Quality Assurance

Wasserschlacht Simulator

Gruppe 04

Inhalt

[Einleitung 2](#_Toc483199388)

[Zuverlässigkeit 3](#_Toc483199389)

[CharacterTest 4](#_Toc483199390)

[ClientGameContollerTest 4](#_Toc483199391)

[GameMapTest 4](#_Toc483199392)

[PlayerTest 4](#_Toc483199393)

[ServerGameContollerTest 5](#_Toc483199394)

[TeamTest 5](#_Toc483199395)

[TileTest 5](#_Toc483199396)

[TileTypeTest 5](#_Toc483199397)

[TurnControllerTest 5](#_Toc483199398)

[WeaponTest 6](#_Toc483199399)

[WinningConditionTest 6](#_Toc483199400)

[WorldTest 6](#_Toc483199401)

[Effizienz 7](#_Toc483199402)

[Schlusswort 8](#_Toc483199403)

# Einleitung

Bereits am zweiten Meilenstein mussten wir unseren Tutoren ein Konzept für die Qualitätssicherung vorstellen. Damals hatten wir uns die Qualitätsmerkmale Funktionalität und Effizienz ausgesucht.

Wir wählten diese Merkmale, da sie ein reibungsloses Spiel versprachen.

Diese zwei Merkmale stellten wir dann auch am dritten Meilenstein dem Plenum vor.

Danach starteten wir mit den Unit-Tests für die Zuverlässigkeit und den Effizienzmessungen.

Für den letzten Meilenstein fassen wir nun alles zusammen.

# Zuverlässigkeit

Mit der Zuverlässigkeit als Qualitätsmerkmal wollen wir sicherstellen, dass das Spiel nicht abgebrochen wird. Denn uns ist es wichtig, dass die Spielperson während des Spielens nicht von Bugs gestört wird. Wir hatten daher entschieden, die Spielengine zu testen.

Die Anzahl der zu testeten Klassen, war zu Beginn auf 50 % der Engine angesetzt. Da wir unsere Engine besser testen wollten und viele Klassen essenziel sind, haben wir die Zahl erhöht. Wir sind nun bei 80 % plus drei Klassen aus dem Package game. Zudem wollten wir, dass unsere Klassen gründlich getestet werden, weshalb eine Code Coverage von 90 % angesetzt wurde.

Die Unit-Tests werden nach dem Alphabet geordnet kurz angesprochen.

## CharacterTest

Die Character Klasse ist zuständig für die Zustände der Kinder. Sie kann die Actionpoints eines Kindes entfernen, die Actionpoints zurücksetzen, ein Kind entfernen, ein Kind um einen Tile bewegen, ein Kind zu einem bestimmten Feld bewegen und alle Kinder in Reichweite zurückgeben. In den Tests wird getestet, ob ein Kind ausserhalb seiner Reichweite bewegt werden kann, ob Actionpoints entfernt und wieder zurückgesetzt werden können. Alle Tests funktionieren einwandfrei.

Der Code Coverage liegt bei 50 %.

## ClientGameContollerTest

Die Klasse ClientGameController ist zuständig für den Aufbau des Spieles zwischen Client und Server. Die Klasse kann einen Spieler zum Spiel hinzufügen, ein Benutzer als wartend einstufen, dem Server ein Startsignal für den Spielanfang senden, ein Spieler entfernen, den Server fragen, ob man das Spiel verlassen kann und das Spiel beenden. In den Tests wird das Hinzufügen der Spieler und die Liste der wartenden Clients getestet. Eine Vielzahl der Methoden konnte nicht getestet werden, da für die Tests ein Server initialisiert werden müsste. Alle Tests Laufen einwandfrei.

## GameMapTest

Mit der Klasse GameMap werden die Spielfelder kreiert, auf denen die Spielfiguren gesetzt werden. Es gibt zwei Hauptkarten und eine kleinere Testkarte. Die Testkarte wird nicht getestet. Die Karten werden beim Ausführen der Main von den Textdateien eingelesen. Auch bei dem Test werden im Set-up alle Karten geladen. Zudem wird manuell eine dritte Karte mit einem char array erstellt.

Die im Set-up ausgeführte einlese Methode, wird durch einen Test geprüft, der die Existenz der Karten beweist. Alle restlichen Methoden werden in ihren Testmethoden aufgerufen und die Resultate der Ausführung mit Assert überprüft. Soweit es ging oder sinnvoll war, wurden alle Maps getestet. Zum Beispiel wurde eine Methode, die die Tiles einer Karte als char array zurückgibt, nur an der manuell erstellten Karte getestet, da das char array zur Erstellung als Vergleich dienen konnte.

