im Rahmen eines geregelten Entwicklungsprozesses mit funktionierendem Projektmanagement eine zuverlässig funktionierende und die Anforderungen erfüllende Anwendung mit hoher Qualität erstellt werden.

## 4.8 Dokumentation

Jedes Team führt ein gemeinsames **Projekttagebuch.** Darin trägt jedes Team-Mitglied für sich selbst (!) alle Tätigkeiten und den dazugehörenden Zeitaufwand ein, den es für das Projekt erbracht hat. Dazu gehören Zeiten fürs Besuchen von Tutorien, den später stattfindenden Sprint Meetings, sonstige Besprechungen im Team, das Bearbeiten von Team-Meilensteinen, Aufwand für selbstständiges Einarbeiten in Technologien und Frameworks, Implementierungstätigkeiten, Zeiten für die Erstellung von Dokumenten – einfach alles was ihr für das Team-Projekt im Softwaregrundprojekt macht. Die Zeiten für das Besuchen der Softwaretechnik-Vorlesung und der Sopra-Begleitveranstaltung sollen jedoch nicht im Projekttagebuch eingetragen werden. Eine Vorlage für das Projekttagebuch wird im Moodle bereitgestellt.

Die Teams verfassen auch ein **Entwicklerhandbuch** zum Produkt. Es soll die Architektur und die implementierte Funktionalität beschreiben. Listet darin auch auf, welche Technologien und Frameworks ihr benutzt, und die Gründe dafür. Haltet außerdem alle wichtigen Entscheidungen, die im Team oder gemeinsam mit dem Tutor getroffen werden, fest. Besonders für die auf der Messe gehandelten Komponenten ist ein ordentliches Entwicklerhandbuch ein gutes Verkaufsargument, bzw. umgekehrt etwas, auf das man als Käufer achten sollte.

[4] a.k.a. LAN-Parties

Selbstverständlich muss auch der geschriebene Source Code ausführlich **kommentiert** werden. Maßnahmen zur **Qualitätssicherung** wie Code-Analyse, Reviews oder Tests müssen auch nachvollziehbar dokumentiert werden.

Für die Komponenten des Spiels soll jeweils ein **Benutzerhandbuch** erstellt werden, das ihre Funktionalität und Verwendung erklärt.

All dies ist besonders für die auf der Messe verkaufte Komponente wichtig, um den Käufer-Teams eine leichte Einarbeitung und Weiterentwicklung zu ermöglichen. Alle Dokumente sollen in sinnvoller Verzeichnisstruktur im Git Repository des Teams abgelegt werden.

## 4.9 Standardisierungskomitee

*Marvelous Mashup* ist ein verteiltes System. Das stellt Anforderungen an die Interoperabilität der verschiedenen Komponenten, wie Benutzer-Clients, KI-Clients, Server und Editoren. Jeder Client muss mit jedem Server über das Netzwerk kommunizieren können. Alle Clients und Server müssen von beliebigen Editoren erstellten Content und Configs verarbeiten können.

Das bedeutet, um *Marvelous Mashup* betreiben zu können, müssen ein gemeinsames **Nachrichtenprotokoll** und gemeinsame **Formate für Szenarios, Charakter-Konfigurationen und Partie-Konfigurationen** definiert werden, die von allen Teams in ihren Komponenten gleich implementiert werden. Um das zu erreichen, wird ein teamübergreifendes **Standardisierungskomitee** gebildet. Jedes Team bestimmt unter seinen Mitgliedern einen **Standardisierungsbeauftragten,** der in das Standardisierungskomitee entsandt wird. Für die Sitzungen des Komitees besteht Anwesenheitspflicht. Falls der Beauftragte eines Teams verhindert ist, so liegt es in seiner Verantwortung einen Vertreter zu schicken.

Beim konstituierenden Treffen des Komitees wählen die Ehrenwerten Mitglieder einen Sehr Ehrenwerten **Vorsitzenden.** Der Vorsitzende hat die Aufgabe, Termine für die Sitzungen des Komitees zu planen, zu besagten Sitzungen einzuladen, und selbige zu leiten. Bei jeder Sitzung wird ein **Protokollführer** bestimmt, der den Ablauf der Sitzung und die dort getroffenen Entscheidungen schriftlich festhält. Die Sitzungsprotokolle und alle Standardisierungsdokumente werden den Teams über ein Git Repository[5] des Komitees zur Verfügung gestellt.

