Erstellt am: 02.08.2016

Dozent: Dr. Karsten Hölscher
Tutor: Dr. Karsten Hölscher

Testprotokoll

zu

Gold & Greed

von der Gruppe:

ProjectGG

Mitglieder:

Fabian Schneekloth

Benjamin Brennecke

Inhaltsverzeichnis

1. Einführung zum Protokoll

Das Protokoll gibt in den unterschiedlichen Testverfahren an, was getestet wurde und welches Ergebnis dabei erzielt wurde. Hierbei werden Datum und Verantwortlicher dokumentiert und eine Analyse der dabei erzielten Testabdeckung geliefert. Bei den Systemtests werden die Funktionstests priorisiert, da sie ausschlaggebend für das Bestehen dieses Projektes sind. Weitere Systemtests werden nicht spezifisch beschrieben, da sie teilweise als Produkt von Integrationstests, Akzeptanztests oder Spezialfällen entstehen und in diesen erwähnt werden, wenn sie als wichtig erachtet werden.

2. Systemüberblick

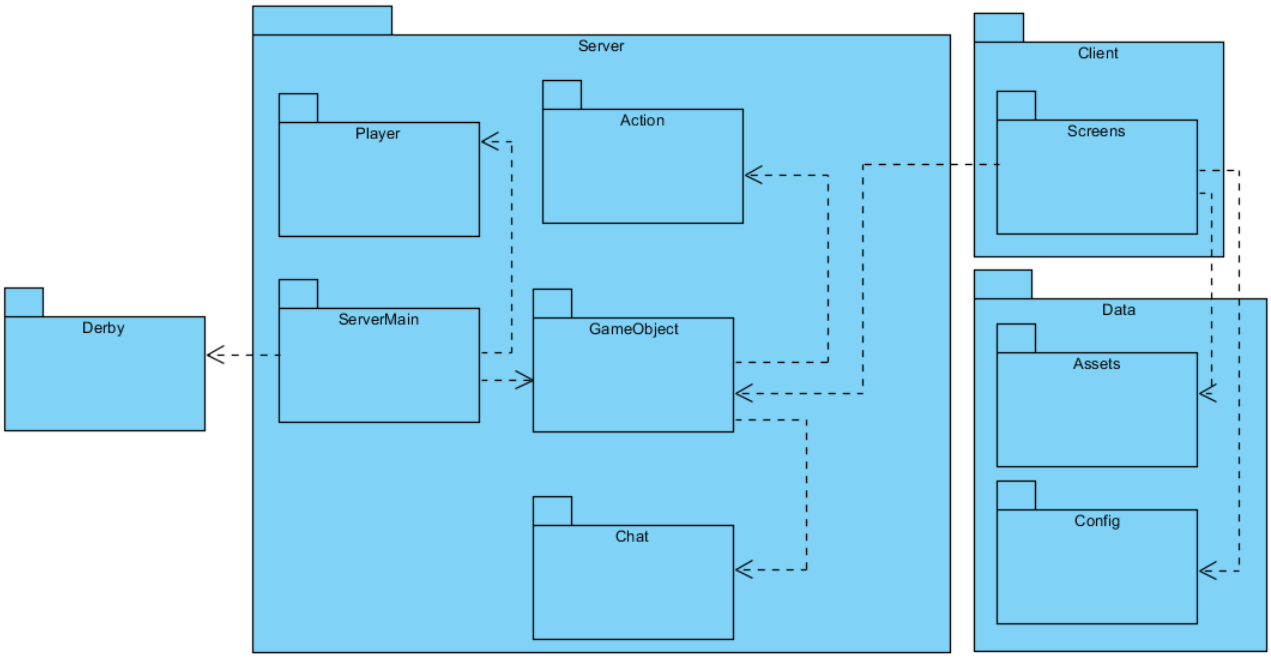


Abb. 1: Paketdiagramm aus der Architekturbeschreibung zu Gold & Greed

Wie in der Abbildung 1 zu sehen, besteht unser System aus vier großen Komponenten, der Datenbank, dem Server, dem Client und Data. Das Paket Data enthält keine zu testenden Klassen und wird dabei ignoriert. Wir testen bei den

Integrationstests mit dem Top-Down-Verfahren, wir beginnen also bei den Screens im Client und durchlaufen über diese alle Komponenten bis zur Datenbank. Die spezifischen Klassen, die getestet werden, sind aus der Architekturbeschreibung zu entnehmen, da diese im Test-Protokoll nicht weiter beschrieben werden.

3. Komponententests

Die Komponententests werden durch JUnit-Tests realisiert. Die Testabdeckung messen wir in den möglichen Rückgabewerten der Methoden, wobei wir hierbei vereinfachen müssen, da Methoden ohne primitive Rückgabewerte nicht auf jeden möglichen Rückgabewert getestet werden können. Wir prüfen hierbei also nur, ob das Ergebnis korrekt ist, dies gilt auch bei Methoden ohne Rückgabewert oder Methoden die Zahlenwerte zurückgeben.