Nicht getestet wird die Überschreibmethode toString. Die Code Coverage war zu Beginn bei 20 % und wurde im Laufe der Zeit auf 60 %, 70 % und schliesslich 90 % erhöht. Die Klasse ist gründlich getestet und zeigte sich so weit fehlerfrei.

## PlayerTest

Hinter dem Player steht der User. Der Spieler wird automatisch einem Team hinzugefügt. Vom Player kann man den Namen, das Team, den User, die Farbe und Informationen bezüglich besiegten und verlorenen Spielfiguren erfahren. Für eine bessere Überprüfung der Ausführungsresultate wurden die CoreMatchers Methoden importiert. Diese Testmethoden bzw. Methoden funktionieren einwandfrei. Die Methoden zur Abfrage des momentanen Spielers und ob der Player noch Spielfiguren besitzt, funktionieren, konnten jedoch in der Testmethode nicht direkt vom Playerobjekt aufgerufen werden. Es handelt sich vermutlich um einen Fehler im Setup des Tests.

Die Code Coverage wurde von 60 % auf 80 % gesteigert. Nicht getestet wurden das Starten eines neuen Zuges und das Abfragen des Clients. Bei beiden Fällen liegt es am Set-up des Testes.

## ServerGameContollerTest

Die Klasse ServerGameContorller kontrolliert das Verschwinden und Erstellen von Kindern. Sie ist auch zuständig dafür zu kontrollieren, mit wie vielen Punkten ein Spieler gewonnen hat. Die Klasse kann ein Spiel starten, ein Spieler entfernen, alle Kinder eines Spielers zurückgeben und das Spiel beenden. Im Test wird getestet, ob ein Spieler hinzugefügt und entfernt werden kann. Alle Tests laufen einwandfrei.

Die ServerGameController Klasse erreicht eine Code Coverage von 50 %.

## TeamTest

In unserem Spiel gibt es ein Player pro Team. Der Name des Teams ist im Spiel selbst nicht wählbar und wird automatisch erstellt. Im Test wird dem Testteam jedoch einen gewählten Namen gegeben. Der Name sollte für das Funktionieren der Methoden irrelevant sein. Mit einer Methode der Klasse, die alle Teammitglieder als ArrayList zurückgibt, wird nebenbei auch das erfolgreiche Hinzufügen des Testplayers bewiesen. Obwohl es nur ein Player pro Team gibt, testen wir das Hinzufügen in ein Team nochmals direkt mit einem zweiten Player. Die ArrayList enthält nun zwei Player, die Methode funktioniert und zeigt, dass die Erweiterung auf Teambildung möglich wäre. Der Test für die kleine Teamklasse erreicht eine Code Coverage von 100 %.

## TileTest

Nach dem Spielstart bewegt man die Spielfiguren von Tile zu Tile. Dafür braucht man einige Informationen und Aktionen. Im Set-up wird eine eingelesene Karte ausgewählt und mit weiteren Objekten eine world erstellt. Mit einer Methode werden zwei Tiles geholt, einmal der Eckpunkt links oben und einmal ein Feld, auf dem sich eine Spielfigur befinden sollte. Für manche Methoden spielt es eine Rolle, ob sich auf dem Feld eine Figur befindet. Bei der Methode, die die Begehbarkeit eines Feldes überprüft, muss als Parameter mitgegeben werden, ob die Spielfigur beachtet werden soll oder nicht. Die Methoden, die das nächste Feld in den Himmelsrichtungen zurückgeben, wurden in einer Testmethode zusammengefasst. Die Code Coverage hat sich von 46 % auf 96 % gesteigert. Nicht getestet wird die ÜberschreibMethode toString. Die Klasse Tile ist damit gründlich getestet und anhand der Tests von Bugs frei.

## TileTypeTest

Die Klasse TileType ist eigentlich eine kleine enum Klasse. Dennoch gibt es Methoden, die in der Tile und sogar in der world Klasse gebraucht werden. Daher wird die Klasse getestet. Bei der Erstellung der Karten muss der richtige Feldtyp für die eingelesenen chars zurückgegeben werden. Weiter muss die richtige Grafik für den entsprechenden Typ geholt werden. Je nach Typus ist das Feld begehbar oder nicht. Es wird 100 % der Klasse erfolgreich getestet.

## TurnControllerTest

Die TurnController Klasse kann den Zug eines Spielers beenden und einen Spieler entfernen. Es werden in den Tests sowohl das Enden des Zuges so wie das Entfernen eines Spielers getestet. Beides funktioniert einwandfrei.