Um bestimmte, abgegrenzte Themen zu diskutieren oder Entwürfe für Protokolle und Formate zu erstellen, kann das Standardisierungskomitee kleinere **Arbeitsausschüsse** einrichten. Die von den Ausschüssen erarbeiteten Entwürfe werden auf Sitzungen des ganzen Komitees zur **Abstimmung** gestellt, und werden durch Mehrheitsbeschluss angenommen. Der Auftraggeber hat dabei eine beratende Rolle und ein Vetorecht.

Die vom Standardisierungskomitee definierten Kommunikationsprotokolle und Dateiformate werden in **Standardisierungsdokumenten** beschrieben, die nach Verabschiedung im Komitee den Teams in der Implementierungsphase zur Verfügung gestellt werden. Falls es durch geänderte Anforderungen oder durch Erkenntnisse bei der Implementierung nötig wird, die Standards anzupassen, zu erweitern, oder Fehler und Widersprüchlichkeiten zu beheben, soll das Komitee diese Dokumente zeitnah überarbeiten.

## 4.10 Messe

Jedes Team implementiert einen **Benutzer-Client** und einen **KI-Client.** Von den übrigen beiden Komponenten, **Server** und **Editor,** wählt jedes Team *eine* aus, die es implementiert. Die andere, fehlende Komponente, die noch benötigt wird, um das komplette Multiplayerspiel zu haben, "kaufen" die Teams dann auf einer Messe von einem anderen Team ein.

Die **Messe** wird ungefähr zur Halbzeit der Implementierungsphase stattfinden. Jedes Team muss bis dahin seinen Server bzw. seinen Editor als fertiges, lauffähiges Package haben. Dazu gehören auch eine umfassende Dokumentation (Benutzer-/Entwicklerhandbücher, Entwurfsdiagramme, etc.) sowie Unit Tests. Auf der Messe präsentiert dann jedes Team an seinem Stand seine Komponente den anderen Teams, denen diese Komponente noch fehlt. Für jede "verkaufte"

[5] `gitlab.informatik.uni-ulm.de/sopra/ws20-marvelous-mashup/standard`

Lizenz bekommt das Team entsprechend Punkte für den Highscore gutgeschrieben.

Umgekehrt muss sich jedes Team die ihm noch fehlende Komponente unter den von anderen Teams angebotenen aussuchen. Nach dem "Verkauf" geht die Komponente mit Source Code und aller Dokumentation, Tests, usw. in den Besitz des Käuferteams über, das ab da ganz allein für die Wartung und ggf. Weiterentwicklung verantwortlich ist. Das bedeutet, das Käuferteam macht einen "hard fork". Da die Teams also mit gekauften Komponenten selbst weiterarbeiten müssen, sollte beim "Kauf" entsprechend Augenmerk darauf gelegt werden, eine gut designte, wart- und erweiterbare Komponente mit hoher Testabdeckung und guter Dokumentation zu erwerben. Die Programmiersprache sollte natürlich auch eine sein, mit der das erwerbende Team vertraut ist.

Da wir *Marvelous Mashup* agil entwickeln, kann es durchaus sein *(*räusper*)*, dass dem Auftraggeber nach der Messe noch ein paar neue Anforderungen einfallen. Das sind typischerweise zusätzliche Features für das Spiel, die nichts "kaputtmachen", also nicht dazu führen, dass bisher entwickelte Funktionalität weggeworfen werden muss, sondern ein paar Kleinigkeiten, die sich bei gutem Design leicht in das einbetten lassen, was bisher entwickelt wurde. Die neuen Features können aber durchaus einen "cross-cut-Charakter" haben, also Erweiterungen in allen Komponenten (auch der von einem anderen Team gekauften) erfordern.