Betrachtet man die existenten Komponententests stellt man fest dass diese durch Kommentare "deaktiviert" wurden, der Grund dafür ist, dass sie auf einer leicht veralteten Version des Spieles liefen, in der das ID-System noch nicht vorhanden war. Dadurch haben sich die für Konstruktoren benötigten Kontexte geändert, da seit dem Einführen des ID-Systems jedes Objekt beim Erstellen eine GameSession benötigt, außerdem fielen viele mögliche RemoteExceptions weg, weshalb die Tests angepasst werden müssten. Da diese Änderungen jedoch nicht die interne Logik der getesteten Klassen betreffen(abgesehen von den Konstruktoren, aber auch hier nur auf trivialer Ebene) und die Tests bereits vorher erfolgreich waren haben wir uns, auch aus Zeitmangel, dazu entschieden die entsprechenden Tests nicht zu aktualisieren.

Insgesamt gibt es jedoch recht wenige Komponententests, die Ursache dafür ist das viele Klassen entweder aus größtenteils trivialen Methoden bestehen oder ohnehin mit visuellem Feedback getestet werden mussten, da die Tests selbst nicht simpel genug gehalten werden könnten um ihre Korrektheit zu garantieren(Beispielmethode: Map.init()). Einen Großteil machen dabei viele der GameSession Methoden aus, da diese lediglich die Kommunikation zwischen Client und Server ermöglichen und durch das ID-System nur schwer in Komponententests zu testen sind. Zusätzlich haben wir durch die Funktionstest ein akzeptables Ergebnis erhalten, weshalb wir uns entschieden haben die Komponententests größtenteils aus zu lassen.

3.1 Ablauf des Verfahrens

Wir haben für alle relevanten Klassen des Systems, JUnit-Tests erstellt, die wir wie folgt abarbeiten: Zuerst wird nach Komponenten gegliedert und dann nacheinander

jede dazugehörige Klasse und die dazugehörige Test-Klasse erwähnt. Daraufhin wird jede ihrer zu testenden Methoden nach dem Schema der folgenden Tabelle bearbeitet:

Methodenname	Testmethodenname	Eingabeparameter	Ausgabeparameter	Erfolgreich	Abdeckung %

Zuerst wird also der Name der zu testenden Methode beschrieben, dann der Name der Testmethode, ihre Eingabeparameter und Ausgabeparameter, falls vorhanden und dann ob der Test erfolgreich war, also die Methode ihre Funktionalität erfüllt hat. Die letzte Spalte wird wie folgt berechnet: Wir nehmen alle zu testenden Ausgabeparameter der Methode und prüfen wie viele die Testmethode abdeckt. Dies ergibt dann den prozentualen Anteil an den möglichen Ausgabewerten. Dies ist natürlich nicht sofort ersichtlich, wenn man nicht den Rumpf der Testmethode betrachtet. Dies wird jedoch nicht genauer erwähnt, da dies eine Darstellung im Protokoll unnötig verkomplizieren würde. Wir müssen daher immer davon ausgehen, das die JUnit-Tests selbst ihre Funktionalität erfüllen und bei der Abdeckung ein realistischer Wert eingetragen wurde. Außerdem werden die Methoden durch die Integrationstests und Funktionstests zusätzlich getestet, durch diese erreichen wir eine noch höhere Abdeckung der möglichen Eingabe- und Ausgabeparameter.

3.2 Tests

Komponente: Action

Klasse: Buff

Testklasse: BuffTest

Methodenname	Testmethodenname	Eingabeparameter	Ausgabeparameter	Erfolg-reich	Abdeckung %
`Buff()`	`testeAusfuehren()`	`Unit,Unit, Player,BuffInfo`	Buff	Ja	100,00%
`appliesForUnit()`	`testeGiltFuerUnit()`	`Unit`	Boolean	Ja	100,00%

Klasse: EmpowerShield

Testklasse: EmpowerShieldTest

`EmpowerShield()`	`testeAusfuehren()`	`Hero,Player, Player`	EmpowerShield	Ja	100,00%

Klasse: Heal

Testklasse: HealTest

Heal()	testeAusfuehren()	Unit,Unit, Player,BuffInfo	Heal	Ja	100,00%

Klasse: ReduceUnitCosts

Testklasse: ReducedUnitCostTest

ReduceUnitCosts()	testeAusfuehren()	Unit, Unit, Player	ReduceUnitCosts	Ja	100,00%

