Studiengang Informatik (B.Sc.) WiSe 2022/23

Software Engineering 2
Übung 5

19. Januar 2023

Ali Al Jasim (123741)
Daniel Ocks (124219)

Task 1

Von den aufgelisteten noch nicht gelösten Bugs:

1. Positioning ships without placing
2. Collsion with items
3. Placing ships with erratic rotation

wurde nur der 2. Bug gelöst. Die Kamera hat nun mehr die Eigenschaften einer Person als einer schwebenden Kamera, denn es ist nun nicht mehr möglich, den Spielraum zu verlassen bzw. durch den Tisch zu gleiten. Bei Versuch dieser Aktionen wird der Spieler nun eingeschränkt, er kann sich somit nur in einem bestimmen Bereich bewegen.

Bug 1. und 3. traten nicht auf, bzw. wurden von uns nicht als Bug angesehen.

Zum 1. Bug kann man zwar sagen, dass es möglich ist Schiffe unabsichtlich zu verschieben, dies geht aber nur, falls diese zuvor noch nicht mit E fest platziert wurden. Bei fest platzierten Schiffen traf dieser Fehler nicht auf, weshalb wir sagen, dass es sich hierbei nicht um einen Bug handelt.

Der 3. Bug hingegen trat garnicht erst auf, egal wie lange wir die Schiffe gedreht haben, war es uns nicht möglich diese fest inneinander zu platzieren. Das spiel hat von selbst erkannt, dass es sich um eine illegale Aktion handelt.

Task 2

Black-box testing (L7, slide 25)

Mit Black-box testing für das Projekt 'sinking-ship' konnten wir die Bugs im Projekt schnell erkennen. Es hat uns auch geholfen, einen schnellen Überblick zu gewinnen, was die Requirements waren, was implementiert wurde und was noch im Projekt fehlt.
Black-box testing hilft vorallem logische Implementierungsfehler zu finden und zu diskutieren. Damit ist gemeint, während der Software Entwicklung könnten manche Requirements von Developers nicht richtig verstanden werden und falsch bzw. nicht so implementiert, wie es erwartet war. Manchmal sind auch Requirements nicht so realisierbar wie es gewünscht wurde. Daher kommt es bei der Software Entwicklung zu ähnlichen funktionalen Features. Das alles lässt sich mit black-box testing feststellen und diskutieren.
Ein Problem bei Black-box testing ist, dass manche Features mit Testfunktionen (z.B. JUnit) nicht testbar sind, ohne den Code anzuschauen. Dazu hätte es vielleicht helfen können, ein Kalssendiagramm in der implementation_description und in der maintenance_description anzugeben. Das Testen mit Testfunktionen (z.B. JUnit) kann aber nicht immer hilfreich sein. Damit könnten Bugs in Klassen des Projekts gefunden, die eigentlich im Laufe des Programms nicht auftreten oder nicht wichtig sind. Es wird dann trotzdem als Bug beschrieben und man versucht diese Bugs wegzumachen und dadurch ensteht redundanter Code. Daher eignet sich Black-box testing besser für das Testen des Projekts auf Requirements und was davon implementiert wurde. Wie die einzelnen Funktionen und Klassen implementiert sind, sollte bei black-box testing nicht das wichtigste sein. Dafür eignet sich white-box testing.

Comments (L6, slide 42)

Gute auskommentierter Code hat uns geholfen bzw. hätte uns helfen können. Wir haben oft Code gesehen, welchen wir selbst nach mehrmaligem Lesen nicht verstanden haben, da dieser einfach nicht ausreichen auskommentiert war. Oft haben nur Kleinigkeiten gefehlt wie z. B. warum ein int einen bestimmten Wert hat oder warum auf einen Wert wie z. B. die 0 im jeweiligen Fall besonders geachtet werden muss. Es ging dacher viel Zeit für das Recherchieren von Informationen drauf, welche sehr einfach aus der vorherigen Arbeiten hätten mitgegeben werden können. Oft waren Kommentare auch falsch platziert. Vor allem, wenn es um das Vorwarnen auf ein Problem ging. Sie haben zwar das Problem erklärt aber waren Fehlplatziert, was diese einfach nicht brauchbar gemacht hat, sie haben in solchen Fällen sogar mehr verwirrt als geholfen. Es wurde auch selten bedacht Kommentare anzupassen. So kam es auch dazu, dass veränderter Code, mit alten Kommentaren verschlechtert wurde. Auch hat man gemerkt, dass viele Kommentare zum Ende der ganzen Arbeit geschrieben wurden und nur als eine Aufgabe die halt gemacht werden muss angesehen wurde. Dadurch wurden viele Informationen vergessen. Am meisten war aber Code nicht auskommentiert, welche eigentlich einfach verständlich war. Eigentlich, weil der Code nur, wenn man tief im Projekt drinnen ist, sich den Rest denken kann.

Für die Zukunft sollte man Kommentare als einen wichtigen Teil ansehen, man sollte sie genauso achten wie den geschrieben Code, nicht nur weil der darauffolgende Gruppe die Arbeit dadurch erleichtert wird, sondern auch, weil zum einlesen nach einen längeren Pause auch von Vorteil ist.