## 4.11 Abschlussturnier

Am Ende des Projektzeitraums, meist in der letzten Vorlesungswoche, wird ein **Abschlussturnier** stattfinden, in dem die **KI-Clients** der Teams gegeneinander antreten. Das Turnier läuft in zwei Phasen ab. Zuerst findet eine Vorauswahl statt, um die besten KI-Clients zu bestimmen. Danach treten diese in einem öffentlichen Abschlussturnier in Finalrunden im K.O.-System gegeneinander an. Die Partien werden vom Publikum über Benutzer-Clients im Zuschauer-Modus verfolgt. Dafür werden die schönsten Benutzer-Clients von den Tutoren ausgewählt. Das Turnier ist eine Gelegenheit für die Teams, ihr Produkt vor einem größeren Publikum zu präsentieren, und in einem spannenden Wettbewerb ihre geistigen Kräfte und Coding Skills aneinander zu messen. Für Studierende aus niedrigeren Semestern, die das Softwaregrundprojekt später selbst noch absolvieren müssen, ist das Turnier ebenfalls interessant, um einen Eindruck vom Ergebnis des Projekts bekommen.

## 4.12 Highscore und Preise

Während des Softwaregrundprojekts wird ein **Highscore** geführt. Jedes Team bekommt Highscore-Punkte für die Leistung seiner Mitglieder bei Team-Meilensteinen, entsprechend der Bewertungen durch die Tutoren. Es gibt auch Punkte für die auf der Messe "verkauften" Lizenzen.

Darüberhinaus behalten wir uns vor, bei herausragenden Leistungen oder besonderem Engagement im Projekt Achievements zu vergeben, die mit zusätzlichen Punkten in den Highscore eingehen.

Am Ende des Softwaregrundprojekts werden zwei Preise vergeben:

- Der **Highscore-Preis** für die beste Leistung über die gesamte Projektdauer hinweg bei Abgaben und sonstigen Aufgaben.
- Der **Turnier-Preis** für den Sieg im Abschlussturnier.

Es kann auch sein, dass wir Sonderpreise in einigen anderen Kategorien an Teams vergeben, die in bestimmten Aspekten des Projekts herausragende Leistungen erbracht haben.

Den Gewinnerteams der Preise winkt ewiger Ruhm und vielleicht sogar Reichtum...

# 5 Kriterien für die Abnahme

Um als Team eine erfolgreiche Projektabnahme zu erreichen und für das individuelle Bestehen des Softwaregrundprojekts müssen folgende Bedingungen erfüllt sein:

- Fristgerechte und korrekte Abgabe aller Team-Meilensteine mit ausreichender Bewertung.
- Fristgerechte Abgabe und Bestehen von mindestens $n-1$ der $n$ Einzelübungsblätter.
- Regelmäßige Anwesenheit und Mitarbeit im Tutorium.
- Aktive Mitarbeit im Team. Dies wird durch das Projekttagebuch dokumentiert.
- Eigene Programmiertätigkeit in ausreichendem Umfang.
- Fristgerechte erfolgreiche Abnahmesitzung zum Ende des Projekts. Dazu gehört:
  - Vorlage einer lauffähigen Version der Software.
  - Umsetzung der Kernfunktionalität, d.h. aller verbindlichen Anforderungen.
  - Gute Qualität der entwickelten Software, mit sinnvoller Testabdeckung und nachgewiesen durch umfangreiche und dokumentierte Maßnahmen zur Qualitätssicherung.
  - Einhaltung des Scrum-Entwicklungsprozesses und sinnvolle Verwendung von Team-Repository und Scrum-Board im Gitlab.
  - Ordentliche Dokumentation der Software und des Projektverlaufs. Das umfasst insbesondere die zum Scrum-Prozess gehörenden Dokumente (Scrum-Board etc.).
- Anwesenheit beim Abschlussturnier.

## 5.1 Abnahmesitzung

Am Ende des Sommersemesters 2021 findet für jedes Team eine Abnahmesitzung statt, bei der die entwickelte Anwendung dem Auftraggeber präsentiert und ihre Features demonstriert werden. Dabei soll jedes Teammitglied seinen persönlichen Anteil am Ergebnis präsentieren.

Beim Abnahmetreffen werden auch die geforderten Dokumente abgegeben.

# A Beispiele für Charaktere aus der Charakter-Konfiguration

Hier ist eine beispielhafte Auswahl an Charakteren mit zugewiesenen Eigenschaften, wie sie in einer Charakter-Konfiguration stehen könnten.