Klasse: Shield

Testklasse: ShieldTest

Shield()	testeAusfuehren()	Hero,Unit, Player	Shield	Ja	100,00%

Komponente: Chat

Klasse: Chat

Testklasse: ChatTest

addParticipant()	testTeilnehmerHinzufuegenUndEntfernen()	Player	-	Ja	50,00%
addParticipant()	testTeilnehmerHinzufuegenNull()	Null	IllegalArgumentException	Ja	50,00%
blockPlayer()	testTeilnehmerBlockieren()	Player	-	Ja	100,00%

unblockPlayer()	testTeilnehmerFreigeben()	Player	-	Ja	100,00%
addParticipant()	testBlockiertenTeilnehmerHinzufuegen()	Player	IllegalArgumentException	Ja	50,00%
addMessage()	testNachrichtHinzufuegen()	Player,String	-	Ja	50,00%
addMessage()	testNachrichtHinzufuegenNull()	Player,String	IllegalArgumentException	Ja	50,00%
addMessage()	testNachrichtHinzufuegenKeinTeilnehmer()	Player,String	IllegalArgumentException	Ja	50,00%
clear()	testLeeren()	-	-	Ja	100,00%

(addMessage, addParticipant wurden zu 100% abgedeckt)

Klasse: Message

Testklasse: MessageTest

makeVisibleFor()	testeSichtbarMachen()	Player	-	Ja	50,00%

(Der Fall für null als Eingabeparameter ergibt die restlichen 50%)

Komponente: GameObject

Klasse: Market

Testklasse: MarketTest

buy()	testKaufen()	String, int, int	True	Ja	50,00%
buy()	testKaufenNichtGenugGold()	String, int, int	False	Ja	50,00%
buy()	testKaufenNichtGenugHolzVorhanden()	String, int, int	False	Ja	50,00%
buy()	testKaufenNichtGenugEisenVorhanden()	String, int, int	False	Ja	50,00%

`sell()`	`testVerkaufen()`	`String, int, int`	True	Ja	50,00%
`sell()`	`testVerkaufenNichtGenugHolz()`	`String, int, int`	False	Ja	50,00%
`sell()`	`testVerkaufenNichtGenugEisen()`	`String, int, int`	False	Ja	50,00%

(buy und sell wurden zu 100% abgedeckt)

Komponente: Player

Klasse: Player

Testklasse: PlayerTest

addPermaBuff(),addTemporaryBuff()	testBuffHinzufuegen()	Buff	-	Ja	50,00%
addPermaBuff(),addTemporaryBuff()	testBuffHinzufuegenNull()	Buff	-	Ja	50,00%
advanceOnTechTree()	testAufStahlZweigVoranschreiten()	TreeElement	-	Ja	33,33%
advanceOnTechTree()	testAufMagieZweigVoranschreiten()	TreeElement	-	Ja	33,33%
advanceOnTechTree()	testAufKulturZweigVoranschreiten()	TreeElement	-	Ja	33,33%

(Die Methoden: addPermaBuff(),addTemporaryBuff(),advanceOnTechTree haben eine 100% Abdeckung)

Komponente: Server

Klasse: DBManager

Testklasse: DBManagerTest

registerAccount(), checkAccount()	RegisterAndCheckAccountTest()	-	-	Ja	100,00%
saveSession(),loadSession()	saveAndLoadGameSessionTest()	-	-	Ja	100,00%
saveSession(),loadSession()	updateAndLoadGameSessionTest()	-	-	Ja	100,00%

3.3 Testabdeckung

Zu der Testabdeckung ist noch zu sagen, das diese ziemlich gering ist, da wir nur Äquivalenzfälle betrachten und nicht auf die vielen möglichen Eingabeparameter testen. Da nicht alle Klassen durch JUnit getestet worden sind, nimmt diese noch zusätzlich ab. Pauschal würden wir diese Testabdeckung als vernachlässigbar bezeichnen, also unter 5%, jedoch ist sie trotzdem wichtig, da es sich um essentielle Funktionen des Spiels handelt und diese Tests schnell durchgeführt werden können.

4. Integrationstests

Wir prüfen hierbei das Zusammenspiel verschiedener Komponenten, wobei wir diese immer Top-Down testen. Die Testfälle werden durch Anwendungsfälle bestimmt, da sie über die Screens durchgeführt werden.

4.1 Ablauf des Verfahrens

Da wir über die Screens testen, beschäftigen wir uns nicht spezifisch mit allen Komponenten die abgedeckt werden, sondern eher mit den Ausgaben, die wir durch das System erhalten. Wir nehmen uns also einen Anwendungsfall vor und versuchen diesen mit unserem System zu rekonstruieren. Die dabei durchgeführten Aufrufe werden schrittweise dokumentiert und am Ende des Tests, wird geprüft, ob der gewünschte Zustand erreicht worden ist. Wir durchlaufen dabei natürlich viele Komponenten, die wir auch aufzählen werden. Die genauen Methodenaufrufe werden hierbei nicht beschrieben, da sie das Ergebnis unnötig verkomplizieren und bei einfachen Fällen schon den Aufwand für das Testen überschreiten.