Es wird ein Code Coverage von 60 % erreicht.

## WeaponTest

Jede Spielfigur bekommt eine Wasserwaffe. Der Spieler kann dabei zwischen zwei Modellen aussuchen. Zur zusätzlichen Überprüfung wird im Set-up eine Testwaffe hinzugefügt. Bereits schon in der Main, wird die Methode zur Erstellung der Waffenprototypen aufgerufen. Diese Methode wird mit Hilfe einer anderen getestet. Gibt die Methode, die eine ArrayList der Waffen erstellt, null zurück, hat das Erstellen nicht funktioniert. Von der Methode, die Prototyps hinzufügt, gibt es zwei Versionen. Wird die Methode aufgerufen, ohne Parameter für die Wirkungsreichweite, wird sie ein zweites Mal aufgerufen, und die Reichweite automatisch auf 0 gesetzt. Diese Handhabung ermöglicht eine Erweiterung auf Waffen mit einem Wirkungsradius. Die Code Coverage beträgt über 90 %. Es werden jeweils alle drei Waffen getestet. Nicht getestet wird die Überschreibmethode toString. Die Klasse gilt so weit als fehlerfrei.

## WinningConditionTest

Mit der Klasse WinningCondition wird das Gewinnerteam ermittelt. Gewonnen hat ein Team, wenn es als einziges noch Spielfiguren besitzt. Wird die Kondition getestet, während die Teams noch Figuren haben, wird null zurückgegeben. Getestet wird die Methode an zwei Testwelten. Einmal wird nur ein Team der Welt hinzugefügt und muss somit das Gewinnerteam sein. Das zweite Mal wird die Methode getestet, obwohl noch kein Team gewonnen hat, also beide noch Figuren übrighaben. Der WinnigConditionTest deckt 100 % der Klasse ab.

## WorldTest

Die World Klasse beinhaltet die gesamte Spielwelt. Die Klasse kann ein Tile an einer bestimmten Position und alle Kinder eines Spielers zurückgeben, ein Kind eines Spielers entfernen, nachschauen, ob ein Spieler gewinnt, ein Kind nass machen und ein Kind schubsen. Es wird getestet, ob ein Tile richtig wiedergegeben wird.

# Effizienz

Wir testen den Arbeitsspeichergebrauch des Spieles in verschiedenen Zuständen und zu verschiedenen Kalenderdaten. Ausgesucht wurden die Daten nach grösseren Änderungen und Mergers auf dem Master.

|  |  |  |  |
| --- | --- | --- | --- |
| **Datum** | **Client** | **Server** | **Bemerkung** |
| 14.04.2017 | **-** | 0.789 MB | Server ohne Client Verbindung |
| 14.04 | 16.683 MB | 9.73 MB | Verbundener Server und Client aber ohne Spiel |
| 19.04 | - | 0.789 MB | Server ohne Client Verbindung |
| 19.94 | 16.66 MB | 9.622 MB | Verbundener Server und Client aber ohne Spiel |
| 19.04 | 18.308 MB | 11.266 MB | Verbundener Server und Client mit Spiel |
| 17.05 | - | 0.789 MB | Server ohne Client Verbindung |
| 17.05 | 18.093 MB | 16.345 MB | Verbundener Server und Client aber ohne Spiel |
| 17.05 | 20.515 MB | 17.652 MB | Verbundener Server und Client mit Spiel |
| 11.05 | - | 0.789 MB | Server ohne Client Verbindung |
| 11.05 | 18.169 MB | 16.504 MB | Verbundener Server und Client aber ohne Spiel |
| 11.05 | 20.570 MB | 17.619 MB | Verbundener Server und Client mit Spiel |

Der Unterschied zwischen den verschiedenen Zuständen ist verständlich. Im Mai stieg der Gebrauch an Memory, jedoch ist dies durchaus normal und nicht übertrieben. Des Weiteren steigt unsere Speichernutzung nicht über ein zehn Minuten YouTube Video in schlechter Qualität, was beeindruckend ist.

# Schlusswort

Schlussendlich haben wir für die Zuverlässigkeit nur eine durchschnittliche Code Coverage von 70 % erreicht. Das liegt unteranderem an der Verknüpfung mit dem Server, den man für manche Tests initialisieren hätte müssen. Andererseits waren die Klassen oft verschachtelt und allgemein schwer zu testen. Man sollte daher die Tests und das Projekt zusammen entwickeln, um die Klassen und die Tests gegenseitig anpassen zu können.

Für die Effizienz wäre es für ein nächstes Mal interessant, die Geschwindigkeit zu messen.