- **Rocket Raccoon:** Knuffig und flauschig, aber mit Raketenwerfer.
  - HP: 100
  - MP: 2
  - AP: 2
  - Nahkampf: Schaden 10
  - Fernkampf: Schaden 30, Reichweite 5
- **Pietro Maximoff a.k.a. Quicksilver:** Verdammt flink.
  - HP: 100
  - MP: 6
  - AP: 1
  - Nahkampf: Schaden 10
  - Fernkampf: Schaden 10, Reichweite 3
- **Dr. Bruce Banner a.k.a. The Incredible Hulk:** Schwerfällig, aber gut im smashen.
  - HP: 300
  - MP: 1
  - AP: 2
  - Nahkampf: Schaden 40
  - Fernkampf: Schaden 20, Reichweite 3
- **Natalia Alianovna Romanoff a.k.a. Black Widow:** Agil und geschickt im Nahkampf.
  - HP: 200
  - MP: 3
  - AP: 3
  - Nahkampf: Schaden 20
  - Fernkampf: Schaden 10, Reichweite 3
- **Clinton Francis Barton a.k.a. Hawkeye:** Exzellenter Bogenschütze.
  - HP: 100
  - MP: 1
  - AP: 3
  - Nahkampf: Schaden 10
  - Fernkampf: Schaden 30, Reichweite 6

Die Charaktere haben also unterschiedliche Stärken und Schwächen. Es gibt etwa solche, die unbeweglich sind, aber gut im Nahkampf, oder solche, die ihre Stärke im Fernkampf haben, oder solche, die nicht viel Schaden machen, aber sehr mobil sind, was gut ist, um Steine aufzusammeln, etc.

Die Idee hier ist, dass man beim Design der Charaktere das Schere-Stein-Papier Prinzip beachtet. Wenn ein Charakter in einem der Aspekte Beweglichkeit, Nahkampf oder Fernkampf überragend gut ist, sollte er dafür in einem anderen eher schlecht sein. Kein Charakter sollte übermächtig sein, aber eben auch keiner vollkommen nutzlos.

Für das Balancing des Spiels hat man also mehrere Stellschrauben. Wenn der Spielverlauf zu zäh ist, kann man die Movement Points und Action Points höher setzen, sodass die Charaktere pro Zug weiter laufen und mehr Aktionen unternehmen können. Falls nach zwei Runden die meisten Charaktere schon ausgeknockt sind, setzt man entsprechend die Schadenswerte herunter. Viele APs, d.h. viele Angriffe pro Zug mit wenig Schaden oder lieber wenig Angriffe mit viel Schaden? Beim Fernkampf große Reichweite aber wenig Schaden, oder umgekehrt? Wenn die Fähigkeit eines Infinity Stones sich als total overpowered herausstellt, dann setzt man eben die Cooldown Zeit hoch, damit er nur selten verwendet werden kann.

Außerdem beeinflusst das Design des Szenarios den Spielverlauf und die relative Stärke der Charaktere: ein offenes Feld begünstigt Charaktere mit großer Fernkampf-Reichweite, ein Spielfeld mit vielen Felsen führt umgekehrt dazu, dass Fernkämpfer weniger wertvoll sind, weil die Gegner meist durch Felsen gedeckt sind. Längere Wege um Hindernisse herum machen dafür Charaktere mit vielen MPs wertvoller, etc.

Der Vorteil, alle diese Parameter in Konfigurationsdateien ausgelagert zu haben, besteht darin, dass man nachträglich noch viel Freiraum hat, um zu beeinflussen, wie sich das Spiel dann letztendlich anfühlt. Änderungen an diesen Werten haben aber keinen Einfluss auf den Code, der bleibt stabil und muss nicht jedes Mal geändert werden, wenn man am Balancing des Spiels herumprobiert.

Wir werden also erstmal ein paar Spiele mit verschiedenen Szenarien und Charakter-Konfigurationen spielen müssen, bis sich sinnvolle Werte für Charakter-Stats und die Partie-Konfiguration herauskristallisieren...

## Literatur

[1] Docker, Inc. Docker. `https://www.docker.com/`. [Online; accessed 2019-11-20].

[2] IETF. RFC 6455: The WebSocket Protocol. `https://tools.ietf.org/html/rfc6455`. [Online; accessed 2019-11-20].

[3] IETF. RFC 8259: The JavaScript Object Notation (JSON) Data Interchange Format. `https://tools.ietf.org/html/rfc8259`. [Online; accessed 2019-11-20].

[4] SonarSource S.A. SonarQube. `https://www.sonarqube.org/`. [Online; accessed 2019-11-20].