Wir geben die Struktur wie folgt vor:

Testfallbezeichner : Funktion	
Eingabe:	
Ausgabe:	
Vorbedingungen:	
Durchlaufene Module:	
Schweregrad bei Fehlfunktion:	

Als Hilfsmittel für folgende Struktur, haben wir uns nach dem Testplan der Code:Breakers aus dem letzten Semester orientiert. Diese haben eine ähnliche Struktur für ihre Funktionstests angegeben. Wir haben jedoch, um eine gewisse Form der Integration zu dokumentieren, eine Zeile eingefügt, in der die Module eingetragen werden, die beim Testen durchlaufen werden.

Da es sich hierbei prinzipiell auch um Funktionstests handeln soll, die jedoch eher dafür vorgesehen sind, einzelne Komponenten zusammen zu testen, haben wir uns für den Bezeichner: Komponententest entschieden, da wir unsere Funktionstests anders gliedern werden. Diese sind bei uns nämlich nur für die Abdeckung der Mindestanforderungen vorgesehen.

4.2 Tests

Testfallbezeichner : Funktion	T1: Spiel wird erstellt
Eingabe:	Im MenuScreen wird: „Neues Spiel“ angeklickt und dann im InitScreen wird „Bestätigen“ gedrückt.
Ausgabe:	Eine neue GameSession wurde vom Server erstellt.
Vorbedingungen:	Datenbank wurde erstellt.
Durchlaufene Module:	MenuScreen, InitScreen, Server, GameSession, DBManager
Schweregrad bei Fehlfunktion:	Mittel, da ein Demo-Spiel automatisch erzeugt wird, was genutzt werden kann.

Testfallbezeichner : Funktion	T2: Team wird gewählt und Spiel wird gestartet
Eingabe:	Im InitScreen wird ein Team ausgewählt und auf „Spielen “ gedrückt.
Ausgabe:	Spieler wurde in der GameSession einem Team zugeordnet und der GameScreen wird aufgerufen.
Vorbedingungen:	T1
Durchlaufene Module:	Module wie bei T1 + GameScreen und alle enthaltenen Klassen
Schweregrad bei Fehlfunktion:	Hoch, da es eine Mindestanforderung ist, das ein Team gewählt werden kann.

Testfallbezeichner : Funktion	T3: Spielfigur Fähigkeiten werden aktiviert
Eingabe:	Spielfigur Fähigkeit: „Linke Hand“ wird aktiviert und die Fähigkeit „Rechte Hand“ wird aktiviert.
Ausgabe:	Es wird die Heal-Fähigkeit für den Helden aufgerufen und die Shield-Fähigkeit
Vorbedingungen:	Wie bei T2
Durchlaufene Module:	Wie bei T2+Buff,Shield,Heal
Schweregrad bei Fehlfunktion:	Hoch, da Mindestanforderung

Testfallbezeichner : Funktion	T4: Spielfigur Fähigkeit ändert sich
Eingabe:	Spielfigur Fähigkeit: „Linke Hand“ wird aktiviert, während sie in der Nähe einer eigenen Einheit steht.
Ausgabe:	Die Fähigkeit der linken Hand ändert sich zu einer, die zusätzlich die eigene Einheit heilt.
Vorbedingungen:	Wie bei T2 und Hero musste neben eine eigene Einheit bewegt werden.
Durchlaufene Module:	Wie bei T2
Schweregrad bei Fehlfunktion:	Hoch, da Mindestanforderung

Testfallbezeichner : Funktion	T5: Abbau von Holz
Eingabe:	Die Einheit: „Worker“ wird neben einen Wald bewegt
Ausgabe:	Der Einheit wird Holz gutgeschrieben.
Vorbedingungen:	Wie bei T2 + Bau einer Worker Einheit + Nächster Zug.
Durchlaufene Module:	Wie bei T2 + Worker
Schweregrad bei Fehlfunktion:	Hoch - Mindestanforderung

Testfallbezeichner : Funktion	T6: Einsammeln von Mana
Eingabe:	Mana wird von Einheit auf dem Spielfeld eingesammelt.
Ausgabe:	Der Besitzer der Einheit bekommt das eingesammelte Mana gutgeschrieben.
Vorbedingungen:	T2, Einheit die sich bewegen kann und Mana das im Bewegungsumfeld dieser Einheit gespawnt ist.
Durchlaufene Module:	Wie bei T2+ SpearFighter
Schweregrad bei Fehlfunktion:	Mittel – Da Mana auch anderweitig produziert werden kann, nämlich durch ein Labor.

Testfallbezeichner : Funktion	T7: Basis - Erhalten von Gold
Eingabe:	Der eigene Zug beginnt, es gibt also keine Eingabe.
Ausgabe:	Gold erhöht sich je nach Anzahl der eingenommen Basen und durch eine Marktplatz noch zusätzlich.
Vorbedingungen:	T2 und Marktplatz wurde gebaut, der eigene Zug beginnt.
Durchlaufene Module:	Wie bei T2
Schweregrad bei Fehlfunktion:	Hoch - Mindestanforderung

Testfallbezeichner : Funktion	T8: Basis – Bau von Kaserne und Einheit
Eingabe:	Eigene Basis wird angeklickt, Button: „Kaserne bauen“ wird angewählt, dann wird auf den Button: „Kaserne gedrückt“ und das Symbol mit den Schwertern angeklickt.
Ausgabe:	Eine SwordFighter-Einheit sollte nun erstellt worden sein.
Vorbedingungen:	T2
Durchlaufene Module:	Wie T2 + SwordFighter
Schweregrad bei Fehlfunktion:	Hoch - Mindestanforderung

Testfallbezeichner : Funktion	T9: Basis – Erforschung neuer Technologie
Eingabe:	Technologiebaum wird ausgewählt, Der Eisen-Zweig bei Stufe 1 wird ausgewählt.
Ausgabe:	Die Technologie wurde erforscht.
Vorbedingungen:	Wie T2 + genug Ressourcen sind vorhanden.
Durchlaufene Module:	Wie T2 + TechTree
Schweregrad bei Fehlfunktion:	Hoch - Mindestanforderung

Testfallbezeichner : Funktion	T10: Basis – Eroberung
Eingabe:	Einheit wird vor gegnerische Basis bewegt, diese kämpfen.
Ausgabe:	Gegnerische Basis wird besiegt und eingenommen.
Vorbedingungen:	Wie bei T2+ Einheit, die sich zu einer gegnerischen Basis bewegen kann und diese bei einem Kampf besiegt.
Durchlaufene Module:	T2
Schweregrad bei Fehlfunktion:	Hoch - Mindestanforderung

Testfallbezeichner : Funktion	T11: Basis – Labor und Marktplatz
Eingabe:	Eigene Basis wird ausgewählt und es wird auf den Button: „Labor bauen geklickt“ und auf den Button: „Marktplatz bauen“. Beim Marktplatz werden nun Ressourcen verkauft.
Ausgabe:	Beide Erweiterungen sollten erstellt und die Kosten für diese, von den Ressourcen des Spielers abgezogen worden sein. Beim Verkauf von Ressourcen auf dem Markplatz, sollte der Spieler Gold erhalten haben.
Vorbedingungen:	T2, genügend Ressourcen vorhanden und Marktplatz und Labor noch nicht gebaut.
Durchlaufene Module:	T2+Marketplace
Schweregrad bei Fehlfunktion:	Hoch - Mindestanforderung

Testfallbezeichner : Funktion	T12: Einheit – Holztransport zur Basis
Eingabe:	Abgebautes Holz wird zur eigenen Basis transportiert.
Ausgabe:	Es wurde die Ressource Holz von der Worker-Einheit entfernt und dem Spieler gutgeschrieben
Vorbedingungen:	Wie T5, + neuer Zug und Basis in Reichweite der Worker-Einheit
Durchlaufene Module:	Wie T5
Schweregrad bei Fehlfunktion:	Hoch - Mindestanforderung

Testfallbezeichner : Funktion	T13: Forschung komplett
Eingabe:	Alle Forschungszweige werden aufgewertet, bis die höchste Stufe erreicht wurde.
Ausgabe:	Alle Forschungen und deren Funktionen sind freigeschaltet.
Vorbedingungen:	Wie T2+ genügend Ressourcen sind vorhanden.
Durchlaufene Module:	Wie T9
Schweregrad bei Fehlfunktion:	Hoch - Mindestanforderung

Testfallbezeichner : Funktion	T14: Einheiten kämpfen
Eingabe:	Die eigene SwordFighter-Einheit wird vor eine gegnerische SwordFighter-Einheit bewegt.
Ausgabe:	Die beiden Einheiten kämpfen gegeneinander und die eigene SwordFighter-Einheit verliert HP, gewinnt jedoch und dem Sieger wird Gold gutgeschrieben.
Vorbedingungen:	T8+ neuer Zug + gegnerische Einheit ist im Bewegungsradius der eigenen SwordFighter-Einheit und hat wenig HP.
Durchlaufene Module:	Wie T8
Schweregrad bei Fehlfunktion:	Hoch - Mindestanforderung

Testfallbezeichner : Funktion	T15: Einsatz von Aufwertern
Eingabe:	Der Button: „Labor“ wird aufgerufen und alle neu-angezeigten Buttons werden betätigt.
Ausgabe:	Alle Aufwerter haben ihre Funktion ausgeführt und die Kosten wurden von den Ressourcen des Spielers abgezogen.
Vorbedingungen:	Wie T11 + genügend Ressourcen
Durchlaufene Module:	Wie T11 + ReduceRessourceCosts, EmpowerShield
Schweregrad bei Fehlfunktion:	Hoch - Mindestanforderung

Testfallbezeichner : Funktion	T16: Nachrichten schreiben
Eingabe:	Nachricht wird in das Eingabefeld unter dem rechten Chat-Fenster geschrieben und es wird die Enter-Taste betätigt.
Ausgabe:	Der Chat hat eine neue Message im Backlog und wird aktualisiert. Diese Nachricht ist nun für andere Spieler sichtbar.
Vorbedingungen:	Wie T2
Durchlaufene Module:	Wie T2 +Chat+ Message
Schweregrad bei Fehlfunktion:	Hoch - Mindestanforderung

Testfallbezeichner : Funktion	T17: An die Pinnwand schreiben
Eingabe:	Über dem rechten Chatfenster wird der Button: „Pinnwand“ ausgewählt. Nachricht wird in das Eingabefeld unter dem rechten Chat-Fenster geschrieben und es wird die Enter-Taste betätigt.
Ausgabe:	Wie bei T16, außer das nur Team-Mitglieder die Nachricht lesen können.
Vorbedingungen:	Wie T2
Durchlaufene Module:	Wie T16
Schweregrad bei Fehlfunktion:	Hoch - Mindestanforderung

Testfallbezeichner : Funktion	T18: In Teamkasse einzahlen
Eingabe:	Das obere Truhen-Symbol auswählen und in die Eingabefelder beliebige Ressourcen eintragen.
Ausgabe:	Wenn genügend Ressourcen vorhanden, werden diese vom Spieler abgezogen und in die Teamkasse eingezahlt.
Vorbedingungen:	Wie T2
Durchlaufene Module:	Wie T2
Schweregrad bei Fehlfunktion:	Hoch - Mindestanforderung

Testfallbezeichner : Funktion	T19: Verrechnung von Teamkasse im Kampf
Eingabe:	Wie T14
Ausgabe:	Wie T14
Vorbedingungen:	T18 wurde durchgeführt.
Durchlaufene Module:	Bei Schwertkämpfern wie T14, abhängig von den kämpfenden Einheiten.
Schweregrad bei Fehlfunktion:	Hoch - Mindestanforderung

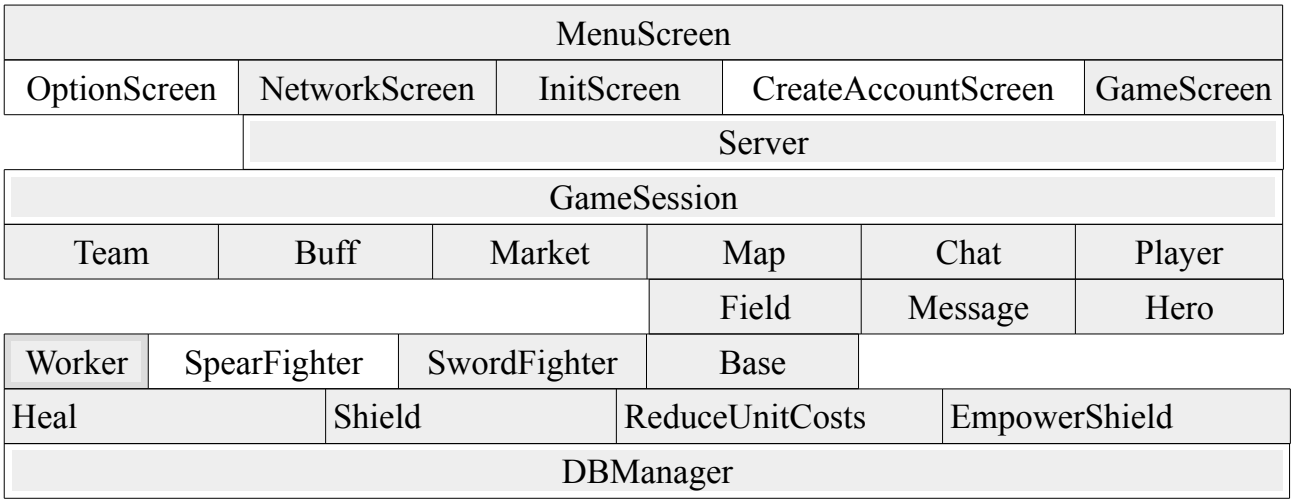
Testfallbezeichner : Funktion	T20: Spiel unterbrechen und fortsetzen
Eingabe:	Während des Spiels auf den Button: „Optionen“ klicken und „Beenden“ auswählen. Dann im Menü auf „Spiel starten“ drücken und das zu ladene Spiel auswählen, mit dem Nutzernamen anmelden und das Spiel durch Betätigung des Buttons: „Spielen“ fortsetzen.
Ausgabe:	Das Spiel wird gespeichert und beim Fortsetzen wieder geladen.
Vorbedingungen:	Wie T2
Durchlaufene Module:	Wie T2
Schweregrad bei Fehlfunktion:	Hoch - Mindestanforderung

Testfallbezeichner : Funktion	T21: Server neustarten und weiterspielen
Eingabe:	Wie T20, nur das vor dem Fortsetzen des Spiels das Spiel mit Server beendet wird.
Ausgabe:	Wie bei T20
Vorbedingungen:	Wie T2
Durchlaufene Module:	Wie T2
Schweregrad bei Fehlfunktion:	Hoch - Mindestanforderung

Testfallbezeichner : Funktion	T22: Test auf allen Betriebssystemen
Eingabe:	Das Spiel auf allen Betriebssystemen testen.
Ausgabe:	Alle Tests sollten gleich verlaufen.
Vorbedingungen:	Spiel auf System installiert.
Durchlaufene Module:	Alle
Schweregrad bei Fehlfunktion:	Hoch - Mindestanforderung

4.3 Testabdeckung

Als Kriterium für die Testabdeckung haben wir hier eine Liste aller Komponenten, die wir im Zusammenspiel testen wollen. Um eine komplette Pfadabdeckung zu messen, müssten wir dabei jedoch alle Wege durchlaufen, die möglich wären, was natürlich nicht möglich ist. Da wir Top-Down testen, unterteilen wir tabellarisch die Module nach Ebenen und vermerken in jedem relevanten Modul, ob dieses bei den Integrationstests durchlaufen worden ist. Hierbei ist die Zahl der möglichen Pfade natürlich geringer und wäre theoretisch auch zu berechnen, ist aber in diesem Protokoll nicht vorgesehen. Wir messen daher die Testabdeckung mit der Gesamtzahl der Module und der bei den Integrationstests angesprochenen Module. Das Verhältnis dieser Zahlen gibt hierbei unsere Testabdeckung an. Diese ist logischerweise gering zum Verhältnis der möglichen realen Fälle, die wir jedoch nicht betrachten wollen, da es auch in diesem Falle eher um die Abdeckung der Mindestanforderungen geht und nicht um Spezialfälle.



Alle zum Testen relevanten Klassen wurden hier abgebildet und die durch grau markierten Spalten sind durch die Integrationstests durchlaufen worden. Wir erhalten nach der Zahl der aufgelisteten Komponenten von 27 und einer Nutzung von 24 Klassen eine Testabdeckung von: 88%.

5. Funktionstests

Wie schon in den Komponententests erwähnt, wurden die Komponententests nach dem Schema von Funktionstests durchgeführt und dabei wurden bereits einzelne Funktionalitäten geprüft. Wir nutzen daher die Testfälle aus dem vorherigen Kapitel, um die Mindestanforderungen nochmal konkret abzudecken. Dafür erwähnen wir jeden Aspekt der Mindestanforderungen in der gegebenen Reihenfolge als Bezeichner, beschreiben welche Testfälle wir nehmen müssen, um den gewünschten Zustand zu erreichen und vermerken hier, ob dies erfüllt werden konnte. Hierbei muss jeder Zustand erreicht werden.

Mindestanforderung	Benötigte Testfälle/Schritte	Funktioniert
Allgemein		
Mehrspielerinnen-Strategie-Aufbauspiel	T2 durchführen und mit zweiten Client dem Spiel beitreten.	ja
Client-Server-Anwendung	Server ist vorhanden, Client läuft über LibGDX	ja
Spielfigur		
Jede Spielerin übernimmt im Spiel die Rolle einer Spielfigur und agiert durch diese. Eine Spielfigur verfügt über gewisse Fähigkeiten, die sich im Laufe des Spiels mindestens temporär ändern können.	Wenn T2 durchgeführt wird, ist eine Spielfigur neben der Basis vorhanden. Die Fähigkeiten werden bei T3 und T4 ausgeführt und sind temporär veränderlich.	ja
Es gibt mindestens zwei Fähigkeiten pro Spielfigur.	T3	ja
Ressourcen		
Es gibt mindestens drei verschiedene Kategorien von Ressourcen.	Mana ist zufällig auf dem Spielfeld, Holz und Eisen muss eingesammelt und zur Basis transportiert werden und Gold bekommt man durch Verkauf, Kampf, Basen und Marktplätze.	ja
Es gibt mindestens eine Ressource für jede Ressourcen-Kategorie.	Mana, Holz, Gold	ja
Es gibt insgesamt mindestens vier Ressourcen.	Mana, Holz, Eisen, Gold	ja

Basis

Spielerinnen können im Spielverlauf weitere Basen gründen/besiedeln/erobern.	Siehe T10 und die Bewegung der Figuren bei z.B. T12 findet zwischen Basen statt.	ja
Erlaubt Erweiterungen zu bauen	Siehe T11.	ja
Ressourcen zu produzieren	Siehe T7, Gold wird produziert.	ja
Einheiten zu bauen	Siehe T8.	ja
neue Technologien zu erforschen	Siehe T9	ja

Erweiterungen

Erweiterungen werden in einer Basis gebaut und kosten Ressourcen.	Siehe T11	ja
Erzeugung von Rohstoffen (z. B. Metallmine, Börse)	Siehe Marktplatz T11 (produziert Gold), sonst gibt es auch Minen, die logischerweise aber nicht in der Basis sind, diese werden durch Worker-Einheiten gebaut, die in der Basis vorher erzeugt werden müssen.	ja
Bau von Einheiten (z. B. Raumschiffswerft, Kaserne)	Siehe T8.	ja
Erforschung neuer Technologien (z. B. Forschungslabor, Magieturm)	Siehe T11 Labor .	ja
Warenhandel mit anderen Spielerinnen (z. B. Handelsposten)	Siehe T11 Marktplatz.	ja

Einheiten

Einheiten werden in Einheitenbau-Erweiterungen einer Basis für Ressourcen gebaut	Siehe T8.	ja
und können sich zwischen Basen bewegen.	Siehe T12.	ja

Transport von Rohstoffen	Siehe T12.	ja

Angriff anderer Einheiten	Siehe T14.	ja

Forschung

Freischaltung neuer Spielelemente	Wenn der Kulturpfad im Technologiebaum voll ausgebildet ist, können eigene Basen gebaut werden.	ja

Verbesserung bestimmter Werte	Siehe T9.	ja

Jede Forschung kostet Ressourcen.	Siehe T9.	ja

Es gibt mindestens 3 separate Forschungsrichtungen.	Siehe den bei T9 aufgerufenen Technologiebaum.	ja

Es gibt mindestens 5 zu erforschende Technologien pro Forschungsrichtung.	Siehe den bei T9 aufgerufenen Technologiebaum.	ja

Kampf

Es gibt mindestens Kämpfe zwischen Einheiten.	Siehe T14.	ja

Es kann um eine Basis gekämpft werden.	Siehe T10.	ja

Die Siegerin bekommt einen Teil der in der Basis vorhandenen Ressourcen.	Siehe T10.	ja

Aufwerter

verbessert temporär die Werte	Siehe T15 + gehe zum nächsten Zug.	ja

Der Einsatz eines Aufwerters kostet Ressourcen.	Siehe T15.	ja

Werte von Einheiten;	Siehe T15.	ja

Werte von Erweiterungen;	Siehe T15.	ja

Werte von Fähigkeiten der Spielfiguren.	Siehe T15.	ja

PN

Spielerinnen können anderen Spielerinnen Nachrichten senden.	Siehe T16.	ja
Die Nachrichten werden archiviert und können von den Empfängern auch später noch eingesehen werden.	Siehe T16 und lasse etwas Zeit vergehen.	ja

Gemeinschaften

Spielerinnen können sich zu Gemeinschaften zusammenschließen.	Siehe Teamauswahl aus T2.	ja
eine Pinnwand, an der Spielerinnen der Gemeinschaft etwas schreiben können;	Siehe T17.	ja
eine Gemeinschaftskasse, in die Spielerinnen Ressourcen einzahlen können	Siehe T18.	ja
Wenn eine Spielerin einen Teil ihrer Einheiten verliert, bekommt sie einen Teil der Baukosten ihrer verlorenen Einheiten aus der Gemeinschaftskasse erstattet.	Führe T18 aus und dann T14.	ja

Persistenz

Spiele können unterbrochen und später fortgesetzt werden.	Siehe T20.	ja
Spielstände stehen bei Server-Neustart zur Verfügung.	Siehe T21.	ja

Variante B

Server	Java 7/8	ja
Client	LibGDX.	ja
Kommunikationsprotokoll	RMI	ja

Zeit(Keine Mindestanforderung)

Der Bau von Erweiterungen, Einheiten und Forschungen kostet außerdem Zeit.	Beispiele sind im Spiel und Architektur vorhanden, wurde jedoch nicht in den Testfällen beachtet, da keine Mindestanforderung.	ja

6. Spezialfall-Dokumentation